

pymipspec MIPS Architecture Reference Manual

Table of Contents

1. Instruction Summary	2
1.1. Availability Expressions	2
1.2. MIPS	4
1.3. MIPS16E	29
1.4. microMIPS	30
1.5. nanoMIPS	56
2. System Registers	82
2.1. C0	82
2.2. C1	143
3. Privileged Architecture Algorithms	154
3.1. Address Translation	154
3.2. Addressing	162
3.3. Exception Processing	163
3.4. Operating Modes	168
3.5. Shadow Register Sets	168
3.6. TLB Operations	169
4. CP1 Floating-Point Architecture	175
4.1. Architectural state access	175
4.2. IEEE-754 primitives	178
5. DSP Module Architecture	181
5.1. Architectural state	181
5.2. Resource access	182
5.3. Shared DSP primitives	183
6. Shared Architectural Algorithms	186
6.1. Algorithm summary	186
6.2. <code>crc32</code>	186
6.3. <code>bitswap_word</code>	186
6.4. <code>save_restore_register</code>	187
6.5. <code>count_leading_bits</code>	187
7. Instruction Encoding and Decode	188
7.1. Decode result	188
7.2. Reversible field mappings	188
7.3. Compressed-register encodings	189
8. Instruction Definitions	194
8.1. ABS.D	195
8.2. ABS.PS	197
8.3. ABS.S	199
8.4. ABSQ_S.PH	201

8.5. ABSQ_S.QB	203
8.6. ABSQ_S.W	205
8.7. ADD	207
8.8. ADD.D	209
8.9. ADD.PS	212
8.10. ADD.S	214
8.11. ADDI	217
8.12. ADDIU	219
8.13. ADDIU	231
8.14. ADDIUPC	233
8.15. ADDQ.PH	237
8.16. ADDQ_S.PH	239
8.17. ADDQ_S.W	241
8.18. ADDQH.PH	243
8.19. ADDQH.W	245
8.20. ADDQH_R.PH	247
8.21. ADDQH_R.W	249
8.22. ADDSC	251
8.23. ADDU	253
8.24. ADDU.PH	257
8.25. ADDU.QB	259
8.26. ADDU_S.PH	261
8.27. ADDU_S.QB	263
8.28. ADDUH.QB	265
8.29. ADDUH_R.QB	267
8.30. ADDWC	269
8.31. ALIGN	271
8.32. ALNV.PS	273
8.33. ALUIPC	275
8.34. AND	278
8.35. ANDI	282
8.36. APPEND	285
8.37. AUI	287
8.38. AUIPC	289
8.39. BAL	291
8.40. BALC	293
8.41. BALIGN	296
8.42. BALRSC	298
8.43. BBEQZC	300
8.44. BBNEZC	302
8.45. BC	304

8.46. BC1EQZ	307
8.47. BC1EQZC	309
8.48. BC1F	311
8.49. BC1FL	313
8.50. BC1NEZ	315
8.51. BC1NEZC	317
8.52. BC1T	319
8.53. BC1TL	321
8.54. BC2EQZC	323
8.55. BC2F	325
8.56. BC2NEZC	327
8.57. BC2T	329
8.58. BEQ	331
8.59. BEQC	334
8.60. BEQIC	337
8.61. BEQL	339
8.62. BEQZALC	341
8.63. BEQZC	343
8.64. BGEC	347
8.65. BGEIC	350
8.66. BGEIUC	352
8.67. BGEUC	354
8.68. BGEZ	357
8.69. BGEZAL	359
8.70. BGEZALC	361
8.71. BGEZALL	363
8.72. BGEZALS	365
8.73. BGEZC	366
8.74. BGEZL	369
8.75. BGTZ	371
8.76. BGTZALC	373
8.77. BGTZC	375
8.78. BGTZL	377
8.79. BITREV	379
8.80. BITSWAP	381
8.81. BLEZ	383
8.82. BLEZALC	385
8.83. BLEZC	387
8.84. BLEZL	389
8.85. BLTC	391
8.86. BLTIC	394

8.87. BLTIUC	396
8.88. BLTUC	398
8.89. BLTZ	401
8.90. BLTZAL	403
8.91. BLTZALC	405
8.92. BLTZALL	407
8.93. BLTZALS	409
8.94. BLTZC	410
8.95. BLTZL	412
8.96. BNE	414
8.97. BNEC	417
8.98. BNEIC	420
8.99. BNEL	422
8.100. BNEZALC	424
8.101. BNEZC	426
8.102. BNVC	430
8.103. BOVC	432
8.104. BPOSGE32	434
8.105. BPOSGE32C	436
8.106. BREAK	438
8.107. BRSC	442
8.108. C.cond.D	443
8.109. C.cond.PS	445
8.110. C.cond.S	448
8.111. CACHE	450
8.112. CEIL.L.D	454
8.113. CEIL.L.S	456
8.114. CEIL.W.D	458
8.115. CEIL.W.S	460
8.116. CFC1	462
8.117. CFC2	464
8.118. CLASS.D	466
8.119. CLASS.S	469
8.120. CLO	472
8.121. CLZ	475
8.122. CMP.D	478
8.123. CMPEQ.PH	481
8.124. CMPLE.PH	483
8.125. CMPLT.PH	485
8.126. CMPS	487
8.127. CMPGDU.EQ.QB	490

8.128. CMPGDU.LE.QB	492
8.129. CMPGDU.LT.QB	494
8.130. CMPGU.EQ.QB	496
8.131. CMPGU.LE.QB	498
8.132. CMPGU.LT.QB	500
8.133. CMPU.EQ.QB	502
8.134. CMPU.LE.QB	504
8.135. CMPU.LT.QB	506
8.136. COP2	508
8.137. CRC32B	510
8.138. CRC32CB	512
8.139. CRC32CD	513
8.140. CRC32CH	515
8.141. CRC32CW	516
8.142. CRC32D	517
8.143. CRC32H	519
8.144. CRC32W	520
8.145. CTC1	521
8.146. CTC2	524
8.147. CVT.D.L	526
8.148. CVT.D.S	529
8.149. CVT.D.W	532
8.150. CVT.L.D	535
8.151. CVT.L.S	538
8.152. CVT.PS.S	541
8.153. CVT.S.D	543
8.154. CVT.S.L	546
8.155. CVT.S.PL	549
8.156. CVT.S.PU	551
8.157. CVT.S.W	553
8.158. CVT.W.D	556
8.159. CVT.W.S	559
8.160. DADD	562
8.161. DADDI	564
8.162. DADDIU	566
8.163. DADDIUPC	571
8.164. DADDU	573
8.165. DAHI	575
8.166. DALIGN	577
8.167. DATI	579
8.168. DAUI	581

8.169. DBITSWAP	583
8.170. DCLO	585
8.171. DCLZ	588
8.172. DDIV	591
8.173. DDIVU	594
8.174. DERET	597
8.175. DEXT	599
8.176. DEXTM	601
8.177. DEXTU	603
8.178. DI	605
8.179. DINS	607
8.180. DINSM	609
8.181. DINSU	611
8.182. DIV	613
8.183. DIV.D	616
8.184. DIV.S	619
8.185. DIVU	622
8.186. DLSA	626
8.187. DLUI	628
8.188. DMFC0	629
8.189. DMFC1	631
8.190. DMOD	633
8.191. DMODU	635
8.192. DMTC0	637
8.193. DMTC1	639
8.194. DMUH	641
8.195. DMUHU	643
8.196. DMUL	645
8.197. DMULT	647
8.198. DMULTU	649
8.199. DMULU	651
8.200. DPA.W.PH	653
8.201. DPAQ_S.W.PH	655
8.202. DPAQ_SA.L.W	657
8.203. DPAQX_S.W.PH	659
8.204. DPAQX_SA.W.PH	661
8.205. DPAU.H.QBL	663
8.206. DPAU.H.QBR	665
8.207. DPAX.W.PH	667
8.208. DPS.W.PH	669
8.209. DPSQ_S.W.PH	671

8.210. DPSQ_SA.L.W	673
8.211. DPSQX_S.W.PH	675
8.212. DPSQX_SA.W.PH	677
8.213. DPSU.H.QBL	679
8.214. DPSU.H.QBR	681
8.215. DPSX.W.PH	683
8.216. DROTR	685
8.217. DROTR32	687
8.218. DROTRV	689
8.219. DROTX	691
8.220. DSBH	692
8.221. DSHD	694
8.222. DSSL	696
8.223. DSSL32	698
8.224. DSSLV	700
8.225. DSRA	702
8.226. DSRA32	704
8.227. DSRV	706
8.228. DSRL	708
8.229. DSRL32	710
8.230. DSRLV	712
8.231. DSUB	714
8.232. DSUBU	716
8.233. DVP	718
8.234. EHB	720
8.235. EI	721
8.236. ERET	723
8.237. ERETNC	725
8.238. EVP	727
8.239. EXT	729
8.240. EXTD	732
8.241. EXTP	734
8.242. EXTPDP	736
8.243. EXTPDPV	738
8.244. EXTPV	740
8.245. EXTR.W	742
8.246. EXTR_R.W	744
8.247. EXTR_RS.W	746
8.248. EXTR_S.H	748
8.249. EXTRV.W	750
8.250. EXTRV_R.W	752

8.251. EXTRV_RS.W	754
8.252. EXTRV_S.H	756
8.253. EXTW	758
8.254. FLOOR.L.D	760
8.255. FLOOR.L.S	762
8.256. FLOOR.W.D	764
8.257. FLOOR.W.S	766
8.258. GINVI	768
8.259. GINVT	770
8.260. INS	771
8.261. INSV	774
8.262. J	776
8.263. JAL	778
8.264. JALR	780
8.265. JALRC	783
8.266. JALRS	787
8.267. JALS	789
8.268. JALX	790
8.269. JIALC	792
8.270. JIC	794
8.271. JR	796
8.272. JRADDIUSP	798
8.273. JRC	799
8.274. JRCADDIUSP	801
8.275. LB	802
8.276. LBE	806
8.277. LBU	808
8.278. LBUE	812
8.279. LBUX	814
8.280. LBX	817
8.281. LD	819
8.282. LDC1	822
8.283. LDC2	826
8.284. LDL	828
8.285. LDM	830
8.286. LDPC	832
8.287. LDR	835
8.288. LDX	837
8.289. LDXC1	839
8.290. LDXS	841
8.291. LH	843

8.292. LHE	847
8.293. LHU	849
8.294. LHUE	853
8.295. LHUX	855
8.296. LHUXS	857
8.297. LHX	859
8.298. LHXS	862
8.299. LI	864
8.300. LL	866
8.301. LLD	870
8.302. LLDLP	873
8.303. LLE	875
8.304. LLWP	877
8.305. LLWPE	880
8.306. LSA	882
8.307. LUI	884
8.308. LUXC1	886
8.309. LW	888
8.310. LWC1	895
8.311. LWC2	899
8.312. LWE	901
8.313. LWL	903
8.314. LWM	905
8.315. LWP	908
8.316. LWPC	910
8.317. LWR	913
8.318. LWU	916
8.319. LWUPC	919
8.320. LWUX	921
8.321. LWUXS	923
8.322. LWX	925
8.323. LWXC1	928
8.324. LWXS	930
8.325. MADD	933
8.326. MADD.D	936
8.327. MADD.PS	938
8.328. MADD.S	940
8.329. MADDF.D	942
8.330. MADDF.S	945
8.331. MADDU	948
8.332. MAQ_S.W.PHL	951

8.333. MAQ_S.W.PHR	953
8.334. MAQ_SA.W.PHL	955
8.335. MAQ_SA.W.PHR	957
8.336. MAX.D	959
8.337. MAX.S	962
8.338. MAXA.D	965
8.339. MAXA.S	968
8.340. MFC0	971
8.341. MFC1	973
8.342. MFC2	975
8.343. MFHC0	977
8.344. MFHC1	979
8.345. MFHC2	981
8.346. MFHI	983
8.347. MFLO	987
8.348. MIN.D	991
8.349. MIN.S	994
8.350. MINA.D	997
8.351. MINA.S	1000
8.352. MOD	1003
8.353. MODSUB	1005
8.354. MODU	1007
8.355. MOV.D	1010
8.356. MOV.PS	1012
8.357. MOV.S	1014
8.358. MOVE	1016
8.359. MOVE.BALC	1018
8.360. MOVEP	1020
8.361. MOVF	1023
8.362. MOVF.D	1025
8.363. MOVF.PS	1027
8.364. MOVF.S	1029
8.365. MOVN	1031
8.366. MOVN.D	1033
8.367. MOVN.PS	1035
8.368. MOVN.S	1037
8.369. MOVT	1039
8.370. MOVT.D	1041
8.371. MOVT.PS	1043
8.372. MOVT.S	1045
8.373. MOVZ	1047

8.374. MOVZ.D	1049
8.375. MOVZ.PS	1051
8.376. MOVZ.S	1053
8.377. MSUB	1055
8.378. MSUB.D	1058
8.379. MSUB.PS	1060
8.380. MSUB.S	1062
8.381. MSUBF.D	1064
8.382. MSUBF.S	1067
8.383. MSUBU	1070
8.384. MTC0	1073
8.385. MTC1	1075
8.386. MTC2	1077
8.387. MTHC0	1079
8.388. MTHC1	1081
8.389. MTHC2	1084
8.390. MTHI	1086
8.391. MTHLIP	1089
8.392. MTLO	1091
8.393. MUH	1094
8.394. MUHU	1096
8.395. MUL	1098
8.396. MUL.D	1102
8.397. MUL.PH	1105
8.398. MUL.PS	1108
8.399. MUL.S	1110
8.400. MUL_S.PH	1113
8.401. MULEQ_S.W.PHL	1116
8.402. MULEQ_S.W.PHR	1118
8.403. MULEU_S.PH.QBL	1120
8.404. MULEU_S.PH.QBR	1122
8.405. MULQ_RS.PH	1124
8.406. MULQ_RS.W	1126
8.407. MULQ_S.PH	1128
8.408. MULQ_S.W	1130
8.409. MULSA.W.PH	1132
8.410. MULSAQ_S.W.PH	1134
8.411. MULT	1137
8.412. MULTU	1140
8.413. MULU	1143
8.414. NAL	1145

8.415. NEG.D	1146
8.416. NEG.PS	1148
8.417. NEG.S	1150
8.418. NMADD.D	1152
8.419. NMADD.PS	1154
8.420. NMADD.S	1156
8.421. NMSUB.D	1158
8.422. NMSUB.PS	1160
8.423. NMSUB.S	1162
8.424. NOP	1164
8.425. NOR	1167
8.426. NOT	1169
8.427. OR	1171
8.428. ORI	1175
8.429. PACKRL.PH	1177
8.430. PAUSE	1179
8.431. PICK.PH	1181
8.432. PICK.QB	1183
8.433. PLL.PS	1185
8.434. PLU.PS	1187
8.435. PRECEQ.W.PHL	1189
8.436. PRECEQ.W.PHR	1191
8.437. PRECEQU.PH.QBL	1193
8.438. PRECEQU.PH.QBLA	1195
8.439. PRECEQU.PH.QBR	1197
8.440. PRECEQU.PH.QBRA	1199
8.441. PRECEU.PH.QBL	1201
8.442. PRECEU.PH.QBLA	1203
8.443. PRECEU.PH.QBR	1205
8.444. PRECEU.PH.QBRA	1207
8.445. PRECR.QB.PH	1209
8.446. PRECR_SRA.PH.W	1211
8.447. PRECR_SRA_R.PH.W	1213
8.448. PRECRQ.PH.W	1215
8.449. PRECRQ.QB.PH	1217
8.450. PRECRQ_RS.PH.W	1219
8.451. PRECRQU_S.QB.PH	1222
8.452. PREF	1225
8.453. PREFX	1231
8.454. PREPEND	1233
8.455. PUL.PS	1235

8.456. PUU.PS	1237
8.457. RADDU.W.QB	1239
8.458. RDDSP	1241
8.459. RDHWR	1244
8.460. RDPGPR	1247
8.461. RECIP.D	1249
8.462. RECIP.S	1251
8.463. REPL.PH	1253
8.464. REPL.QB	1255
8.465. REPLV.PH	1257
8.466. REPLV.QB	1259
8.467. RESTORE	1261
8.468. RESTORE.JRC	1263
8.469. RINT.D	1265
8.470. RINT.S	1267
8.471. ROTR	1269
8.472. ROTRV	1272
8.473. ROTX	1274
8.474. ROUND.L.D	1276
8.475. ROUND.L.S	1278
8.476. ROUND.W.D	1280
8.477. ROUND.W.S	1282
8.478. RSQRT.D	1284
8.479. RSQRT.S	1286
8.480. SAVE	1288
8.481. SB	1290
8.482. SBE	1294
8.483. SBX	1296
8.484. SC	1298
8.485. SCD	1302
8.486. SCDP	1305
8.487. SCE	1307
8.488. SCWP	1309
8.489. SCWPE	1312
8.490. SD	1314
8.491. SDBBP	1317
8.492. SDC1	1320
8.493. SDC2	1324
8.494. SDL	1326
8.495. SDM	1328
8.496. SDPC	1330

8.497. SDR	1332
8.498. SDX	1334
8.499. SDXC1	1336
8.500. SDXS	1338
8.501. SEB.....	1340
8.502. SEH	1342
8.503. SEL.D.....	1344
8.504. SEL.S	1347
8.505. SELEQZ.....	1350
8.506. SELEQZ.D.....	1352
8.507. SELEQZ.S	1355
8.508. SELNEZ.....	1358
8.509. SELNEZ.D.....	1360
8.510. SELNEZ.S	1363
8.511. SEQI.....	1366
8.512. SH.....	1367
8.513. SHE	1371
8.514. SHILO	1373
8.515. SHILOV	1375
8.516. SHLL.PH.....	1377
8.517. SHLL.QB.....	1379
8.518. SHLL_S.PH.....	1381
8.519. SHLL_S.W.....	1383
8.520. SHLLV.PH.....	1385
8.521. SHLLV.QB.....	1387
8.522. SHLLV_S.PH.....	1389
8.523. SHLLV_S.W	1391
8.524. SHRA.PH.....	1393
8.525. SHRA.QB.....	1395
8.526. SHRA_R.PH	1397
8.527. SHRA_R.QB	1399
8.528. SHRA_R.W	1401
8.529. SHRAV.PH.....	1403
8.530. SHRAV.QB.....	1405
8.531. SHRAV_R.PH	1407
8.532. SHRAV_R.QB	1409
8.533. SHRAV_R.W	1411
8.534. SHRL.PH.....	1413
8.535. SHRL.QB.....	1415
8.536. SHRLV.PH.....	1417
8.537. SHRLV.QB.....	1419

8.538. SHX	1421
8.539. SHXS	1423
8.540. SIGRIE	1425
8.541. SLL	1427
8.542. SLLV	1431
8.543. SLT	1433
8.544. SLTI	1435
8.545. SLTIU	1437
8.546. SLTU	1440
8.547. SOV	1443
8.548. SQRT.D	1444
8.549. SQRT.S	1446
8.550. SRA	1448
8.551. SRAV	1450
8.552. SRL	1452
8.553. SRLV	1455
8.554. SUB	1457
8.555. SUB.D	1460
8.556. SUB.PS	1463
8.557. SUB.S	1465
8.558. SUBQ.PH	1468
8.559. SUBQ_S.PH	1470
8.560. SUBQ_S.W	1472
8.561. SUBQH.PH	1474
8.562. SUBQH.W	1476
8.563. SUBQH_R.PH	1478
8.564. SUBQH_R.W	1480
8.565. SUBU	1482
8.566. SUBU.PH	1486
8.567. SUBU.QB	1488
8.568. SUBU_S.PH	1490
8.569. SUBU_S.QB	1492
8.570. SUBUH.QB	1494
8.571. SUBUH_R.QB	1496
8.572. SUXC1	1498
8.573. SW	1500
8.574. SWC1	1506
8.575. SWC2	1510
8.576. SWE	1512
8.577. SWL	1514
8.578. SWM	1517

8.579. SWP	1520
8.580. SWPC	1522
8.581. SWR	1524
8.582. SWX	1527
8.583. SWXC1	1529
8.584. SWXS	1531
8.585. SYNC	1533
8.586. SYNCL	1535
8.587. SYNCIE	1538
8.588. SYSCALL	1540
8.589. TEQ	1543
8.590. TEQI	1545
8.591. TGE	1547
8.592. TGEI	1549
8.593. TGEIU	1551
8.594. TGEU	1553
8.595. TLBINV	1555
8.596. TLBINVF	1557
8.597. TLBP	1559
8.598. TLBR	1561
8.599. TLBWI	1563
8.600. TLBWR	1565
8.601. TLT	1567
8.602. TLTI	1569
8.603. TLTIU	1571
8.604. TLTU	1573
8.605. TNE	1575
8.606. TNEI	1577
8.607. TRUNC.L.D	1579
8.608. TRUNC.L.S	1581
8.609. TRUNC.W.D	1583
8.610. TRUNC.W.S	1585
8.611. UALDM	1587
8.612. UALH	1589
8.613. UALWM	1591
8.614. UASDM	1593
8.615. UASH	1595
8.616. UASWM	1597
8.617. WAIT	1599
8.618. WRDSP	1601
8.619. WRPGR	1604

8.620. WSBH	1606
8.621. XOR	1608
8.622. XORI	1612
Appendix A: MIPS Encoding Summary	1614
A.1. MIPS pool	1614
A.2. SPECIAL pool	1616
A.3. REGIMM pool	1618
A.4. COP0 pool	1619
A.5. COP0.CO pool	1620
A.6. COP1 pool	1621
A.7. COP1.COND.S pool	1621
A.8. COP1.S pool	1622
A.9. COP1.COND.D pool	1623
A.10. COP1.D pool	1623
A.11. COP1.CMP.S pool	1624
A.12. COP1.FMT20 pool	1625
A.13. COP1.CMP.D pool	1626
A.14. COP1.FMT21 pool	1626
A.15. COP1.COND.PS pool	1627
A.16. COP1.PS pool	1627
A.17. COP1X pool	1628
A.18. SPECIAL2 pool	1629
A.19. SPECIAL3 pool	1630
A.20. SPECIAL3.INDEXED pool	1633
A.21. SPECIAL3.CRC pool	1634
A.22. ADDU.QB pool	1634
A.23. CMPU.EQ.QB pool	1635
A.24. ABSQ_S.PH pool	1636
A.25. ADDUH.QB pool	1637
A.26. DPA.W.PH pool	1638
A.27. APPEND pool	1642
A.28. EXTR.W pool	1643
A.29. PCREL pool	1643
A.30. PCREL.LS pool	1644
Appendix B: MIPS16E Encoding Summary	1645
B.1. MIPS16E pool	1645
B.2. I8 pool	1645
B.3. EXTEND pool	1646
B.4. EXTEND.I8 pool	1647
Appendix C: microMIPS Encoding Summary	1648
C.1. microMIPS pool	1648

C.2. POOL32A pool	1655
C.3. _POOL32A0 pool	1657
C.4. _POOL32A4 pool	1658
C.5. POOL32Ayf pool	1659
C.6. POOL32Ayf_3 pool	1660
C.7. POOL32Axf pool	1660
C.8. POOL32Axf_3 pool	1665
C.9. POOL32Axf.DSP pool	1665
C.10. _POOL32A5 pool	1667
C.11. _POOL32A7 pool	1670
C.12. POOL32B.MULTI32 pool	1670
C.13. POOL32B.PAIR32 pool	1671
C.14. POOL32B.BRANCH pool	1672
C.15. POOL32I pool	1672
C.16. POOL32I_0 pool	1673
C.17. POOL32I_1 pool	1673
C.18. POOL32I_2 pool	1674
C.19. POOL32I_3 pool	1674
C.20. POOL32I_4 pool	1675
C.21. POOL32I_6 pool	1675
C.22. POOL32I_7.LEGACY pool	1676
C.23. REGIMM32 pool	1676
C.24. POOL16C pool	1677
C.25. POOL16C.LOGICAL pool	1678
C.26. POOL16C.LOGICAL_R6 pool	1678
C.27. POOL16C.MULTI pool	1679
C.28. POOL16C.R6.BREAK pool	1679
C.29. POOL16C.R6.JR pool	1680
C.30. POOL16C.R6.JRSP pool	1681
C.31. POOL16C.R6.RR pool	1681
C.32. POOL32F pool	1682
C.33. POOL32FXF pool	1685
C.34. POOL32FXF.WIDE pool	1687
C.35. POOL32S pool	1689
C.36. POOL32Sxf_3 pool	1691
C.37. POOL32Sxf_4 pool	1692
C.38. POOL32B pool	1698
C.39. POOL32C pool	1699
C.40. R6PCREL pool	1699
C.41. R6PCREL_11 pool	1700
C.42. R6PCREL_11_1 pool	1701

Appendix D: nanoMIPS Encoding Summary	1702
D.1. NM pool	1702
D.2. P32 pool	1702
D.3. P32.BREAK pool	1706
D.4. P32.SDBBP pool	1707
D.5. P32.SYSCALL pool	1707
D.6. P32A pool	1708
D.7. _POOL32A0 pool	1708
D.8. _POOL32A0_0 pool	1709
D.9. _POOL32A0_1 pool	1711
D.10. CRC32 pool	1712
D.11. _POOL32A5 pool	1713
D.12. _POOL32A7 pool	1715
D.13. P.LSX pool	1716
D.14. PP.LSX pool	1716
D.15. PP.LSXS pool	1717
D.16. POOL32Axf pool	1717
D.17. POOL32Axf_1 pool	1718
D.18. POOL32Axf_1_0 pool	1719
D.19. POOL32Axf_1_1 pool	1719
D.20. POOL32Axf_1_3 pool	1720
D.21. POOL32Axf_1_4 pool	1720
D.22. POOL32Axf_1_5 pool	1721
D.23. MAQ_S[A].W.PHR pool	1722
D.24. MAQ_S[A].W.PHL pool	1722
D.25. POOL32Axf_1_7 pool	1723
D.26. POOL32Axf_2 pool	1723
D.27. POOL32Axf_2(DSP)_0_7 pool	1724
D.28. POOL32Axf_2(DSP)_8_15 pool	1724
D.29. POOL32Axf_2(DSP)_16_23 pool	1725
D.30. POOL32Axf_2(DSP)_24_31 pool	1725
D.31. POOL32Axf_4 pool	1726
D.32. P.DI pool	1728
D.33. POOL32Axf_5 pool	1729
D.34. POOL32Axf_5_group0 pool	1730
D.35. POOL32Axf_7 pool	1730
D.36. P32A.CP0 pool	1731
D.37. P.CP0 pool	1731
D.38. P32A.ERET pool	1732
D.39. P.ERET pool	1733
D.40. P32A.TLB pool	1733

D.41. P.TLB pool	1734
D.42. P.BAL pool	1735
D.43. P.GP.D pool	1735
D.44. P.GP.W pool	1736
D.45. P.GP.BH pool	1736
D.46. P.GP.M64 pool	1737
D.47. P.J pool	1738
D.48. P48 pool	1739
D.49. P48I pool	1739
D.50. P.ROTX pool	1740
D.51. P.U12 pool	1741
D.52. P.SR pool	1742
D.53. PP.SR pool	1742
D.54. P.SHIFT pool	1743
D.55. P.EXT pool	1744
D.56. P32.SHIFT pool	1744
D.57. P32A.U12 pool	1745
D.58. P.LS.U12 pool	1745
D.59. P32.LSU pool	1746
D.60. P.BR1 pool	1746
D.61. P.BR3A pool	1747
D.62. P32B pool	1748
D.63. POOL32F pool	1748
D.64. POOL32F_0 pool	1749
D.65. RINT.fmt pool	1749
D.66. SELEQZ.fmt pool	1750
D.67. CLASS.fmt pool	1750
D.68. SELNEZ.fmt pool	1751
D.69. SEL.fmt pool	1751
D.70. MADDF.fmt pool	1752
D.71. MSUBF.fmt pool	1752
D.72. POOL32F_3 pool	1753
D.73. MIN.fmt pool	1753
D.74. MAX.fmt pool	1754
D.75. MINA.fmt pool	1754
D.76. MAXA.fmt pool	1755
D.77. POOL32Fxf pool	1755
D.78. POOL32Fxf_0 pool	1756
D.79. CVT.L.D.fmt pool	1758
D.80. CVT.L.S.fmt pool	1758
D.81. RSQRT.fmt pool	1759

D.82. FLOOR.L.D.fmt pool	1759
D.83. FLOOR.L.S.fmt pool	1760
D.84. CVT.W.D.fmt pool	1760
D.85. CVT.W.S.fmt pool	1761
D.86. SQRT.fmt pool	1761
D.87. FLOOR.W.D.fmt pool	1762
D.88. FLOOR.W.S.fmt pool	1762
D.89. RECIP.fmt pool	1763
D.90. CEIL.L.D.fmt pool	1763
D.91. CEIL.L.S.fmt pool	1764
D.92. CEIL.W.D.fmt pool	1764
D.93. CEIL.W.S.fmt pool	1765
D.94. TRUNC.L.D.fmt pool	1765
D.95. TRUNC.L.S.fmt pool	1766
D.96. TRUNC.W.D.fmt pool	1766
D.97. TRUNC.W.S.fmt pool	1767
D.98. ROUND.L.D.fmt pool	1767
D.99. ROUND.L.S.fmt pool	1768
D.100. ROUND.W.D.fmt pool	1768
D.101. ROUND.W.S.fmt pool	1769
D.102. POOL32Fxf_1 pool	1769
D.103. CVT.D.L.fmt pool	1770
D.104. CVT.D.S.fmt pool	1771
D.105. CVT.S.D.fmt pool	1771
D.106. CVT.S.L.fmt pool	1772
D.107. CVT.D.W.fmt pool	1772
D.108. CVT.S.W.fmt pool	1773
D.109. CMP.fmt pool	1773
D.110. P.LS.S9 pool	1774
D.111. P.LS.S0 pool	1774
D.112. P.LS.E0 pool	1775
D.113. P32.LS.PAIR pool	1775
D.114. P32.LS.S9 pool	1776
D.115. P32A.EVA pool	1776
D.116. P32A.MEM pool	1777
D.117. P.BR2 pool	1777
D.118. POOL32S pool	1778
D.119. POOL32S_0 pool	1778
D.120. POOL32S_4 pool	1779
D.121. P.BRI pool	1780
D.122. P16 pool	1780

D.123. P16.MV pool	1783
D.124. P16.RI pool	1783
D.125. P16.SYSCALL pool	1784
D.126. P16.SR pool	1784
D.127. P16C pool	1785
D.128. POOL16C_0 pool	1785
D.129. POOL16C_00 pool	1786
D.130. P16.A1 pool	1786
D.131. P16.A2 pool	1787
D.132. P16.ADDU pool	1788

This reference is intended for software developers and hardware designers implementing, extending, or verifying the MIPS architecture. It provides machine-readable encodings, instruction decode, executable operation semantics, processor state, and shared architectural behavior from one Python specification.

Chapter 1. Instruction Summary

This chapter lists each instruction format, its assembly syntax, a brief summary of its operation, and its architectural availability. The precise behavior is defined by the corresponding instruction definition.

1.1. Availability Expressions

Availability is shown as a Boolean expression over the architecture profile and optional-feature flags. `&&`, `||`, and `!` mean AND, OR, and NOT. These are the same predicates used by the executable Python specification.

1.1.1. Architecture Profile

Field	Architectural source	Values	Meaning
AR	Config.AR	R1 = 0 — Release 1. R2_3_5 = 1 — Release 2, 3, or 5; Release 3/5 additions use feature flags. R6 = 2 — Release 6.	MIPS Architecture Revision level.
AT	Config.AT	MIPS32 = 0 — 32-bit architecture. MIPS64_32 = 1 — 64-bit architecture restricted to compatibility segments. MIPS64 = 2 — 64-bit architecture with all address segments.	Architecture type and address-space capability.
MMAR	Config3.MMAR	R3_5 = 0 — microMIPS Release 3/5. R6 = 2 — microMIPS Release 6. nanoMIPS_R6 = 3 — nanoMIPS Release 6.	microMIPS or nanoMIPS Architecture Revision level.
ISA	Config3.ISA	MIPS_ONLY = 0 — Only MIPS. COMPRESSED_ONLY = 1 — Only nanoMIPS or microMIPS. MIPS_AND_MICROMIPS_MIPS_RESET = 2 — MIPS and microMIPS; reset selects MIPS. MIPS_AND_MICROMIPS_MICROMIPS_RESET = 3 — MIPS and microMIPS; reset selects microMIPS.	Implemented instruction-set families.
PMJ	Config5.PMJ	0 = 0 — PRA major version 0. 1 = 1 — PRA major version 1. 2 = 2 — PRA major version 2. 3 = 3 — PRA major version 3. 4 = 4 — PRA major version 4. 5 = 5 — PRA major version 5. 6 = 6 — PRA major version 6. 7 = 7 — PRA major version 7.	Privileged Resource Architecture major version.

Field	Architectural source	Values	Meaning
PABITS	Processor profile	Integer in the range 13 through 59, or 11 through 59 when one-kilobyte pages are implemented.	Number of implemented physical-address bits.
SEGBITS	Processor profile	32 for MIPS32; integer in the range 32 through 62 for MIPS64.	Number of implemented virtual-address bits in each MIPS64 address-space region; 32 for a 32-bit architecture.

1.1.2. Optional Architecture Features

An optional-feature flag is true only when that capability is implemented. The configuration-field spelling below is the architectural discovery mechanism; a processor profile uses the identically named Python flag when defining instruction, register, or field availability.

Flag	Architectural definition	Meaning
NMS	Config5.NMS == 1	The nanoMIPS subset.
EVA	Config5.EVA == 1	Enhanced Virtual Addressing.
CP1	Config1.FP == 1	Coprocessor 1 is present.
CP2	Config1.C2 == 1	Coprocessor 2 is present.
DSP	Config3.DSP == 1	The DSP ASE is present.
DSP2	Config3.DSP2 == 1	DSP Module Revision 2 is present.
FPU64	FIR.F64 == 1	64-bit floating-point registers.
FP_D	FIR.D == 1	D-format floating-point operations.
FP_L	FIR.L == 1	L-format integer conversions.
FP_PS	FIR.PS == 1	Paired-single floating-point operations.
FP_2008	FIR.HAS2008 == 1	IEEE-754-2008 floating-point behavior.
FREP	FIR.FREP == 1	User-mode FR emulation switching.
UFRP	FIR.UFRP == 1	User-mode Status.FR switching.
MIPS16E	Config1.CA == 1	The MIPS16E ASE.
MIPS16E2	MIPS16E && processor feature MIPS16E2	The MIPS16E2 ASE extension.
DEBUG	Config1.EP == 1	EJTAG/debug support.
CRCP	Config5.CRCP == 1	CRC instructions.
VP	Config5.VP == 1	Virtual processors.
GI	Config5.GI != 0	Global TLB invalidation.

Flag	Architectural definition	Meaning
MVH	Config5.MVH == 1	High CP0 moves.
BPG	Config3.BPG == 1	Big-page CP0 registers.
LPA	Config3.LPA == 1	Large physical addresses.
XPA	Config5.XPA == 1	Extended physical addressing.
SP	Config3.SP == 1	Small pages.
RXI	Config3.RXI == 1	Read/execute-inhibit TLB bits.
ULRI	Config3.ULRI == 1	The UserLocal register.
BADINSTR	Config3.BI == 1	The BadInstr register.
BADINSTRP	Config3.BP == 1	The BadInstrP register.
EXTENDED_ASID	Config4.AE == 1	Ten-bit ASIDs.
TLBINV	Config4.IE >= 2	TLB invalidation instructions.
SMARTMIPS	Config3.SM == 1	The SmartMIPS ASE.
KSCRATCH	Config4.KScrExist != 0	One or more kernel scratch registers.
SEGMENT_CONTROL	Config3.SC == 1	Programmable segmentation control.
LLX_SCX	Config5.XNP == 0	Extended LL/SC instructions.
LLB	Config5.LLB == 1	The architectural LL bit.
SUPERVISOR	processor feature Supervisor	Supervisor mode.
VINT	Config3.VInt == 1	Vectored interrupts.
VEIC	Config3.VEIC == 1	External interrupt controller mode.
L2C	Config2.SL != 0	A secondary cache.
L3C	Config2.TL != 0	A tertiary cache.
PERF_COUNTER	Config1.PC == 1	Performance counters.
SRS	SRSctl.HSS != 0	GPR shadow register sets.

1.2. MIPS

Format	Assembly	Operation	Availability
abs.d	ABS.D fd, fs	abs.d.operation	true
abs.ps	ABS.PS fd, fs	abs.ps.operation	AR < R6
abs.s	ABS.S fd, fs	abs.s.operation	true
absq_s.ph	ABSQ_S.PH rd, rs	Take the saturating absolute value of two signed halfword lanes.	AR >= R2_3_5 && DSP
absq_s.qb	ABSQ_S.QB rd, rs	Take the saturating absolute value of four signed byte lanes.	AR >= R2_3_5 && DSP2

Format	Assembly	Operation	Availability
absq_s.w	ABSQ_S.W rd, rs	Take the saturating absolute value of a signed word lane.	AR >= R2_3_5 && DSP
add	ADD rd, rs, rt	Add signed 32-bit GPR operands and trap on integer overflow.	true
add.d	ADD.D fd, fs, ft	add.d.operation	true
add.ps	ADD.PS fd, fs, ft	add.ps.operation	AR < R6
add.s	ADD.S fd, fs, ft	add.s.operation	true
addi	ADDI rt, rs, imm	Add a signed 16-bit immediate to a signed 32-bit GPR and trap on overflow.	AR < R6
addiu	ADDIU rt, rs, imm	Add modulo 2**32 and sign-extend the 32-bit result to the GPR width.	true
addiupc	ADDIUPC rt, imm	Add a 32-bit displacement to the word-aligned architectural PC. ADDIUPC/ADDIUPCR3 are the 32-bit PC-relative operations. This remains a 32-bit result even when the surrounding processor has 64-bit GPRs; the nanoMIPS P64 decoder selects DADDIUPC instead of reaching this operation.	AR >= R6
addq.ph	ADDQ.PH rd, rs, rt	Add signed halfword lanes without saturation.	AR >= R2_3_5 && DSP
addq_s.ph	ADDQ_S.PH rd, rs, rt	Add signed halfword lanes with signed saturation.	AR >= R2_3_5 && DSP
addq_s.w	ADDQ_S.W rd, rs, rt	Add signed word lanes with signed saturation.	AR >= R2_3_5 && DSP
addqh.ph	ADDQH.PH rd, rs, rt	Add signed halfword lanes and halve the results.	AR >= R2_3_5 && DSP2
addqh.w	ADDQH.W rd, rs, rt	Add signed word lanes and halve the result.	AR >= R2_3_5 && DSP2
addqh_r.ph	ADDQH_R.PH rd, rs, rt	Rounded signed half-add of two halfword lanes.	AR >= R2_3_5 && DSP2
addqh_r.w	ADDQH_R.W rd, rs, rt	Rounded signed half-add of two word lanes.	AR >= R2_3_5 && DSP2
addsc	ADDSC rd, rs, rt	Add unsigned words and write the carry to DSPControl.C.	AR >= R2_3_5 && DSP

Format	Assembly	Operation	Availability
addu	ADDU rd, rs, rt	Add two 32-bit GPR values and sign-extend the result to GPR width.	true
addu.ph	ADDU.PH rd, rs, rt	Add unsigned halfword lanes modulo 16 bits.	AR >= R2_3_5 && DSP2
addu.qb	ADDU.QB rd, rs, rt	Add unsigned byte lanes modulo 8 bits and report carry.	AR >= R2_3_5 && DSP
addu_s.ph	ADDU_S.PH rd, rs, rt	Add unsigned halfword lanes with saturation and report carry.	AR >= R2_3_5 && DSP2
addu_s.qb	ADDU_S.QB rd, rs, rt	Add unsigned byte lanes with saturation and report carry.	AR >= R2_3_5 && DSP
adduh.qb	ADDUH.QB rd, rs, rt	Average unsigned byte lanes by an unrounded right shift.	AR >= R2_3_5 && DSP
adduh_r.qb	ADDUH_R.QB rd, rs, rt	Average unsigned byte lanes with rounding.	AR >= R2_3_5 && DSP
addwc	ADDWC rd, rs, rt	Add signed words with DSPControl.C and report signed overflow.	AR >= R2_3_5 && DSP
align	ALIGN rd, rs, rt, bp	align.operation	AR >= R6
alnv.ps	ALNV.PS fd, fs, ft, rs	alnv.ps.operation	AR < R6
aluipc	ALUIPC rt, imm	Add the aligned signed displacement to next_pc and clear low bits.	AR >= R6
and	AND rd, rs, rt	Compute a full-GPR logical AND and write the result to a GPR.	true
andi	ANDI rt, rs, imm	Compute a full-GPR logical AND with a zero-extended immediate.	true
append	APPEND rt, rs, sa	append.operation	AR >= R2_3_5 && DSP2
aui	AUI rt, rs, imm	aui.operation	AR >= R6
auipc	AUIPC rs, offset	auipc.operation	AR >= R6
bal	BAL offset	Branch after one delay slot and write the post-slot return address.	AR >= R6
balc	BALC offset	Perform a compact PC-relative call without a delay slot.	AR >= R6
balgn	BALIGN rt, rs, bp	balgn.operation	AR >= R2_3_5 && DSP2
bc	BC offset	Perform an unconditional compact PC-relative branch.	AR >= R6

Format	Assembly	Operation	Availability
bc1eqz	BC1EQZ ft, offset	bc1eqz.operation	AR >= R6
bc1f	BC1F cc, offset	bc1f.operation	AR < R6
bc1fl	BC1FL cc, offset	bc1fl.operation	AR < R6
bc1nez	BC1NEZ ft, offset	bc1nez.operation	AR >= R6
bc1t	BC1T cc, offset	bc1t.operation	AR < R6
bc1tl	BC1TL cc, offset	bc1tl.operation	AR < R6
beq	BEQ rs, rt, offset	Compare two GPRs and resolve the branch after one delay slot.	true
beqc	BEQC rs, rt, offset	Compare two GPRs and perform a compact branch when equal.	AR >= R6
beql	BEQL rs, rt, offset	beql.operation	AR < R6
beqzalc	BEQZALC rt, offset	beqzalc.operation	AR >= R6
beqzc	BEQZC rt, offset	Compact-branch when the compact source GPR is zero.	AR >= R6
bgec	BGEC rs, rt, offset	Branch compactly when signed rs is greater than or equal to rt.	AR >= R6
bgeuc	BGEUC rs, rt, offset	Branch compactly when unsigned rs is greater than or equal to rt.	AR >= R6
bgez	BGEZ rs, offset	bgez.operation	true
bgezal	BGEZAL rs, offset	bgezal.operation	AR < R6
bgezalc	BGEZALC rt, offset	bgezalc.operation	AR >= R6
bgezall	BGEZALL rs, offset	bgezall.operation	AR < R6
bgezc	BGEZC rt, offset	Compact-branch when the full-width source GPR is signed non-negative.	AR >= R6
bgezl	BGEZL rs, offset	bgezl.operation	AR < R6
bgtz	BGTZ rs, offset	bgtz.operation	true
bgtzalc	BGTZALC rt, offset	bgtzalc.operation	AR >= R6
bgtzc	BGTZC rt, offset	bgtzc.operation	AR >= R6
bgtzl	BGTZL rs, offset	bgtzl.operation	AR < R6
bitrev	BITREV rd, rt	bitrev.operation	DSP && AR >= R2_3_5
bitswap	BITSWAP rd, rt	bitswap.operation	AR >= R6
blez	BLEZ rs, offset	blez.operation	true
blezalc	BLEZALC rt, offset	blezalc.operation	AR >= R6

Format	Assembly	Operation	Availability
blezc	BLEZC <i>rt</i> , <i>offset</i>	blezc.operation	AR >= R6
blezl	BLEZL <i>rs</i> , <i>offset</i>	blezl.operation	AR < R6
bltc	BLTC <i>rs</i> , <i>rt</i> , <i>offset</i>	Branch compactly when signed <i>rs</i> is less than signed <i>rt</i> .	AR >= R6
bltuc	BLTUC <i>rs</i> , <i>rt</i> , <i>offset</i>	Branch compactly when unsigned <i>rs</i> is less than <i>rt</i> .	AR >= R6
bltz	BLTZ <i>rs</i> , <i>offset</i>	bltz.operation	true
bltzal	BLTZAL <i>rs</i> , <i>offset</i>	bltzal.operation	AR < R6
bltzalc	BLTZALC <i>rt</i> , <i>offset</i>	bltzalc.operation	AR >= R6
bltzall	BLTZALL <i>rs</i> , <i>offset</i>	bltzall.operation	AR < R6
bltzc	BLTZC <i>rt</i> , <i>offset</i>	bltzc.operation	AR >= R6
bltzl	BLTZL <i>rs</i> , <i>offset</i>	bltzl.operation	AR < R6
bne	BNE <i>rs</i> , <i>rt</i> , <i>offset</i>	bne.operation	true
bnecl	BNECL <i>rs</i> , <i>rt</i> , <i>offset</i>	Compare two GPRs and perform a compact branch when they differ.	AR >= R6
bnel	BNEL <i>rs</i> , <i>rt</i> , <i>offset</i>	bnel.operation	AR < R6
bnezalc	BNEZALC <i>rt</i> , <i>offset</i>	bnezalc.operation	AR >= R6
bnezcl	BNEZCL <i>rt</i> , <i>offset</i>	Compact-branch when the compact source GPR is nonzero.	AR >= R6
bnvc	BNVC <i>rs</i> , <i>rt</i> , <i>offset</i>	bnvc.operation	AR >= R6
bovc	BOVC <i>rs</i> , <i>rt</i> , <i>offset</i>	bovc.operation	AR >= R6
bposge32	BPOSGE32 <i>offset</i>	bposge32.operation	DSP
bposge32c	BPOSGE32C <i>offset</i>	bposge32c.operation	AR >= R6 && DSP && DSP2
break	BREAK <i>code</i>	Raise a breakpoint exception.	true
c.cond.d	C.cond.D <i>cc</i> , <i>fs</i> , <i>ft</i> , <i>condition</i>	c.cond.d.operation	AR < R6
c.cond.ps	C.cond.PS <i>cc</i> , <i>fs</i> , <i>ft</i> , <i>condition</i>	Compare both PS lanes and write the selected FCC pair.	AR < R6
c.cond.s	C.cond.S <i>cc</i> , <i>fs</i> , <i>ft</i> , <i>condition</i>	c.cond.s.operation	AR < R6
cache	CACHE <i>op</i> , <i>offset(rs)</i>	Issue an implementation-defined cache operation through the backend.	AR < R6
cache.r6	CACHE <i>op</i> , <i>offset(rs)</i>	Issue an implementation-defined cache operation through the backend.	AR >= R6

Format	Assembly	Operation	Availability
ceil.l.d	CEIL.L.D fd, fs	ceil.l.d.operation	true
ceil.l.s	CEIL.L.S fd, fs	ceil.l.s.operation	true
ceil.w.d	CEIL.W.D fd, fs	ceil.w.d.operation	true
ceil.w.s	CEIL.W.S fd, fs	ceil.w.s.operation	true
cfc1	CFC1 rt, cs	cfc1.operation	true
class.d	CLASS.D fd, fs	class.d.operation	AR >= R6
class.s	CLASS.S fd, fs	class.s.operation	AR >= R6
clo	CLO rt, rs	Count leading one bits in the low 32 bits of rs and write the count.	AR < R6
r6.clo	CLO rt, rs	Count leading one bits in the low 32 bits of rs and write the count.	AR >= R6
clz	CLZ rt, rs	Count leading zeroes in the low 32 bits and write the count to rt.	AR < R6
r6.clz	CLZ rt, rs	Count leading zeroes in the low 32 bits and write the count to rt.	AR >= R6
cmp.d	CMP.D fd, fs, ft, condition	cmp.d.operation	AR >= R6
cmp.eq.ph	CMP.EQ.PH rs, rt	Compare signed halfword lanes for equality and write Ccond.	AR >= R2_3_5 && DSP
cmp.le.ph	CMP.LE.PH rs, rt	Compare signed halfword lanes for less-than-or-equal and write Ccond.	AR >= R2_3_5 && DSP
cmp.lt.ph	CMP.LT.PH rs, rt	Compare signed halfword lanes for less-than and write Ccond.	AR >= R2_3_5 && DSP
cmp.s	CMP.S fd, fs, ft, condition	cmp.s.operation	AR >= R6
cmpgdu.eq.qb	CMPGDU.EQ.QB rd, rs, rt	Compare unsigned byte lanes, writing both Ccond and a GPR mask.	AR >= R2_3_5 && DSP2
cmpgdu.le.qb	CMPGDU.LE.QB rd, rs, rt	Compare unsigned byte lanes for ≤, writing Ccond and a GPR mask.	AR >= R2_3_5 && DSP2
cmpgdu.lt.qb	CMPGDU.LT.QB rd, rs, rt	Compare unsigned byte lanes for less-than, writing Ccond and a GPR mask.	AR >= R2_3_5 && DSP2
cmpgu.eq.qb	CMPGU.EQ.QB rd, rs, rt	Compare unsigned byte lanes for equality and write a GPR mask.	AR >= R2_3_5 && DSP
cmpgu.le.qb	CMPGU.LE.QB rd, rs, rt	Compare unsigned byte lanes for less-than-or-equal and write a GPR mask.	AR >= R2_3_5 && DSP

Format	Assembly	Operation	Availability
cmpgu.lt.qb	CMPGU.LT.QB rd, rs, rt	Compare unsigned byte lanes for less-than and write a GPR mask.	AR >= R2_3_5 && DSP
cmpu.eq.qb	CMPU.EQ.QB rs, rt	Compare unsigned byte lanes for equality and write Ccond.	AR >= R2_3_5 && DSP
cmpu.le.qb	CMPU.LE.QB rs, rt	Compare unsigned byte lanes for less-than-or-equal and write Ccond.	AR >= R2_3_5 && DSP
cmpu.lt.qb	CMPU.LT.QB rs, rt	Compare unsigned byte lanes for less-than and write Ccond.	AR >= R2_3_5 && DSP
crc32cd	CRC32CD rs	crc32cd.operation	AR >= R6 && AT >= MIPS64_32 && CRCP
crc32d	CRC32D rs	crc32d.operation	AR >= R6 && AT >= MIPS64_32 && CRCP
ctc1	CTC1 rt, cs	ctc1.operation	true
cvt.d.l	CVT.D.L fd, fs	cvt.d.l.operation	true
cvt.d.s	CVT.D.S fd, fs	cvt.d.s.operation	true
cvt.d.w	CVT.D.W fd, fs	cvt.d.w.operation	true
cvt.l.d	CVT.L.D fd, fs	cvt.l.d.operation	true
cvt.l.s	CVT.L.S fd, fs	cvt.l.s.operation	true
cvt.ps.s	CVT.PS.S fd, fs, ft	cvt.ps.s.operation	AR < R6
cvt.s.d	CVT.S.D fd, fs	cvt.s.d.operation	true
cvt.s.l	CVT.S.L fd, fs	cvt.s.l.operation	true
cvt.s.pl	CVT.S.PL fd, fs	cvt.s.pl.operation	AR < R6
cvt.s.pu	CVT.S.PU fd, fs	cvt.s.pu.operation	AR < R6
cvt.s.w	CVT.S.W fd, fs	cvt.s.w.operation	true
cvt.w.d	CVT.W.D fd, fs	cvt.w.d.operation	true
cvt.w.s	CVT.W.S fd, fs	cvt.w.s.operation	true
dadd	DADD rs, rt	dadd.operation	AT >= MIPS64_32
daddi	DADDI rd, rs, imm	daddi.operation	AT >= MIPS64_32 && AR < R6
daddiu	DADDIU rd, rs, imm	daddiu.operation	AT >= MIPS64_32
daddu	DADDU rs, rt	daddu.operation	AT >= MIPS64_32
dahi	DAHI imm	dahi.operation	AR >= R6 && AT >= MIPS64_32
dalign	DALIGN rd, rs, rt, bp	dalign.operation	AR >= R6 && AT >= MIPS64_32

Format	Assembly	Operation	Availability
<code>dati</code>	<code>DATI imm</code>	<code>dati.operation</code>	AR >= R6 && AT >= MIPS64_32
<code>dau</code>	<code>DAUI rt, rs, imm</code>	<code>dau.operation</code>	AR >= R6 && AT >= MIPS64_32
<code>dbitswap</code>	<code>DBITSWAP rs</code>	<code>dbitswap.operation</code>	AT >= MIPS64_32 && AR >= R6
<code>dclo</code>	<code>DCLO rs</code>	<code>dclo.operation</code>	AT >= MIPS64_32 && AR < R6
<code>dclo.r6</code>	<code>DCLO rs</code>	<code>dclo.operation</code>	AT >= MIPS64_32 && AR >= R6
<code>dclz</code>	<code>DCLZ rs</code>	<code>dclz.operation</code>	AT >= MIPS64_32 && AR < R6
<code>dclz.r6</code>	<code>DCLZ rs</code>	<code>dclz.operation</code>	AT >= MIPS64_32 && AR >= R6
<code>ddiv.r6</code>	<code>DDIV rs, rt</code>	<code>ddiv.operation</code>	AT >= MIPS64_32 && AR >= R6
<code>ddiv</code>	<code>DDIV rt</code>	<code>ddiv.operation</code>	AT >= MIPS64_32 && AR < R6
<code>ddivu.r6</code>	<code>DDIVU rs, rt</code>	<code>ddivu.operation</code>	AT >= MIPS64_32 && AR >= R6
<code>ddivu</code>	<code>DDIVU rt</code>	<code>ddivu.operation</code>	AT >= MIPS64_32 && AR < R6
<code>deret</code>	<code>DERET</code>	<code>deret.operation</code>	true
<code>dext</code>	<code>DEXT rs, pos, size</code>	<code>dext.operation</code>	AT >= MIPS64_32 && AR >= R2_3_5
<code>dextm</code>	<code>DEXTM rs, pos, size</code>	<code>dextm.operation</code>	AT >= MIPS64_32 && AR >= R2_3_5
<code>dextu</code>	<code>DEXTU rs, pos, size</code>	<code>dextu.operation</code>	AT >= MIPS64_32 && AR >= R2_3_5
<code>di</code>	<code>DI rt</code>	<code>di.operation</code>	AR >= R2_3_5
<code>dins</code>	<code>DINS rs, pos, size</code>	<code>dins.operation</code>	AT >= MIPS64_32 && AR >= R2_3_5
<code>dinsm</code>	<code>DINSM rs, pos, size</code>	<code>dinsm.operation</code>	AT >= MIPS64_32 && AR >= R2_3_5
<code>dinsu</code>	<code>DINSU rs, pos, size</code>	<code>dinsu.operation</code>	AT >= MIPS64_32 && AR >= R2_3_5
<code>div.r6</code>	<code>DIV rd, rs, rt</code>	Divide signed low 32-bit GPR operands and write a sign-extended quotient.	AR >= R6

Format	Assembly	Operation	Availability
div	DIV rs, rt	Divide signed low 32-bit GPR operands and write a sign-extended quotient.	AR < R6
div.d	DIV.D fd, fs, ft	div.d.operation	true
div.s	DIV.S fd, fs, ft	div.s.operation	true
divu.r6	DIVU rd, rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended quotient.	AR >= R6
divu	DIVU rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended quotient.	AR < R6
dlsa	DLSA rd, rs, rt, shift	dlsa.operation	AT >= MIPS64_32 && AR >= R6
dmfc0	DMFC0 c0s, sel	dmfc0.operation	AT >= MIPS64_32
dmfc1	DMFC1 rt, fs	dmfc1.operation	AT >= MIPS64_32
dmod.r6	DMOD rs, rt	dmod.operation	AT >= MIPS64_32 && AR >= R6
dmodu.r6	DMODU rs, rt	dmodu.operation	AT >= MIPS64_32 && AR >= R6
dmtc0	DMTC0 c0s, sel	dmtc0.operation	AT >= MIPS64_32
dmtc1	DMTC1 rt, fs	dmtc1.operation	AT >= MIPS64_32
dmuh.r6	DMUH rs, rt	dmuh.operation	AT >= MIPS64_32 && AR >= R6
dmuhu.r6	DMUHU rs, rt	dmuhu.operation	AT >= MIPS64_32 && AR >= R6
dmul.r6	DMUL rs, rt	dmul.operation	AT >= MIPS64_32 && AR >= R6
dmult	DMULT rt	dmult.operation	AT >= MIPS64_32 && AR < R6
dmultu	DMULTU rt	dmultu.operation	AT >= MIPS64_32 && AR < R6
dmulu.r6	DMULU rs, rt	dmulu.operation	AT >= MIPS64_32 && AR >= R6
dpa.w.ph	DPA.W.PH ac, rs, rt	dpa.w.ph.operation	AR >= R2_3_5 && DSP2
dpaq_s.w.ph	DPAQ_S.W.PH ac, rs, rt	dpaq_s.w.ph.operation	AR >= R2_3_5 && DSP

Format	Assembly	Operation	Availability
dpaq_sa.l.w	DPAQ_SA.L.W ac, rs, rt	dpaq_s.l.w.operation	AR >= R2_3_5 && DSP
dpaqx_s.w.ph	DPAQX_S.W.PH ac, rs, rt	dpaqx_s.w.ph.operation	AR >= R2_3_5 && DSP
dpaqx_sa.w.ph	DPAQX_SA.W.PH ac, rs, rt	dpaqx_sa.w.ph.operation	AR >= R2_3_5 && DSP
dpau.h.qbl	DPAU.H.QBL ac, rs, rt	dpau.h.qbl.operation	AR >= R2_3_5 && DSP
dpau.h.qbr	DPAU.H.QBR ac, rs, rt	dpau.h.qbr.operation	AR >= R2_3_5 && DSP
dpax.w.ph	DPAX.W.PH ac, rs, rt	dpax.w.ph.operation	AR >= R2_3_5 && DSP
dps.w.ph	DPS.W.PH ac, rs, rt	dps.w.ph.operation	AR >= R2_3_5 && DSP2
dpsq_s.w.ph	DPSQ_S.W.PH ac, rs, rt	dpsq_s.w.ph.operation	AR >= R2_3_5 && DSP
dpsq_sa.l.w	DPSQ_SA.L.W ac, rs, rt	dpsq_s.l.w.operation	AR >= R2_3_5 && DSP
dpsqx_s.w.ph	DPSQX_S.W.PH ac, rs, rt	dpsqx_s.w.ph.operation	AR >= R2_3_5 && DSP
dpsqx_sa.w.ph	DPSQX_SA.W.PH ac, rs, rt	dpsqx_sa.w.ph.operation	AR >= R2_3_5 && DSP
dpsu.h.qbl	DPSU.H.QBL ac, rs, rt	dpsu.h.qbl.operation	AR >= R2_3_5 && DSP
dpsu.h.qbr	DPSU.H.QBR ac, rs, rt	dpsu.h.qbr.operation	AR >= R2_3_5 && DSP
dpsx.w.ph	DPSX.W.PH ac, rs, rt	dpsx.w.ph.operation	AR >= R2_3_5 && DSP
drotr	DROTR rs, shift	drotr.operation	AT >= MIPS64_32 && AR >= R2_3_5
drotr32	DROTR32 rs, shift	drotr32.operation	AT >= MIPS64_32 && AR >= R2_3_5
drotrv	DROTRV rs, rt	drotrv.operation	AT >= MIPS64_32 && AR >= R2_3_5
dsbh	DSBH rs	dsbh.operation	AT >= MIPS64_32 && AR >= R2_3_5
dshd	DSHD rs	dshd.operation	AT >= MIPS64_32 && AR >= R2_3_5
dsll	DSLL rs, shift	dsll.operation	AT >= MIPS64_32

Format	Assembly	Operation	Availability
dsll32	DSLL32 <i>rs</i> , <i>shift</i>	dsll32.operation	AT >= MIPS64_32
dsllv	DSLLV <i>rs</i> , <i>rt</i>	dsllv.operation	AT >= MIPS64_32
dsra	DSRA <i>rs</i> , <i>shift</i>	dsra.operation	AT >= MIPS64_32
dsra32	DSRA32 <i>rs</i> , <i>shift</i>	dsra32.operation	AT >= MIPS64_32
dsrav	DSRAV <i>rs</i> , <i>rt</i>	dsrav.operation	AT >= MIPS64_32
dsrl	DSRL <i>rs</i> , <i>shift</i>	dsrl.operation	AT >= MIPS64_32
dsrl32	DSRL32 <i>rs</i> , <i>shift</i>	dsrl32.operation	AT >= MIPS64_32
dsrlv	DSRLV <i>rs</i> , <i>rt</i>	dsrlv.operation	AT >= MIPS64_32
dsub	DSUB <i>rs</i> , <i>rt</i>	dsub.operation	AT >= MIPS64_32
dsubu	DSUBU <i>rs</i> , <i>rt</i>	dsubu.operation	AT >= MIPS64_32
dvp	DVP <i>rt</i>	dvp.operation	AR >= R6
ei	EI <i>rt</i>	ei.operation	AR >= R2_3_5
eret	ERET	Return from an exception and clear the LL reservation.	true
eretnc	ERETNC	Return from an exception without clearing the LL reservation.	LLB
evp	EVP <i>rt</i>	evp.operation	AR >= R6
ext	EXT <i>rt</i> , <i>rs</i> , <i>pos</i> , <i>size</i>	Extract a 32-bit field and sign-extend the result to the GPR width.	AR >= R2_3_5
extp	EXTP <i>rt</i> , <i>ac</i> , <i>size</i>	extp.operation	AR >= R2_3_5 && DSP
extpdp	EXTPDP <i>rt</i> , <i>ac</i> , <i>size</i>	extpdp.operation	AR >= R2_3_5 && DSP
extpdpv	EXTPDPV <i>rt</i> , <i>ac</i> , <i>rs</i>	extpdpv.operation	AR >= R2_3_5 && DSP
extpv	EXTPV <i>rt</i> , <i>ac</i> , <i>rs</i>	extpv.operation	AR >= R2_3_5 && DSP
extr.w	EXTR.W <i>rt</i> , <i>ac</i> , <i>shift</i>	extr.w.operation	AR >= R2_3_5 && DSP
extr_r.w	EXTR_R.W <i>rt</i> , <i>ac</i> , <i>shift</i>	extr_r.w.operation	AR >= R2_3_5 && DSP
extr_rs.w	EXTR_RS.W <i>rt</i> , <i>ac</i> , <i>shift</i>	extr_rs.w.operation	AR >= R2_3_5 && DSP
extr_s.h	EXTR_S.H <i>rt</i> , <i>ac</i> , <i>shift</i>	extr_s.h.operation	AR >= R2_3_5 && DSP

Format	Assembly	Operation	Availability
extrv.w	EXTRV.W rt, ac, rs	extrv.w.operation	AR >= R2_3_5 && DSP
extrv_r.w	EXTRV_R.W rt, ac, rs	extrv_r.w.operation	AR >= R2_3_5 && DSP
extrv_rs.w	EXTRV_RS.W rt, ac, rs	extrv_rs.w.operation	AR >= R2_3_5 && DSP
extrv_s.h	EXTRV_S.H rt, ac, rs	extrv_s.h.operation	AR >= R2_3_5 && DSP
floor.l.d	FLOOR.L.D fd, fs	floor.l.d.operation	true
floor.l.s	FLOOR.L.S fd, fs	floor.l.s.operation	true
floor.w.d	FLOOR.W.D fd, fs	floor.w.d.operation	true
floor.w.s	FLOOR.W.S fd, fs	floor.w.s.operation	true
ins	INS rt, rs, pos, size	Merge a right-justified source field into a 32-bit destination word.	AR >= R2_3_5
insv	INSV rt, rs	insv.operation	AR >= R2_3_5 && DSP
j	J index	j.operation	true
jal	JAL index	jal.operation	true
jalr	JALR rd, rs	jalr.operation	true
jalr.hb	JALR rd, rs	jalr.operation	true
jalx	JALX index	Call a region-relative target and exchange the ISA mode after its slot.	AR < R6 && (MMAR >= R3_5 && MMAR < nanoMIPS_R6 MIPS16E)
jialc	JIALC offset	jialc.operation	AR >= R6
jic	JIC offset	jic.operation	AR >= R6
jr	JR rs	jr.operation	AR < R6
jr.hb	JR rs	jr.operation	AR < R6
lb	LB rt, offset(rs)	Load one byte and sign-extend it to the implemented GPR width.	true
lbu	LBU rt, offset(rs)	Load one byte and zero-extend it to the implemented GPR width.	true
lbux.dsp	LBUX rs(rt)	lbux.operation	AR >= R2_3_5 && DSP
ld	LD offset(rs)	ld.operation	AT >= MIPS64_32
ldc1	LDC1 ft, offset(base)	ldc1.operation	true

Format	Assembly	Operation	Availability
ldl	LDL offset(rs)	ldl.operation	AT >= MIPS64_32 && AR < R6
ldpc	LDPC rt, offset	ldpc.operation	AR >= R6 && AT >= MIPS64_32
ldr	LDR offset(rs)	ldr.operation	AT >= MIPS64_32 && AR < R6
ldxc1	LDXC1 fd, index(base)	ldxc1.operation	AR >= R2_3_5 && AR < R6
lh	LH rt, offset(rs)	Load a signed halfword from memory and extend it to GPR width.	true
lhu	LHU rt, offset(rs)	Load an unsigned halfword from memory and extend it to GPR width.	true
lhx.dsp	LHX rs(rt)	lhx.operation	AR >= R2_3_5 && DSP
ll	LL rt, offset(rs)	Load a sign-extended word and establish a linked reservation.	AR < R6
ll.r6	LL rt, offset(rs)	Load a sign-extended word and establish a linked reservation.	AR >= R6
lld	LLD offset(rs)	lld.operation	AT >= MIPS64_32 && AR < R6
lld.r6	LLD offset(rs)	lld.operation	AT >= MIPS64_32 && AR >= R6
lldp	LLDP ru, (rs)	lldp.operation	AT >= MIPS64_32 && AR >= R6
llwp	LLWP rt, ru, (rs)	Load a linked pair of sign-extended words.	AR >= R6
lsa	LSA rd, rs, rt, shift	Add a scaled 32-bit GPR value to another and sign-extend the result.	AR >= R6
lui	LUI rt, imm	Load the sign-extended 32-bit upper-immediate value into a GPR.	AR < R6
luxc1	LUXC1 fd, index(base)	luxc1.operation	AR >= R2_3_5 && AR < R6
lw	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lwc1	LWC1 ft, offset(base)	lwc1.operation	true

Format	Assembly	Operation	Availability
lwl	LWL offset(rs)	lwl.operation	AR < R6
lwpc	LWPC offset	Load a signed word using the active architecture's PC-relative form.	AR >= R6
lwr	LWR rt, offset(rs)	Merge the right part of an aligned word into the low 32-bit GPR value.	AR < R6
lwu	LWU offset(rs)	lwu.operation	AT >= MIPS64_32
lwupc	LWUPC offset	Load an unsigned word using the MIPS PC-relative form.	AR >= R6 && AT >= MIPS64_32
lwx.dsp	LWX rs(rt)	lwx.operation	AR >= R2_3_5 && DSP
lwxc1	LWXC1 fd, index(base)	lwxc1.operation	AR >= R2_3_5 && AR < R6
madd.dsp	MADD ac, rs, rt	madd.operation	AR >= R2_3_5 && DSP
madd	MADD rs, rt	madd.operation	AR < R6
madd.d	MADD.D fd, fr, fs, ft	madd.d.legacy.operation	AR < R6
madd.ps	MADD.PS fd, fr, fs, ft	madd.ps.legacy.operation	AR < R6
madd.s	MADD.S fd, fr, fs, ft	madd.s.legacy.operation	AR < R6
maddf.d	MADDF.D fd, fs, ft	maddf.d.operation	AR >= R6
maddf.s	MADDF.S fd, fs, ft	maddf.s.operation	AR >= R6
maddu.dsp	MADDU ac, rs, rt	maddu.operation	AR >= R2_3_5 && DSP
maddu	MADDU rs, rt	maddu.operation	AR < R6
maq_s.w.phl	MAQ_S.W.PHL ac, rs, rt	maq_s.w.phl.operation	AR >= R2_3_5 && DSP
maq_s.w.phr	MAQ_S.W.PHR ac, rs, rt	maq_s.w.phr.operation	AR >= R2_3_5 && DSP
maq_sa.w.phl	MAQ_SA.W.PHL ac, rs, rt	maq_sa.w.phl.operation	AR >= R2_3_5 && DSP
maq_sa.w.phr	MAQ_SA.W.PHR ac, rs, rt	maq_sa.w.phr.operation	AR >= R2_3_5 && DSP
max.d	MAX.D fd, fs, ft	max.d.operation	AR >= R6
max.s	MAX.S fd, fs, ft	max.s.operation	AR >= R6
maxa.d	MAXA.D fd, fs, ft	maxa.d.operation	AR >= R6
maxa.s	MAXA.S fd, fs, ft	maxa.s.operation	AR >= R6

Format	Assembly	Operation	Availability
mfc0	MFC0 rt, c0s, sel	Read a CP0 register and sign-extend its low 32-bit result to GPR width.	true
mfc1	MFC1 rt, fs	Copy the low word of an FPR to a GPR after checking CP1 access.	true
mfhc0	MFHC0 rt, c0s, sel	mfhc0.operation	AT >= MIPS64_32 MVH
mfhc1	MFHC1 rt, fs	mfhc1.operation	AR >= R2_3_5 && (AT >= MIPS64_32 FPU64)
mfhi	MFHI rd	mfhi.operation	AR < R6
mfhi.dsp	MFHI rd, ac	mfhi.operation	AR >= R2_3_5 && DSP
mflo	MFLO rd	mflo.operation	AR < R6
mflo.dsp	MFLO rd, ac	mflo.operation	AR >= R2_3_5 && DSP
min.d	MIN.D fd, fs, ft	min.d.operation	AR >= R6
min.s	MIN.S fd, fs, ft	min.s.operation	AR >= R6
mina.d	MINA.D fd, fs, ft	mina.d.operation	AR >= R6
mina.s	MINA.S fd, fs, ft	mina.s.operation	AR >= R6
mod	MOD rd, rs, rt	Divide signed low 32-bit GPR operands and write a sign-extended remainder.	AR >= R6
modsub	MODSUB rd, rs, rt	Subtract a decrement, wrapping to the supplied last index at zero.	AR >= R2_3_5 && DSP
modu	MODU rd, rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended remainder.	AR >= R6
mov.d	MOV.D fd, fs	mov.d.operation	true
mov.ps	MOV.PS fd, fs	mov.ps.operation	AR < R6
mov.s	MOV.S fd, fs	mov.s.operation	true
movf	MOVF rd, rs, cc	movf.operation	AR < R6
movf.d	MOVF.D fd, fs, cc	movf.d.operation	AR < R6
movf.ps	MOVF.PS fd, fs, cc	movf.ps.operation	AR < R6
movf.s	MOVF.S fd, fs, cc	movf.s.operation	AR < R6
movn	MOVN rd, rs, rt	Copy rs to rd when the full-width value in rt is nonzero.	AR < R6

Format	Assembly	Operation	Availability
movn.d	MOVN.D fd, fs, rt	movn.d.legacy.operation	AR < R6
movn.ps	MOVN.PS fd, fs, rt	movn.ps.legacy.operation	AR < R6
movn.s	MOVN.S fd, fs, rt	movn.s.legacy.operation	AR < R6
movt	MOVT rd, rs, cc	movt.operation	AR < R6
movt.d	MOVT.D fd, fs, cc	movt.d.operation	AR < R6
movt.ps	MOVT.PS fd, fs, cc	movt.ps.operation	AR < R6
movt.s	MOVT.S fd, fs, cc	movt.s.operation	AR < R6
movz	MOVZ rd, rs, rt	Copy rs to rd when the full-width value in rt is zero.	AR < R6
movz.d	MOVZ.D fd, fs, rt	movz.d.legacy.operation	AR < R6
movz.ps	MOVZ.PS fd, fs, rt	movz.ps.legacy.operation	AR < R6
movz.s	MOVZ.S fd, fs, rt	movz.s.legacy.operation	AR < R6
msub.dsp	MSUB ac, rs, rt	msub.operation	AR >= R2_3_5 && DSP
msub	MSUB rs, rt	msub.operation	AR < R6
msub.d	MSUB.D fd, fr, fs, ft	msub.d.legacy.operation	AR < R6
msub.ps	MSUB.PS fd, fr, fs, ft	msub.ps.legacy.operation	AR < R6
msub.s	MSUB.S fd, fr, fs, ft	msub.s.legacy.operation	AR < R6
msubf.d	MSUBF.D fd, fs, ft	msubf.d.operation	AR >= R6
msubf.s	MSUBF.S fd, fs, ft	msubf.s.operation	AR >= R6
msubu.dsp	MSUBU ac, rs, rt	msubu.operation	AR >= R2_3_5 && DSP
msubu	MSUBU rs, rt	msubu.operation	AR < R6
mtc0	MTC0 rt, c0s, sel	Write a GPR to an encoded CP0 register. On a 64-bit MIPS core the MTC0 data path is the full GPR width; the MFC0/DMFC0 pair then exposes the low-word and full-width read views. A 32-bit core retains the ordinary 32-bit path.	true
mtc1	MTC1 rt, fs	mtc1.operation	true
mt hc0	MTHC0 rt, c0s, sel	mt hc0.operation	AT >= MIPS64_32 MVH
mt hc1	MTHC1 rt, fs	mt hc1.operation	AR >= R2_3_5 && (AT >= MIPS64_32 FPU64)
mt hi	MTHI rs	mt hi.operation	AR < R6

Format	Assembly	Operation	Availability
mthi.dsp	MTHI rs, ac	mthi.operation	AR >= R2_3_5 && DSP
mthlip	MTHLIP rs, ac	mthlip.operation	DSP && AR >= R2_3_5
mtlo	MTLO rs	mtlo.operation	AR < R6
mtlo.dsp	MTLO rs, ac	mtlo.operation	AR >= R2_3_5 && DSP
muh	MUH rd, rs, rt	Multiply signed words and write the sign-extended high word.	AR >= R6
muhu	MUHU rd, rs, rt	Multiply unsigned words and write the high word.	AR >= R6
mul	MUL rd, rs, rt	Multiply signed 32-bit GPR operands and sign-extend the low word.	AR < R6
mul.r6	MUL rd, rs, rt	Multiply signed 32-bit GPR operands and sign-extend the low word.	AR >= R6
mul.d	MUL.D fd, fs, ft	mul.d.operation	true
mul.ph	MUL.PH rd, rs, rt	mul.ph.operation	AR >= R2_3_5 && DSP
mul.ps	MUL.PS fd, fs, ft	mul.ps.operation	AR < R6
mul.s	MUL.S fd, fs, ft	mul.s.operation	true
mul_s.ph	MUL_S.PH rd, rs, rt	mul_s.ph.operation	AR >= R2_3_5 && DSP
muleq_s.w.ph l	MULEQ_S.W.PHL rd, rs, rt	muleq_s.w.phl.operation	AR >= R2_3_5 && DSP
muleq_s.w.ph r	MULEQ_S.W.PHR rd, rs, rt	muleq_s.w.phr.operation	AR >= R2_3_5 && DSP
muleu_s.ph.q bl	MULEU_S.PH.QBL rd, rs, rt	muleu_s.ph.qbl.operation	AR >= R2_3_5 && DSP
muleu_s.ph.q br	MULEU_S.PH.QBR rd, rs, rt	muleu_s.ph.qbr.operation	AR >= R2_3_5 && DSP
mulq_rs.ph	MULQ_RS.PH rd, rs, rt	mulq_rs.ph.operation	AR >= R2_3_5 && DSP
mulq_rs.w	MULQ_RS.W rd, rs, rt	mulq_rs.w.operation	AR >= R2_3_5 && DSP2
mulq_s.ph	MULQ_S.PH rd, rs, rt	mulq_s.ph.operation	AR >= R2_3_5 && DSP2

Format	Assembly	Operation	Availability
mulq_s.w	MULQ_S.W rd, rs, rt	mulq_s.w.operation	AR >= R2_3_5 && DSP2
mulsa.w.ph	MULSA.W.PH ac, rs, rt	mulsa.w.ph.operation	AR >= R2_3_5 && DSP2
mulsaq_s.w.ph	MULSAQ_S.W.PH ac, rs, rt	mulsaq_s.w.ph.operation	AR >= R2_3_5 && DSP
mult.dsp	MULT ac, rs, rt	mult.operation	AR >= R2_3_5 && DSP
mult	MULT rs, rt	mult.operation	AR < R6
multu.dsp	MULTU ac, rs, rt	multu.operation	AR >= R2_3_5 && DSP
multu	MULTU rs, rt	multu.operation	AR < R6
mulu	MULU rd, rs, rt	Multiply unsigned words and write the sign-extended low word.	AR >= R6
nal	NAL	nal.operation	AR >= R6
neg.d	NEG.D fd, fs	neg.d.operation	true
neg.ps	NEG.PS fd, fs	neg.ps.operation	AR < R6
neg.s	NEG.S fd, fs	neg.s.operation	true
nmadd.d	NMADD.D fd, fr, fs, ft	nmadd.d.legacy.operation	AR < R6
nmadd.ps	NMADD.PS fd, fr, fs, ft	nmadd.ps.legacy.operation	AR < R6
nmadd.s	NMADD.S fd, fr, fs, ft	nmadd.s.legacy.operation	AR < R6
nmsub.d	NMSUB.D fd, fr, fs, ft	nmsub.d.legacy.operation	AR < R6
nmsub.ps	NMSUB.PS fd, fr, fs, ft	nmsub.ps.legacy.operation	AR < R6
nmsub.s	NMSUB.S fd, fr, fs, ft	nmsub.s.legacy.operation	AR < R6
nop	NOP	Perform no architectural operation.	true
nor	NOR rd, rs, rt	Compute a bitwise NOR at the implemented GPR width.	true
or	OR rd, rs, rt	Compute a full-GPR logical OR and write the result to a GPR.	true
ori	ORI rt, rs, imm	Compute a full-GPR logical OR with a zero-extended immediate.	true
packrl.ph	PACKRL.PH rd, rs, rt	packrl.ph.operation	DSP && AR >= R2_3_5
pause	PAUSE	Pause instruction issue while CP0 LLAddr.LLB remains set.	true

Format	Assembly	Operation	Availability
pick.ph	PICK.PH rd, rs, rt	Select halfword lanes from rs or rt according to Ccond.	AR >= R2_3_5 && DSP
pick.qb	PICK.QB rd, rs, rt	Select byte lanes from rs or rt according to Ccond.	AR >= R2_3_5 && DSP
pll.ps	PLL.PS fd, fs, ft	pll.ps.operation	AR < R6
plu.ps	PLU.PS fd, fs, ft	plu.ps.operation	AR < R6
preceq.w.phl	PRECEQ.W.PHL rd, rt	preceq.w.phl.operation	DSP && AR >= R2_3_5
preceq.w.phr	PRECEQ.W.PHR rd, rt	preceq.w.phr.operation	DSP && AR >= R2_3_5
precequ.ph.qbl	PRECEQU.PH.QBL rd, rt	precequ.ph.qbl.operation	DSP && AR >= R2_3_5
precequ.ph.qbla	PRECEQU.PH.QBLA rd, rt	precequ.ph.qbla.operation	DSP && AR >= R2_3_5
precequ.ph.qbr	PRECEQU.PH.QBR rd, rt	precequ.ph.qbr.operation	DSP && AR >= R2_3_5
precequ.ph.qbra	PRECEQU.PH.QBRA rd, rt	precequ.ph.qbra.operation	DSP && AR >= R2_3_5
preceu.ph.qbl	PRECEU.PH.QBL rd, rt	preceu.ph.qbl.operation	DSP && AR >= R2_3_5
preceu.ph.qbla	PRECEU.PH.QBLA rd, rt	preceu.ph.qbla.operation	DSP && AR >= R2_3_5
preceu.ph.qbr	PRECEU.PH.QBR rd, rt	preceu.ph.qbr.operation	DSP && AR >= R2_3_5
preceu.ph.qbra	PRECEU.PH.QBRA rd, rt	preceu.ph.qbra.operation	DSP && AR >= R2_3_5
precr.qb.ph	PRECR.QB.PH rd, rs, rt	precr.qb.ph.operation	DSP2 && AR >= R2_3_5
precr_sra.ph.w	PRECR_SRA.PH.W rt, rs, sa	precr_sra.ph.w.operation	DSP2 && AR >= R2_3_5
precr_sra_r.ph.w	PRECR_SRA_R.PH.W rt, rs, sa	precr_sra_r.ph.w.operation	DSP2 && AR >= R2_3_5
precrq.ph.w	PRECRQ.PH.W rd, rs, rt	precrq.ph.w.operation	DSP && AR >= R2_3_5
precrq.qb.ph	PRECRQ.QB.PH rd, rs, rt	precrq.qb.ph.operation	DSP && AR >= R2_3_5
precrq_rs.ph.w	PRECRQ_RS.PH.W rd, rs, rt	precrq_rs.ph.w.operation	DSP && AR >= R2_3_5

Format	Assembly	Operation	Availability
precrqu_s.qb.ph	PRECRQU_S.QB.PH rd, rs, rt	precrqu_s.qb.ph.operation	DSP && AR >= R2_3_5
pref	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	AR < R6
pref.r6	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	AR >= R6
prefx	PREFX hint, index(base)	Issue an implementation-dependent prefetch at base plus index.	AR >= R2_3_5 && AR < R6
prepend	PREPEND rt, rs, sa	prepend.operation	AR >= R2_3_5 && DSP2
pul.ps	PUL.PS fd, fs, ft	pul.ps.operation	AR < R6
puu.ps	PUU.PS fd, fs, ft	puu.ps.operation	AR < R6
raddu.w.qb	RADDU.W.QB rd, rs	Reduce four unsigned bytes to a zero-extended ten-bit word.	AR >= R2_3_5 && DSP
rddsp	RDDSP rd, mask	rddsp.operation	DSP && AR >= R2_3_5
rdhwr	RDHWR rt, hs, sel	rdhwr.operation	AR >= R2_3_5
rdpgpr	RDPGPR dst, src	Read a GPR from the previous shadow set into the current set.	AR >= R2_3_5
recip.d	RECIP.D fd, fs	recip.d.operation	true
recip.s	RECIP.S fd, fs	recip.s.operation	true
repl.ph	REPL.PH rd, immediate	repl.ph.operation	DSP && AR >= R2_3_5
repl.qb	REPL.QB rd, immediate	repl.qb.operation	DSP && AR >= R2_3_5
replv.ph	REPLV.PH rd, rt	replv.ph.operation	DSP && AR >= R2_3_5
replv.qb	REPLV.QB rd, rt	replv.qb.operation	DSP && AR >= R2_3_5
rint.d	RINT.D fd, fs	rint.d.operation	AR >= R6
rint.s	RINT.S fd, fs	rint.s.operation	AR >= R6
rotr	ROTR rt, rs, shift	Rotate the low 32-bit source word right and sign-extend the result.	AR >= R2_3_5
rotrv	ROTRV rd, rs, rt	Rotate a 32-bit GPR word right by a GPR-supplied amount.	AR >= R2_3_5

Format	Assembly	Operation	Availability
round.l.d	ROUND.L.D fd, fs	round.l.d.operation	true
round.l.s	ROUND.L.S fd, fs	round.l.s.operation	true
round.w.d	ROUND.W.D fd, fs	round.w.d.operation	true
round.w.s	ROUND.W.S fd, fs	round.w.s.operation	true
rsqrt.d	RSQRT.D fd, fs	rsqrt.d.operation	true
rsqrt.s	RSQRT.S fd, fs	rsqrt.s.operation	true
sb	SB rt, offset(rs)	Store the low byte of rt at the effective address rs plus offset.	true
sc	SC rt, offset(rs)	Conditionally store a word and return success in rt.	AR < R6
sc.r6	SC rt, offset(rs)	Conditionally store a word and return success in rt.	AR >= R6
scd	SCD offset(rs)	scd.operation	AT >= MIPS64_32 && AR < R6
scd.r6	SCD offset(rs)	scd.operation	AT >= MIPS64_32 && AR >= R6
scdp	SCDP ru, (rs)	scdp.operation	AT >= MIPS64_32 && AR >= R6
scwp	SCWP rt, ru, (rs)	Conditionally store a word pair and return success in rt.	AR >= R6
sd	SD offset(rs)	sd.operation	AT >= MIPS64_32
sdbbp	SDBBP code	Raise a debug breakpoint and set CP0.Debug.DBp when debug is present.	AR < R6
sdbbp.r6	SDBBP code	Raise a debug breakpoint and set CP0.Debug.DBp when debug is present.	AR >= R6
sdc1	SDC1 ft, offset(base)	sdc1.operation	true
sdl	SDL offset(rs)	sdl.operation	AT >= MIPS64_32 && AR < R6
sdr	SDR offset(rs)	sdr.operation	AT >= MIPS64_32 && AR < R6
sdxc1	SDXC1 fd, index(base)	sdxc1.operation	AR >= R2_3_5 && AR < R6
seb	SEB rt, rs	Sign-extend the low byte of rs to the implemented GPR width.	AR >= R2_3_5
seh	SEH rt, rs	Sign-extend the low halfword of rs to the implemented GPR width.	AR >= R2_3_5

Format	Assembly	Operation	Availability
sel.d	SEL.D fd, fs, ft	sel.d.operation	AR >= R6
sel.s	SEL.S fd, fs, ft	sel.s.operation	AR >= R6
seleqz	SELEQZ rs, rt	seleqz.operation	AR >= R6
seleqz.d	SELEQZ.D fd, fs, ft	seleqz.d.operation	AR >= R6
seleqz.s	SELEQZ.S fd, fs, ft	seleqz.s.operation	AR >= R6
selnez	SELNEZ rs, rt	selnez.operation	AR >= R6
selnez.d	SELNEZ.D fd, fs, ft	selnez.d.operation	AR >= R6
selnez.s	SELNEZ.S fd, fs, ft	selnez.s.operation	AR >= R6
sh	SH rt, offset(rs)	Store the low 16 bits of rt at rs plus offset.	true
shilo	SHILO ac, shift	shilo.operation	AR >= R2_3_5 && DSP
shilov	SHILOV ac, rs	shilov.operation	AR >= R2_3_5 && DSP
shll.ph	SHLL.PH rd, rt, sa	dsp.shll.ph	AR >= R2_3_5 && DSP
shll.qb	SHLL.QB rd, rt, sa	dsp.shll.qb	AR >= R2_3_5 && DSP
shll_s.ph	SHLL_S.PH rd, rt, sa	dsp.shll_s.ph	AR >= R2_3_5 && DSP
shll_s.w	SHLL_S.W rd, rt, sa	dsp.shll_s.w	AR >= R2_3_5 && DSP
shllv.ph	SHLLV.PH rd, rt, rs	dsp.shllv.ph	AR >= R2_3_5 && DSP
shllv.qb	SHLLV.QB rd, rt, rs	dsp.shllv.qb	AR >= R2_3_5 && DSP
shllv_s.ph	SHLLV_S.PH rd, rt, rs	dsp.shllv_s.ph	AR >= R2_3_5 && DSP
shllv_s.w	SHLLV_S.W rd, rt, rs	dsp.shllv_s.w	AR >= R2_3_5 && DSP
shra.ph	SHRA.PH rd, rt, sa	dsp.shra.ph	AR >= R2_3_5 && DSP2
shra.qb	SHRA.QB rd, rt, sa	dsp.shra.qb	AR >= R2_3_5 && DSP2
shra_r.ph	SHRA_R.PH rd, rt, sa	dsp.shra_r.ph	AR >= R2_3_5 && DSP2

Format	Assembly	Operation	Availability
shra_r.qb	SHRA_R.QB rd, rt, sa	dsp.shra_r.qb	AR >= R2_3_5 && DSP2
shra_r.w	SHRA_R.W rd, rt, sa	dsp.shra_r.w	AR >= R2_3_5 && DSP
shrav.ph	SHRAV.PH rd, rt, rs	dsp.shrav.ph	AR >= R2_3_5 && DSP
shrav.qb	SHRAV.QB rd, rt, rs	dsp.shrav.qb	AR >= R2_3_5 && DSP2
shrav_r.ph	SHRAV_R.PH rd, rt, rs	dsp.shrav_r.ph	AR >= R2_3_5 && DSP
shrav_r.qb	SHRAV_R.QB rd, rt, rs	dsp.shrav_r.qb	AR >= R2_3_5 && DSP2
shrav_r.w	SHRAV_R.W rd, rt, rs	dsp.shrav_r.w	AR >= R2_3_5 && DSP
shrl.ph	SHRL.PH rd, rt, sa	dsp.shrl.ph	AR >= R2_3_5 && DSP2
shrl.qb	SHRL.QB rd, rt, sa	dsp.shrl.qb	AR >= R2_3_5 && DSP
shrlv.ph	SHRLV.PH rd, rt, rs	dsp.shrlv.ph	AR >= R2_3_5 && DSP2
shrlv.qb	SHRLV.QB rd, rt, rs	dsp.shrlv.qb	AR >= R2_3_5 && DSP
sigrie	SIGRIE code	sigrie.operation	AR >= R6
sll	SLL rt, rs, shift	Shift a 32-bit GPR word left and sign-extend the result.	true
sllv	SLLV rd, rs, rt	Shift a 32-bit GPR word left by the low five bits of a GPR.	true
slt	SLT rd, rs, rt	Set rd when the signed, implemented-width rs is below rt.	true
slti	SLTI rt, rs, imm	Set the destination when a signed full-width GPR is below imm.	true
sltiu	SLTIU rt, rs, imm	Set the destination to one when an unsigned GPR is below the immediate.	true
sltu	SLTU rd, rs, rt	Set rd when the unsigned, implemented-width rs is below rt.	true
sqrt.d	SQRT.D fd, fs	sqrt.d.operation	true
sqrt.s	SQRT.S fd, fs	sqrt.s.operation	true

Format	Assembly	Operation	Availability
sra	SRA rt, rs, shift	Arithmetic-shift a signed 32-bit GPR word right and extend it.	true
srav	SRAV rd, rs, rt	Arithmetic-shift a 32-bit GPR word by a register-supplied amount.	true
srl	SRL rt, rs, shift	Shift a 32-bit GPR word right logically and sign-extend the result.	true
srlv	SRLV rd, rs, rt	Shift a 32-bit GPR word right logically by a register-supplied amount.	true
sub	SUB rd, rs, rt	Subtract signed 32-bit GPR operands and trap on integer overflow.	true
sub.d	SUB.D fd, fs, ft	sub.d.operation	true
sub.ps	SUB.PS fd, fs, ft	sub.ps.operation	AR < R6
sub.s	SUB.S fd, fs, ft	sub.s.operation	true
subq.ph	SUBQ.PH rd, rs, rt	Subtract signed halfword lanes without saturation.	AR >= R2_3_5 && DSP
subq_s.ph	SUBQ_S.PH rd, rs, rt	Subtract signed halfword lanes with signed saturation.	AR >= R2_3_5 && DSP
subq_s.w	SUBQ_S.W rd, rs, rt	Subtract signed word lanes with signed saturation.	AR >= R2_3_5 && DSP
subqh.ph	SUBQH.PH rd, rs, rt	Subtract signed halfword lanes and halve the results.	AR >= R2_3_5 && DSP2
subqh.w	SUBQH.W rd, rs, rt	Subtract signed word lanes and halve the result.	AR >= R2_3_5 && DSP2
subqh_r.ph	SUBQH_R.PH rd, rs, rt	Rounded signed half-subtract of two halfword lanes.	AR >= R2_3_5 && DSP2
subqh_r.w	SUBQH_R.W rd, rs, rt	Rounded signed half-subtract of two word lanes.	AR >= R2_3_5 && DSP2
subu	SUBU rd, rs, rt	Subtract 32-bit GPR rt from rs and sign-extend to GPR width.	true
subu.ph	SUBU.PH rd, rs, rt	Subtract unsigned halfword lanes modulo 16 bits.	AR >= R2_3_5 && DSP2
subu.qb	SUBU.QB rd, rs, rt	Subtract unsigned byte lanes modulo 8 bits.	AR >= R2_3_5 && DSP
subu_s.ph	SUBU_S.PH rd, rs, rt	Subtract unsigned halfword lanes with saturation.	AR >= R2_3_5 && DSP2

Format	Assembly	Operation	Availability
subu_s.qb	SUBU_S.QB rd, rs, rt	Subtract unsigned byte lanes with saturation.	AR >= R2_3_5 && DSP
subuh.qb	SUBUH.QB rd, rs, rt	Average unsigned byte differences by an unrounded right shift.	AR >= R2_3_5 && DSP
subuh_r.qb	SUBUH_R.QB rd, rs, rt	Average unsigned byte differences with rounding.	AR >= R2_3_5 && DSP
suxc1	SUXC1 fd, index(base)	suxc1.operation	AR >= R2_3_5 && AR < R6
sw	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	true
swc1	SWC1 ft, offset(base)	swc1.operation	true
swl	SWL offset(rs)	swl.operation	AR < R6
swr	SWR offset(rs)	swr.operation	AR < R6
swxc1	SWXC1 fd, index(base)	swxc1.operation	AR >= R2_3_5 && AR < R6
sync	SYNC stype	Impose the memory-ordering constraints selected by stype.	true
synci	SYNCI offset(rs)	synci.operation	AR >= R2_3_5
syscall	SYSCALL code	Raise a system-call exception.	true
teq	TEQ rs, rt, code	Raise Trap when the full-width source GPR values are equal.	true
teqi	TEQI imm	teqi.operation	AR < R6
tge	TGE rs, rt, code	Raise Trap when signed, full-width rs is greater than or equal to rt.	true
tgei	TGEI imm	tgei.operation	AR < R6
tgeiu	TGEIU imm	tgeiu.operation	AR < R6
tgeu	TGEU rs, rt, code	Raise Trap when unsigned, full-width rs is greater than or equal to rt.	true
tlbinv	TLBINV	tlbinv.operation	AR >= R2_3_5 && TLBINV
tlbinvf	TLBINVF	tlbinvf.operation	AR >= R2_3_5 && TLBINV
tlbp	TLBP	Probe the TLB for the translation selected by EntryHi.	true
tlbr	TLBR	tlbr.operation	true

Format	Assembly	Operation	Availability
tlbwi	TLBWI	Write the current CP0 translation state to the indexed TLB entry.	true
tlbwr	TLBWR	tlbwr.operation	true
tlt	TLT rs, rt, code	Raise Trap when signed, full-width rs is less than rt.	true
tlti	TLTI imm	tlti.operation	AR < R6
tltiu	TLTIU imm	tltiu.operation	AR < R6
tltu	TLTU rs, rt, code	Raise Trap when unsigned, full-width rs is less than rt.	true
tne	TNE rs, rt, code	Raise Trap when the full-width source GPR values differ.	true
tnei	TNEI imm	tnei.operation	AR < R6
trunc.l.d	TRUNC.L.D fd, fs	trunc.l.d.operation	true
trunc.l.s	TRUNC.L.S fd, fs	trunc.l.s.operation	true
trunc.w.d	TRUNC.W.D fd, fs	trunc.w.d.operation	true
trunc.w.s	TRUNC.W.S fd, fs	trunc.w.s.operation	true
wait	WAIT code	wait.operation	true
wrdsp	WRDSP rs, mask	wrdsp.operation	DSP && AR >= R2_3_5
wrpgpr	WRPGPR dst, src	Write the previous shadow-set GPR, or copy when no SRS is present.	AR >= R2_3_5
wsbh	WSBH rt, rs	Swap bytes within each halfword of a 32-bit GPR word.	AR >= R2_3_5
xor	XOR rd, rs, rt	Compute a full-GPR logical XOR and write the result to a GPR.	true
xori	XORI rt, rs, imm	Compute a full-GPR logical XOR with a zero-extended immediate.	true

1.3. MIPS16E

Format	Assembly	Operation	Availability
addiu.sp	ADDIU \$sp, imm	Add modulo 2**32 and sign-extend the 32-bit result to the GPR width.	MIPS16E
addiu.sp.extended	ADDIU \$sp, imm	Add modulo 2**32 and sign-extend the 32-bit result to the GPR width.	MIPS16E

Format	Assembly	Operation	Availability
addiu.rx.sp. extended	ADDIU rt, \$sp, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	MIPS16E
addiu.rx.sp	ADDIU rt, \$sp, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	MIPS16E
addiupc.extended	ADDIU rt, pc, imm	Add a displacement to the word-aligned PC in MIPS16 mode.	MIPS16E
addiupc	ADDIU rt, pc, imm	Add a displacement to the word-aligned PC in MIPS16 mode.	MIPS16E
addiu.rx.extended	ADDIU rt, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	MIPS16E
addiu.rx	ADDIU rt, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	MIPS16E
addiu.ry_rx. extended	ADDIU rt, rs, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	MIPS16E
addiu.ry_rx	ADDIU rt, rs, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	MIPS16E
nop.16	NOP	Perform no architectural operation.	MIPS16E

1.4. microMIPS

Format	Assembly	Operation	Availability
abs.d	ABS.D fd, fs	abs.d.operation	true
abs.ps	ABS.PS fd, fs	abs.ps.operation	MMAR < R6
abs.s	ABS.S fd, fs	abs.s.operation	true
absq_s.ph.32	ABSQ_S.PH rd, rs	Take the saturating absolute value of two signed halfword lanes.	DSP
absq_s.qb.32	ABSQ_S.QB rd, rs	Take the saturating absolute value of four signed byte lanes.	DSP2
absq_s.w.32	ABSQ_S.W rd, rs	Take the saturating absolute value of a signed word lane.	DSP
add.32	ADD rd, rs, rt	Add signed 32-bit GPR operands and trap on integer overflow.	true

Format	Assembly	Operation	Availability
add.d	ADD.D fd, fs, ft	add.d.operation	true
add.ps	ADD.PS fd, fs, ft	add.ps.operation	MMAR < R6
add.s	ADD.S fd, fs, ft	add.s.operation	true
addi.32	ADDI rt, rs, imm	Add a signed 16-bit immediate to a signed 32-bit GPR and trap on overflow.	MMAR < R6
addiu.sp.16	ADDIU imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.r1sp.16	ADDIU rt, imm(sp)	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.r2.16	ADDIU rt, rs, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.s5.16	ADDIU rt, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu	ADDIU rt, rs, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiupc.r3	ADDIUPC rt, imm	Add a 32-bit displacement to the word-aligned architectural PC. ADDIUPC/ADDIUPCR3 are the 32-bit PC-relative operations. This remains a 32-bit result even when the surrounding processor has 64-bit GPRs; the nanoMIPS P64 decoder selects DADDIUPC instead of reaching this operation.	MMAR < R6
addiupc.32.micromips	ADDIUPC rt, imm	Add a 32-bit displacement to the word-aligned architectural PC. ADDIUPC/ADDIUPCR3 are the 32-bit PC-relative operations. This remains a 32-bit result even when the surrounding processor has 64-bit GPRs; the nanoMIPS P64 decoder selects DADDIUPC instead of reaching this operation.	MMAR >= R6
addq.ph.32	ADDQ.PH rd, rs, rt	Add signed halfword lanes without saturation.	DSP

Format	Assembly	Operation	Availability
addq_s.ph.32	ADDQ_S.PH rd, rs, rt	Add signed halfword lanes with signed saturation.	DSP
addq_s.w.32	ADDQ_S.W rd, rs, rt	Add signed word lanes with signed saturation.	DSP
addqh.ph.32	ADDQH.PH rd, rs, rt	Add signed halfword lanes and halve the results.	DSP2
addqh.w.32	ADDQH.W rd, rs, rt	Add signed word lanes and halve the result.	DSP2
addqh_r.ph.32	ADDQH_R.PH rd, rs, rt	Rounded signed half-add of two halfword lanes.	DSP2
addqh_r.w.32	ADDQH_R.W rd, rs, rt	Rounded signed half-add of two word lanes.	DSP2
addsc.32	ADDSC rd, rs, rt	Add unsigned words and write the carry to DSPControl.C.	DSP
addu.16	ADDU rd, rs, rt	Add two 32-bit GPR values and sign-extend the result to GPR width.	MMAR < R6
addu.16.r6	ADDU rd, rs, rt	Add two 32-bit GPR values and sign-extend the result to GPR width.	MMAR >= R6
addu.32	ADDU rd, rs, rt	Add two 32-bit GPR values and sign-extend the result to GPR width.	true
addu.ph.32	ADDU.PH rd, rs, rt	Add unsigned halfword lanes modulo 16 bits.	DSP2
addu.qb.32	ADDU.QB rd, rs, rt	Add unsigned byte lanes modulo 8 bits and report carry.	DSP
addu_s.ph.32	ADDU_S.PH rd, rs, rt	Add unsigned halfword lanes with saturation and report carry.	DSP2
addu_s.qb.32	ADDU_S.QB rd, rs, rt	Add unsigned byte lanes with saturation and report carry.	DSP
adduh.qb.32	ADDUH.QB rd, rs, rt	Average unsigned byte lanes by an unrounded right shift.	DSP
adduh_r.qb.32	ADDUH_R.QB rd, rs, rt	Average unsigned byte lanes with rounding.	DSP
addwc.32	ADDWC rd, rs, rt	Add signed words with DSPControl.C and report signed overflow.	DSP
align.32	ALIGN rd, rs, rt, bp	align.operation	MMAR >= R6

Format	Assembly	Operation	Availability
alnv.ps.32	ALNV.PS fd, fs, ft, rs	alnv.ps.operation	MMAR < R6
aluipc.32.micromips	ALUIPC rt, imm	Add the aligned signed displacement to next_pc and clear low bits.	MMAR >= R6
and.16	AND rd, rs	Compute a full-GPR logical AND and write the result to a GPR.	MMAR < R6
and.16.r6	AND rd, rs	Compute a full-GPR logical AND and write the result to a GPR.	MMAR >= R6
and.32	AND rd, rs, rt	Compute a full-GPR logical AND and write the result to a GPR.	true
andi.16	ANDI rt, rs, imm	Compute a full-GPR logical AND with a zero-extended immediate.	true
andi.32	ANDI rt, rs, imm	Compute a full-GPR logical AND with a zero-extended immediate.	true
append.32	APPEND rt, rs, sa	append.operation	DSP2
aui.32	AUI rt, rs, imm	aui.operation	MMAR >= R6
auipc.32.micromips	AUIPC rs, offset	auipc.operation	MMAR >= R6
balc.32	BALC offset	Perform a compact PC-relative call without a delay slot.	MMAR >= R6
balign.32	BALIGN rt, rs, bp	balign.operation	DSP2
bc.r6.16	BC offset	Perform an unconditional compact PC-relative branch.	MMAR >= R6
bc.r6.32	BC offset	Perform an unconditional compact PC-relative branch.	MMAR >= R6
bc1eqzc.32	BC1EQZC ft, offset	bc1eqzc.operation	MMAR >= R6
bc1f.32	BC1F cc, offset	bc1f.operation	MMAR < R6
bc1nezc.32	BC1NEZC ft, offset	bc1nezc.operation	MMAR >= R6
bc1t.32	BC1T cc, offset	bc1t.operation	MMAR < R6
bc2eqzc.32	BC2EQZC ct, offset	bc2eqzc.operation	MMAR >= R6
bc2f.32	BC2F cc, offset	bc2f.operation	MMAR < R6
bc2nezc.32	BC2NEZC ct, offset	bc2nezc.operation	MMAR >= R6
bc2t.32	BC2T cc, offset	bc2t.operation	MMAR < R6
b.16	BEQ offset	Compare two GPRs and resolve the branch after one delay slot.	MMAR < R6
beqz.16	BEQ rs, offset	Compare two GPRs and resolve the branch after one delay slot.	MMAR < R6

Format	Assembly	Operation	Availability
beq.32	BEQ <i>rs</i> , <i>rt</i> , <i>offset</i>	Compare two GPRs and resolve the branch after one delay slot.	MMAR < R6
beqc.32	BEQC <i>rs</i> , <i>rt</i> , <i>offset</i>	Compare two GPRs and perform a compact branch when equal.	MMAR >= R6
beqzalc.r6.3 2	BEQZALC <i>rt</i> , <i>offset</i>	beqzalc.operation	MMAR >= R6
beqzc.r6.16	BEQZC <i>rt</i> , <i>offset</i>	Compact-branch when the compact source GPR is zero.	MMAR >= R6
beqzc.32	BEQZC <i>rt</i> , <i>offset</i>	Compact-branch when the compact source GPR is zero.	MMAR < R6
beqzc.r6.32	BEQZC <i>rt</i> , <i>offset</i>	Compact-branch when the compact source GPR is zero.	MMAR >= R6
bgec.r6.32	BGEC <i>rs</i> , <i>rt</i> , <i>offset</i>	Branch compactly when signed <i>rs</i> is greater than or equal to <i>rt</i> .	MMAR >= R6
bgeuc.r6.32	BGEUC <i>rs</i> , <i>rt</i> , <i>offset</i>	Branch compactly when unsigned <i>rs</i> is greater than or equal to <i>rt</i> .	MMAR >= R6
bgez.32	BGEZ <i>rs</i> , <i>offset</i>	bgez.operation	MMAR < R6
bgezal.32	BGEZAL <i>rs</i> , <i>offset</i>	bgezal.operation	MMAR < R6
bgezalc.r6.3 2	BGEZALC <i>rt</i> , <i>offset</i>	bgezalc.operation	MMAR >= R6
bgezals.32	BGEZALS <i>rs</i> , <i>offset</i>	bgezals.operation	MMAR < R6
bgezc.r6.32	BGEZC <i>rt</i> , <i>offset</i>	Compact-branch when the full-width source GPR is signed non-negative.	MMAR >= R6
bgtz.32	BGTZ <i>rs</i> , <i>offset</i>	bgtz.operation	MMAR < R6
bgtzalc.r6.3 2	BGTZALC <i>rt</i> , <i>offset</i>	bgtzalc.operation	MMAR >= R6
bgtzc.r6.32	BGTZC <i>rt</i> , <i>offset</i>	bgtzc.operation	MMAR >= R6
bitrev.32	BITREV <i>rd</i> , <i>rt</i>	bitrev.operation	DSP
bitswap.32	BITSWAP <i>rd</i> , <i>rt</i>	bitswap.operation	MMAR >= R6
blez.32	BLEZ <i>rs</i> , <i>offset</i>	blez.operation	MMAR < R6
blezalc.r6.3 2	BLEZALC <i>rt</i> , <i>offset</i>	blezalc.operation	MMAR >= R6
blezc.r6.32	BLEZC <i>rt</i> , <i>offset</i>	blezc.operation	MMAR >= R6
bltc.r6.32	BLTC <i>rs</i> , <i>rt</i> , <i>offset</i>	Branch compactly when signed <i>rs</i> is less than signed <i>rt</i> .	MMAR >= R6
bltuc.r6.32	BLTUC <i>rs</i> , <i>rt</i> , <i>offset</i>	Branch compactly when unsigned <i>rs</i> is less than <i>rt</i> .	MMAR >= R6

Format	Assembly	Operation	Availability
bltz.32	BLTZ rs, offset	bltz.operation	MMAR < R6
bltzal.32	BLTZAL rs, offset	bltzal.operation	MMAR < R6
bltzalc.r6.32	BLTZALC rt, offset	bltzalc.operation	MMAR >= R6
bltzals.32	BLTZALS rs, offset	bltzals.operation	MMAR < R6
bltzc.r6.32	BLTZC rt, offset	bltzc.operation	MMAR >= R6
bnez.16	BNE rs, offset	bne.operation	MMAR < R6
bne.32	BNE rs, rt, offset	bne.operation	MMAR < R6
bnecc.r6.32	BNECC rs, rt, offset	Compare two GPRs and perform a compact branch when they differ.	MMAR >= R6
bnezalc.r6.32	BNEZALC rt, offset	bnezalc.operation	MMAR >= R6
bnezcc.r6.16	BNEZC rt, offset	Compact-branch when the compact source GPR is nonzero.	MMAR >= R6
bnezcc.32	BNEZC rt, offset	Compact-branch when the compact source GPR is nonzero.	MMAR < R6
bnezcc.r6.32	BNEZC rt, offset	Compact-branch when the compact source GPR is nonzero.	MMAR >= R6
bnvc.r6.32	BNVC rt, rs, offset	bnvc.operation	MMAR >= R6
bovc.r6.32	BOVC rt, rs, offset	bovc.operation	MMAR >= R6
bposge32.32	BPOSGE32 offset	bposge32.operation	MMAR < R6 && DSP
bposge32c.32	BPOSGE32C offset	bposge32c.operation	MMAR >= R6 && DSP && DSP2
break.32	BREAK code	Raise a breakpoint exception.	true
break.16	BREAK code	Raise a breakpoint exception.	MMAR < R6
break.16.r6	BREAK code	Raise a breakpoint exception.	MMAR >= R6
c.cond.d	C.cond.D cc, fs, ft, condition	c.cond.d.operation	MMAR < R6
c.cond.ps	C.cond.PS cc, fs, ft, condition	Compare both PS lanes and write the selected FCC pair.	MMAR < R6
c.cond.s	C.cond.S cc, fs, ft, condition	c.cond.s.operation	MMAR < R6
cache.32	CACHE op, offset(rs)	Issue an implementation-defined cache operation through the backend.	MMAR < R6
ceil.l.d	CEIL.L.D fd, fs	ceil.l.d.operation	true
ceil.l.s	CEIL.L.S fd, fs	ceil.l.s.operation	true
ceil.w.d	CEIL.W.D fd, fs	ceil.w.d.operation	true

Format	Assembly	Operation	Availability
ceil.w.s	CEIL.W.S fd, fs	ceil.w.s.operation	true
cfc1.32	CFC1 rt, cs	cfc1.operation	true
cfc2.32	CFC2 rt, cs	cfc2.operation	true
class.d	CLASS.D fd, fs	class.d.operation	MMAR >= R6
class.s	CLASS.S fd, fs	class.s.operation	MMAR >= R6
clo.32	CLO rt, rs	Count leading one bits in the low 32 bits of rs and write the count.	true
clz.32	CLZ rt, rs	Count leading zeroes in the low 32 bits and write the count to rt.	true
cmp.d	CMP.D fd, fs, ft, condition	cmp.d.operation	MMAR >= R6
cmp.eq.ph.32	CMP.EQ.PH rs, rt	Compare signed halfword lanes for equality and write Ccond.	DSP
cmp.le.ph.32	CMP.LE.PH rs, rt	Compare signed halfword lanes for less-than-or-equal and write Ccond.	DSP
cmp.lt.ph.32	CMP.LT.PH rs, rt	Compare signed halfword lanes for less-than and write Ccond.	DSP
cmp.s	CMP.S fd, fs, ft, condition	cmp.s.operation	MMAR >= R6
cmpgdu.eq.qb.32	CMPGDU.EQ.QB rd, rs, rt	Compare unsigned byte lanes, writing both Ccond and a GPR mask.	DSP2
cmpgdu.le.qb.32	CMPGDU.LE.QB rd, rs, rt	Compare unsigned byte lanes for ≤, writing Ccond and a GPR mask.	DSP2
cmpgdu.lt.qb.32	CMPGDU.LT.QB rd, rs, rt	Compare unsigned byte lanes for less-than, writing Ccond and a GPR mask.	DSP2
cmpgu.eq.qb.32	CMPGU.EQ.QB rd, rs, rt	Compare unsigned byte lanes for equality and write a GPR mask.	DSP
cmpgu.le.qb.32	CMPGU.LE.QB rd, rs, rt	Compare unsigned byte lanes for less-than-or-equal and write a GPR mask.	DSP
cmpgu.lt.qb.32	CMPGU.LT.QB rd, rs, rt	Compare unsigned byte lanes for less-than and write a GPR mask.	DSP
cmpu.eq.qb.32	CMPU.EQ.QB rs, rt	Compare unsigned byte lanes for equality and write Ccond.	DSP

Format	Assembly	Operation	Availability
cmpl.qb.32	CMPU.LE.QB rs, rt	Compare unsigned byte lanes for less-than-or-equal and write Ccond.	DSP
cmpl.qb.32	CMPU.LT.QB rs, rt	Compare unsigned byte lanes for less-than and write Ccond.	DSP
cop2.32	COP2 cofun	cop2.operation	true
ctc1.32	CTC1 rt, cs	ctc1.operation	true
ctc2.32	CTC2 rt, cs	ctc2.operation	true
cvt.d.l	CVT.D.L fd, fs	cvt.d.l.operation	true
cvt.d.s	CVT.D.S fd, fs	cvt.d.s.operation	true
cvt.d.w	CVT.D.W fd, fs	cvt.d.w.operation	true
cvt.l.d	CVT.L.D fd, fs	cvt.l.d.operation	true
cvt.l.s	CVT.L.S fd, fs	cvt.l.s.operation	true
cvt.ps.s	CVT.PS.S fd, fs, ft	cvt.ps.s.operation	MMAR < R6
cvt.s.d	CVT.S.D fd, fs	cvt.s.d.operation	true
cvt.s.l	CVT.S.L fd, fs	cvt.s.l.operation	true
cvt.s.pl	CVT.S.PL fd, fs	cvt.s.pl.operation	MMAR < R6
cvt.s.pu	CVT.S.PU fd, fs	cvt.s.pu.operation	MMAR < R6
cvt.s.w	CVT.S.W fd, fs	cvt.s.w.operation	true
cvt.w.d	CVT.W.D fd, fs	cvt.w.d.operation	true
cvt.w.s	CVT.W.S fd, fs	cvt.w.s.operation	true
dadd	DADD rd, rs, rt	dadd.operation	AT >= MIPS64_32
daddi	DADDI rd, rs, imm	daddi.operation	AT >= MIPS64_32 && MMAR < R6
daddiu	DADDIU rd, rs, imm	daddiu.operation	AT >= MIPS64_32
daddu	DADDU rd, rs, rt	daddu.operation	AT >= MIPS64_32
dahi.32.micromips	DAHI rs, imm	dahi.operation	MMAR >= R6 && AT >= MIPS64_32
dalign.32	DALIGN rd, rs, rt, bp	dalign.operation	MMAR >= R6 && AT >= MIPS64_32
dati.32.micromips	DATI rs, imm	dati.operation	MMAR >= R6 && AT >= MIPS64_32
dau.32.micromips	DAUI rt, rs, imm	dau.operation	MMAR >= R6 && AT >= MIPS64_32
dbitswap.32.micromips	DBITSWAP rs	dbitswap.operation	MMAR >= R6 && AT >= MIPS64_32

Format	Assembly	Operation	Availability
dcl0.32	DCL0 rt, rs	dcl0.operation	AT >= MIPS64_32
dclz.32	DCLZ rt, rs	dclz.operation	AT >= MIPS64_32
ddiv.r6.32	DDIV rd, rs, rt	ddiv.operation	AT >= MIPS64_32 && MMAR >= R6
ddiv.32	DDIV rs, rt	ddiv.operation	AT >= MIPS64_32 && MMAR < R6
ddivu.r6.32	DDIVU rd, rs, rt	ddivu.operation	AT >= MIPS64_32 && MMAR >= R6
ddivu.32	DDIVU rs, rt	ddivu.operation	AT >= MIPS64_32 && MMAR < R6
dext	DEXT rt, rs, pos, size	dext.operation	AT >= MIPS64_32
dextm	DEXTM rt, rs, pos, size	dextm.operation	AT >= MIPS64_32
dextu	DEXTU rt, rs, pos, size	dextu.operation	AT >= MIPS64_32
di.32	DI rt	di.operation	true
dins	DINS rt, rs, pos, size	dins.operation	AT >= MIPS64_32
dinsm	DINSM rt, rs, pos, size	dinsm.operation	AT >= MIPS64_32
dinsu	DINSU rt, rs, pos, size	dinsu.operation	AT >= MIPS64_32
div.r6.32	DIV rd, rs, rt	Divide signed low 32-bit GPR operands and write a sign-extended quotient.	MMAR >= R6
div.32	DIV rs, rt	Divide signed low 32-bit GPR operands and write a sign-extended quotient.	MMAR < R6
div.d	DIV.D fd, fs, ft	div.d.operation	true
div.s	DIV.S fd, fs, ft	div.s.operation	true
divu.r6.32	DIVU rd, rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended quotient.	MMAR >= R6
divu.32	DIVU rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended quotient.	MMAR < R6
dlsa.32.micromips	DLSA rd, rs, rt, shift	dlsa.operation	MMAR >= R6 && AT >= MIPS64_32
dmfc0	DMFC0 rt, c0s, sel	dmfc0.operation	AT >= MIPS64_32
dmfc1.32	DMFC1 rt, fs	dmfc1.operation	AT >= MIPS64_32
dmod.r6.32	DMOD rd, rs, rt	dmod.operation	AT >= MIPS64_32 && MMAR >= R6

Format	Assembly	Operation	Availability
dmodu.r6.32	DMODU rd, rs, rt	dmodu.operation	AT >= MIPS64_32 && MMAR >= R6
dmtc0	DMTC0 rt, c0s, sel	dmtc0.operation	AT >= MIPS64_32
dmtc1.32	DMTC1 rt, fs	dmtc1.operation	AT >= MIPS64_32
dmuh.r6.32	DMUH rd, rs, rt	dmuh.operation	AT >= MIPS64_32 && MMAR >= R6
dmuhu.r6.32	DMUHU rd, rs, rt	dmuhu.operation	AT >= MIPS64_32 && MMAR >= R6
dmul.r6.32	DMUL rd, rs, rt	dmul.operation	AT >= MIPS64_32 && MMAR >= R6
dmult.32	DMULT rs, rt	dmult.operation	AT >= MIPS64_32 && MMAR < R6
dmultu.32	DMULTU rs, rt	dmultu.operation	AT >= MIPS64_32 && MMAR < R6
dmulu.r6.32	DMULU rd, rs, rt	dmulu.operation	AT >= MIPS64_32 && MMAR >= R6
dpa.w.ph.32	DPA.W.PH ac, rs, rt	dpa.w.ph.operation	DSP2
dpaq_s.w.ph.32	DPAQ_S.W.PH ac, rs, rt	dpaq_s.w.ph.operation	DSP
dpaq_sa.l.w.32	DPAQ_SA.L.W ac, rs, rt	dpaq_s.l.w.operation	DSP
dpaqx_s.w.ph.32	DPAQX_S.W.PH ac, rs, rt	dpaqx_s.w.ph.operation	DSP
dpaqx_sa.w.ph.32	DPAQX_SA.W.PH ac, rs, rt	dpaqx_sa.w.ph.operation	DSP
dpau.h.qbl.32	DPAU.H.QBL ac, rs, rt	dpau.h.qbl.operation	DSP
dpau.h.qbr.32	DPAU.H.QBR ac, rs, rt	dpau.h.qbr.operation	DSP
dpax.w.ph.32	DPAX.W.PH ac, rs, rt	dpax.w.ph.operation	DSP
dps.w.ph.32	DPS.W.PH ac, rs, rt	dps.w.ph.operation	DSP2
dpsq_s.w.ph.32	DPSQ_S.W.PH ac, rs, rt	dpsq_s.w.ph.operation	DSP
dpsq_sa.l.w.32	DPSQ_SA.L.W ac, rs, rt	dpsq_s.l.w.operation	DSP
dpsqx_s.w.ph.32	DPSQX_S.W.PH ac, rs, rt	dpsqx_s.w.ph.operation	DSP
dpsqx_sa.w.ph.32	DPSQX_SA.W.PH ac, rs, rt	dpsqx_sa.w.ph.operation	DSP
dpsu.h.qbl.32	DPSU.H.QBL ac, rs, rt	dpsu.h.qbl.operation	DSP

Format	Assembly	Operation	Availability
dpsu.h.qbr.32	DPSU.H.QBR ac, rs, rt	dpsu.h.qbr.operation	DSP
dpsx.w.ph.32	DPSX.W.PH ac, rs, rt	dpsx.w.ph.operation	DSP
drotr	DROTR rd, rs, shift	drotr.operation	AT >= MIPS64_32
drotr32	DROTR32 rd, rs, shift	drotr32.operation	AT >= MIPS64_32
drotrv	DROTRV rd, rs, rt	drotrv.operation	AT >= MIPS64_32
dsbh.32	DSBH rt, rs	dsbh.operation	AT >= MIPS64_32
dshd.32	DSHD rt, rs	dshd.operation	AT >= MIPS64_32
dsll	DSLL rd, rs, shift	dsll.operation	AT >= MIPS64_32
dsll32	DSLL32 rd, rs, shift	dsll32.operation	AT >= MIPS64_32
dsllv	DSLLV rd, rs, rt	dsllv.operation	AT >= MIPS64_32
dsra	DSRA rd, rs, shift	dsra.operation	AT >= MIPS64_32
dsra32	DSRA32 rd, rs, shift	dsra32.operation	AT >= MIPS64_32
dsrav	DSRAV rd, rs, rt	dsrav.operation	AT >= MIPS64_32
dsrl	DSRL rd, rs, shift	dsrl.operation	AT >= MIPS64_32
dsrl32	DSRL32 rd, rs, shift	dsrl32.operation	AT >= MIPS64_32
dsrlv	DSRLV rd, rs, rt	dsrlv.operation	AT >= MIPS64_32
dsub	DSUB rd, rs, rt	dsub.operation	AT >= MIPS64_32
dsubu	DSUBU rd, rs, rt	dsubu.operation	AT >= MIPS64_32
dvp.32.micro mips	DVP rt	dvp.operation	MMAR >= R6
ei.32	EI rt	ei.operation	true
eret.32	ERET	Return from an exception and clear the LL reservation.	true
eretnc.32	ERETNC	Return from an exception without clearing the LL reservation.	MMAR >= R6
evp.32.micro mips	EVP rt	evp.operation	MMAR >= R6
ext.32	EXT rt, rs, pos, size	Extract a 32-bit field and sign-extend the result to the GPR width.	true
extp.32	EXTP rt, ac, size	extp.operation	DSP
extpdp.32	EXTPDP rt, ac, size	extpdp.operation	DSP
extpdpv.32	EXTPDPV rt, ac, rs	extpdpv.operation	DSP
extpv.32	EXTPV rt, ac, rs	extpv.operation	DSP
extr.w.32	EXTR.W rt, ac, shift	extr.w.operation	DSP

Format	Assembly	Operation	Availability
extr_r.w.32	EXTR_R.W rt, ac, shift	extr_r.w.operation	DSP
extr_rs.w.32	EXTR_RS.W rt, ac, shift	extr_rs.w.operation	DSP
extr_s.h.32	EXTR_S.H rt, ac, shift	extr_s.h.operation	DSP
extrv.w.32	EXTRV.W rt, ac, rs	extrv.w.operation	DSP
extrv_r.w.32	EXTRV_R.W rt, ac, rs	extrv_r.w.operation	DSP
extrv_rs.w.32	EXTRV_RS.W rt, ac, rs	extrv_rs.w.operation	DSP
extrv_s.h.32	EXTRV_S.H rt, ac, rs	extrv_s.h.operation	DSP
floor.l.d	FLOOR.L.D fd, fs	floor.l.d.operation	true
floor.l.s	FLOOR.L.S fd, fs	floor.l.s.operation	true
floor.w.d	FLOOR.W.D fd, fs	floor.w.d.operation	true
floor.w.s	FLOOR.W.S fd, fs	floor.w.s.operation	true
ins.32	INS rt, rs, pos, size	Merge a right-justified source field into a 32-bit destination word.	true
insv.32	INSV rt, rs	insv.operation	DSP
j.32	J index	j.operation	MMAR < R6
jal.32	JAL index	jal.operation	MMAR < R6
jalr.32	JALR rd, rs	jalr.operation	MMAR < R6
jalr.hb.32	JALR rd, rs	jalr.operation	MMAR < R6
jalr.16	JALR rs	jalr.operation	MMAR < R6
jalrc.32	JALRC dst, src	Jump to a GPR address and save the prepared sequential next PC.	MMAR >= R6
jalrc.hb.32	JALRC dst, src	Jump to a GPR address and save the prepared sequential next PC.	MMAR >= R6
jalrc.16	JALRC src	Jump to a GPR address and save the prepared sequential next PC.	MMAR >= R6
jalrs.32	JALRS rd, rs	Jump through a GPR and link across a 16-bit delay-slot instruction.	MMAR < R6
jalrs.hb.32	JALRS rd, rs	Jump through a GPR and link across a 16-bit delay-slot instruction.	MMAR < R6
jalrs.16	JALRS rs	Jump through a GPR and link across a 16-bit delay-slot instruction.	MMAR < R6
jals.32	JALS index	Jump and link with a 16-bit microMIPS delay-slot instruction.	MMAR < R6

Format	Assembly	Operation	Availability
jalx.32	JALX index	Call a region-relative target and exchange the ISA mode after its slot.	MMAR < R6
jialc.32.micromips	JIALC rt, offset	jialc.operation	MMAR >= R6
jic.32.micromips	JIC rt, offset	jic.operation	MMAR >= R6
jr.16	JR rs	jr.operation	MMAR < R6
jraddiusp.16	JRADDIUSP imm	Adjust the stack pointer and jump immediately through GPR 31.	MMAR < R6
jrc.16	JRC rt	Jump immediately to the address held in the source GPR.	MMAR < R6
jrc.r6.16	JRC rt	Jump immediately to the address held in the source GPR.	MMAR >= R6
jrcaddiusp.16	JRCADDIUSP imm	Adjust SP by a scaled immediate and compact-jump through RA.	MMAR >= R6
lb.32	LB rt, offset(rs)	Load one byte and sign-extend it to the implemented GPR width.	true
lbu.16	LBU rt, offset(rs)	Load one byte and zero-extend it to the implemented GPR width.	true
lbu.32	LBU rt, offset(rs)	Load one byte and zero-extend it to the implemented GPR width.	true
lbux.dsp.32	LBUX rs(rt)	lbux.operation	DSP
ld	LD rt, offset(rs)	ld.operation	AT >= MIPS64_32
ldc1.32	LDC1 ft, offset(base)	ldc1.operation	true
ldc2.32	LDC2 ct, offset(base)	ldc2.operation	MMAR < R6
ldc2.r6.32	LDC2 ct, offset(base)	ldc2.operation	MMAR >= R6
ldl.32	LDL rt, offset(rs)	ldl.operation	MMAR < R6 && AT >= MIPS64_32
ldpc.32.micromips	LDPC rt, offset	ldpc.operation	MMAR >= R6 && AT >= MIPS64_32
ldr.32	LDR rt, offset(rs)	ldr.operation	MMAR < R6 && AT >= MIPS64_32
ldxc1.32	LDXC1 fd, index(base)	ldxc1.operation	MMAR < R6
lh.32	LH rt, offset(rs)	Load a signed halfword from memory and extend it to GPR width.	true

Format	Assembly	Operation	Availability
lhu.16	LHU rt, offset(rs)	Load an unsigned halfword from memory and extend it to GPR width.	true
lhu.32	LHU rt, offset(rs)	Load an unsigned halfword from memory and extend it to GPR width.	true
lhx.dsp.32	LHX rs(rt)	lhx.operation	DSP
li.16	LI rt, imm	Write the decoded immediate to the destination GPR.	true
ll.32	LL rt, offset(rs)	Load a sign-extended word and establish a linked reservation.	MMAR < R6
ll.r6.32	LL rt, offset(rs)	Load a sign-extended word and establish a linked reservation.	MMAR >= R6
lld.32	LLD rt, offset(rs)	lld.operation	MMAR < R6 && AT >= MIPS64_32
lld.r6.32	LLD rt, offset(rs)	lld.operation	MMAR >= R6 && AT >= MIPS64_32
lldp.32	LLDP rt, ru, (rs)	lldp.operation	MMAR >= R6 && AT >= MIPS64_32
llwp.32	LLWP rt, ru, (rs)	Load a linked pair of sign-extended words.	MMAR >= R6
lsa.32.micro mips	LSA rd, rs, rt, shift	Add a scaled 32-bit GPR value to another and sign-extend the result.	MMAR >= R6
lui.32	LUI rt, imm	Load the sign-extended 32-bit upper-immediate value into a GPR.	MMAR < R6
luxc1.32	LUXC1 fd, index(base)	luxc1.operation	MMAR < R6
lw.16	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.gp.16	LW rt, offset(gp)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.32	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.sp.16	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lwc1.32	LWC1 ft, offset(base)	lwc1.operation	true
lwc2.32	LWC2 ct, offset(base)	lwc2.operation	MMAR < R6
lwc2.r6.32	LWC2 ct, offset(base)	lwc2.operation	MMAR >= R6

Format	Assembly	Operation	Availability
lwl.32	LWL rt, offset(rs)	lwl.operation	MMAR < R6
lwm.32	LWM offset(rs)	Load consecutive aligned words into sequential GPRs.	true
lwm.16	LWM offset(sp)	Load consecutive aligned words into sequential GPRs.	MMAR < R6
lwm.r6.16	LWM offset(sp)	Load consecutive aligned words into sequential GPRs.	MMAR >= R6
lwp.32	LWP rd, offset(rs)	micromips.lwp.operation	true
lwpc.32.micromips	LWPC rt, offset	Load a signed word using the active architecture's PC-relative form.	MMAR >= R6
lwr.32	LWR rt, offset(rs)	Merge the right part of an aligned word into the low 32-bit GPR value.	MMAR < R6
lwu	LWU rt, offset(rs)	lwu.operation	AT >= MIPS64_32
lwupc.32.micromips	LWUPC rt, offset	Load an unsigned word using the MIPS PC-relative form.	MMAR >= R6 && AT >= MIPS64_32
lwx.dsp.32	LWX rs(rt)	lwx.operation	DSP
lwxc1.32	LWXC1 fd, index(base)	lwxc1.operation	MMAR < R6
lwxs.32	LWXS rd, rs(rt)	lwxs.operation	MMAR < R6
madd.dsp.32	MADD ac, rs, rt	madd.operation	DSP
madd.32	MADD rs, rt	madd.operation	MMAR < R6
madd.d	MADD.D fd, fr, fs, ft	madd.d.legacy.operation	MMAR < R6
madd.ps	MADD.PS fd, fr, fs, ft	madd.ps.legacy.operation	MMAR < R6
madd.s	MADD.S fd, fr, fs, ft	madd.s.legacy.operation	MMAR < R6
maddf.d	MADDF.D fd, fs, ft	maddf.d.operation	MMAR >= R6
maddf.s	MADDF.S fd, fs, ft	maddf.s.operation	MMAR >= R6
maddu.dsp.32	MADDU ac, rs, rt	maddu.operation	DSP
maddu.32	MADDU rs, rt	maddu.operation	MMAR < R6
maq_s.w.phl.32	MAQ_S.W.PHL ac, rs, rt	maq_s.w.phl.operation	DSP
maq_s.w.phr.32	MAQ_S.W.PHR ac, rs, rt	maq_s.w.phr.operation	DSP
maq_sa.w.phl.32	MAQ_SA.W.PHL ac, rs, rt	maq_sa.w.phl.operation	DSP
maq_sa.w.phr.32	MAQ_SA.W.PHR ac, rs, rt	maq_sa.w.phr.operation	DSP
max.d	MAX.D fd, fs, ft	max.d.operation	MMAR >= R6

Format	Assembly	Operation	Availability
max.s	MAX.S fd, fs, ft	max.s.operation	MMAR >= R6
maxa.d	MAXA.D fd, fs, ft	maxa.d.operation	MMAR >= R6
maxa.s	MAXA.S fd, fs, ft	maxa.s.operation	MMAR >= R6
mfc0.32	MFC0 rt, c0s, sel	Read a CP0 register and sign-extend its low 32-bit result to GPR width.	true
mfc1.32	MFC1 rt, fs	Copy the low word of an FPR to a GPR after checking CP1 access.	true
mfc2.32	MFC2 rt, cs	mfc2.operation	true
mfhc0.32	MFHC0 rt, c0s, sel	mfhc0.operation	AT >= MIPS64_32 MVH
mfhc1.32	MFHC1 rt, fs	mfhc1.operation	AT >= MIPS64_32 FPU64
mfhc2.32	MFHC2 rt, cs	mfhc2.operation	true
mfhi.16	MFHI rd	mfhi.operation	MMAR < R6
mfhi.32	MFHI rd	mfhi.operation	MMAR < R6
mfhi.dsp.32	MFHI rd, ac	mfhi.operation	DSP
mflo.16	MFLO rd	mflo.operation	MMAR < R6
mflo.32	MFLO rd	mflo.operation	MMAR < R6
mflo.dsp.32	MFLO rd, ac	mflo.operation	DSP
min.d	MIN.D fd, fs, ft	min.d.operation	MMAR >= R6
min.s	MIN.S fd, fs, ft	min.s.operation	MMAR >= R6
mina.d	MINA.D fd, fs, ft	mina.d.operation	MMAR >= R6
mina.s	MINA.S fd, fs, ft	mina.s.operation	MMAR >= R6
mod.32.micro mips	MOD rd, rs, rt	Divide signed low 32-bit GPR operands and write a sign-extended remainder.	MMAR >= R6
modsub.32	MODSUB rd, rs, rt	Subtract a decrement, wrapping to the supplied last index at zero.	DSP
modu.32.micr omips	MODU rd, rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended remainder.	MMAR >= R6
mov.d	MOV.D fd, fs	mov.d.operation	true
mov.ps	MOV.PS fd, fs	mov.ps.operation	MMAR < R6
mov.s	MOV.S fd, fs	mov.s.operation	true

Format	Assembly	Operation	Availability
move.16	MOVE rt, rs	Copy the complete implemented-width value from rs to rt.	true
movep.16	MOVEP dst1, dst2, src1, src2	Copy a pair of registers; overlapping pairs are architecturally unknown.	MMAR < R6
movep.r6.16	MOVEP dst1, dst2, src1, src2	Copy a pair of registers; overlapping pairs are architecturally unknown.	MMAR >= R6
movf	MOVF rd, rs, cc	movf.operation	MMAR < R6
movf.d.32	MOVF.D fd, fs, cc	movf.d.operation	MMAR < R6
movf.ps.32	MOVF.PS fd, fs, cc	movf.ps.operation	MMAR < R6
movf.s.32	MOVF.S fd, fs, cc	movf.s.operation	MMAR < R6
movn.32	MOVN rd, rs, rt	Copy rs to rd when the full-width value in rt is nonzero.	MMAR < R6
movn.d	MOVN.D fd, fs, rt	movn.d.legacy.operation	MMAR < R6
movn.ps	MOVN.PS fd, fs, rt	movn.ps.legacy.operation	MMAR < R6
movn.s	MOVN.S fd, fs, rt	movn.s.legacy.operation	MMAR < R6
movt	MOVT rd, rs, cc	movt.operation	MMAR < R6
movt.d.32	MOVT.D fd, fs, cc	movt.d.operation	MMAR < R6
movt.ps.32	MOVT.PS fd, fs, cc	movt.ps.operation	MMAR < R6
movt.s.32	MOVT.S fd, fs, cc	movt.s.operation	MMAR < R6
movz.32	MOVZ rd, rs, rt	Copy rs to rd when the full-width value in rt is zero.	MMAR < R6
movz.d	MOVZ.D fd, fs, rt	movz.d.legacy.operation	MMAR < R6
movz.ps	MOVZ.PS fd, fs, rt	movz.ps.legacy.operation	MMAR < R6
movz.s	MOVZ.S fd, fs, rt	movz.s.legacy.operation	MMAR < R6
msub.dsp.32	MSUB ac, rs, rt	msub.operation	DSP
msub.32	MSUB rs, rt	msub.operation	MMAR < R6
msub.d	MSUB.D fd, fr, fs, ft	msub.d.legacy.operation	MMAR < R6
msub.ps	MSUB.PS fd, fr, fs, ft	msub.ps.legacy.operation	MMAR < R6
msub.s	MSUB.S fd, fr, fs, ft	msub.s.legacy.operation	MMAR < R6
msubf.d	MSUBF.D fd, fs, ft	msubf.d.operation	MMAR >= R6
msubf.s	MSUBF.S fd, fs, ft	msubf.s.operation	MMAR >= R6
msubu.dsp.32	MSUBU ac, rs, rt	msubu.operation	DSP
msubu.32	MSUBU rs, rt	msubu.operation	MMAR < R6

Format	Assembly	Operation	Availability
mtc0.32	MTC0 rt, c0s, sel	Write a GPR to an encoded CP0 register. On a 64-bit MIPS core the MTC0 data path is the full GPR width; the MFC0/DMFC0 pair then exposes the low-word and full-width read views. A 32-bit core retains the ordinary 32-bit path.	true
mtc1.32	MTC1 rt, fs	mtc1.operation	true
mtc2.32	MTC2 rt, cs	mtc2.operation	true
mthc0.32	MTHC0 rt, c0s, sel	mthc0.operation	AT >= MIPS64_32 MVH
mthc1.32	MTHC1 rt, fs	mthc1.operation	AT >= MIPS64_32 FPU64
mthc2.32	MTHC2 rt, cs	mthc2.operation	true
mthi.32	MTHI rs	mthi.operation	MMAR < R6
mthi.dsp.32	MTHI rs, ac	mthi.operation	DSP
mthlip.32	MTHLIP rs, ac	mthlip.operation	DSP
mtlo.32	MTLO rs	mtlo.operation	MMAR < R6
mtlo.dsp.32	MTLO rs, ac	mtlo.operation	DSP
muh.32.micro mips	MUH rd, rs, rt	Multiply signed words and write the sign-extended high word.	MMAR >= R6
muhu.32.micr omips	MUHU rd, rs, rt	Multiply unsigned words and write the high word.	MMAR >= R6
mul.32	MUL rd, rs, rt	Multiply signed 32-bit GPR operands and sign-extend the low word.	MMAR < R6
mul.r6.32	MUL rd, rs, rt	Multiply signed 32-bit GPR operands and sign-extend the low word.	MMAR >= R6
mul.d	MUL.D fd, fs, ft	mul.d.operation	true
mul.ph.32	MUL.PH rd, rs, rt	mul.ph.operation	DSP
mul.ps	MUL.PS fd, fs, ft	mul.ps.operation	MMAR < R6
mul.s	MUL.S fd, fs, ft	mul.s.operation	true
mul_s.ph.32	MUL_S.PH rd, rs, rt	mul_s.ph.operation	DSP
muleq_s.w.ph l.32	MULEQ_S.W.PHL rd, rs, rt	muleq_s.w.phl.operation	DSP
muleq_s.w.ph r.32	MULEQ_S.W.PHR rd, rs, rt	muleq_s.w.phr.operation	DSP

Format	Assembly	Operation	Availability
muleu_s.ph.qbl.32	MULEU_S.PH.QBL rd, rs, rt	muleu_s.ph.qbl.operation	DSP
muleu_s.ph.qbr.32	MULEU_S.PH.QBR rd, rs, rt	muleu_s.ph.qbr.operation	DSP
mulq_rs.ph.32	MULQ_RS.PH rd, rs, rt	mulq_rs.ph.operation	DSP
mulq_rs.w.32	MULQ_RS.W rd, rs, rt	mulq_rs.w.operation	DSP2
mulq_s.ph.32	MULQ_S.PH rd, rs, rt	mulq_s.ph.operation	DSP2
mulq_s.w.32	MULQ_S.W rd, rs, rt	mulq_s.w.operation	DSP2
mulsa.w.ph.32	MULSA.W.PH ac, rs, rt	mulsa.w.ph.operation	DSP2
mulsaq_s.w.ph.32	MULSAQ_S.W.PH ac, rs, rt	mulsaq_s.w.ph.operation	DSP
mult.dsp.32	MULT ac, rs, rt	mult.operation	DSP
mult.32	MULT rs, rt	mult.operation	MMAR < R6
multu.dsp.32	MULTU ac, rs, rt	multu.operation	DSP
multu.32	MULTU rs, rt	multu.operation	MMAR < R6
mulu.32.micromips	MULU rd, rs, rt	Multiply unsigned words and write the sign-extended low word.	MMAR >= R6
neg.d	NEG.D fd, fs	neg.d.operation	true
neg.ps	NEG.PS fd, fs	neg.ps.operation	MMAR < R6
neg.s	NEG.S fd, fs	neg.s.operation	true
nmadd.d	NMADD.D fd, fr, fs, ft	nmadd.d.legacy.operation	MMAR < R6
nmadd.ps	NMADD.PS fd, fr, fs, ft	nmadd.ps.legacy.operation	MMAR < R6
nmadd.s	NMADD.S fd, fr, fs, ft	nmadd.s.legacy.operation	MMAR < R6
nmsub.d	NMSUB.D fd, fr, fs, ft	nmsub.d.legacy.operation	MMAR < R6
nmsub.ps	NMSUB.PS fd, fr, fs, ft	nmsub.ps.legacy.operation	MMAR < R6
nmsub.s	NMSUB.S fd, fr, fs, ft	nmsub.s.legacy.operation	MMAR < R6
nop.32	NOP	Perform no architectural operation.	true
nor.32	NOR rd, rs, rt	Compute a bitwise NOR at the implemented GPR width.	true
not.16	NOT rt, rs	Invert the complete implemented-width source GPR.	MMAR < R6
not.r6.16	NOT rt, rs	Invert the complete implemented-width source GPR.	MMAR >= R6
or.16	OR rd, rs	Compute a full-GPR logical OR and write the result to a GPR.	MMAR < R6

Format	Assembly	Operation	Availability
or.16.r6	OR rd, rs	Compute a full-GPR logical OR and write the result to a GPR.	MMAR >= R6
or.32	OR rd, rs, rt	Compute a full-GPR logical OR and write the result to a GPR.	true
ori.32	ORI rt, rs, imm	Compute a full-GPR logical OR with a zero-extended immediate.	true
packrl.ph.32	PACKRL.PH rd, rs, rt	packrl.ph.operation	DSP
pick.ph.32	PICK.PH rd, rs, rt	Select halfword lanes from rs or rt according to Ccond.	DSP
pick.qb.32	PICK.QB rd, rs, rt	Select byte lanes from rs or rt according to Ccond.	DSP
pll.ps.32	PLL.PS fd, fs, ft	pll.ps.operation	MMAR < R6
plu.ps.32	PLU.PS fd, fs, ft	plu.ps.operation	MMAR < R6
preceq.w.phl.32	PRECEQ.W.PHL rd, rt	preceq.w.phl.operation	DSP
preceq.w.phr.32	PRECEQ.W.PHR rd, rt	preceq.w.phr.operation	DSP
precequ.ph.qbl.32	PRECEQU.PH.QBL rd, rt	precequ.ph.qbl.operation	DSP
precequ.ph.qbla.32	PRECEQU.PH.QBLA rd, rt	precequ.ph.qbla.operation	DSP
precequ.ph.qbr.32	PRECEQU.PH.QBR rd, rt	precequ.ph.qbr.operation	DSP
precequ.ph.qbra.32	PRECEQU.PH.QBRA rd, rt	precequ.ph.qbra.operation	DSP
preceu.ph.qbl.32	PRECEU.PH.QBL rd, rt	preceu.ph.qbl.operation	DSP
preceu.ph.qbla.32	PRECEU.PH.QBLA rd, rt	preceu.ph.qbla.operation	DSP
preceu.ph.qbr.32	PRECEU.PH.QBR rd, rt	preceu.ph.qbr.operation	DSP
preceu.ph.qbra.32	PRECEU.PH.QBRA rd, rt	preceu.ph.qbra.operation	DSP
precr.qb.ph.32	PRECR.QB.PH rd, rs, rt	precr.qb.ph.operation	DSP2
precr_sra.ph.w.32	PRECR_SRA.PH.W rt, rs, sa	precr_sra.ph.w.operation	DSP2
precr_sra_r.ph.w.32	PRECR_SRA_R.PH.W rt, rs, sa	precr_sra_r.ph.w.operation	DSP2
precrq.ph.w.32	PRECRQ.PH.W rd, rs, rt	precrq.ph.w.operation	DSP
precrq.qb.ph.32	PRECRQ.QB.PH rd, rs, rt	precrq.qb.ph.operation	DSP

Format	Assembly	Operation	Availability
precrq_rs.ph.w.32	PRECRQ_RS.PH.W rd, rs, rt	precrq_rs.ph.w.operation	DSP
precrqu_s.qb.ph.32	PRECRQU_S.QB.PH rd, rs, rt	precrqu_s.qb.ph.operation	DSP
pref.32	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	MMAR < R6
pref.32.r6	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	MMAR >= R6
prefe.32	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	true
prefx	PREFX hint, index(base)	Issue an implementation-dependent prefetch at base plus index.	MMAR < R6
prepend.32	PREPEND rt, rs, sa	prepend.operation	DSP2
pul.ps.32	PUL.PS fd, fs, ft	pul.ps.operation	MMAR < R6
puu.ps.32	PUU.PS fd, fs, ft	puu.ps.operation	MMAR < R6
raddu.w.qb.32	RADDU.W.QB rd, rs	Reduce four unsigned bytes to a zero-extended ten-bit word.	DSP
rddsp.32	RDDSP rd, mask	rddsp.operation	DSP
rdhwr.32	RDHWR rt, hs	rdhwr.operation	MMAR < R6
rdhwr.r6.32	RDHWR rt, hs, sel	rdhwr.operation	MMAR >= R6
rdpgpr.32	RDPGPR dst, src	Read a GPR from the previous shadow set into the current set.	true
recip.d	RECIP.D fd, fs	recip.d.operation	true
recip.s	RECIP.S fd, fs	recip.s.operation	true
repl.ph.32	REPL.PH rd, immediate	repl.ph.operation	DSP
repl.qb.32	REPL.QB rd, immediate	repl.qb.operation	DSP
replv.ph.32	REPLV.PH rd, rt	replv.ph.operation	DSP
replv.qb.32	REPLV.QB rd, rt	replv.qb.operation	DSP
rint.d	RINT.D fd, fs	rint.d.operation	MMAR >= R6
rint.s	RINT.S fd, fs	rint.s.operation	MMAR >= R6
rotr.32	ROTR rt, rs, shift	Rotate the low 32-bit source word right and sign-extend the result.	true
rotrv.32	ROTRV rd, rs, rt	Rotate a 32-bit GPR word right by a GPR-supplied amount.	true

Format	Assembly	Operation	Availability
round.l.d	ROUND.L.D fd, fs	round.l.d.operation	true
round.l.s	ROUND.L.S fd, fs	round.l.s.operation	true
round.w.d	ROUND.W.D fd, fs	round.w.d.operation	true
round.w.s	ROUND.W.S fd, fs	round.w.s.operation	true
rsqrt.d	RSQRT.D fd, fs	rsqrt.d.operation	true
rsqrt.s	RSQRT.S fd, fs	rsqrt.s.operation	true
sb.16	SB rt, offset(rs)	Store the low byte of rt at the effective address rs plus offset.	true
sb.32	SB rt, offset(rs)	Store the low byte of rt at the effective address rs plus offset.	true
sc.32	SC rt, offset(rs)	Conditionally store a word and return success in rt.	MMAR < R6
sc.r6.32	SC rt, offset(rs)	Conditionally store a word and return success in rt.	MMAR >= R6
scd.32	SCD rt, offset(rs)	scd.operation	MMAR < R6 && AT >= MIPS64_32
scd.r6.32	SCD rt, offset(rs)	scd.operation	MMAR >= R6 && AT >= MIPS64_32
scdp.32	SCDP rt, ru, (rs)	scdp.operation	MMAR >= R6 && AT >= MIPS64_32
scwp.32	SCWP rt, ru, (rs)	Conditionally store a word pair and return success in rt.	MMAR >= R6
sd	SD rt, offset(rs)	sd.operation	AT >= MIPS64_32
sdc1.32	SDC1 ft, offset(base)	sdc1.operation	true
sdc2.32	SDC2 ct, offset(base)	sdc2.operation	MMAR < R6
sdc2.r6.32	SDC2 ct, offset(base)	sdc2.operation	MMAR >= R6
sdl.32	SDL rt, offset(rs)	sdl.operation	MMAR < R6 && AT >= MIPS64_32
sdr.32	SDR rt, offset(rs)	sdr.operation	MMAR < R6 && AT >= MIPS64_32
sdx1.32	SDXC1 fd, index(base)	sdx1.operation	MMAR < R6
seb.32	SEB rt, rs	Sign-extend the low byte of rs to the implemented GPR width.	true
seh.32	SEH rt, rs	Sign-extend the low halfword of rs to the implemented GPR width.	true
sel.d	SEL.D fd, fs, ft	sel.d.operation	MMAR >= R6
sel.s	SEL.S fd, fs, ft	sel.s.operation	MMAR >= R6

Format	Assembly	Operation	Availability
seleqz.32	SELEQZ rd, rs, rt	seleqz.operation	MMAR >= R6
seleqz.d	SELEQZ.D fd, fs, ft	seleqz.d.operation	MMAR >= R6
seleqz.s	SELEQZ.S fd, fs, ft	seleqz.s.operation	MMAR >= R6
selnez.32	SELNEZ rd, rs, rt	selnez.operation	MMAR >= R6
selnez.d	SELNEZ.D fd, fs, ft	selnez.d.operation	MMAR >= R6
selnez.s	SELNEZ.S fd, fs, ft	selnez.s.operation	MMAR >= R6
sh.16	SH rt, offset(rs)	Store the low 16 bits of rt at rs plus offset.	true
sh.32	SH rt, offset(rs)	Store the low 16 bits of rt at rs plus offset.	true
shilo.32	SHILO ac, shift	shilo.operation	DSP
shilov.32	SHILOV ac, rs	shilov.operation	DSP
shll.ph	SHLL.PH rd, rt, sa	dsp.shll.ph	DSP
shll.qb	SHLL.QB rd, rt, sa	dsp.shll.qb	DSP
shll_s.ph	SHLL_S.PH rd, rt, sa	dsp.shll_s.ph	DSP
shll_s.w	SHLL_S.W rd, rt, sa	dsp.shll_s.w	DSP
shllv.ph	SHLLV.PH rd, rt, rs	dsp.shllv.ph	DSP
shllv.qb	SHLLV.QB rd, rt, rs	dsp.shllv.qb	DSP
shllv_s.ph	SHLLV_S.PH rd, rt, rs	dsp.shllv_s.ph	DSP
shllv_s.w	SHLLV_S.W rd, rt, rs	dsp.shllv_s.w	DSP
shra.ph	SHRA.PH rd, rt, sa	dsp.shra.ph	DSP2
shra.qb	SHRA.QB rd, rt, sa	dsp.shra.qb	DSP2
shra_r.ph	SHRA_R.PH rd, rt, sa	dsp.shra_r.ph	DSP2
shra_r.qb	SHRA_R.QB rd, rt, sa	dsp.shra_r.qb	DSP2
shra_r.w	SHRA_R.W rd, rt, sa	dsp.shra_r.w	DSP
shrav.ph	SHRAV.PH rd, rt, rs	dsp.shrav.ph	DSP
shrav.qb	SHRAV.QB rd, rt, rs	dsp.shrav.qb	DSP2
shrav_r.ph	SHRAV_R.PH rd, rt, rs	dsp.shrav_r.ph	DSP
shrav_r.qb	SHRAV_R.QB rd, rt, rs	dsp.shrav_r.qb	DSP2
shrav_r.w	SHRAV_R.W rd, rt, rs	dsp.shrav_r.w	DSP
shrl.ph	SHRL.PH rd, rt, sa	dsp.shrl.ph	DSP2
shrl.qb	SHRL.QB rd, rt, sa	dsp.shrl.qb	DSP
shrlv.ph	SHRLV.PH rd, rt, rs	dsp.shrlv.ph	DSP2
shrlv.qb	SHRLV.QB rd, rt, rs	dsp.shrlv.qb	DSP

Format	Assembly	Operation	Availability
sll.16	SLL rt, rs, shift	Shift a 32-bit GPR word left and sign-extend the result.	true
sll.32	SLL rt, rs, shift	Shift a 32-bit GPR word left and sign-extend the result.	true
sllv.32	SLLV rd, rs, rt	Shift a 32-bit GPR word left by the low five bits of a GPR.	true
slt.32	SLT rd, rs, rt	Set rd when the signed, implemented-width rs is below rt.	true
slti.32	SLTI rt, rs, imm	Set the destination when a signed full-width GPR is below imm.	true
sltiu.32	SLTIU rt, rs, imm	Set the destination to one when an unsigned GPR is below the immediate.	true
sltu.32	SLTU rd, rs, rt	Set rd when the unsigned, implemented-width rs is below rt.	true
sqrtd	SQRT.D fd, fs	sqrtd.operation	true
sqrts	SQRT.S fd, fs	sqrts.operation	true
sra.32	SRA rt, rs, shift	Arithmetic-shift a signed 32-bit GPR word right and extend it.	true
srav.32	SRAV rd, rs, rt	Arithmetic-shift a 32-bit GPR word by a register-supplied amount.	true
srl.16	SRL rt, rs, shift	Shift a 32-bit GPR word right logically and sign-extend the result.	true
srl.32	SRL rt, rs, shift	Shift a 32-bit GPR word right logically and sign-extend the result.	true
srlv.32	SRLV rd, rs, rt	Shift a 32-bit GPR word right logically by a register-supplied amount.	true
sub.32	SUB rd, rs, rt	Subtract signed 32-bit GPR operands and trap on integer overflow.	true
sub.d	SUB.D fd, fs, ft	sub.d.operation	true
sub.ps	SUB.PS fd, fs, ft	sub.ps.operation	MMAR < R6
sub.s	SUB.S fd, fs, ft	sub.s.operation	true
subq.ph.32	SUBQ.PH rd, rs, rt	Subtract signed halfword lanes without saturation.	DSP

Format	Assembly	Operation	Availability
subq_s.ph.32	SUBQ_S.PH rd, rs, rt	Subtract signed halfword lanes with signed saturation.	DSP
subq_s.w.32	SUBQ_S.W rd, rs, rt	Subtract signed word lanes with signed saturation.	DSP
subqh.ph.32	SUBQH.PH rd, rs, rt	Subtract signed halfword lanes and halve the results.	DSP2
subqh.w.32	SUBQH.W rd, rs, rt	Subtract signed word lanes and halve the result.	DSP2
subqh_r.ph.32	SUBQH_R.PH rd, rs, rt	Rounded signed half-subtract of two halfword lanes.	DSP2
subqh_r.w.32	SUBQH_R.W rd, rs, rt	Rounded signed half-subtract of two word lanes.	DSP2
subu.16	SUBU rd, rs, rt	Subtract 32-bit GPR rt from rs and sign-extend to GPR width.	MMAR < R6
subu.16.r6	SUBU rd, rs, rt	Subtract 32-bit GPR rt from rs and sign-extend to GPR width.	MMAR >= R6
subu.32	SUBU rd, rs, rt	Subtract 32-bit GPR rt from rs and sign-extend to GPR width.	true
subu.ph.32	SUBU.PH rd, rs, rt	Subtract unsigned halfword lanes modulo 16 bits.	DSP2
subu.qb.32	SUBU.QB rd, rs, rt	Subtract unsigned byte lanes modulo 8 bits.	DSP
subu_s.ph.32	SUBU_S.PH rd, rs, rt	Subtract unsigned halfword lanes with saturation.	DSP2
subu_s.qb.32	SUBU_S.QB rd, rs, rt	Subtract unsigned byte lanes with saturation.	DSP
subuh.qb.32	SUBUH.QB rd, rs, rt	Average unsigned byte differences by an unrounded right shift.	DSP
subuh_r.qb.32	SUBUH_R.QB rd, rs, rt	Average unsigned byte differences with rounding.	DSP
suxc1.32	SUXC1 fd, index(base)	suxc1.operation	MMAR < R6
sw.16	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	true
sw.32	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	true
sw.sp.16	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	true
swc1.32	SWC1 ft, offset(base)	swc1.operation	true

Format	Assembly	Operation	Availability
swc2.32	SWC2 ct, offset(base)	swc2.operation	MMAR < R6
swc2.r6.32	SWC2 ct, offset(base)	swc2.operation	MMAR >= R6
swl.32	SWL rt, offset(rs)	swl.operation	MMAR < R6
swm.32	SWM offset(rs)	Store consecutive aligned words from sequential GPRs.	true
swm.16	SWM offset(sp)	Store consecutive aligned words from sequential GPRs.	MMAR < R6
swm.r6.16	SWM offset(sp)	Store consecutive aligned words from sequential GPRs.	MMAR >= R6
swp.32	SWP rd, offset(rs)	micromips.swp.operation	true
swr.32	SWR rt, offset(rs)	swr.operation	MMAR < R6
swxc1.32	SWXC1 fd, index(base)	swxc1.operation	MMAR < R6
sync.32	SYNC stype	Impose the memory-ordering constraints selected by stype.	true
synci.32	SYNCI offset(rs)	synci.operation	MMAR < R6
synci.r6.32	SYNCI offset(rs)	synci.operation	MMAR >= R6
syscall.32	SYSCALL code	Raise a system-call exception.	true
teq.32	TEQ rs, rt, code	Raise Trap when the full-width source GPR values are equal.	true
teqi.32	TEQI rs, imm	teqi.operation	MMAR < R6
tge.32	TGE rs, rt, code	Raise Trap when signed, full-width rs is greater than or equal to rt.	true
tgei.32	TGEI rs, imm	tgei.operation	MMAR < R6
tgeiu.32	TGEIU rs, imm	tgeiu.operation	MMAR < R6
tgeu.32	TGEU rs, rt, code	Raise Trap when unsigned, full-width rs is greater than or equal to rt.	true
tlbinv.32	TLBINV	tlbinv.operation	TLBINV
tlbinvf.32	TLBINVF	tlbinvf.operation	TLBINV
tlbp.32	TLBP	Probe the TLB for the translation selected by EntryHi.	true
tlbr.32	TLBR	tlbr.operation	true
tlbwi.32	TLBWI	Write the current CP0 translation state to the indexed TLB entry.	true
tlbwr.32	TLBWR	tlbwr.operation	true

Format	Assembly	Operation	Availability
tlt.32	TLT <i>rs</i> , <i>rt</i> , <i>code</i>	Raise Trap when signed, full-width <i>rs</i> is less than <i>rt</i> .	true
tlti.32	TLTI <i>rs</i> , <i>imm</i>	tlti.operation	MMAR < R6
tltiu.32	TLTIU <i>rs</i> , <i>imm</i>	tltiu.operation	MMAR < R6
tltu.32	TLTU <i>rs</i> , <i>rt</i> , <i>code</i>	Raise Trap when unsigned, full-width <i>rs</i> is less than <i>rt</i> .	true
tne.32	TNE <i>rs</i> , <i>rt</i> , <i>code</i>	Raise Trap when the full-width source GPR values differ.	true
tnei.32	TNEI <i>rs</i> , <i>imm</i>	tnei.operation	MMAR < R6
trunc.l.d	TRUNC.L.D <i>fd</i> , <i>fs</i>	trunc.l.d.operation	true
trunc.l.s	TRUNC.L.S <i>fd</i> , <i>fs</i>	trunc.l.s.operation	true
trunc.w.d	TRUNC.W.D <i>fd</i> , <i>fs</i>	trunc.w.d.operation	true
trunc.w.s	TRUNC.W.S <i>fd</i> , <i>fs</i>	trunc.w.s.operation	true
wait.mm.32	WAIT <i>code</i>	wait.operation	MMAR >= R6
wrdsp.32	WRDSP <i>rs</i> , <i>mask</i>	wrdsp.operation	DSP
wrpgpr.32	WRPGPR <i>dst</i> , <i>src</i>	Write the previous shadow-set GPR, or copy when no SRS is present.	true
wsbh.32	WSBH <i>rt</i> , <i>rs</i>	Swap bytes within each halfword of a 32-bit GPR word.	true
xor.16	XOR <i>rd</i> , <i>rs</i>	Compute a full-GPR logical XOR and write the result to a GPR.	MMAR < R6
xor.16.r6	XOR <i>rd</i> , <i>rs</i>	Compute a full-GPR logical XOR and write the result to a GPR.	MMAR >= R6
xor.32	XOR <i>rd</i> , <i>rs</i> , <i>rt</i>	Compute a full-GPR logical XOR and write the result to a GPR.	true
xori.32	XORI <i>rt</i> , <i>rs</i> , <i>imm</i>	Compute a full-GPR logical XOR with a zero-extended immediate.	true

1.5. nanoMIPS

Format	Assembly	Operation	Availability
abs.d	ABS.D <i>fd</i> , <i>fs</i>	abs.d.operation	true
abs.s	ABS.S <i>fd</i> , <i>fs</i>	abs.s.operation	true
absq_s.ph.32	ABSQ_S.PH <i>rd</i> , <i>rs</i>	Take the saturating absolute value of two signed halfword lanes.	DSP

Format	Assembly	Operation	Availability
absq_s.qb.32	ABSQ_S.QB rd, rs	Take the saturating absolute value of four signed byte lanes.	DSP2
absq_s.w.32	ABSQ_S.W rd, rs	Take the saturating absolute value of a signed word lane.	DSP
add.32	ADD rd, rs, rt	Add signed 32-bit GPR operands and trap on integer overflow.	!(NMS)
add.d	ADD.D fd, fs, ft	add.d.operation	true
add.s	ADD.S fd, fs, ft	add.s.operation	true
addiu.r1_sp	ADDIU rt, \$sp, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.r2	ADDIU rt, rs, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.gp48	ADDIU rt, \$gp, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	!(NMS)
addiu.gp_b	ADDIU rt, \$gp, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.gp_w	ADDIU rt, \$gp, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.48	ADDIU rt, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	!(NMS)
addiu.rs5	ADDIU rt, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.neg	ADDIU rt, rs, -u	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true
addiu.32	ADDIU rt, rs, imm	Add modulo $2^{**}32$ and sign-extend the 32-bit result to the GPR width.	true

Format	Assembly	Operation	Availability
addiupc.32	ADDIUPC rt, imm	Add a 32-bit displacement to the word-aligned architectural PC. ADDIUPC/ADDIUPCR3 are the 32-bit PC-relative operations. This remains a 32-bit result even when the surrounding processor has 64-bit GPRs; the nanoMIPS P64 decoder selects DADDIUPC instead of reaching this operation.	true
addiupc.48	ADDIUPC rt, imm	Add a 32-bit displacement to the word-aligned architectural PC. ADDIUPC/ADDIUPCR3 are the 32-bit PC-relative operations. This remains a 32-bit result even when the surrounding processor has 64-bit GPRs; the nanoMIPS P64 decoder selects DADDIUPC instead of reaching this operation.	!(NMS)
addq.ph.32	ADDQ.PH rd, rs, rt	Add signed halfword lanes without saturation.	DSP
addq_s.ph.32	ADDQ_S.PH rd, rs, rt	Add signed halfword lanes with signed saturation.	DSP
addq_s.w.32	ADDQ_S.W rd, rs, rt	Add signed word lanes with signed saturation.	DSP
addqh.ph.32	ADDQH.PH rd, rs, rt	Add signed halfword lanes and halve the results.	DSP2
addqh.w.32	ADDQH.W rd, rs, rt	Add signed word lanes and halve the result.	DSP2
addqh_r.ph.32	ADDQH_R.PH rd, rs, rt	Rounded signed half-add of two halfword lanes.	DSP2
addqh_r.w.32	ADDQH_R.W rd, rs, rt	Rounded signed half-add of two word lanes.	DSP2
addsc.32	ADDSC rd, rs, rt	Add unsigned words and write the carry to DSPControl.C.	DSP
addu.16	ADDU rd, rs, rt	Add two 32-bit GPR values and sign-extend the result to GPR width.	true
addu.4x4	ADDU rd, rt	Add two 32-bit GPR values and sign-extend the result to GPR width.	!(NMS)

Format	Assembly	Operation	Availability
addu.32	ADDU rd, rs, rt	Add two 32-bit GPR values and sign-extend the result to GPR width.	true
addu.ph.32	ADDU.PH rd, rs, rt	Add unsigned halfword lanes modulo 16 bits.	DSP2
addu.qb.32	ADDU.QB rd, rs, rt	Add unsigned byte lanes modulo 8 bits and report carry.	DSP
addu_s.ph.32	ADDU_S.PH rd, rs, rt	Add unsigned halfword lanes with saturation and report carry.	DSP2
addu_s.qb.32	ADDU_S.QB rd, rs, rt	Add unsigned byte lanes with saturation and report carry.	DSP
adduh.qb.32	ADDUH.QB rd, rs, rt	Average unsigned byte lanes by an unrounded right shift.	DSP
adduh_r.qb.32	ADDUH_R.QB rd, rs, rt	Average unsigned byte lanes with rounding.	DSP
addwc.32	ADDWC rd, rs, rt	Add signed words with DSPControl.C and report signed overflow.	DSP
aluipc.32	ALUIPC rt, imm	Add the aligned signed displacement to next_pc and clear low bits.	true
and.16	AND rd, rs	Compute a full-GPR logical AND and write the result to a GPR.	true
and.32	AND rd, rs, rt	Compute a full-GPR logical AND and write the result to a GPR.	true
andi.16	ANDI rt, rs, imm	Compute a full-GPR logical AND with a zero-extended immediate.	true
andi.32	ANDI rt, rs, imm	Compute a full-GPR logical AND with a zero-extended immediate.	true
append.32	APPEND rt, rs, sa	append.operation	DSP2
balc.16	BALC offset	Perform a compact PC-relative call without a delay slot.	true
balc.32	BALC offset	Perform a compact PC-relative call without a delay slot.	true
balrsc.32	BALRSC link, rs	Branch to the scaled register-relative address and write the link.	true
bbeqzc.32	BBEQZC bit, offset	Compact-branch when the selected GPR bit is zero.	!(NMS)

Format	Assembly	Operation	Availability
bbnezc.32	BBNEZC bit, offset	Compact-branch when the selected GPR bit is one.	!(NMS)
bc.16	BC offset	Perform an unconditional compact PC-relative branch.	true
bc.32	BC offset	Perform an unconditional compact PC-relative branch.	true
bc1eqzc.32	BC1EQZC ft, offset	bc1eqzc.operation	true
bc1nezc.32	BC1NEZC ft, offset	bc1nezc.operation	true
beqc.16	BEQC rs, rt, offset	Compare two GPRs and perform a compact branch when equal.	!(NMS)
beqc.32	BEQC rs, rt, offset	Compare two GPRs and perform a compact branch when equal.	true
beqic.32	BEQIC immediate, offset	Branch compactly when a GPR equals the immediate.	true
beqzc.16	BEQZC rt, offset	Compact-branch when the compact source GPR is zero.	true
bgec.32	BGEC rt, offset	Branch compactly when signed rs is greater than or equal to rt.	true
bgeic.32	BGEIC immediate, offset	Branch when a signed GPR is greater than or equal to the immediate.	true
bgeiuc.32	BGEIUC immediate, offset	Branch when an unsigned GPR is greater than or equal to the immediate.	true
bgeuc.32	BGEUC rt, offset	Branch compactly when unsigned rs is greater than or equal to rt.	true
bgezc.32	BGEZC rt, offset	Compact-branch when the full-width source GPR is signed non-negative.	true
bitrev.32	BITREV rd, rt	bitrev.operation	DSP
bltc.32	BLTC rs, rt, offset	Branch compactly when signed rs is less than signed rt.	true
bltic.32	BLTIC immediate, offset	Branch compactly when a signed GPR is less than the immediate.	true
bltiuc.32	BLTIUC immediate, offset	Branch compactly when an unsigned GPR is less than the immediate.	true
bltuc.32	BLTUC rt, offset	Branch compactly when unsigned rs is less than rt.	true

Format	Assembly	Operation	Availability
bne _c .16	BNEC <i>rs</i> , <i>rt</i> , <i>offset</i>	Compare two GPRs and perform a compact branch when they differ.	!(NMS)
bne _c .32	BNEC <i>rs</i> , <i>rt</i> , <i>offset</i>	Compare two GPRs and perform a compact branch when they differ.	true
bneic.32	BNEIC <i>immediate</i> , <i>offset</i>	Branch compactly when a GPR does not equal the immediate.	true
bnez _c .16	BNEZC <i>rt</i> , <i>offset</i>	Compact-branch when the compact source GPR is nonzero.	true
bposge32c.no.32	BPOSGE32C <i>offset</i>	bposge32c.operation	DSP && DSP2
break.32	BREAK <i>code</i>	Raise a breakpoint exception.	true
break.16	BREAK <i>code</i>	Raise a breakpoint exception.	true
brsc.32	BRSC <i>rs</i>	Branch to the following-instruction address plus twice <i>rs</i> .	true
cache.eva	CACHE <i>op</i> , <i>offset(rs)</i>	Issue an implementation-defined cache operation through the backend.	EVA
cache.s9	CACHE <i>op</i> , <i>offset(rs)</i>	Issue an implementation-defined cache operation through the backend.	true
ceil.l.d	CEIL.L.D <i>fd</i> , <i>fs</i>	ceil.l.d.operation	true
ceil.l.s	CEIL.L.S <i>fd</i> , <i>fs</i>	ceil.l.s.operation	true
ceil.w.d	CEIL.W.D <i>fd</i> , <i>fs</i>	ceil.w.d.operation	true
ceil.w.s	CEIL.W.S <i>fd</i> , <i>fs</i>	ceil.w.s.operation	true
cfc1.32	CFC1 <i>rt</i> , <i>cs</i>	cfc1.operation	true
class.d	CLASS.D <i>fd</i> , <i>fs</i>	class.d.operation	true
class.s	CLASS.S <i>fd</i> , <i>fs</i>	class.s.operation	true
clo.32	CLO <i>rt</i> , <i>rs</i>	Count leading one bits in the low 32 bits of <i>rs</i> and write the count.	!(NMS)
clz.32	CLZ <i>rt</i> , <i>rs</i>	Count leading zeroes in the low 32 bits and write the count to <i>rt</i> .	!(NMS)
cmp.d	CMP.D <i>fd</i> , <i>fs</i> , <i>ft</i> , <i>condition</i>	cmp.d.operation	true
cmp.eq.ph.32	CMP.EQ.PH <i>rs</i> , <i>rt</i>	Compare signed halfword lanes for equality and write Ccond.	DSP
cmp.le.ph.32	CMP.LE.PH <i>rs</i> , <i>rt</i>	Compare signed halfword lanes for less-than-or-equal and write Ccond.	DSP

Format	Assembly	Operation	Availability
cmp.lt.ph.32	CMP.LT.PH rs, rt	Compare signed halfword lanes for less-than and write Ccond.	DSP
cmp.s	CMP.S fd, fs, ft, condition	cmp.s.operation	true
cmpgdu.eq.qb.32	CMPGDU.EQ.QB rd, rs, rt	Compare unsigned byte lanes, writing both Ccond and a GPR mask.	DSP2
cmpgdu.le.qb.32	CMPGDU.LE.QB rd, rs, rt	Compare unsigned byte lanes for $\leq$, writing Ccond and a GPR mask.	DSP2
cmpgdu.lt.qb.32	CMPGDU.LT.QB rd, rs, rt	Compare unsigned byte lanes for less-than, writing Ccond and a GPR mask.	DSP2
cmpgu.eq.qb.32	CMPGU.EQ.QB rd, rs, rt	Compare unsigned byte lanes for equality and write a GPR mask.	DSP
cmpgu.le.qb.32	CMPGU.LE.QB rd, rs, rt	Compare unsigned byte lanes for less-than-or-equal and write a GPR mask.	DSP
cmpgu.lt.qb.32	CMPGU.LT.QB rd, rs, rt	Compare unsigned byte lanes for less-than and write a GPR mask.	DSP
cmpe.eq.qb.32	CMPE.EQ.QB rs, rt	Compare unsigned byte lanes for equality and write Ccond.	DSP
cmpe.le.qb.32	CMPE.LE.QB rs, rt	Compare unsigned byte lanes for less-than-or-equal and write Ccond.	DSP
cmpe.lt.qb.32	CMPE.LT.QB rs, rt	Compare unsigned byte lanes for less-than and write Ccond.	DSP
crc32b.32	CRC32B rt, rs	Update the CRC-32 value with the low byte of rs.	CRCP
crc32cb.32	CRC32CB rt, rs	Update the CRC-32C value with the low byte of rs.	CRCP
crc32cd.32	CRC32CD rt, rs	crc32cd.operation	AT $\geq$ MIPS64_32 && CRCP
crc32ch.32	CRC32CH rt, rs	Update the CRC-32C value with the low halfword of rs.	CRCP
crc32cw.32	CRC32CW rt, rs	Update the CRC-32C value with the low word of rs.	CRCP
crc32d.32	CRC32D rt, rs	crc32d.operation	AT $\geq$ MIPS64_32 && CRCP
crc32h.32	CRC32H rt, rs	Update the CRC-32 value with the low halfword of rs.	CRCP

Format	Assembly	Operation	Availability
crc32w.32	CRC32W rt, rs	Update the CRC-32 value with the low word of rs.	CRCP
ctc1.32	CTC1 rt, cs	ctc1.operation	true
cvt.d.l	CVT.D.L fd, fs	cvt.d.l.operation	true
cvt.d.s	CVT.D.S fd, fs	cvt.d.s.operation	true
cvt.d.w	CVT.D.W fd, fs	cvt.d.w.operation	true
cvt.l.d	CVT.L.D fd, fs	cvt.l.d.operation	true
cvt.l.s	CVT.L.S fd, fs	cvt.l.s.operation	true
cvt.s.d	CVT.S.D fd, fs	cvt.s.d.operation	true
cvt.s.l	CVT.S.L fd, fs	cvt.s.l.operation	true
cvt.s.w	CVT.S.W fd, fs	cvt.s.w.operation	true
cvt.w.d	CVT.W.D fd, fs	cvt.w.d.operation	true
cvt.w.s	CVT.W.S fd, fs	cvt.w.s.operation	true
dadd.32	DADD rd, rs, rt	dadd.operation	AT >= MIPS64_32
daddiu.r1_sp	DADDIU rd, \$sp, imm	daddiu.operation	AT >= MIPS64_32
daddiu.gp48	DADDIU rd, \$gp, imm	daddiu.operation	AT >= MIPS64_32 && !(NMS)
daddiu.gp_b	DADDIU rd, \$gp, imm	daddiu.operation	AT >= MIPS64_32
daddiu.gp_w	DADDIU rd, \$gp, imm	daddiu.operation	AT >= MIPS64_32
daddiu.48	DADDIU rd, imm	daddiu.operation	AT >= MIPS64_32 && !(NMS)
daddiu.neg	DADDIU rd, rs, -u	daddiu.operation	AT >= MIPS64_32
daddiu.u12	DADDIU rd, rs, imm	daddiu.operation	AT >= MIPS64_32
daddiupc.32	DADDIUPC rt, imm	Add a signed displacement to the architectural sequential PC.	AT >= MIPS64_32
daddiupc.48	DADDIUPC rt, imm	Add a signed displacement to the architectural sequential PC.	AT >= MIPS64_32 && !(NMS)
daddu.32	DADDU rd, rs, rt	daddu.operation	AT >= MIPS64_32
dclo.32	DCLO rt, rs	dclo.operation	AT >= MIPS64_32
dclz.32	DCLZ rt, rs	dclz.operation	AT >= MIPS64_32
ddiv.32	DDIV rd, rs, rt	ddiv.operation	AT >= MIPS64_32
ddivu.32	DDIVU rd, rs, rt	ddivu.operation	AT >= MIPS64_32
deret.32	DERET	deret.operation	true
dext.32	DEXT rt, rs, pos, size	dext.operation	AT >= MIPS64_32
dextm.32	DEXTM rt, rs, pos, size	dextm.operation	AT >= MIPS64_32

Format	Assembly	Operation	Availability
dextu.32	DEXTU rt, rs, pos, size	dextu.operation	AT >= MIPS64_32
di.32	DI rt	di.operation	true
dins.32	DINS rt, rs, pos, size	dins.operation	AT >= MIPS64_32
dinsm.32	DINSM rt, rs, pos, size	dinsm.operation	AT >= MIPS64_32
dinsu.32	DINSU rt, rs, pos, size	dinsu.operation	AT >= MIPS64_32
div.32	DIV rd, rs, rt	Divide signed low 32-bit GPR operands and write a sign-extended quotient.	true
div.d	DIV.D fd, fs, ft	div.d.operation	true
div.s	DIV.S fd, fs, ft	div.s.operation	true
divu.32	DIVU rd, rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended quotient.	true
dlsa.32	DLSA rd, rs, rt, shift	dlsa.operation	AT >= MIPS64_32
dlui.48	DLUI rt, imm	dlui.operation	AT >= MIPS64_32 && !(NMS)
dmfc0.32	DMFC0 rt, c0s, sel	dmfc0.operation	AT >= MIPS64_32
dmfc1.32	DMFC1 rt, fs	dmfc1.operation	AT >= MIPS64_32
dmod.32	DMOD rd, rs, rt	dmod.operation	AT >= MIPS64_32
dmodu.32	DMODU rd, rs, rt	dmodu.operation	AT >= MIPS64_32
dmtc0.32	DMTC0 rt, c0s, sel	dmtc0.operation	AT >= MIPS64_32
dmtc1.32	DMTC1 rt, fs	dmtc1.operation	AT >= MIPS64_32
dmuh.32	DMUH rd, rs, rt	dmuh.operation	AT >= MIPS64_32
dmuhu.32	DMUHU rd, rs, rt	dmuhu.operation	AT >= MIPS64_32
dmul.32	DMUL rd, rs, rt	dmul.operation	AT >= MIPS64_32
dmulu.32	DMULU rd, rs, rt	dmulu.operation	AT >= MIPS64_32
dpa.w.ph.32	DPA.W.PH ac, rs, rt	dpa.w.ph.operation	DSP2
dpaq_s.w.ph.32	DPAQ_S.W.PH ac, rs, rt	dpaq_s.w.ph.operation	DSP
dpaq_sa.l.w.32	DPAQ_SA.L.W ac, rs, rt	dpaq_s.l.w.operation	DSP
dpaqx_s.w.ph.32	DPAQX_S.W.PH ac, rs, rt	dpaqx_s.w.ph.operation	DSP
dpaqx_sa.w.ph.32	DPAQX_SA.W.PH ac, rs, rt	dpaqx_sa.w.ph.operation	DSP
dpau.h.qbl.32	DPAU.H.QBL ac, rs, rt	dpau.h.qbl.operation	DSP

Format	Assembly	Operation	Availability
dpau.h.qbr.32	DPAU.H.QBR ac, rs, rt	dpau.h.qbr.operation	DSP
dpax.w.ph.32	DPAX.W.PH ac, rs, rt	dpax.w.ph.operation	DSP
dps.w.ph.32	DPS.W.PH ac, rs, rt	dps.w.ph.operation	DSP2
dpsq_s.w.ph.32	DPSQ_S.W.PH ac, rs, rt	dpsq_s.w.ph.operation	DSP
dpsq_sa.l.w.32	DPSQ_SA.L.W ac, rs, rt	dpsq_s.l.w.operation	DSP
dpsqx_s.w.ph.32	DPSQX_S.W.PH ac, rs, rt	dpsqx_s.w.ph.operation	DSP
dpsqx_sa.w.ph.32	DPSQX_SA.W.PH ac, rs, rt	dpsqx_sa.w.ph.operation	DSP
dpsu.h.qbl.32	DPSU.H.QBL ac, rs, rt	dpsu.h.qbl.operation	DSP
dpsu.h.qbr.32	DPSU.H.QBR ac, rs, rt	dpsu.h.qbr.operation	DSP
dpsx.w.ph.32	DPSX.W.PH ac, rs, rt	dpsx.w.ph.operation	DSP
drotr.32	DROTR rd, rs, shift	drotr.operation	AT >= MIPS64_32
drotr32.32	DROTR32 rd, rs, shift	drotr32.operation	AT >= MIPS64_32
drotrv.32	DROTRV rd, rs, rt	drotrv.operation	AT >= MIPS64_32
drotx.32	DROTX rt, rs, shift, shiftx, stripe, flip	drotx.operation	AT >= MIPS64_32
dsll.32	DSLL rd, rs, shift	dsll.operation	AT >= MIPS64_32
dsll32.32	DSLL32 rd, rs, shift	dsll32.operation	AT >= MIPS64_32
dsllv.32	DSLLV rd, rs, rt	dsllv.operation	AT >= MIPS64_32
dsra.32	DSRA rd, rs, shift	dsra.operation	AT >= MIPS64_32
dsra32.32	DSRA32 rd, rs, shift	dsra32.operation	AT >= MIPS64_32
dsrav.32	DSRAV rd, rs, rt	dsrav.operation	AT >= MIPS64_32
dsrl.32	DSRL rd, rs, shift	dsrl.operation	AT >= MIPS64_32
dsrl32.32	DSRL32 rd, rs, shift	dsrl32.operation	AT >= MIPS64_32
dsrlv.32	DSRLV rd, rs, rt	dsrlv.operation	AT >= MIPS64_32
dsub.32	DSUB rd, rs, rt	dsub.operation	AT >= MIPS64_32
dsubu.32	DSUBU rd, rs, rt	dsubu.operation	AT >= MIPS64_32
dvp.32	DVP rt	dvp.operation	true
ehb.32	EHB	ehb.operation	true
ei.32	EI rt	ei.operation	true
eret.32	ERET	Return from an exception and clear the LL reservation.	true

Format	Assembly	Operation	Availability
eretnc.32	ERETNC	Return from an exception without clearing the LL reservation.	true
evp.32	EVP rt	evp.operation	true
ext.32	EXT rt, rs, pos, size	Extract a 32-bit field and sign-extend the result to the GPR width.	!(NMS)
extd.32	EXTD rd, rs, rt, shift	extd.operation	AT >= MIPS64_32
extd32.32	EXTD rd, rs, rt, shift	extd.operation	AT >= MIPS64_32
extp.32	EXTP rt, ac, size	extp.operation	DSP
extpdp.32	EXTPDP rt, ac, size	extpdp.operation	DSP
extpdpv.32	EXTPDPV rt, ac, rs	extpdpv.operation	DSP
extpv.32	EXTPV rt, ac, rs	extpv.operation	DSP
extr.w.32	EXTR.W rt, ac, shift	extr.w.operation	DSP
extr_r.w.32	EXTR_R.W rt, ac, shift	extr_r.w.operation	DSP
extr_rs.w.32	EXTR_RS.W rt, ac, shift	extr_rs.w.operation	DSP
extr_s.h.32	EXTR_S.H rt, ac, shift	extr_s.h.operation	DSP
extrv.w.32	EXTRV.W rt, ac, rs	extrv.w.operation	DSP
extrv_r.w.32	EXTRV_R.W rt, ac, rs	extrv_r.w.operation	DSP
extrv_rs.w.32	EXTRV_RS.W rt, ac, rs	extrv_rs.w.operation	DSP
extrv_s.h.32	EXTRV_S.H rt, ac, rs	extrv_s.h.operation	DSP
extw.32	EXTW rd, rs, rt, shift	Extract a shifted 32-bit word from the rt:rs concatenation.	true
floor.l.d	FLOOR.L.D fd, fs	floor.l.d.operation	true
floor.l.s	FLOOR.L.S fd, fs	floor.l.s.operation	true
floor.w.d	FLOOR.W.D fd, fs	floor.w.d.operation	true
floor.w.s	FLOOR.W.S fd, fs	floor.w.s.operation	true
ginvi.32	GINVI rs	ginvi.operation	true
ginv_t.32	GINVT rs, type	ginvt.operation	true
ins.32	INS rt, rs, pos, size	Merge a right-justified source field into a 32-bit destination word.	!(NMS)
insv.32	INSV rt, rs	insv.operation	DSP
jalrc.32	JALRC dst, src	Jump to a GPR address and save the prepared sequential next PC.	true
jalrc.hb.32	JALRC dst, src	Jump to a GPR address and save the prepared sequential next PC.	true

Format	Assembly	Operation	Availability
jalrc.16	JALRC src	Jump to a GPR address and save the prepared sequential next PC.	true
jrc.16	JRC rt	Jump immediately to the address held in the source GPR.	true
lb.16	LB rt, offset(rs)	Load one byte and sign-extend it to the implemented GPR width.	true
lb.s9	LB rt, offset(rs)	Load one byte and sign-extend it to the implemented GPR width.	true
lb.u12	LB rt, offset(rs)	Load one byte and sign-extend it to the implemented GPR width.	true
lb.gp	LB rt, offset(gp)	Load one byte and sign-extend it to the implemented GPR width.	true
lbe.32	LBE offset(rs)	Load a signed byte through EVA translation.	EVA
lbu.16	LBU rt, offset(rs)	Load one byte and zero-extend it to the implemented GPR width.	true
lbu.s9	LBU rt, offset(rs)	Load one byte and zero-extend it to the implemented GPR width.	true
lbu.u12	LBU rt, offset(rs)	Load one byte and zero-extend it to the implemented GPR width.	true
lbu.gp	LBU rt, offset(gp)	Load one byte and zero-extend it to the implemented GPR width.	true
lbue.32	LBUE offset(rs)	Load an unsigned byte through EVA translation.	EVA
lbux.32	LBUX rs(rt)	lbux.operation	true
lbx.32	LBX rs(rt)	lbx.operation	true
ld.s9	LD rt, offset(rs)	ld.operation	AT >= MIPS64_32
ld.u12	LD rt, offset(rs)	ld.operation	AT >= MIPS64_32
ld.gp	LD rt, offset(gp)	ld.operation	AT >= MIPS64_32
ldc1.gp	LDC1 ft, ...	ldc1.operation	true
ldc1.s9	LDC1 ft, ...	ldc1.operation	true
ldc1.u12	LDC1 ft, ...	ldc1.operation	true
ldm.32	LDM rt, offset(rs), count	ldm.operation	AT >= MIPS64_32 && !(NMS)
ldpc.48	LDPC rt, offset	ldpc.operation	AT >= MIPS64_32 && !(NMS)
ldx	LDX rs(rt)	ldx.operation	AT >= MIPS64_32

Format	Assembly	Operation	Availability
ldxs	LDXS rs(rt)	ldxs.operation	AT >= MIPS64_32
lh.16	LH rt, offset(rs)	Load a signed halfword from memory and extend it to GPR width.	true
lh.s9	LH rt, offset(rs)	Load a signed halfword from memory and extend it to GPR width.	true
lh.u12	LH rt, offset(rs)	Load a signed halfword from memory and extend it to GPR width.	true
lh.gp	LH rt, offset(\$gp)	Load a signed halfword from memory and extend it to GPR width.	true
lhe.32	LHE offset(rs)	Load a signed halfword through EVA translation.	EVA
lhu.16	LHU rt, offset(rs)	Load an unsigned halfword from memory and extend it to GPR width.	true
lhu.s9	LHU rt, offset(rs)	Load an unsigned halfword from memory and extend it to GPR width.	true
lhu.u12	LHU rt, offset(rs)	Load an unsigned halfword from memory and extend it to GPR width.	true
lhu.gp	LHU rt, offset(\$gp)	Load an unsigned halfword from memory and extend it to GPR width.	true
lhue.32	LHUE offset(rs)	Load an unsigned halfword through EVA translation.	EVA
lhux.32	LHUX rs(rt)	lhux.operation	true
lhuxs.32	LHUXS rs(rt)	lhuxs.operation	true
lhx.32	LHX rs(rt)	lhx.operation	true
lhxs.32	LHXS rs(rt)	lhxs.operation	true
li.16	LI rt, imm	Write the decoded immediate to the destination GPR.	true
li.48	LI rt, imm	Write the decoded immediate to the destination GPR.	!(NMS)
ll.32	LL rt, offset(rs)	Load a sign-extended word and establish a linked reservation.	true

Format	Assembly	Operation	Availability
lld.32	LLD rt, offset(rs)	lld.operation	AT >= MIPS64_32
lldp.32	LLDP rt, ru, (rs)	lldp.operation	AT >= MIPS64_32
lle.32	LLE rt, offset(rs)	Load an EVA sign-extended word and establish a linked reservation.	EVA
llwp.32	LLWP rt, ru, (rs)	Load a linked pair of sign-extended words.	!(NMS)
llwpe.32	LLWPE rt, ru, (rs)	Load an EVA linked pair of sign-extended words.	EVA
lsa.32	LSA rd, rs, rt, shift	Add a scaled 32-bit GPR value to another and sign-extend the result.	true
lui.32	LUI rt, imm	Load the sign-extended 32-bit upper-immediate value into a GPR.	true
lw.16	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.gp16	LW rt, offset(gp)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.4x4	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	!(NMS)
lw.s9	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.u12	LW rt, offset(rs)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.gp	LW rt, offset(gp)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lw.sp	LW rt, offset(sp)	Load a word from rs plus offset and sign-extend it to GPR width.	true
lwc1.gp	LWC1 ft, ...	lwc1.operation	true
lwc1.s9	LWC1 ft, ...	lwc1.operation	true
lwc1.u12	LWC1 ft, ...	lwc1.operation	true
lwe.32	LWE offset(rs)	Load a signed word through EVA translation.	EVA
lwm.32	LWM offset(rs), count	Load consecutive aligned words into sequential GPRs.	!(NMS)
lwpc.48	LWPC rt, offset	Load a signed word using the active architecture's PC-relative form.	!(NMS)

Format	Assembly	Operation	Availability
lwu.s9	LWU rt, offset(rs)	lwu.operation	AT >= MIPS64_32
lwu.u12	LWU rt, offset(rs)	lwu.operation	AT >= MIPS64_32
lwu.gp	LWU rt, offset(gp)	lwu.operation	AT >= MIPS64_32
lwux	LWUX rs(rt)	lwux.operation	AT >= MIPS64_32
lwuxs	LWUXS rs(rt)	lwuxs.operation	AT >= MIPS64_32
lwx.32	LWX rs(rt)	lwx.operation	true
lwxs.16	LWXS rd, rs(rt)	lwxs.operation	true
lwxs.32	LWXS rs(rt)	lwxs.operation	true
madd.dsp.r7	MADD ac, rs, rt	madd.operation	DSP
maddf.d	MADDF.D fd, fs, ft	maddf.d.operation	true
maddf.s	MADDF.S fd, fs, ft	maddf.s.operation	true
maddu.dsp.r7	MADDU ac, rs, rt	maddu.operation	DSP
maq_s.w.phl.32	MAQ_S.W.PHL ac, rs, rt	maq_s.w.phl.operation	DSP
maq_s.w.phr.32	MAQ_S.W.PHR ac, rs, rt	maq_s.w.phr.operation	DSP
maq_sa.w.phl.32	MAQ_SA.W.PHL ac, rs, rt	maq_sa.w.phl.operation	DSP
maq_sa.w.phr.32	MAQ_SA.W.PHR ac, rs, rt	maq_sa.w.phr.operation	DSP
max.d	MAX.D fd, fs, ft	max.d.operation	true
max.s	MAX.S fd, fs, ft	max.s.operation	true
maxa.d	MAXA.D fd, fs, ft	maxa.d.operation	true
maxa.s	MAXA.S fd, fs, ft	maxa.s.operation	true
mfc0.32	MFC0 rt, c0s, sel	Read a CP0 register and sign-extend its low 32-bit result to GPR width.	true
mfc1.32	MFC1 rt, fs	Copy the low word of an FPR to a GPR after checking CP1 access.	true
mfhc0.32	MFHC0 rt, c0s, sel	mfhc0.operation	true
mfhc1.32	MFHC1 rt, fs	mfhc1.operation	true
mfhi.dsp.r7	MFHI rd, ac	mfhi.operation	DSP
mflo.dsp.r7	MFLO rd, ac	mflo.operation	DSP
min.d	MIN.D fd, fs, ft	min.d.operation	true
min.s	MIN.S fd, fs, ft	min.s.operation	true
mina.d	MINA.D fd, fs, ft	mina.d.operation	true

Format	Assembly	Operation	Availability
mina.s	MINA.S fd, fs, ft	mina.s.operation	true
mod.32	MOD rd, rs, rt	Divide signed low 32-bit GPR operands and write a sign-extended remainder.	true
modsub.32	MODSUB rd, rs, rt	Subtract a decrement, wrapping to the supplied last index at zero.	DSP
modu.32	MODU rd, rs, rt	Divide zero-extended 32-bit GPR operands and write a sign-extended remainder.	true
mov.d	MOV.D fd, fs	mov.d.operation	true
mov.s	MOV.S fd, fs	mov.s.operation	true
move.16	MOVE rt, rs	Copy the complete implemented-width value from rs to rt.	true
move.balc.32	MOVE.BALC rd, rt, offset	Copy rt, then perform an unconditional compact PC-relative call.	!(NMS)
movep.16	MOVEP dst1, dst2, src1, src2	Copy a pair of registers; overlapping pairs are architecturally unknown.	!(NMS)
movep.rev	MOVEP dst1, dst2, src1, src2	Copy a pair of registers; overlapping pairs are architecturally unknown.	!(NMS)
movn.32	MOVN rd, rs, rt	Copy rs to rd when the full-width value in rt is nonzero.	true
movz.32	MOVZ rd, rs, rt	Copy rs to rd when the full-width value in rt is zero.	true
msub.dsp.r7	MSUB ac, rs, rt	msub.operation	DSP
msubf.d	MSUBF.D fd, fs, ft	msubf.d.operation	true
msubf.s	MSUBF.S fd, fs, ft	msubf.s.operation	true
msubu.dsp.r7	MSUBU ac, rs, rt	msubu.operation	DSP
mtc0.32	MTC0 rt, c0s, sel	Write a GPR to an encoded CP0 register. On a 64-bit MIPS core the MTC0 data path is the full GPR width; the MFC0/DMFC0 pair then exposes the low-word and full-width read views. A 32-bit core retains the ordinary 32-bit path.	true
mtc1.32	MTC1 rt, fs	mtc1.operation	true
mthc0.32	MTHC0 rt, c0s, sel	mthc0.operation	true

Format	Assembly	Operation	Availability
mthc1.32	MTHC1 rt, fs	mthc1.operation	true
mthi.dsp.r7	MTHI rs, ac	mthi.operation	DSP
mthlip.32	MTHLIP rs, ac	mthlip.operation	DSP
mtlo.dsp.r7	MTLO rs, ac	mtlo.operation	DSP
muh.32	MUH rd, rs, rt	Multiply signed words and write the sign-extended high word.	true
muhu.32	MUHU rd, rs, rt	Multiply unsigned words and write the high word.	true
mul.4x4	MUL rd, rt, rs	Multiply signed 32-bit GPR operands and sign-extend the low word.	!(NMS)
mul.32	MUL rd, rs, rt	Multiply signed 32-bit GPR operands and sign-extend the low word.	true
mul.d	MUL.D fd, fs, ft	mul.d.operation	true
mul.ph.32	MUL.PH rd, rs, rt	mul.ph.operation	DSP
mul.s	MUL.S fd, fs, ft	mul.s.operation	true
mul_s.ph.32	MUL_S.PH rd, rs, rt	mul_s.ph.operation	DSP
muleq_s.w.ph l.32	MULEQ_S.W.PHL rd, rs, rt	muleq_s.w.phl.operation	DSP
muleq_s.w.ph r.32	MULEQ_S.W.PHR rd, rs, rt	muleq_s.w.phr.operation	DSP
muleu_s.ph.q bl.32	MULEU_S.PH.QBL rd, rs, rt	muleu_s.ph.qbl.operation	DSP
muleu_s.ph.q br.32	MULEU_S.PH.QBR rd, rs, rt	muleu_s.ph.qbr.operation	DSP
mulq_rs.ph.3 2	MULQ_RS.PH rd, rs, rt	mulq_rs.ph.operation	DSP
mulq_rs.w.32	MULQ_RS.W rd, rs, rt	mulq_rs.w.operation	DSP2
mulq_s.ph.32	MULQ_S.PH rd, rs, rt	mulq_s.ph.operation	DSP2
mulq_s.w.32	MULQ_S.W rd, rs, rt	mulq_s.w.operation	DSP2
mulsa.w.ph.3 2	MULSA.W.PH ac, rs, rt	mulsa.w.ph.operation	DSP2
mulsaq_s.w.p h.32	MULSAQ_S.W.PH ac, rs, rt	mulsaq_s.w.ph.operation	DSP
mult.dsp.r7	MULT ac, rs, rt	mult.operation	DSP
multu.dsp.r7	MULTU ac, rs, rt	multu.operation	DSP
mulu.32	MULU rd, rs, rt	Multiply unsigned words and write the sign-extended low word.	true

Format	Assembly	Operation	Availability
neg.d	NEG.D fd, fs	neg.d.operation	true
neg.s	NEG.S fd, fs	neg.s.operation	true
nop.16	NOP	Perform no architectural operation.	true
nop.32	NOP	Perform no architectural operation.	true
nor.32	NOR rd, rs, rt	Compute a bitwise NOR at the implemented GPR width.	true
not.16	NOT rt, rs	Invert the complete implemented-width source GPR.	true
or.16	OR rd, rs	Compute a full-GPR logical OR and write the result to a GPR.	true
or.32	OR rd, rs, rt	Compute a full-GPR logical OR and write the result to a GPR.	true
ori.32	ORI rt, rs, imm	Compute a full-GPR logical OR with a zero-extended immediate.	true
packrl.ph.32	PACKRL.PH rd, rs, rt	packrl.ph.operation	DSP
pause.32	PAUSE	Pause instruction issue while CP0 LLAddr.LLB remains set.	true
pick.ph.32	PICK.PH rd, rs, rt	Select halfword lanes from rs or rt according to Ccond.	DSP
pick.qb.32	PICK.QB rd, rs, rt	Select byte lanes from rs or rt according to Ccond.	DSP
preceq.w.phl.32	PRECEQ.W.PHL rd, rt	preceq.w.phl.operation	DSP
preceq.w.phr.32	PRECEQ.W.PHR rd, rt	preceq.w.phr.operation	DSP
precequ.ph.qbl.32	PRECEQU.PH.QBL rd, rt	precequ.ph.qbl.operation	DSP
precequ.ph.qbla.32	PRECEQU.PH.QBLA rd, rt	precequ.ph.qbla.operation	DSP
precequ.ph.qbr.32	PRECEQU.PH.QBR rd, rt	precequ.ph.qbr.operation	DSP
precequ.ph.qbra.32	PRECEQU.PH.QBRA rd, rt	precequ.ph.qbra.operation	DSP
preceu.ph.qbl.32	PRECEU.PH.QBL rd, rt	preceu.ph.qbl.operation	DSP
preceu.ph.qbla.32	PRECEU.PH.QBLA rd, rt	preceu.ph.qbla.operation	DSP
preceu.ph.qbr.32	PRECEU.PH.QBR rd, rt	preceu.ph.qbr.operation	DSP

Format	Assembly	Operation	Availability
preceu.ph.qbra.32	PRECEU.PH.QBRA rd, rt	preceu.ph.qbra.operation	DSP
precr.qb.ph.32	PRECR.QB.PH rd, rs, rt	precr.qb.ph.operation	DSP2
precr_sra.ph.w.32	PRECR_SRA.PH.W rt, rs, sa	precr_sra.ph.w.operation	DSP2
precr_sra_r.ph.w.32	PRECR_SRA_R.PH.W rt, rs, sa	precr_sra_r.ph.w.operation	DSP2
precrq.ph.w.32	PRECRQ.PH.W rd, rs, rt	precrq.ph.w.operation	DSP
precrq.qb.ph.32	PRECRQ.QB.PH rd, rs, rt	precrq.qb.ph.operation	DSP
precrq_rs.ph.w.32	PRECRQ_RS.PH.W rd, rs, rt	precrq_rs.ph.w.operation	DSP
precrqu_s.qb.ph.32	PRECRQU_S.QB.PH rd, rs, rt	precrqu_s.qb.ph.operation	DSP
pref.eva	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	EVA
pref.s9	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	true
pref.u12	PREF hint, offset(rs)	Issue an implementation-defined prefetch without changing data state.	true
raddu.w.qb.32	RADDU.W.QB rd, rs	Reduce four unsigned bytes to a zero-extended ten-bit word.	DSP
rddsp.32	RDDSP rd, mask	rddsp.operation	DSP
rdhwr.32	RDHWR rt, hs, sel	rdhwr.operation	!(NMS)
rdpgpr.32	RDPGPR dst, src	Read a GPR from the previous shadow set into the current set.	true
recip.d	RECIP.D fd, fs	recip.d.operation	true
recip.s	RECIP.S fd, fs	recip.s.operation	true
repl.ph.32	REPL.PH rd, immediate	repl.ph.operation	DSP
repl.qb.32	REPL.QB rd, immediate	repl.qb.operation	DSP
replv.ph.32	REPLV.PH rd, rt	replv.ph.operation	DSP
replv.qb.32	REPLV.QB rd, rt	replv.qb.operation	DSP
restore.32	RESTORE rt, ...]	Restore a consecutive register list and increment the stack pointer.	true

Format	Assembly	Operation	Availability
restore.jrc.16	RESTORE.JRC rt, ...]	Restore registers, increment the stack pointer, and jump through \$31.	true
restore.jrc.32	RESTORE.JRC rt, ...]	Restore registers, increment the stack pointer, and jump through \$31.	true
rint.d	RINT.D fd, fs	rint.d.operation	true
rint.s	RINT.S fd, fs	rint.s.operation	true
rotr.32	ROTR rt, rs, shift	Rotate the low 32-bit source word right and sign-extend the result.	true
rotrv.32	ROTRV rd, rs, rt	Rotate a 32-bit GPR word right by a GPR-supplied amount.	!(NMS)
rotx.32	ROTX rt, rs, shift, shiftx, stripe	Rotate and exchange bits according to the architectural five-stage mux.	!(NMS)
round.l.d	ROUND.L.D fd, fs	round.l.d.operation	true
round.l.s	ROUND.L.S fd, fs	round.l.s.operation	true
round.w.d	ROUND.W.D fd, fs	round.w.d.operation	true
round.w.s	ROUND.W.S fd, fs	round.w.s.operation	true
rsqrt.d	RSQRT.D fd, fs	rsqrt.d.operation	true
rsqrt.s	RSQRT.S fd, fs	rsqrt.s.operation	true
save.16	SAVE rt, ...]	Save a consecutive register list and decrement the stack pointer.	true
save.32	SAVE rt, ...]	Save a consecutive register list and decrement the stack pointer.	true
sb.16	SB rt, offset(rs)	Store the low byte of rt at the effective address rs plus offset.	true
sb.s9	SB rt, offset(rs)	Store the low byte of rt at the effective address rs plus offset.	true
sb.u12	SB rt, offset(rs)	Store the low byte of rt at the effective address rs plus offset.	true
sb.gp	SB rt, offset(gp)	Store the low byte of rt at the effective address rs plus offset.	true
sbe.32	SBE offset(rs)	Store a byte through EVA translation.	EVA
sbx.32	SBX rs(rt)	sbx.operation	!(NMS)
sc.32	SC rt, offset(rs)	Conditionally store a word and return success in rt.	true

Format	Assembly	Operation	Availability
scd.32	SCD rt, offset(rs)	scd.operation	AT >= MIPS64_32
scdp.32	SCDP rt, ru, (rs)	scdp.operation	AT >= MIPS64_32
sce.32	SCE rt, offset(rs)	Conditionally store an EVA word and return success in rt.	EVA
scwp.32	SCWP rt, ru, (rs)	Conditionally store a word pair and return success in rt.	!(NMS)
scwpe.32	SCWPE rt, ru, (rs)	Conditionally store an EVA word pair and return success in rt.	EVA
sd.s9	SD rt, offset(rs)	sd.operation	AT >= MIPS64_32
sd.u12	SD rt, offset(rs)	sd.operation	AT >= MIPS64_32
sd.gp	SD rt, offset(gp)	sd.operation	AT >= MIPS64_32
sdbbp.32	SDBBP code	Raise a debug breakpoint and set CP0.Debug.DBp when debug is present.	DEBUG
sdbbp.16	SDBBP code	Raise a debug breakpoint and set CP0.Debug.DBp when debug is present.	DEBUG
sdc1.gp	SDC1 ft, ...	sdc1.operation	true
sdc1.s9	SDC1 ft, ...	sdc1.operation	true
sdc1.u12	SDC1 ft, ...	sdc1.operation	true
sdm.32	SDM rt, offset(rs), count	sdm.operation	AT >= MIPS64_32 && !(NMS)
sdpc.48	SDPC rt, offset	sdpc.operation	AT >= MIPS64_32 && !(NMS)
sdx	SDX rs(rt)	sdx.operation	AT >= MIPS64_32
sdxs	SDXS rs(rt)	sdxs.operation	AT >= MIPS64_32
seb.32	SEB rt, rs	Sign-extend the low byte of rs to the implemented GPR width.	!(NMS)
seh.32	SEH rt, rs	Sign-extend the low halfword of rs to the implemented GPR width.	true
sel.d	SEL.D fd, fs, ft	sel.d.operation	true
sel.s	SEL.S fd, fs, ft	sel.s.operation	true
seleqz.d	SELEQZ.D fd, fs, ft	seleqz.d.operation	true
seleqz.s	SELEQZ.S fd, fs, ft	seleqz.s.operation	true
selnez.d	SELNEZ.D fd, fs, ft	selnez.d.operation	true
selnez.s	SELNEZ.S fd, fs, ft	selnez.s.operation	true

Format	Assembly	Operation	Availability
seqi.32	SEQI rt, rs, imm	Set rt to one when the source GPR equals the immediate.	true
sh.16	SH rt, offset(rs)	Store the low 16 bits of rt at rs plus offset.	true
sh.s9	SH rt, offset(rs)	Store the low 16 bits of rt at rs plus offset.	true
sh.u12	SH rt, offset(rs)	Store the low 16 bits of rt at rs plus offset.	true
sh.gp	SH rt, offset(gp)	Store the low 16 bits of rt at rs plus offset.	true
she.32	SHE offset(rs)	Store a halfword through EVA translation.	EVA
shilo.32	SHILO ac, shift	shilo.operation	DSP
shilov.32	SHILOV ac, rs	shilov.operation	DSP
shll.ph	SHLL.PH rd, rt, sa	dsp.shll.ph	DSP
shll.qb	SHLL.QB rd, rt, sa	dsp.shll.qb	DSP
shll_s.ph	SHLL_S.PH rd, rt, sa	dsp.shll_s.ph	DSP
shll_s.w	SHLL_S.W rd, rt, sa	dsp.shll_s.w	DSP
shllv.ph	SHLLV.PH rd, rt, rs	dsp.shllv.ph	DSP
shllv.qb	SHLLV.QB rd, rt, rs	dsp.shllv.qb	DSP
shllv_s.ph	SHLLV_S.PH rd, rt, rs	dsp.shllv_s.ph	DSP
shllv_s.w	SHLLV_S.W rd, rt, rs	dsp.shllv_s.w	DSP
shra.ph	SHRA.PH rd, rt, sa	dsp.shra.ph	DSP2
shra.qb	SHRA.QB rd, rt, sa	dsp.shra.qb	DSP2
shra_r.ph	SHRA_R.PH rd, rt, sa	dsp.shra_r.ph	DSP2
shra_r.qb	SHRA_R.QB rd, rt, sa	dsp.shra_r.qb	DSP2
shra_r.w	SHRA_R.W rd, rt, sa	dsp.shra_r.w	DSP
shrav.ph	SHRAV.PH rd, rt, rs	dsp.shrav.ph	DSP
shrav.qb	SHRAV.QB rd, rt, rs	dsp.shrav.qb	DSP2
shrav_r.ph	SHRAV_R.PH rd, rt, rs	dsp.shrav_r.ph	DSP
shrav_r.qb	SHRAV_R.QB rd, rt, rs	dsp.shrav_r.qb	DSP2
shrav_r.w	SHRAV_R.W rd, rt, rs	dsp.shrav_r.w	DSP
shrl.ph	SHRL.PH rd, rt, sa	dsp.shrl.ph	DSP2
shrl.qb	SHRL.QB rd, rt, sa	dsp.shrl.qb	DSP
shrlv.ph	SHRLV.PH rd, rt, rs	dsp.shrlv.ph	DSP2

Format	Assembly	Operation	Availability
shrlv.qb	SHRLV.QB rd, rt, rs	dsp.shrlv.qb	DSP
shx.32	SHX rs(rt)	shx.operation	!(NMS)
shxs.32	SHXS rs(rt)	shxs.operation	!(NMS)
sigrie.32	SIGRIE code	sigrie.operation	true
sll.16	SLL rt, rs, shift	Shift a 32-bit GPR word left and sign-extend the result.	true
sll.32	SLL rt, rs, shift	Shift a 32-bit GPR word left and sign-extend the result.	true
sllv.32	SLLV rd, rs, rt	Shift a 32-bit GPR word left by the low five bits of a GPR.	true
slt.32	SLT rd, rs, rt	Set rd when the signed, implemented-width rs is below rt.	true
slti.32	SLTI rt, rs, imm	Set the destination when a signed full-width GPR is below imm.	true
sltiu.32	SLTIU rt, rs, imm	Set the destination to one when an unsigned GPR is below the immediate.	true
sltu.32	SLTU rd, rs, rt	Set rd when the unsigned, implemented-width rs is below rt.	true
sov.32	SOV rd, rs, rt	Set rd when the signed 32-bit source sum overflows.	true
sqrt.d	SQRT.D fd, fs	sqrt.d.operation	true
sqrt.s	SQRT.S fd, fs	sqrt.s.operation	true
sra.32	SRA rt, rs, shift	Arithmetic-shift a signed 32-bit GPR word right and extend it.	true
srav.32	SRAV rd, rs, rt	Arithmetic-shift a 32-bit GPR word by a register-supplied amount.	true
srl.16	SRL rt, rs, shift	Shift a 32-bit GPR word right logically and sign-extend the result.	true
srl.32	SRL rt, rs, shift	Shift a 32-bit GPR word right logically and sign-extend the result.	true
srlv.32	SRLV rd, rs, rt	Shift a 32-bit GPR word right logically by a register-supplied amount.	true
sub.32	SUB rd, rs, rt	Subtract signed 32-bit GPR operands and trap on integer overflow.	!(NMS)

Format	Assembly	Operation	Availability
sub.d	SUB.D fd, fs, ft	sub.d.operation	true
sub.s	SUB.S fd, fs, ft	sub.s.operation	true
subq.ph.32	SUBQ.PH rd, rs, rt	Subtract signed halfword lanes without saturation.	DSP
subq_s.ph.32	SUBQ_S.PH rd, rs, rt	Subtract signed halfword lanes with signed saturation.	DSP
subq_s.w.32	SUBQ_S.W rd, rs, rt	Subtract signed word lanes with signed saturation.	DSP
subqh.ph.32	SUBQH.PH rd, rs, rt	Subtract signed halfword lanes and halve the results.	DSP2
subqh.w.32	SUBQH.W rd, rs, rt	Subtract signed word lanes and halve the result.	DSP2
subqh_r.ph.32	SUBQH_R.PH rd, rs, rt	Rounded signed half-subtract of two halfword lanes.	DSP2
subqh_r.w.32	SUBQH_R.W rd, rs, rt	Rounded signed half-subtract of two word lanes.	DSP2
subu.16	SUBU rd, rs, rt	Subtract 32-bit GPR rt from rs and sign-extend to GPR width.	true
subu.32	SUBU rd, rs, rt	Subtract 32-bit GPR rt from rs and sign-extend to GPR width.	true
subu.ph.32	SUBU.PH rd, rs, rt	Subtract unsigned halfword lanes modulo 16 bits.	DSP2
subu.qb.32	SUBU.QB rd, rs, rt	Subtract unsigned byte lanes modulo 8 bits.	DSP
subu_s.ph.32	SUBU_S.PH rd, rs, rt	Subtract unsigned halfword lanes with saturation.	DSP2
subu_s.qb.32	SUBU_S.QB rd, rs, rt	Subtract unsigned byte lanes with saturation.	DSP
subuh.qb.32	SUBUH.QB rd, rs, rt	Average unsigned byte differences by an unrounded right shift.	DSP
subuh_r.qb.32	SUBUH_R.QB rd, rs, rt	Average unsigned byte differences with rounding.	DSP
sw.16	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	true
sw.gp16	SW rt, offset(gp)	Store the low 32 bits of rt at rs plus offset.	true
sw.4x4	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	!(NMS)

Format	Assembly	Operation	Availability
sw.s9	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	true
sw.u12	SW rt, offset(rs)	Store the low 32 bits of rt at rs plus offset.	true
sw.gp	SW rt, offset(gp)	Store the low 32 bits of rt at rs plus offset.	true
sw.sp	SW rt, offset(sp)	Store the low 32 bits of rt at rs plus offset.	true
swc1.gp	SWC1 ft, ...	swc1.operation	true
swc1.s9	SWC1 ft, ...	swc1.operation	true
swc1.u12	SWC1 ft, ...	swc1.operation	true
swe.32	SWE offset(rs)	Store a word through EVA translation.	EVA
swm.32	SWM offset(rs), count	Store consecutive aligned words from sequential GPRs.	!(NMS)
swpc.48	SWPC rt, offset	Store a word using the nanoMIPS PC-relative form.	!(NMS)
swx.32	SWX rs(rt)	swx.operation	!(NMS)
swxs.32	SWXS rs(rt)	swxs.operation	!(NMS)
sync.32	SYNC stype	Impose the memory-ordering constraints selected by stype.	true
synci.s9	SYNCI offset(rs)	synci.operation	true
synci.u12	SYNCI offset(rs)	synci.operation	true
synci.eva	SYNCIE offset(rs)	syncie.operation	EVA
syscall.32	SYSCALL code	Raise a system-call exception.	true
syscall.16	SYSCALL code	Raise a system-call exception.	true
teq.32	TEQ rs, rt, code	Raise Trap when the full-width source GPR values are equal.	!(NMS)
tlbinv.32	TLBINV	tlbinv.operation	true
tlbinvf.32	TLBINVF	tlbinvf.operation	true
tlbp.32	TLBP	Probe the TLB for the translation selected by EntryHi.	true
tlbr.32	TLBR	tlbr.operation	true
tlbwi.32	TLBWI	Write the current CP0 translation state to the indexed TLB entry.	true
tlbwr.32	TLBWR	tlbwr.operation	true

Format	Assembly	Operation	Availability
tne.32	TNE <i>rs</i> , <i>rt</i> , <i>code</i>	Raise Trap when the full-width source GPR values differ.	!(NMS)
trunc.l.d	TRUNC.L.D <i>fd</i> , <i>fs</i>	trunc.l.d.operation	true
trunc.l.s	TRUNC.L.S <i>fd</i> , <i>fs</i>	trunc.l.s.operation	true
trunc.w.d	TRUNC.W.D <i>fd</i> , <i>fs</i>	trunc.w.d.operation	true
trunc.w.s	TRUNC.W.S <i>fd</i> , <i>fs</i>	trunc.w.s.operation	true
ualdm.32	UALDM <i>rt</i> , <i>offset(rs)</i> , <i>count</i>	ualdm.operation	AT >= MIPS64_32 && !(NMS)
ualh.32	UALH <i>offset(rs)</i>	Load an unaligned signed halfword.	!(NMS)
ualwm.32	UALWM <i>offset(rs)</i> , <i>count</i>	Load consecutive possibly unaligned words into sequential GPRs.	!(NMS)
uasdm.32	UASDM <i>rt</i> , <i>offset(rs)</i> , <i>count</i>	uasdm.operation	AT >= MIPS64_32 && !(NMS)
uash.32	UASH <i>offset(rs)</i>	Store an unaligned halfword.	!(NMS)
uaswm.32	UASWM <i>offset(rs)</i> , <i>count</i>	Store consecutive possibly unaligned words from sequential GPRs.	!(NMS)
wait.32	WAIT <i>code</i>	wait.operation	true
wrdsp.32	WRDSP <i>rs</i> , <i>mask</i>	wrdsp.operation	DSP
wrgpr.32	WRPGPR <i>dst</i> , <i>src</i>	Write the previous shadow-set GPR, or copy when no SRS is present.	true
wsbh.32	WSBH <i>rt</i> , <i>rs</i>	Swap bytes within each halfword of a 32-bit GPR word.	!(NMS)
xor.16	XOR <i>rd</i> , <i>rs</i>	Compute a full-GPR logical XOR and write the result to a GPR.	true
xor.32	XOR <i>rd</i> , <i>rs</i> , <i>rt</i>	Compute a full-GPR logical XOR and write the result to a GPR.	true
xori.32	XORI <i>rt</i> , <i>rs</i> , <i>imm</i>	Compute a full-GPR logical XOR with a zero-extended immediate.	true

Chapter 2. System Registers

This chapter is generated from the declarative coprocessor-register specifications. Each field has one architectural description and named-value vocabulary. Field availability controls architectural presence; configuration conditions select access and reset behavior. Named `on_read` and `on_write` callbacks describe field behavior that depends on other processor state.

2.1. C0

Coprocessor

0

Availability

true

Register	Encoding	Width	Availability	Fields
Index	0, 0	32	true	3
VPControl	0, 4	32	VP	1
Random	1, 0	32	AR < R6	1
EntryLo0	2, 0	32 / 64	true	13
EntryLo1	3, 0	32 / 64	true	13
Context	4, 0	32 / 64	true	3
UserLocal	4, 2	32 / 64	MMAR >= nanoMIPS_R6 && !(NMS) AR >= R2_3_5 && ULRI	2
MemoryMapID	4, 5	32	GI	1
PageMask	5, 0	32 / 64	true	3
PageGrain	5, 1	32	AR >= R6 AR >= R2_3_5 && (LPA SP RXI SMARTMIPS)	8
Wired	6, 0	32	true	3
HWREna	7, 0	32	AR >= R2_3_5	7
BadVAddr	8, 0	32 / 64	true	2
BadInstr	8, 1	32	MMAR >= nanoMIPS_R6 && !(NMS) BADINSTR	1
BadInstrP	8, 2	32	BADINSTRP	1

Register	Encoding	Width	Availability	Fields
BadInstrX	8, 3	32	MMAR >= nanoMIPS_R6 && !(NMS)	1
Count	9, 0	32	true	1
EntryHi	10, 0	32 / 64	true	7
Compare	11, 0	32	true	1
Status	12, 0	32	true	16
IntCtl	12, 1	32	AR >= R2_3_5	4
SRSctl	12, 2	32	AR >= R2_3_5	5
SRSMap	12, 3	32	AR >= R2_3_5 && SRS && (VINT VEIC)	8
Cause	13, 0	32	true	9
EPC	14, 0	32 / 64	true	2
PRId	15, 0	32	true	3
EBase	15, 1	32 / 64	AR >= R2_3_5	7
Config	16, 0	32	true	5
Config1	16, 1	32	true	15
Config2	16, 2	32	true	9
Config3	16, 3	32	AR >= R2_3_5	17
Config4	16, 4	32	true	11
Config5	16, 5	32	true	16
LLAddr	17, 0	32 / 64	true	5
XContext	20, 0	64	AT >= MIPS64_32	1
Debug	23, 0	32	DEBUG	9
DEPC	24, 0	32 / 64	DEBUG	2
ErrorEPC	30, 0	32 / 64	true	2
KScratch1	31, 2	32 / 64	AR >= R6 KSCRATCH	2
KScratch2	31, 3	32 / 64	AR >= R6 KSCRATCH	2
KScratch3	31, 4	32 / 64	AR >= R6 KSCRATCH	2
KScratch4	31, 5	32 / 64	AR >= R6 KSCRATCH	2
KScratch5	31, 6	32 / 64	AR >= R6 KSCRATCH	2
KScratch6	31, 7	32 / 64	AR >= R6 KSCRATCH	2

2.1.1. Index

Encoding

coprocessor 0, register 0, select 0

Width

32 bits

Availability

true

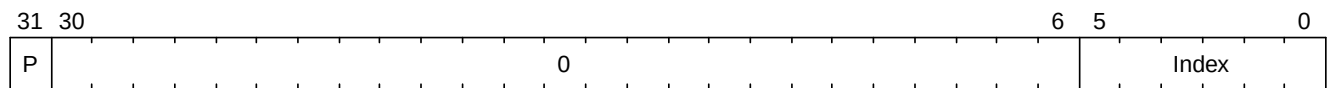
Release 1-5

Width

32 bits

Availability

AR < R6



Bits	Field	Access	Reset	Availabilit y	Behavior	Description
31	P	w1s	0x0	true	—	A TLB probe did not find a matching entry; software may set but cannot clear this field.
5:0	Index	rw	0x0	true	on write: write_impleme nted_index	TLB entry selected for indexed operations.

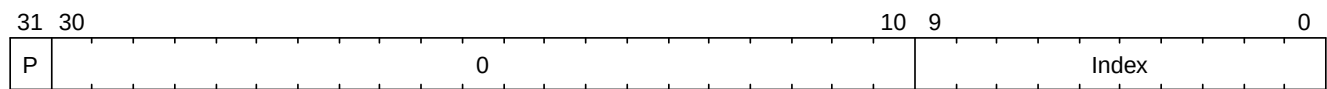
Release 6

Width

32 bits

Availability

AR >= R6



Bits	Field	Access	Reset	Availability	Behavior	Description
31	P	w1s	0x0	true	—	A TLB probe did not find a matching entry; software may set but cannot clear this field.
9:0	Index	rw	0x0	true	on write: write_implemented_index	TLB entry selected for indexed operations.

2.1.2. VPControl

Encoding

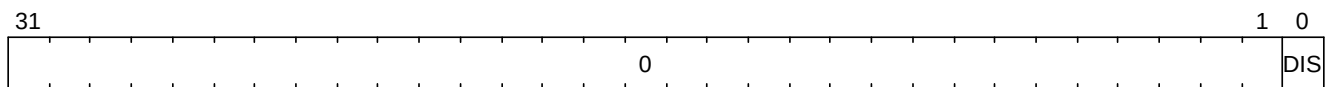
coprocessor 0, register 0, select 4

Width

32 bits

Availability

VP



Bits	Field	Access	Reset	Availability	Behavior	Description
0	DIS	ro	0x0	true	—	—

2.1.3. Random

Encoding

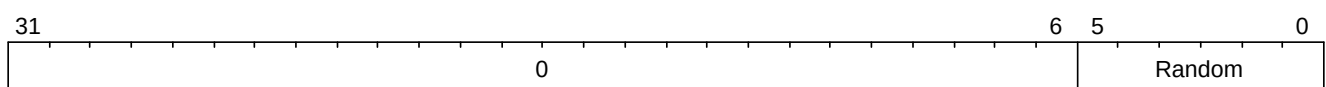
coprocessor 0, register 1, select 0

Width

32 bits

Availability

AR < R6



Width

32 bits

Availability

```
!(AT >= MIPS64_32) && !(XPA) && (MMAR >= nanoMIPS_R6 \|\| RXI)
```

31 30 29								6 5	3 2	1 0
RI	XI	PFN						C	D	V G

Bits	Field	Access	Reset	Availability	Behavior	Description
31	RI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_ri_if_enabled; on write: write_ri_if_enabled	Inhibit reads through this translation.
30	XI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_xi_if_enabled; on write: write_xi_if_enabled	Inhibit instruction execution through this translation.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

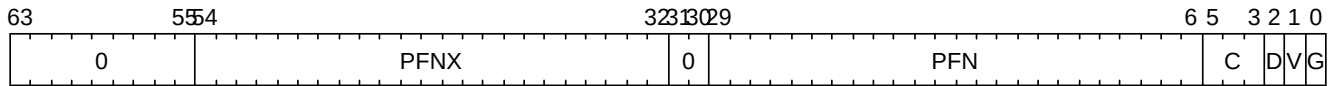
32-bit architecture with extended physical addressing

Width

64 bits

Availability

```
!(AT >= MIPS64 32) && XPA && !(MMAR >= nanoMIPS R6 \|\| RXI)
```



Bits	Field	Access	Reset	Availabilit y	Behavior	Description
54:32	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical- address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

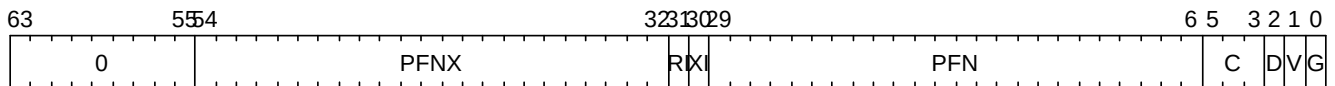
32-bit architecture with extended physical addressing and RI/XI

Width

64 bits

Availability

!(AT >= MIPS64_32) && XPA && (MMAR >= nanoMIPS_R6 \|\| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
54:32	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical- address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.
31	RI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_ri_if_enabl ed; on write: write_ri_if_enab led	Inhibit reads through this translation.
30	XI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_xi_if_enab led; on write: write_xi_if_ena bled	Inhibit instruction execution through this translation.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

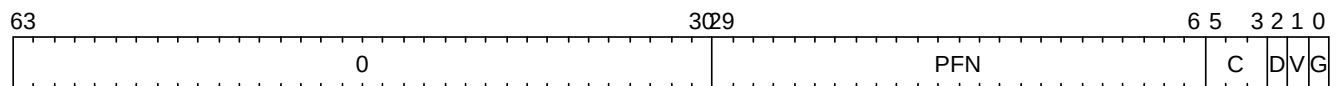
64-bit architecture, Release 1

Width

64 bits

Availability

AT >= MIPS64_32 && AR < R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

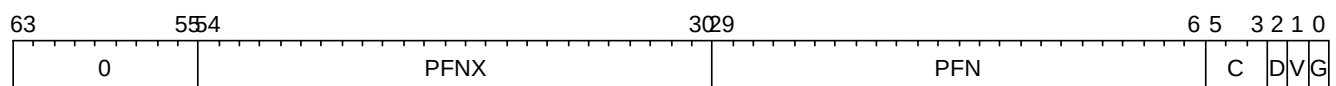
64-bit architecture with PFNX

Width

64 bits

Availability

AT >= MIPS64_32 && AR >= R2_3_5 && !(MMAR >= nanoMIPS_R6 \|\| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
54:30	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical-address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.

Bits	Field	Access	Reset	Availability	Behavior	Description
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

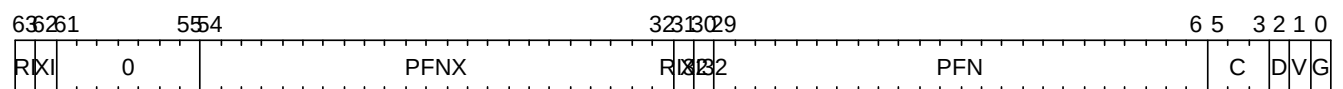
64-bit architecture with PFNX and RI/XI

Width

64 bits

Availability

AT >= MIPS64_32 && AR >= R2_3_5 && (MMAR >= nanoMIPS_R6 \\\| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
63	RI	rw	0x0	true	on read: read_ri64_if_enabled; on write: write_ri64_if_enabled	64-bit read-inhibit view selected by PageGrain.ELPA.
62	XI	rw	0x0	true	on read: read_xi64_if_enabled; on write: write_xi64_if_enabled	64-bit execute-inhibit view selected by PageGrain.ELPA.

Bits	Field	Access	Reset	Availability	Behavior	Description
54:32	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical- address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.
31	RI32	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_ri_if_enabl ed; on write: write_ri_if_enab led	Inhibit reads through this translation.
30	XI32	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_xi_if_enab led; on write: write_xi_if_ena bled	Inhibit instruction execution through this translation.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

2.1.5. EntryLo1

Encoding

coprocessor 0, register 3, select 0

Width

32 / 64 bits

Availability

true

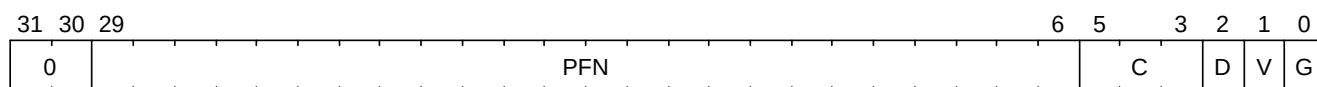
32-bit architecture without RI/XI

Width

32 bits

Availability

!(AT >= MIPS64_32) && !(XPA) && !(MMAR >= nanoMIPS_R6 \\| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

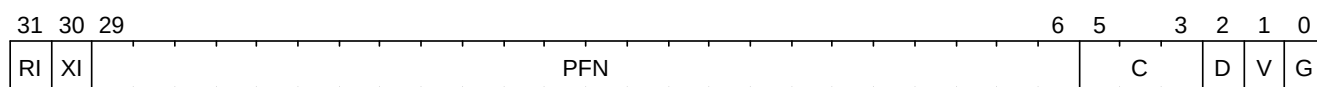
32-bit architecture with RI/XI

Width

32 bits

Availability

!(AT >= MIPS64_32) && !(XPA) && (MMAR >= nanoMIPS_R6 \\| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
31	RI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_ri_if_enabled; on write: write_ri_if_enabled	Inhibit reads through this translation.
30	XI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_xi_if_enabled; on write: write_xi_if_enabled	Inhibit instruction execution through this translation.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

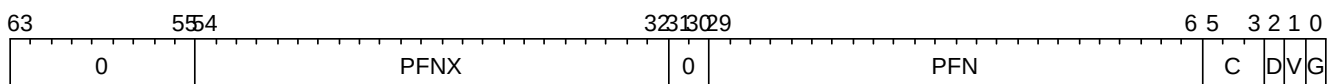
32-bit architecture with extended physical addressing

Width

64 bits

Availability

!(AT >= MIPS64_32) && XPA && !(MMAR >= nanoMIPS_R6 || || RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
54:32	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical- address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

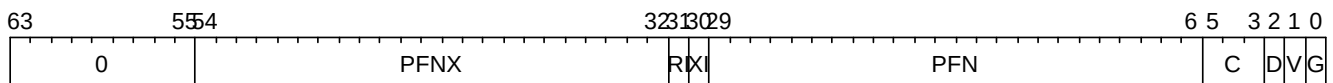
32-bit architecture with extended physical addressing and RI/XI

Width

64 bits

Availability

!(AT >= MIPS64_32) && XPA && (MMAR >= nanoMIPS_R6 ||| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
54:32	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical- address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.
31	RI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_ri_if_enabl ed; on write: write_ri_if_enab led	Inhibit reads through this translation.
30	XI	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_xi_if_enab led; on write: write_xi_if_ena bled	Inhibit instruction execution through this translation.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

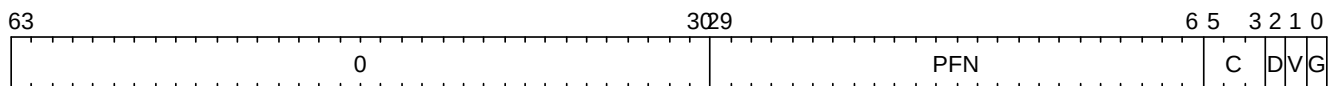
64-bit architecture, Release 1

Width

64 bits

Availability

AT >= MIPS64_32 && AR < R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

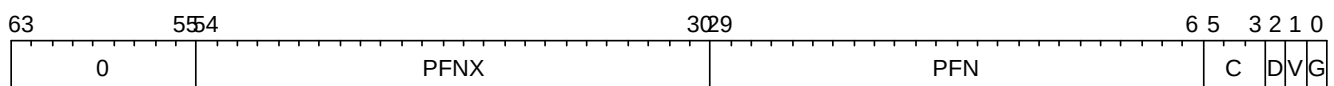
64-bit architecture with PFNX

Width

64 bits

Availability

AT >= MIPS64_32 && AR >= R2_3_5 && !(MMAR >= nanoMIPS_R6 \|\| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
54:30	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical-address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.

Bits	Field	Access	Reset	Availability	Behavior	Description
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

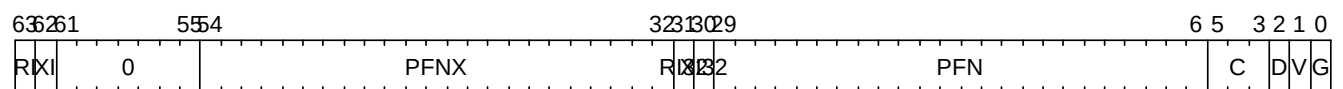
64-bit architecture with PFNX and RI/XI

Width

64 bits

Availability

AT >= MIPS64_32 && AR >= R2_3_5 && (MMAR >= nanoMIPS_R6 \\\| RXI)



Bits	Field	Access	Reset	Availability	Behavior	Description
63	RI	rw	0x0	true	on read: read_ri64_if_enabled; on write: write_ri64_if_enabled	64-bit read-inhibit view selected by PageGrain.ELPA.
62	XI	rw	0x0	true	on read: read_xi64_if_enabled; on write: write_xi64_if_enabled	64-bit execute-inhibit view selected by PageGrain.ELPA.

Bits	Field	Access	Reset	Availability	Behavior	Description
54:32	PFNX	rw if LPA XPA; ro if !(LPA XPA)	0x0	true	on read: read_pfnx; on write: write_pfnx	Large-physical- address extension of PFN. Implemented bits are concatenated above PFN when PageGrain.ELPA is one; bits beyond PABITS and the complete field while ELPA is zero read as zero.
31	RI32	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_ri_if_enabl ed; on write: write_ri_if_enab led	Inhibit reads through this translation.
30	XI32	rw if MMAR >= nanoMIPS_R6 RXI; ro if !(MMAR >= nanoMIPS_R6 RXI)	0x0	true	on read: read_xi_if_enab led; on write: write_xi_if_ena bled	Inhibit instruction execution through this translation.
29:6	PFN	rw	0x0	true	on write: write_pfn	Page frame number. Implemented bits correspond to physical address bits below PABITS.
5:3	C	rw	0x0	true	—	Cache coherency attribute.
2	D	rw	0x0	true	—	Page is writable.
1	V	rw	0x0	true	—	Translation is valid.
0	G	rw	0x0	true	—	Translation is global.

2.1.6. Context

Encoding

coprocessor 0, register 4, select 0

Width

32 / 64 bits

Availability

true

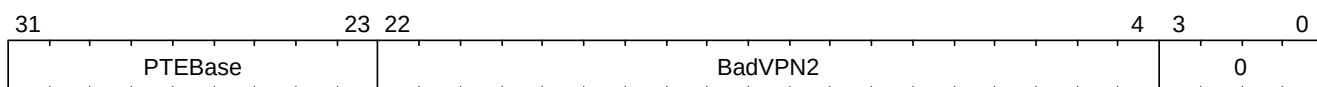
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:23	PTEBase	rw	0x0	true	—	Base of the page-table entry array.
22:4	BadVPN2	ro	0x0	true	—	VPN2 recorded by the latest TLB exception.

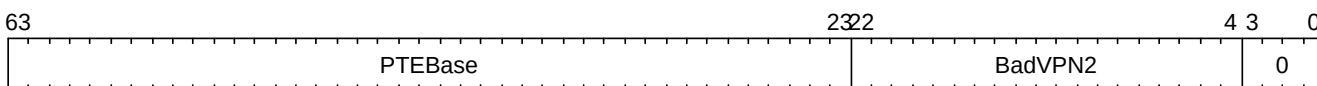
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:23	PTEBase	rw	0x0	true	—	Base of the page-table entry array.
22:4	BadVPN2	ro	0x0	true	—	VPN2 recorded by the latest TLB exception.

2.1.7. UserLocal

Encoding

coprocessor 0, register 4, select 2

Width

32 / 64 bits

Availability

MMAR >= nanoMIPS_R6 && !(NMS) \\\| AR >= R2_3_5 && ULRI

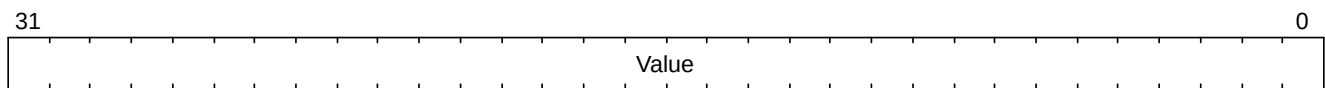
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availabilit y	Behavior	Description
31:0	Value	rw	0x0	true	—	Per-hardware- thread user-local value.

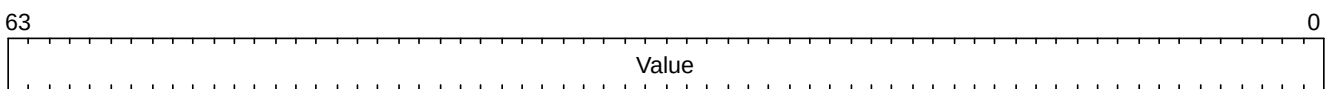
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availabilit y	Behavior	Description
63:0	Value	rw	0x0	true	—	Per-hardware- thread user-local value.

2.1.8. MemoryMapID

Encoding

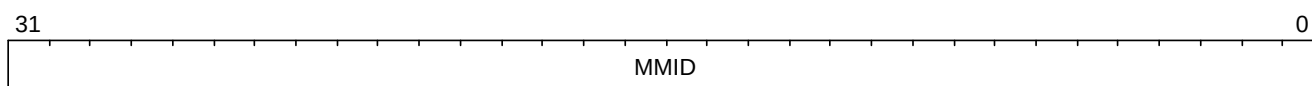
coprocessor 0, register 4, select 5

Width

32 bits

Availability

GI



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	MMID	rw	0x0	true	—	—

2.1.9. PageMask

Encoding

coprocessor 0, register 5, select 0

Width

32 / 64 bits

Availability

true

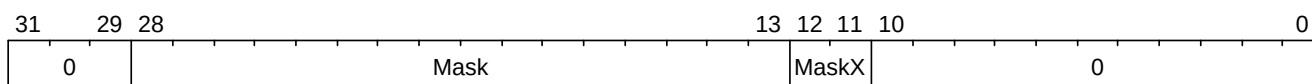
Standard page mask

Width

32 bits

Availability

!(AT >= MIPS64_32 && BPG)



Bits	Field	Access	Reset	Availability	Behavior	Description
28:13	Mask	rw	0x0	true	on write: write_legal_mask	TLB page-size mask. With big-page support the mask extends through bit 59.

Bits	Field	Access	Reset	Availability	Behavior	Description
12:11	MaskX	rw if AR >= R2_3_5 && AR < R6 && SP; ro if !(AR >= R2_3_5 && AR < R6 && SP)	0x0	true	—	One-kilobyte-page mask extension in Releases 2 through 5. Release 1 and Release 6 read this field as zero.

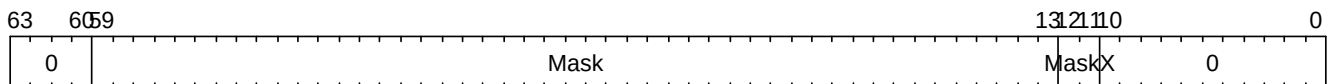
Big-page extension

Width

64 bits

Availability

AT >= MIPS64_32 && BPG



Bits	Field	Access	Reset	Availability	Behavior	Description
59:13	Mask	rw	0x0	true	on write: write_legal_mask	TLB page-size mask. With big-page support the mask extends through bit 59.
12:11	MaskX	rw if AR >= R2_3_5 && AR < R6 && SP; ro if !(AR >= R2_3_5 && AR < R6 && SP)	0x0	true	—	One-kilobyte-page mask extension in Releases 2 through 5. Release 1 and Release 6 read this field as zero.

2.1.10. PageGrain

Encoding

coprocessor 0, register 5, select 1

Width

32 bits

Availability

AR >= R6 \\| AR >= R2_3_5 && (LPA \\| SP \\| RXI \\| SMARTMIPS)

31	30	29	28	27	26	25											13	12											8	7											5	4											0
RIEXIEPLAESPIECS32							0										ASE										0										MCCause																

Bits	Field	Access	Reset	Availability	Behavior	Description
31	RIE	ro if AR >= R6; rw if AR < R6 && RXI; ro if AR < R6 && !(RXI)	0x1 if AR >= R6; 0x0 if AR < R6 && RXI; 0x0 if AR < R6 && !(RXI)	true	—	Enable the read-inhibit bit in EntryLo0 and EntryLo1.
30	XIE	ro if AR >= R6; rw if AR < R6 && RXI; ro if AR < R6 && !(RXI)	0x1 if AR >= R6; 0x0 if AR < R6 && RXI; 0x0 if AR < R6 && !(RXI)	true	—	Enable the execute-inhibit bit in EntryLo0 and EntryLo1.
29	ELPA	rw if LPA; ro if !(LPA)	0x0	true	—	Enable the PFNX large-physical-address extension.
28	ESP	rw if AR < R6 && SP; ro if !(AR < R6 && SP)	0x0	true	—	Enable one-kilobyte pages before Release 6; Release 6 disallows one-kilobyte pages.
27	IEC	ro if AR >= R6; rw if AR < R6	0x1 if AR >= R6; 0x0 if AR < R6	true	—	Select distinct exception codes for RI and XI faults.
26	S32	rw	0x0	SMARTMIPS	—	SmartMIPS control field.
12:8	ASE	rw	0x0	SMARTMIPS	—	SmartMIPS control fields.
4:0	MCCause	ro	0x0	true	—	Machine-check cause reported by the TLB implementation.

2.1.11. Wired

Encoding

coprocessor 0, register 6, select 0

Width

32 bits

Availability

true

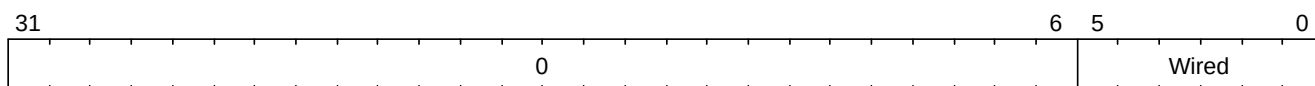
Release 1-5

Width

32 bits

Availability

AR < R6



Bits	Field	Access	Reset	Availability	Behavior	Description
5:0	Wired	rw	0x0	true	on write: write_wired	Number of TLB entries excluded from TLBWR.

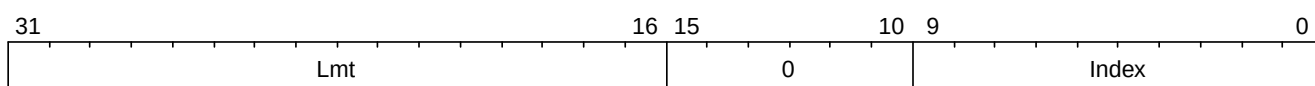
Release 6

Width

32 bits

Availability

AR >= R6



Bits	Field	Access	Reset	Availability	Behavior	Description
31:16	Lmt	ro	0x0	true	on read: read_wired_limit	Highest implemented wired TLB index.
9:0	Index	rw	0x0	true	on write: write_wired_index	Number of TLB entries excluded from TLBWR.

2.1.12. HWREna

Encoding

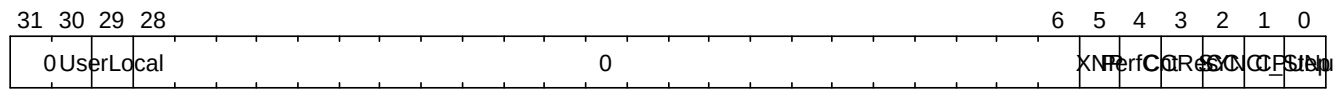
coprocessor 0, register 7, select 0

Width

32 bits

Availability

AR >= R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
29	UserLocal	rw	0x0	true	—	—
5	XNP	rw if AR >= R6; ro if !(AR >= R6)	0x0	true	—	Release 6 extended LL/SC enable; reserved before Release 6.
4	PerfCnt	rw if PERF_COUNTER; ro if !(PERF_COUNTER)	0x0	true	—	—
3	CCRes	rw	0x0	true	—	—
2	CC	rw	0x0	true	—	—
1	SYNCI_Step	rw	0x0	true	—	—
0	CPUNum	rw	0x0	true	—	—

2.1.13. BadVAddr

Encoding

coprocessor 0, register 8, select 0

Width

32 / 64 bits

Availability

true

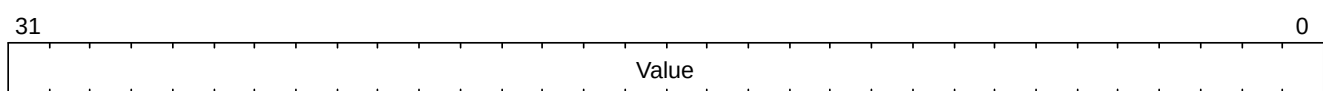
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	ro	0x0	true	—	Virtual address associated with the exception.

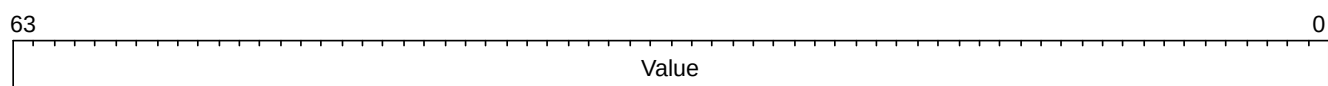
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	ro	0x0	true	—	Virtual address associated with the exception.

2.1.14. BadInstr

Encoding

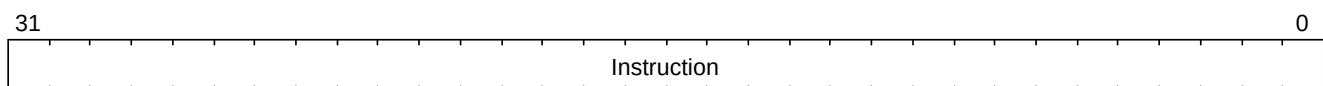
coprocessor 0, register 8, select 1

Width

32 bits

Availability

MMAR >= nanoMIPS_R6 && !(NMS) \|\| BADINSTR



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Instruction	ro	0x0	true	—	Instruction bits associated with an exception.

2.1.15. BadInstrP

Encoding

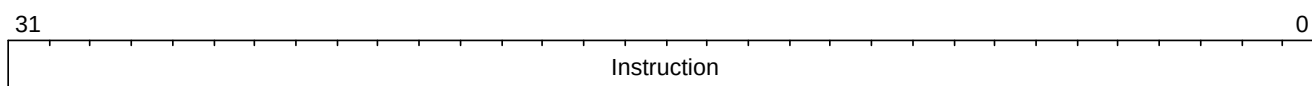
coprocessor 0, register 8, select 2

Width

32 bits

Availability

BADINSTRP



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Instruction	ro	0x0	true	—	Prior branch instruction associated with an exception.

2.1.16. BadInstrX

Encoding

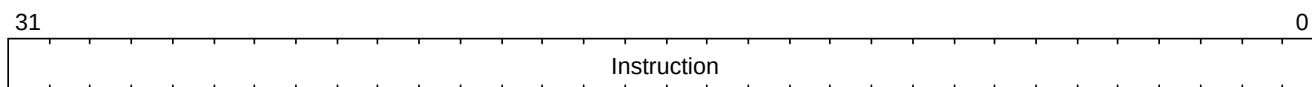
coprocessor 0, register 8, select 3

Width

32 bits

Availability

MMAR >= nanoMIPS_R6 && !(NMS)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Instruction	ro	0x0	true	—	Additional nanoMIPS instruction bits associated with an exception.

2.1.17. Count

Encoding

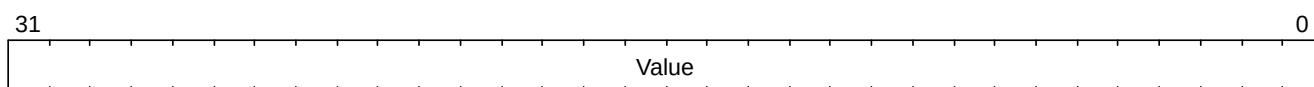
coprocessor 0, register 9, select 0

Width

32 bits

Availability

true



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	—

2.1.18. EntryHi

Encoding

coprocessor 0, register 10, select 0

Width

32 / 64 bits

Availability

true

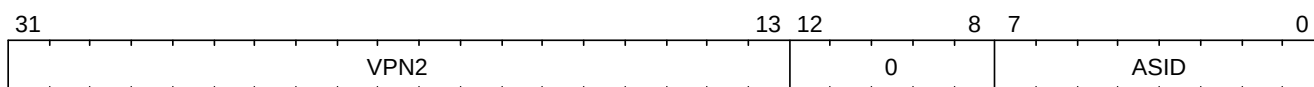
32-bit Release 1 architecture

Width

32 bits

Availability

!(AT >= MIPS64_32) && AR < R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
31:13	VPN2	rw	0x0	true	on write: write_vpn2	Virtual page number divided by two. The implemented high bit is SEGBITS - 1; higher storage bits read as zero.

Bits	Field	Access	Reset	Availability	Behavior	Description
7:0	ASID	rw	0x0	true	—	Low eight bits of the address-space identifier.

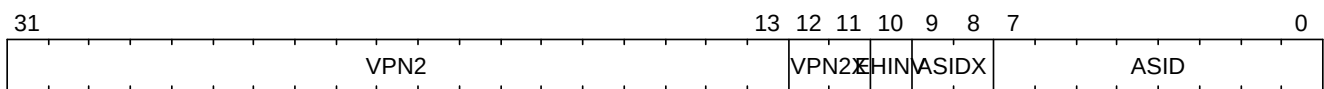
32-bit Release 2 or later architecture

Width

32 bits

Availability

!(AT >= MIPS64_32) && AR >= R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
31:13	VPN2	rw	0x0	true	on write: write_vpn2	Virtual page number divided by two. The implemented high bit is SEGBITS - 1; higher storage bits read as zero.
12:11	VPN2X	rw if SP && AR < R6; ro if !(SP && AR < R6)	0x0	true	—	VPN2 extension for one-kilobyte pages before Release 6; reserved and read as zero in Release 6.
10	EHINV	rw if AR >= R6 TLBINV; ro if !(AR >= R6 TLBINV)	0x0	true	—	Marks a TLB entry invalid independently of its VPN.
9:8	ASIDX	rw if EXTENDED_ASID; ro if !(EXTENDED_ASID)	0x0	true	—	High two bits of an extended address-space identifier.
7:0	ASID	rw	0x0	true	—	Low eight bits of the address-space identifier.

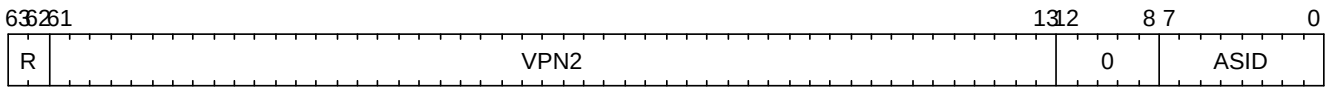
64-bit architecture, Release 1

Width

64 bits

Availability

AT >= MIPS64_32 && AR < R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
63:62	R	rw	0x0	true	on write: write_r	Virtual-address region.
61:13	VPN2	rw	0x0	true	on write: write_vpn2	Virtual page number divided by two. The implemented high bit is SEGBITS - 1; higher storage bits read as zero.
7:0	ASID	rw	0x0	true	—	Low eight bits of the address-space identifier.

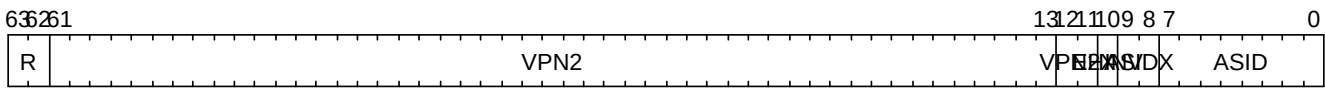
64-bit Release 2 or later architecture

Width

64 bits

Availability

AT >= MIPS64_32 && AR >= R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
63:62	R	rw	0x0	true	on write: write_r	Virtual-address region.

Bits	Field	Access	Reset	Availability	Behavior	Description
61:13	VPN2	rw	0x0	true	on write: write_vpn2	Virtual page number divided by two. The implemented high bit is SEGBITS - 1; higher storage bits read as zero.
12:11	VPN2X	rw if SP && AR < R6; ro if !(SP && AR < R6)	0x0	true	—	VPN2 extension for one-kilobyte pages before Release 6; reserved and read as zero in Release 6.
10	EHINV	rw if AR >= R6 TLBINV; ro if !(AR >= R6 TLBINV)	0x0	true	—	Marks a TLB entry invalid independently of its VPN.
9:8	ASIDX	rw if EXTENDED_ASID; ro if !(EXTENDED_ASID)	0x0	true	—	High two bits of an extended address-space identifier.
7:0	ASID	rw	0x0	true	—	Low eight bits of the address-space identifier.

2.1.19. Compare

Encoding

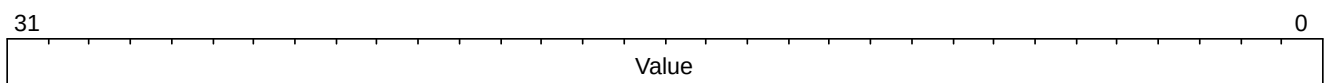
coprocessor 0, register 11, select 0

Width

32 bits

Availability

true



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	on write: write_compare	—

2.1.20. Status

Encoding

coprocessor 0, register 12, select 0

Width

32 bits

Availability

true

31	30	29	28	27	26	25	24	23	22	21	16	15	8	7	6	5	4	3	2	1	0
CU3	CU2	CU1	CU0	0	FR	0	MX	PX	BEV	0			IM		KX	SX	UX	KSU	ER	EXL	IE

Bits	Field	Access	Reset	Availability	Behavior	Description
31	CU3	ro if AR >= R2_3_5; rw if !(AR >= R2_3_5)	0x0	true	—	Enable Coprocessor 3 instructions.
30	CU2	rw if CP2; ro if !(CP2)	0x0	true	—	Enable Coprocessor 2 instructions.
29	CU1	rw if CP1; ro if !(CP1)	0x0	true	—	Enable Coprocessor 1 instructions.
28	CU0	rw	0x0	true	—	Enable Coprocessor 0 access.
26	FR	rw if AR < R6; ro if !(AR < R6)	0x0 if AR < R6; 0x1 if !(AR < R6)	true	—	Select the 32- or 64-bit floating-point register mode.
24	MX	rw if DSP; ro if !(DSP)	0x0	true	—	Enable access to DSP Module resources; reserved and read-only zero when the DSP Module is not implemented.
23	PX	rw if AT >= MIPS64_32; ro if !(AT >= MIPS64_32)	0x0	true	—	Enable 64-bit operations; reserved and read-only zero without MIPS64.
22	BEV	rw	0x1	true	—	Select boot exception vectors.
15:8	IM	rw	0x0	true	—	Interrupt mask bits.

Bits	Field	Access	Reset	Availability	Behavior	Description
7	KX	rw if AT >= MIPS64_32; ro if !(AT >= MIPS64_32)	0x0	true	—	Enable 64-bit kernel addresses; reserved and read-only zero without MIPS64.
6	SX	rw if AT >= MIPS64_32; ro if !(AT >= MIPS64_32)	0x0	true	on write: write_sx	Enable 64-bit supervisor addresses.
5	UX	rw if AT >= MIPS64_32; ro if !(AT >= MIPS64_32)	0x0	true	on write: write_ux	Enable 64-bit user addresses.
4:3	KSU	rw	0x0	true	on write: write_ksu	—
2	ERL	rw	0x0	true	—	—
1	EXL	rw	0x0	true	—	—
0	IE	rw	0x0	true	—	Global interrupt enable.

MX Values

Bits

24

Value	Name	Description
0	Disabled	DSP Module resource access is disabled.
1	Enabled	DSP Module resource access is enabled.

2.1.21. IntCtl

Encoding

coprocessor 0, register 12, select 1

Width

32 bits

Availability

AR >= R2_3_5

31	29	28	26	25	23	22					10	9			5	4			0
IPTI		IPPCI		IPFDC						0				VS			0		

Bits	Field	Access	Reset	Availability	Behavior	Description
31:29	IPTI	ro	0x7	true	—	Interrupt pin used by the Count/Compare timer request.
28:26	IPPCI	ro	0x0	true	—	Interrupt pin used by performance-counter requests.
25:23	IPFDC	ro	0x0	true	—	Interrupt pin used by fast-debug-channel requests.
9:5	VS	rw if VINT VEIC; ro if !(VINT VEIC)	0x0	true	—	Vectored-interrupt spacing, encoded in units of 32 bytes.

VS Values

Bits

9:5

Value	Name	Description
0	Disabled	Use the common interrupt vector.
1	32 bytes	—
2	64 bytes	—
4	128 bytes	—
8	256 bytes	—
16	512 bytes	—

2.1.22. SRScTl

Encoding

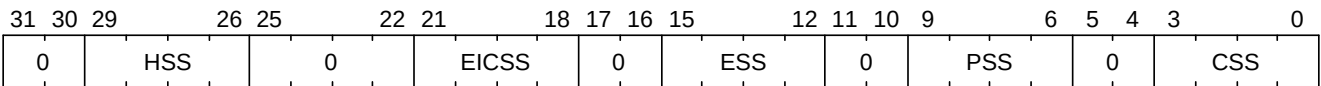
coprocessor 0, register 12, select 2

Width

32 bits

Availability

AR >= R2_3_5



Bits	Field	Access	Reset	Availability	Behavior	Description
29:26	HSS	ro	0x1 if SRS; 0x0 otherwise	true	—	Highest implemented shadow-register-set number.
21:18	EICSS	ro	0x0	true	—	Shadow set selected by an external interrupt controller.
15:12	ESS	rw if SRS; ro if !(SRS)	0x0	true	—	Shadow set selected for non-vectorized exceptions.
9:6	PSS	rw if SRS; ro if !(SRS)	0x0	true	—	Shadow set active before the current exception.
3:0	CSS	ro	0x0	true	—	Currently selected GPR shadow set.

2.1.23. SRSMap

Encoding

coprocessor 0, register 12, select 3

Width

32 bits

Availability

AR >= R2_3_5 && SRS && (VINT \|\| VEIC)

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
SSV7		SSV6		SSV5		SSV4		SSV3		SSV2		SSV1		SSV0	

Bits	Field	Access	Reset	Availability	Behavior	Description
31:28	SSV7	rw	0x0	true	—	Shadow-register set for interrupt vector 7.
27:24	SSV6	rw	0x0	true	—	Shadow-register set for interrupt vector 6.
23:20	SSV5	rw	0x0	true	—	Shadow-register set for interrupt vector 5.

Bits	Field	Access	Reset	Availability	Behavior	Description
19:16	SSV4	rw	0x0	true	—	Shadow-register set for interrupt vector 4.
15:12	SSV3	rw	0x0	true	—	Shadow-register set for interrupt vector 3.
11:8	SSV2	rw	0x0	true	—	Shadow-register set for interrupt vector 2.
7:4	SSV1	rw	0x0	true	—	Shadow-register set for interrupt vector 1.
3:0	SSV0	rw	0x0	true	—	Shadow-register set for interrupt vector 0.

2.1.24. Cause

Encoding

coprocessor 0, register 13, select 0

Width

32 bits

Availability

true

31	30	29	28	27	26	24	23	22	16	15	14	10	9	8	7	6	2	1	0
BD	TI	CE	DC	0	IV	0			IP7	0		IP1	IP0	0		ExcCode	0		

Bits	Field	Access	Reset	Availability	Behavior	Description
31	BD	ro	0x0	true	—	Exception occurred in a branch delay slot.
30	TI	ro	0x0	true	—	Count has matched Compare.
29:28	CE	ro	0x0	true	—	—
27	DC	rw	0x0	true	—	—
23	IV	rw	0x0	true	—	—
15	IP7	ro	0x0	true	—	—

Bits	Field	Access	Reset	Availability	Behavior	Description
9	IP1	rw	0x0	true	—	—
8	IP0	rw	0x0	true	—	—
6:2	ExcCode	ro	0x0	true	—	Cause code for the most recent exception.

2.1.25. EPC

Encoding

coprocessor 0, register 14, select 0

Width

32 / 64 bits

Availability

true

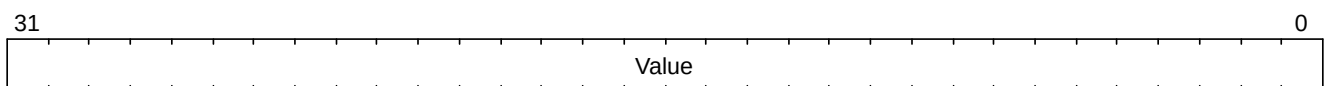
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	Program counter at which an exception occurred.

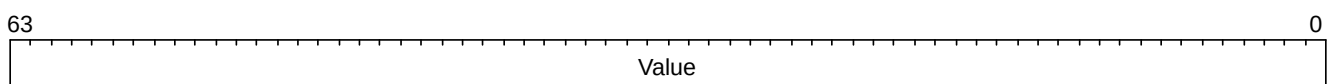
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	Program counter at which an exception occurred.

2.1.26. P_{RI}D

Encoding

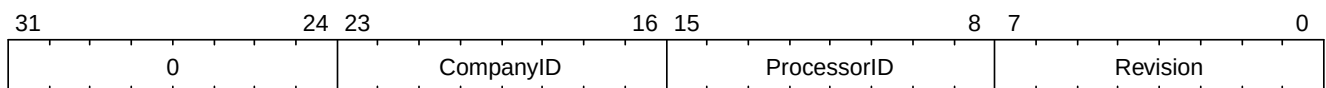
coprocessor 0, register 15, select 0

Width

32 bits

Availability

true



Bits	Field	Access	Reset	Availability	Behavior	Description
23:16	Company ID	ro	0x1	true	—	MIPS Technologies company identifier.
15:8	ProcessorID	ro	0x0	true	—	Implementation-defined processor identifier.
7:0	Revision	ro	0x0	true	—	Implementation-defined processor revision.

2.1.27. E_{Base}

Encoding

coprocessor 0, register 15, select 1

Width

32 / 64 bits

Availability

AR ≥ R2_3_5

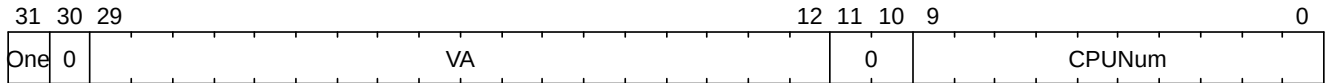
32-bit legacy exception base

Width

32 bits

Availability

!(AT >= MIPS64_32) && !(SEGMENT_CONTROL)



Bits	Field	Access	Reset	Availability	Behavior	Description
31	One	ro	0x1	true	—	Architecturally fixed one.
29:12	VA	rw	0x0	true	—	Exception base within kseg0 or kseg1.
9:0	CPUNum	ro	0x0	true	—	Processor or virtual-processor number.

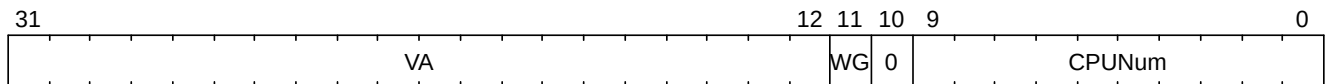
32-bit relocatable exception base

Width

32 bits

Availability

!(AT >= MIPS64_32) && SEGMENT_CONTROL



Bits	Field	Access	Reset	Availability	Behavior	Description
31:12	VA	rw	0x80000	true	—	Relocatable base address for normal exception vectors.
11	WG	rw	0x0	true	—	Write gate for the high bits of EBase.VA.
9:0	CPUNum	ro	0x0	true	—	Processor or virtual-processor number.

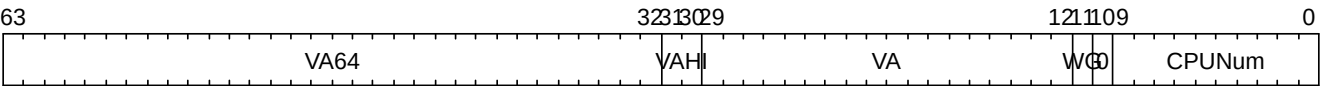
64-bit relocatable exception base

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:32	VA64	ro	0xffffffff	true	—	Sign-extension half of the 32-bit exception base.
31:30	VAHI	ro	0x2	true	—	Read-only high region bits of the exception base.
29:12	VA	rw	0x0	true	—	Relocatable low exception-base address bits.
11	WG	rw	0x0	true	—	Write gate for the high bits of EBase.VA.
9:0	CPUNum	ro	0x0	true	—	Processor or virtual-processor number.

2.1.28. Config

Encoding

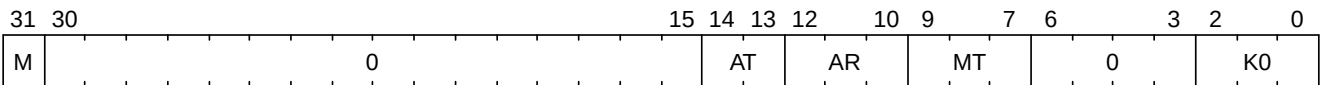
coprocessor 0, register 16, select 0

Width

32 bits

Availability

true



Bits	Field	Access	Reset	Availability	Behavior	Description
31	M	ro	0x1	true	—	Config1 is implemented.
14:13	AT	ro	Config.AT	true	—	Architecture type implemented by the processor.
12:10	AR	ro	Config.AR	true	—	Architecture release encoding.

Bits	Field	Access	Reset	Availability	Behavior	Description
9:7	MT	ro	0x1	true	—	Memory-management unit type.
2:0	K0	rw	0x2	true	—	Kseg0 cacheability and coherency attribute.

AT Values

Bits

14:13

Value	Name	Description
0	MIPS32	MIPS32 architecture with 32-bit addresses.
1	MIPS64_32	MIPS64 architecture restricted to 32-bit compatibility segments.
2	MIPS64	MIPS64 architecture with 32-bit and 64-bit addresses.

AR Values

Bits

12:10

Value	Name	Description
0	R1	Release 1.
1	R2_3_5	Release 2, Release 3, or Release 5.
2	R6	Release 6.

MT Values

Bits

9:7

Value	Name	Description
1	TLB	Standard translation lookaside buffer MMU.

K0 Values

Bits

2:0

Value	Name	Description
2	UNCACHED	Uncached.
3	CACHEABLE_NONCOHERENT	Cacheable, noncoherent, write-back with write allocation.

2.1.29. Config1

Encoding

coprocessor 0, register 16, select 1

Width

32 bits

Availability

true

31	30	25		24	22		21	19		18	16		15	13		12	10		9	7		6	5	4	3	2	1	0
M	MMUSize			IS		IL		IA		DS		DL		DA		C2	MD	PC	WR	CA	EP	FP						

Bits	Field	Access	Reset	Availability	Behavior	Description
31	M	ro	0x1	true	—	Config2 is implemented.
30:25	MMUSize	ro	0xf	true	—	The number of TLB entries minus one.
24:22	IS	ro	0x0	true	—	Encoded number of instruction-cache sets per way.
21:19	IL	ro	0x0	true	—	Encoded instruction-cache line size.
18:16	IA	ro	0x0	true	—	Encoded instruction-cache associativity.
15:13	DS	ro	0x0	true	—	Encoded number of data-cache sets per way.
12:10	DL	ro	0x0	true	—	Encoded data-cache line size.
9:7	DA	ro	0x0	true	—	Encoded data-cache associativity.

Bits	Field	Access	Reset	Availability	Behavior	Description
6	C2	ro	0x0	true	—	Indicates whether Coprocessor 2 is implemented.
5	MD	ro	0x0	true	—	Indicates whether the MDMX ASE is implemented.
4	PC	ro	0x1 if PERF_COUNTER; 0x0 otherwise	true	—	Indicates whether performance counters are implemented.
3	WR	ro	0x0	true	—	Indicates whether Watch registers are implemented.
2	CA	ro	0x1 if MIPS16E; 0x0 otherwise	true	—	Indicates whether the MIPS16E ASE is implemented.
1	EP	ro	0x1 if DEBUG; 0x0 otherwise	true	—	Indicates whether EJTAG/debug support is implemented.
0	FP	ro	0x1 if CP1; 0x0 otherwise	true	—	Indicates whether Coprocessor 1 is implemented.

CA Values

Bits

2

Value	Name	Description
0	Not implemented	—
1	MIPS16E implemented	—

EP Values

Bits

1

Value	Name	Description
0	Not implemented	—
1	EJTAG implemented	—

2.1.30. Config2

Encoding

coprocessor 0, register 16, select 2

Width

32 bits

Availability

true

31	30	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
M	TU	TS	TL	TA	SU	SS	SL	SA								

Bits	Field	Access	Reset	Availability	Behavior	Description
31	M	ro	0x1 if AR >= R2_3_5; 0x0 otherwise	true	—	Config3 is implemented.
30:28	TU	rw if L3C; ro if !(L3C)	0x0	true	—	Implementation-dependent tertiary-cache control.
27:24	TS	ro	0x0	true	—	Encoded tertiary-cache sets per way.
23:20	TL	ro	0x1 if L3C; 0x0 otherwise	true	—	Encoded tertiary-cache line size.
19:16	TA	ro	0x0	true	—	Encoded tertiary-cache associativity.
15:12	SU	rw if L2C; ro if !(L2C)	0x0	true	—	Implementation-dependent secondary-cache control.
11:8	SS	ro	0x0	true	—	Encoded secondary-cache sets per way.
7:4	SL	ro	0x1 if L2C; 0x0 otherwise	true	—	Encoded secondary-cache line size.

Bits	Field	Access	Reset	Availability	Behavior	Description
3:0	SA	ro	0x0	true	—	Encoded secondary-cache associativity.

2.1.31. Config3

Encoding

coprocessor 0, register 16, select 3

Width

32 bits

Availability

AR >= R2_3_5

31	30	29	28	27	26	25	24		21	20		18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	BPG	0	BP	BI	SC		0		MMAR	ISA	OnExt	ISA	ULRR	RX	DSP	SPP	0	LPA	VEIC	Int	SP	0	SM	0						

Bits	Field	Access	Reset	Availability	Behavior	Description
31	M	ro	0x1	true	—	Config4 is implemented.
30	BPG	ro	0x1 if AT >= MIPS64_32 && BPG; 0x0 otherwise	true	—	Indicates whether PageMask has the 64-bit big-page layout.
27	BP	ro	0x1 if BADINSTRP; 0x0 otherwise	true	—	Indicates whether the BadInstrP register is implemented.
26	BI	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) BADINSTR; 0x0 otherwise	true	—	Indicates whether the BadInstr register is implemented.
25	SC	ro	0x1 if SEGMENT_CONTROL; 0x0 otherwise	true	—	Indicates whether segmentation-control registers are implemented.

Bits	Field	Access	Reset	Availability	Behavior	Description
20:18	MMAR	ro	Config3.MMAR	MMAR >= nanoMIPS_R6 MMAR >= R3_5 && MMAR < nanoMIPS_R6	—	Compressed-ISA architecture revision encoding.
16	ISAOnExc	ro if MMAR >= nanoMIPS_R6 ; rw if ISA != COMPRESSED_ONLY && MMAR >= R3_5 && MMAR < nanoMIPS_R6	0x1 if MMAR >= nanoMIPS_R6 ; 0x0 if ISA != COMPRESSED_ONLY && MMAR >= R3_5 && MMAR < nanoMIPS_R6	MMAR >= nanoMIPS_R6 MMAR >= R3_5 && MMAR < nanoMIPS_R6	—	ISA mode selected on exception entry.
15:14	ISA	ro	Config3.ISA	true	—	Implemented ISA-family encoding.
13	ULRI	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) ULRI; 0x0 otherwise	true	—	Indicates whether the UserLocal register is implemented.
12	RXI	ro	0x1 if MMAR >= nanoMIPS_R6 RXI; 0x0 otherwise	true	—	Indicates whether read- and execute-inhibit TLB bits are implemented.
11	DSP2P	ro	0x1 if DSP2; 0x0 otherwise	true	—	Indicates whether DSP Module Revision 2 is implemented.
10	DSPP	ro	0x1 if DSP; 0x0 otherwise	true	—	Indicates whether the DSP ASE is implemented.
7	LPA	ro	0x1 if LPA; 0x0 otherwise	true	—	Indicates whether large physical addresses are implemented.
6	VEIC	ro	0x1 if VEIC; 0x0 otherwise	true	—	An external interrupt controller supplies vector numbers.

Bits	Field	Access	Reset	Availability	Behavior	Description
5	VInt	ro	0x1 if VINT; 0x0 otherwise	true	—	Vectored interrupt mode is implemented.
4	SP	ro	0x1 if SP; 0x0 otherwise	true	—	Indicates whether one-kilobyte pages are implemented.
1	SM	ro	0x1 if SMARTMIPS; 0x0 otherwise	true	—	Indicates whether the SmartMIPS ASE is implemented.

MMAR Values

Bits

20:18

Value	Name	Description
0	R3_5	microMIPS Release 3 or Release 5.
2	R6	microMIPS Release 6.
3	nanoMIPS_R6	nanoMIPS Release 6.

ISAOnExc Values

Bits

16

Value	Name	Description
0	MIPS	—
1	nano/microMIPS	—

ISA Values

Bits

15:14

Value	Name	Description
0	MIPS	Only the MIPS instruction set is implemented.
1	nanoMIPS	Only nanoMIPS is implemented.
2	MIPS and microMIPS	Reset selects MIPS.
3	MIPS and microMIPS	Reset selects microMIPS.

2.1.32. Config4

Encoding

coprocessor 0, register 16, select 4

Width

32 bits

Availability

true

31	30	29	28	27	24	23	22	21	20	19	18	17	16	15	14	13	0
M	IE	AE	VTLBSizeExt	Exist	KScr6Exist	KScr5Exist	KScr4Exist	KScr3Exist	KScr2Exist	KScr1Exist	KScr0Exist	MMUExtDef					0

Bits	Field	Access	Reset	Availability	Behavior	Description
31	M	ro	0x1	true	—	Config5 is implemented.
30:29	IE	ro	0x2 if MMAR >= nanoMIPS_R6 && !(NMS) TLBINV; 0x0 otherwise	true	—	TLB invalidate operations are implemented.
28	AE	ro	0x1 if EXTENDED_ASID; 0x0 otherwise	true	—	Ten-bit ASIDs are implemented.
27:24	VTLBSizeExt	ro	0x0	true	—	High bits of the implemented standard-TLB size.
23	KScr6Exist	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) KSCRATCH; 0x0 otherwise	true	—	KScratch6 is implemented.
22	KScr5Exist	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) KSCRATCH; 0x0 otherwise	true	—	KScratch5 is implemented.
21	KScr4Exist	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) KSCRATCH; 0x0 otherwise	true	—	KScratch4 is implemented.
20	KScr3Exist	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) KSCRATCH; 0x0 otherwise	true	—	KScratch3 is implemented.

Bits	Field	Access	Reset	Availability	Behavior	Description
19	KScr2Exist	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) KSCRATCH; 0x0 otherwise	true	—	KScratch2 is implemented.
18	KScr1Exist	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) KSCRATCH; 0x0 otherwise	true	—	KScratch1 is implemented.
15:14	MMUExtDef	ro	0x0	true	—	No implementation-specific MMU extension.

2.1.33. Config5

Encoding

coprocessor 0, register 16, select 5

Width

32 bits

Availability

true

31	29	28	27	26	25	23	22	21	20	19	18	17	16	15	14	13	12	10	9	8	7	6	5	4	3	2	1	0
0	EVA	0		PMJ	0	NMS	ULS	XP	RC	PMI	GI	0	KNP	0	UFE	REV	PSB	RVH	LLB	0	UFR	0						

Bits	Field	Access	Reset	Availability	Behavior	Description
28	EVA	ro	0x1 if EVA; 0x0 otherwise	true	—	Indicates whether Enhanced Virtual Addressing is implemented.
25:23	PMJ	ro	Config5.PMJ	MMAR >= nanoMIPS_R6	—	Privileged Resource Architecture major version.
21	NMS	ro	0x1 if NMS; 0x0 otherwise	MMAR >= nanoMIPS_R6	—	Indicates whether the nanoMIPS subset is implemented.

Bits	Field	Access	Reset	Availability	Behavior	Description
20	ULS	ro	0x0	AR >= R6	—	Uncached load-linked/store-conditional accesses are supported.
19	XPA	ro	0x1 if XPA; 0x0 otherwise	true	—	Indicates that extended physical-address CP0 registers are implemented and may be accessed by the high CP0 move instructions.
18	CRCP	ro	0x1 if CRCP; 0x0 otherwise	AR >= R6	—	Indicates whether CRC instructions are implemented.
17	MI	rw if GI; ro if !(GI)	0x1 if GI; 0x0 if !(GI)	AR >= R6	—	Enable MemoryMapID-based TLB matching.
16:15	GI	ro	0x3 if GI; 0x0 otherwise	AR >= R6	—	Global-invalidation capability encoding.
13	XNP	ro	0x0 if MMAR >= nanoMIPS_R6 && !(NMS) LLX_SCX; 0x1 otherwise	AR >= R6	—	Indicates that paired LL/SC instructions are not implemented.
9	UFE	rw if CP1 && ISA != COMPRESSED_ONLY && FREP; ro if !(CP1 && ISA != COMPRESSED_ONLY && FREP)	0x0	AR >= R2_3_5	—	Enable user-mode access to the C1 FRE and NFRE aliases.
8	FRE	rw if CP1 && ISA != COMPRESSED_ONLY && FREP; ro if !(CP1 && ISA != COMPRESSED_ONLY && FREP)	0x0	AR >= R2_3_5	—	Enable FR=0 compatibility interpretation while Status.FR is one.

Bits	Field	Access	Reset	Availability	Behavior	Description
7	VP	ro	0x1 if VP; 0x0 otherwise	AR >= R6	—	Indicates whether virtual processors are implemented.
6	SBRI	rw if DEBUG; ro if !(DEBUG)	0x0	AR >= R6	—	Restrict SDBBP outside kernel mode.
5	MVH	ro	0x1 if MMAR >= nanoMIPS_R6 && !(NMS) MVH; 0x0 otherwise	true	—	Indicates whether high CP0 move instructions are implemented.
4	LLB	ro	0x1 if MMAR >= nanoMIPS_R6 LLB; 0x0 otherwise	true	—	Indicates whether ERETNC and LLAddr.LLB are implemented.
2	UFR	rw if CP1 && ISA != COMPRESSED_ONLY && UFRP; ro if !(CP1 && ISA != COMPRESSED_ONLY && UFRP)	0x0	AR >= R2_3_5	—	Enable user-mode access to the C1 UFR and UNFR aliases.

2.1.34. LLAddr

Encoding

coprocessor 0, register 17, select 0

Width

32 / 64 bits

Availability

true

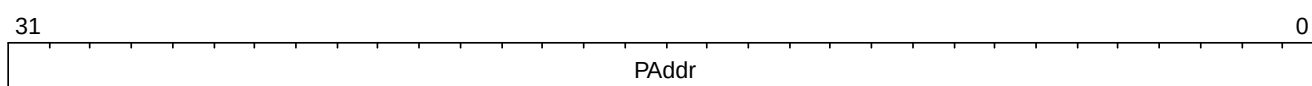
32-bit architecture without LLB

Width

32 bits

Availability

!(AT >= MIPS64_32) && !(MMAR >= nanoMIPS_R6 || LLB)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	PAddr	ro	0x0	true	—	Physical address recorded by the most recent linked load.

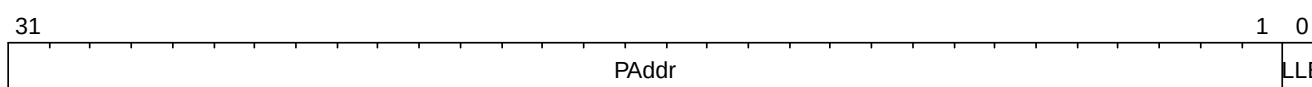
32-bit architecture with LLB

Width

32 bits

Availability

!(AT >= MIPS64_32) && (MMAR >= nanoMIPS_R6 \\\| LLB)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:1	PAddr	ro	0x0	true	—	Physical address recorded by the most recent linked load; implemented low address bits are implementation-dependent.
0	LLB	w0c	0x0	true	on write: write_llb	Architectural load-linked bit. Software may clear this bit but cannot set it.

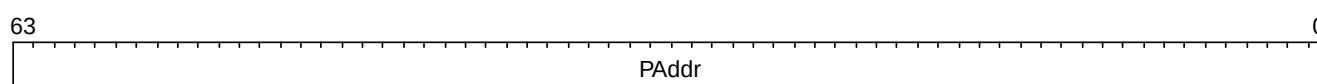
64-bit architecture without LLB

Width

64 bits

Availability

AT >= MIPS64_32 && !(MMAR >= nanoMIPS_R6 \\\| LLB)



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	PAddr	ro	0x0	true	—	Physical address recorded by the most recent linked load.

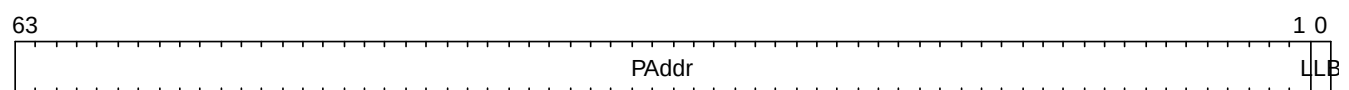
64-bit architecture with LLB

Width

64 bits

Availability

AT >= MIPS64_32 && (MMAR >= nanoMIPS_R6 \\\| LLB)



Bits	Field	Access	Reset	Availability	Behavior	Description
63:1	PAddr	ro	0x0	true	—	Physical address recorded by the most recent linked load; implemented low address bits are implementation-dependent.
0	LLB	w0c	0x0	true	on write: write_llb	Architectural load-linked bit. Software may clear this bit but cannot set it.

2.1.35. XContext

Encoding

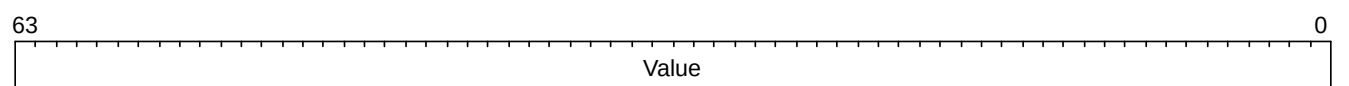
coprocessor 0, register 20, select 0

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	on write: write_value	Extended page-table context. The processor updates the R and BadVPN2 portions on a TLB exception while preserving the PTEBase portion. Software writes update only PTEBase; bits 3:0 read as zero.

2.1.36. Debug

Encoding

coprocessor 0, register 23, select 0

Width

32 bits

Availability

DEBUG

31	30	29		26	25	24				15	14			10	9		6	5	4	3	2	1	0
DBD	DM		0	CountDM					0				DExcCode		0		DINT	DIB	DDBS	DDBL	DBp		0

Bits	Field	Access	Reset	Availability	Behavior	Description
31	DBD	ro	0x0	true	—	—
30	DM	ro	0x0	true	—	—
25	CountDM	rw	0x0	true	—	—
14:10	DExcCode	ro	0x0	true	—	—
5	DINT	ro	0x0	true	—	—
4	DIB	ro	0x0	true	—	—
3	DDBS	ro	0x0	true	—	—
2	DDBL	ro	0x0	true	—	—
1	DBp	ro	0x0	true	—	—

2.1.37. DEPC

Encoding

coprocessor 0, register 24, select 0

Width

32 / 64 bits

Availability

DEBUG

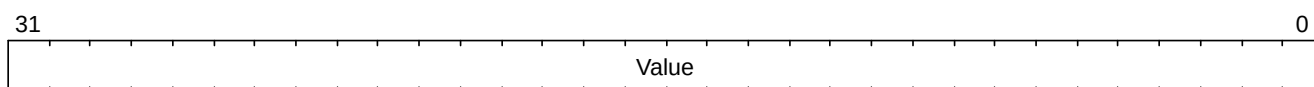
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	—

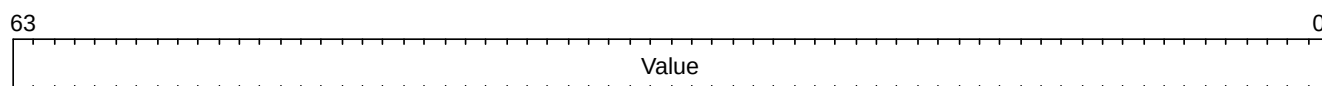
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	—

2.1.38. ErrorEPC

Encoding

coprocessor 0, register 30, select 0

Width

32 / 64 bits

Availability

true

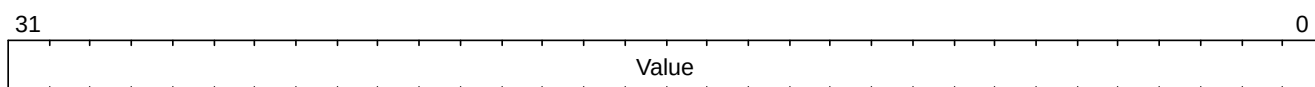
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	Program counter at which an error exception occurred.

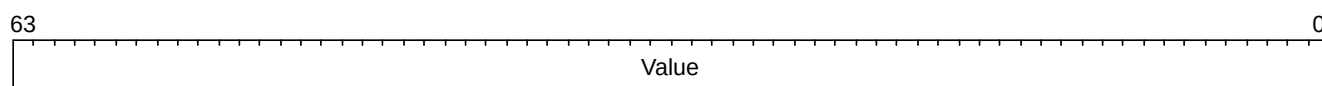
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	Program counter at which an error exception occurred.

2.1.39. KScratch1

Encoding

coprocessor 0, register 31, select 2

Width

32 / 64 bits

Availability

AR >= R6 \\\ KSCRATCH

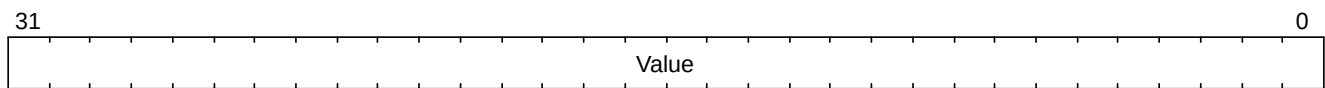
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availabilit y	Behavior	Description
31:0	Value	rw	0x0	true	—	—

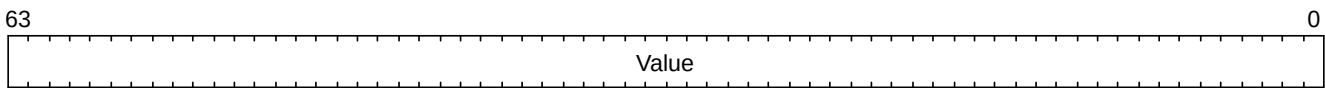
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availabilit y	Behavior	Description
63:0	Value	rw	0x0	true	—	—

2.1.40. KScratch2

Encoding

coprocessor 0, register 31, select 3

Width

32 / 64 bits

Availability

AR >= R6 \\\ KSCRATCH

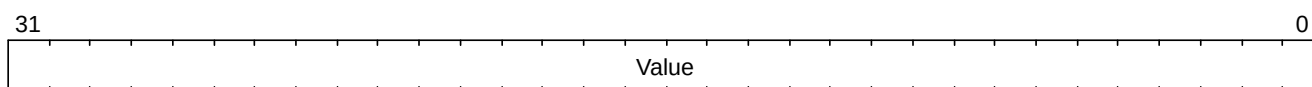
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	—

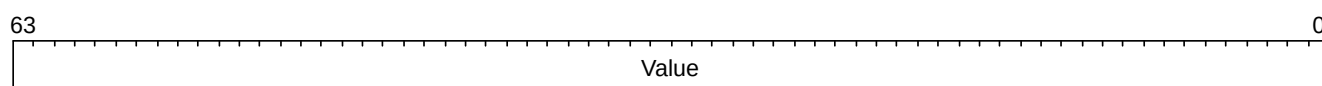
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	—

2.1.41. KScratch3

Encoding

coprocessor 0, register 31, select 4

Width

32 / 64 bits

Availability

AR >= R6 \\| KSCRATCH

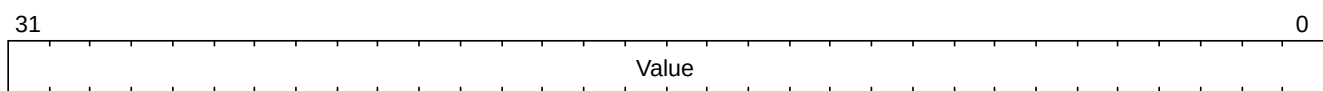
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	—

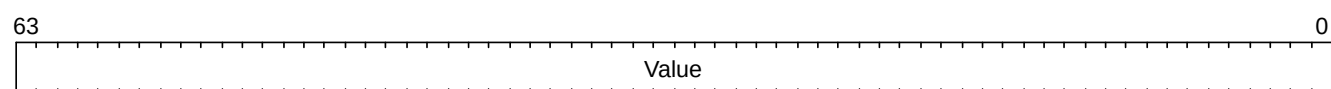
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	—

2.1.42. KScratch4

Encoding

coprocessor 0, register 31, select 5

Width

32 / 64 bits

Availability

AR >= R6 \|\| KSCRATCH

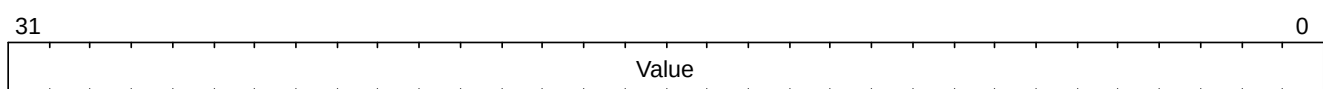
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	—

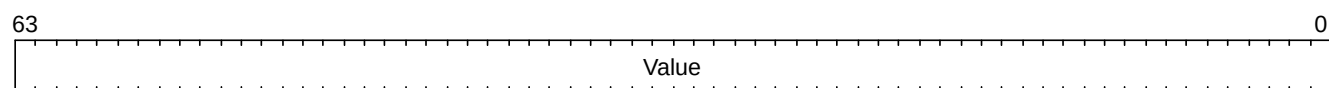
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	—

2.1.43. KScratch5

Encoding

coprocessor 0, register 31, select 6

Width

32 / 64 bits

Availability

AR >= R6 \\\| KSCRATCH

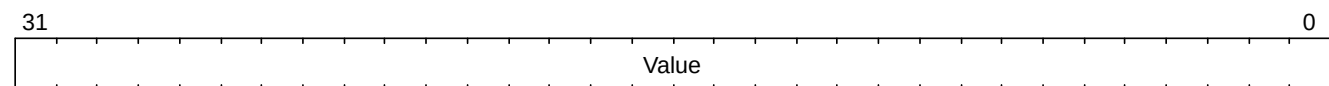
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	—

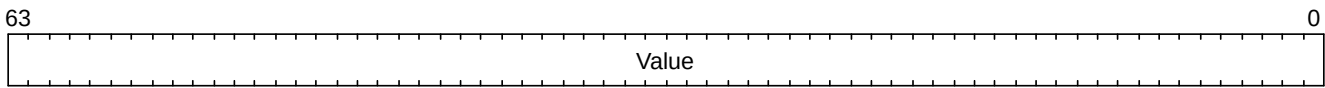
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	—

2.1.44. KScratch6

Encoding

coprocessor 0, register 31, select 7

Width

32 / 64 bits

Availability

AR >= R6 ||| KSCRATCH

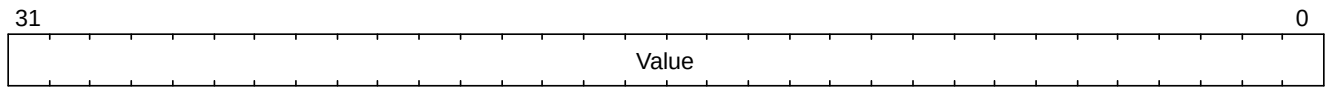
32-bit architecture

Width

32 bits

Availability

!(AT >= MIPS64_32)



Bits	Field	Access	Reset	Availability	Behavior	Description
31:0	Value	rw	0x0	true	—	—

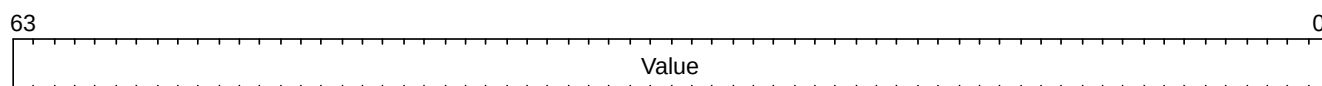
64-bit architecture

Width

64 bits

Availability

AT >= MIPS64_32



Bits	Field	Access	Reset	Availability	Behavior	Description
63:0	Value	rw	0x0	true	—	—

2.2. C1

Coprocessor

1

Availability

CP1

Register	Encoding	Width	Availability	Fields
FIR	0, 0	32	true	14
UFR	1, 0	32	ISA != COMPRESSED_ONLY && UFRP	2
UNFR	4, 0	32	ISA != COMPRESSED_ONLY && UFRP	2
FRE	5, 0	32	ISA != COMPRESSED_ONLY && FREP	2
NFRE	6, 0	32	ISA != COMPRESSED_ONLY && FREP	2
FCCR	25, 0	32	ISA != COMPRESSED_ONLY && AR < R6	9
FEXR	26, 0	32	true	16
FENR	28, 0	32	true	10
FCSR	31, 0	32	true	33

2.2.1. FIR

Encoding

coprocessor 1, register 0, select 0

Width

32 bits

Availability

true

31	30	29	28	27		24	23	22	21	20	19	18	17	16	15		8	7											0
Reserved	31	30	FREP	UFRP	Impl	HAS2008	F64	L	W	3D	PS	D	S	ProcessorID								Revision							

Bits	Field	Access	Reset	Availability	Behavior	Description
31:30	Reserved31_30	ro	0x0	true	—	—
29	FREP	ro	0x1 if FREP; 0x0 otherwise	true	—	User-mode FR emulation switching is implemented.
28	UFRP	ro	0x1 if UFRP; 0x0 otherwise	true	—	User-mode FR switching is implemented.
27:24	Impl	ro	0x0	true	—	—
23	HAS2008	ro	0x1 if AR >= R6 FP_2008; 0x0 otherwise	true	—	One or more IEEE-754-2008 features are implemented.
22	F64	ro	0x1 if FPU64; 0x0 otherwise	true	—	—
21	L	ro	0x1 if FPU64 FP_L; 0x0 otherwise	true	—	—
20	W	ro	0x1 if CP1; 0x0 otherwise	true	—	—
19	3D	ro	0x0	true	—	—
18	PS	ro	0x1 if FP_PS; 0x0 otherwise	true	—	—
17	D	ro	0x1 if FPU64 FP_D; 0x0 otherwise	true	—	Indicates whether 64-bit floating-point registers are implemented.
16	S	ro	0x1 if CP1; 0x0 otherwise	true	—	—
15:8	ProcessorID	ro	0x0	true	—	—
7:0	Revision	ro	0x0	true	—	—

D Values

Bits

17

Value	Name	Description
0	FP32	Floating-point registers are 32 bits wide.
1	FP64	Floating-point registers are 64 bits wide.

2.2.2. UFR

Encoding

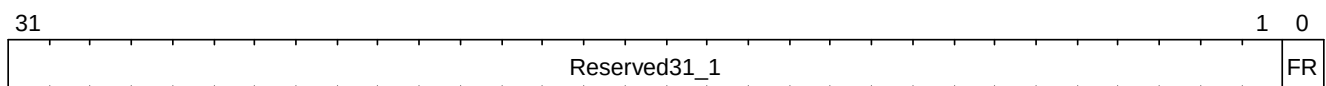
coprocessor 1, register 1, select 0

Width

32 bits

Availability

ISA := COMPRESSED_ONLY && UFRP



Bits	Field	Access	Reset	Availability	Behavior	Description
31:1	Reserved31_1	ro	0x0	true	—	—
0	FR	rw	0x0	true	on read: read_status_fr; on write: write_status_fr	User-mode alias used to read or clear C0.Status.FR.

2.2.3. UNFR

Encoding

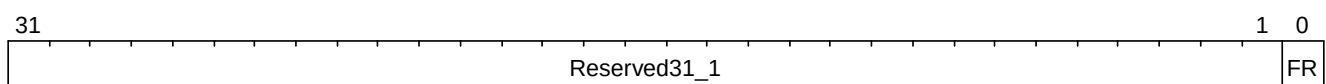
coprocessor 1, register 4, select 0

Width

32 bits

Availability

ISA != COMPRESSED_ONLY && UFRP



Bits	Field	Access	Reset	Availability	Behavior	Description
31:1	Reserved31_1	ro	0x0	true	—	—
0	FRE	ro	0x0	true	on read: read_config5_fre; on write: write_config5_fre	Negated legacy FR emulation mode.

2.2.6. FCCR

Encoding

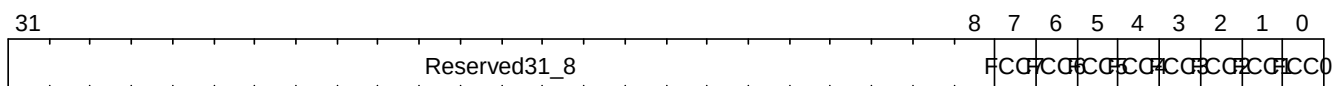
coprocessor 1, register 25, select 0

Width

32 bits

Availability

ISA != COMPRESSED_ONLY && AR < R6



Bits	Field	Access	Reset	Availability	Behavior	Description
31:8	Reserved31_8	ro	0x0	true	—	—
7	FCC7	rw	0x0	true	on read: read_fcsr_fcc7; on write: write_fcsr_fcc7	Alias to C1.FCSR.FCC7.
6	FCC6	rw	0x0	true	on read: read_fcsr_fcc6; on write: write_fcsr_fcc6	Alias to C1.FCSR.FCC6.
5	FCC5	rw	0x0	true	on read: read_fcsr_fcc5; on write: write_fcsr_fcc5	Alias to C1.FCSR.FCC5.
4	FCC4	rw	0x0	true	on read: read_fcsr_fcc4; on write: write_fcsr_fcc4	Alias to C1.FCSR.FCC4.

Bits	Field	Access	Reset	Availability	Behavior	Description
3	FCC3	rw	0x0	true	on read: read_fcsr_fcc3; on write: write_fcsr_fcc3	Alias to C1.FCSR.FCC3.
2	FCC2	rw	0x0	true	on read: read_fcsr_fcc2; on write: write_fcsr_fcc2	Alias to C1.FCSR.FCC2.
1	FCC1	rw	0x0	true	on read: read_fcsr_fcc1; on write: write_fcsr_fcc1	Alias to C1.FCSR.FCC1.
0	FCC0	rw	0x0	true	on read: read_fcsr_fcc0; on write: write_fcsr_fcc0	Alias to C1.FCSR.FCC0.

2.2.7. FEXR

Encoding

coprocessor 1, register 26, select 0

Width

32 bits

Availability

true

31	29	28	27	26	25	24	23	22		17	16		12	11	10	9	8	7	6		2	1	0
Reserved	31_CauseE	30_CauseE	29_CauseE	28_CauseE	27_CauseE	26_CauseE	25_CauseE	24_CauseE	Cause	Reserved	11_7	Flags	Flags	Flags	Flags	Flags	Flags	Flags	Flags	Flags	Reserved	1	

Bits	Field	Access	Reset	Availability	Behavior	Description
31:18	Reserved31_18	ro	0x0	true	—	—
17	CauseE	rw	0x0	true	on read: read_fcsr_causee; on write: write_fcsr_causee	Alias to C1.FCSR.CauseE.

Bits	Field	Access	Reset	Availability	Behavior	Description
17:12	Cause	rw	0x0	true	on read: read_fcsr_cause; on write: write_fcsr_cause	Alias to C1.FCSR.Cause.
16	CauseV	rw	0x0	true	on read: read_fcsr_causev; on write: write_fcsr_causev	Alias to C1.FCSR.CauseV.
15	CauseZ	rw	0x0	true	on read: read_fcsr_causez; on write: write_fcsr_causez	Alias to C1.FCSR.CauseZ.
14	CauseO	rw	0x0	true	on read: read_fcsr_causeo; on write: write_fcsr_causeo	Alias to C1.FCSR.CauseO.
13	CauseU	rw	0x0	true	on read: read_fcsr_causeu; on write: write_fcsr_causeu	Alias to C1.FCSR.CauseU.
12	CauseI	rw	0x0	true	on read: read_fcsr_causei; on write: write_fcsr_causei	Alias to C1.FCSR.CauseI.
11:7	Reserved11_7	ro	0x0	true	—	—
6	FlagsV	rw	0x0	true	on read: read_fcsr_flagsv; on write: write_fcsr_flagsv	Alias to C1.FCSR.FlagsV.
6:2	Flags	rw	0x0	true	on read: read_fcsr_flags; on write: write_fcsr_flags	—

Bits	Field	Access	Reset	Availability	Behavior	Description
11	Enables V	rw	0x0	true	on read: read_fcsr_enablesv; on write: write_fcsr_enablesv	Alias to C1.FCSR.EnablesV.
11:7	Enables	rw	0x0	true	on read: read_fcsr_enables; on write: write_fcsr_enables	Alias to C1.FCSR.Enables.
10	Enables Z	rw	0x0	true	on read: read_fcsr_enablesz; on write: write_fcsr_enablesz	Alias to C1.FCSR.EnablesZ.
9	Enables O	rw	0x0	true	on read: read_fcsr_enableso; on write: write_fcsr_enableso	Alias to C1.FCSR.EnablesO.
8	Enables U	rw	0x0	true	on read: read_fcsr_enablesu; on write: write_fcsr_enablesu	Alias to C1.FCSR.EnablesU.
7	Enables I	rw	0x0	true	on read: read_fcsr_enablesi; on write: write_fcsr_enablesi	Alias to C1.FCSR.EnablesI.
6:3	Reserved6_3	ro	0x0	true	—	—
2	FS	rw	0x0	true	on read: read_fcsr_fs; on write: write_fcsr_fs	—
1:0	RM	rw	0x0	true	on read: read_fcsr_rm; on write: write_fcsr_rm	—

2.2.9. FCSR

Encoding

coprocessor 1, register 31, select 0

Width

32 bits

Availability

true

31	30	29	28	27		22	21	20	19	18	17	16		12	11	10	9	8	7	6		2	1	0
Cause	Cause	Cause	Cause	Cause		Enable	Enable	Enable	Enable	Enable	Enable	Enable		Flags	Flags	Flags	Flags	Flags	Flags	Flags		Flags		RM

Bits	Field	Access	Reset	Availability	Behavior	Description
31	FCC7	rw	0x0	true	—	—
30	FCC6	rw	0x0	true	—	—
29	FCC5	rw	0x0	true	—	—
28	FCC4	rw	0x0	true	—	—
27	FCC3	rw	0x0	true	—	—
26	FCC2	rw	0x0	true	—	—
25	FCC1	rw	0x0	true	—	—
24	FS	rw	0x0	true	—	—
23	FCC0	rw	0x0	true	—	—
22:21	Impl	rw	0x0	true	—	—
20	Reserved20	ro	0x0	true	—	—
19	ABS2008	rw if AR >= R6 FP_2008; ro if AR < R6 && !(FP_2008)	0x1 if AR >= R6; 0x0 otherwise if AR >= R6 FP_2008; 0x0 if AR < R6 && !(FP_2008)	true	—	IEEE-754-2008 ABS/NEG NaN behavior.
18	NAN2008	rw if AR >= R6 FP_2008; ro if AR < R6 && !(FP_2008)	0x1 if AR >= R6; 0x0 otherwise if AR >= R6 FP_2008; 0x0 if AR < R6 && !(FP_2008)	true	—	IEEE-754-2008 NaN encoding.
17	CauseE	rw	0x0	true	—	—
17:12	Cause	rw	0x0	true	—	—

Bits	Field	Access	Reset	Availability	Behavior	Description
16	CauseV	rw	0x0	true	—	—
15	CauseZ	rw	0x0	true	—	—
14	Cause0	rw	0x0	true	—	—
13	CauseU	rw	0x0	true	—	—
12	CauseI	rw	0x0	true	—	—
11	EnablesV	rw	0x0	true	—	—
11:7	Enables	rw	0x0	true	—	—
10	EnablesZ	rw	0x0	true	—	—
9	Enables0	rw	0x0	true	—	—
8	EnablesU	rw	0x0	true	—	—
7	EnablesI	rw	0x0	true	—	—
6	FlagsV	rw	0x0	true	—	—
6:2	Flags	rw	0x0	true	—	—
5	FlagsZ	rw	0x0	true	—	—
4	Flags0	rw	0x0	true	—	—
3	FlagsU	rw	0x0	true	—	—
2	FlagsI	rw	0x0	true	—	—
1:0	RM	rw	0x0	true	—	—

Chapter 3. Privileged Architecture Algorithms

These explicitly catalogued functions are normative Python pseudocode for privileged state, address translation, memory access, and exception processing. Each function identifies the published PRA pseudocode or architectural section against which it is maintained. Processor scheduling, image loading, host services, and other simulator support are intentionally excluded.

3.1. Address Translation

3.1.1. AddressTranslation

Apply architectural segmentation, privilege, and TLB translation.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.4, 4.9.4 (Virtual Address Segments and Address Translation)

```
def AddressTranslation(
    ctx: ExecutionContext,
    va: int,
    access_type: MemoryAccessType,
    *,
    eva: bool = False,
) -> int:
    """Apply architectural segmentation, privilege, and TLB translation."""

    va = int(va)
    ksu = EffectiveKSU(ctx, eva)
    va_u = va & 0xFFFF_FFFF_FFFF_FFFF
    low_word = va_u & 0xFFFF_FFFF
    upper_word = va_u >> 32
    # The 32-bit compatibility decoder is selected for values representable
    # as signed 32-bit pointers. Sign-extended kseg addresses likewise use
    # compatibility segmentation; all other values enter the MIPS64 map.
    compatibility_address = (upper_word == 0 and low_word < 0x8000_0000) or (
        upper_word == 0xFFFF_FFFF and low_word >= 0x8000_0000
    )
    # In pre-Release-6 MIPS64, clearing the active KX/SX/UX bit selects the
    # 32-bit compatibility address type. A 32-bit value whose bit 31 is set
    # can still arrive zero-extended in a GPR (notably the U32 AVP cases), so
    # it must use the compatibility segments even though it is not a
    # canonical signed-32-bit pointer. R6 deliberately keeps such values in
    # XUSEG and reports the non-canonical address error instead.
    if (
        ctx.profile.AT is AT.MIPS64
        and not is_r6(ctx)
        and ksu == 2
        and upper_word == 0
    )
```

```

        and not ctx.C0.Status.UX
    ):
        compatibility_address = True
    # In Error Level the low compatibility useg is the direct-mapped
    # recovery segment. Otherwise a zero-extended low pointer selects XUSEG
    # when UX is enabled; this is distinct from a sign-extended 32-bit
    # compatibility pointer.
    if (
        ctx.profile.AT is AT.MIPS64
        and ctx.C0.Status.UX
        and upper_word == 0
        and not ctx.C0.Status.ERL
    ):
        compatibility_address = False
    # Sign-extended CKSEG2/CKSEG3 pointers use the kernel address-type
    # control, even when CKSEG2 is accessed from Supervisor Mode. With KX
    # set, a CKSEG2 miss uses the XTLB refill vector; CKSEG3 is still
    # inaccessible outside Kernel Mode.
    if (
        ctx.profile.AT is AT.MIPS64
        and upper_word == 0xFFFF_FFFF
        and low_word >= 0xC000_0000
    ):
        if ctx.C0.Status.KX:
            compatibility_address = False
    if ctx.profile.AT < AT.MIPS64_32 or compatibility_address:
        return _translate_compatibility_segment(ctx, va_u, access_type, ksu)

    if ctx.profile.AT is not AT.MIPS64:
        _raise_address_error(
            va,
            access_type,
            "processor profile does not implement extended MIPS64 segments",
        )

    region = va_u >> 62
    # CKSEG2/CKSEG3 pointers are sign-extended when passed through a 64-bit
    # GPR. In P64 mode they select XKSEG, but their implemented address bits
    # are still the low compatibility word; do not mistake the sign-extension
    # above bit 31 for an out-of-range 62-bit XKSEG offset.
    sign_extended_xkseg = (va_u >> 32) == 0xFFFF_FFFF and (
        va_u & 0xFFFF_FFFF
    ) >= 0xC000_0000
    sign_extended_xksseg = sign_extended_xkseg and (va_u & 0xFFFF_FFFF) <= 0xDFFF_FFFF
    region_address = (
        (va_u & 0xFFFF_FFFF) if sign_extended_xkseg else va_u & ((1 << 62) - 1)
    )
    if sign_extended_xksseg:
        if ksu > 1:
            _raise_address_error(va, access_type, "XKSSEG privilege error")
        return TLBMap(ctx, va_u, access_type, use_xtlb_vector=True)

```

```

if region == 0: # XUSEG
    if not ctx.C0.Status.UX:
        _raise_address_error(va, access_type, "XUSEG access requires Status.UX")
    if region_address >> ctx.profile.SEGBITS:
        _raise_address_error(
            va,
            access_type,
            "XUSEG address sets bits above SEGBITS",
        )
    bat_pa = avp_bat_translate(ctx, va, access_type)
    if bat_pa is not None:
        return bat_pa
    return TLBMap(ctx, va_u, access_type, use_xtlb_vector=True)

if region == 1: # XSSEG
    if not ctx.C0.Status.SX:
        _raise_address_error(va, access_type, "XSSEG access requires Status.SX")
    if ksu > 1:
        _raise_address_error(va, access_type, "XSSEG privilege error")
    if region_address >> ctx.profile.SEGBITS:
        _raise_address_error(
            va,
            access_type,
            "XSSEG address sets bits above SEGBITS",
        )
    return TLBMap(ctx, va_u, access_type, use_xtlb_vector=True)

if region == 2: # XKPHYS
    if not ctx.C0.Status.KX:
        _raise_address_error(va, access_type, "XKPHYS access requires Status.KX")
    if ksu > 0:
        _raise_address_error(va, access_type, "XKPHYS privilege error")
    physical_address = region_address & ((1 << 59) - 1)
    if physical_address >= 1 << ctx.profile.PABITS:
        _raise_address_error(va, access_type, "XKPHYS PA exceeds PABITS")
    return physical_address

# XKSEG has a two-gigabyte compatibility hole at the top of its
# SEGBITS-sized mapped range.
if not ctx.C0.Status.KX:
    _raise_address_error(va, access_type, "XKSEG access requires Status.KX")
if ksu > 0:
    _raise_address_error(va, access_type, "XKSEG privilege error")
if region_address >= (1 << ctx.profile.SEGBITS) - (1 << 31):
    _raise_address_error(va, access_type, "XKSEG address exceeds SEGBITS")
return TLBMap(ctx, va_u, access_type, use_xtlb_vector=True)

```

3.1.2. `get_asid`

Return the active address-space identifier for TLB operations.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.9.4 (Address Translation)

```
def get_asid(ctx: ExecutionContext) -> int:
    """Return the active address-space identifier for TLB operations."""

    try:
        if ctx.C0.Config5.MI:
            return ctx.C0.MemoryMapID.MMID
    except (UnavailableRegister, UnavailableRegisterField):
        pass
    try:
        if ctx.C0.Config4.AE:
            return (ctx.C0.EntryHi.ASIDX << 8) | ctx.C0.EntryHi.ASID
    except (UnavailableRegister, UnavailableRegisterField):
        pass
    return ctx.C0.EntryHi.ASID
```

3.1.3. got_tlb

Return whether Config.MT describes a standard TLB MMU.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.4.2 (TLB Lookup)

```
def got_tlb(ctx: ExecutionContext) -> bool:
    """Return whether Config.MT describes a standard TLB MMU."""

    return ctx.C0.Config.MT == 1
```

3.1.4. set_asid_from_tlb

Update the feature-selected TLB interface identifier after TLBR.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 9.5 (Translation Lookaside Buffer Instructions)

```
def set_asid_from_tlb(ctx: ExecutionContext, asid: int) -> None:
    """Update the feature-selected TLB interface identifier after TLBR."""

    try:
        if ctx.C0.Config5.MI:
            ctx.C0.MemoryMapID.MMID = int(asid) & 0xFFFF_FFFF
            return
    except (UnavailableRegister, UnavailableRegisterField):
        pass
    ctx.C0.EntryHi.ASID = int(asid) & 0xFF
```

3.1.5. tlb_exception

Apply TLB-specific CP0 updates, then signal the exception.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 6.2.14-6.2.17 (TLB Exceptions)

```
def tlb_exception(
    ctx: ExecutionContext,
    exception_type: type[TLBException],
    va: int,
    description: str,
    *,
    miss: bool = False,
    use_xtlb_vector: bool = False,
) -> None:
    """Apply TLB-specific CP0 updates, then signal the exception."""

    address_width = 64 if ctx.profile.AT >= AT.MIPS64_32 else 32
    va_u = int(va) & ((1 << address_width) - 1)
    ctx.C0.EntryHi.VPN2 = (va_u & ((1 << ctx.profile.SEGBITS) - 1)) >> 13
    if ctx.profile.AT >= AT.MIPS64_32:
        ctx.C0.EntryHi.R = (va_u >> 62) & 0x3
    ctx.C0.Context.BadVPN2 = (va_u >> 13) & 0x7FFF
    if ctx.profile.AT >= AT.MIPS64_32:
        # MIPS64 XContext preserves PTEBase and updates the region and
        # selected BadVPN2 bits on every TLB exception. This is the
        # get_c0_xcontext_value() algorithm described by avp-ng's privileged
        # architecture pseudocode. Keep the mask profile-driven so the same
        # model works for implementations with different SEGBITS values.
        xcontext = ctx.C0.XContext.read()
        mask = (1 << (ctx.profile.SEGBITS - 9)) - 0x10
        if mask:
            leading_zeroes = 64 - mask.bit_length()
            ptebase_mask = ((1 << 64) - 1) ^ ((1 << (66 - leading_zeroes)) - 1)
            region = ((va_u >> 62) & 0x3) << (64 - leading_zeroes)
            badvpn2 = va_u >> (leading_zeroes - (64 - ctx.profile.SEGBITS))
            ctx.C0.XContext.Value = (
                (xcontext & ptebase_mask) | region | (badvpn2 & mask)
            )
    ctx.C0.BadVAddr.Value = va_u
    # A miss in an extended (64-bit) virtual-address segment uses the
    # dedicated XTLB refill vector. The ordinary compatibility segments use
    # the TLB refill vector. Once EXL is already set both cases take the
    # general exception vector.
    if miss and not ctx.C0.Status.EXL:
        vector_offset = 0x80 if use_xtlb_vector else 0
    else:
        vector_offset = 0x180
    raise exception_type(va, description, vector_offset=vector_offset)
```

3.1.6. TLBMap

Translate a mapped VA through the standard paired-page TLB.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.4.2, 4.9.4 (TLB Lookup and Address Translation)

```
def TLBMap(
    ctx: ExecutionContext,
    va: int,
    access_type: MemoryAccessType,
    *,
    use_xtlb_vector: bool = False,
) -> int:
    """Translate a mapped VA through the standard paired-page TLB."""

    address_width = 64 if ctx.profile.AT >= AT.MIPS64_32 else 32
    va_u = int(va) & ((1 << address_width) - 1)
    asid = get_asid(ctx)
    i = TLBSearch(ctx, va_u, asid)

    exception_type = TLBStore if access_type == "Store" else TLBLoad
    if i is None:
        tlb_exception(
            ctx,
            exception_type,
            va,
            f"{access_type.lower()} address missed in the TLB",
            miss=True,
            use_xtlb_vector=use_xtlb_vector,
        )

    matching_entry = TLBEntries(ctx)[i]
    small_pages = 1 <= ctx.C0.Config.AR < 2 and bool(ctx.C0.Config3.SP)
    page_mask = matching_entry.Mask << 13
    if small_pages:
        page_mask |= matching_entry.MaskX << 11
    page_mask |= 0x7FF if small_pages else 0x1FFF
    even_odd_bit = page_mask ^ (page_mask >> 1)
    odd_page = bool(va_u & even_odd_bit)
    pfn = matching_entry.PFN1 if odd_page else matching_entry.PFN0
    valid = matching_entry.V1 if odd_page else matching_entry.V0
    dirty = matching_entry.D1 if odd_page else matching_entry.D0
    read_inhibit = matching_entry.RI1 if odd_page else matching_entry.RI0
    execute_inhibit = matching_entry.XI1 if odd_page else matching_entry.XI0
    if not valid:
        tlb_exception(
            ctx,
            exception_type,
            va,
```

```

        f"{access_type.lower()} address matched an invalid TLB page",
    )
    if access_type == "Store" and not dirty:
        tlb_exception(
            ctx,
            TLBModified,
            va,
            "store address matched a clean TLB page",
        )
    if access_type == "Load" and read_inhibit:
        exception = TLBReadInhibit if ctx.C0.PageGrain.IEC else TLBLoad
        tlb_exception(
            ctx,
            exception,
            va,
            "load address matched a read-inhibited TLB page",
        )
    if access_type == "Fetch" and execute_inhibit:
        exception = TLBExecuteInhibit if ctx.C0.PageGrain.IEC else TLBLoad
        tlb_exception(
            ctx,
            exception,
            va,
            "fetch address matched an execute-inhibited TLB page",
        )

    page_offset_mask = even_odd_bit - 1
    physical_page = (pfn << 12) & ~page_offset_mask
    physical_address = physical_page | (va_u & page_offset_mask)
    return physical_address & ((1 << ctx.profile.PABITS) - 1)

```

3.1.7. TLBMatch

Return whether one TLB entry matches a virtual address and ASID.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.4.2 (TLB Lookup)

```

def TLBMatch(ctx: ExecutionContext, entry: TLBEntry, va: int, asid: int) -> bool:
    """Return whether one TLB entry matches a virtual address and ASID."""

    return TLBVPMMatch(ctx, entry, va) and (entry.G or entry.ASID == asid)

```

3.1.8. TLBSearch

Return the index of the matching TLB entry, or no index on a miss.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.4.2 (TLB Lookup)

```
def TLBSearch(ctx: ExecutionContext, va: int, asid: int) -> int | None:
    """Return the index of the matching TLB entry, or no index on a miss."""

    TLB = TLBEntries(ctx)
    for i in range(len(TLB)):
        if TLBMatch(ctx, TLB[i], va, asid):
            return i
    return None
```

3.1.9. TLBVPNMatch

Return whether one valid TLB entry matches a virtual page number.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.4.2 (TLB Lookup)

```
def TLBVPNMatch(ctx: ExecutionContext, entry: TLBEntry, va: int) -> bool:
    """Return whether one valid TLB entry matches a virtual page number."""

    if entry.EHINV:
        return False
    # A MIPS64 processor still has the MIPS32-compatible address segments.
    # Their sign-extended addresses match TLB VPN2 using the low 32-bit
    # address, while the extended MIPS64 segments use the implementation's
    # SEGBITS width. Using SEGBITS for a compatibility address shifts the
    # sign-extension bits into VPN2 and turns an invalid page into a refill
    # miss (and therefore the wrong exception vector).
    va_u = int(va) & ((1 << 64) - 1)
    low_word = va_u & 0xFFFF_FFFF
    upper_word = va_u >> 32
    compatibility_address = (upper_word == 0 and low_word < 0x8000_0000) or (
        upper_word == 0xFFFF_FFFF and low_word >= 0x8000_0000
    )
    # Error Level retains the direct-mapped compatibility use; otherwise a
    # zero-extended low address selects XUSEG when UX is enabled.
    if (
        ctx.profile.AT is AT.MIPS64
        and ctx.C0.Status.UX
        and upper_word == 0
        and not ctx.C0.Status.ERL
    ):
        compatibility_address = False
    if (
        ctx.profile.AT is AT.MIPS64
        and upper_word == 0xFFFF_FFFF
        and low_word >= 0xC000_0000
    ):
        ksu = EffectiveKSU(ctx)
        p64 = (
```



```

        bool(ctx.C0.Status.KX)
        if ksu == 0
        else bool(ctx.C0.Status.SX)
        if ksu == 1
        else bool(ctx.C0.Status.UX)
    )
    if p64:
        compatibility_address = False
# EntryHi.R selects one of the extended MIPS64 regions. It is not part
# of matching a sign-extended MIPS32-compatible address, even when the
# processor implements the MIPS64 address type (AT=2). The VPN2 field
# itself is still the full implementation-width field: a compatibility
# address is sign-extended before matching, so high VPN2 bits distinguish
# (for example) a programmed VPN2 of 0x01000000 from VA 0x2c.
if ctx.C0.Config.AT == 2 and not compatibility_address:
    if entry.R != (va_u >> 62) & 0x3:
        return False
vpn_bits = ctx.profile.SEGBITS
vpn2 = (va_u & ((1 << vpn_bits) - 1)) >> 13
entry_vpn2 = entry.VPN2
entry_mask = entry.Mask
if (entry_vpn2 & ~entry_mask) != (vpn2 & ~entry_mask):
    return False
if 1 <= ctx.C0.Config.AR < 2 and ctx.C0.Config3.SP:
    return (entry.VPN2X & ~entry.MaskX) == (((va >> 11) & 0x3) & ~entry.MaskX)
return True

```

3.2. Addressing

3.2.1. P64_Mode

Derive pointer width from CP0 state, following `common_functions.md`.

Architecture reference: MIPS Architecture Base: 64-bit Privileged Resource Architecture, Revision 06.03, Section 4 (Virtual Memory)

```

def P64_Mode(
    ctx: ExecutionContext,
    access_type: AccessType | None = None,
    eva: bool = False,
) -> bool:
    """Derive pointer width from CP0 state, following common_functions.md."""

    if access_type not in {None, "Load", "Store", "Fetch", "Cacheop"}:
        raise ValueError(f"invalid P64 access type {access_type!r}")

    if ctx.C0.Config.AT < 2:
        return False

```

```
ksu = EffectiveKSU(ctx, eva)

if is_r6(ctx) or ctx.C0.Config5.EVA:
    return bool(
        (ksu == 0 and ctx.C0.Status.KX)
        or (ksu == 1 and ctx.C0.Status.SX)
        or (ksu == 2 and ctx.C0.Status.UX)
    )

if ksu == 2 and access_type in {"Load", "Store"} and not ctx.C0.Status.UX:
    return False
return True
```

3.3. Exception Processing

3.3.1. debug_exception

Enter debug mode and redirect to the architectural debug vector.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 6.2.4 (EJTAG Debug Exception)

```
def debug_exception(
    ctx: ExecutionContext,
    *,
    cause: int | None = None,
    epc: int | None = None,
    in_delay_slot: bool = False,
) -> int:
    """Enter debug mode and redirect to the architectural debug vector."""

    if cause is not None:
        ctx.C0.Debug.DExcCode = int(cause)
    restart = int(ctx.pc if epc is None else epc)
    ctx.C0.DEPC.Value = restart & ((1 << 64) - 1)
    ctx.C0.Debug.DBD = int(in_delay_slot)
    ctx.C0.Debug.DM = 1
    ctx.C0.Debug.DBp = 0
    ctx.next_pc = 0xBFC0_0480
    ctx.pending_control_transfer = None
    return ctx.next_pc
```

3.3.2. exception_entry

Record a general exception and redirect `next_pc` to its CP0 vector.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 6.2.3 (General Exception Processing)

```

def exception_entry(
    ctx: ExecutionContext,
    cause: str | int,
    *,
    epc: int,
    coprocessor: int = 0,
    badva: int | None = None,
    badinstr: int | None = None,
    badinstrx: int | None = None,
    badinstrp: int | None = None,
    offset: int | None = None,
    vector_number: int | None = None,
    in_delay_slot: bool = False,
    prior_pc: int | None = None,
) -> int:
    """Record a general exception and redirect ``next_pc`` to its CP0 vector.

    Instruction-boundary facts are explicit arguments because the
    architectural context does not need to retain decoded instructions.
    ``offset`` selects a type-specific vector; when omitted it is derived for
    ordinary exceptions and interrupts.
    """

    cause_code = EXCEPTION_CAUSE_CODES[cause] if isinstance(cause, str) else int(
        cause)
    if cause_code < 0 or cause_code >= 32:
        raise ValueError("Cause.ExcCode must fit five bits")
    if coprocessor < 0 or coprocessor >= 4:
        raise ValueError("Cause.CE must fit two bits")
    if in_delay_slot and prior_pc is None:
        raise ValueError("prior_pc is required for a delay-slot exception")

    address_mask = (1 << (32 if ctx.C0.Config.AT == 0 else 64)) - 1
    already_in_exception = bool(ctx.C0.Status.EXL)
    try:
        nanomips = ctx.C0.Config3.MMAR == 3
    except (UnavailableRegister, UnavailableRegisterField):
        nanomips = False

    if already_in_exception:
        offset = 0x180
    else:
        # nanoMIPS has no architectural branch delay slots. For other
        # families the processor boundary supplies the prior branch PC.
        in_delay_slot = bool(in_delay_slot and not nanomips)
        restart_pc = int(prior_pc) if in_delay_slot else int(epc)
        try:
            mode_in_epc = bool(ctx.C0.Config1.CA or ctx.C0.Config3.ISA > 1)
        except (UnavailableRegister, UnavailableRegisterField):
            mode_in_epc = bool(ctx.C0.Config1.CA)

```

```

if mode_in_epc:
    restart_pc = (restart_pc & ~1) | int(ctx.ISAMode)
ctx.C0.EPC.Value = restart_pc & address_mask
ctx.C0.Cause.BD = int(in_delay_slot)

if offset is None:
    if cause_code == EXCEPTION_CAUSE_CODES["INT"] and ctx.C0.Cause.IV:
        try:
            spacing = ctx.C0.IntCtl.VS << 5
        except (UnavailableRegister, UnavailableRegisterField):
            spacing = 0
        if not ctx.C0.Status.BEV and spacing:
            if vector_number is None:
                pending = (
                    (ctx.C0.Cause.IP7 << 7)
                    | (ctx.C0.Cause.IP1 << 1)
                    | ctx.C0.Cause.IP0
                )
                vector_number = max(
                    (bit for bit in range(8) if pending & (1 << bit)),
                    default=0,
                )
                offset = 0x200 + int(vector_number) * spacing
            else:
                offset = 0x200
        else:
            offset = 0x180

    try:
        if ctx.C0.Config.AR >= 1 and ctx.C0.SRSCtl.HSS and not ctx.C0.Status.BEV:
            new_shadow_set = ctx.C0.SRSCtl.ESS
            if (
                cause_code == EXCEPTION_CAUSE_CODES["INT"]
                and ctx.C0.Cause.IV
                and ctx.C0.IntCtl.VS
                and vector_number is not None
            ):
                new_shadow_set = getattr(
                    ctx.C0.SRSCtl, f"SSV{int(vector_number) & 7}"
                )
                new_shadow_set = min(new_shadow_set, ctx.C0.SRSCtl.HSS)
                ctx.C0.SRSCtl.PSS = ctx.C0.SRSCtl.CSS
                ctx.C0.SRSCtl.CSS = new_shadow_set
                ctx.GPR.select_set(new_shadow_set)
            except (UnavailableRegister, UnavailableRegisterField):
                pass

    ctx.C0.Cause.CE = coprocessor if cause_code == 11 else 0
    ctx.C0.Cause.ExcCode = cause_code
    if badva is not None:
        try:

```

```

        ctx.C0.BadVAddr.Value = int(badva) & address_mask
    except (UnavailableRegister, UnavailableRegisterField):
        pass
if badinstr is not None:
    try:
        ctx.C0.BadInstr.Instruction = int(badinstr) & 0xFFFF_FFFF
    except (UnavailableRegister, UnavailableRegisterField):
        pass
if badinstrx is not None:
    try:
        ctx.C0.BadInstrX.Instruction = int(badinstrx) & 0xFFFF_FFFF
    except (UnavailableRegister, UnavailableRegisterField):
        pass
if badinstrp is not None:
    try:
        ctx.C0.BadInstrP.Instruction = int(badinstrp) & 0xFFFF_FFFF
    except (UnavailableRegister, UnavailableRegisterField):
        pass

# MIPS16 exceptions select MIPS mode. Dual MIPS/microMIPS processors use
# Config3.ISAOnExc. A compressed-only processor retains its sole mode.
try:
    if ctx.C0.Config1.CA:
        ctx.ISAMode = ISAMode.MIPS
    elif ctx.C0.Config3.ISA > 1:
        ctx.ISAMode = ISAMode(ctx.C0.Config3.ISAOnExc)
except (UnavailableRegister, UnavailableRegisterField):
    pass

if ctx.C0.Status.BEV:
    vector_base = 0xBFC0_0200
elif ctx.C0.Config.AR == 0:
    vector_base = 0x8000_0000
else:
    # EBase.VA names a 32-bit compatibility-segment address even in a
    # MIPS64 processor. The architectural exception pseudocode first
    # forms that 32-bit address and sign-extends it; using the complete
    # register value as an unsigned address sends vectors such as
    # 0x800f0000 into XUSEG and immediately raises a second ADEL.
    vector_base = int(ctx.C0.EBase.read()) & 0xFFFF_F000

if address_mask != 0xFFFF_FFFF:
    vector_base = int(sign_extend(vector_base, from_nbits=32))
else:
    vector_base &= address_mask

ctx.C0.Status.EXL = 1
ctx.pending_control_transfer = None
assert offset is not None
ctx.next_pc = (vector_base & ~0x3FFF_FFFF) | (
    (vector_base + int(offset)) & 0x3FFF_FFFF

```

```
)
ctx.next_pc &= address_mask
return ctx.next_pc
```

3.3.3. exception_return

Perform ERET/ERETNC state restoration and redirect instruction fetch.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 9.5 (ERET and ERETNC)

```
def exception_return(ctx: ExecutionContext, *, no_clear_llbit: bool) -> None:
    """Perform ERET/ERETNC state restoration and redirect instruction fetch."""

    if no_clear_llbit and not ctx.C0.Config5.LLB:
        raise ReservedInstruction("ERETNC requires Config5.LLB")
    if not IsCoprocessor0Enabled(ctx):
        raise CoprocessorUnusable(0)

    if ctx.C0.Status.ERL:
        target = ctx.C0.ErrorEPC.read()
        ctx.C0.Status.ERL = 0
    else:
        target = ctx.C0.EPC.read()
        ctx.C0.Status.EXL = 0
        if "SRSCtl" in ctx.C0.definition.register_map:
            if ctx.C0.SRSCtl.HSS:
                previous_set = min(ctx.C0.SRSCtl.PSS, ctx.C0.SRSCtl.HSS)
                ctx.C0.SRSCtl.CSS = previous_set
                ctx.GPR.select_set(previous_set)

        # Compatibility-segment exception PCs are commonly written through the
        # 32-bit CP0 path even on MIPS64. Such a value is architecturally the
        # sign-extended kseg address; retaining a zero-extended 0x80000000..
        # value would send ERET into XUSEG and raise a recursive ADEL.
        if ctx.profile.AT >= AT.MIPS64_32 and 0x8000_0000 <= target <= 0xFFFF_FFFF:
            target = int(sign_extend(target, from_nbits=32))
        config1 = ctx.C0.definition.register_map.get("Config1")
        config3 = ctx.C0.definition.register_map.get("Config3")
        mode_in_epc = bool(
            config1 is not None and "CA" in config1.field_map and ctx.C0.Config1.CA
        )
        if config3 is not None and "ISA" in config3.field_map:
            mode_in_epc = mode_in_epc or ctx.C0.Config3.ISA > 1
        if mode_in_epc:
            ctx.ISAMode = ISAMode(target & 1)
            target &= ~1
        compact_branch(ctx, target)

    if not no_clear_llbit:
        SetLLBit(ctx, False)
```

```
ctx.linked_physical_address = None
ctx.linked_nbytes = 0
```

3.4. Operating Modes

3.4.1. EffectiveKSU

Return the effective kernel/supervisor/user operating mode.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 3.2 (Operating Modes)

```
def EffectiveKSU(ctx: ExecutionContext, eva: bool = False) -> int:
    """Return the effective kernel/supervisor/user operating mode."""

    if eva:
        return 2
    try:
        debug_mode = bool(ctx.C0.Debug.DM)
    except (AttributeError, KeyError):
        debug_mode = False
    if ctx.C0.Status.EXL or ctx.C0.Status.ERL or debug_mode:
        return 0
    return ctx.C0.Status.KSU
```

3.5. Shadow Register Sets

3.5.1. read_previous_gpr

Read one GPR from the set selected by SRSCtl.PSS.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 7.1 (Introduction to Shadow Sets)

```
def read_previous_gpr(ctx: ExecutionContext, register: int) -> int:
    """Read one GPR from the set selected by SRSCtl.PSS."""

    try:
        previous_set = ctx.C0.SRSCtl.PSS
    except (UnavailableRegister, UnavailableRegisterField):
        previous_set = ctx.GPR.active_set
    return ctx.GPR.read_set(previous_set, register)
```

3.5.2. write_previous_gpr

Write one GPR in the set selected by SRSCtl.PSS.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 7.1 (Introduction to Shadow Sets)

```
def write_previous_gpr(ctx: ExecutionContext, register: int, value: int) -> None:
    """Write one GPR in the set selected by SRSctl.PSS."""

    try:
        previous_set = ctx.C0.SRSctl.PSS
    except (UnavailableRegister, UnavailableRegisterField):
        previous_set = ctx.GPR.active_set
    ctx.GPR.write_set(previous_set, register, value)
```

3.6. TLB Operations

3.6.1. CheckTLBAccess

Check the architectural prerequisites for a privileged TLB operation.

Architecture reference: MIPS32 Instruction Set Manual, Revision 06.06, Section TLBP, TLBR, TLBWI, TLBWR (Translation Lookaside Buffer Operations)

```
def CheckTLBAccess(ctx: ExecutionContext) -> None:
    """Check the architectural prerequisites for a privileged TLB operation."""

    if not got_tlb(ctx):
        raise ReservedInstruction("the processor does not implement a TLB")
    if not IsCoproprocessor0Enabled(ctx):
        raise CoprocessorUnusable(0)
```

3.6.2. TLBEntries

Return the configured standard-TLB entry array.

Architecture reference: MIPS Architecture Base: 32-bit Privileged Resource Architecture, Revision 06.09, Section 4.4.2 (TLB Lookup)

```
def TLBEntries(ctx: ExecutionContext) -> list[TLBEntry]:
    """Return the configured standard-TLB entry array."""

    count = ctx.C0.Config1.MMUSize + 1
    if not ctx.tlb_entries:
        ctx.tlb_entries[:] = [TLBEntry() for _ in range(count)]
    return ctx.tlb_entries
```


3.6.3. `tlbinv`

Invalidate matching entries by ASID, or all entries for TLBINVF.

Architecture reference: MIPS32 Instruction Set Manual, Revision 06.06, Section TLBP, TLBR, TLBWI, TLBWR (Translation Lookaside Buffer Operations)

```
def tlbinv(ctx: ExecutionContext, *, flush: bool = False) -> None:
    """Invalidate matching entries by ASID, or all entries for TLBINVF."""

    if ctx.C0.Config4.IE < 2:
        raise ReservedInstruction("TLB invalidate is not implemented")
    CheckTLBAccess(ctx)
    asid = ctx.C0.EntryHi.ASID
    TLB = TLBEntries(ctx)
    for i in range(len(TLB)):
        if TLB[i].EHINV:
            continue
        if flush or (not TLB[i].G and TLB[i].ASID == asid):
            TLB[i] = TLBEntry()
```

3.6.4. `tlbp`

Probe for the current EntryHi VPN/ASID and update Index.

Architecture reference: MIPS32 Instruction Set Manual, Revision 06.06, Section TLBP, TLBR, TLBWI, TLBWR (Translation Lookaside Buffer Operations)

```
def tlbp(ctx: ExecutionContext) -> None:
    """Probe for the current EntryHi VPN/ASID and update Index."""

    CheckTLBAccess(ctx)
    va = ctx.C0.EntryHi.VPN2 << 13
    if 1 <= ctx.C0.Config.AR < 2 and ctx.C0.Config3.SP:
        va |= ctx.C0.EntryHi.VPN2X << 11
    if ctx.C0.Config.AT == 2:
        va |= ctx.C0.EntryHi.R << 62
    i = TLBSearch(ctx, va, get_asid(ctx))
    ctx.C0.Index.P = int(i is None)
    ctx.C0.Index.Index = 0 if i is None else i
```

3.6.5. `tlbr`

Read the indexed TLB entry into CP0's interface registers.

Architecture reference: MIPS32 Instruction Set Manual, Revision 06.06, Section TLBP, TLBR, TLBWI, TLBWR (Translation Lookaside Buffer Operations)

```

def tlbr(ctx: ExecutionContext) -> None:
    """Read the indexed TLB entry into CP0's interface registers."""

    CheckTLBAccess(ctx)
    i = ctx.C0.Index.Index
    TLB = TLBEntries(ctx)
    if i >= len(TLB):
        return

    ctx.C0.PageMask.Mask = TLB[i].Mask
    ctx.C0.EntryHi.VPN2 = TLB[i].VPN2
    ctx.C0.EntryLo1.PFN = TLB[i].PFN1 & 0xFF_FFFF
    ctx.C0.EntryLo1.C = TLB[i].C1
    ctx.C0.EntryLo1.D = TLB[i].D1
    ctx.C0.EntryLo1.V = TLB[i].V1
    ctx.C0.EntryLo1.G = TLB[i].G
    ctx.C0.EntryLo0.PFN = TLB[i].PFN0 & 0xFF_FFFF
    ctx.C0.EntryLo0.C = TLB[i].C0
    ctx.C0.EntryLo0.D = TLB[i].D0
    ctx.C0.EntryLo0.V = TLB[i].V0
    ctx.C0.EntryLo0.G = TLB[i].G

    if ctx.C0.Config.AR >= 1 and ctx.C0.Config4.IE >= 1:
        ctx.C0.EntryHi.EHINV = TLB[i].EHINV
    if ctx.C0.Config.AT == 2:
        ctx.C0.EntryHi.R = TLB[i].R
    if 1 <= ctx.C0.Config.AR < 2 and ctx.C0.Config3.SP:
        ctx.C0.PageMask.MaskX = TLB[i].MaskX
        ctx.C0.EntryHi.VPN2X = TLB[i].VPN2X
    if ctx.C0.Config.AR >= 1 and ctx.C0.Config3.RXI:
        if ctx.C0.PageGrain.ELPA:
            _write_entrylo_inhibit(ctx, 0, "RI", True, TLB[i].RI0)
            _write_entrylo_inhibit(ctx, 0, "XI", True, TLB[i].XI0)
            if _entrylo_inhibit_name(ctx, 0, "RI", True) != _entrylo_inhibit_name(
                ctx, 0, "RI", False
            ):
                _write_entrylo_inhibit(ctx, 0, "RI", False, 0)
            if _entrylo_inhibit_name(ctx, 0, "XI", True) != _entrylo_inhibit_name(
                ctx, 0, "XI", False
            ):
                _write_entrylo_inhibit(ctx, 0, "XI", False, 0)
            _write_entrylo_inhibit(ctx, 1, "RI", True, TLB[i].RI1)
            _write_entrylo_inhibit(ctx, 1, "XI", True, TLB[i].XI1)
            if _entrylo_inhibit_name(ctx, 1, "RI", True) != _entrylo_inhibit_name(
                ctx, 1, "RI", False
            ):
                _write_entrylo_inhibit(ctx, 1, "RI", False, 0)
            if _entrylo_inhibit_name(ctx, 1, "XI", True) != _entrylo_inhibit_name(
                ctx, 1, "XI", False
            ):
                _write_entrylo_inhibit(ctx, 1, "XI", False, 0)

```

```

        _write_entrylo_inhibit(ctx, 1, "XI", False, 0)
    else:
        _write_entrylo_inhibit(ctx, 0, "RI", False, TLB[i].RI0)
        _write_entrylo_inhibit(ctx, 0, "XI", False, TLB[i].XI0)
        _write_entrylo_inhibit(ctx, 1, "RI", False, TLB[i].RI1)
        _write_entrylo_inhibit(ctx, 1, "XI", False, TLB[i].XI1)
        if _entrylo_inhibit_name(ctx, 0, "RI", True) != _entrylo_inhibit_name(
            ctx, 0, "RI", False
        ):
            _write_entrylo_inhibit(ctx, 0, "RI", True, 0)
        if _entrylo_inhibit_name(ctx, 0, "XI", True) != _entrylo_inhibit_name(
            ctx, 0, "XI", False
        ):
            _write_entrylo_inhibit(ctx, 0, "XI", True, 0)
        if _entrylo_inhibit_name(ctx, 1, "RI", True) != _entrylo_inhibit_name(
            ctx, 1, "RI", False
        ):
            _write_entrylo_inhibit(ctx, 1, "RI", True, 0)
        if _entrylo_inhibit_name(ctx, 1, "XI", True) != _entrylo_inhibit_name(
            ctx, 1, "XI", False
        ):
            _write_entrylo_inhibit(ctx, 1, "XI", True, 0)
    if ctx.C0.Config3.LPA and ctx.C0.PageGrain.ELPA:
        ctx.C0.EntryLo0.PFNX = TLB[i].PFN0 >> 24
        ctx.C0.EntryLo1.PFNX = TLB[i].PFN1 >> 24

    if ctx.C0.Config.AR >= 2 and ctx.C0.Config5.MI:
        ctx.C0.MemoryMapID.MMID = TLB[i].ASID
    elif ctx.C0.Config.AR >= 1 and ctx.C0.Config4.AE:
        ctx.C0.EntryHi.ASID = TLB[i].ASID & 0xFF
        ctx.C0.EntryHi.ASIDX = TLB[i].ASID >> 8
    else:
        ctx.C0.EntryHi.ASID = TLB[i].ASID & 0xFF

```

3.6.6. tlbwi

Write CP0's TLB interface state to an indexed TLB entry.

Architecture reference: MIPS32 Instruction Set Manual, Revision 06.06, Section TLBP, TLBR, TLBWI, TLBWR (Translation Lookaside Buffer Operations)

```

def tlbwi(ctx: ExecutionContext, i: int | None = None) -> None:
    """Write CP0's TLB interface state to an indexed TLB entry."""

    CheckTLBAccess(ctx)
    if i is None:
        i = ctx.C0.Index.Index
    TLB = TLBEntries(ctx)
    if i >= len(TLB):
        return

```

```

TLB[i] = TLBEntry(EHINV=0)

if ctx.C0.Config.AR >= 1 and ctx.C0.Config4.IE >= 1:
    if ctx.C0.EntryHi.EHINV:
        TLB[i].EHINV = 1
        return

TLB[i].Mask = ctx.C0.PageMask.Mask
TLB[i].VPN2 = ctx.C0.EntryHi.VPN2
TLB[i].ASID = get_asid(ctx)
TLB[i].G = int(ctx.C0.EntryLo1.G and ctx.C0.EntryLo0.G)
TLB[i].PFN1 = ctx.C0.EntryLo1.PFN
TLB[i].C1 = ctx.C0.EntryLo1.C
TLB[i].D1 = ctx.C0.EntryLo1.D
TLB[i].V1 = ctx.C0.EntryLo1.V
TLB[i].PFN0 = ctx.C0.EntryLo0.PFN
TLB[i].C0 = ctx.C0.EntryLo0.C
TLB[i].D0 = ctx.C0.EntryLo0.D
TLB[i].V0 = ctx.C0.EntryLo0.V

if ctx.C0.Config.AT == 2:
    TLB[i].R = ctx.C0.EntryHi.R
if 1 <= ctx.C0.Config.AR < 2 and ctx.C0.Config3.SP:
    TLB[i].MaskX = ctx.C0.PageMask.MaskX
    TLB[i].VPN2X = ctx.C0.EntryHi.VPN2X
if ctx.C0.Config.AR >= 1 and ctx.C0.Config3.RXI:
    # ELPA selects the 64-bit RI/XI view. Without ELPA the ordinary
    # EntryLo bit-31/30 view is architecturally active.
    if ctx.C0.PageGrain.ELPA:
        TLB[i].RI0 = _entrylo_inhibit(ctx, 0, "RI", True)
        TLB[i].XI0 = _entrylo_inhibit(ctx, 0, "XI", True)
        TLB[i].RI1 = _entrylo_inhibit(ctx, 1, "RI", True)
        TLB[i].XI1 = _entrylo_inhibit(ctx, 1, "XI", True)
    else:
        TLB[i].RI0 = _entrylo_inhibit(ctx, 0, "RI", False)
        TLB[i].XI0 = _entrylo_inhibit(ctx, 0, "XI", False)
        TLB[i].RI1 = _entrylo_inhibit(ctx, 1, "RI", False)
        TLB[i].XI1 = _entrylo_inhibit(ctx, 1, "XI", False)
# ELPA selects the high physical-address portion on both MIPS64 and
# 32-bit XPA implementations. The latter uses MVH to program PFNX even
# though Config.AT remains the 32-bit encoding.
if ctx.C0.Config3.LPA and ctx.C0.PageGrain.ELPA:
    TLB[i].PFN0 |= ctx.C0.EntryLo0.PFNX << 24
    TLB[i].PFN1 |= ctx.C0.EntryLo1.PFNX << 24

```

3.6.7. tlbwr

Write the current entry through the implementation's random index.

Architecture reference: MIPS32 Instruction Set Manual, Revision 06.06, Section TLBP, TLBR,

```

def tlbwr(ctx: ExecutionContext) -> None:
    """Write the current entry through the implementation's random index."""

    CheckTLBAccess(ctx)
    TLB = TLBEntries(ctx)
    if not TLB:
        return
    if ctx.C0.Config.AR >= 2:
        wired = ctx.C0.Wired.Index
    else:
        wired = ctx.C0.Wired.Wired
    wired = min(wired, len(TLB))
    upper = len(TLB) - 1
    i = ctx.tlb_random_index
    if i is None or i < wired or i > upper:
        i = upper

    # Release 6 no longer exposes CP0 Random. Its TLBWR algorithm still
    # selects an implementation-dependent index, excluding Index.Index when
    # Index.P is clear. A descending round-robin is deterministic while
    # satisfying those architectural constraints.
    candidates = upper - wired + 1
    for _ in range(candidates):
        forbidden = (
            ctx.C0.Config.AR >= 2 and not ctx.C0.Index.P and i == ctx.C0.Index.Index
        )
        if not forbidden:
            break
        i = upper if i <= wired else i - 1
    else:
        return
    tlbwi(ctx, i)

    next_index = i - 1
    if next_index < wired:
        next_index = upper
    ctx.tlb_random_index = next_index
    if ctx.C0.Config.AR < 2:
        ctx.C0.Random.Random = next_index

```

Chapter 4. CP1 Floating-Point Architecture

CP1 instructions operate on architectural floating-point bit patterns. The helpers in this chapter define FPR access, floating-point control state, IEEE-754 results, and exception delivery shared by the instruction operations. Simulator scheduling and host floating-point state are not part of this specification.

4.1. Architectural state access

Function	Inputs and result	Architectural role
<code>validate_access_to_cp1_resources</code>	<code>validate_access_to_cp1_resources(ctx, *fmts: 'str', require_mips64: 'bool' = False) → 'None'</code>	Apply CU1, FPU implementation, format, and register-mode checks in architectural exception-precedence order.
<code>get_fpr</code>	<code>get_fpr(ctx, index: 'int', fmt: 'str') → 'int'</code>	Read an FPR using the architectural FR=0 pairing and format-width rules.
<code>set_fpr</code>	<code>set_fpr(ctx, index: 'int', value: 'int', fmt: 'str') → 'None'</code>	Write an FPR using the architectural FR=0 pairing and narrow-result rules.
<code>commit_fpu_flags</code>	<code>commit_fpu_flags(ctx, flags: 'int') → 'None'</code>	Accumulate IEEE exception flags and raise Floating Point when an asserted exception is enabled.
<code>read_fcc</code>	<code>read_fcc(ctx, cc: 'int') → 'int'</code>	Read one of the eight floating-point condition codes.
<code>write_fcc</code>	<code>write_fcc(ctx, cc: 'int', value: 'int') → 'None'</code>	Write one of the eight floating-point condition codes.

4.1.1. `validate_access_to_cp1_resources`

Apply CU1, FPU implementation, format, and register-mode checks in architectural exception-precedence order.

```
def validate_access_to_cp1_resources(
    ctx, *fmts: str, require_mips64: bool = False
) -> None:
    """Mirror validate_access_to_cp1_resources() from common CP1 pseudocode."""

    nano = ctx.profile.architecture is Architecture.NANOMIPS
    if not nano and not ctx.C0.Status.CU1:
        raise CoprocessorUnusable(1)

    if not ctx.C0.Config1.FP:
        raise ReservedInstruction("FPU is not implemented")

    for raw_fmt in fmts:
        fmt = normalize_format(raw_fmt)
```

```

if fmt == "D" and not ctx.C1.FIR.D:
    raise ReservedInstruction("D format is not supported")
if fmt == "S" and (not ctx.C1.FIR.S or ctx.C0.Config5.FRE):
    raise ReservedInstruction("S format is not supported")
if fmt == "W" and (not ctx.C1.FIR.W or ctx.C0.Config5.FRE):
    raise ReservedInstruction("W format is not supported")
if fmt == "L":
    if not ctx.C1.FIR.L or not ctx.C0.Status.FR:
        if not ctx.C1.FIR.L:
            raise ReservedInstruction("L format is not supported")
            raise ValueError("L format with Status.FR=0 is UNPREDICTABLE")
        if ctx.profile.AR < AR.R2_3_5 and not P64_Mode(ctx):
            raise ReservedInstruction("L format requires 64-bit operation mode")
if fmt == "PS":
    if not ctx.C1.FIR.PS or not ctx.C0.Status.FR:
        if not ctx.C1.FIR.PS:
            raise ReservedInstruction("PS format is not supported")
            raise ValueError("PS format with Status.FR=0 is UNPREDICTABLE")
        if ctx.profile.AR < AR.R2_3_5 and not P64_Mode(ctx):
            raise ReservedInstruction("PS format requires 64-bit operation mode")
if fmt == "64" and not ctx.C1.FIR.D:
    raise ReservedInstruction("64-bit format is not supported")
if fmt == "32" and ctx.C0.Config5.FRE:
    raise ReservedInstruction("32-bit format is not supported")
if require_mips64 and not P64_Mode(ctx):
    raise ReservedInstruction("64-bit operations are not enabled")
if ctx.C0.Status.CU1 == 0:
    raise CoprocessorUnusable(1)

```

4.1.2. get_fpr

Read an FPR using the architectural FR=0 pairing and format-width rules.

```

def get_fpr(ctx, index: int, fmt: str) -> int:
    """Read an FPR using the FR=0 even/odd pairing rules."""

    if not isinstance(index, int) or not 0 <= index < 32:
        raise ValueError(f"invalid FPR index {index!r}")
    if is_64bit_fpr_format(fmt) and not ctx.C0.Status.FR:
        if index & 1:
            raise ValueError("64-bit FPR access to an odd register with FR=0")
        return ((ctx.FPR[index + 1] & 0xFFFF_FFFF) << 32) | (
            ctx.FPR[index] & 0xFFFF_FFFF
        )
    return ctx.FPR[index] & ((1 << nbits_of(fmt)) - 1)

```

4.1.3. set_fpr

Write an FPR using the architectural FR=0 pairing and narrow-result rules.

```
def set_fpr(ctx, index: int, value: int, fmt: str) -> None:
    """Write an FPR using the set_fpr() upper-bit and pairing rules."""

    if not isinstance(index, int) or not 0 <= index < 32:
        raise ValueError(f"invalid FPR index {index!r}")
    value = int(value) & ((1 << nbits_of(fmt)) - 1)
    if is_64bit_fpr_format(fmt) and not ctx.C0.Status.FR:
        if index & 1:
            raise ValueError("64-bit FPR access to an odd register with FR=0")
        ctx.FPR[index] = value & 0xFFFF_FFFF
        ctx.FPR[index + 1] = value >> 32
        return
    if is_64bit_fpr_format(fmt):
        ctx.FPR[index] = value
    elif ctx.C1.FIR.F64:
        ctx.FPR[index] = (ctx.FPR[index] & 0xFFFF_FFFF_0000_0000) | value
    else:
        ctx.FPR[index] = value
```

4.1.4. commit_fpu_flags

Accumulate IEEE exception flags and raise Floating Point when an asserted exception is enabled.

```
def commit_fpu_flags(ctx, flags: int) -> None:
    """Commit IEEE flags for an operation whose result is not an FPR value."""

    cause = int(flags) & 0x3F
    ctx.C1.FCSR.Cause = cause
    if cause & (int(ctx.C1.FCSR.Enables) | int(IEEEFlags.UNIMPLEMENTED)):
        raise FloatingPointException(cause)
    ctx.C1.FCSR.Flags = int(ctx.C1.FCSR.Flags) | (cause & 0x1F)
```

4.1.5. read_fcc

Read one of the eight floating-point condition codes.

```
def read_fcc(ctx, cc: int) -> int:
    """Read floating-point condition code *cc* from FCSR."""

    cc = int(cc)
    if not 0 <= cc < 8:
        raise ValueError(f"invalid floating condition-code index {cc!r}")
    return int(
        (
```



```

        ctx.C1.FCSR.FCC0,
        ctx.C1.FCSR.FCC1,
        ctx.C1.FCSR.FCC2,
        ctx.C1.FCSR.FCC3,
        ctx.C1.FCSR.FCC4,
        ctx.C1.FCSR.FCC5,
        ctx.C1.FCSR.FCC6,
        ctx.C1.FCSR.FCC7,
    )[cc]
)

```

4.1.6. write_fcc

Write one of the eight floating-point condition codes.

```

def write_fcc(ctx, cc: int, value: int) -> None:
    """Write floating-point condition code *cc* in FCSR."""

    cc = int(cc)
    if not 0 <= cc < 8:
        raise ValueError(f"invalid floating condition-code index {cc!r}")
    value = int(bool(value))
    if cc == 0:
        ctx.C1.FCSR.FCC0 = value
    elif cc == 1:
        ctx.C1.FCSR.FCC1 = value
    elif cc == 2:
        ctx.C1.FCSR.FCC2 = value
    elif cc == 3:
        ctx.C1.FCSR.FCC3 = value
    elif cc == 4:
        ctx.C1.FCSR.FCC4 = value
    elif cc == 5:
        ctx.C1.FCSR.FCC5 = value
    elif cc == 6:
        ctx.C1.FCSR.FCC6 = value
    else:
        ctx.C1.FCSR.FCC7 = value

```

4.2. IEEE-754 primitives

These functions consume and return raw binary32 or binary64 encodings. A result also carries the Invalid Operation, Division by Zero, Overflow, Underflow, and Inexact conditions generated by the operation. Instruction operations commit those conditions to FCSR only after the result has been computed, so an enabled exception suppresses the destination write.

Primitive	Inputs and result	Architectural role
ieee_add	ieee_add(fmt: 'str', rounding_mode, fs: 'int', ft: 'int', *, subtract: 'bool' = False) → 'tuple[int, int]'	IEEE-754 addition or subtraction.
ieee_mul	ieee_mul(fmt: 'str', rounding_mode, fs: 'int', ft: 'int') → 'tuple[int, int]'	IEEE-754 multiplication.
ieee_div	ieee_div(fmt: 'str', rounding_mode, fs: 'int', ft: 'int') → 'tuple[int, int]'	IEEE-754 division.
ieee_sqrt	ieee_sqrt(fmt: 'str', rounding_mode, fs: 'int') → 'tuple[int, int]'	IEEE-754 square root.
ieee_recip	ieee_recip(fmt: 'str', rounding_mode, fs: 'int') → 'tuple[int, int]'	IEEE-754 reciprocal.
ieee_rsqrt	ieee_rsqrt(fmt: 'str', rounding_mode, fs: 'int') → 'tuple[int, int]'	IEEE-754 reciprocal square root.
ieee_madd	ieee_madd(fmt: 'str', rounding_mode, fd: 'int', fs: 'int', ft: 'int', *, subtract: 'bool' = False) → 'tuple[int, int]'	Fused multiply-add or multiply-subtract.
ieee_mul_add	ieee_mul_add(fmt: 'str', rounding_mode, fr: 'int', fs: 'int', ft: 'int', *, subtract: 'bool' = False) → 'tuple[int, int]'	Pre-Release-6 multiply-add or multiply-subtract with an independently rounded product.
ieee_negated_arithmetic_result	ieee_negated_arithmetic_result(fmt: 'str', rounding_mode, result: 'int') → 'int'	Negate a numeric NMADD or NMSUB result while preserving a NaN result.
ieee_minmax	ieee_minmax(fmt: 'str', rounding_mode, fs: 'int', ft: 'int', *, maximum: 'bool', absolute: 'bool' = False) → 'tuple[int, int]'	Release 6 minimum, maximum, minimum-magnitude, or maximum-magnitude selection.
ieee_rint	ieee_rint(fmt: 'str', rounding_mode, fs: 'int') → 'tuple[int, int]'	Round a floating-point value to an integral floating-point value.
ieee_float_to_float	ieee_float_to_float(dst_fmt: 'str', src_fmt: 'str', rounding_mode, fs: 'int') → 'tuple[int, int]'	Convert between binary32 and binary64 formats.
ieee_float_to_int	ieee_float_to_int(src_fmt: 'str', dst_fmt: 'str', rounding_mode, fs: 'int', *, rm: 'int None' = None) → 'tuple[int, int]'	Convert a floating-point operand to a W or L integer format.

Primitive	Inputs and result	Architectural role
ieee_int_to_float	ieee_int_to_float(dst_fmt: 'str', src_fmt: 'str', rounding_mode, fs: 'int') → 'tuple[int, int]'	Convert a W or L integer operand to a floating-point format.
ieee_compare	ieee_compare(fmt: 'str', rounding_mode, fs: 'int', ft: 'int', condition: 'str', *, signaling: 'bool' = False) → 'tuple[bool, int]'	Evaluate an architectural floating-point comparison condition.
ieee_class_mask	ieee_class_mask(fmt: 'str', rounding_mode, fs: 'int') → 'int'	Produce the Release 6 CLASS category mask.

Chapter 5. DSP Module Architecture

The DSP Module adds DSPControl state, four accumulator pairs, explicit resource-access checks, and packed integer primitives. These definitions describe architectural state and shared operation behavior; simulator scheduling, instruction dispatch, and host-language implementation details are not part of this chapter.

5.1. Architectural state

Function	Inputs and result	Architectural role
DSPControl.read	DSPControl.read(self, selector: 'int' = 63) → 'int'	Read the DSPControl fields selected by an RDDSP mask; reserved bits read as zero.
DSPControl.write	DSPControl.write(self, value: 'int', selector: 'int' = 63) → 'None'	Write the DSPControl fields selected by a WRDSP mask while preserving unselected fields.
DSPControl.set_overflow	DSPControl.set_overflow(self, flags: 'int') → 'None'	Set sticky DSP overflow indicators without clearing earlier events.
AccumulatorFile.read	AccumulatorFile.read(self, index: 'int') → 'int'	Read one complete architectural DSP accumulator as a bit vector.
AccumulatorFile.write	AccumulatorFile.write(self, index: 'int', value: 'int') → 'None'	Write one complete architectural DSP accumulator, masking it to the implemented width.

5.1.1. DSPControl.read

Read the DSPControl fields selected by an RDDSP mask; reserved bits read as zero.

```
def read(self, selector: int = 0x3F) -> int:
    """Return the fields selected by an RDDSP mask; other bits are zero."""

    selector = self._validate_selector(selector)
    selected = 0
    for specification in self.fields:
        if selector & (1 << specification.selector):
            selected |= self._value & specification.mask
    return selected
```

5.1.2. DSPControl.write

Write the DSPControl fields selected by a WRDSP mask while preserving unselected fields.

```
def write(self, value: int, selector: int = 0x3F) -> None:
    """Overwrite the fields selected by a WRDSP mask."""
```

```

selector = self._validate_selector(selector)
value = int(value) & 0xFFFF_FFFF
for specification in self.fields:
    if selector & (1 << specification.selector):
        self._write_field(
            specification,
            (value & specification.mask) >> specification.bits.low,
        )

```

5.1.3. DSPControl.set_overflow

Set sticky DSP overflow indicators without clearing earlier events.

```

def set_overflow(self, flags: int) -> None:
    """Set sticky bits in the Ouflag field without clearing prior events."""

    flags = int(flags)
    if flags < 0 or flags > 0xFF:
        raise ValueError("DSPControl overflow flags must fit eight bits")
    self.Ouflag |= flags

```

5.1.4. AccumulatorFile.read

Read one complete architectural DSP accumulator as a bit vector.

```

def read(self, index: int) -> int:
    index = self._validate_index(index)
    return (self._hi[index] << self.part_width) | self._lo[index]

```

5.1.5. AccumulatorFile.write

Write one complete architectural DSP accumulator, masking it to the implemented width.

```

def write(self, index: int, value: int) -> None:
    index = self._validate_index(index)
    value = int(value) & ((1 << self.width) - 1)
    self._hi[index] = (value >> self.part_width) & self.part_mask
    self._lo[index] = value & self.part_mask

```

5.2. Resource access

DSP instructions first check that the relevant module revision is implemented and that Status.MX permits DSP resource access. The two checks distinguish the original DSP Module from instructions introduced by DSP Module Revision 2.

Function	Inputs and result	Architectural role
<code>validate_access_to_dsp_resources</code>	<code>validate_access_to_dsp_resources(ctx: 'ExecutionContext') → 'None'</code>	Check DSP implementation and Status.MX before an instruction accesses DSP resources.
<code>validate_access_to_dsp2_resources</code>	<code>validate_access_to_dsp2_resources(ctx: 'ExecutionContext') → 'None'</code>	Check DSP Module Revision 2 implementation and Status.MX before a DSP2 instruction accesses resources.

5.2.1. `validate_access_to_dsp_resources`

Check DSP implementation and Status.MX before an instruction accesses DSP resources.

```
def validate_access_to_dsp_resources(ctx: ExecutionContext) -> None:
    """Raise the exception selected for an unavailable DSP Module resource."""

    if ctx.C0.Config3.DSPP == 0:
        raise ReservedInstruction("the DSP Module is not implemented")
    if ctx.C0.Status.MX == 0:
        raise DSPDisabled("DSP Module resource access is disabled")
```

5.2.2. `validate_access_to_dsp2_resources`

Check DSP Module Revision 2 implementation and Status.MX before a DSP2 instruction accesses resources.

```
def validate_access_to_dsp2_resources(ctx: ExecutionContext) -> None:
    """Validate access to an instruction added by DSP Module Revision 2."""

    if ctx.C0.Config3.DSPP == 0 or ctx.C0.Config3.DSP2P == 0:
        raise ReservedInstruction("DSP Module Revision 2 is not implemented")
    if ctx.C0.Status.MX == 0:
        raise DSPDisabled("DSP Module resource access is disabled")
```

5.3. Shared DSP primitives

These pure helpers operate on architectural bit vectors, packed lanes, and integer products. Instruction definitions use them to express lane order, saturation, rounding, and overflow behavior without importing simulator-only machinery.

Primitive	Inputs and result	Architectural role
<code>signed</code>	<code>signed(value: 'int', width: 'int') → 'int'</code>	Interpret a bit vector as a two's-complement signed value.
<code>lanes</code>	<code>lanes(value: 'int', width: 'int', count: 'int') → 'tuple[int, ...]'</code>	Split a packed vector into most-significant-lane-first elements.

Primitive	Inputs and result	Architectural role
<code>pack_lanes</code>	<code>pack_lanes(values: 'tuple[int, ...]', width: 'int') → 'int'</code>	Pack lane values into a register-sized bit vector.
<code>signed_add</code>	<code>signed_add(a: 'int', b: 'int', width: 'int', saturate: 'bool' = False) → 'tuple[int, bool]'</code>	Add signed lanes and report signed overflow.
<code>unsigned_add</code>	<code>unsigned_add(a: 'int', b: 'int', width: 'int', saturate: 'bool' = False) → 'tuple[int, bool]'</code>	Add unsigned lanes and report unsigned overflow.
<code>signed_sub</code>	<code>signed_sub(a: 'int', b: 'int', width: 'int', saturate: 'bool' = False) → 'tuple[int, bool]'</code>	Subtract signed lanes and report signed overflow.
<code>unsigned_sub</code>	<code>unsigned_sub(a: 'int', b: 'int', width: 'int', saturate: 'bool' = False) → 'tuple[int, bool]'</code>	Subtract unsigned lanes and report unsigned underflow.
<code>packed_abs</code>	<code>packed_abs(value: 'int', width: 'int', count: 'int') → 'int'</code>	Compute saturating absolute values independently in packed lanes.
<code>packed_signed_add</code>	<code>packed_signed_add(left: 'int', right: 'int', width: 'int', count: 'int', *, subtract: 'bool' = False, saturate: 'bool' = False, half: 'bool' = False, rounded: 'bool' = False) → 'tuple[int, bool]'</code>	Apply signed packed add, subtract, half, and saturating variants.
<code>packed_unsigned_add</code>	<code>packed_unsigned_add(left: 'int', right: 'int', width: 'int', count: 'int', *, subtract: 'bool' = False, saturate: 'bool' = False, half: 'bool' = False, rounded: 'bool' = False) → 'tuple[int, bool]'</code>	Apply unsigned packed add, subtract, half, and saturating variants.
<code>packed_compare</code>	<code>packed_compare(left: 'int', right: 'int', width: 'int', count: 'int', relation: 'str', *, signed_values: 'bool' = False) → 'int'</code>	Compare corresponding packed lanes and return the architectural condition mask.
<code>shift_left</code>	<code>shift_left(value: 'int', width: 'int', amount: 'int', *, saturate: 'bool' = False) → 'tuple[int, bool]'</code>	Shift packed lanes left and report lane overflow.
<code>shift_right</code>	<code>shift_right(value: 'int', width: 'int', amount: 'int', *, rounded: 'bool' = False, logical: 'bool' = False) → 'int'</code>	Shift packed lanes arithmetically or logically, with optional rounding.
<code>product_q15</code>	<code>product_q15(a: 'int', b: 'int') → 'tuple[int, bool]'</code>	Compute a doubled signed Q15 product and its saturation condition.
<code>product_q31</code>	<code>product_q31(a: 'int', b: 'int') → 'tuple[int, bool]'</code>	Compute a doubled signed Q31 product and its saturation condition.

Primitive	Inputs and result	Architectural role
sat_signed	sat_signed(value: 'int', width: 'int') → 'tuple[int, bool]'	Saturate an integer to a signed bit width.
sat_unsigned	sat_unsigned(value: 'int', width: 'int') → 'tuple[int, bool]'	Saturate an integer to an unsigned bit width.
q15_pair	q15_pair(a: 'int', b: 'int', rounded: 'bool' = False, saturating: 'bool' = False) → 'tuple[int, bool]'	Compute a signed Q15 pair product with optional rounding or saturation.
mulq_word	mulq_word(rs: 'int', rt: 'int', rounded: 'bool') → 'tuple[int, bool]'	Compute the high word of a signed Q31 multiply.

Chapter 6. Shared Architectural Algorithms

These pure functions define algorithms shared by multiple instructions. They depend only on their explicit inputs, making them directly usable as reference models in simulators, verification environments, and RTL testbenches. Stateful behavior is specified separately in the Shared Architectural Behavior chapter.

6.1. Algorithm summary

Algorithm	Inputs and result	Architectural role
<code>crc32</code>	<code>crc32(value: 'int', message: 'int', nbits: 'int', poly: 'int') → 'int'</code>	Compute the reflected CRC operation on right-justified bit vectors.
<code>bitswap_word</code>	<code>bitswap_word(value: 'int') → 'int'</code>	Reverse the bit order independently in each byte of a word.
<code>save_restore_register</code>	<code>save_restore_register(rt: 'int', index: 'int', gp: 'bool', count: 'int') → 'int'</code>	Return the architectural GPR selected by a register-list element.
<code>count_leading_bits</code>	<code>count_leading_bits(value: 'int', *, nbits: 'int', bit: 'int') → 'int'</code>	Count consecutive bit values from the most-significant position.

6.2. `crc32`

Compute the reflected CRC operation on right-justified bit vectors.

```
def crc32(value: int, message: int, nbits: int, poly: int) -> int:
    """Compute the reflected CRC operation on right-justified bit vectors."""

    if nbits <= 0:
        raise ValueError("CRC message width must be positive")
    mask = (1 << nbits) - 1
    result = (value & 0xFFFF_FFFF) ^ (message & mask)
    for _ in range(nbits):
        if result & 1:
            result = (result >> 1) ^ poly
        else:
            result >>= 1
    return result & 0xFFFF_FFFF
```

6.3. `bitswap_word`

Reverse the bit order independently in each byte of a word.

```
def bitswap_word(value: int) -> int:
```

```
"""Reverse the bit order independently in each byte of a word."""
```

```
result = 0
for shift in range(0, 32, 8):
    byte = (value >> shift) & 0xFF
    byte = ((byte & 0x55) << 1) | ((byte >> 1) & 0x55)
    byte = ((byte & 0x33) << 2) | ((byte >> 2) & 0x33)
    byte = ((byte & 0x0F) << 4) | ((byte >> 4) & 0x0F)
    result |= byte << shift
return result
```

6.4. save_restore_register

Return the architectural GPR selected by a register-list element.

```
def save_restore_register(rt: int, index: int, gp: bool, count: int) -> int:
    """Return the architectural GPR selected by a register-list element."""

    if gp and index + 1 == count:
        return 28
    return rt + index if rt + index < 32 else rt + index - 16
```

6.5. count_leading_bits

Count consecutive bit values from the most-significant position.

```
def count_leading_bits(value: int, *, nbits: int, bit: int) -> int:
    """Count consecutive bit values from the most-significant position."""
    source = value & ((1 << nbits) - 1)
    count = 0
    for position in range(nbits - 1, -1, -1):
        if ((source >> position) & 1) != bit:
            break
        count += 1
    return count
```

Chapter 7. Instruction Encoding and Decode

Each instruction format defines one bidirectional relationship between instruction bits and architectural operands. Decode first selects a format from fixed bits, opcode-pool entries, availability, and processor state; it then converts the named encoding fields into the operands used by Operation.

Immediate operands explicitly state sign or zero extension so the result is unambiguous for 32-bit, 64-bit, and wider arithmetic pipelines. Compressed register fields use the tables below. The inverse mappings are used for assembly encoding, so assemblers, disassemblers, simulators, verification models, and hardware decoders share the same definition.

7.1. Decode result

In addition to the operands consumed by Operation, Decode classifies control-transfer behavior. All three flags are `False` unless the Decode block explicitly sets a flag to `True`.

Result	Architectural meaning
<code>operands</code>	Named register, immediate, address, and selector values consumed by Operation.
<code>exec.isCTI</code>	The instruction is a control-transfer instruction.
<code>exec.hasDelaySlot</code>	The instruction executes the following instruction as a delay slot.
<code>exec.hasForbiddenSlot</code>	The instruction creates a forbidden slot after the control transfer.

7.2. Reversible field mappings

These forms describe how a semantic operand corresponds to encoded fields. The mappings are applied in declaration order, which permits a later operand to refer to an earlier one.

Specification form	Decode	Encode
<code>unsigned(field)</code>	Use the field as a non-negative integer.	Require the value to fit the field width.
<code>signed(field)</code>	Sign-extend from the field width.	Require a signed value of that width and store its two's-complement bit pattern.
<code>gpr(field)</code>	Use the field value as GPR index 0..31.	Store the GPR index directly.
<code>gpr(field, table)</code>	Map the encoded value through the named compressed register table.	Store the table index for the requested GPR.
<code>fpr(field)</code>	Use the field value as FPR index 0..31.	Store the FPR index directly.
<code>constant(value)</code>	Produce the implicit value without consuming a field.	Accept only that value and emit no field bits.

Specification form	Decode	Encode
<code>zero_means_maximum(field, width)</code>	Interpret zero as 2^{width} ; other values are unchanged.	Encode 2^{width} as zero.
<code>same_as(operand)</code>	Reuse an earlier operand.	Require the same value and emit no additional field bits.
<code>negated(mapping)</code>	Negate the value produced by the enclosed mapping.	Negate before applying the enclosed mapping.
<code>biased(mapping, bias)</code>	Add <code>bias</code> to the decoded value.	Subtract <code>bias</code> before encoding.

7.3. Compressed-register encodings

Each named format has its own mapping table. The encoded value is the value held in the instruction field; the architectural register is the GPR selected by that value.

7.3.1. GPR1

Field width

1 bit

Encoded value	Architectural register
0	GPR4
1	GPR5

7.3.2. GPR2_REG1

Field width

2 bits

Encoded value	Architectural register
0	GPR4
1	GPR5
2	GPR6
3	GPR7

7.3.3. GPR2_REG2

Field width

2 bits

Encoded value	Architectural register
0	GPR5

Encoded value	Architectural register
1	GPR6
2	GPR7
3	GPR8

7.3.4. GPR3

Field width

3 bits

Encoded value	Architectural register
0	GPR16
1	GPR17
2	GPR18
3	GPR19
4	GPR4
5	GPR5
6	GPR6
7	GPR7

7.3.5. GPR3_STORE

Field width

3 bits

Encoded value	Architectural register
0	GPR0
1	GPR17
2	GPR18
3	GPR19
4	GPR4
5	GPR5
6	GPR6
7	GPR7

7.3.6. GPR4

Field width

4 bits

Encoded value	Architectural register
0	GPR8

Encoded value	Architectural register
1	GPR9
2	GPR10
3	GPR11
4	GPR4
5	GPR5
6	GPR6
7	GPR7
8	GPR16
9	GPR17
10	GPR18
11	GPR19
12	GPR20
13	GPR21
14	GPR22
15	GPR23

7.3.7. GPR4_ZERO

Field width

4 bits

Encoded value	Architectural register
0	GPR8
1	GPR9
2	GPR10
3	GPR0
4	GPR4
5	GPR5
6	GPR6
7	GPR7
8	GPR16
9	GPR17
10	GPR18
11	GPR19
12	GPR20
13	GPR21
14	GPR22
15	GPR23

7.3.8. MICROMIPS_GPR3

Field width

3 bits

Encoded value	Architectural register
0	GPR16
1	GPR17
2	GPR2
3	GPR3
4	GPR4
5	GPR5
6	GPR6
7	GPR7

7.3.9. MICROMIPS_GPR3_STORE

Field width

3 bits

Encoded value	Architectural register
0	GPR0
1	GPR17
2	GPR2
3	GPR3
4	GPR4
5	GPR5
6	GPR6
7	GPR7

7.3.10. MIPS16_GPR3

Field width

3 bits

Encoded value	Architectural register
0	GPR16
1	GPR17
2	GPR2
3	GPR3
4	GPR4
5	GPR5

Encoded value	Architectural register
6	GPR6
7	GPR7

Chapter 8. Instruction Definitions

This part contains 622 instruction definitions in canonical-name order.

8.1. ABS.D

8.1.1. Purpose

Compute the absolute value of a D-format value.

8.1.2. Formats

MIPS

abs.d

Assembly

ABS.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

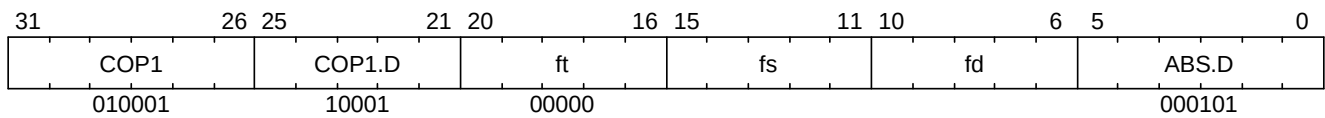


Figure 1. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

abs.d

Assembly

ABS.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

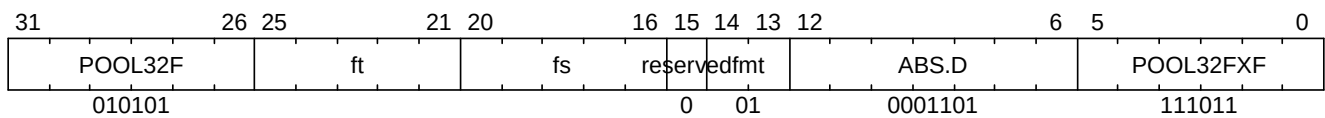


Figure 2. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

abs.d

Assembly

ABS.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

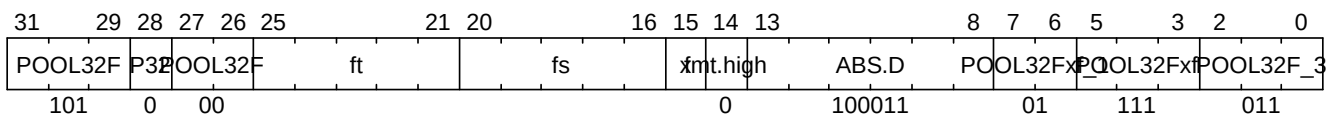


Figure 3. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.1.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_abs(
    "D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.1.4. Description

ABS.D validates CP1 and D-format support before updating the destination FPR.

8.1.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.2. ABS.PS

8.2.1. Purpose

Perform the paired-single abs operation.

8.2.2. Formats

MIPS

abs.ps

Assembly

ABS.PS {fd:fpr}, {fs:fpr}

Availability

AR < R6

Decode priority

0

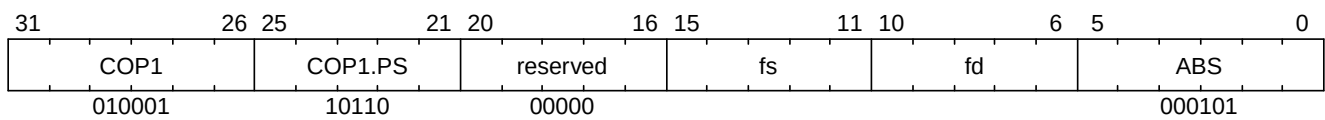


Figure 4. Encoding (32 bits)

Decode

```
return FpuPSUnaryOperands(fd=fd, fs=fs)
```

microMIPS

abs.ps

Assembly

ABS.PS {fd:fpr}, {fs:fpr}

Availability

MMAR < R6

Decode priority

0

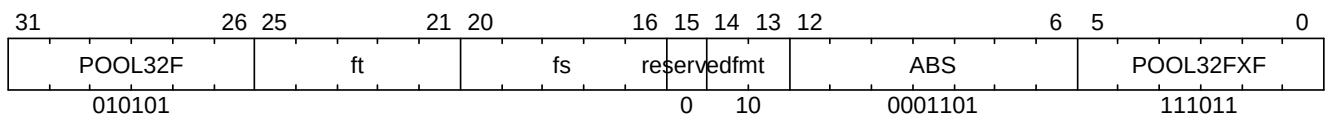


Figure 5. Encoding (32 bits)

```
return FpuPSUnaryOperands(fd=ft, fs=fs)
```

8.2.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
source = get_fpr(ctx, operands.fs, "PS")
low, low_flags = ieee_abs("S", rounding, source & 0xFFFF_FFFF)
high, high_flags = ieee_abs("S", rounding, source >> 32)
write_fpu_result(ctx, operands.fd, low | (high << 32), low_flags | high_flags, "PS")
```

8.2.4. Description

ABS.PS applies independently to the upper and lower S-format lanes.

8.2.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.3. ABS.S

8.3.1. Purpose

Compute the absolute value of an S-format value.

8.3.2. Formats

MIPS

abs.s

Assembly

ABS.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

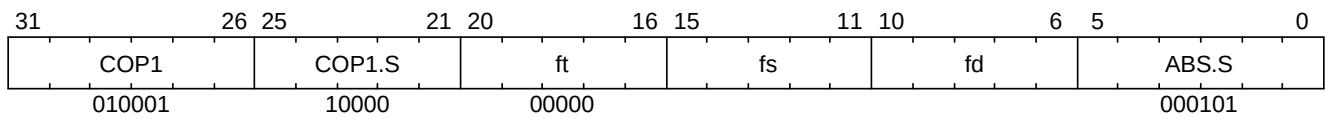


Figure 6. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

abs.s

Assembly

ABS.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

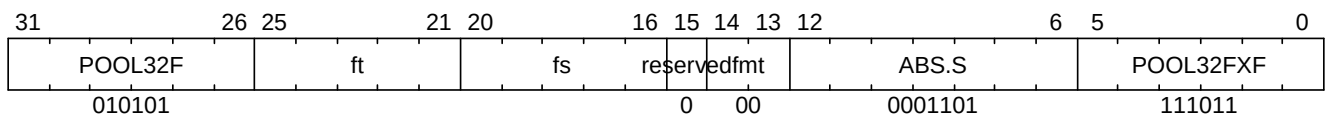


Figure 7. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

abs.s

Assembly

ABS.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

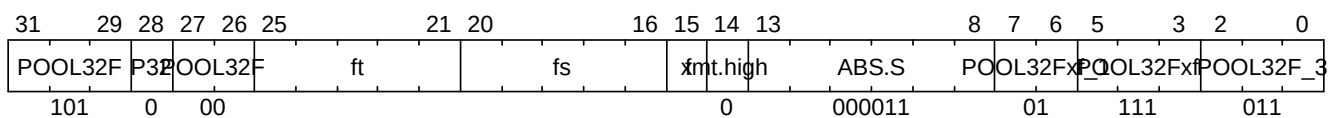


Figure 8. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.3.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_abs(
    "S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.3.4. Description

ABS.S validates CP1 and S-format support before updating the destination FPR.

8.3.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.4. ABSQ_S.PH

8.4.1. Purpose

DSP packed operation ABSQ_S.PH.

8.4.2. Formats

MIPS

absq_s.ph

Assembly

ABSQ_S.PH {rd:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

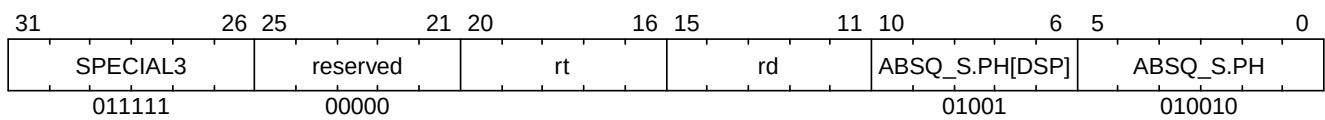


Figure 9. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rd, rs=rt)
```

microMIPS

absq_s.ph.32

Assembly

ABSQ_S.PH {rd:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

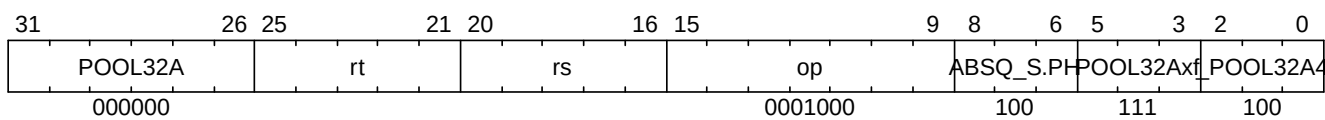


Figure 10. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

nanoMIPS

absq_s.ph.32

Assembly

ABSQ_S.PH {rd:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

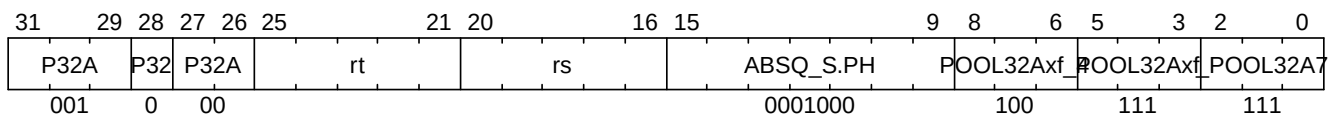


Figure 11. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

8.4.3. Operation

```
validate_access_to_dsp_resources(ctx)
if any(
    signed(lane, 16) == -(1 << 15) for lane in lanes(ctx.GPR[operands.rs], 16, 2)
):
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(
    packed_abs(ctx.GPR[operands.rs], 16, 2), from_nbits=32
)
```

8.4.4. Description

ABSQ_S.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.4.5. Exceptions

None.

8.5. ABSQ_S.QB

8.5.1. Purpose

DSP packed operation ABSQ_S.QB.

8.5.2. Formats

MIPS

absq_s.qb

Assembly

ABSQ_S.QB {rd:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

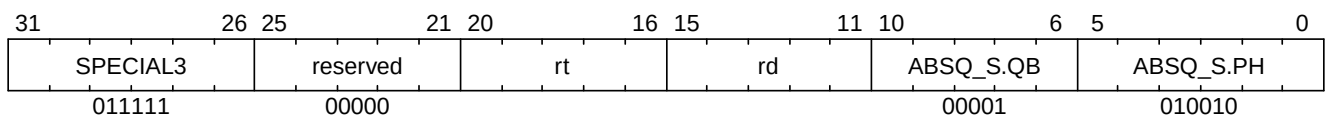


Figure 12. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rd, rs=rt)
```

microMIPS

absq_s.qb.32

Assembly

ABSQ_S.QB {rd:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

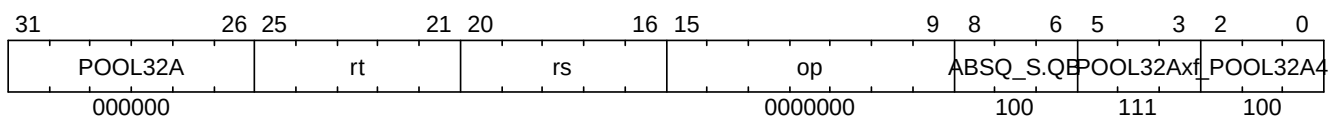


Figure 13. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

nanoMIPS

absq_s.qb.32

Assembly

ABSQ_S.QB {rd:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

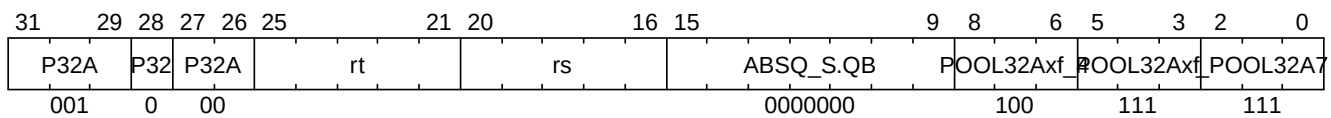


Figure 14. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

8.5.3. Operation

```
validate_access_to_dsp2_resources(ctx)
if any(signed(lane, 8) == -(1 << 7) for lane in lanes(ctx.GPR[operands.rs], 8, 4)):
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(
    packed_abs(ctx.GPR[operands.rs], 8, 4), from_nbits=32
)
```

8.5.4. Description

ABSQ_S.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.5.5. Exceptions

None.

8.6. ABSQ_S.W

8.6.1. Purpose

DSP packed operation ABSQ_S.W.

8.6.2. Formats

MIPS

absq_s.w

Assembly

ABSQ_S.W {rd:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

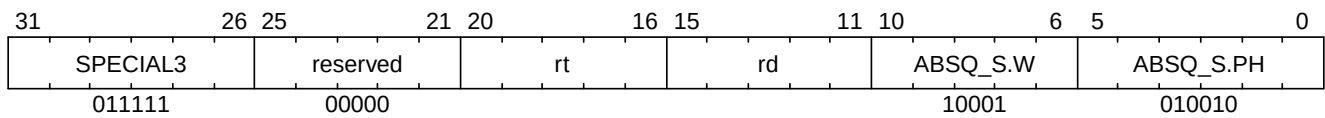


Figure 15. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rd, rs=rt)
```

microMIPS

absq_s.w.32

Assembly

ABSQ_S.W {rd:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

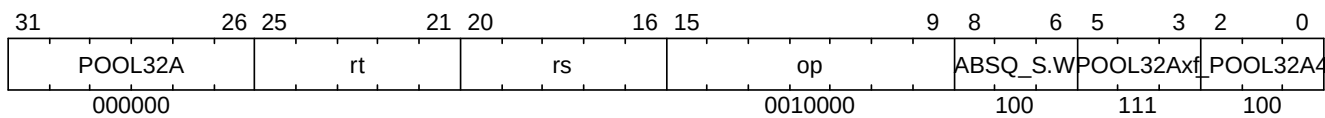


Figure 16. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

nanoMIPS

absq_s.w.32

Assembly

ABSQ_S.W {rd:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

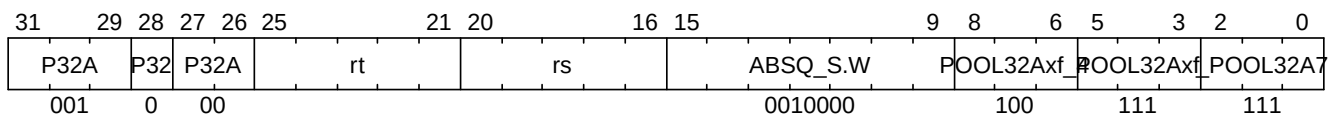


Figure 17. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

8.6.3. Operation

```
validate_access_to_dsp_resources(ctx)
if signed(ctx.GPR[operands.rs], 32) == -(1 << 31):
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(
    packed_abs(ctx.GPR[operands.rs], 32, 1), from_nbits=32
)
```

8.6.4. Description

ABSQ_S.W is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.6.5. Exceptions

None.

8.7. ADD

8.7.1. Purpose

Add two signed 32-bit GPR values and trap on overflow.

8.7.2. Formats

MIPS

add

Assembly

ADD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

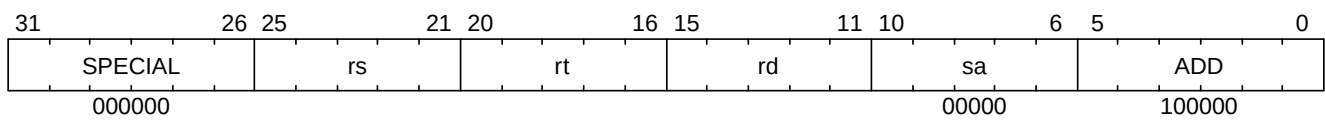


Figure 18. Encoding (32 bits)

Decode

```
return Add0perands(rd=rd, rs=rs, rt=rt)
```

microMIPS

add.32

Assembly

ADD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

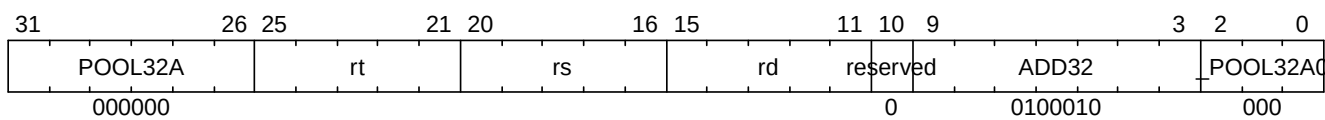


Figure 19. Encoding (32 bits)

Decode

```
return AddOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

add.32

Assembly

ADD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

!(NMS)

Decode priority

0

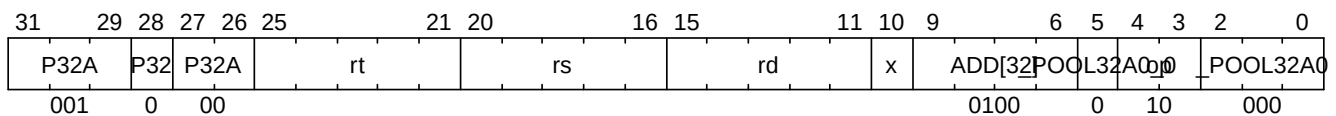


Figure 20. Encoding (32 bits)

Decode

```
return AddOperands(rd=rd, rs=rs, rt=rt)
```

8.7.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
result = int(lhs) + int(rhs)
if int(sign_extend(result, from_nbits=32)) != result:
    raise IntegerOverflowException("ADD signed 32-bit overflow")
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.7.4. Description

ADD sign-extends both source GPRs from 32 bits, traps when their mathematical sum cannot be represented as a signed 32-bit value, and sign-extends the 32-bit result to the active GPR width. ADD is not available in the nanoMIPS subset.

8.7.5. Exceptions

- **OV** (Integer Overflow) — ADD signed 32-bit overflow
- **RI** (Reserved Instruction) — ADD is reserved when CP0 Config5.NMS is set.

8.8. ADD.D

8.8.1. Purpose

Add two D-format floating-point values.

8.8.2. Formats

MIPS

add.d

Assembly

ADD.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

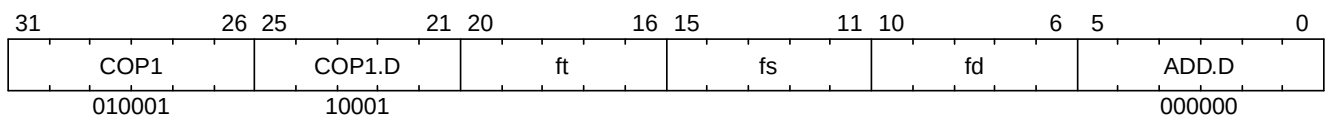


Figure 21. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

add.d

Assembly

ADD.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

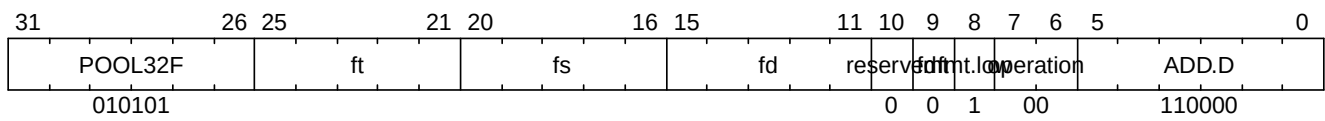


Figure 22. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

add.d

Assembly

ADD.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

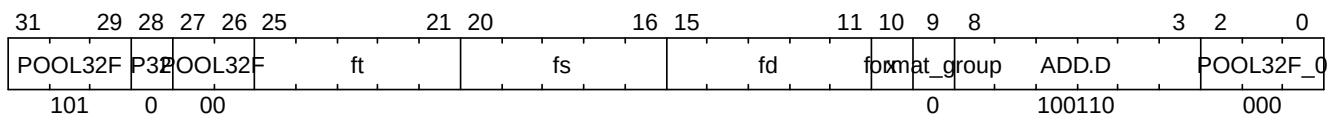


Figure 23. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.8.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_add(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.8.4. Description

ADD.D validates CP1 and D-format support before updating the destination FPR.

8.8.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.9. ADD.PS

8.9.1. Purpose

Perform the paired-single add operation.

8.9.2. Formats

MIPS

add.ps

Assembly

ADD.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

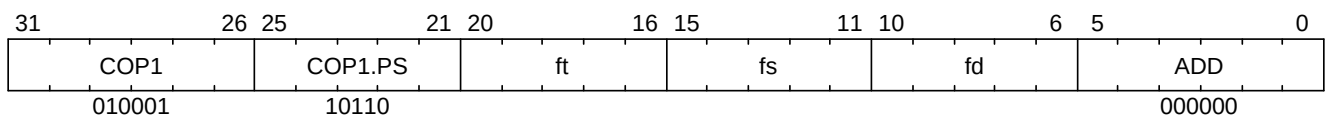


Figure 24. Encoding (32 bits)

Decode

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

add.ps

Assembly

ADD.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

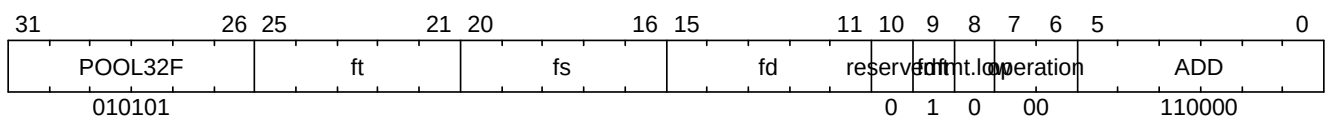


Figure 25. Encoding (32 bits)

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.9.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
left = get_fpr(ctx, operands.fs, "PS")
right = get_fpr(ctx, operands.ft, "PS")
low, low_flags = ieee_add("S", rounding, left & 0xFFFF_FFFF, right & 0xFFFF_FFFF)
high, high_flags = ieee_add("S", rounding, left >> 32, right >> 32)
write_fpu_result(ctx, operands.fd, low | (high << 32), low_flags | high_flags, "PS")
```

8.9.4. Description

ADD.PS applies independently to the upper and lower S-format lanes.

8.9.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.10. ADD.S

8.10.1. Purpose

Add two S-format floating-point values.

8.10.2. Formats

MIPS

`add.s`

Assembly

`ADD.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

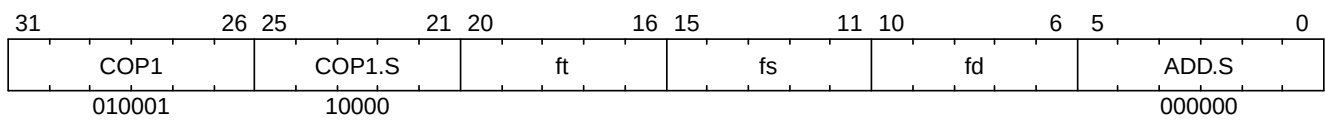


Figure 26. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`add.s`

Assembly

`ADD.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

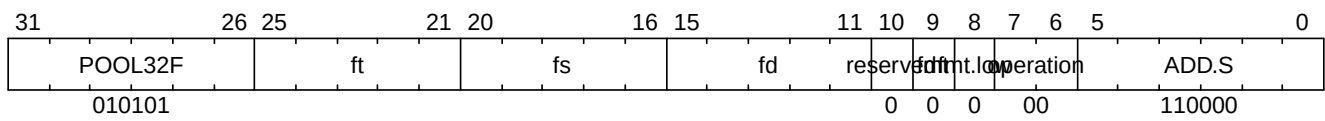


Figure 27. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

add.s

Assembly

ADD.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

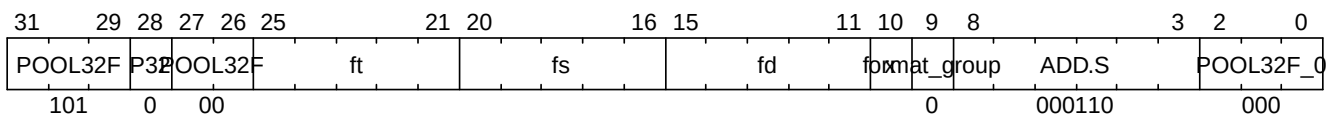


Figure 28. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.10.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_add(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.10.4. Description

ADD.S validates CP1 and S-format support before updating the destination FPR.

8.10.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.11. ADDI

8.11.1. Purpose

Add a signed immediate to a 32-bit GPR and trap on overflow.

8.11.2. Formats

MIPS

`addi`

Assembly

`ADDI {rt:gpr}, {rs:gpr}, {imm:s16}`

Availability

AR < R6

Decode priority

0

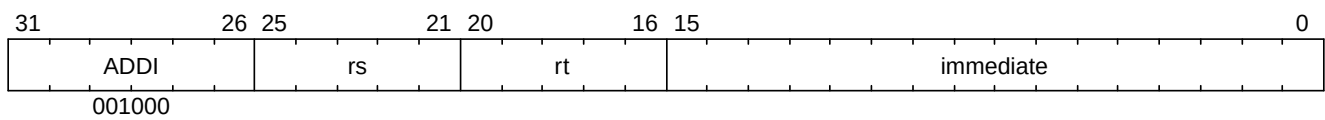


Figure 29. Encoding (32 bits)

Decode

```
return AddiOperands(rt=rt, rs=rs, imm=sign_extend(immediate, from_nbits=16))
```

microMIPS

`addi.32`

Assembly

`ADDI {rt:gpr}, {rs:gpr}, {imm:s16}`

Availability

MMAR < R6

Decode priority

0

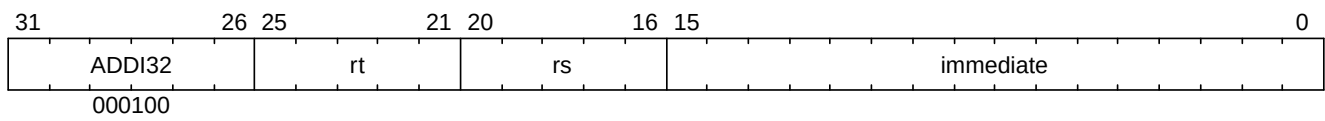


Figure 30. Encoding (32 bits)


```
return AddiOperands(rt=rt, rs=rs, imm=sign_extend(immediate, from_nbits=16))
```

8.11.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
result = int(lhs) + int(operands.imm)
if int(sign_extend(result, from_nbits=32)) != result:
    raise IntegerOverflowException("ADDI signed 32-bit overflow")
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.11.4. Description

ADDI sign-extends its 16-bit immediate and source GPR from 32 bits, traps on signed integer overflow, and sign-extends the 32-bit result to the active GPR width. nanoMIPS has no ADDI encoding; its analogous untrapped immediate operation is ADDIU.

8.11.5. Exceptions

- **OV** (Integer Overflow) — ADDI signed 32-bit overflow

8.12. ADDIU

8.12.1. Purpose

Add an immediate value to a 32-bit integer source, place the sign-extended 32-bit result in the destination GPR, and do not trap on integer overflow.

8.12.2. Formats

MIPS

addiu

Assembly

ADDIU {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

true

Decode priority

0

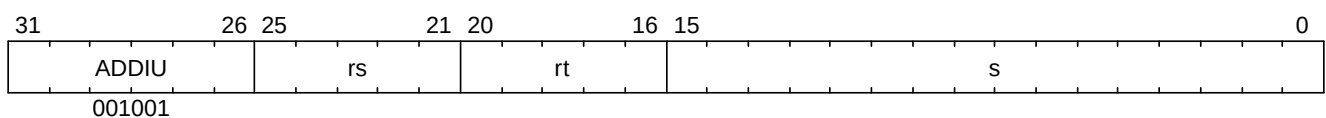


Figure 31. Encoding (32 bits)

Decode

```
imm = sign_extend(s, from_nbits=16)
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

MIPS16E

addiu.rx

Assembly

ADDIU {rt:gpr3}, {imm:s8}

Availability

MIPS16E

Decode priority

0

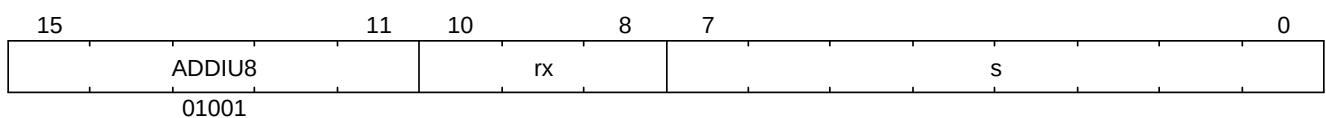


Figure 32. Encoding (16 bits)

Decode

```
rt = MIPS16_GPR3.decode(rx)
rs = rt
imm = sign_extend(s, from_nbits=8)
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.rx.extended

Assembly

ADDIU {rt:gpr3}, {imm:s16}

Availability

MIPS16E

Decode priority

0

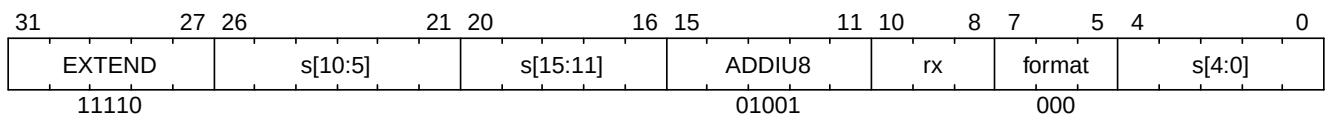


Figure 33. Encoding (32 bits)

Decode

```
rt = MIPS16_GPR3.decode(rx)
rs = rt
imm = sign_extend(s, from_nbits=16)
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.ry_rx

Assembly

ADDIU {rt:gpr3}, {rs:gpr3}, {imm:s4}

Availability

MIPS16E

Decode priority

0

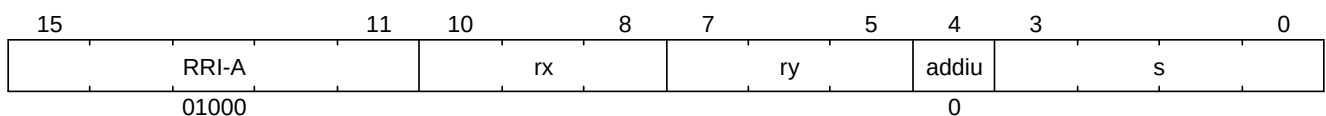


Figure 34. Encoding (16 bits)

Decode

```
rt = MIPS16_GPR3.decode(ry)
rs = MIPS16_GPR3.decode(rx)
```

```

imm = sign_extend(s, from_nbits=4)
return AddiuOperands(rt=rt, rs=rs, imm=imm)

```

addiu.ry_rx.extended

Assembly

ADDIU {rt:gpr3}, {rs:gpr3}, {imm:s15}

Availability

MIPS16E

Decode priority

0

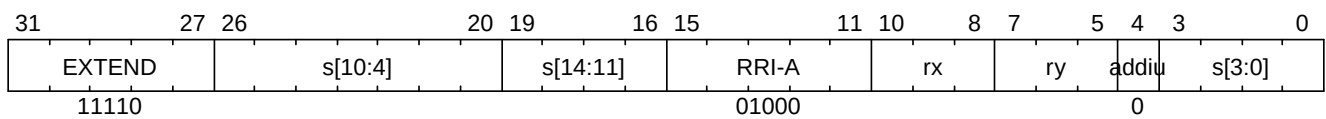


Figure 35. Encoding (32 bits)

Decode

```

rt = MIPS16_GPR3.decode(ry)
rs = MIPS16_GPR3.decode(rx)
imm = sign_extend(s, from_nbits=15)
return AddiuOperands(rt=rt, rs=rs, imm=imm)

```

addiu.sp

Assembly

ADDIU \$sp, {imm:s11}

Availability

MIPS16E

Decode priority

0

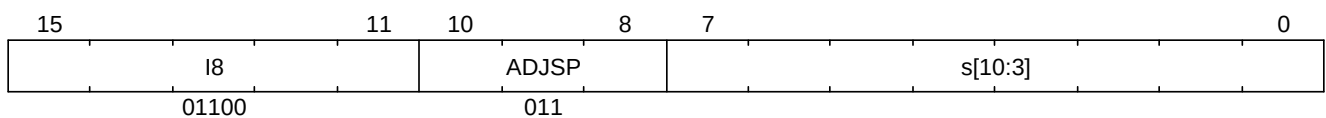


Figure 36. Encoding (16 bits)

Decode

```

imm = sign_extend(s, from_nbits=11)
return AddiuOperands(rt=29, rs=29, imm=imm)

```

addiu.sp.extended

Assembly

ADDIU \$sp, {imm:s16}

Availability

MIPS16E

Decode priority

0

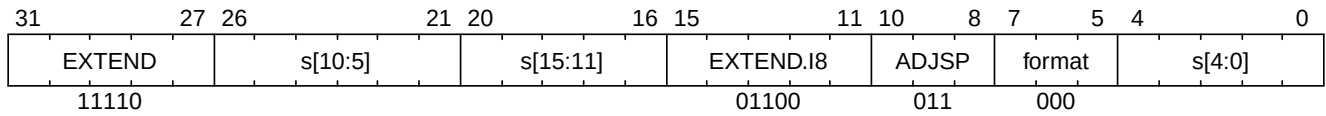


Figure 37. Encoding (32 bits)

Decode

```
imm = sign_extend(s, from_nbits=16)
return AddiuOperands(rt=29, rs=29, imm=imm)
```

addiu.rx_sp

Assembly

ADDIU {rt:gpr3}, \$sp, {imm:u10}

Availability

MIPS16E

Decode priority

0

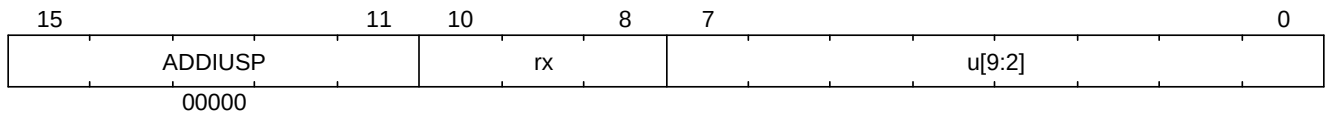


Figure 38. Encoding (16 bits)

Decode

```
rt = MIPS16_GPR3.decode(rx)
imm = zero_extend(u, from_nbits=10)
return AddiuOperands(rt=rt, rs=29, imm=imm)
```

addiu.rx_sp.extended

Assembly

ADDIU {rt:gpr3}, \$sp, {imm:s16}

Availability

MIPS16E

Decode priority

0

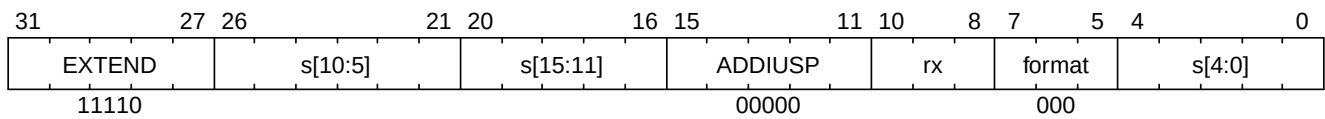


Figure 39. Encoding (32 bits)

Decode

```
rt = MIPS16_GPR3.decode(rx)
imm = sign_extend(s, from_nbits=16)
return AddiuOperands(rt=rt, rs=29, imm=imm)
```

microMIPS

addiu

Assembly

ADDIU {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

true

Decode priority

0

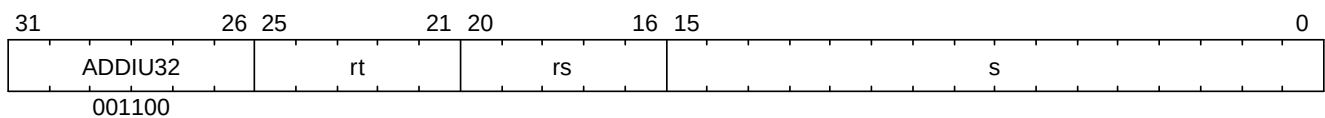


Figure 40. Encoding (32 bits)

Decode

```
imm = sign_extend(s, from_nbits=16)
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.r1sp.16

Assembly

ADDIU {rt:gpr3}, {imm:u8}(sp)

Availability

true

Decode priority

0

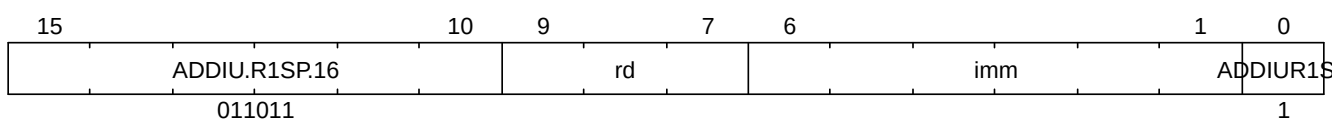


Figure 41. Encoding (16 bits)

Decode

```
return AddiuOperands(  
    rt=MICROMIPS_GPR3.decode(rd),  
    rs=29,  
    imm=zero_extend(zero_extend(imm, from_nbits=6) << 2, from_nbits=8),  
)
```

addiu.sp.16

Assembly

ADDIU {imm:s10}

Availability

true

Decode priority

0

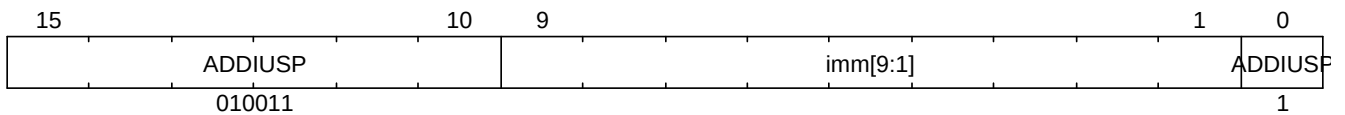


Figure 42. Encoding (16 bits)

Decode

```
decoded_imm = sign_extend(imm, from_nbits=10) << 1  
decoded_imm = (  
    -1032  
    if decoded_imm == -8  
    else -1028  
    if decoded_imm == -4  
    else 1024  
    if decoded_imm == 0  
    else 1028  
    if decoded_imm == 4  
    else decoded_imm  
)  
return AddiuOperands(  
    rt=29,  
    rs=29,  
    imm=sign_extend(decoded_imm, from_nbits=32),  
)
```

addiu.s5.16

Assembly

ADDIU {rt:gpr}, {imm:s4}

Availability

true

Decode priority

0

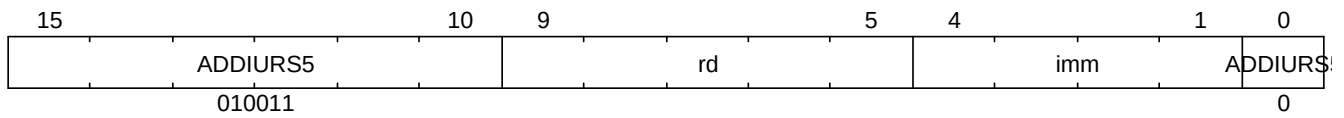


Figure 43. Encoding (16 bits)

Decode

```
return AddiuOperands(  
    rt=rd,  
    rs=rd,  
    imm=sign_extend(imm, from_nbits=4),  
)
```

addiu.r2.16

Assembly

ADDIU {rt:gpr3}, {rs:gpr3}, {imm:s5}

Availability

true

Decode priority

0

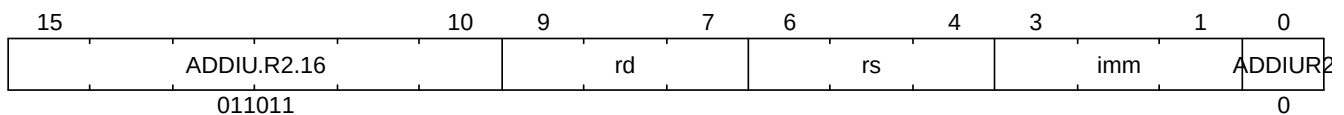


Figure 44. Encoding (16 bits)

Decode

```
del ctx, exec  
decoded_imm = sign_extend(  
    1 if imm == 0 else -1 if imm == 7 else imm << 2,  
    from_nbits=32,  
)  
return AddiuOperands(  
    rt=MICROMIPS_GPR3.decode(rd),  
    rs=MICROMIPS_GPR3.decode(rs),  
    imm=decoded_imm,  
)
```

nanoMIPS

addiu.32

Assembly

```
ADDIU {rt:gpr}, {rs:gpr}, {imm:u16}
```

Availability

true

Decode priority

0

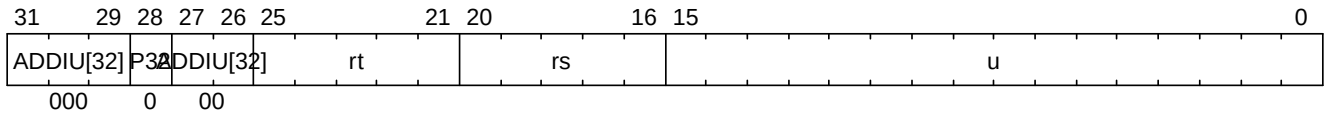


Figure 45. Encoding (32 bits)

Encoding constraints

- `rt != 0`

Decode

```
imm = zero_extend(u, from_nbits=16)
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.48

Assembly

```
ADDIU {rt:gpr}, {imm:s32}
```

Availability

!(NMS)

Decode priority

0

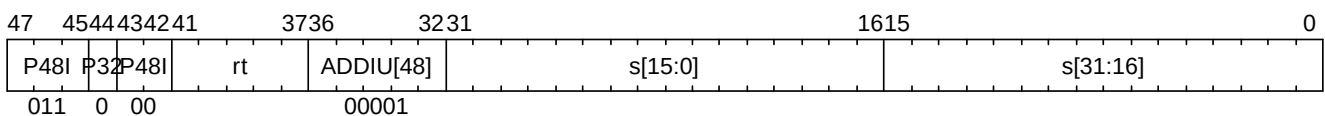


Figure 46. Encoding (48 bits)

Decode

```
imm = sign_extend(s, from_nbits=32)
rs = rt
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.gp_b

Assembly

```
ADDIU {rt:gpr}, $gp, {imm:u18}
```

Availability

true

Decode priority

0

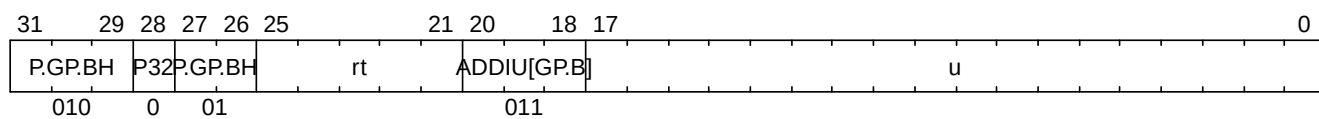


Figure 47. Encoding (32 bits)

Decode

```
if P64_Mode(ctx):
    raise behaves_like("daddiu.gp_b")
imm = zero_extend(u, from_nbits=18)
rs = 28
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.gp_w

Assembly

ADDIU {rt:gpr}, \$gp, {imm:u21}

Availability

true

Decode priority

0

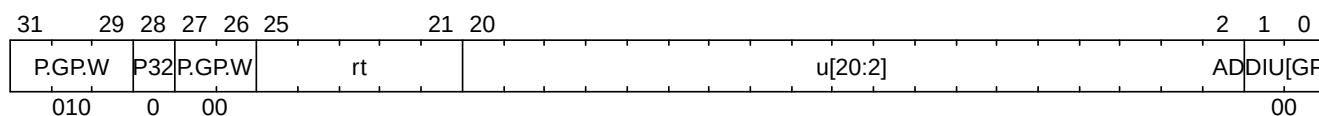


Figure 48. Encoding (32 bits)

Decode

```
if P64_Mode(ctx):
    raise behaves_like("daddiu.gp_w")
imm = zero_extend(u, from_nbits=21)
rs = 28
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.gp48

Assembly

ADDIU {rt:gpr}, \$gp, {imm:s32}

Availability

!(NMS)

Decode priority

0

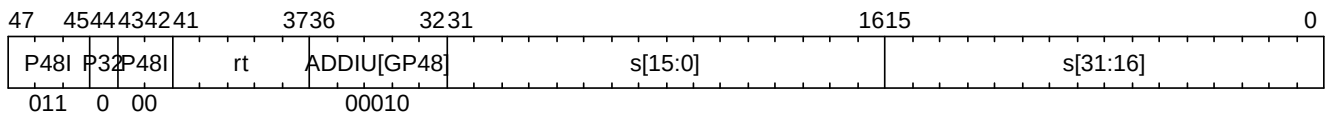


Figure 49. Encoding (48 bits)

Decode

```
if P64_Mode(ctx):
    raise behaves_like("daddiu.gp48")
imm = sign_extend(s, from_nbits=32)
rs = 28
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.neg

Assembly

ADDIU {rt:gpr}, {rs:gpr}, -{u:u12}

Availability

true

Decode priority

0

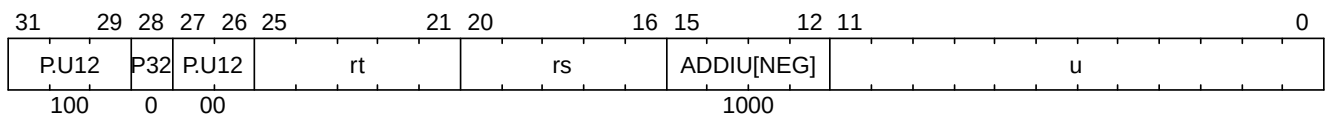


Figure 50. Encoding (32 bits)

Decode

```
imm = sign_extend(-u, from_nbits=13)
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

addiu.r1_sp

Assembly

ADDIU {rt:gpr3}, \$sp, {imm:u8}

Availability

true

Decode priority

0

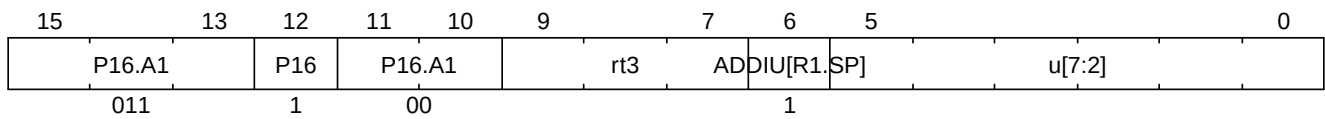


Figure 51. Encoding (16 bits)

Decode

```

if P64_Mode(ctx):
    raise behaves_like("daddiu.r1_sp")
rt = GPR3.decode(rt3)
rs = 29
imm = zero_extend(u, from_nbits=8)
return AddiuOperands(rt=rt, rs=rs, imm=imm)

```

addiu.r2

Assembly

ADDIU {rt:gpr3}, {rs:gpr3}, {imm:u5}

Availability

true

Decode priority

0

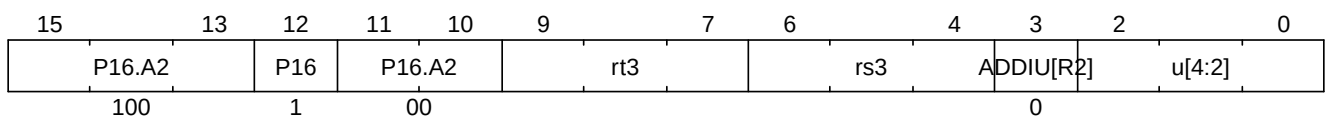


Figure 52. Encoding (16 bits)

Decode

```

rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
imm = zero_extend(u, from_nbits=5)
return AddiuOperands(rt=rt, rs=rs, imm=imm)

```

addiu.rs5

Assembly

ADDIU {rt:gpr}, {imm:s5}

Availability

true

Decode priority

0

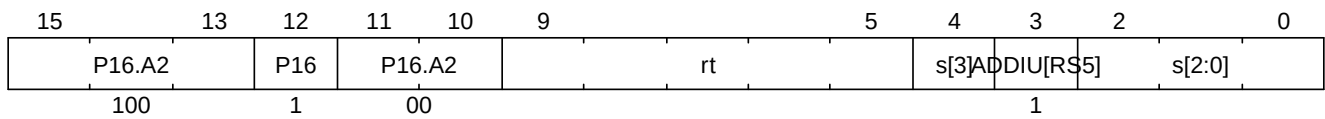


Figure 53. Encoding (16 bits)

Encoding constraints

- `rt != 0`

Decode

```
rs = rt
imm = sign_extend(s, from_nbits=4)
return AddiuOperands(rt=rt, rs=rs, imm=imm)
```

8.12.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(operands.imm, from_nbits=32)
sum = lhs + rhs
ctx.GPR[operands.rt] = sign_extend(sum, from_nbits=32)
```

8.12.4. Description

The addition is performed modulo 2^{32} . The low 32 bits of the sum are sign-extended to the architectural GPR width before being written to `rt`. The encoding forms provide different immediate widths, implicit source registers, compact-register maps, and instruction lengths without changing this operation.

8.12.5. Exceptions

None.

8.12.6. Programming Notes

The `ADDIU[RS5]` encoding with `rt=0` is reserved for `NOP[16]`; therefore the `ADDIU` form requires `rt!=0`.

The nanoMIPS manual names the `ADDIU[RS5]` assembly immediate `s5`, while its format encodes four `s` bits and its decode operation sign-extends from four bits.

8.13. ADDIU

8.13.1. Purpose

Add an immediate displacement to the word-aligned PC in MIPS16 mode.

8.13.2. Formats

MIPS16E

`addiupc`

Assembly

`ADDIU {rt:gpr3}, pc, {imm:u10}`

Availability

MIPS16E

Decode priority

0

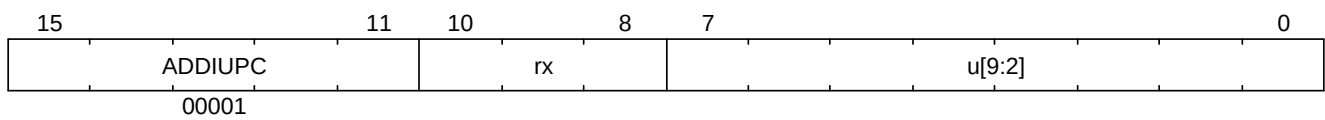


Figure 54. Encoding (16 bits)

Decode

```
rt = MIPS16_GPR3.decode(rx)
imm = zero_extend(u, from_nbits=10)
return AddiupcOperands(rt=rt, imm=imm, base_is_next_pc=False)
```

`addiupc.extended`

Assembly

`ADDIU {rt:gpr3}, pc, {imm:s16}`

Availability

MIPS16E

Decode priority

0

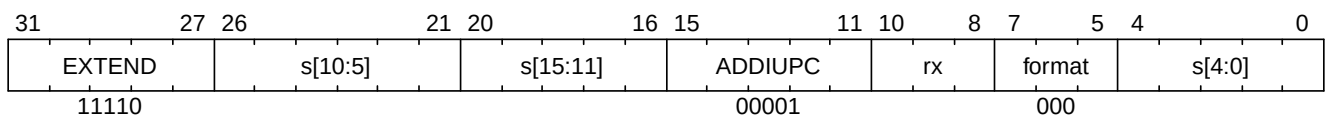


Figure 55. Encoding (32 bits)

Decode

```
rt = MIPS16_GPR3.decode(rx)
imm = sign_extend(s, from_nbits=16)
return AddiupcOperands(rt=rt, imm=imm, base_is_next_pc=False)
```

8.13.3. Operation

```
base_pc = PriorPC(ctx) if JumpDelaySlot(ctx) else ctx.pc
ctx.GPR[operands.rt] = (base_pc & ~0b11) + operands.imm
```

8.13.4. Description

The MIPS16 PC-relative ADDIU forms use the word-aligned current PC rather than a GPR source.

8.13.5. Exceptions

None.

8.13.6. Programming Notes

The assembler **LA** pseudo-instruction can use the PC-relative form.

8.14. ADDIUPC

8.14.1. Purpose

Add an immediate displacement to a word-aligned PC-derived address and place the result in a GPR without trapping on integer overflow.

8.14.2. Formats

MIPS

addiupc

Assembly

ADDIUPC {rt:gpr}, {imm:s21}

Availability

AR >= R6

Decode priority

0

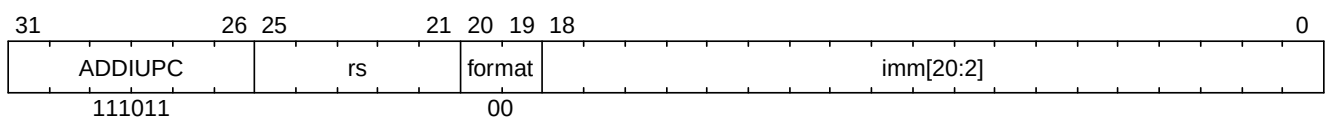


Figure 56. Encoding (32 bits)

Decode

```
imm = sign_extend(imm, from_nbits=21)
return AddiupcOperands(rt=rs, imm=imm, base_is_next_pc=False)
```

microMIPS

addiupc.r3

Assembly

ADDIUPC {rt:gpr3}, {imm:s25}

Availability

MMAR < R6

Decode priority

0



Figure 57. Encoding (32 bits)

Decode

```
return AddiupcOperands(  
    rt=MICROMIPS_GPR3.decode(rs),  
    imm=sign_extend(immediate, from_nbits=25),  
    base_is_next_pc=False,  
)
```

addiupc.32.micromips

Assembly

ADDIUPC {rt:gpr}, {imm:s21}

Availability

MMAR >= R6

Decode priority

0

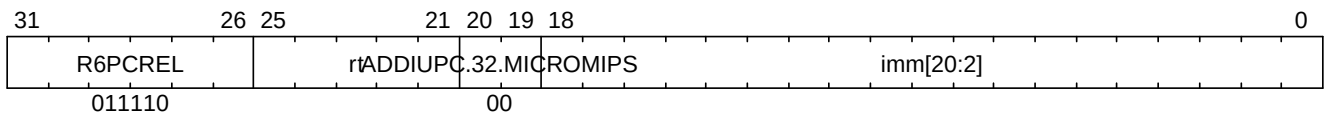


Figure 58. Encoding (32 bits)

Decode

```
del ctx, exec  
return AddiupcOperands(  
    rt=rt,  
    imm=sign_extend(imm, from_nbits=21),  
    base_is_next_pc=False,  
)
```

nanoMIPS

addiupc.32

Assembly

ADDIUPC {rt:gpr}, {imm:s22}

Availability

true

Decode priority

0

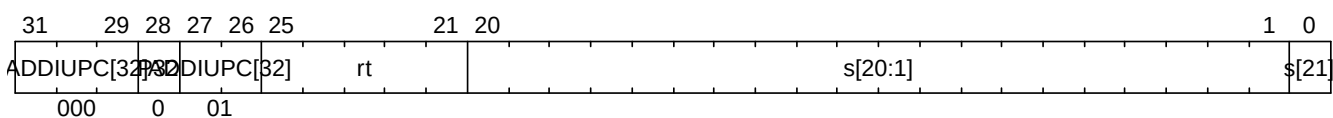


Figure 59. Encoding (32 bits)

Decode

```
if P64_Mode(ctx):
    raise behaves_like("daddiupc.32")
imm = sign_extend(s, from_nbits=22)
return NanoAddiupcOperands(rt=rt, imm=imm, base_is_next_pc=True)
```

addiupc.48

Assembly

ADDIUPC {rt:gpr}, {imm:s32}

Availability

!(NMS)

Decode priority

0

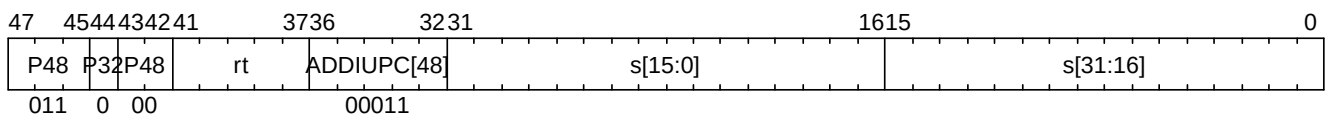


Figure 60. Encoding (48 bits)

Decode

```
if P64_Mode(ctx):
    raise behaves_like("daddiupc.48")
imm = sign_extend(s, from_nbits=32)
return NanoAddiupcOperands(rt=rt, imm=imm, base_is_next_pc=True)
```

8.14.3. Operation

```
if operands.base_is_next_pc:
    base_pc = ctx.next_pc
else:
    base_pc = ctx.pc
    base_pc &= ~0b11
# In a MIPS64 P64 address mode the PC-relative operation is a full-width
# address calculation. Truncating the low word before writing the GPR
# loses the canonical sign-extension when a displacement crosses the
# 0x80000000 boundary (the R6 AVP ADDIUPC cases exercise that boundary).
if ctx.profile.AT is AT.MIPS64 and P64_Mode(ctx):
    result = int(base_pc) + int(operands.imm)
    # The microMIPS R6 LAPC encoding retains the 32-bit ADDIUPC result
    # width even in a 64-bit address mode. Sign-extend that low word;
    # the AVP PC-relative forms exercise both sides of the 0x80000000
    # boundary.
    if ctx.ISAMode is ISAMode.MICROMIPS:
        ctx.GPR[operands.rt] = sign_extend(result & 0xFFFF_FFFF, from_nbits=32)
```

```
else:
    ctx.GPR[operands.rt] = result & 0xFFFF_FFFF_FFFF_FFFF
else:
    result = (int(base_pc) + int(operands.imm)) & 0xFFFF_FFFF
    ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.14.4. Description

The MIPS16E assembly spelling is `ADDIU rt, pc, immediate`. The short form adds a scaled unsigned displacement to the current instruction address, or to the owning jump address when executed in its delay slot. The extended form adds a signed 16-bit displacement and is not permitted in a jump delay slot.

8.14.5. Exceptions

None.

8.14.6. Programming Notes

The assembler `LA` pseudo-instruction can use the PC-relative form.

8.15. ADDQ.PH

8.15.1. Purpose

DSP packed operation ADDQ.PH.

8.15.2. Formats

MIPS

addq.ph

Assembly

ADDQ.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

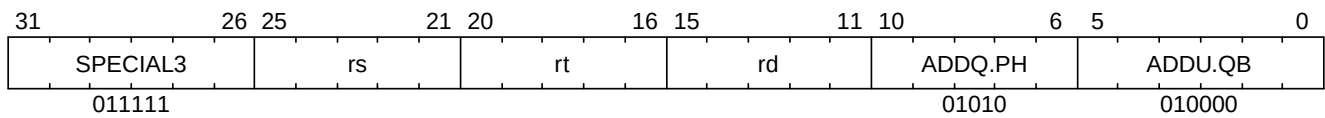


Figure 61. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addq.ph.32

Assembly

ADDQ.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

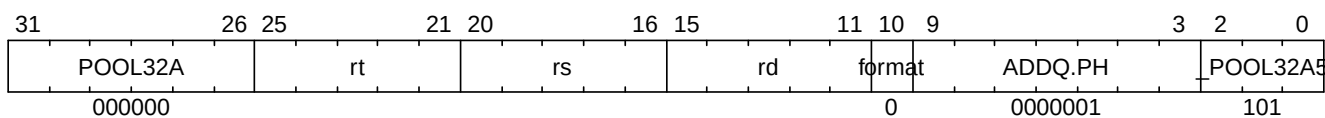


Figure 62. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`addq.ph.32`

Assembly

`ADDQ.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

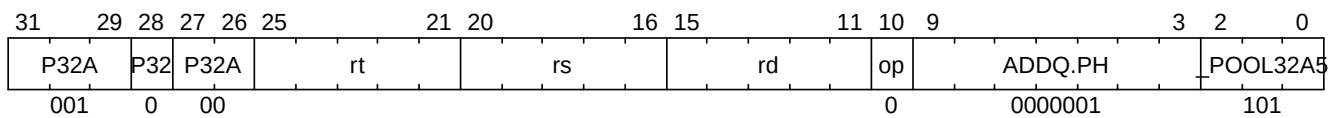


Figure 63. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.15.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.15.4. Description

ADDQ.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.15.5. Exceptions

None.

8.16. ADDQ_S.PH

8.16.1. Purpose

DSP packed operation ADDQ_S.PH.

8.16.2. Formats

MIPS

addq_s.ph

Assembly

ADDQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

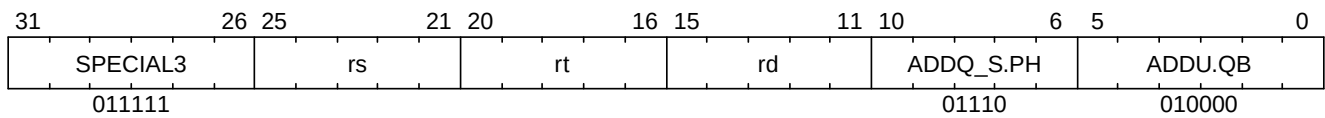


Figure 64. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addq_s.ph.32

Assembly

ADDQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

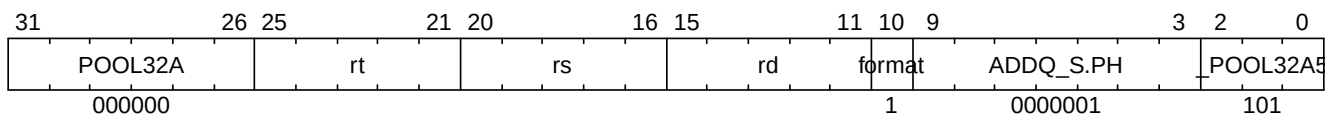


Figure 65. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

addq_s.ph.32

Assembly

ADDQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

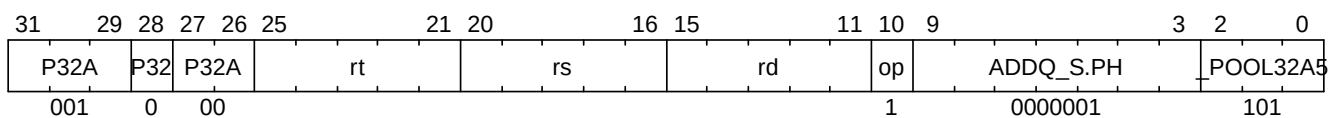


Figure 66. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.16.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.16.4. Description

ADDQ_S.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.16.5. Exceptions

None.

8.17. ADDQ_S.W

8.17.1. Purpose

DSP packed operation ADDQ_S.W.

8.17.2. Formats

MIPS

addq_s.w

Assembly

ADDQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

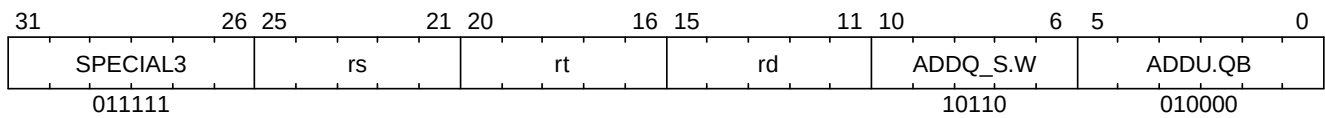


Figure 67. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addq_s.w.32

Assembly

ADDQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

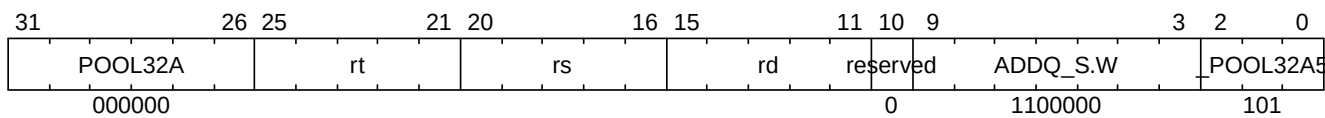


Figure 68. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

addq_s.w.32

Assembly

ADDQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

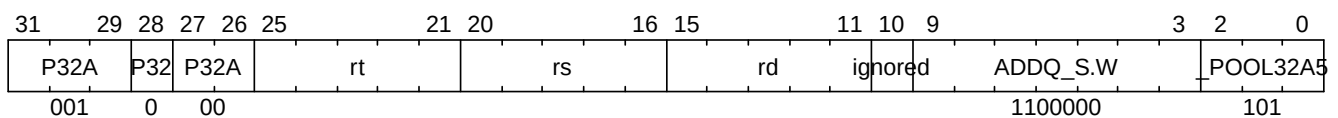


Figure 69. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.17.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 32, 1, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.17.4. Description

ADDQ_S.W is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.17.5. Exceptions

None.

8.18. ADDQH.PH

8.18.1. Purpose

DSP packed operation ADDQH.PH.

8.18.2. Formats

MIPS

addqh.ph

Assembly

ADDQH.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

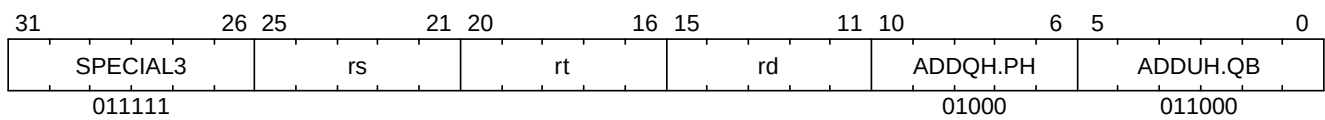


Figure 70. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addqh.ph.32

Assembly

ADDQH.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

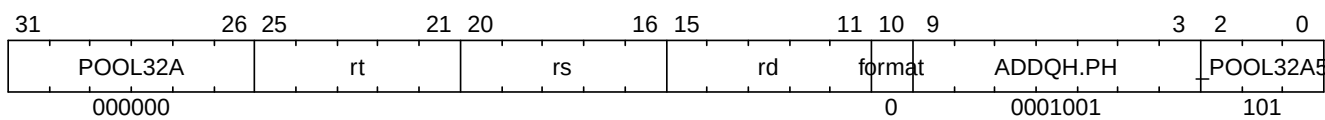


Figure 71. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`addqh.ph.32`

Assembly

`ADDQH.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP2

Decode priority

0

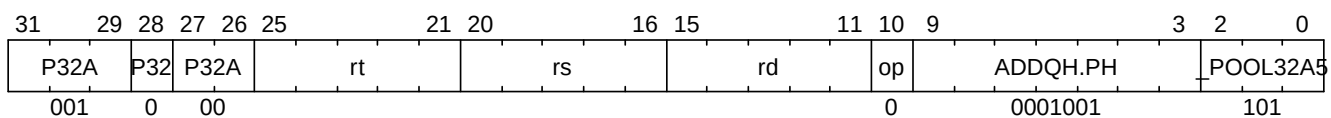


Figure 72. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.18.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, half=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.18.4. Description

ADDQH.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.18.5. Exceptions

None.

8.19. ADDQH.W

8.19.1. Purpose

DSP packed operation ADDQH.W.

8.19.2. Formats

MIPS

addqh.w

Assembly

ADDQH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

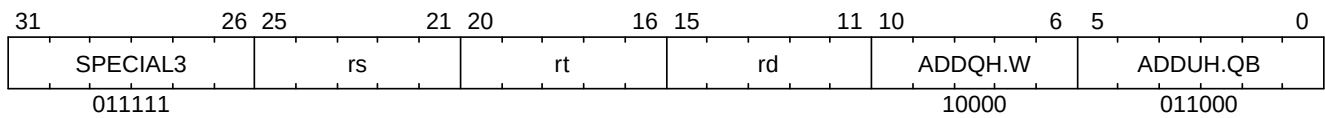


Figure 73. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addqh.w.32

Assembly

ADDQH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

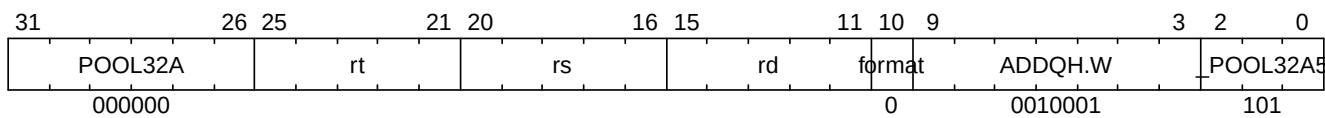


Figure 74. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

addqh.w.32

Assembly

ADDQH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

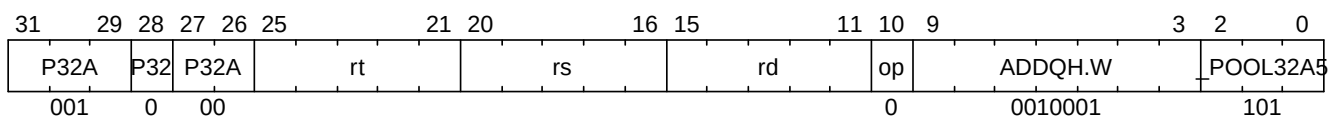


Figure 75. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.19.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 32, 1, half=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.19.4. Description

ADDQH.W is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.19.5. Exceptions

None.

8.20. ADDQH_R.PH

8.20.1. Purpose

DSP packed operation ADDQH_R.PH.

8.20.2. Formats

MIPS

addqh_r.ph

Assembly

ADDQH_R.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

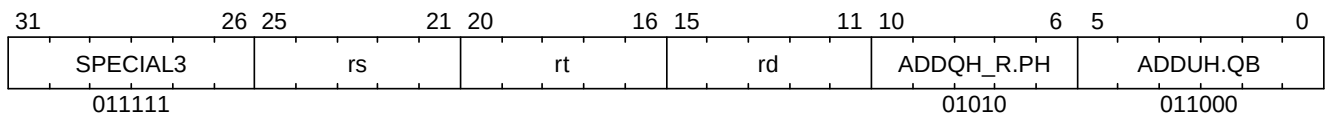


Figure 76. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addqh_r.ph.32

Assembly

ADDQH_R.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

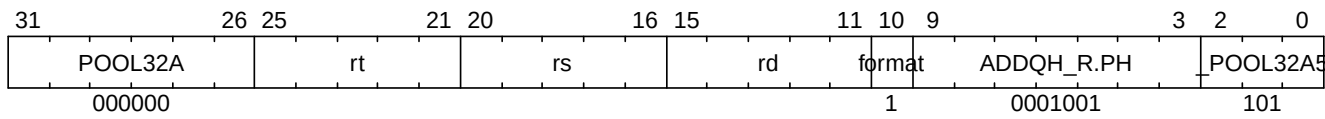


Figure 77. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

addqh_r.ph.32

Assembly

ADDQH_R.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

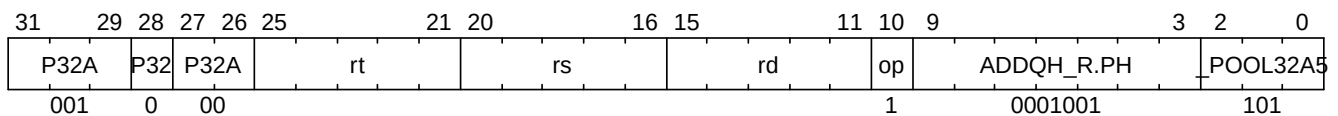


Figure 78. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.20.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, half=True, rounded=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.20.4. Description

ADDQH_R.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.20.5. Exceptions

None.

8.21. ADDQH_R.W

8.21.1. Purpose

DSP packed operation ADDQH_R.W.

8.21.2. Formats

MIPS

addqh_r.w

Assembly

ADDQH_R.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

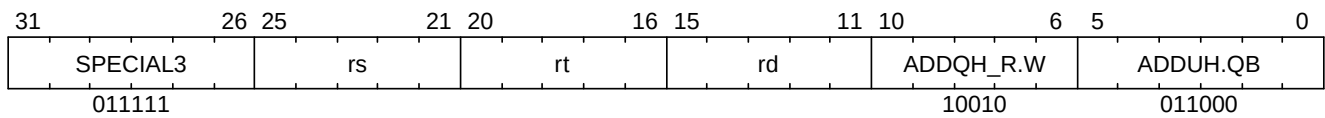


Figure 79. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addqh_r.w.32

Assembly

ADDQH_R.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

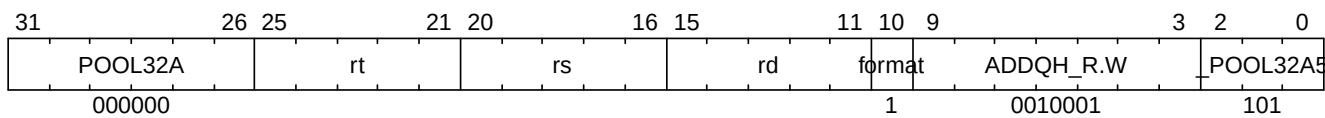


Figure 80. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

addqh_r.w.32

Assembly

ADDQH_R.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

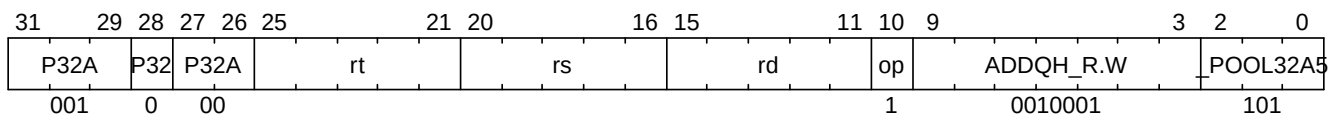


Figure 81. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.21.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 32, 1, half=True, rounded=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.21.4. Description

ADDQH_R.W is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.21.5. Exceptions

None.

8.22. ADDSC

8.22.1. Purpose

DSP packed operation ADDSC.

8.22.2. Formats

MIPS

`addsc`

Assembly

`ADDSC {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

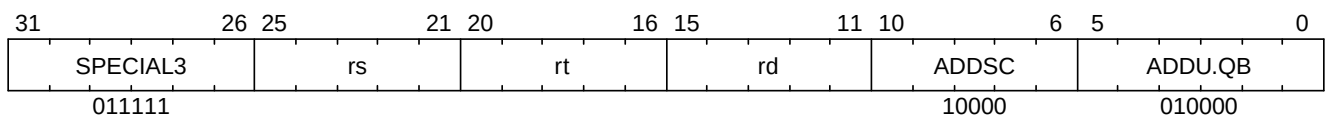


Figure 82. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`addsc.32`

Assembly

`ADDSC {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

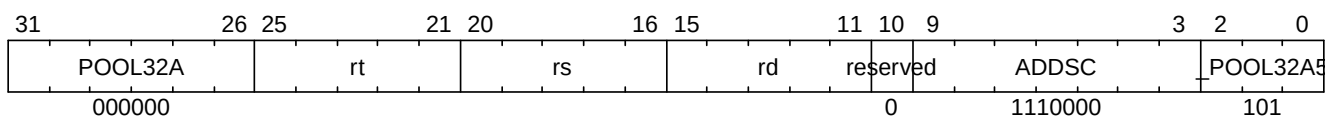


Figure 83. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`addsc.32`

Assembly

`ADDSC {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

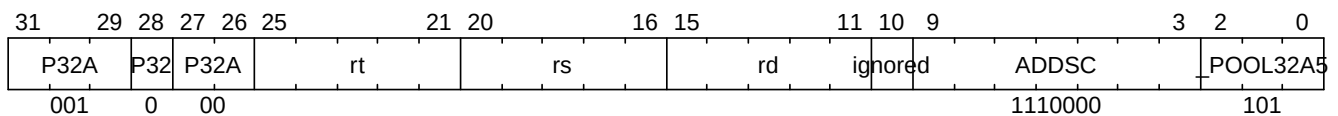


Figure 84. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.22.3. Operation

```
validate_access_to_dsp_resources(ctx)
total = (ctx.GPR[operands.rs] & 0xFFFF_FFFF) + (ctx.GPR[operands.rt] & 0xFFFF_FFFF)
ctx.DSPControl.C = (total >> 32) & 1
ctx.GPR[operands.rd] = sign_extend(total & 0xFFFF_FFFF, from_nbits=32)
```

8.22.4. Description

ADDSC is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.22.5. Exceptions

None.

8.23. ADDU

8.23.1. Purpose

Add two 32-bit integers in GPRs, place the 32-bit result in a GPR, and do not trap on overflow.

8.23.2. Formats

MIPS

`addu`

Assembly

`ADDU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

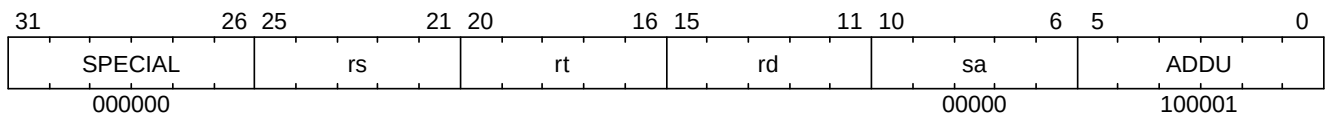


Figure 85. Encoding (32 bits)

Decode

```
return AdduOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`addu.32`

Assembly

`ADDU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

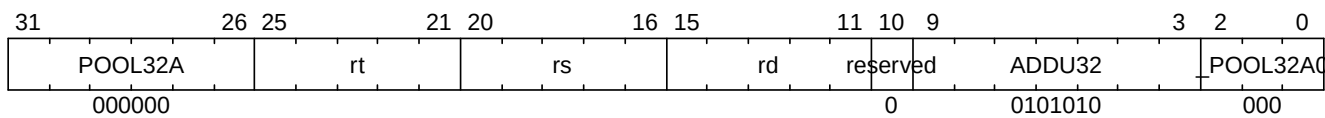


Figure 86. Encoding (32 bits)

Decode

```
del ctx, exec
return AdduOperands(rd=rd, rs=rs, rt=rt)
```

addu.16

Assembly

ADDU {rd:gpr3}, {rs:gpr3}, {rt:gpr3}

Availability

MMAR < R6

Decode priority

0

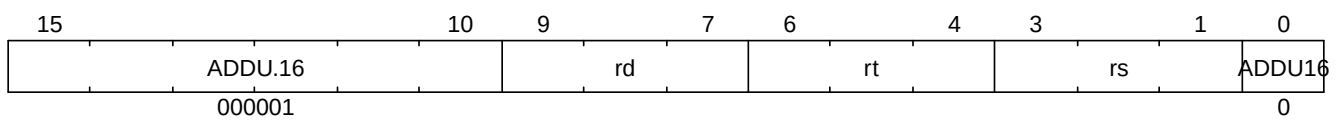


Figure 87. Encoding (16 bits)

Decode

```
del ctx, exec
return AdduOperands(
    rd=MICROMIPS_GPR3.decode(rd),
    rs=MICROMIPS_GPR3.decode(rs),
    rt=MICROMIPS_GPR3.decode(rt),
)
```

addu.16.r6

Assembly

ADDU {rd:gpr3}, {rs:gpr3}, {rt:gpr3}

Availability

MMAR >= R6

Decode priority

0

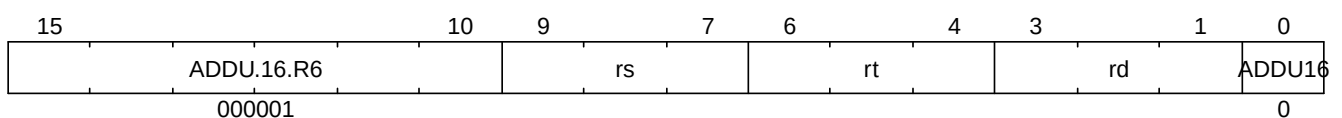


Figure 88. Encoding (16 bits)

Decode

```
del ctx, exec
return AdduOperands(
```

```

rd=MICROMIPS_GPR3.decode(rd),
rs=MICROMIPS_GPR3.decode(rs),
rt=MICROMIPS_GPR3.decode(rt),
)

```

nanoMIPS

addu.32

Assembly

ADDU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

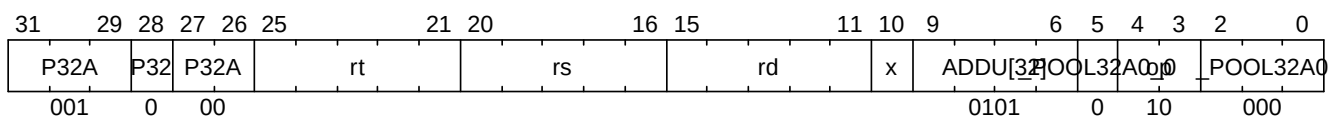


Figure 89. Encoding (32 bits)

Decode

```

return AdduOperands(rd=rd, rs=rs, rt=rt)

```

addu.16

Assembly

ADDU {rd:gpr3}, {rs:gpr3}, {rt:gpr3}

Availability

true

Decode priority

0

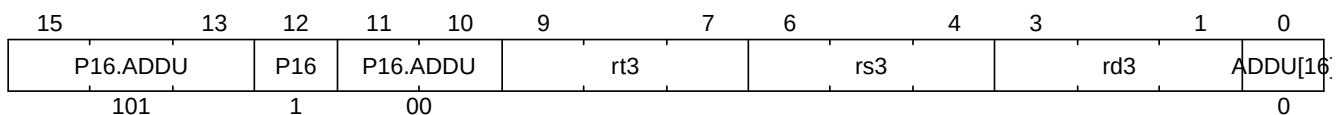


Figure 90. Encoding (16 bits)

Decode

```

return AdduOperands(
    rd=GPR3.decode(rd3),
    rs=GPR3.decode(rs3),
    rt=GPR3.decode(rt3),
)

```

addu.4x4

Assembly

ADDU {rd:gpr4}, {rt:gpr4}

Availability

!(NMS)

Decode priority

0

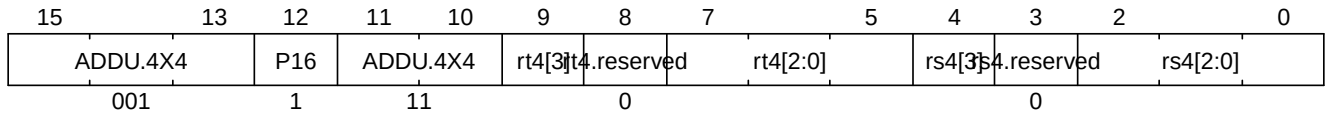


Figure 91. Encoding (16 bits)

Decode

```
rd = GPR4.decode(rt4)
return AdduOperands(rd=rd, rs=rd, rt=GPR4.decode(rs4))
```

8.23.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
result = lhs + rhs
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.23.4. Description

ADDU computes the sum of the selected 32-bit source values modulo 2^{32} and sign-extends the low 32 bits to the implemented GPR width. nanoMIPS provides full-register, compact 16-bit, and legacy 4x4 forms; the latter is unavailable when the NMS subset is enabled.

8.23.5. Exceptions

None.

8.24. ADDU.PH

8.24.1. Purpose

DSP packed operation ADDU.PH.

8.24.2. Formats

MIPS

`addu.ph`

Assembly

`ADDU.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

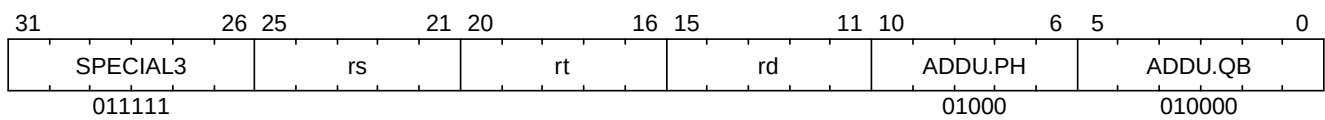


Figure 92. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`addu.ph.32`

Assembly

`ADDU.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP2

Decode priority

0

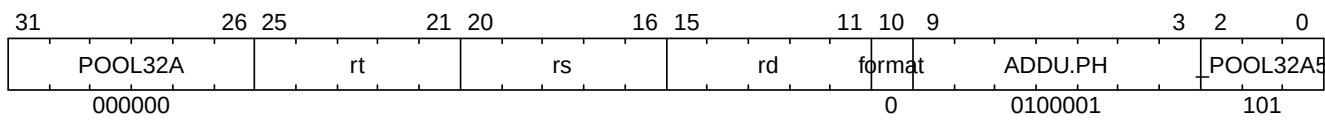


Figure 93. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`addu.ph.32`

Assembly

`ADDU.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP2

Decode priority

0

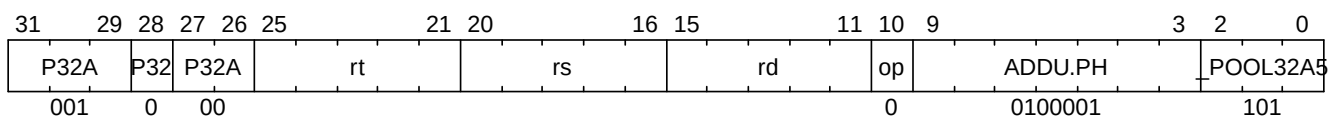


Figure 94. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.24.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.24.4. Description

ADDU.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.24.5. Exceptions

None.

8.25. ADDU.QB

8.25.1. Purpose

DSP packed operation ADDU.QB.

8.25.2. Formats

MIPS

addu.qb

Assembly

ADDU.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

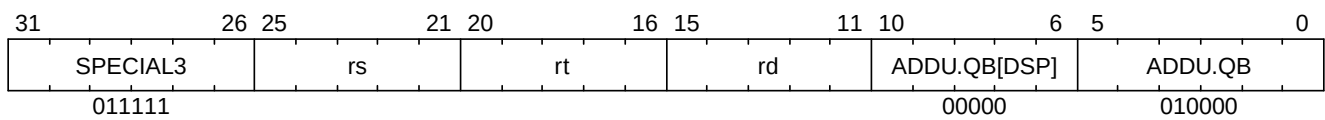


Figure 95. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addu.qb.32

Assembly

ADDU.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

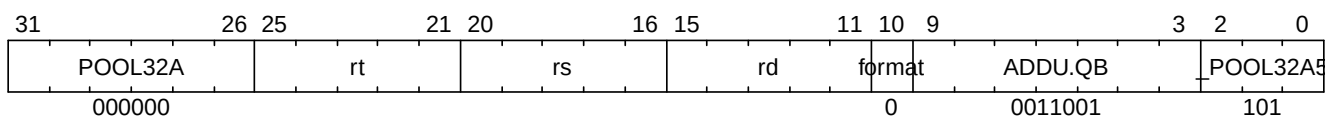


Figure 96. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`addu.qb.32`

Assembly

`ADDU.QB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

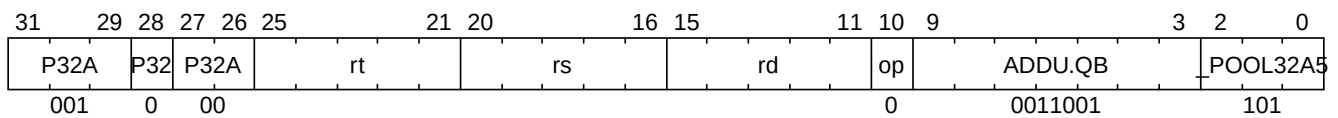


Figure 97. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.25.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4
)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.25.4. Description

ADDU.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.25.5. Exceptions

None.

8.26. ADDU_S.PH

8.26.1. Purpose

DSP packed operation ADDU_S.PH.

8.26.2. Formats

MIPS

addu_s.ph

Assembly

ADDU_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

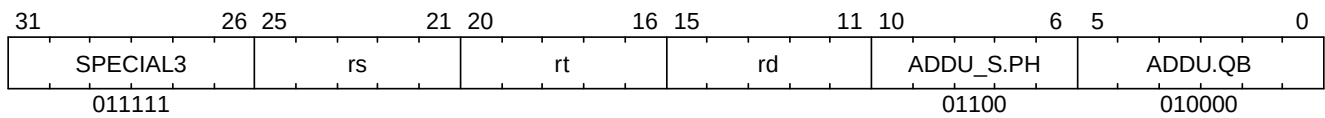


Figure 98. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addu_s.ph.32

Assembly

ADDU_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

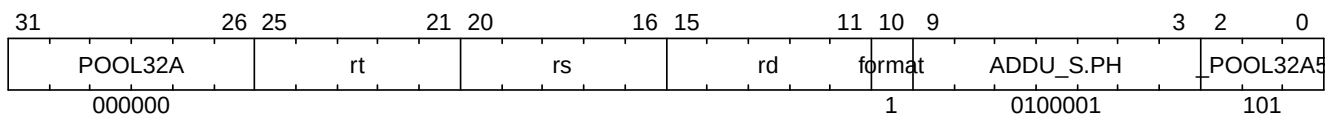


Figure 99. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

addu_s.ph.32

Assembly

ADDU_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

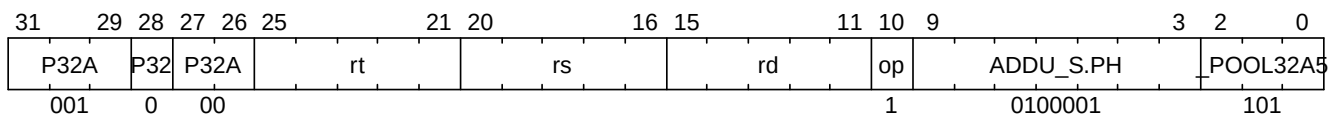


Figure 100. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.26.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.26.4. Description

ADDU_S.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.26.5. Exceptions

None.

8.27. ADDU_S.QB

8.27.1. Purpose

DSP packed operation ADDU_S.QB.

8.27.2. Formats

MIPS

addu_s.qb

Assembly

ADDU_S.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

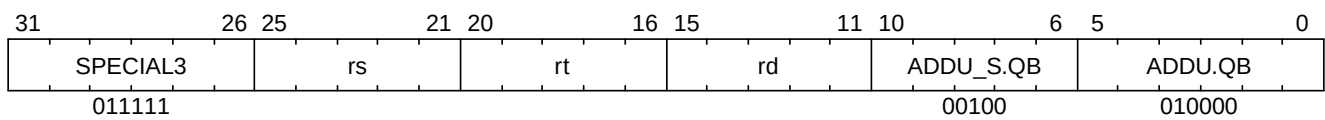


Figure 101. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

addu_s.qb.32

Assembly

ADDU_S.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

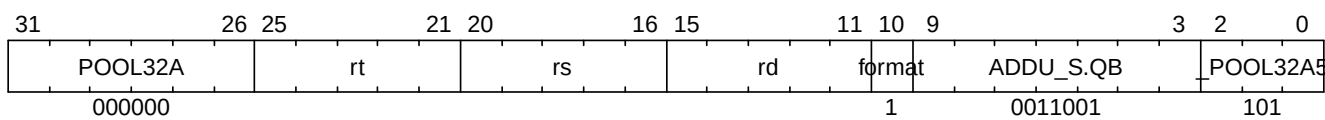


Figure 102. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

addu_s.qb.32

Assembly

ADDU_S.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

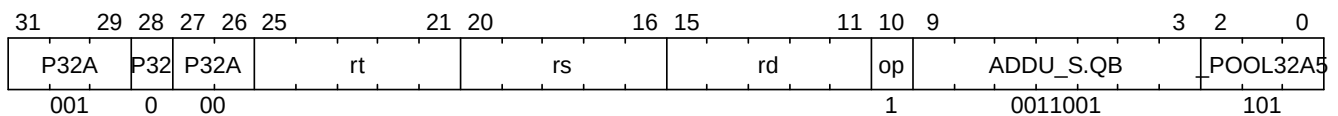


Figure 103. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.27.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.27.4. Description

ADDU_S.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.27.5. Exceptions

None.

8.28. ADDUH.QB

8.28.1. Purpose

DSP packed operation ADDUH.QB.

8.28.2. Formats

MIPS

adduh.qb

Assembly

ADDUH.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

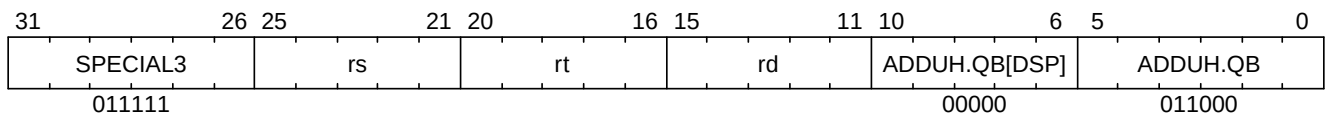


Figure 104. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

adduh.qb.32

Assembly

ADDUH.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

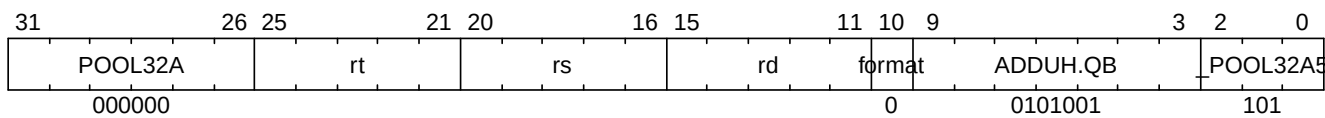


Figure 105. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`adduh.qb.32`

Assembly

`ADDUH.QB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

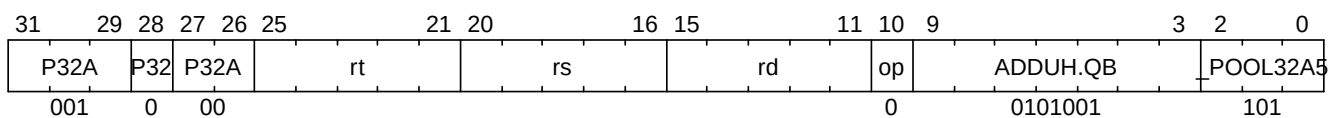


Figure 106. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.28.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, unused = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, half=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.28.4. Description

ADDUH.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.28.5. Exceptions

None.

8.29. ADDUH_R.QB

8.29.1. Purpose

DSP packed operation ADDUH_R.QB.

8.29.2. Formats

MIPS

adduh_r.qb

Assembly

ADDUH_R.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

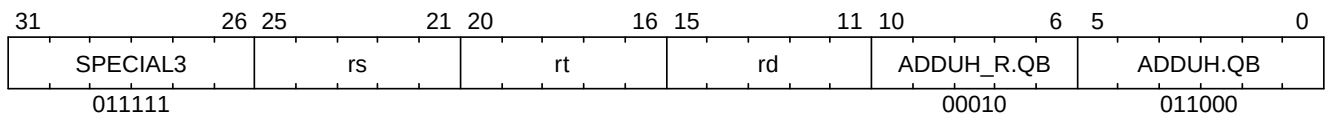


Figure 107. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

adduh_r.qb.32

Assembly

ADDUH_R.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

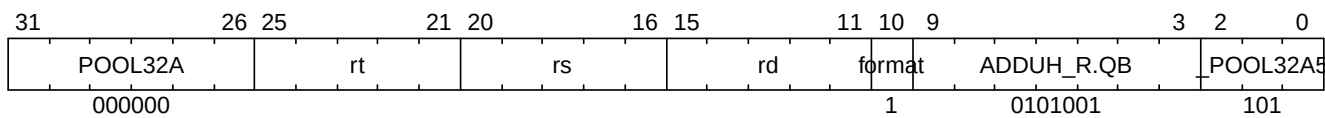


Figure 108. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

adduh_r.qb.32

Assembly

ADDUH_R.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

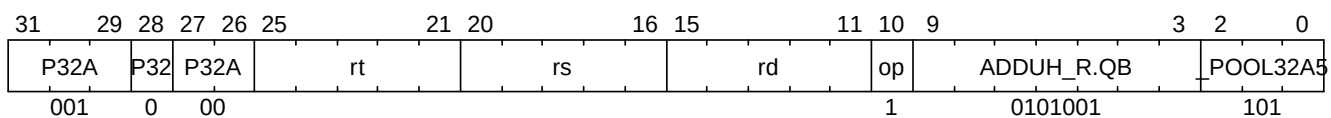


Figure 109. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.29.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, unused = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, half=True, rounded=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.29.4. Description

ADDUH_R.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.29.5. Exceptions

None.

8.30. ADDWC

8.30.1. Purpose

DSP packed operation ADDWC.

8.30.2. Formats

MIPS

`addwc`

Assembly

`ADDWC {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

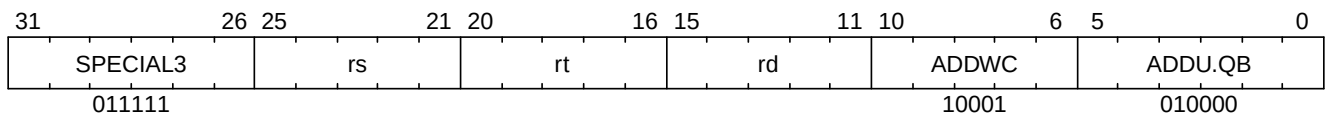


Figure 110. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`addwc.32`

Assembly

`ADDWC {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

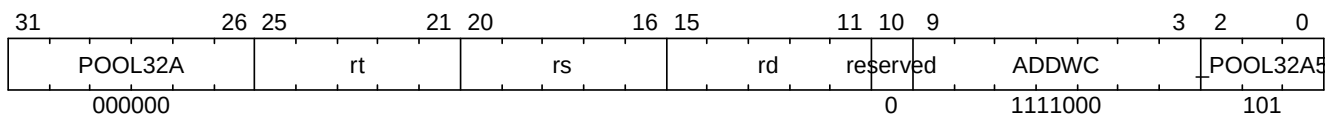


Figure 111. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`addwc.32`

Assembly

`ADDWC {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

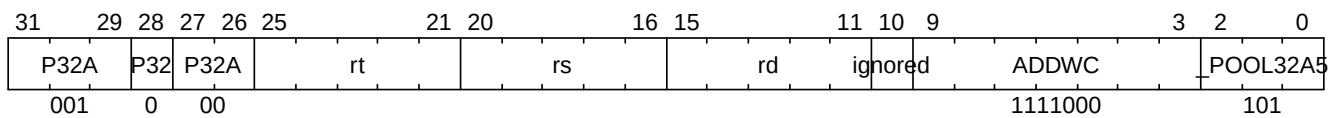


Figure 112. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.30.3. Operation

```
validate_access_to_dsp_resources(ctx)
total = (
    signed(ctx.GPR[operands.rs], 32)
    + signed(ctx.GPR[operands.rt], 32)
    + ctx.DSPControl.C
)
if total < -(1 << 31) or total > (1 << 31) - 1:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x10
ctx.GPR[operands.rd] = sign_extend(total & 0xFFFF_FFFF, from_nbits=32)
```

8.30.4. Description

ADDWC is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.30.5. Exceptions

None.

8.31. ALIGN

8.31.1. Purpose

Extract a word at a byte position from a pair of GPR words.

8.31.2. Formats

MIPS

align

Assembly

ALIGN {rd:gpr}, {rs:gpr}, {rt:gpr}, {bp:u2}

Availability

AR >= R6

Decode priority

0

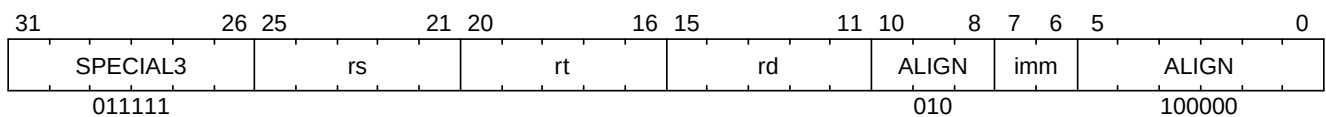


Figure 113. Encoding (32 bits)

Decode

```
return Align0perands(rd=rd, rs=rs, rt=rt, bp=zero_extend(imm, from_nbits=2))
```

microMIPS

align.32

Assembly

ALIGN {rd:gpr}, {rs:gpr}, {rt:gpr}, {bp:u2}

Availability

MMAR >= R6

Decode priority

0

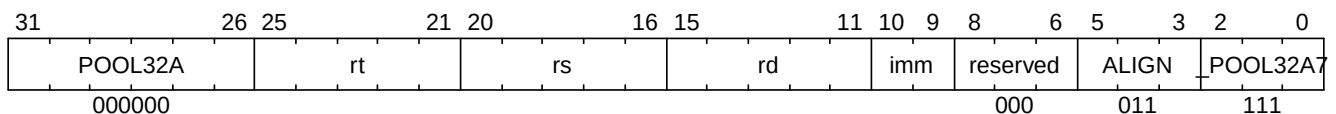


Figure 114. Encoding (32 bits)

```
return AlignOperands(rd=rd, rs=rs, rt=rt, bp=zero_extend(imm, from_nbits=2))
```

8.31.3. Operation

```
shift = (4 - int(operands.bp)) * 8  
pair = (u32(ctx.GPR[operands.rt]) << 32) | u32(ctx.GPR[operands.rs])  
ctx.GPR[operands.rd] = sign_extend(pair >> shift, from_nbits=32)
```

8.31.4. Description

ALIGN concatenates the low words of rs and rt and selects a byte-aligned word.

8.31.5. Exceptions

None.

8.32. ALNV.PS

8.32.1. Purpose

Align a misaligned pair of paired-single values.

8.32.2. Formats

MIPS

alnv.ps

Assembly

ALNV.PS {fd:fpr}, {fs:fpr}, {ft:fpr}, {rs:gpr}

Availability

AR < R6

Decode priority

0

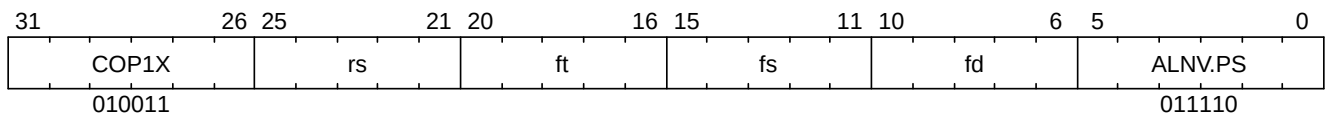


Figure 115. Encoding (32 bits)

Decode

```
return FpuPSAlignOperands(fd=fd, fs=fs, ft=ft, rs=rs)
```

microMIPS

alnv.ps.32

Assembly

ALNV.PS {fd:fpr}, {fs:fpr}, {ft:fpr}, {rs:gpr}

Availability

MMAR < R6

Decode priority

0

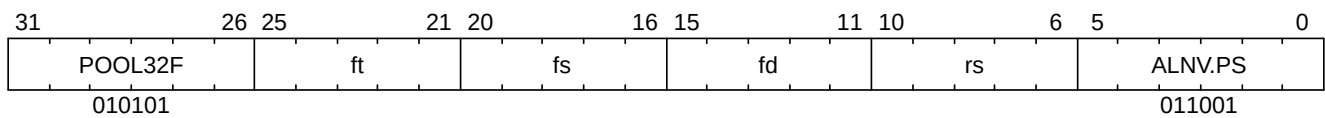


Figure 116. Encoding (32 bits)


```
return FpuPSAlignOperands(fd=fd, fs=fs, ft=ft, rs=rs)
```

8.32.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
alignment = int(ctx.GPR[operands.rs]) & 0x7
if alignment == 0:
    result = fs
else:
    # The architectural byte funnel is expressed as fs || ft in the
    # register bit order. At the 4-byte boundary, byte order determines
    # which paired-single half occupies the upper result lane.
    result = (((fs << 64) | ft) << (alignment * 8) >> 64) & 0xFFFF_FFFF_FFFF_FFFF
    if alignment == 4 and ctx.profile.endianness is Endianness.LITTLE:
        result = ((ft & 0xFFFF_FFFF) << 32) | (fs >> 32)
set_fpr(ctx, operands.fd, result, "PS")
```

8.32.4. Description

FPR[fd] receives the byte-aligned funnel of FPR[fs] and FPR[ft].

8.32.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.33. ALUIPC

8.33.1. Purpose

Add a signed aligned immediate to the sequential PC and place the result in a GPR.

8.33.2. Formats

MIPS

aluipc

Assembly

ALUIPC {rt:gpr}, {imm:s32}

Availability

AR >= R6

Decode priority

0

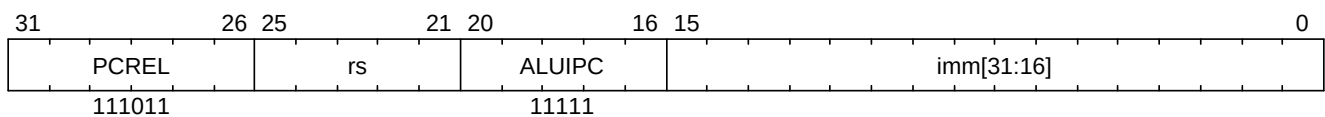


Figure 117. Encoding (32 bits)

Decode

```
return AluipcOperands(  
    rt=rs,  
    imm=sign_extend(imm, from_nbits=32),  
    base_next_pc=False,  
    alignment_mask=~0xFFFF,  
)
```

microMIPS

aluipc.32.micromips

Assembly

ALUIPC {rt:gpr}, {imm:s16}

Availability

MMAR >= R6

Decode priority

0

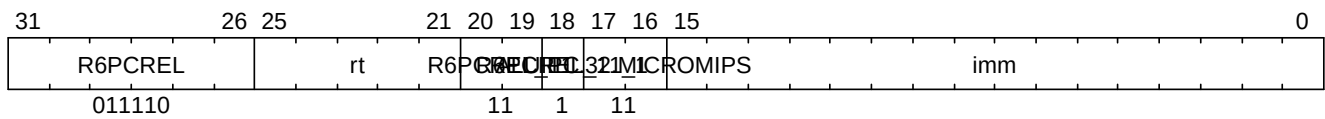


Figure 118. Encoding (32 bits)

Decode

```
del ctx, exec
return AluipcOperands(
    rt=rt,
    imm=sign_extend(int(imm) << 16, from_nbits=32),
    base_next_pc=False,
    alignment_mask=~0xFFFF,
)
```

nanoMIPS

aluipc.32

Assembly

ALUIPC {rt:gpr}, {imm:s32}

Availability

true

Decode priority

0

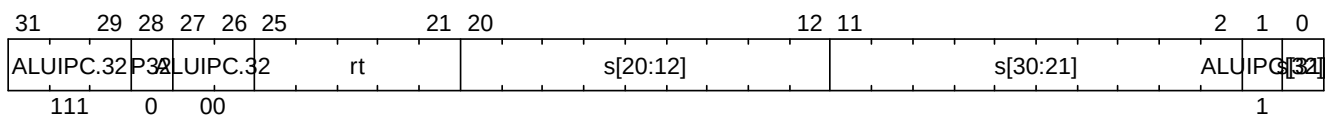


Figure 119. Encoding (32 bits)

Decode

```
del ctx, exec
return AluipcOperands(rt=rt, imm=sign_extend(s, from_nbits=32))
```

8.33.3. Operation

```
base = ctx.next_pc if operands.base_next_pc else ctx.pc
address = effective_address(ctx, base, operands.imm)
ctx.GPR[operands.rt] = address & operands.alignment_mask
```

8.33.4. Description

ALUIPC[32] adds its signed 32-bit, 4-KiB-aligned displacement to next_pc, clears the low 12 result bits, and writes the address to rt. The sequential PC is used so the instruction works correctly at a

page boundary.

8.33.5. Exceptions

None.

8.34. AND

8.34.1. Purpose

Compute the logical AND of two GPRs and place it in the destination GPR.

8.34.2. Formats

MIPS

and

Assembly

AND {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

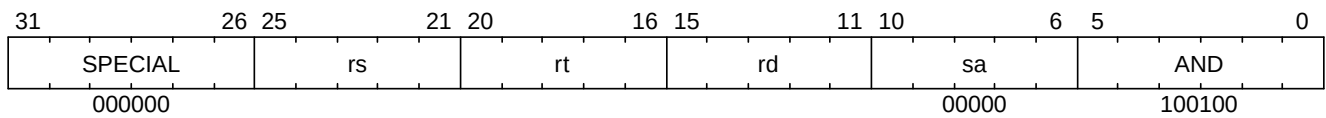


Figure 120. Encoding (32 bits)

Decode

```
del ctx, exec
return And0perands(rd=rd, rs=rs, rt=rt)
```

microMIPS

and.32

Assembly

AND {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

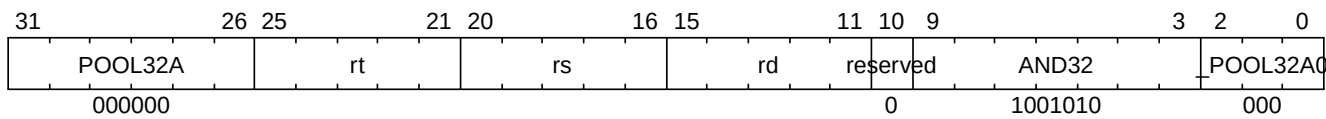


Figure 121. Encoding (32 bits)

Decode

```
del ctx, exec
return AndOperands(rd=rd, rs=rs, rt=rt)
```

and.16

Assembly

```
AND {rd:gpr3}, {rs:gpr3}
```

Availability

MMAR < R6

Decode priority

0

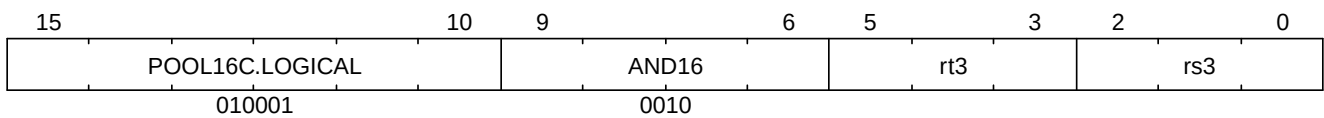


Figure 122. Encoding (16 bits)

Decode

```
del ctx, exec
rt = MICROMIPS_GPR3.decode(rt3)
rs = MICROMIPS_GPR3.decode(rs3)
return AndOperands(rd=rt, rs=rs, rt=rt)
```

and.16.r6

Assembly

```
AND {rd:gpr3}, {rs:gpr3}
```

Availability

MMAR >= R6

Decode priority

0

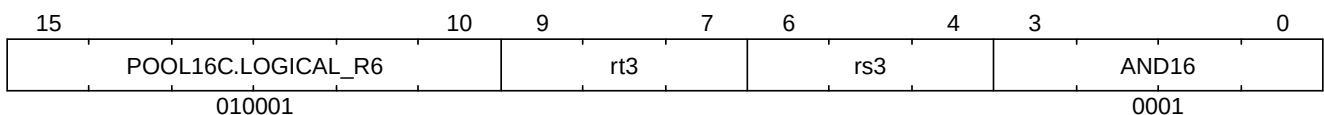


Figure 123. Encoding (16 bits)

Decode

```
del ctx, exec
rt = MICROMIPS_GPR3.decode(rt3)
rs = MICROMIPS_GPR3.decode(rs3)
return AndOperands(rd=rt, rs=rs, rt=rt)
```

nanoMIPS

and.32

Assembly

AND {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

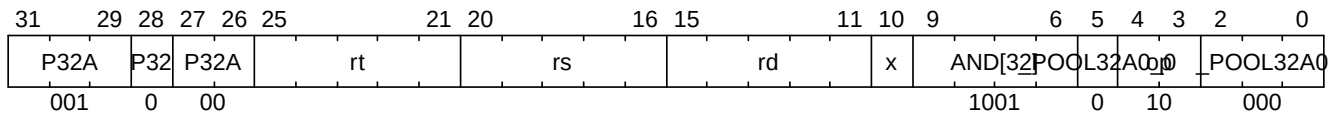


Figure 124. Encoding (32 bits)

Decode

```
return And0operands(rd=rd, rs=rs, rt=rt)
```

and.16

Assembly

AND {rd:gpr3}, {rs:gpr3}

Availability

true

Decode priority

0

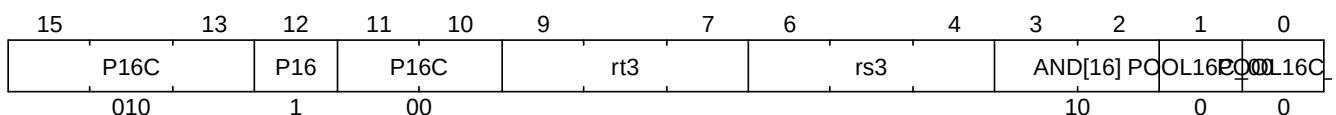


Figure 125. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
return And0operands(rd=rt, rs=rs, rt=rt)
```

8.34.3. Operation

```
ctx.GPR[operands.rd] = ctx.GPR[operands.rs] & ctx.GPR[operands.rt]
```

8.34.4. Description

The nanoMIPS[32] form names an independent destination. The compact nanoMIPS[16] form aliases its encoded rt3 register as both source and destination. The operation is performed across the full implemented GPR width.

8.34.5. Exceptions

None.

8.35. ANDI

8.35.1. Purpose

Compute the logical AND of a GPR and an unsigned immediate.

8.35.2. Formats

MIPS

andi

Assembly

ANDI {rt:gpr}, {rs:gpr}, {imm:u16}

Availability

true

Decode priority

0

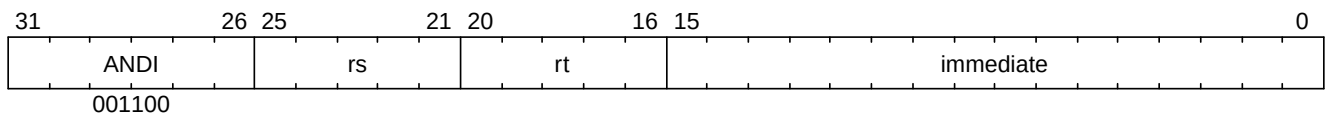


Figure 126. Encoding (32 bits)

Decode

```
del ctx, exec
return AndiOperands(rt=rt, rs=rs, imm=zero_extend(immediate, from_nbits=16))
```

microMIPS

andi.32

Assembly

ANDI {rt:gpr}, {rs:gpr}, {imm:u16}

Availability

true

Decode priority

0

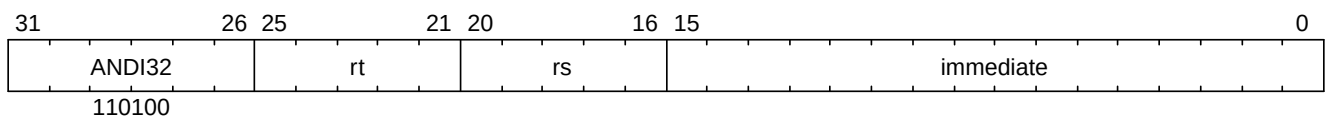


Figure 127. Encoding (32 bits)

Decode

```
del ctx, exec
return AndiOperands(rt=rt, rs=rs, imm=zero_extend(immediate, from_nbits=16))
```

andi.16

Assembly

ANDI {rt:gpr3}, {rs:gpr3}, {imm:u16}

Availability

true

Decode priority

0

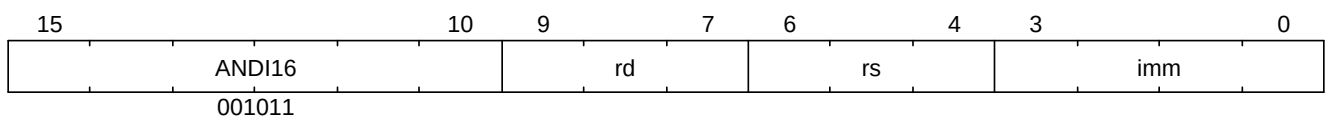


Figure 128. Encoding (16 bits)

Decode

```
del ctx, exec
decoded_imm = zero_extend(_MICROMIPS_ANDI16_IMMEDIATES[imm], from_nbits=16)
return AndiOperands(
    rt=MICROMIPS_GPR3.decode(rd),
    rs=MICROMIPS_GPR3.decode(rs),
    imm=decoded_imm,
)
```

nanoMIPS

andi.32

Assembly

ANDI {rt:gpr}, {rs:gpr}, {imm:u12}

Availability

true

Decode priority

0

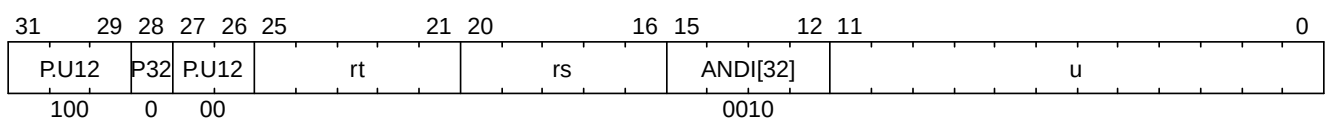


Figure 129. Encoding (32 bits)

Decode

```
imm = zero_extend(u, from_nbits=12)
return AndiOperands(rt=rt, rs=rs, imm=imm)
```

andi.16

Assembly

ANDI {rt:gpr3}, {rs:gpr3}, {imm:u16}

Availability

true

Decode priority

0

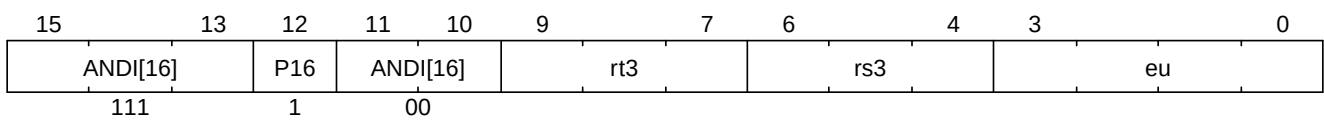


Figure 130. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
u = 0x00FF if eu == 12 else 0xFFFF if eu == 13 else eu
imm = zero_extend(u, from_nbits=16)
return AndiOperands(rt=rt, rs=rs, imm=imm)
```

8.35.3. Operation

```
lhs = ctx.GPR[operands.rs]
rhs = zero_extend(operands.imm, from_nbits=32)
ctx.GPR[operands.rt] = lhs & int(rhs)
```

8.35.4. Description

The immediate is zero-extended before the operation. ANDI[16] uses the compact eu field and reserves eu=12 and eu=13 as aliases for 0xff and 0xffff respectively. The source and result use the full implemented GPR width.

8.35.5. Exceptions

None.

8.36. APPEND

8.36.1. Purpose

DSP bitstream operation APPEND.

8.36.2. Formats

MIPS

append

Assembly

APPEND {rt:gpr}, {rs:gpr}, {sa:u5}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

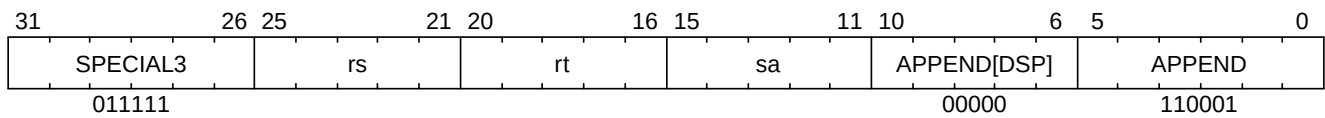


Figure 131. Encoding (32 bits)

Decode

```
return DspAppendOperands(rt=rt, rs=rs, sa=sa)
```

microMIPS

append.32

Assembly

APPEND {rt:gpr}, {rs:gpr}, {sa:u5}

Availability

DSP2

Decode priority

0

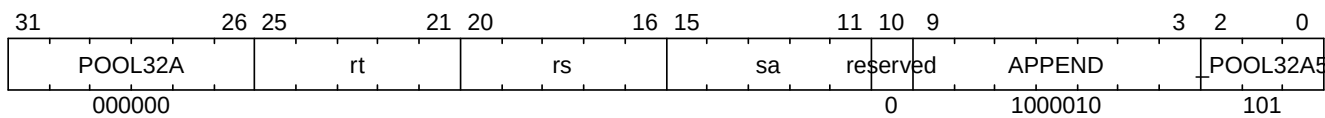


Figure 132. Encoding (32 bits)

Decode

```
return DspAppendOperands(rt=rt, rs=rs, sa=sa)
```

nanoMIPS

append.32

Assembly

APPEND {rt:gpr}, {rs:gpr}, {sa:u5}

Availability

DSP2

Decode priority

0

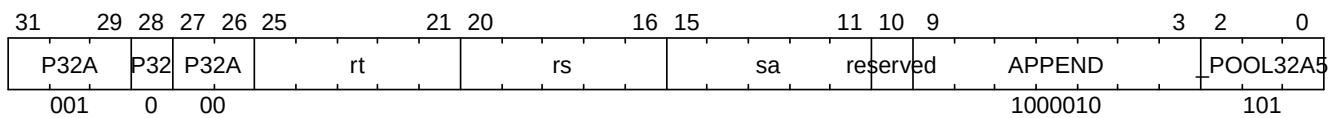


Figure 133. Encoding (32 bits)

Decode

```
return DspAppendOperands(rt=rt, rs=rs, sa=sa)
```

8.36.3. Operation

```
validate_access_to_dsp2_resources(ctx)
amount = operands.sa & 0x1F
if amount:
    target = (ctx.GPR[operands.rt] & 0xFFFF_FFFF) << amount
    source = ctx.GPR[operands.rs] & ((1 << amount) - 1)
    ctx.GPR[operands.rt] = sign_extend(
        (target | source) & 0xFFFF_FFFF,
        from_nbits=32,
    )
```

8.36.4. Description

APPEND is a DSP bitstream or accumulator extraction operation.

8.36.5. Exceptions

None.

8.37. AUI

8.37.1. Purpose

Add a signed upper immediate to a GPR.

8.37.2. Formats

MIPS

aui

Assembly

AUI {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

AR >= R6

Decode priority

0

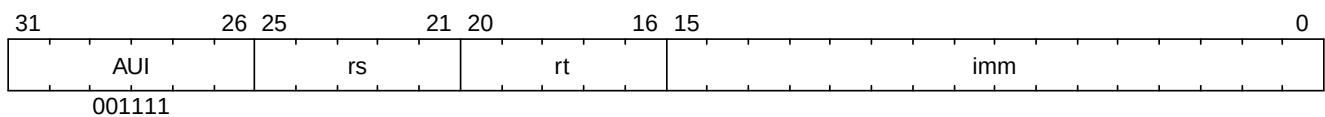


Figure 134. Encoding (32 bits)

Decode

```
return Aui0operands(rt=rt, rs=rs, imm=sign_extend(imm, from_nbits=16))
```

microMIPS

aui.32

Assembly

AUI {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

MMAR >= R6

Decode priority

0

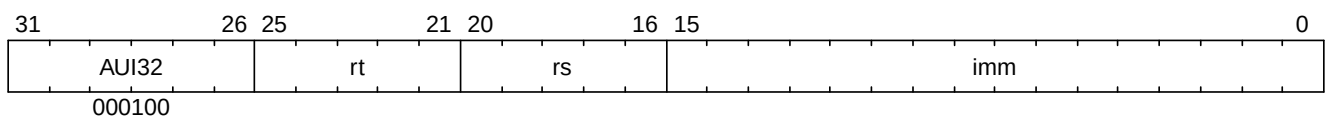


Figure 135. Encoding (32 bits)

```
return AuiOperands(rt=rt, rs=rs, imm=sign_extend(imm, from_nbits=16))
```

8.37.3. Operation

```
result = ctx.GPR[operands.rs] + (int(operands.imm) << 16)
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.37.4. Description

AUI sign-extends its 16-bit immediate, shifts it left by 16, adds it to rs, and sign-extends the low 32-bit result to the GPR width.

8.37.5. Exceptions

None.

8.38. AUIPC

8.38.1. Purpose

Add a signed upper immediate to the current PC.

8.38.2. Formats

MIPS

`auipc`

Assembly

`AUIPC {rs:gpr}, {offset:s32}`

Availability

AR >= R6

Decode priority

0

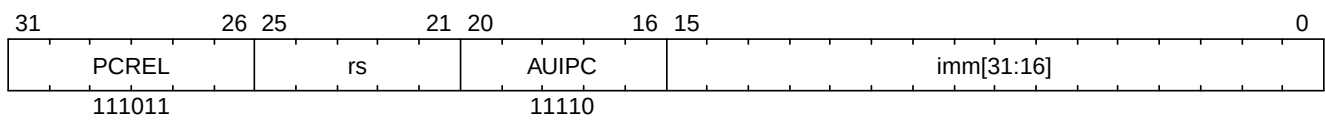


Figure 136. Encoding (32 bits)

Decode

```
return AuipcOperands(rs=rs, offset=sign_extend(imm, from_nbits=32))
```

microMIPS

`auipc.32.micromips`

Assembly

`AUIPC {rs:gpr}, {offset:s16}`

Availability

MMAR >= R6

Decode priority

0

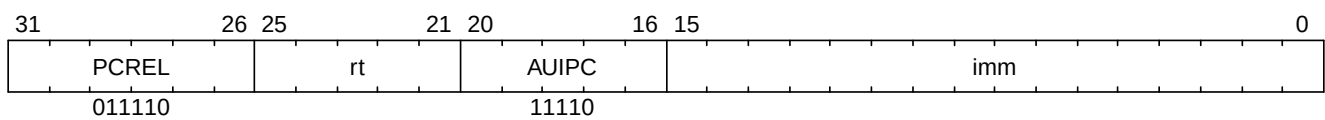


Figure 137. Encoding (32 bits)

Decode

```
del ctx, exec
return AuipcOperands(rs=rt, offset=sign_extend(imm, from_nbits=16) << 16)
```

8.38.3. Operation

```
ctx.GPR[operands.rs] = effective_address(ctx, ctx.pc, operands.offset)
```

8.38.4. Description

AUIPC adds its signed, 16-bit upper immediate displacement to the address of the instruction and writes the result to rs.

8.38.5. Exceptions

None.

8.39. BAL

8.39.1. Purpose

Perform an unconditional PC-relative procedure call, place the return address in GPR 31, and execute one branch delay slot.

8.39.2. Formats

MIPS

bal

Assembly

BAL {offset:s18}

Availability

AR >= R6

Decode priority

0

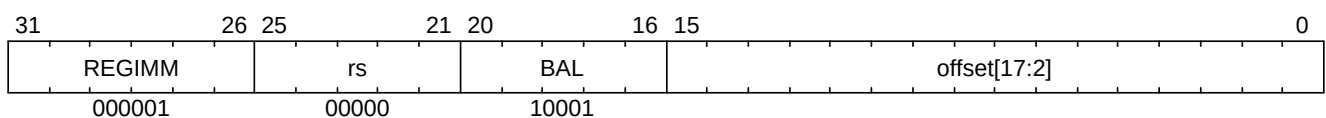


Figure 138. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
offset = sign_extend(offset, from_nbits=18)
return Bal0operands(offset=offset)
```

8.39.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, address)
ctx.GPR[31] = effective_address(ctx, ctx.next_pc, 4)
```

8.39.4. Description

The signed, word-aligned displacement is added to the address of the instruction following BAL. GPR 31 receives the address after the delay-slot instruction. BAL is the architectural BGEZAL-with-rs-zero case retained by Release 6.

8.39.5. Exceptions

None.

8.40. BALC

8.40.1. Purpose

Perform an unconditional compact PC-relative procedure call and place the address of the following instruction in GPR 31.

8.40.2. Formats

MIPS

`balc`

Assembly

`BALC {offset:s28}`

Availability

AR >= R6

Decode priority

0

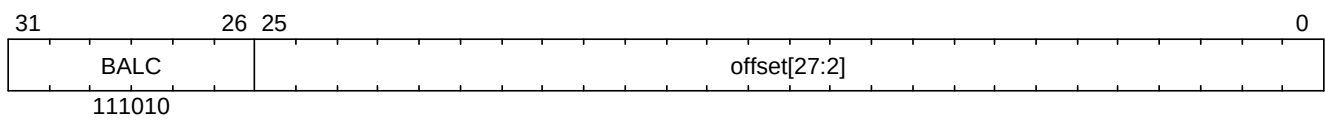


Figure 139. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(offset, from_nbits=28)
return BalcOperands(offset=offset)
```

microMIPS

`balc.32`

Assembly

`BALC {offset:s27}`

Availability

MMAR >= R6

Decode priority

0

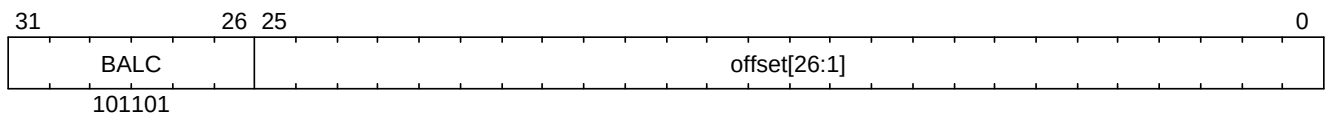


Figure 140. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(offset, from_nbits=27)
return BalcOperands(offset=offset)
```

nanoMIPS

balc.16

Assembly

BALC {offset:s11}

Availability

true

Decode priority

0

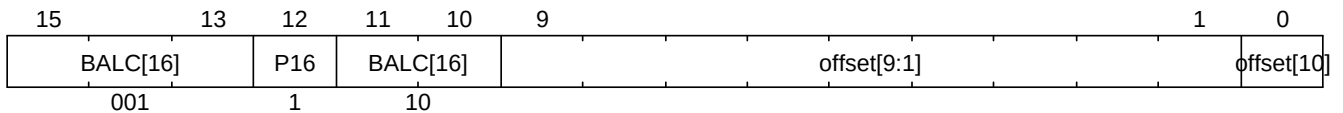


Figure 141. Encoding (16 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(offset, from_nbits=11)
return BalcOperands(offset=offset)
```

b1c.32

Assembly

BALC {offset:s26}

Availability

true

Decode priority

0

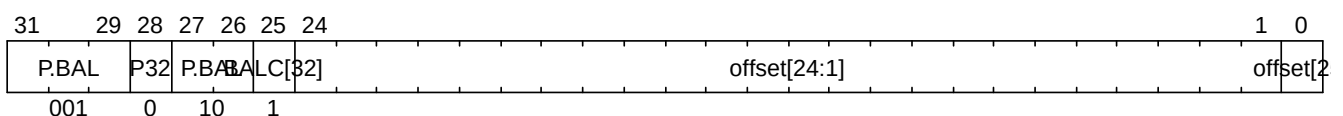


Figure 142. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(offset, from_nbits=26)
```

```
return BalcOperands(offset=offset)
```

8.40.3. Operation

```
link_address = ctx.next_pc
address = effective_address(ctx, link_address, operands.offset)
compact_branch(ctx, address)
# A link value is later consumed by a register jump. On a processor
# implementing both MIPS and microMIPS, preserve the current ISA mode in
# that value (bit zero is the mode tag); otherwise a BALC from microMIPS
# returns into the MIPS decoder.
ctx.GPR[31] = ModeTaggedAddress(ctx, link_address)
```

8.40.4. Description

The signed displacement is added to the address of the instruction following BALC. The branch has no delay slot. Because it is always taken, BALC also has no forbidden slot.

8.40.5. Exceptions

None.

8.41. BALIGN

8.41.1. Purpose

DSP bitstream operation BALIGN.

8.41.2. Formats

MIPS

`balign`

Assembly

`BALIGN {rt:gpr}, {rs:gpr}, {bp:u2}`

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

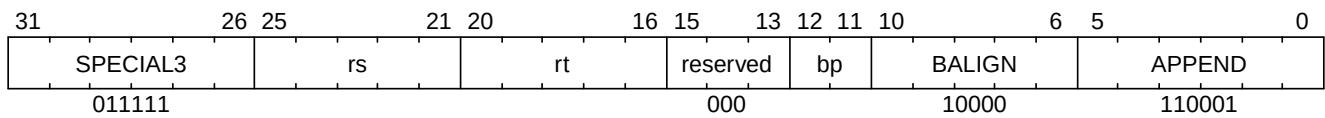


Figure 143. Encoding (32 bits)

Decode

```
return DspAlignOperands(rt=rt, rs=rs, bp=bp)
```

microMIPS

`balign.32`

Assembly

`BALIGN {rt:gpr}, {rs:gpr}, {bp:u2}`

Availability

DSP2

Decode priority

0

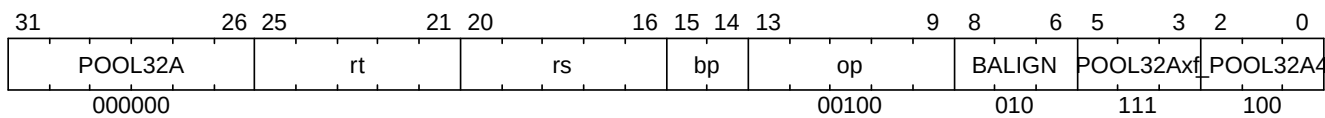


Figure 144. Encoding (32 bits)

```
return DspAlignOperands(rt=rt, rs=rs, bp=bp)
```

8.41.3. Operation

```
validate_access_to_dsp_resources(ctx)
pair = ((ctx.GPR[operands.rt] & 0xFFFF_FFFF) << 32) | (
    ctx.GPR[operands.rs] & 0xFFFF_FFFF
)
result = pair >> (8 * (4 - (operands.bp & 3)))
ctx.GPR[operands.rt] = sign_extend(
    result & 0xFFFF_FFFF,
    from_nbits=32,
)
```

8.41.4. Description

BALIGN is a DSP bitstream operation.

8.41.5. Exceptions

None.

8.42. BALRSC

8.42.1. Purpose

Branch to a scaled register-relative address and write a link GPR.

8.42.2. Formats

nanoMIPS

balrsc.32

Assembly

BALRSC {link:gpr}, {rs:gpr}

Availability

true

Decode priority

0

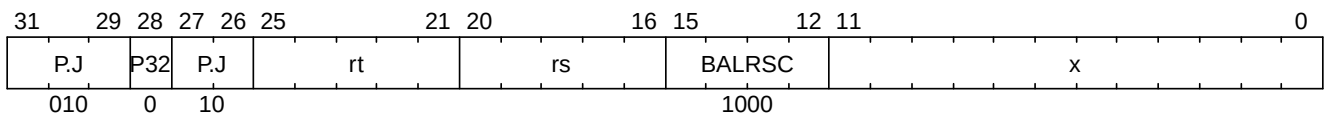


Figure 145. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
return BalrscOperands(rs=rs, link=rt)
```

8.42.3. Operation

```
target = effective_address(ctx, ctx.next_pc, ctx.GPR[operands.rs] << 1)
ctx.GPR[operands.link] = zero_extend(ctx.next_pc, from_nbits=ctx.GPR.width)
compact_branch(ctx, target)
```

8.42.4. Description

BALRSC adds twice the source GPR to the address following the instruction, writes that following address to the link GPR, and performs a compact transfer without a delay or forbidden slot.

8.42.5. Exceptions

None.

8.43. BBEQZC

8.43.1. Purpose

Branch compactly when a selected GPR bit is zero.

8.43.2. Formats

nanoMIPS

bbeqzc.32

Assembly

BBEQZC {bit:u6}, {offset:s12}

Availability

!(NMS)

Decode priority

0

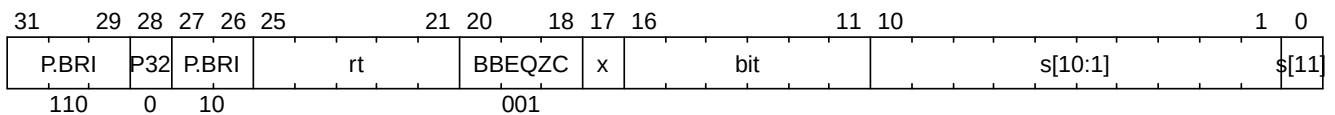


Figure 146. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(s, from_nbits=12)
return BranchBitOperands(rt=rt, bit=bit, offset=offset)
```

8.43.3. Operation

```
if operands.bit >= 32 and not Are64BitOperationsEnabled(ctx):
    raise ReservedInstruction("bit 32 or above requires 64-bit GPR operations")
target = effective_address(ctx, ctx.next_pc, operands.offset)
value = (ctx.GPR[operands.rt] >> operands.bit) & 1
compact_branch(
    ctx, target, taken=value == 0, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.43.4. Description

BBEQZC is unavailable in the nanoMIPS subset.

8.43.5. Exceptions

- **RI** (Reserved Instruction) — bit 32 or above requires 64-bit GPR operations

8.44. BBNEZC

8.44.1. Purpose

Branch compactly when a selected GPR bit is one.

8.44.2. Formats

nanoMIPS

bbnezc.32

Assembly

BBNEZC {bit:u6}, {offset:s12}

Availability

!(NMS)

Decode priority

0

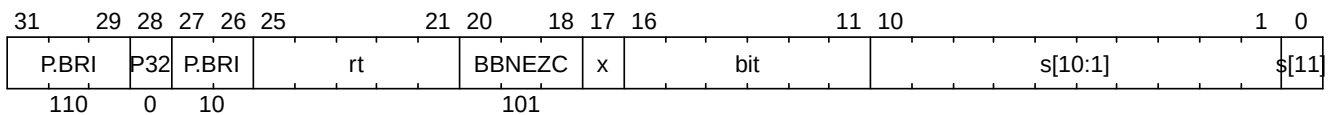


Figure 147. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(s, from_nbits=12)
return BranchBitOperands(rt=rt, bit=bit, offset=offset)
```

8.44.3. Operation

```
if operands.bit >= 32 and not Are64BitOperationsEnabled(ctx):
    raise ReservedInstruction("bit 32 or above requires 64-bit GPR operations")
target = effective_address(ctx, ctx.next_pc, operands.offset)
value = (ctx.GPR[operands.rt] >> operands.bit) & 1
compact_branch(
    ctx, target, taken=value == 1, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.44.4. Description

BBNEZC is unavailable in the nanoMIPS subset.

8.44.5. Exceptions

- **RI** (Reserved Instruction) — bit 32 or above requires 64-bit GPR operations

8.45. BC

8.45.1. Purpose

Perform an unconditional compact PC-relative branch.

8.45.2. Formats

MIPS

bc

Assembly

BC {offset:s28}

Availability

AR >= R6

Decode priority

0

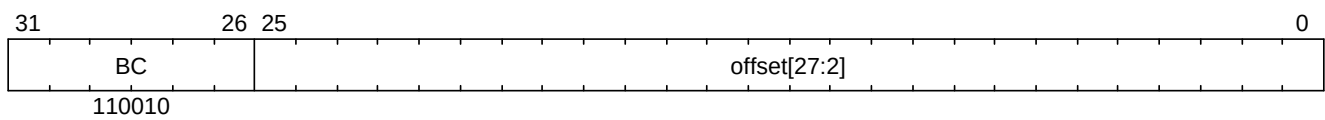


Figure 148. Encoding (32 bits)

Decode

```
exec.isCTI = True
return BcOperands(offset=sign_extend(offset, from_nbits=28))
```

microMIPS

bc.r6.16

Assembly

BC {offset:s11}

Availability

MMAR >= R6

Decode priority

0

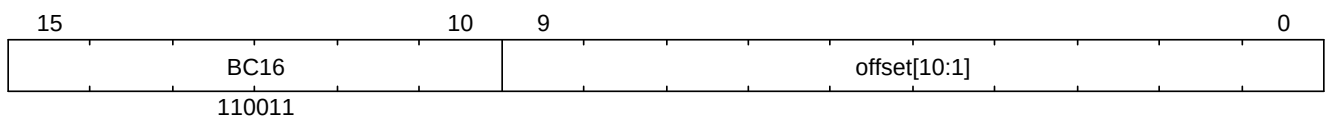


Figure 149. Encoding (16 bits)

Decode

```
exec.isCTI = True
return BcOperands(offset=sign_extend(offset, from_nbits=11))
```

bc.r6.32

Assembly

BC {offset:s27}

Availability

MMAR >= R6

Decode priority

0

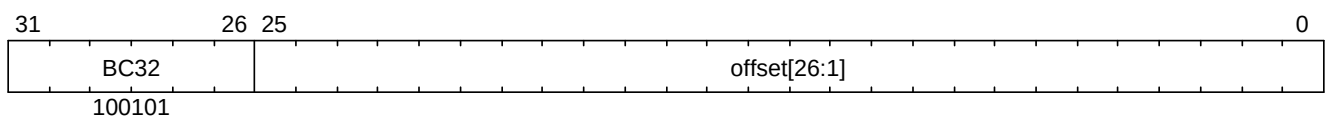


Figure 150. Encoding (32 bits)

Decode

```
exec.isCTI = True
return BcOperands(offset=sign_extend(offset, from_nbits=27))
```

nanoMIPS

bc.16

Assembly

BC {offset:s10}

Availability

true

Decode priority

0

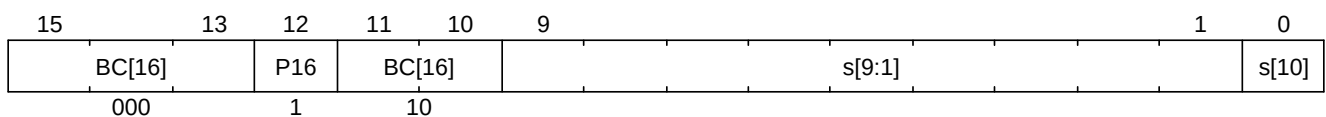


Figure 151. Encoding (16 bits)

Decode

```
exec.isCTI = True
s = sign_extend(s, from_nbits=11)
return BcOperands(offset=s)
```


bc.32

Assembly

BC {offset:s25}

Availability

true

Decode priority

0

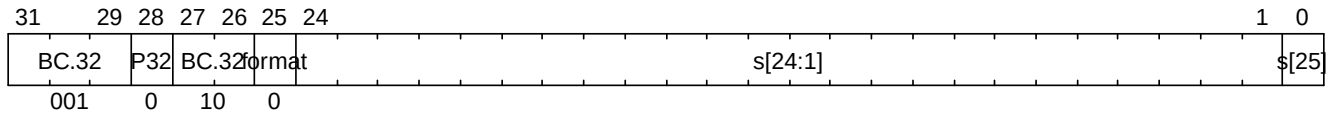


Figure 152. Encoding (32 bits)

Decode

```
exec.isCTI = True
s = sign_extend(s, from_nbits=26)
return BcOperands(offset=s)
```

8.45.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(ctx, address)
```

8.45.4. Description

The signed displacement is added to the address of the instruction following BC. BC has no delay slot and does not create a forbidden slot. The 16-bit and 32-bit nanoMIPS encodings use halfword-aligned displacements; the encoded low zero bit is included in the signed extension width.

8.45.5. Exceptions

None.

8.46. BC1EQZ

8.46.1. Purpose

Branch with a delay slot when FPR ft bit zero is clear.

8.46.2. Formats

MIPS

bc1eqz

Assembly

BC1EQZ {ft:fpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

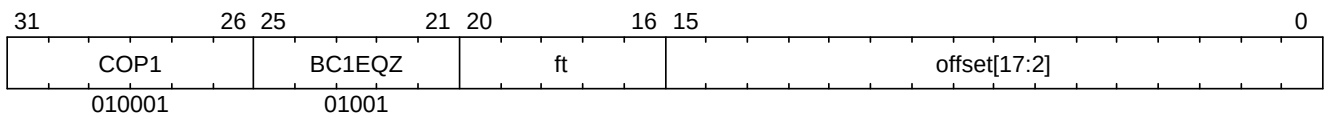


Figure 153. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc1Eqz0operands(ft=ft, offset=sign_extend(offset, from_nbits=18))
```

8.46.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=get_fpr(ctx, operands.ft, "32") & 1 == 0)
```

8.46.4. Description

BC1EQZ is the classic MIPS Release 6 FPR-bit branch.

8.46.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.47. BC1EQZC

8.47.1. Purpose

Branch compactly when FPR ft bit zero is clear.

8.47.2. Formats

microMIPS

bc1eqzc.32

Assembly

BC1EQZC {ft:fpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

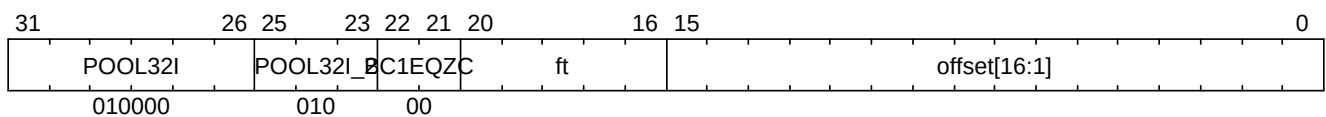


Figure 154. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return Bc1Eqz0operands(ft=ft, offset=sign_extend(offset, from_nbits=17))
```

nanoMIPS

bc1eqzc.32

Assembly

BC1EQZC {ft:fpr}, {offset:s15}

Availability

true

Decode priority

0

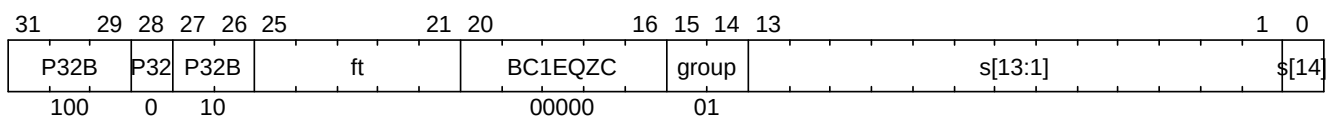


Figure 155. Encoding (32 bits)

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return Bc1EqzOperands(ft=ft, offset=sign_extend(s, from_nbits=15))
```

8.47.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(ctx, target, taken=get_fpr(ctx, operands.ft, "32") & 1 == 0)
```

8.47.4. Description

BC1EQZC is the microMIPS and nanoMIPS Release 6 FPR-bit compact branch.

8.47.5. Exceptions

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.48. BC1F

8.48.1. Purpose

Branch when FCC[cc] is false.

8.48.2. Formats

MIPS

bc1f

Assembly

BC1F {cc:u3}, {offset:s18}

Availability

AR < R6

Decode priority

0

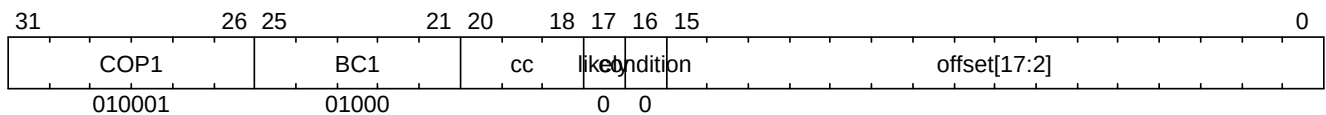


Figure 156. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc10perands(cc=cc, offset=sign_extend(offset, from_nbits=18))
```

microMIPS

bc1f.32

Assembly

BC1F {cc:u3}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

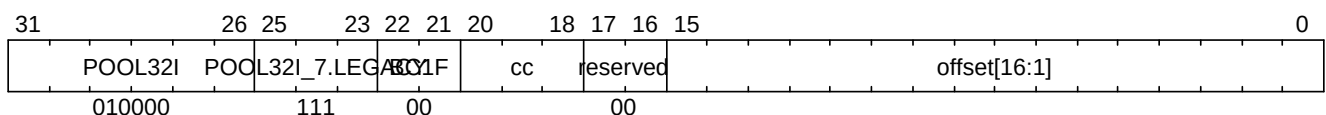


Figure 157. Encoding (32 bits)

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc10perands(cc=cc, offset=sign_extend(offset, from_nbits=17))
```

8.48.3. Operation

```
validate_access_to_cp1_resources(ctx)
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=read_fcc(ctx, operands.cc) == 0)
```

8.48.4. Description

BC1F performs a CP1 access check and uses the selected floating condition code.

8.48.5. Exceptions

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.49. BC1FL

8.49.1. Purpose

Branch when FCC[cc] is false.

8.49.2. Formats

MIPS

bc1fl

Assembly

BC1FL {cc:u3}, {offset:s18}

Availability

AR < R6

Decode priority

0

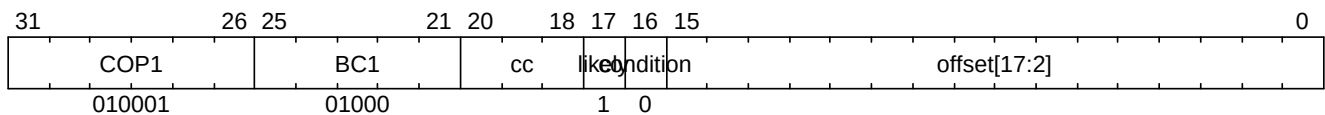


Figure 158. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc10operands(cc=cc, offset=sign_extend(offset, from_nbits=18))
```

8.49.3. Operation

```
validate_access_to_cp1_resources(ctx)
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=read_fcc(ctx, operands.cc) == 0)
```

8.49.4. Description

BC1FL performs a CP1 access check and uses the selected floating condition code.

8.49.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.50. BC1NEZ

8.50.1. Purpose

Branch with a delay slot when FPR ft bit zero is set.

8.50.2. Formats

MIPS

`bc1nez`

Assembly

`BC1NEZ {ft:fpr}, {offset:s18}`

Availability

AR >= R6

Decode priority

0

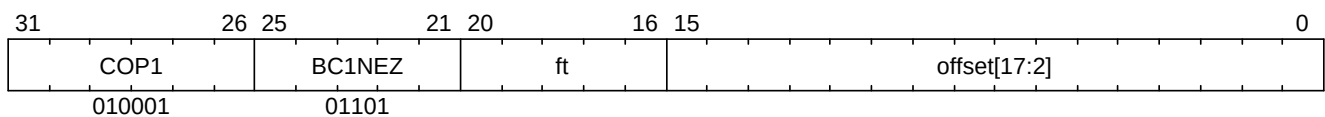


Figure 159. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc1Eqz0operands(ft=ft, offset=sign_extend(offset, from_nbits=18))
```

8.50.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=get_fpr(ctx, operands.ft, "32") & 1 != 0)
```

8.50.4. Description

BC1NEZ is the classic MIPS Release 6 FPR-bit branch.

8.50.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.51. BC1NEZC

8.51.1. Purpose

Branch compactly when FPR ft bit zero is set.

8.51.2. Formats

microMIPS

bc1nezc.32

Assembly

BC1NEZC {ft:fpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

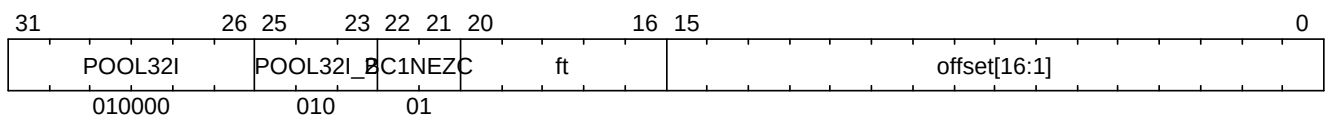


Figure 160. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return Bc1Eqz0operands(ft=ft, offset=sign_extend(offset, from_nbits=17))
```

nanoMIPS

bc1nezc.32

Assembly

BC1NEZC {ft:fpr}, {offset:s15}

Availability

true

Decode priority

0

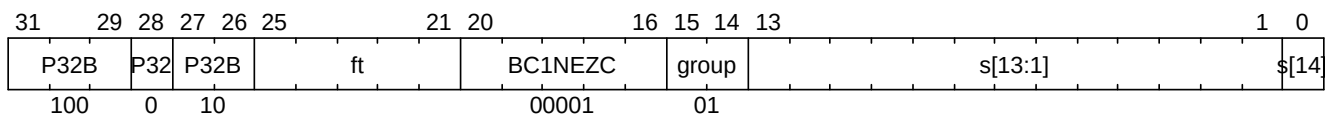


Figure 161. Encoding (32 bits)

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return Bc1EqzOperands(ft=ft, offset=sign_extend(s, from_nbits=15))
```

8.51.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(ctx, target, taken=get_fpr(ctx, operands.ft, "32") & 1 != 0)
```

8.51.4. Description

BC1NEZC is the microMIPS and nanoMIPS Release 6 FPR-bit compact branch.

8.51.5. Exceptions

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.52. BC1T

8.52.1. Purpose

Branch when FCC[cc] is true.

8.52.2. Formats

MIPS

bc1t

Assembly

BC1T {cc:u3}, {offset:s18}

Availability

AR < R6

Decode priority

0

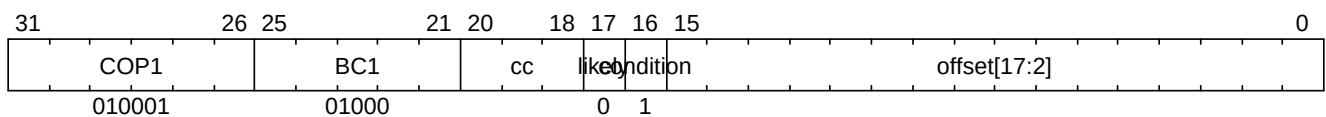


Figure 162. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc1Operands(cc=cc, offset=sign_extend(offset, from_nbits=18))
```

microMIPS

bc1t.32

Assembly

BC1T {cc:u3}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

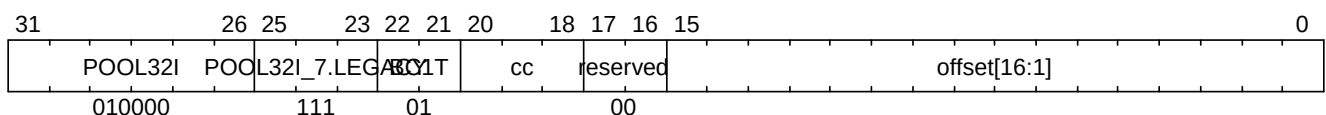


Figure 163. Encoding (32 bits)

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc10operands(cc=cc, offset=sign_extend(offset, from_nbits=17))
```

8.52.3. Operation

```
validate_access_to_cp1_resources(ctx)
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=read_fcc(ctx, operands.cc) != 0)
```

8.52.4. Description

BC1T performs a CP1 access check and uses the selected floating condition code.

8.52.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.53. BC1TL

8.53.1. Purpose

Branch when FCC[cc] is true.

8.53.2. Formats

MIPS

bc1tl

Assembly

BC1TL {cc:u3}, {offset:s18}

Availability

AR < R6

Decode priority

0

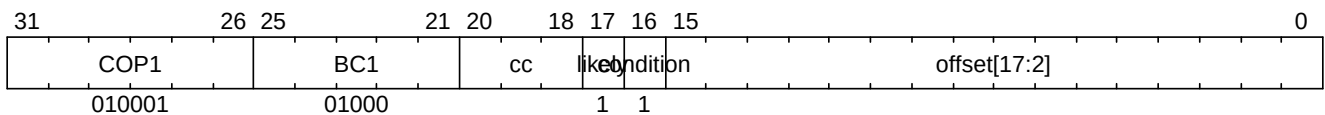


Figure 164. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc10operands(cc=cc, offset=sign_extend(offset, from_nbits=18))
```

8.53.3. Operation

```
validate_access_to_cp1_resources(ctx)
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=read_fcc(ctx, operands.cc) != 0)
```

8.53.4. Description

BC1TL performs a CP1 access check and uses the selected floating condition code.

8.53.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.54. BC2EQZC

8.54.1. Purpose

Branch compactly when a CP2 register is zero.

8.54.2. Formats

microMIPS

`bc2eqzc.32`

Assembly

`BC2EQZC {ct:u5}, {offset:s17}`

Availability

MMAR $\geq$ R6

Decode priority

0

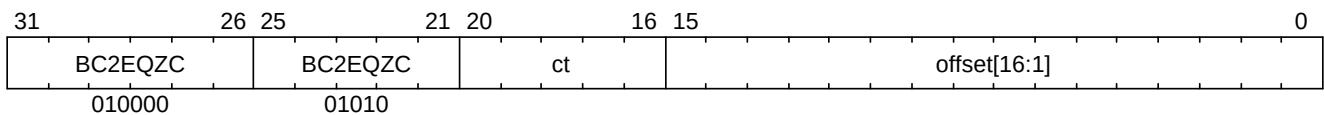


Figure 165. Encoding (32 bits)

Decode

```
del ctx
exec.isCTI = True
return Bc2Eqz0operands(ct=ct, offset=sign_extend(offset, from_nbits=17))
```

8.54.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("BC2EQZC execution is not implemented")
```

8.54.4. Description

BC2EQZC checks CP2 availability before evaluating the condition.

8.54.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; BC2EQZC execution is not implemented

8.55. BC2F

8.55.1. Purpose

Branch on a false CP2 condition.

8.55.2. Formats

microMIPS

bc2f.32

Assembly

BC2F {cc:u3}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

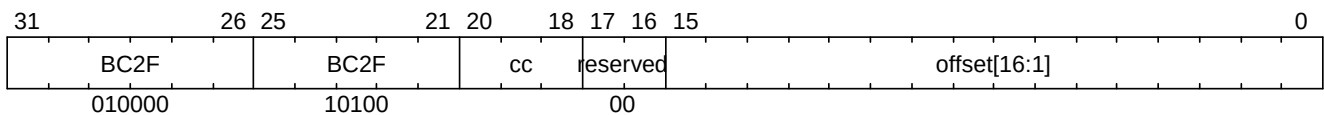


Figure 166. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc20operands(cc=cc, offset=sign_extend(offset, from_nbits=17))
```

8.55.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("BC2F execution is not implemented")
```

8.55.4. Description

BC2F checks CP2 availability before evaluating the condition.

8.55.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; BC2F execution is not implemented

8.56. BC2NEZC

8.56.1. Purpose

Branch compactly when a CP2 register is nonzero.

8.56.2. Formats

microMIPS

`bc2nezc.32`

Assembly

`BC2NEZC {ct:u5}, {offset:s17}`

Availability

MMAR $\geq$ R6

Decode priority

0

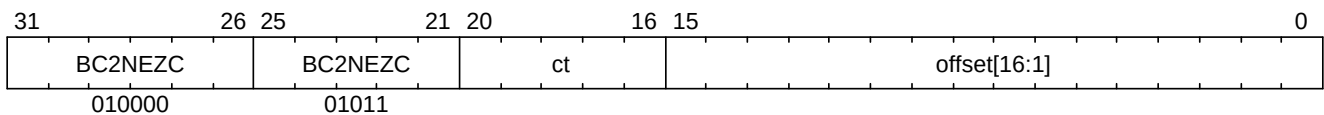


Figure 167. Encoding (32 bits)

Decode

```
del ctx
exec.isCTI = True
return Bc2Eqz0operands(ct=ct, offset=sign_extend(offset, from_nbits=17))
```

8.56.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("BC2NEZC execution is not implemented")
```

8.56.4. Description

BC2NEZC checks CP2 availability before evaluating the condition.

8.56.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; BC2NEZC execution is not implemented

8.57. BC2T

8.57.1. Purpose

Branch on a true CP2 condition.

8.57.2. Formats

microMIPS

bc2t.32

Assembly

BC2T {cc:u3}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

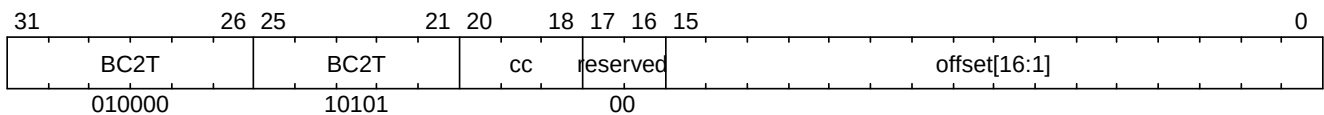


Figure 168. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Bc20operands(cc=cc, offset=sign_extend(offset, from_nbits=17))
```

8.57.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("BC2T execution is not implemented")
```

8.57.4. Description

BC2T checks CP2 availability before evaluating the condition.

8.57.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; BC2T execution is not implemented

8.58. BEQ

8.58.1. Purpose

Compare two GPRs and perform a delayed PC-relative branch when their values are equal.

8.58.2. Formats

MIPS

beq

Assembly

BEQ {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

true

Decode priority

0

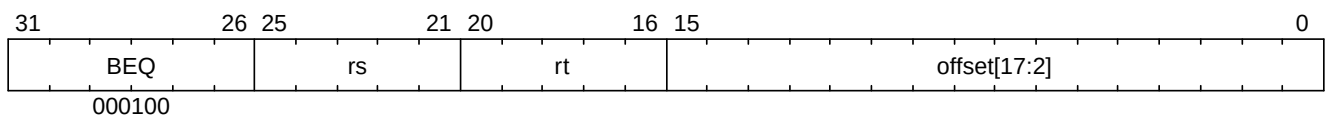


Figure 169. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
offset = sign_extend(offset, from_nbits=18)
return Beq0operands(rs=rs, rt=rt, offset=offset)
```

microMIPS

beq.32

Assembly

BEQ {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

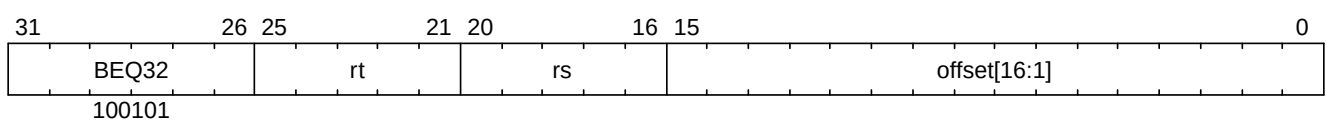


Figure 170. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
offset = sign_extend(offset, from_nbits=17)
return Beq0operands(rs=rs, rt=rt, offset=offset)
```

beqz.16

Assembly

BEQ {rs:gpr3}, {offset:s8}

Availability

MMAR < R6

Decode priority

0

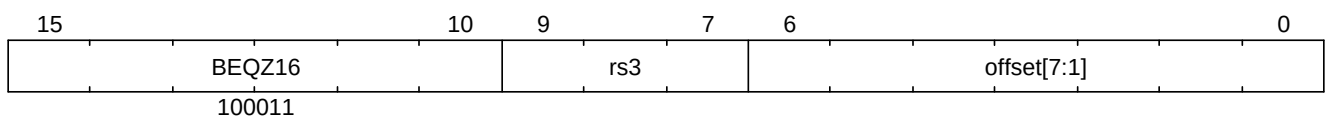


Figure 171. Encoding (16 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Beq0operands(
    rs=MICROMIPS_GPR3.decode(rs3),
    rt=0,
    offset=sign_extend(offset, from_nbits=8),
)
```

b.16

Assembly

BEQ {offset:s11}

Availability

MMAR < R6

Decode priority

0

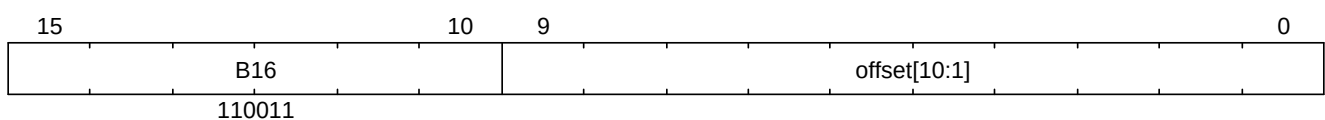


Figure 172. Encoding (16 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return BeqOperands(
    rs=0,
    rt=0,
    offset=sign_extend(offset, from_nbits=11),
)
```

8.58.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
taken = ctx.GPR[operands.rs] == ctx.GPR[operands.rt]
delayed_branch(ctx, address, taken=taken)
```

8.58.4. Description

The signed displacement is added to the address of the instruction following BEQ. It is word-aligned in MIPS mode and halfword-aligned in microMIPS mode. The delay-slot instruction executes whether or not the branch is taken.

8.58.5. Exceptions

None.

8.59. BEQC

8.59.1. Purpose

Compare two GPRs and perform a compact PC-relative branch when their values are equal.

8.59.2. Formats

MIPS

beqc

Assembly

BEQC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

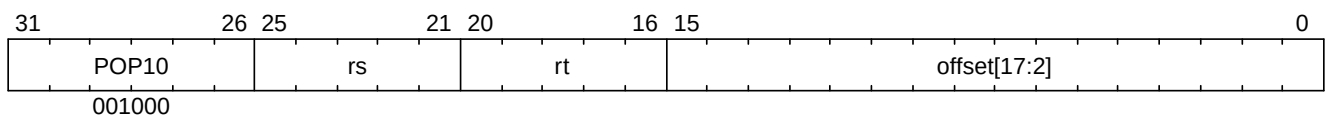


Figure 173. Encoding (32 bits)

Encoding constraints

- **rs** != 0
- **rs** < **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return BeqcOperands(rs=rs, rt=rt, offset=offset)
```

microMIPS

beqc.32

Assembly

BEQC {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

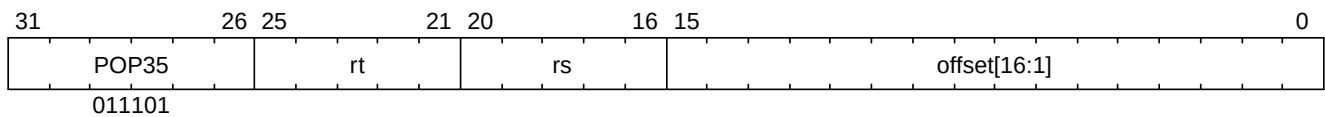


Figure 174. Encoding (32 bits)

Encoding constraints

- $rs \neq 0$
- $rs < rt$

Decode

```

exec.isCTI = True
offset = sign_extend(offset, from_nbits=17)
return BeqcOperands(rs=rs, rt=rt, offset=offset)

```

nanoMIPS

beqc.16

Assembly

BEQC {rs:gpr3}, {rt:gpr3}, {offset:u5}

Availability

!(NMS)

Decode priority

0

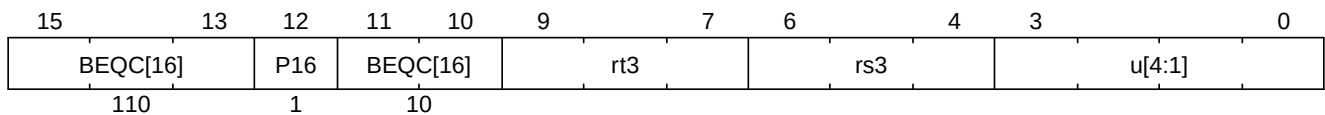


Figure 175. Encoding (16 bits)

Encoding constraints

- $rs3 < rt3$
- $u \neq 0$

Decode

```

exec.isCTI = True
rs = GPR3.decode(rs3)
rt = GPR3.decode(rt3)
offset = zero_extend(u, from_nbits=5)
return BeqcOperands(rs=rs, rt=rt, offset=offset)

```

beqc.32

Assembly

BEQC {rs:gpr}, {rt:gpr}, {offset:s15}

Availability

true

Decode priority

0

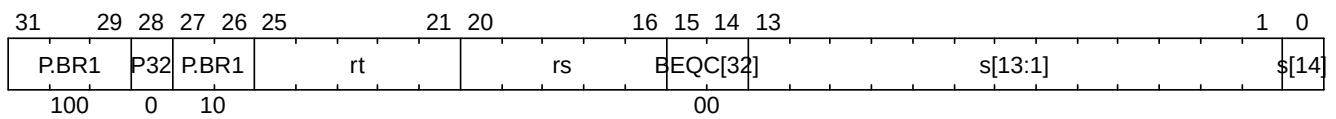


Figure 176. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(s, from_nbits=15)
return BeqcOperands(rs=rs, rt=rt, offset=offset)
```

8.59.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
taken = ctx.GPR[operands.rs] == ctx.GPR[operands.rt]
compact_branch(
    ctx,
    address,
    taken=taken,
    has_forbidden_slot=exec.hasForbiddenSlot,
)
```

8.59.4. Description

The displacement is added to the address of the instruction following BEQC. A taken branch selects the target immediately. A not-taken Release 6 MIPS form creates a forbidden slot; Release 6 microMIPS and nanoMIPS permit another control-transfer instruction to follow.

8.59.5. Exceptions

None.

8.60. BEQIC

8.60.1. Purpose

Branch compactly when a GPR is equal to an immediate.

8.60.2. Formats

nanoMIPS

beqic.32

Assembly

BEQIC {immediate:u7}, {offset:s12}

Availability

true

Decode priority

0

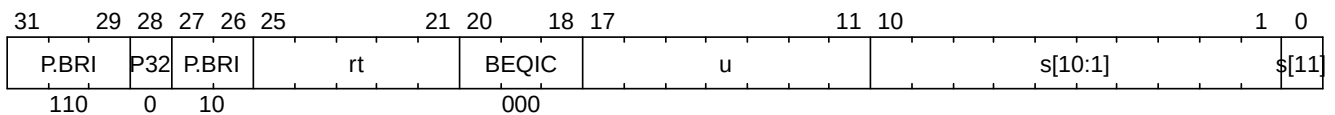


Figure 177. Encoding (32 bits)

Decode

```
exec.isCTI = True
immediate = zero_extend(u, from_nbits=7)
offset = sign_extend(s, from_nbits=12)
return BranchImmediateOperands(rt, immediate, offset)
```

8.60.3. Operation

```
lhs = ctx.GPR[operands.rt]
rhs = int(operands.immediate)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs == rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.60.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.60.5. Exceptions

None.

8.61. BEQL

8.61.1. Purpose

Branch likely when two GPR values are equal.

8.61.2. Formats

MIPS

beql

Assembly

BEQL {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR < R6

Decode priority

0

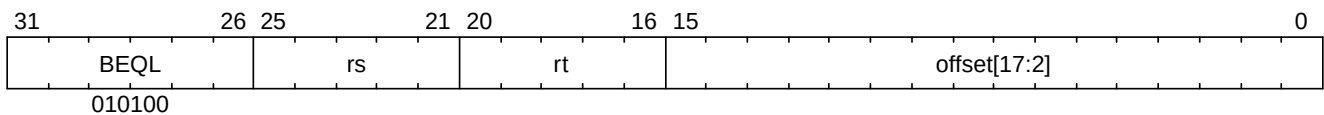


Figure 178. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=rt, offset=sign_extend(offset, from_nbits=18)
)
```

8.61.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs == rhs)
```

8.61.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.61.5. Exceptions

None.

8.62. BEQZALC

8.62.1. Purpose

Perform a compact branch-and-link when a GPR is eq zero.

8.62.2. Formats

MIPS

beqzalc

Assembly

BEQZALC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

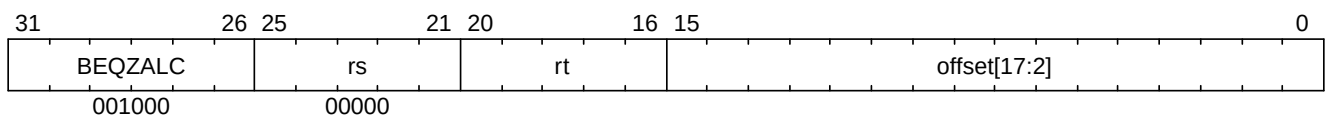


Figure 179. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rt, offset=offset)
```

microMIPS

beqzalc.r6.32

Assembly

BEQZALC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

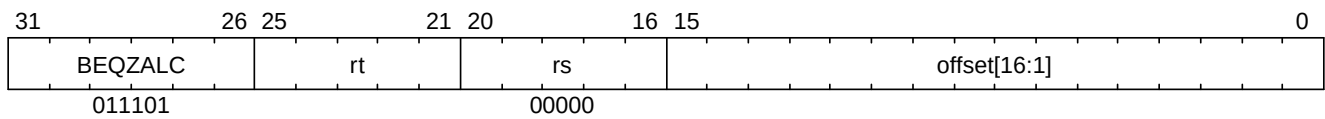


Figure 180. Encoding (32 bits)

Encoding constraints

- `rt != 0`

Decode

```

exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))

```

8.62.3. Operation

```

source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, ctx.next_pc)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source == 0, has_forbidden_slot=exec.hasForbiddenSlot
)

```

8.62.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.62.5. Exceptions

None.

8.63. BEQZC

8.63.1. Purpose

Branch compactly when a GPR is zero.

8.63.2. Formats

MIPS

beqzc

Assembly

BEQZC {rt:gpr}, {offset:s23}

Availability

AR >= R6

Decode priority

0

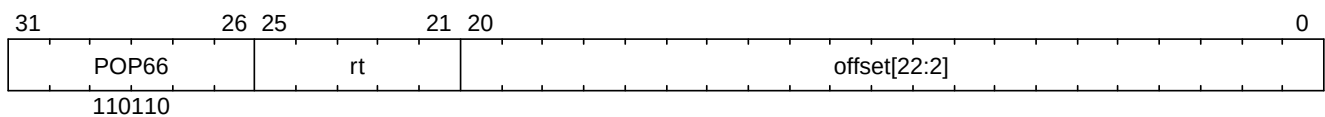


Figure 181. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return BeqzcOperands(rt=rt, offset=sign_extend(offset, from_nbits=23))
```

microMIPS

beqzc.32

Assembly

BEQZC {rt:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

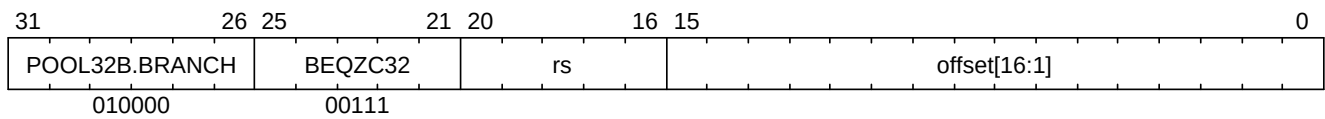


Figure 182. Encoding (32 bits)

Decode

```
exec.isCTI = True
return BeqzcOperands(rt=rs, offset=sign_extend(offset, from_nbits=17))
```

beqzc.r6.32

Assembly

BEQZC {rt:gpr}, {offset:s22}

Availability

MMAR >= R6

Decode priority

0



Figure 183. Encoding (32 bits)

Encoding constraints

- rs != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return BeqzcOperands(rt=rs, offset=sign_extend(offset, from_nbits=22))
```

beqzc.r6.16

Assembly

BEQZC {rt:gpr3}, {offset:s8}

Availability

MMAR >= R6

Decode priority

0

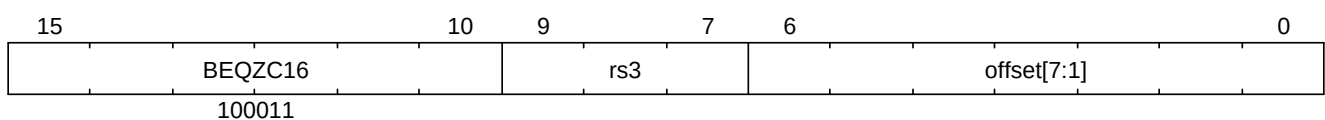


Figure 184. Encoding (16 bits)

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return BeqzcOperands(
    rt=MICROMIPS_GPR3.decode(rs3),
    offset=sign_extend(offset, from_nbits=8),
)
```

nanoMIPS

beqzc.16

Assembly

BEQZC {rt:gpr3}, {offset:s8}

Availability

true

Decode priority

0

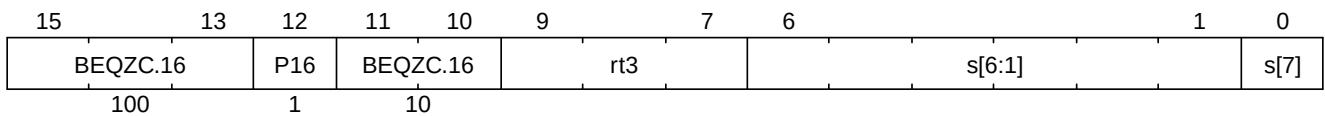


Figure 185. Encoding (16 bits)

Decode

```
exec.isCTI = True
return BeqzcOperands(
    rt=GPR3.decode(rt3),
    offset=sign_extend(s, from_nbits=8),
)
```

8.63.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx,
    address,
    taken=ctx.GPR[operands.rt] == 0,
    has_forbidden_slot=exec.hasForbiddenSlot,
)
```

8.63.4. Description

BEQZC[16] sign-extends its eight-bit aligned displacement from the following instruction and

branches when the compact source GPR is zero.

8.63.5. Exceptions

None.

8.64. BGEC

8.64.1. Purpose

Branch compactly when one signed GPR is greater than or equal another.

8.64.2. Formats

MIPS

bgec

Assembly

BGEC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

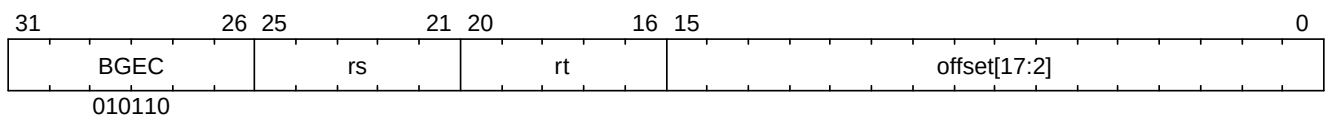


Figure 186. Encoding (32 bits)

Encoding constraints

- **rs** != 0
- **rt** != 0
- **rs** != **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt, rs, offset)
```

microMIPS

bgec.r6.32

Assembly

BGEC {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

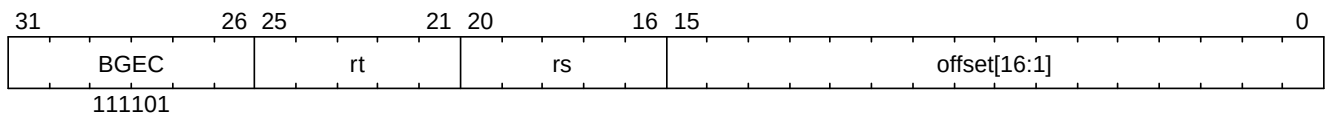


Figure 187. Encoding (32 bits)

Encoding constraints

- $rs \neq 0$
- $rt \neq 0$
- $rs \neq rt$

Decode

```
del ctx
exec.isCTI = True
exec.hasForbiddenSlot = True
return BranchCompareOperands(
    rs=rs,
    rt=rt,
    offset=sign_extend(offset, from_nbits=17),
)
```

nanoMIPS

bgec.32

Assembly

BGEc {rt:gpr}, {offset:s15}

Availability

true

Decode priority

0

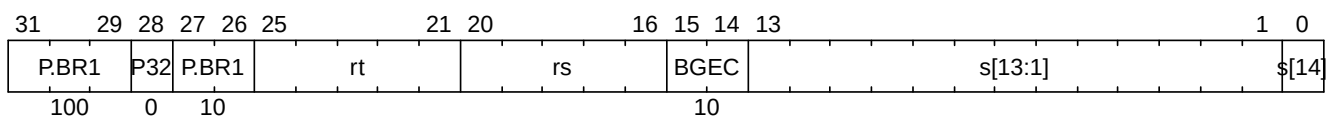


Figure 188. Encoding (32 bits)

Encoding constraints

- $rt \neq 0$

Decode

```
exec.isCTI = True
offset = sign_extend(s, from_nbits=15)
return BranchCompareOperands(rs, rt, offset)
```

8.64.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs >= rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.64.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.64.5. Exceptions

None.

8.65. BGEIC

8.65.1. Purpose

Branch compactly when a GPR is greater than or equal to an immediate.

8.65.2. Formats

nanoMIPS

bgeic.32

Assembly

BGEIC {immediate:u7}, {offset:s12}

Availability

true

Decode priority

0

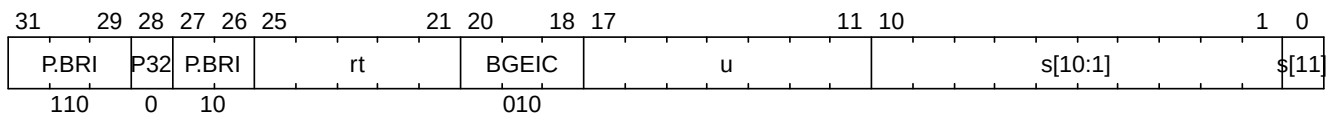


Figure 189. Encoding (32 bits)

Decode

```
exec.isCTI = True
immediate = zero_extend(u, from_nbits=7)
offset = sign_extend(s, from_nbits=12)
return BranchImmediateOperands(rt, immediate, offset)
```

8.65.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width))
rhs = int(operands.immediate)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs >= rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.65.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.65.5. Exceptions

None.

8.66. BGEIUC

8.66.1. Purpose

Branch compactly when a GPR is unsigned greater than or equal to an immediate.

8.66.2. Formats

nanoMIPS

bgeiuc.32

Assembly

BGEIUC {immediate:u7}, {offset:s12}

Availability

true

Decode priority

0

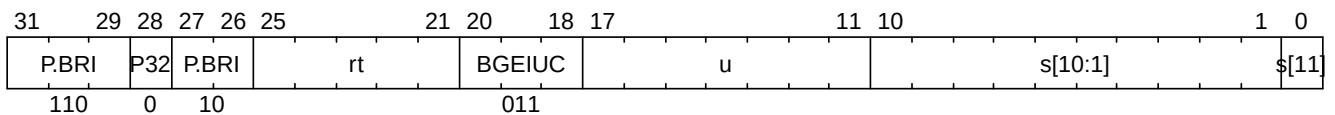


Figure 190. Encoding (32 bits)

Decode

```
exec.isCTI = True
immediate = zero_extend(u, from_nbits=7)
offset = sign_extend(s, from_nbits=12)
return BranchImmediateOperands(rt, immediate, offset)
```

8.66.3. Operation

```
lhs = ctx.GPR[operands.rt]
rhs = int(operands.immediate)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs >= rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.66.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.66.5. Exceptions

None.

8.67. BGEUC

8.67.1. Purpose

Branch compactly when one unsigned GPR is greater than or equal another.

8.67.2. Formats

MIPS

bgeuc

Assembly

BGEUC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

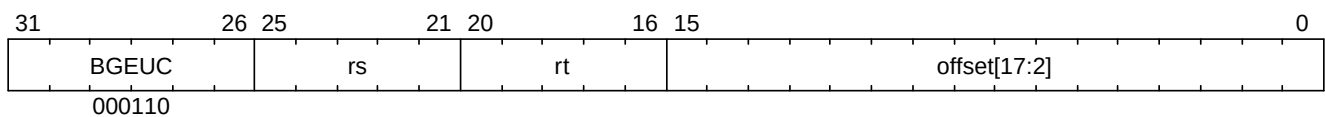


Figure 191. Encoding (32 bits)

Encoding constraints

- **rs** != 0
- **rt** != 0
- **rs** != **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt, rs, offset)
```

microMIPS

bgeuc.r6.32

Assembly

BGEUC {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

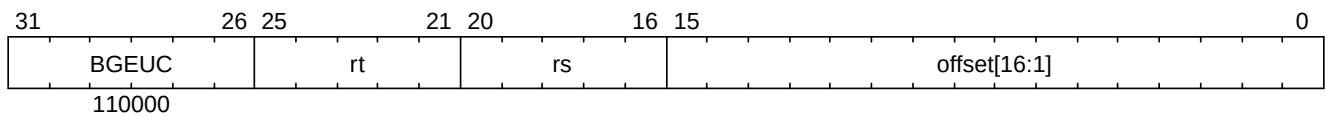


Figure 192. Encoding (32 bits)

Encoding constraints

- $rs \neq 0$
- $rt \neq 0$
- $rs \neq rt$

Decode

```
del ctx
exec.isCTI = True
exec.hasForbiddenSlot = True
return BranchCompareOperands(
    rs=rs,
    rt=rt,
    offset=sign_extend(offset, from_nbits=17),
)
```

nanoMIPS

bgeuc.32

Assembly

BGEUC {rt:gpr}, {offset:s15}

Availability

true

Decode priority

0

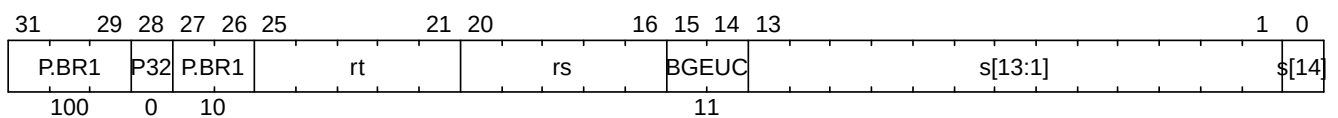


Figure 193. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(s, from_nbits=15)
return BranchCompareOperands(rs, rt, offset)
```

8.67.3. Operation

```
lhs = int(ctx.GPR[operands.rs])
rhs = int(ctx.GPR[operands.rt])
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs >= rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.67.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.67.5. Exceptions

None.

8.68. BGEZ

8.68.1. Purpose

Branch after one delay slot when a signed GPR value is non-negative.

8.68.2. Formats

MIPS

bgez

Assembly

BGEZ {rs:gpr}, {offset:s18}

Availability

true

Decode priority

0

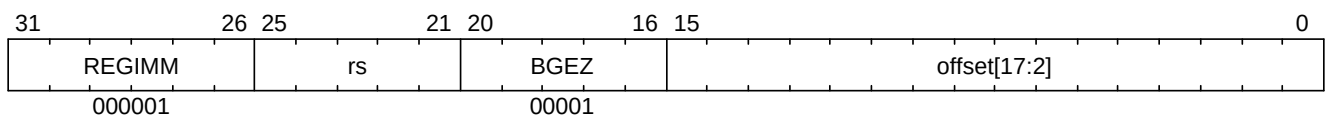


Figure 194. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

microMIPS

bgez.32

Assembly

BGEZ {rs:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

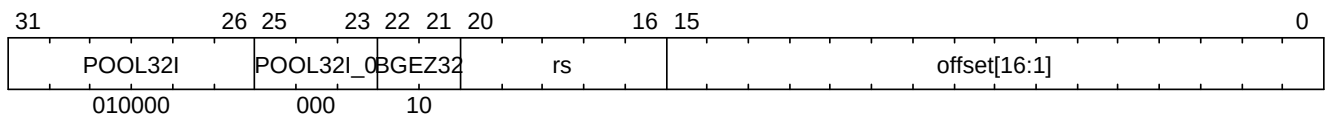


Figure 195. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs,
    rt=0,
    offset=sign_extend(offset, from_nbits=17),
)
```

8.68.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs >= 0)
```

8.68.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.68.5. Exceptions

None.

8.69. BGEZAL

8.69.1. Purpose

Conditionally branch and link when a signed GPR value is non-negative.

8.69.2. Formats

MIPS

`bgezal`

Assembly

`BGEZAL {rs:gpr}, {offset:s18}`

Availability

AR < R6

Decode priority

0

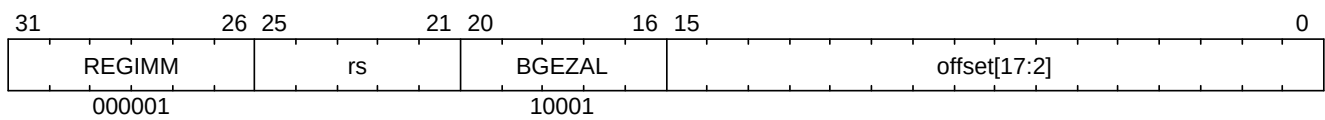


Figure 196. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

microMIPS

`bgezal.32`

Assembly

`BGEZAL {rs:gpr}, {offset:s17}`

Availability

MMAR < R6

Decode priority

0

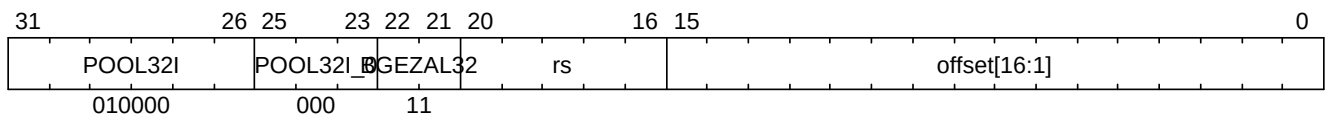


Figure 197. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs,
    rt=0,
    offset=sign_extend(offset, from_nbits=17),
)
```

8.69.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, effective_address(ctx, ctx.pc, 8))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs >= 0)
```

8.69.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.69.5. Exceptions

None.

8.70. BGEZALC

8.70.1. Purpose

Perform a compact branch-and-link when a GPR is ge zero.

8.70.2. Formats

MIPS

bgezalc

Assembly

BGEZALC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

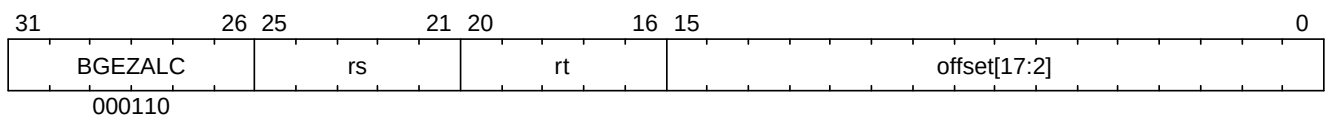


Figure 198. Encoding (32 bits)

Encoding constraints

- **rs == rt**
- **rt != 0**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rs, offset=offset)
```

microMIPS

bgezalc.r6.32

Assembly

BGEZALC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

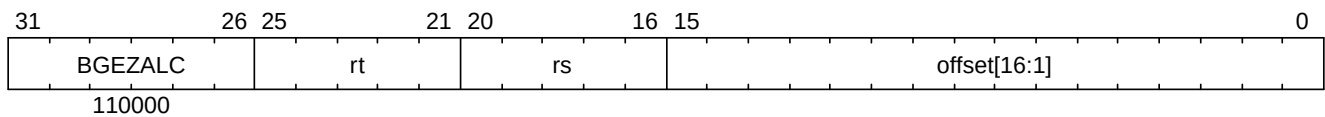


Figure 199. Encoding (32 bits)

Encoding constraints

- `rt != 0`
- `rs == rt`

Decode

```
del ctx, rs
exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))
```

8.70.3. Operation

```
source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, ctx.next_pc)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source >= 0, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.70.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.70.5. Exceptions

None.

8.71. BGEZALL

8.71.1. Purpose

Branch likely and link when a signed GPR is non-negative.

8.71.2. Formats

MIPS

bgezall

Assembly

BGEZALL {rs:gpr}, {offset:s18}

Availability

AR < R6

Decode priority

0

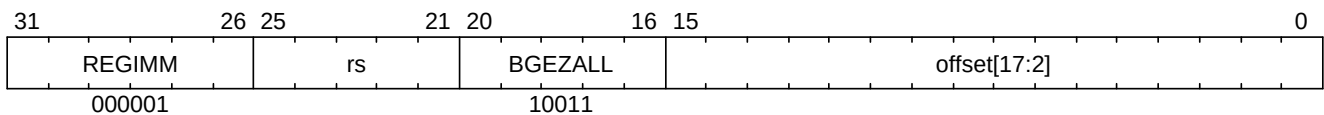


Figure 200. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

8.71.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, effective_address(ctx, ctx.pc, 8))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs >= 0)
```

8.71.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.71.5. Exceptions

None.

8.72. BGEZALS

8.72.1. Purpose

Conditionally branch and link with a 16-bit delay-slot instruction.

8.72.2. Formats

microMIPS

bgezals.32

Assembly

BGEZALS {rs:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

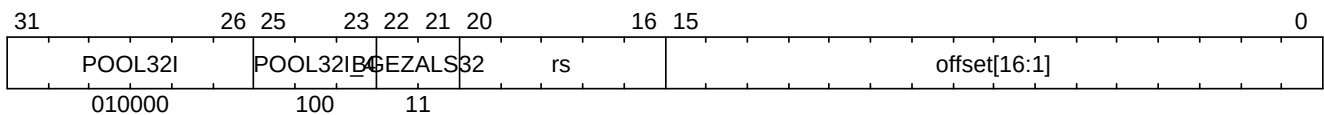


Figure 201. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranch0Operands(rs=rs, rt=0, offset=sign_extend(offset, from_nbits=17))
```

8.72.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, effective_address(ctx, ctx.next_pc, 2))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs >= 0)
```

8.72.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.72.5. Exceptions

None.

8.73. BGEZC

8.73.1. Purpose

Perform a compact PC-relative branch when a signed source GPR is greater than or equal to zero.

8.73.2. Formats

MIPS

bgezc

Assembly

BGEZC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

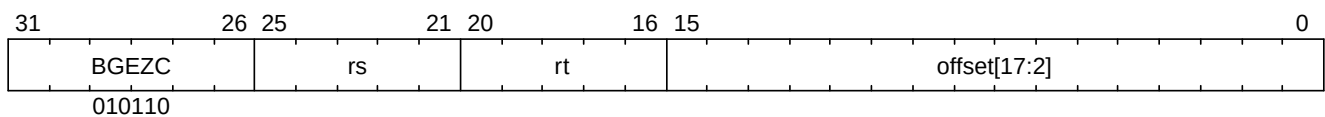


Figure 202. Encoding (32 bits)

Encoding constraints

- **rs == rt**
- **rt != 0**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=rt, offset=offset, rs=0)
```

microMIPS

bgezc.r6.32

Assembly

BGEZC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

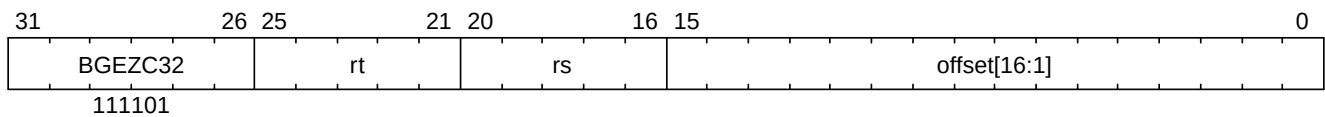


Figure 203. Encoding (32 bits)

Encoding constraints

- `rs == rt`
- `rt != 0`

Decode

```
exec.isCTI = True
# The microMIPS compact branch retains an executable slot. In
# particular, the AVP R6 microMIPS slot test deliberately places BC in
# the following parcel; marking it forbidden incorrectly routes that
# valid instruction through the RI path.
return BgezCOperands(rt=rt, offset=sign_extend(offset, from_nbits=17), rs=0)
```

nanoMIPS

bgezc.32

Assembly

`BGEZC {rt:gpr}, {offset:s15}`

Availability

true

Decode priority

0

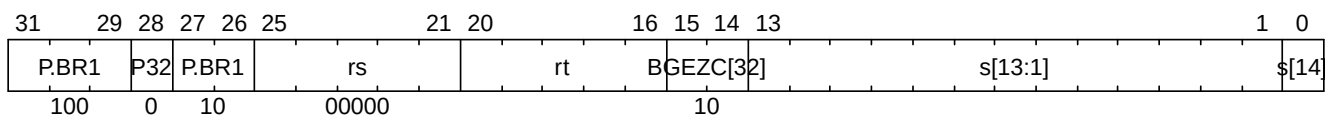


Figure 204. Encoding (32 bits)

Decode

```
exec.isCTI = True
return BgezCOperands(rt=rt, offset=sign_extend(s, from_nbits=15), rs=0)
```

8.73.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
source = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
zero = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
compact_branch(
    ctx,
```

```
address,  
taken=int(source) >= int(zero),  
has_forbidden_slot=exec.hasForbiddenSlot,  
)
```

8.73.4. Description

BGEZC adds the signed, word-aligned displacement to the address of the following instruction. It has no delay or forbidden slot. The encoding is the BGEC form with its first register field fixed to zero, leaving `rt` as the source register.

8.73.5. Exceptions

None.

8.74. BGEZL

8.74.1. Purpose

Branch likely when a signed GPR is non-negative.

8.74.2. Formats

MIPS

bgezl

Assembly

BGEZL {rs:gpr}, {offset:s18}

Availability

AR < R6

Decode priority

0

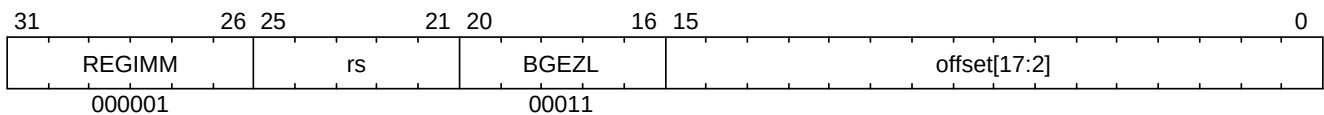


Figure 205. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

8.74.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs >= 0)
```

8.74.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.74.5. Exceptions

None.

8.75. BGTZ

8.75.1. Purpose

Branch after one delay slot when a signed GPR value is greater than zero.

8.75.2. Formats

MIPS

bgtz

Assembly

BGTZ {rs:gpr}, {offset:s18}

Availability

true

Decode priority

0

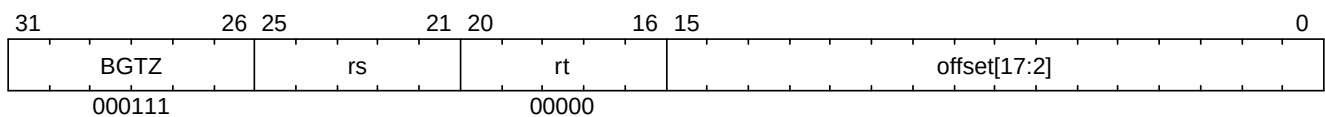


Figure 206. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

microMIPS

bgtz.32

Assembly

BGTZ {rs:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

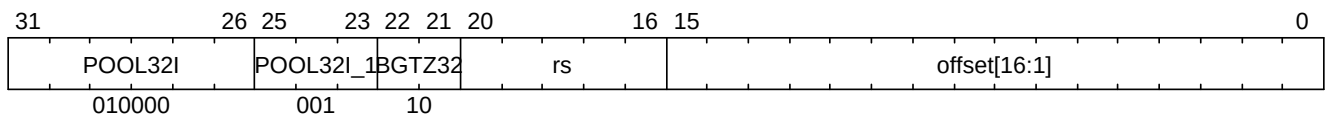


Figure 207. Encoding (32 bits)

Decode

```

exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs,
    rt=0,
    offset=sign_extend(offset, from_nbits=17),
)

```

8.75.3. Operation

```

lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs > 0)

```

8.75.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.75.5. Exceptions

None.

8.76. BGTZALC

8.76.1. Purpose

Perform a compact branch-and-link when a GPR is gt zero.

8.76.2. Formats

MIPS

bgtzalc

Assembly

BGTZALC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

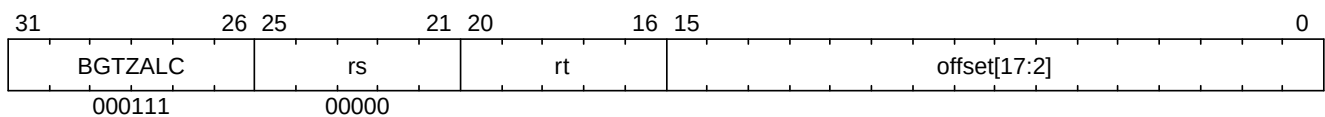


Figure 208. Encoding (32 bits)

Encoding constraints

- **rt != 0**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rt, offset=offset)
```

microMIPS

bgtzalc.r6.32

Assembly

BGTZALC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

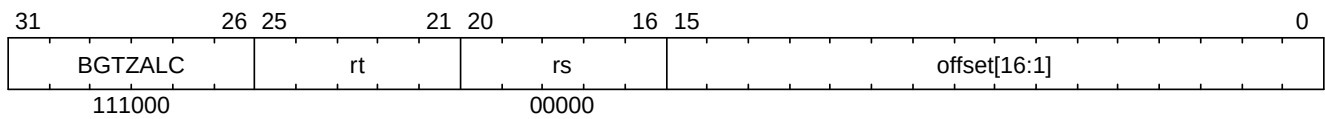


Figure 209. Encoding (32 bits)

Encoding constraints

- `rt != 0`

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))
```

8.76.3. Operation

```
source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, ctx.next_pc)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source > 0, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.76.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.76.5. Exceptions

None.

8.77. BGTZC

8.77.1. Purpose

Perform a compact branch when a GPR is gt zero.

8.77.2. Formats

MIPS

bgtzc

Assembly

BGTZC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

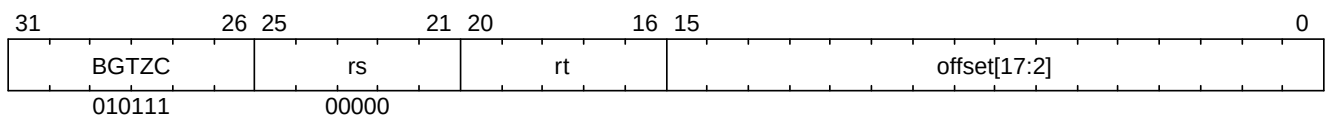


Figure 210. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rt, offset=offset)
```

microMIPS

bgtzc.r6.32

Assembly

BGTZC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

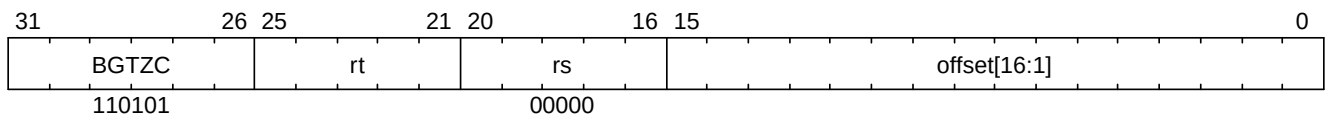


Figure 211. Encoding (32 bits)

Encoding constraints

- `rt != 0`

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))
```

8.77.3. Operation

```
source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source > 0, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.77.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.77.5. Exceptions

None.

8.78. BGTZL

8.78.1. Purpose

Branch likely when a signed GPR is greater than zero.

8.78.2. Formats

MIPS

`bgtzl`

Assembly

`BGTZL {rs:gpr}, {offset:s18}`

Availability

AR < R6

Decode priority

0

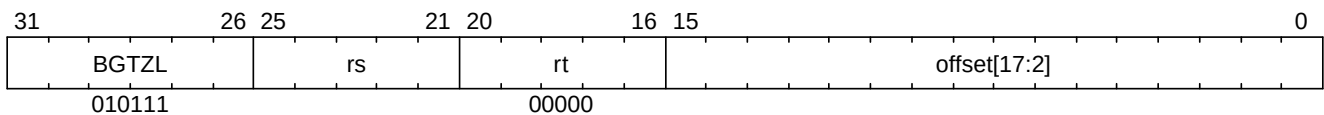


Figure 212. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

8.78.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs > 0)
```

8.78.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.78.5. Exceptions

None.

8.79. BITREV

8.79.1. Purpose

Reverse the bits in the low halfword.

8.79.2. Formats

MIPS

bitrev

Assembly

BITREV {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

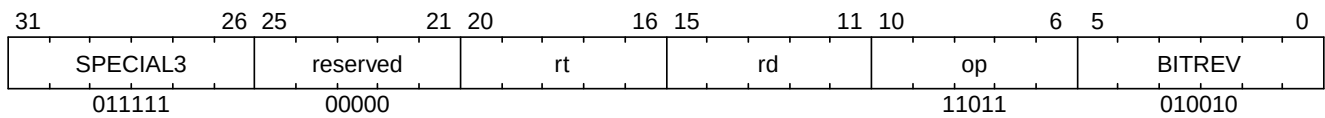


Figure 213. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

bitrev.32

Assembly

BITREV {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

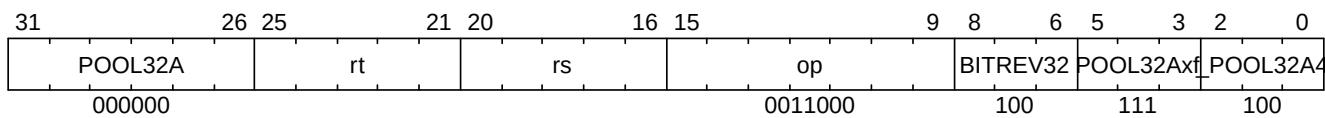


Figure 214. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

bitrev.32

Assembly

BITREV {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

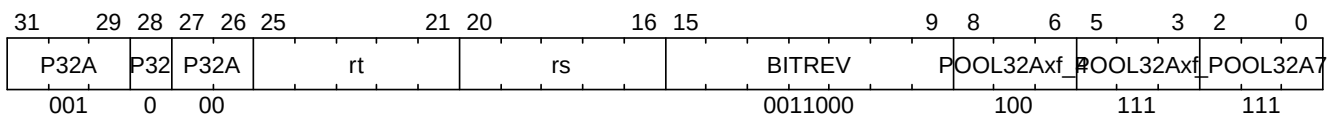


Figure 215. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.79.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = zero_extend(ctx.GPR[operands.rt], from_nbits=16)
result = 0
for bit in range(16):
    result |= ((source >> bit) & 1) << (15 - bit)
ctx.GPR[operands.rd] = zero_extend(result, from_nbits=16)
```

8.79.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.79.5. Exceptions

None.

8.80. BITSWAP

8.80.1. Purpose

Reverse the bit order within every byte of a word.

8.80.2. Formats

MIPS

bitswap

Assembly

BITSWAP {rd:gpr}, {rt:gpr}

Availability

AR >= R6

Decode priority

0

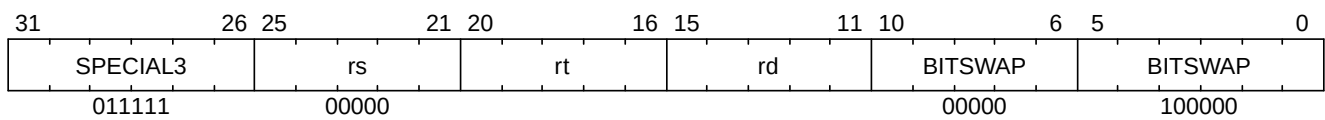


Figure 216. Encoding (32 bits)

Decode

```
return BitswapOperands(rd=rd, rt=rt)
```

microMIPS

bitswap.32

Assembly

BITSWAP {rd:gpr}, {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

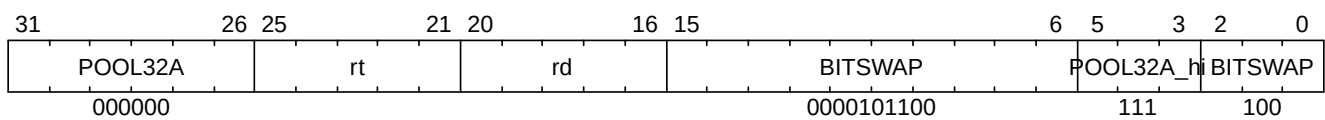


Figure 217. Encoding (32 bits)

Decode

```
return BitswapOperands(rd=rd, rt=rt)
```

8.80.3. Operation

```
source = u32(ctx.GPR[operands.rt])  
result = bitswap_word(source)  
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.80.4. Description

BITSWAP independently reverses bits 7..0 in each of the four source bytes.

8.80.5. Exceptions

None.

8.81. BLEZ

8.81.1. Purpose

Branch after one delay slot when a signed GPR value is at most zero.

8.81.2. Formats

MIPS

`blez`

Assembly

`BLEZ {rs:gpr}, {offset:s18}`

Availability

true

Decode priority

0

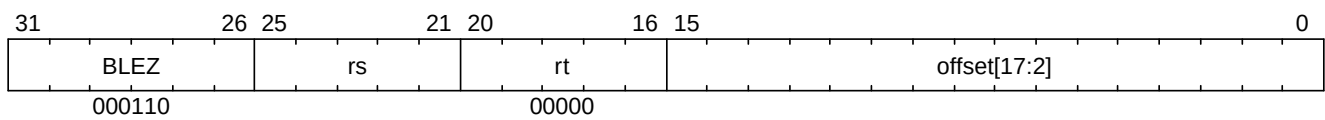


Figure 218. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

microMIPS

`blez.32`

Assembly

`BLEZ {rs:gpr}, {offset:s17}`

Availability

MMAR < R6

Decode priority

0

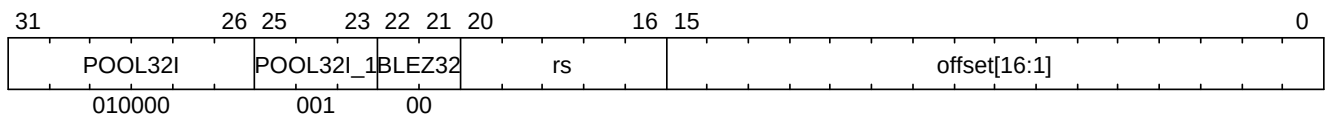


Figure 219. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs,
    rt=0,
    offset=sign_extend(offset, from_nbits=17),
)
```

8.81.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs <= 0)
```

8.81.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.81.5. Exceptions

None.

8.82. BLEZALC

8.82.1. Purpose

Perform a compact branch-and-link when a GPR is le zero.

8.82.2. Formats

MIPS

blezalc

Assembly

BLEZALC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

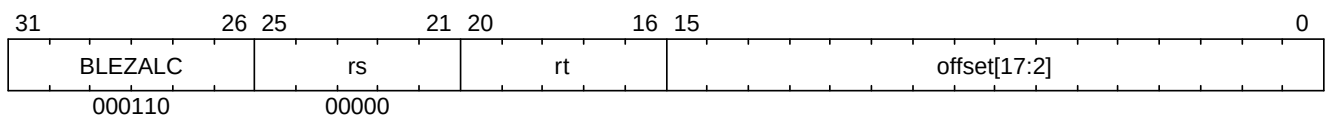


Figure 220. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rt, offset=offset)
```

microMIPS

blezalc.r6.32

Assembly

BLEZALC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

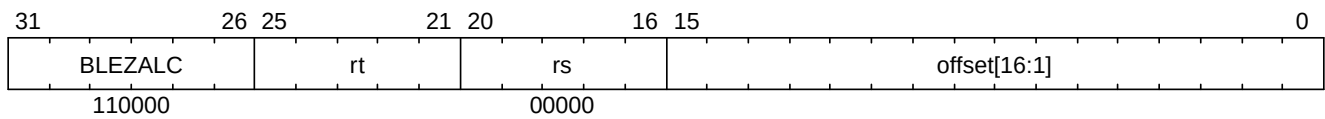


Figure 221. Encoding (32 bits)

Encoding constraints

- `rt != 0`

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))
```

8.82.3. Operation

```
source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, ctx.next_pc)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source <= 0, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.82.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.82.5. Exceptions

None.

8.83. BLEZC

8.83.1. Purpose

Perform a compact branch when a GPR is le zero.

8.83.2. Formats

MIPS

blezc

Assembly

BLEZC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

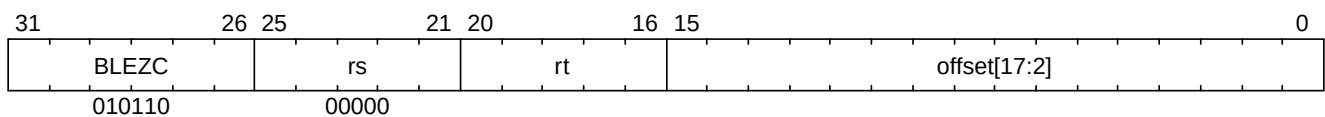


Figure 222. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rt, offset=offset)
```

microMIPS

blezc.r6.32

Assembly

BLEZC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

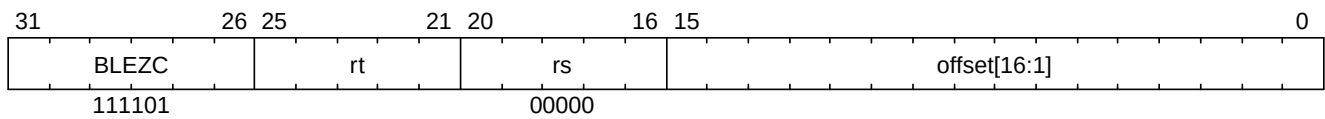


Figure 223. Encoding (32 bits)

Encoding constraints

- `rt != 0`

Decode

```

exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))

```

8.83.3. Operation

```

source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source <= 0, has_forbidden_slot=exec.hasForbiddenSlot
)

```

8.83.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.83.5. Exceptions

None.

8.84. BLEZL

8.84.1. Purpose

Branch likely when a signed GPR is at most zero.

8.84.2. Formats

MIPS

blezl

Assembly

BLEZL {rs:gpr}, {offset:s18}

Availability

AR < R6

Decode priority

0

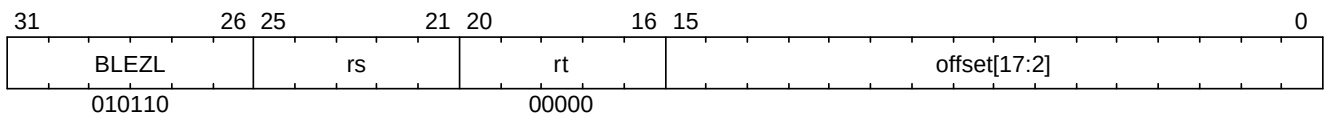


Figure 224. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

8.84.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs <= 0)
```

8.84.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.84.5. Exceptions

None.

8.85. BLTC

8.85.1. Purpose

Perform a compact PC-relative branch when signed rs is less than signed rt.

8.85.2. Formats

MIPS

bltc

Assembly

BLTC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

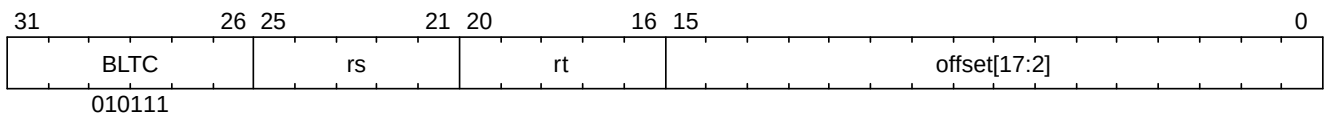


Figure 225. Encoding (32 bits)

Encoding constraints

- **rs** != 0
- **rt** != 0
- **rs** != **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rs=rs, rt=rt, offset=offset)
```

microMIPS

bltc.r6.32

Assembly

BLTC {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

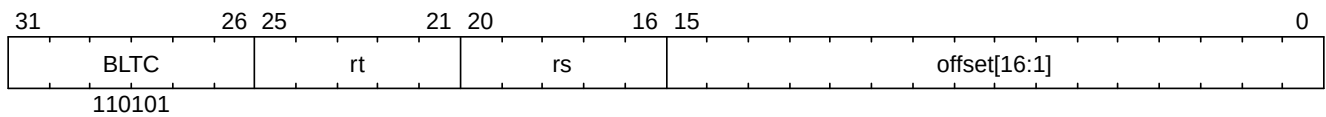


Figure 226. Encoding (32 bits)

Encoding constraints

- $rs \neq 0$
- $rt \neq 0$
- $rs \neq rt$

Decode

```
del ctx
exec.isCTI = True
exec.hasForbiddenSlot = True
return BltcOperands(
    rs=rs,
    rt=rt,
    offset=sign_extend(offset, from_nbits=17),
)
```

nanoMIPS

bltc.32

Assembly

BLTC {rs:gpr}, {rt:gpr}, {offset:s15}

Availability

true

Decode priority

0

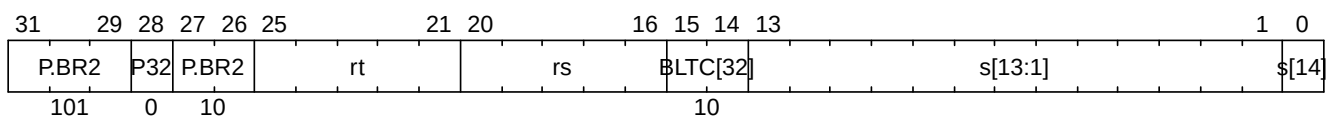


Figure 227. Encoding (32 bits)

Decode

```
exec.isCTI = True
return BltcOperands(
    rs=rs,
    rt=rt,
    offset=sign_extend(s, from_nbits=15),
)
```

8.85.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
compact_branch(
    ctx,
    address,
    taken=int(lhs) < int(rhs),
    has_forbidden_slot=exec.hasForbiddenSlot,
)
```

8.85.4. Description

BLTC compares two full-width GPRs as signed values and adds its signed, word-aligned displacement to the following-instruction address when rs is less than rt. The BLTZC assembler alias fixes rt to zero.

8.85.5. Exceptions

None.

8.86. BLTIC

8.86.1. Purpose

Branch compactly when a GPR is less than an immediate.

8.86.2. Formats

nanoMIPS

bltic.32

Assembly

BLTIC {immediate:u7}, {offset:s12}

Availability

true

Decode priority

0

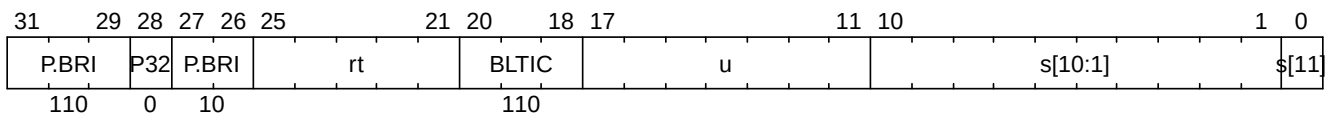


Figure 228. Encoding (32 bits)

Decode

```
exec.isCTI = True
immediate = zero_extend(u, from_nbits=7)
offset = sign_extend(s, from_nbits=12)
return BranchImmediateOperands(rt, immediate, offset)
```

8.86.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width))
rhs = int(operands.immediate)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs < rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.86.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.86.5. Exceptions

None.

8.87. BLTIUC

8.87.1. Purpose

Branch compactly when a GPR is unsigned less than an immediate.

8.87.2. Formats

nanoMIPS

bltiuc.32

Assembly

BLTIUC {immediate:u7}, {offset:s12}

Availability

true

Decode priority

0



Figure 229. Encoding (32 bits)

Decode

```
exec.isCTI = True
immediate = zero_extend(u, from_nbits=7)
offset = sign_extend(s, from_nbits=12)
return BranchImmediateOperands(rt, immediate, offset)
```

8.87.3. Operation

```
lhs = ctx.GPR[operands.rt]
rhs = int(operands.immediate)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs < rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.87.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.87.5. Exceptions

None.

8.88. BLTUC

8.88.1. Purpose

Branch compactly when one unsigned GPR is less than another.

8.88.2. Formats

MIPS

bltuc

Assembly

BLTUC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

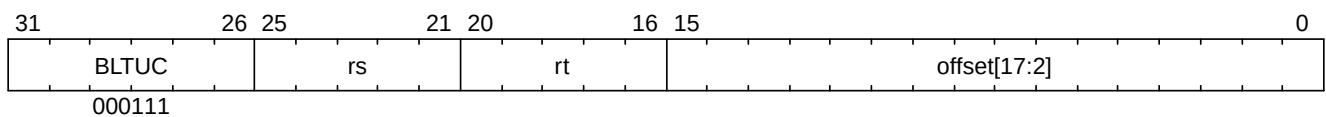


Figure 230. Encoding (32 bits)

Encoding constraints

- **rs** != 0
- **rt** != 0
- **rs** != **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt, rs, offset)
```

microMIPS

bltuc.r6.32

Assembly

BLTUC {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

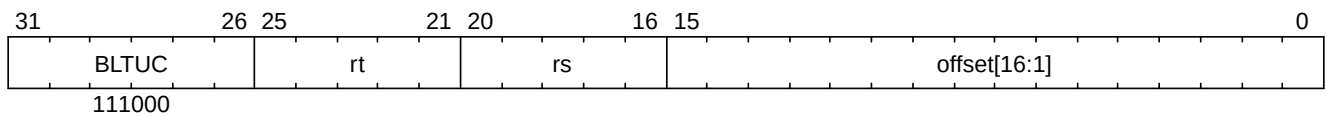


Figure 231. Encoding (32 bits)

Encoding constraints

- $rs \neq 0$
- $rt \neq 0$
- $rs \neq rt$

Decode

```
del ctx
exec.isCTI = True
exec.hasForbiddenSlot = True
return BranchCompareOperands(
    rs=rs,
    rt=rt,
    offset=sign_extend(offset, from_nbits=17),
)
```

nanoMIPS

bltuc.32

Assembly

BLTUC {rt:gpr}, {offset:s15}

Availability

true

Decode priority

0

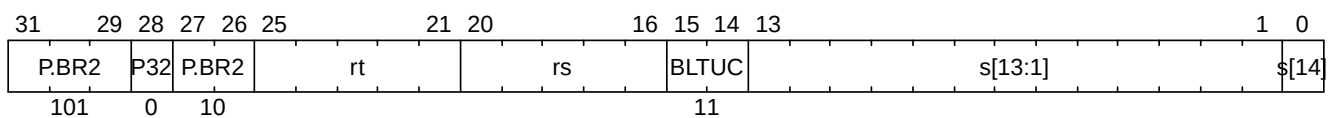


Figure 232. Encoding (32 bits)

Decode

```
exec.isCTI = True
offset = sign_extend(s, from_nbits=15)
return BranchCompareOperands(rs, rt, offset)
```

8.88.3. Operation

```
lhs = int(ctx.GPR[operands.rs])
rhs = int(ctx.GPR[operands.rt])
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs < rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.88.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.88.5. Exceptions

None.

8.89. BLTZ

8.89.1. Purpose

Branch after one delay slot when a signed GPR value is negative.

8.89.2. Formats

MIPS

bltz

Assembly

BLTZ {rs:gpr}, {offset:s18}

Availability

true

Decode priority

0

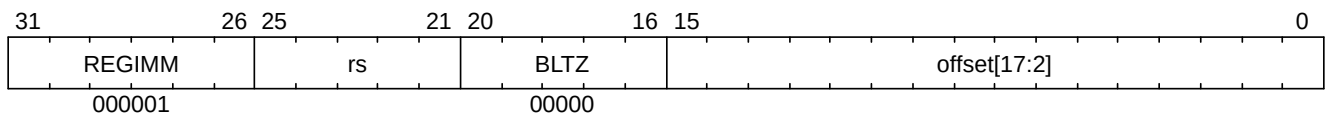


Figure 233. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

microMIPS

bltz.32

Assembly

BLTZ {rs:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

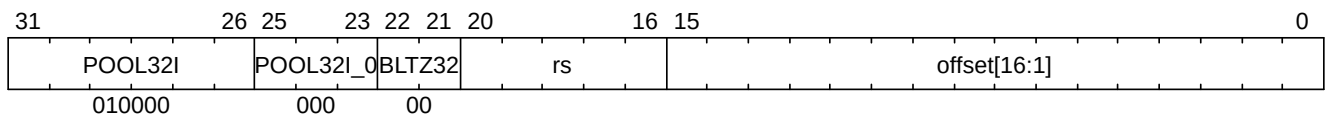


Figure 234. Encoding (32 bits)

Decode

```

exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs,
    rt=0,
    offset=sign_extend(offset, from_nbits=17),
)

```

8.89.3. Operation

```

lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs < 0)

```

8.89.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.89.5. Exceptions

None.

8.90. BLTZAL

8.90.1. Purpose

Conditionally branch and link when a signed GPR value is negative.

8.90.2. Formats

MIPS

`bltzal`

Assembly

`BLTZAL {rs:gpr}, {offset:s18}`

Availability

AR < R6

Decode priority

0

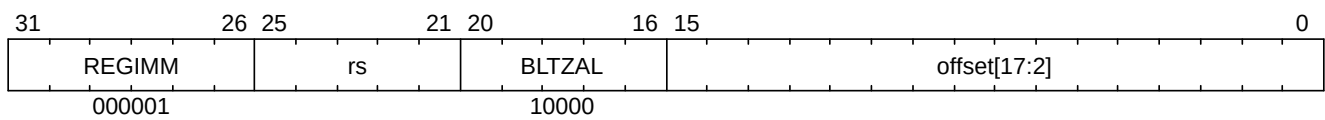


Figure 235. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

microMIPS

`bltzal.32`

Assembly

`BLTZAL {rs:gpr}, {offset:s17}`

Availability

MMAR < R6

Decode priority

0

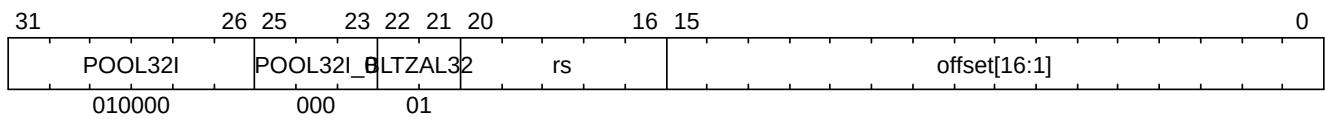


Figure 236. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs,
    rt=0,
    offset=sign_extend(offset, from_nbits=17),
)
```

8.90.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, effective_address(ctx, ctx.pc, 8))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs < 0)
```

8.90.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.90.5. Exceptions

None.

8.91. BLTZALC

8.91.1. Purpose

Perform a compact branch-and-link when a GPR is lt zero.

8.91.2. Formats

MIPS

`bltzalc`

Assembly

`BLTZALC {rt:gpr}, {offset:s18}`

Availability

AR >= R6

Decode priority

0

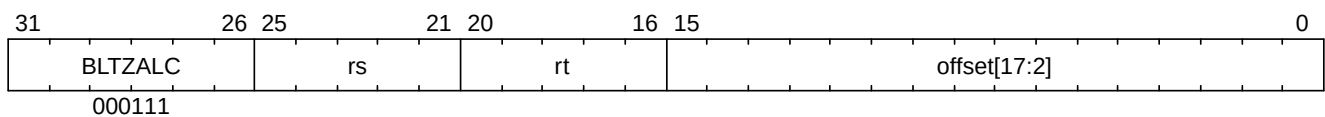


Figure 237. Encoding (32 bits)

Encoding constraints

- `rs == rt`
- `rt != 0`

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rs, offset=offset)
```

microMIPS

`bltzalc.r6.32`

Assembly

`BLTZALC {rt:gpr}, {offset:s17}`

Availability

MMAR >= R6

Decode priority

0

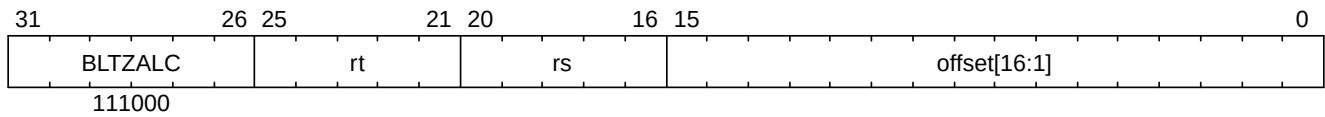


Figure 238. Encoding (32 bits)

Encoding constraints

- `rt != 0`
- `rs == rt`

Decode

```
del ctx, rs
exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))
```

8.91.3. Operation

```
source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, ctx.next_pc)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source < 0, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.91.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.91.5. Exceptions

None.

8.92. BLTZALL

8.92.1. Purpose

Branch likely and link when a signed GPR is negative.

8.92.2. Formats

MIPS

bltzall

Assembly

BLTZALL {rs:gpr}, {offset:s18}

Availability

AR < R6

Decode priority

0

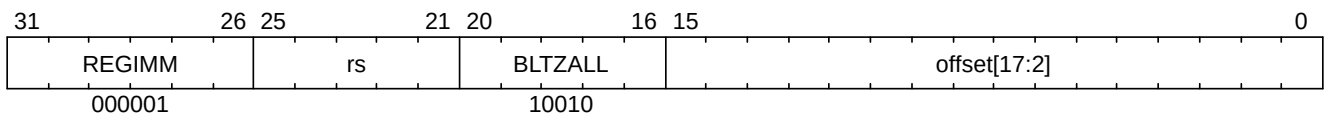


Figure 239. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

8.92.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, effective_address(ctx, ctx.pc, 8))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs < 0)
```

8.92.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.92.5. Exceptions

None.

8.93. BLTZALS

8.93.1. Purpose

Conditionally branch and link with a 16-bit delay-slot instruction.

8.93.2. Formats

microMIPS

bltzals.32

Assembly

BLTZALS {rs:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

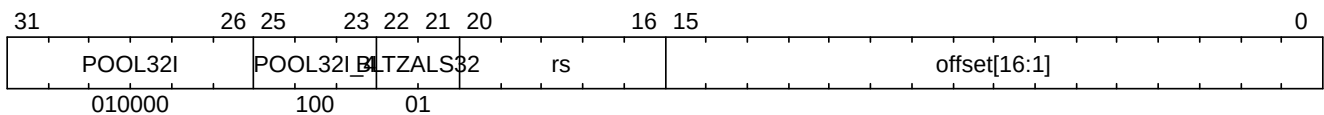


Figure 240. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(rs=rs, rt=0, offset=sign_extend(offset, from_nbits=17))
```

8.93.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, effective_address(ctx, ctx.next_pc, 2))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs < 0)
```

8.93.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.93.5. Exceptions

None.

8.94. BLTZC

8.94.1. Purpose

Perform a compact branch when a GPR is lt zero.

8.94.2. Formats

MIPS

bltzc

Assembly

BLTZC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

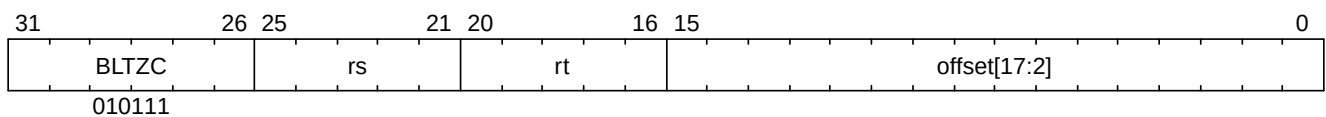


Figure 241. Encoding (32 bits)

Encoding constraints

- **rs == rt**
- **rt != 0**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rs, offset=offset)
```

microMIPS

bltzc.r6.32

Assembly

BLTZC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

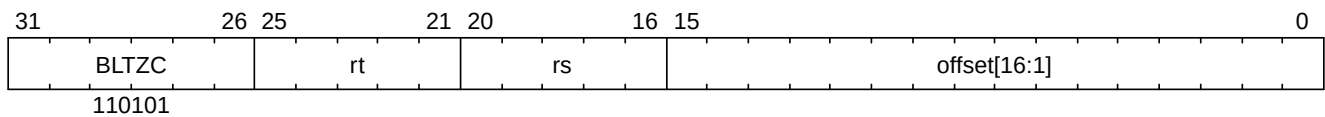


Figure 242. Encoding (32 bits)

Encoding constraints

- `rt != 0`
- `rs == rt`

Decode

```
del ctx, rs
exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))
```

8.94.3. Operation

```
source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source < 0, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.94.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.94.5. Exceptions

None.

8.95. BLTZL

8.95.1. Purpose

Branch likely when a signed GPR is negative.

8.95.2. Formats

MIPS

bltzl

Assembly

BLTZL {rs:gpr}, {offset:s18}

Availability

AR < R6

Decode priority

0

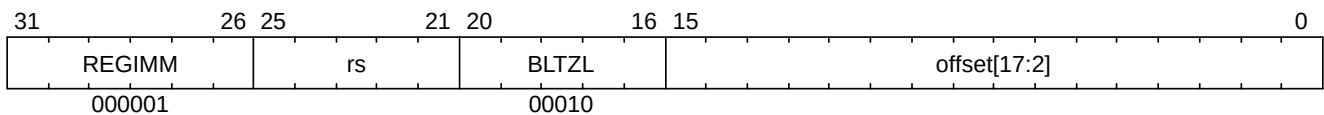


Figure 243. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=0, offset=sign_extend(offset, from_nbits=18)
)
```

8.95.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs < 0)
```

8.95.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.95.5. Exceptions

None.

8.96. BNE

8.96.1. Purpose

Branch after one delay slot when two GPR values are not equal.

8.96.2. Formats

MIPS

bne

Assembly

BNE {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

true

Decode priority

0

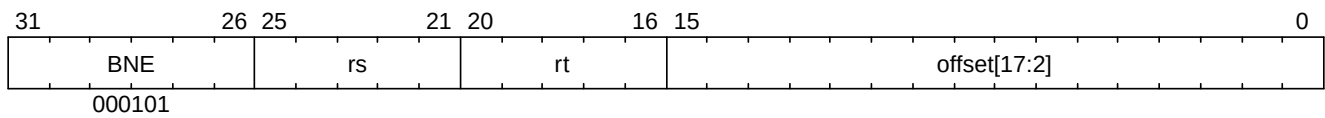


Figure 244. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranch0Operands(rs=rs, rt=rt, offset=sign_extend(offset, from_nbits=18))
```

microMIPS

bne.32

Assembly

BNE {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0

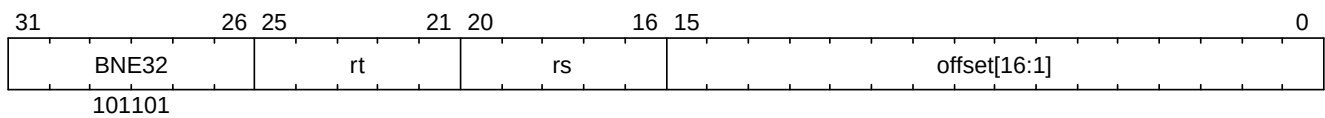


Figure 245. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs,
    rt=rt,
    offset=sign_extend(offset, from_nbits=17),
)
```

bnez.16

Assembly

BNE {rs:gpr3}, {offset:s8}

Availability

MMAR < R6

Decode priority

0

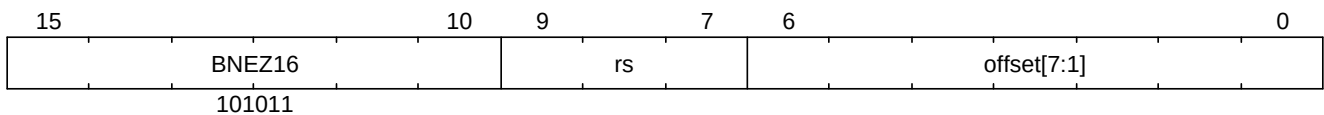


Figure 246. Encoding (16 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=MICROMIPS_GPR3.decode(rs),
    rt=0,
    offset=sign_extend(offset, from_nbits=8),
)
```

8.96.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch(ctx, target, taken=lhs != rhs)
```

8.96.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction is always executed as a delay slot.

8.96.5. Exceptions

None.

8.97. BNEC

8.97.1. Purpose

Compare two GPRs and perform a compact PC-relative branch when their values are different.

8.97.2. Formats

MIPS

bne

Assembly

BNEC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

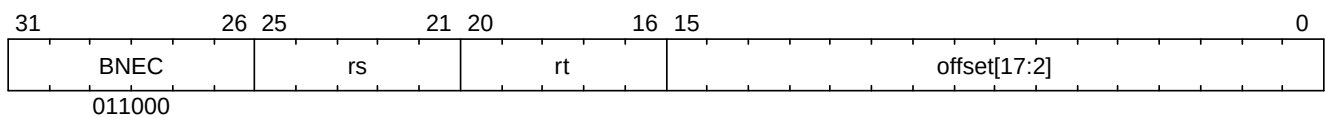


Figure 247. Encoding (32 bits)

Encoding constraints

- **rs** != 0
- **rs** < **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rs=rs, rt=rt, offset=offset)
```

microMIPS

bne.r6.32

Assembly

BNEC {rs:gpr}, {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

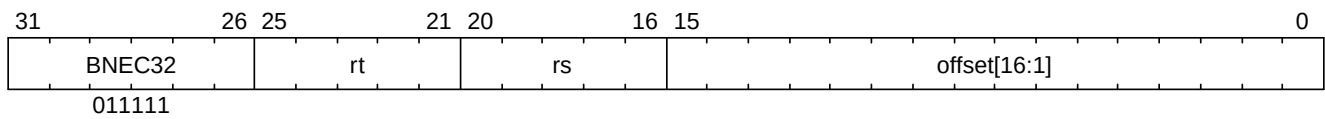


Figure 248. Encoding (32 bits)

Encoding constraints

- $rs \neq 0$
- $rs < rt$

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return BnecOperands(
    rs=rs,
    rt=rt,
    offset=sign_extend(offset, from_nbits=17),
)
```

nanoMIPS

bnecl16

Assembly

BNEC {rs:gpr3}, {rt:gpr3}, {offset:u5}

Availability

!(NMS)

Decode priority

0

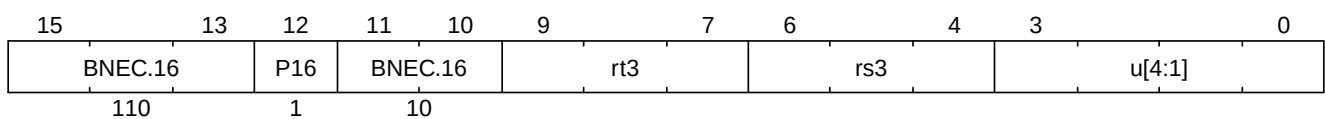


Figure 249. Encoding (16 bits)

Encoding constraints

- $rs3 \geq rt3$
- $u \neq 0$

Decode

```
exec.isCTI = True
rs = GPR3.decode(rs3)
rt = GPR3.decode(rt3)
u = zero_extend(u, from_nbits=5)
return BnecOperands(rs=rs, rt=rt, offset=u)
```

bnecc.32

Assembly

BNEC {rs:gpr}, {rt:gpr}, {offset:s15}

Availability

true

Decode priority

0

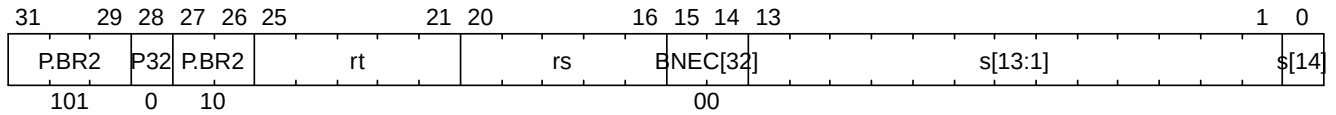


Figure 250. Encoding (32 bits)

Decode

```
exec.isCTI = True
s = sign_extend(s, from_nbits=15)
return BnecOperands(rs=rs, rt=rt, offset=s)
```

8.97.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
taken = ctx.GPR[operands.rs] != ctx.GPR[operands.rt]
compact_branch(
    ctx,
    address,
    taken=taken,
    has_forbidden_slot=exec.hasForbiddenSlot,
)
```

8.97.4. Description

The displacement is added to the address of the instruction following BNEC. The branch has no delay slot. The compact 16-bit form uses the encoded compact GPR mapping and an unsigned scaled displacement; the 32-bit form uses full GPR indices and a signed displacement.

8.97.5. Exceptions

None.

8.98. BNEIC

8.98.1. Purpose

Branch compactly when a GPR is not equal to an immediate.

8.98.2. Formats

nanoMIPS

bneic.32

Assembly

BNEIC {immediate:u7}, {offset:s12}

Availability

true

Decode priority

0



Figure 251. Encoding (32 bits)

Decode

```
exec.isCTI = True
immediate = zero_extend(u, from_nbits=7)
offset = sign_extend(s, from_nbits=12)
return BranchImmediateOperands(rt, immediate, offset)
```

8.98.3. Operation

```
lhs = ctx.GPR[operands.rt]
rhs = int(operands.immediate)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=lhs != rhs, has_forbidden_slot=exec.hasForbiddenSlot
)
```

8.98.4. Description

The target is relative to the following instruction and has no delay or forbidden slot.

8.98.5. Exceptions

None.

8.99. BNEL

8.99.1. Purpose

Branch likely when two GPR values are not equal.

8.99.2. Formats

MIPS

bnel

Assembly

BNEL {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR < R6

Decode priority

0

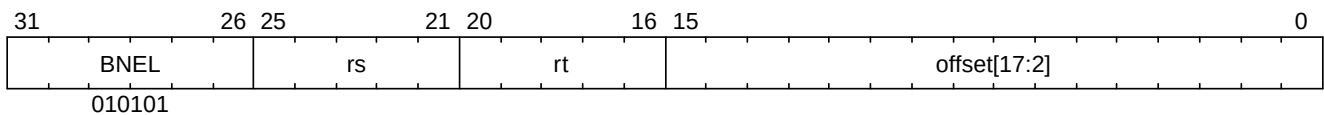


Figure 252. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return LegacyBranchOperands(
    rs=rs, rt=rt, offset=sign_extend(offset, from_nbits=18)
)
```

8.99.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width))
target = effective_address(ctx, ctx.next_pc, operands.offset)
delayed_branch_likely(ctx, target, taken=lhs != rhs)
```

8.99.4. Description

The signed, word-aligned displacement is relative to PC+4. The following instruction executes only when the branch is taken; otherwise that delay slot is nullified.

8.99.5. Exceptions

None.

8.100. BNEZALC

8.100.1. Purpose

Perform a compact branch-and-link when a GPR is ne zero.

8.100.2. Formats

MIPS

bnezalc

Assembly

BNEZALC {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

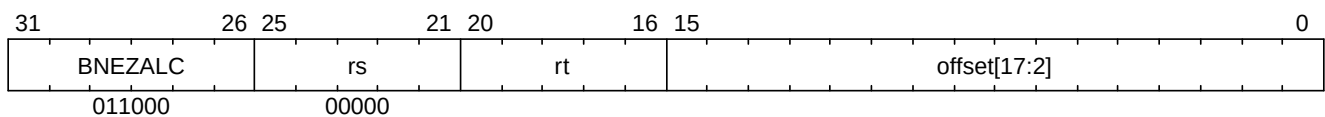


Figure 253. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt=0, rs=rt, offset=offset)
```

microMIPS

bnezalc.r6.32

Assembly

BNEZALC {rt:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

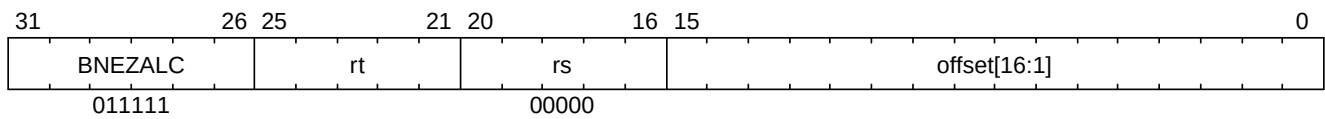


Figure 254. Encoding (32 bits)

Encoding constraints

- `rt != 0`

Decode

```

exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=0, rs=rt, offset=sign_extend(offset, from_nbits=17))

```

8.100.3. Operation

```

source = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
ctx.GPR[31] = ModeTaggedAddress(ctx, ctx.next_pc)
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx, target, taken=source != 0, has_forbidden_slot=exec.hasForbiddenSlot
)

```

8.100.4. Description

The signed displacement is scaled by the encoding's instruction alignment and added to the following-instruction address. The compact branch has no delay slot and has a forbidden slot.

8.100.5. Exceptions

None.

8.101. BNEZC

8.101.1. Purpose

Branch compactly when a GPR is nonzero.

8.101.2. Formats

MIPS

bnezc

Assembly

BNEZC {rt:gpr}, {offset:s23}

Availability

AR >= R6

Decode priority

0

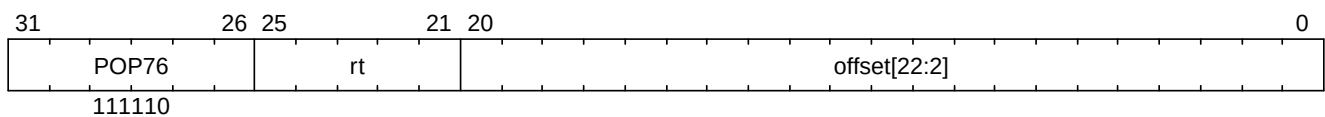


Figure 255. Encoding (32 bits)

Encoding constraints

- **rt** != 0

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return BnezcOperands(rt=rt, offset=sign_extend(offset, from_nbits=23))
```

microMIPS

bnezc.32

Assembly

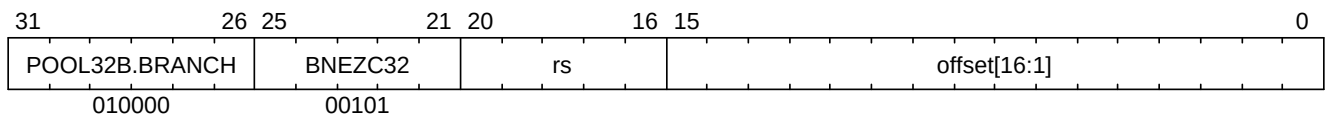
BNEZC {rt:gpr}, {offset:s17}

Availability

MMAR < R6

Decode priority

0



Encoding constraints

- `rs != 0`

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return BnezcOperands(rt=rs, offset=sign_extend(offset, from_nbits=22))
```

nanoMIPS

bnezc.16

Assembly

`BNEZC {rt:gpr3}, {offset:s8}`

Availability

true

Decode priority

0

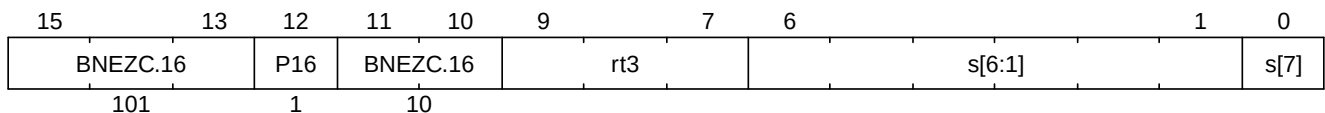


Figure 259. Encoding (16 bits)

Decode

```
exec.isCTI = True
return BnezcOperands(
    rt=GPR3.decode(rt3),
    offset=sign_extend(s, from_nbits=8),
)
```

8.101.3. Operation

```
address = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx,
    address,
    taken=ctx.GPR[operands.rt] != 0,
    has_forbidden_slot=exec.hasForbiddenSlot,
)
```

8.101.4. Description

BNEZC[16] sign-extends its eight-bit aligned displacement from the following instruction and

branches when the compact source GPR is nonzero.

8.101.5. Exceptions

None.

8.102. BNVC

8.102.1. Purpose

Perform a compact branch when signed addition no overflow.

8.102.2. Formats

MIPS

bnvc

Assembly

BNVC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

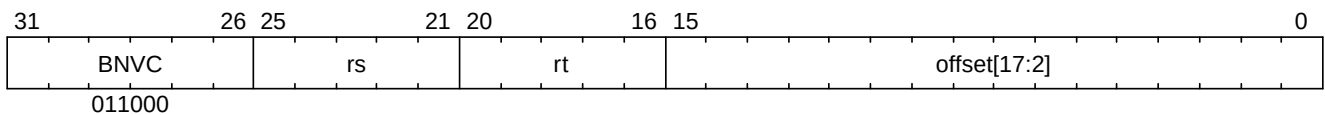


Figure 260. Encoding (32 bits)

Encoding constraints

- **rs** >= **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt, rs, offset)
```

microMIPS

bnvc.r6.32

Assembly

BNVC {rt:gpr}, {rs:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

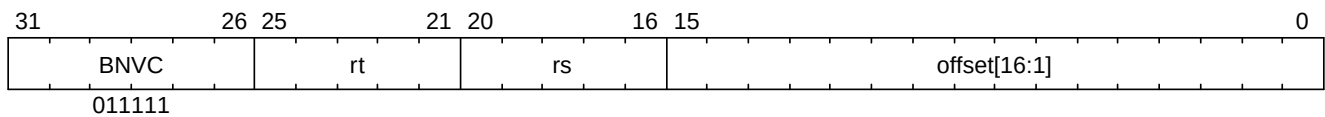


Figure 261. Encoding (32 bits)

Encoding constraints

- `rs >= rt`

Decode

```

exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=17))

```

8.102.3. Operation

```

width = ctx.GPR.width
lhs_bits = ctx.GPR[operands.rs]
rhs_bits = ctx.GPR[operands.rt]
lhs = int(sign_extend(lhs_bits, from_nbits=32))
rhs = int(sign_extend(rhs_bits, from_nbits=32))
result = lhs + rhs
input_overflow = not_word_value(lhs_bits, nbits=width) or not_word_value(
    rhs_bits, nbits=width
)
sum_overflow = result < -(1 << 31) or result > (1 << 31) - 1
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx,
    target,
    taken=not (sum_overflow or input_overflow),
    has_forbidden_slot=exec.hasForbiddenSlot,
)

```

8.102.4. Description

The signed GPR addition is discarded; only its overflow condition controls the branch. The MIPS32 R6 form has no delay slot and has a forbidden slot.

8.102.5. Exceptions

None.

8.103. BOVC

8.103.1. Purpose

Perform a compact branch when signed addition overflow.

8.103.2. Formats

MIPS

bovc

Assembly

BOVC {rs:gpr}, {rt:gpr}, {offset:s18}

Availability

AR >= R6

Decode priority

0

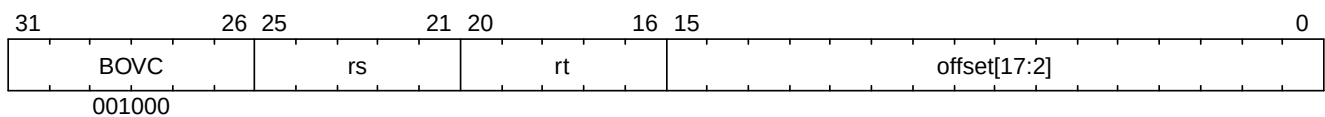


Figure 262. Encoding (32 bits)

Encoding constraints

- **rs** >= **rt**

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
offset = sign_extend(offset, from_nbits=18)
return MipsBranchOperands(rt, rs, offset)
```

microMIPS

bovc.r6.32

Assembly

BOVC {rt:gpr}, {rs:gpr}, {offset:s17}

Availability

MMAR >= R6

Decode priority

0

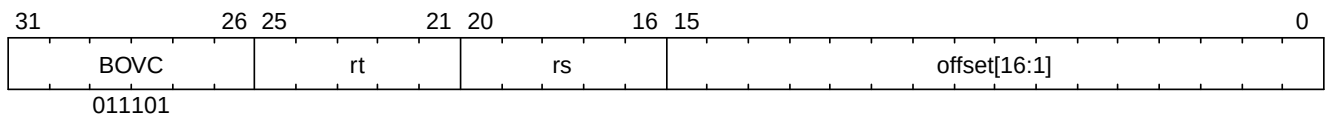


Figure 263. Encoding (32 bits)

Encoding constraints

- `rs >= rt`

Decode

```

exec.isCTI = True
exec.hasForbiddenSlot = True
return MipsBranchOperands(rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=17))

```

8.103.3. Operation

```

width = ctx.GPR.width
lhs_bits = ctx.GPR[operands.rs]
rhs_bits = ctx.GPR[operands.rt]
lhs = int(sign_extend(lhs_bits, from_nbits=32))
rhs = int(sign_extend(rhs_bits, from_nbits=32))
result = lhs + rhs
input_overflow = not_word_value(lhs_bits, nbits=width) or not_word_value(
    rhs_bits, nbits=width
)
sum_overflow = result < -(1 << 31) or result > (1 << 31) - 1
target = effective_address(ctx, ctx.next_pc, operands.offset)
compact_branch(
    ctx,
    target,
    taken=sum_overflow or input_overflow,
    has_forbidden_slot=exec.hasForbiddenSlot,
)

```

8.103.4. Description

The signed GPR addition is discarded; only its overflow condition controls the branch. The MIPS32 R6 form has no delay slot and has a forbidden slot.

8.103.5. Exceptions

None.

8.104. BPOSGE32

8.104.1. Purpose

Branch when DSPControl.Pos is at least 32.

8.104.2. Formats

MIPS

bposge32

Assembly

BPOSGE32 {offset:s18}

Availability

DSP

Decode priority

0

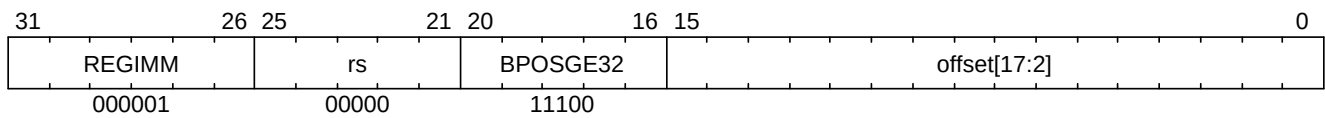


Figure 264. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return DspBranchOperands(sign_extend(offset, from_nbits=18))
```

microMIPS

bposge32.32

Assembly

BPOSGE32 {offset:s17}

Availability

MMAR < R6 && DSP

Decode priority

0

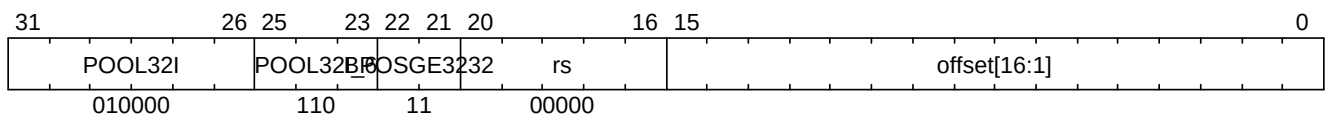


Figure 265. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return DspBranchOperands(sign_extend(offset, from_nbits=17))
```

8.104.3. Operation

```
validate_access_to_dsp_resources(ctx)
target = effective_address(ctx, ctx.next_pc, operands.offset)
taken = ctx.DSPControl.Pos >= 32
delayed_branch(ctx, target, taken=taken)
```

8.104.4. Description

BPOSGE32 validates DSP access and performs a delayed PC-relative branch using the instruction-local sequential next_pc.

8.104.5. Exceptions

None.

8.105. BPOSGE32C

8.105.1. Purpose

Compact branch when DSPControl.Pos is at least 32.

8.105.2. Formats

MIPS

bposge32c

Assembly

BPOSGE32C {offset:s18}

Availability

AR >= R6 && DSP && DSP2

Decode priority

0

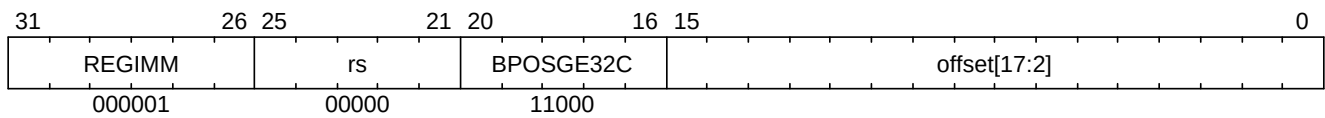


Figure 266. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasForbiddenSlot = True
return DspBranchOperands(sign_extend(offset, from_nbits=18))
```

microMIPS

bposge32c.32

Assembly

BPOSGE32C {offset:s17}

Availability

MMAR >= R6 && DSP && DSP2

Decode priority

0

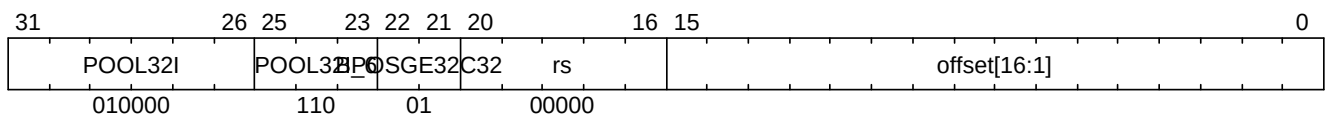


Figure 267. Encoding (32 bits)

Decode

```
exec.isCTI = True
return DspBranchOperands(sign_extend(offset, from_nbits=17))
```

nanoMIPS

bposge32c.nano.32

Assembly

BPOSGE32C {offset:s15}

Availability

DSP && DSP2

Decode priority

0

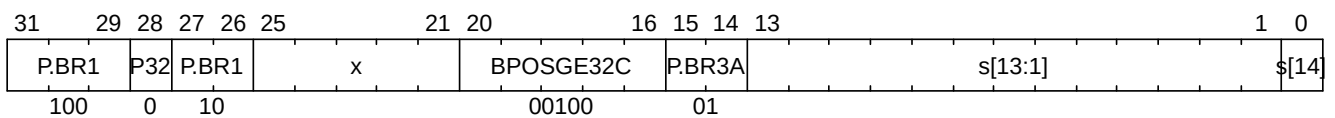


Figure 268. Encoding (32 bits)

Decode

```
exec.isCTI = True
return DspBranchOperands(sign_extend(s, from_nbits=15))
```

8.105.3. Operation

```
validate_access_to_dsp_resources(ctx)
target = effective_address(ctx, ctx.next_pc, operands.offset)
taken = ctx.DSPControl.Pos >= 32
compact_branch(ctx, target, taken=taken, has_forbidden_slot=exec.hasForbiddenSlot)
```

8.105.4. Description

BPOSGE32C validates DSP access and performs a compact PC-relative branch. The MIPS Release 6 form creates a forbidden slot; microMIPS and nanoMIPS forms do not.

8.105.5. Exceptions

None.

8.106. BREAK

8.106.1. Purpose

Unconditionally raise a breakpoint exception.

8.106.2. Formats

MIPS

`break`

Assembly

`BREAK {code:u20}`

Availability

true

Decode priority

0

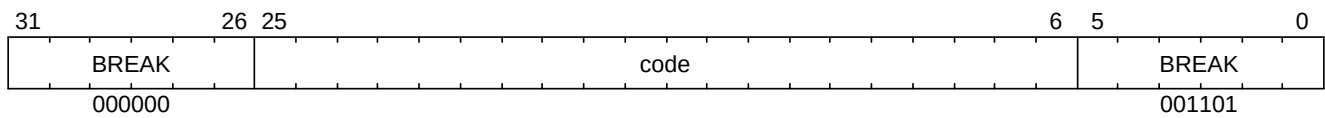


Figure 269. Encoding (32 bits)

Decode

```
return BreakOperands(code=zero_extend(code, from_nbits=20))
```

microMIPS

`break.16`

Assembly

`BREAK {code:u4}`

Availability

MMAR < R6

Decode priority

0

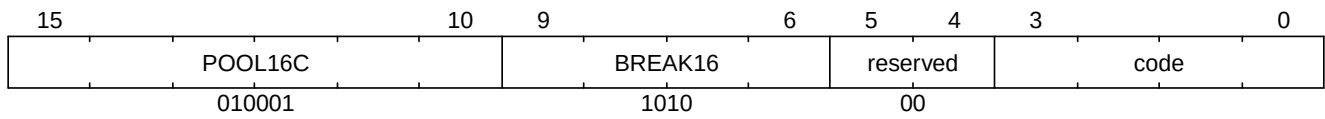


Figure 270. Encoding (16 bits)

Decode

```
return BreakOperands(code=zero_extend(code, from_nbits=4))
```

break.16.r6

Assembly

BREAK {code:u4}

Availability

MMAR >= R6

Decode priority

0

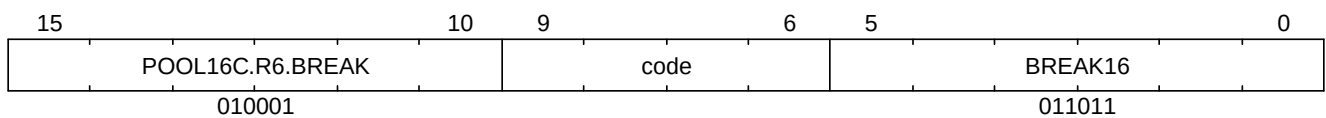


Figure 271. Encoding (16 bits)

Decode

```
return BreakOperands(code=zero_extend(code, from_nbits=4))
```

break.32

Assembly

BREAK {code:u20}

Availability

true

Decode priority

0

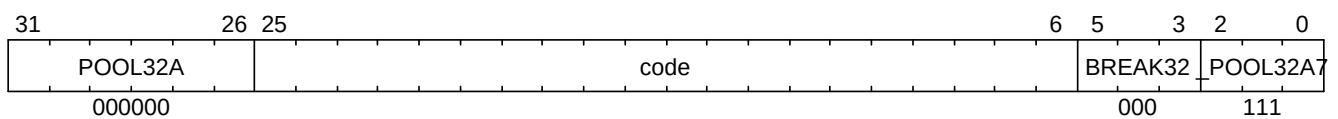


Figure 272. Encoding (32 bits)

Decode

```
return BreakOperands(code=zero_extend(code, from_nbits=20))
```

nanoMIPS

break.16

Assembly

BREAK {code:u3}

Availability

true

Decode priority

0

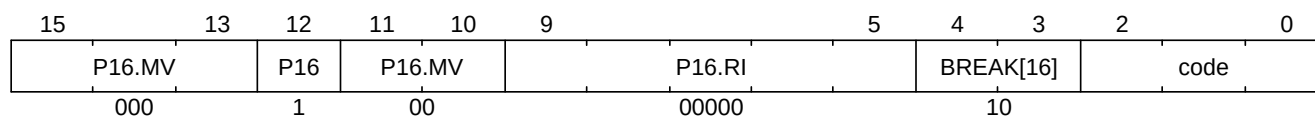


Figure 273. Encoding (16 bits)

Decode

```
code = zero_extend(code, from_nbits=3)
return BreakOperands(code=code)
```

break.32

Assembly

BREAK {code:u19}

Availability

true

Decode priority

0

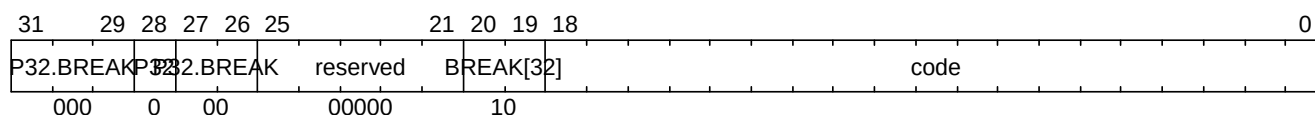


Figure 274. Encoding (32 bits)

Decode

```
code = zero_extend(code, from_nbits=19)
return BreakOperands(code=code)
```

8.106.3. Operation

```
raise BreakpointException("BREAK instruction")
```

8.106.4. Description

BREAK carries an implementation-visible code in either a compact three-bit or full nineteen-bit field. The architectural operation always raises the BP exception.

8.106.5. Exceptions

- **BP** (Breakpoint) — BREAK instruction

8.107. BRSC

8.107.1. Purpose

Branch to a scaled register-relative address.

8.107.2. Formats

nanoMIPS

brsc.32

Assembly

BRSC {rs:gpr}

Availability

true

Decode priority

0

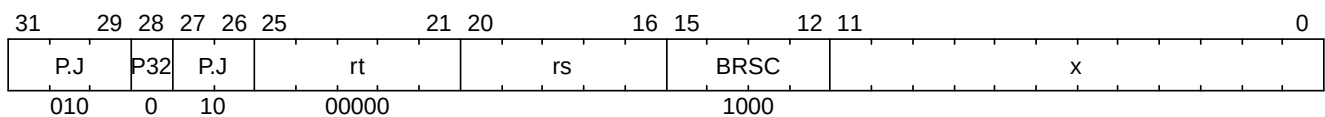


Figure 275. Encoding (32 bits)

Decode

```
exec.isCTI = True
return BrscOperands(rs=rs)
```

8.107.3. Operation

```
target = effective_address(ctx, ctx.next_pc, ctx.GPR[operands.rs] << 1)
compact_branch(ctx, target)
```

8.107.4. Description

BRSC adds twice the source GPR to the address following the instruction and performs a compact transfer.

8.107.5. Exceptions

None.

8.108. C.cond.D

8.108.1. Purpose

Compare D-format values and write an FCC bit.

8.108.2. Formats

MIPS

`c.cond.d`

Assembly

`C.cond.D {cc:u3}, {fs:fpr}, {ft:fpr}, {condition:u4}`

Availability

$AR < R6$

Decode priority

0

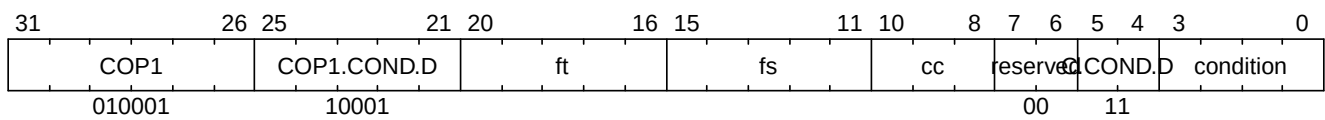


Figure 276. Encoding (32 bits)

Decode

```
return FpuLegacyCompareOperands(  
    fs=fs, ft=ft, cc=cc, condition=zero_extend(condition, from_nbits=4)  
)
```

microMIPS

`c.cond.d`

Assembly

`C.cond.D {cc:u3}, {fs:fpr}, {ft:fpr}, {condition:u4}`

Availability

$MMAR < R6$

Decode priority

0

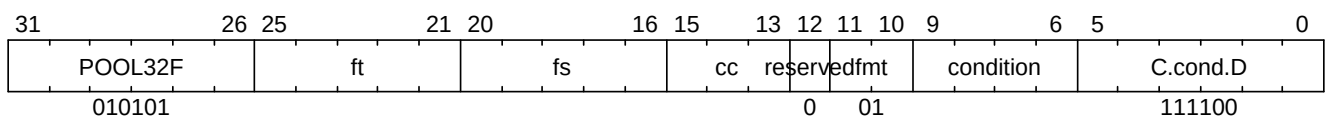


Figure 277. Encoding (32 bits)

```
return FpuLegacyCompareOperands(
    fs=fs, ft=ft, cc=cc, condition=zero_extend(condition, from_nbits=4)
)
```

8.108.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
condition, signaling = interpret_compare_condition(operands.condition)
matched, flags = ieee_compare(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    condition,
    signaling=signaling,
)
commit_fpu_flags(ctx, flags)
write_fcc(ctx, operands.cc, matched)
```

8.108.4. Description

The condition field selects the legacy quiet or signaling comparison; assemblers expose named C.cond.D aliases.

8.108.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.109. C.cond.PS

8.109.1. Purpose

Compare paired-single values and write an FCC pair.

8.109.2. Formats

MIPS

`c.cond.ps`

Assembly

`C.cond.PS {cc:u3}, {fs:fpr}, {ft:fpr}, {condition:u4}`

Availability

AR < R6

Decode priority

0

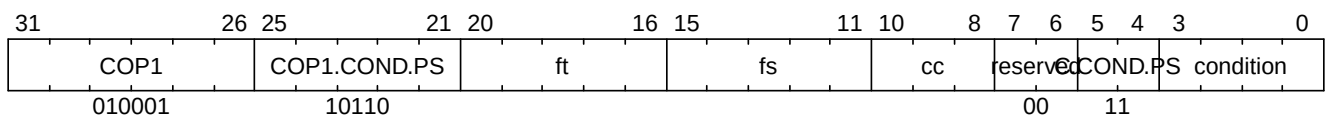


Figure 278. Encoding (32 bits)

Decode

```
return FpuLegacyCompareOperands(
    fs=fs, ft=ft, cc=cc, condition=zero_extend(condition, from_nbits=4)
)
```

microMIPS

`c.cond.ps`

Assembly

`C.cond.PS {cc:u3}, {fs:fpr}, {ft:fpr}, {condition:u4}`

Availability

MMAR < R6

Decode priority

0

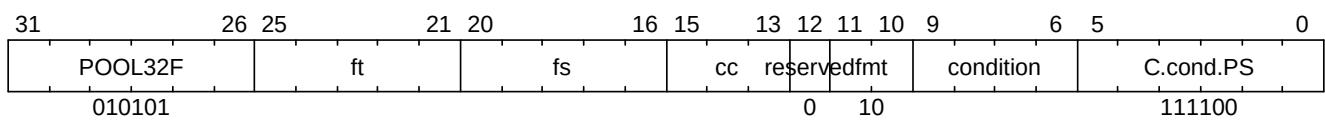


Figure 279. Encoding (32 bits)

```

return FpuLegacyCompareOperands(
    fs=fs, ft=ft, cc=cc, condition=zero_extend(condition, from_nbits=4)
)

```

8.109.3. Operation

```

validate_access_to_cp1_resources(ctx, "PS")
cc = int(operands.cc)
if cc & 1 or cc > 6:
    raise ReservedInstruction("C.cond.PS requires an even FCC pair")
condition, signaling = interpret_compare_condition(operands.condition)
left = get_fpr(ctx, operands.fs, "PS")
right = get_fpr(ctx, operands.ft, "PS")
lower, lower_flags = ieee_compare(
    "S",
    fpu_rounding_mode(ctx),
    left & 0xFFFF_FFFF,
    right & 0xFFFF_FFFF,
    condition,
    signaling=signaling,
)
upper, upper_flags = ieee_compare(
    "S",
    fpu_rounding_mode(ctx),
    left >> 32,
    right >> 32,
    condition,
    signaling=signaling,
)
commit_fpu_flags(ctx, lower_flags | upper_flags)
write_fcc(ctx, cc, lower)
write_fcc(ctx, cc + 1, upper)

```

8.109.4. Description

C.cond.PS compares the lower and upper S-format lanes independently and writes the results to FCC[cc] and FCC[cc+1].

8.109.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit

operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; C.cond.PS requires an even FCC pair

8.110. C.cond.S

8.110.1. Purpose

Compare S-format values and write an FCC bit.

8.110.2. Formats

MIPS

`c.cond.s`

Assembly

`C.cond.S {cc:u3}, {fs:fpr}, {ft:fpr}, {condition:u4}`

Availability

AR < R6

Decode priority

0

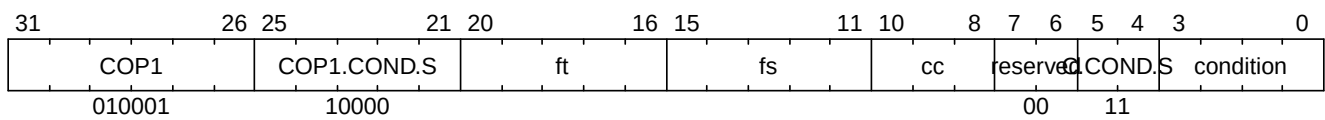


Figure 280. Encoding (32 bits)

Decode

```
return FpuLegacyCompareOperands(
    fs=fs, ft=ft, cc=cc, condition=zero_extend(condition, from_nbits=4)
)
```

microMIPS

`c.cond.s`

Assembly

`C.cond.S {cc:u3}, {fs:fpr}, {ft:fpr}, {condition:u4}`

Availability

MMAR < R6

Decode priority

0

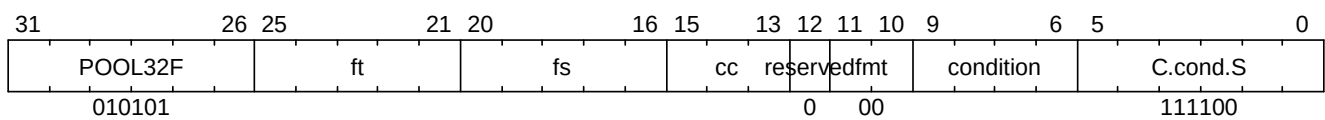


Figure 281. Encoding (32 bits)

```
return FpuLegacyCompareOperands(
    fs=fs, ft=ft, cc=cc, condition=zero_extend(condition, from_nbits=4)
)
```

8.110.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
condition, signaling = interpret_compare_condition(operands.condition)
matched, flags = ieee_compare(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    condition,
    signaling=signaling,
)
commit_fpu_flags(ctx, flags)
write_fcc(ctx, operands.cc, matched)
```

8.110.4. Description

The condition field selects the legacy quiet or signaling comparison; assemblers expose named C.cond.S aliases.

8.110.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.111. CACHE

8.111.1. Purpose

Perform an implementation-defined cache operation at an address.

8.111.2. Formats

MIPS

cache

Assembly

CACHE {op:u5}, {offset:s16}({rs:gpr})

Availability

AR < R6

Decode priority

0

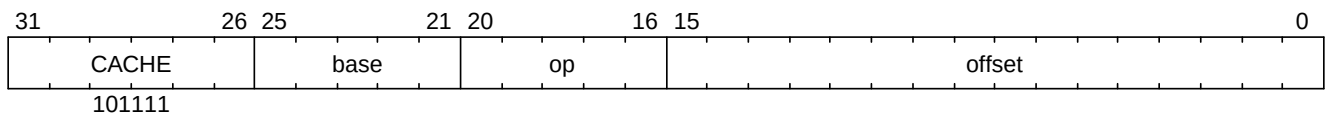


Figure 282. Encoding (32 bits)

Decode

```
return CacheOperands(  
    op=zero_extend(op, from_nbits=5),  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
    eva=False,  
)
```

cache.r6

Assembly

CACHE {op:u5}, {offset:s9}({rs:gpr})

Availability

AR >= R6

Decode priority

0

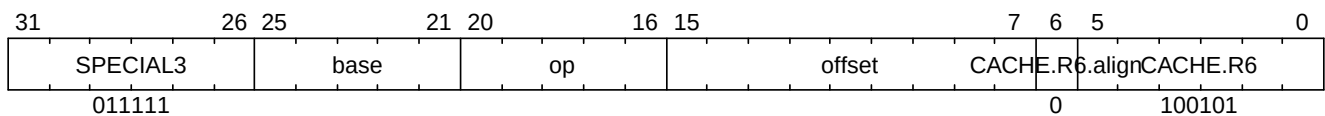


Figure 283. Encoding (32 bits)

Decode

```
return CacheOperands(  
    op=zero_extend(op, from_nbits=5),  
    rs=base,  
    offset=sign_extend(offset, from_nbits=9),  
    eva=False,  
)
```

microMIPS

cache.32

Assembly

CACHE {op:u5}, {offset:s12}({rs:gpr})

Availability

MMAR < R6

Decode priority

0

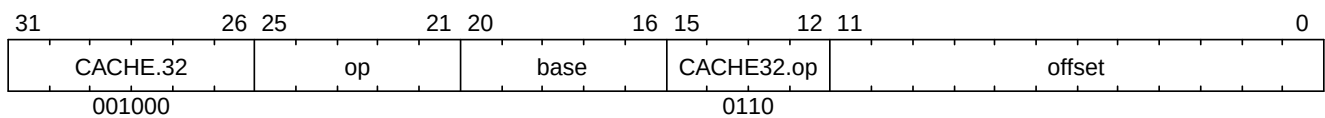


Figure 284. Encoding (32 bits)

Decode

```
return CacheOperands(  
    op=zero_extend(op, from_nbits=5),  
    rs=base,  
    offset=sign_extend(offset, from_nbits=12),  
    eva=False,  
)
```

nanoMIPS

cache.eva

Assembly

CACHE {op:u5}, {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

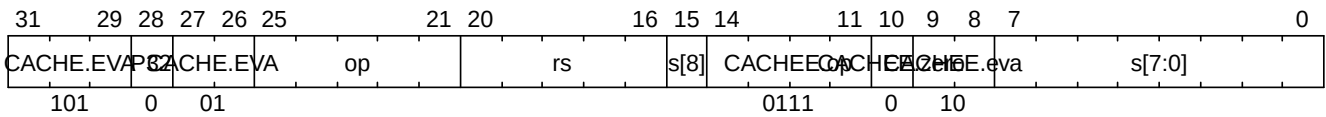


Figure 285. Encoding (32 bits)

Decode

```
return CacheOperands(
    op=zero_extend(op, from_nbits=5),
    rs=rs,
    offset=sign_extend(s, from_nbits=9),
    eva=True,
)
```

cache.s9

Assembly

CACHE {op:u5}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

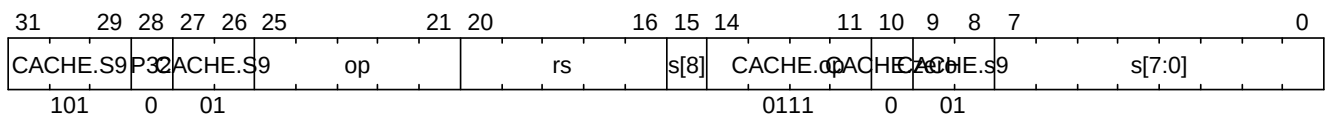


Figure 286. Encoding (32 bits)

Decode

```
return CacheOperands(
    op=zero_extend(op, from_nbits=5),
    rs=rs,
    offset=sign_extend(s, from_nbits=9),
    eva=False,
)
```

8.111.3. Operation

```
if operands.eva and not ctx.C0.Config5.EVA:
    raise ReservedInstruction("CACHEE requires Config5.EVA")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)

base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
address = effective_address(ctx, base, operands.offset, "Cacheop", eva=operands.eva)
```

```
cacheop(ctx, address, operands.op, eva=operands.eva)
```

8.111.4. Description

CACHE computes an effective address from a GPR and signed nine-bit offset, then sends the five-bit operation code to the optional memory backend cache-operation hook. CACHEE requests EVA addressing and requires Config5.EVA. No data-memory mutation is assumed when the backend does not implement the hook.

8.111.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — CACHEE requires Config5.EVA

8.112. CEIL.L.D

8.112.1. Purpose

Convert D format to L format using ceil rounding.

8.112.2. Formats

MIPS

ceil.l.d

Assembly

CEIL.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

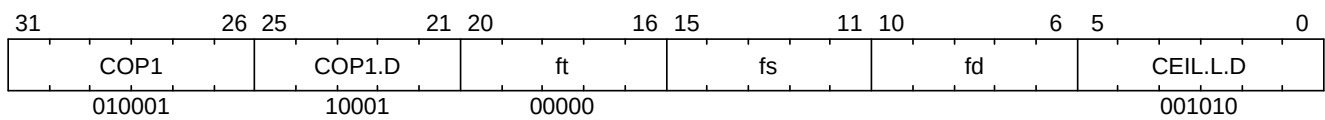


Figure 287. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

ceil.l.d

Assembly

CEIL.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

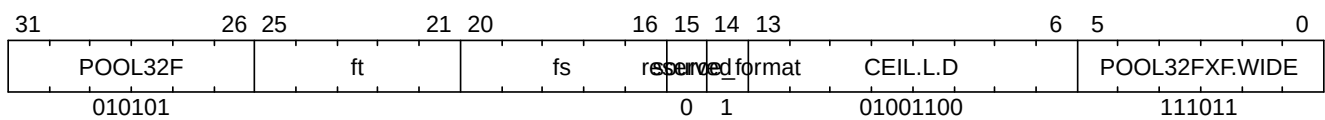


Figure 288. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

ceil.l.d

Assembly

CEIL.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0



Figure 289. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.112.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "D")
result, flags = ieee_float_to_int(
    "D", "L", fpu_rounding_mode(ctx, 2), get_fpr(ctx, operands.fs, "D"), rm=2
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.112.4. Description

CEIL.L.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.112.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.113. CEIL.L.S

8.113.1. Purpose

Convert S format to L format using ceil rounding.

8.113.2. Formats

MIPS

ceil.l.s

Assembly

CEIL.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

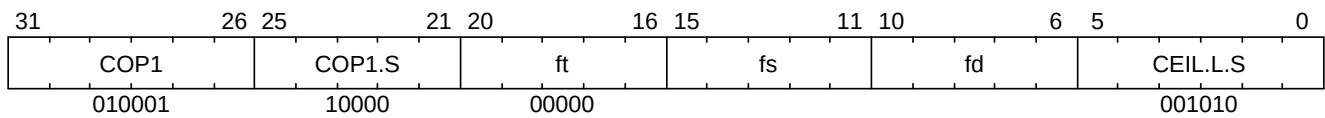


Figure 290. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

ceil.l.s

Assembly

CEIL.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

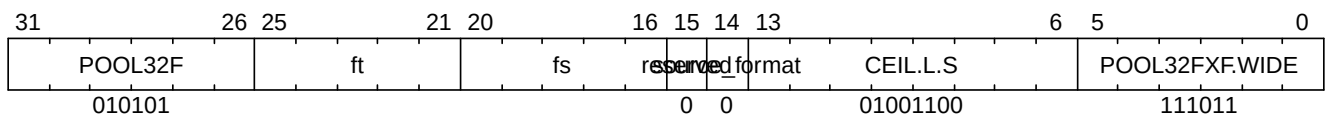


Figure 291. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

ceil.l.s

Assembly

CEIL.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

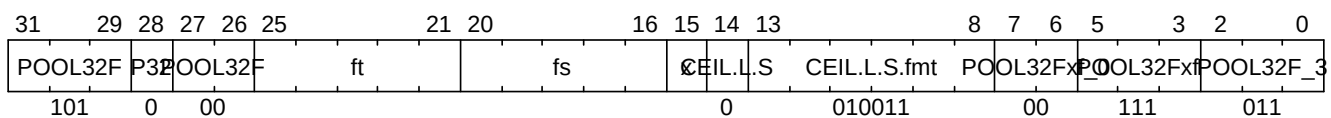


Figure 292. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.113.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "S")
result, flags = ieee_float_to_int(
    "S", "L", fpu_rounding_mode(ctx, 2), get_fpr(ctx, operands.fs, "S"), rm=2
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.113.4. Description

CEIL.L.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.113.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.114. CEIL.W.D

8.114.1. Purpose

Convert D format to W format using ceil rounding.

8.114.2. Formats

MIPS

ceil.w.d

Assembly

CEIL.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

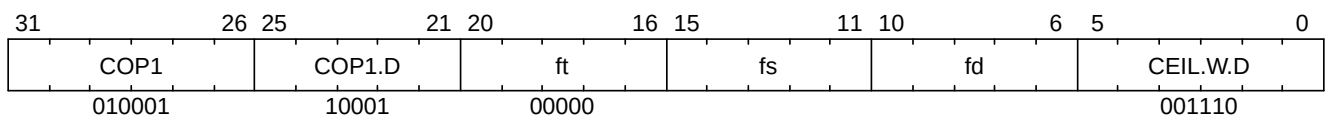


Figure 293. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

ceil.w.d

Assembly

CEIL.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

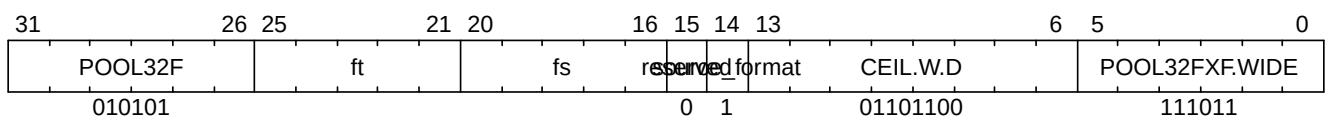


Figure 294. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

ceil.w.d

Assembly

CEIL.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

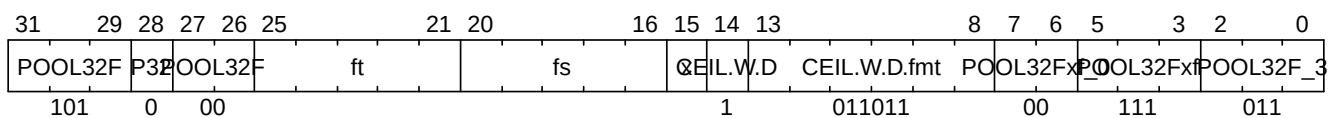


Figure 295. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.114.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "D")
result, flags = ieee_float_to_int(
    "D", "W", fpu_rounding_mode(ctx, 2), get_fpr(ctx, operands.fs, "D"), rm=2
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.114.4. Description

CEIL.W.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.114.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.115. CEIL.W.S

8.115.1. Purpose

Convert S format to W format using ceil rounding.

8.115.2. Formats

MIPS

ceil.w.s

Assembly

CEIL.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

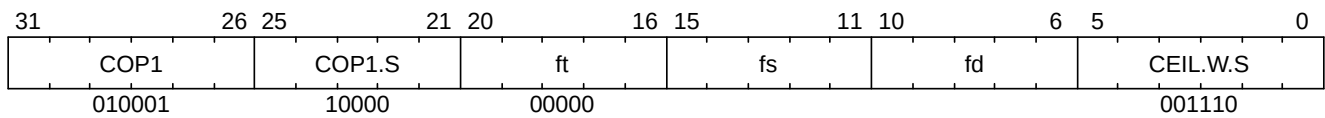


Figure 296. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

ceil.w.s

Assembly

CEIL.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

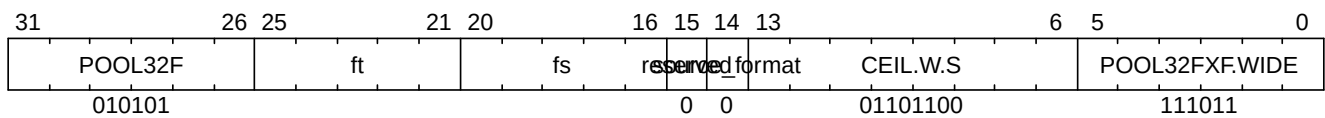


Figure 297. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

ceil.w.s

Assembly

CEIL.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

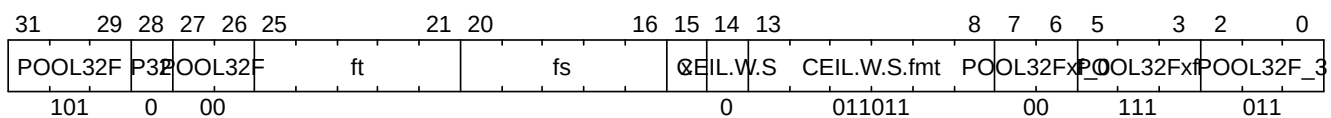


Figure 298. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.115.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "S")
result, flags = ieee_float_to_int(
    "S", "W", fpu_rounding_mode(ctx, 2), get_fpr(ctx, operands.fs, "S"), rm=2
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.115.4. Description

CEIL.W.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.115.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.116. CFC1

8.116.1. Purpose

Transfer a value through Coprocessor 1.

8.116.2. Formats

MIPS

cfc1

Assembly

CFC1 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

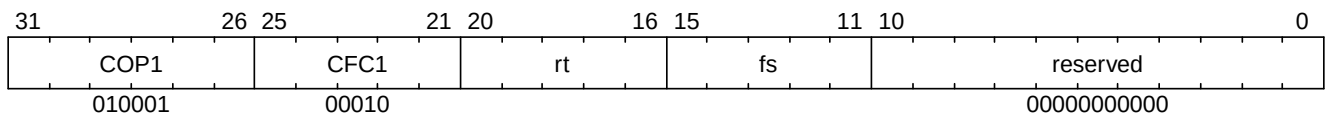


Figure 299. Encoding (32 bits)

Decode

```
return CFC10perands(rt=rt, cs=fs)
```

microMIPS

cfc1.32

Assembly

CFC1 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

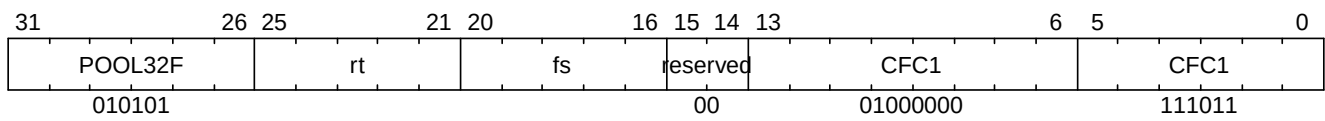


Figure 300. Encoding (32 bits)

Decode

```
return CFC10perands(rt=rt, cs=fs)
```

nanoMIPS

cfc1.32

Assembly

CFC1 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

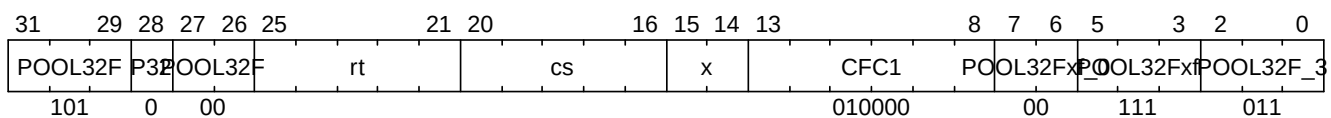


Figure 301. Encoding (32 bits)

Decode

```
return CFC10perands(rt=rt, cs=cs)
```

8.116.3. Operation

```
validate_access_to_cp1_resources(ctx)
ctx.GPR[operands.rt] = sign_extend(
    read_cp1_control(ctx, operands.cs), from_nbits=32
)
```

8.116.4. Description

CFC1 performs its CP1 access checks before transferring data.

8.116.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; The instruction or selected behavior is reserved in the current state.; C1.UNFR does not support reads; C1.NFRE does not support reads; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.

8.117. CFC2

8.117.1. Purpose

Move a CP2 control register to a GPR.

8.117.2. Formats

microMIPS

cfc2.32

Assembly

CFC2 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

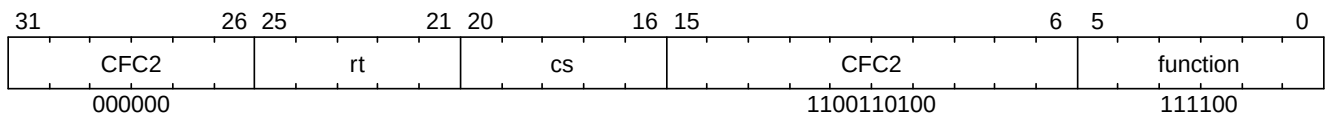


Figure 302. Encoding (32 bits)

Decode

```
return Cop2MoveOperands(rt=rt, cs=cs)
```

8.117.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("CFC2 execution is not implemented")
```

8.117.4. Description

CFC2 checks CP2 availability before accessing the register.

8.117.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; CFC2 execution is not

implemented

8.118. CLASS.D

8.118.1. Purpose

Classify a D-format value.

8.118.2. Formats

MIPS

`class.d`

Assembly

`CLASS.D {fd:fpr}, {fs:fpr}`

Availability

AR >= R6

Decode priority

0

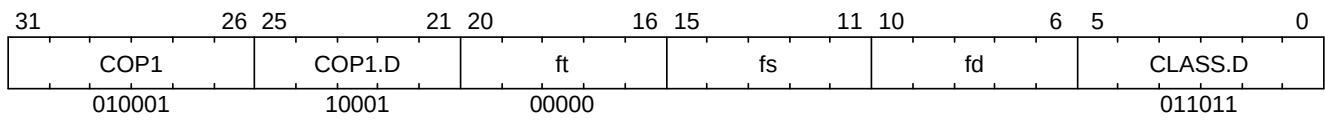


Figure 303. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

`class.d`

Assembly

`CLASS.D {fd:fpr}, {fs:fpr}`

Availability

MMAR >= R6

Decode priority

0

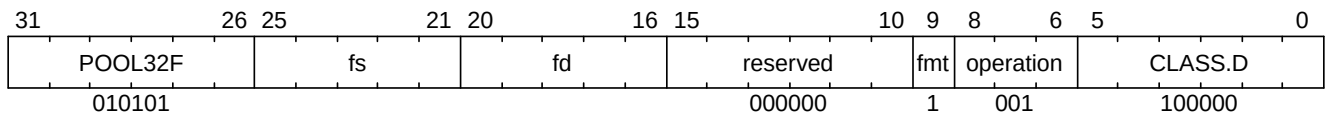


Figure 304. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

nanoMIPS

class.d

Assembly

CLASS.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

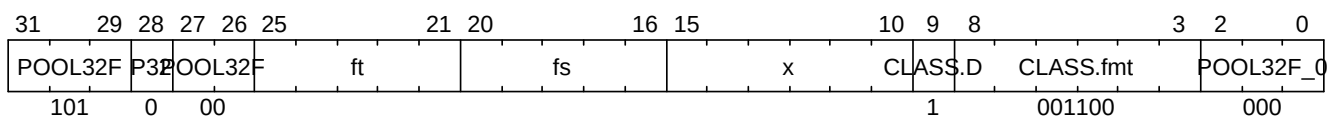


Figure 305. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.118.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
set_fpr(
    ctx,
    operands.fd,
    ieee_class_mask("D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")),
    "D",
)
```

8.118.4. Description

CLASS.D validates CP1 and D-format support before updating the destination FPR.

8.118.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported

8.119. CLASS.S

8.119.1. Purpose

Classify an S-format value.

8.119.2. Formats

MIPS

`class.s`

Assembly

`CLASS.S {fd:fpr}, {fs:fpr}`

Availability

AR >= R6

Decode priority

0

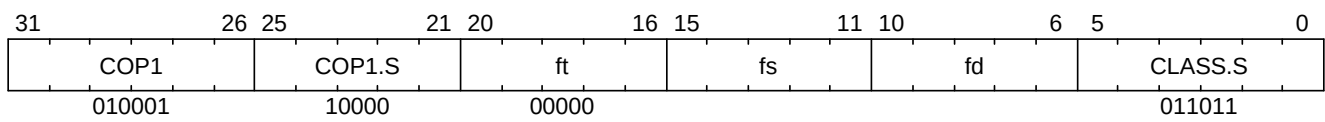


Figure 306. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

`class.s`

Assembly

`CLASS.S {fd:fpr}, {fs:fpr}`

Availability

MMAR >= R6

Decode priority

0

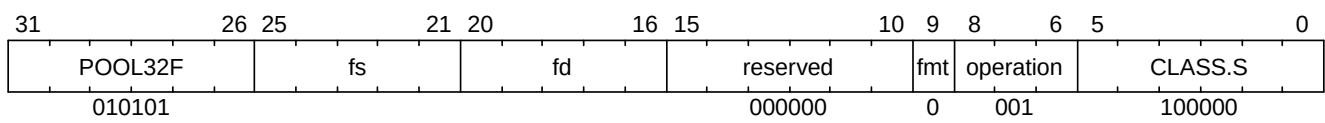


Figure 307. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

nanoMIPS

class.s

Assembly

CLASS.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

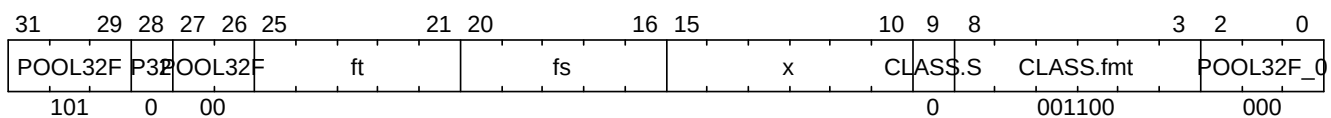


Figure 308. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.119.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
set_fpr(
    ctx,
    operands.fd,
    ieee_class_mask("S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")),
    "S",
)
```

8.119.4. Description

CLASS.S validates CP1 and S-format support before updating the destination FPR.

8.119.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported

8.120. CLO

8.120.1. Purpose

Count leading ones in the 32-bit value in **rs** and place the count in **rt**.

8.120.2. Formats

MIPS

clo

Assembly

CLO {rt:gpr}, {rs:gpr}

Availability

AR < R6

Decode priority

0

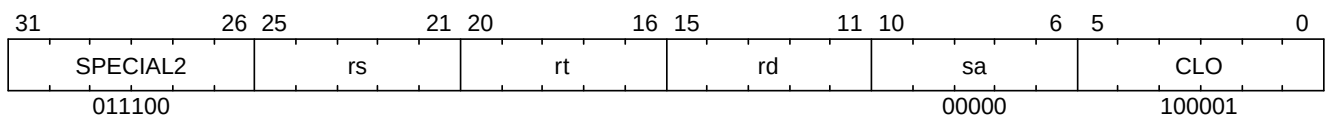


Figure 309. Encoding (32 bits)

Decode

```
return CloOperands(rt=rd, rs=rs)
```

r6.clo

Assembly

CLO {rt:gpr}, {rs:gpr}

Availability

AR >= R6

Decode priority

0

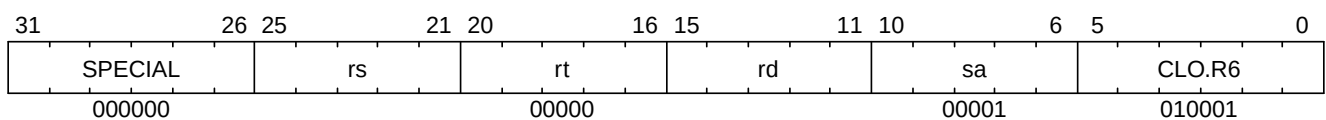


Figure 310. Encoding (32 bits)

Decode

```
return CloOperands(rt=rd, rs=rs)
```

microMIPS

clo.32

Assembly

CLO {rt:gpr}, {rs:gpr}

Availability

true

Decode priority

0

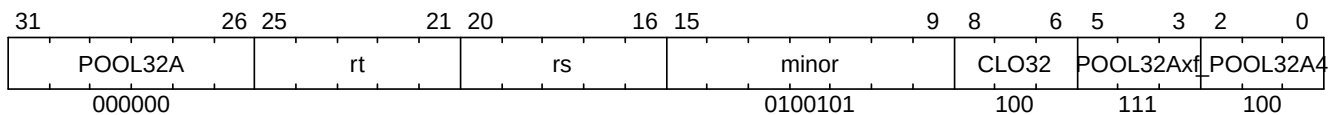


Figure 311. Encoding (32 bits)

Decode

```
return CloOperands(rt=rt, rs=rs)
```

nanoMIPS

clo.32

Assembly

CLO {rt:gpr}, {rs:gpr}

Availability

!(NMS)

Decode priority

0

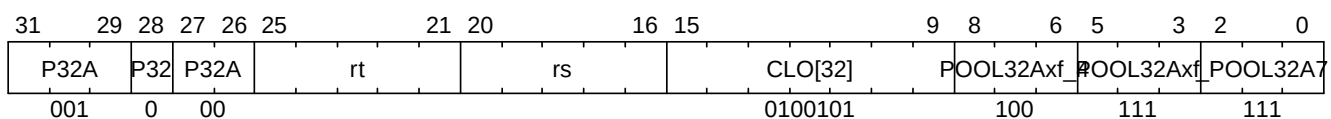


Figure 312. Encoding (32 bits)

Decode

```
return CloOperands(rt=rt, rs=rs)
```

8.120.3. Operation

```
input_value = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
count = 0
while count < 32:
    if int(input_value) & (1 << (31 - count)) == 0:
```



```
break  
count += 1  
ctx.GPR[operands.rt] = count
```

8.120.4. Description

CLO examines bits 31 through 0 of the source GPR, ignoring any higher implemented GPR bits. The result is an integer from 0 through 32 and is written to the destination GPR. The instruction is reserved when the nanoMIPS subset (NMS) is enabled.

8.120.5. Exceptions

- **RI** (Reserved Instruction) — CLO[32] is reserved when the NMS architecture feature is enabled.; CLO[32] is reserved when C0.Config5.NMS is set.

8.121. CLZ

8.121.1. Purpose

Count leading zero bits in the low 32 bits of a source GPR and write the count to a destination GPR.

8.121.2. Formats

MIPS

clz

Assembly

CLZ {rt:gpr}, {rs:gpr}

Availability

AR < R6

Decode priority

0

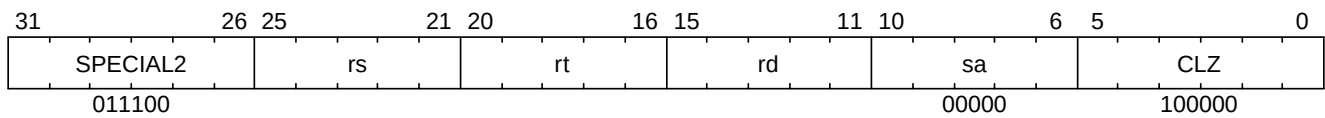


Figure 313. Encoding (32 bits)

Decode

```
return Clz0perands(rt=rd, rs=rs)
```

r6.clz

Assembly

CLZ {rt:gpr}, {rs:gpr}

Availability

AR >= R6

Decode priority

0

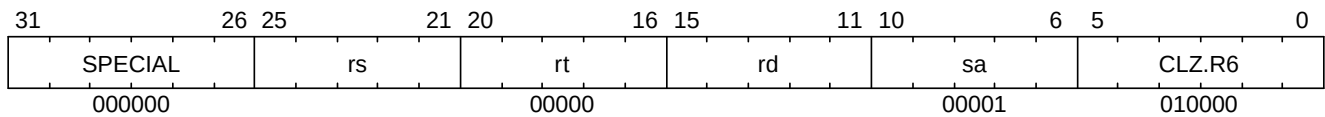


Figure 314. Encoding (32 bits)

Decode

```
return Clz0perands(rt=rd, rs=rs)
```

microMIPS

clz.32

Assembly

CLZ {rt:gpr}, {rs:gpr}

Availability

true

Decode priority

0

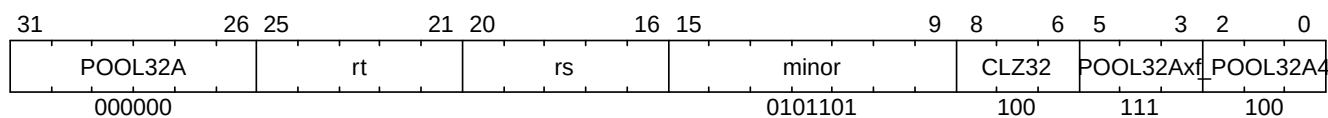


Figure 315. Encoding (32 bits)

Decode

```
return ClzOperands(rt=rt, rs=rs)
```

nanoMIPS

clz.32

Assembly

CLZ {rt:gpr}, {rs:gpr}

Availability

!(NMS)

Decode priority

0

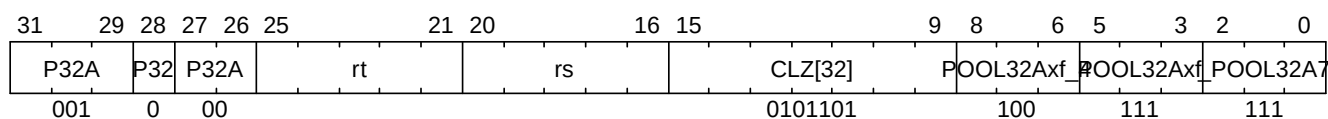


Figure 316. Encoding (32 bits)

Decode

```
return ClzOperands(rt=rt, rs=rs)
```

8.121.3. Operation

```
count = count_leading_bits(ctx.GPR[operands.rs], nbits=32, bit=0)
ctx.GPR[operands.rt] = count
```

8.121.4. Description

CLZ[32] examines exactly 32 input bits, regardless of the implemented GPR width. The result is an integer from 0 through 32 and is written to the full-width destination GPR.

8.121.5. Exceptions

- **RI** (Reserved Instruction) — CLZ is reserved when CP0 Config5.NMS is set.; CLZ[32] is reserved when C0.Config5.NMS is set.

8.122. CMP.D

8.122.1. Purpose

Compare two D-format values and write a Boolean mask to an FPR.

8.122.2. Formats

MIPS

`cmp.d`

Assembly

`CMP.D {fd:fpr}, {fs:fpr}, {ft:fpr}, {condition:u5}`

Availability

AR >= R6

Decode priority

0

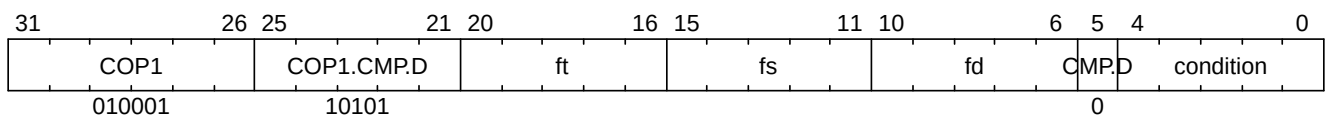


Figure 317. Encoding (32 bits)

Decode

```
return FpuCompareOperands(  
    fd=fd, fs=fs, ft=ft, condition=zero_extend(condition, from_nbits=5)  
)
```

microMIPS

`cmp.d`

Assembly

`CMP.D {fd:fpr}, {fs:fpr}, {ft:fpr}, {condition:u5}`

Availability

MMAR >= R6

Decode priority

0

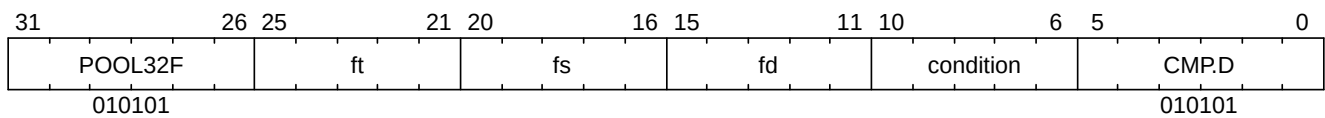


Figure 318. Encoding (32 bits)

Decode

```
return FpuCompareOperands(
    fd=fd, fs=fs, ft=ft, condition=zero_extend(condition, from_nbits=5)
)
```

nanoMIPS

cmp.d

Assembly

CMP.D {fd:fpr}, {fs:fpr}, {ft:fpr}, {condition:u5}

Availability

true

Decode priority

0

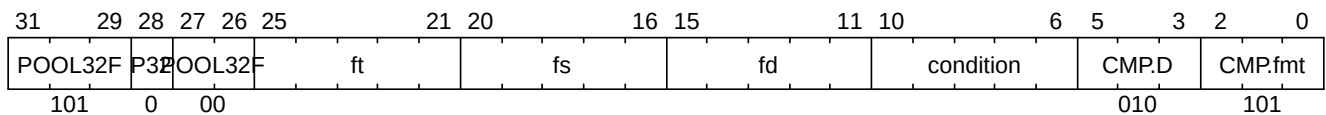


Figure 319. Encoding (32 bits)

Decode

```
return FpuCompareOperands(
    fd=fd, fs=fs, ft=ft, condition=zero_extend(condition, from_nbits=5)
)
```

8.122.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
if operands.condition in {
    0x10,
    0x14,
    0x15,
    0x16,
    0x17,
    0x18,
    0x1C,
    0x1D,
    0x1E,
    0x1F,
}:
    raise ReservedInstruction("floating comparison condition is not implemented")
condition, signaling = interpret_compare_condition(operands.condition)
matched, flags = ieee_compare(
    "D",
```

```

    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    condition,
    signaling=signaling,
)
write_fpu_result(
    ctx, operands.fd, 0xFFFF_FFFF_FFFF_FFFF if matched else 0, flags, "D"
)

```

8.122.4. Description

CMP.D supports the Release 6 quiet and signaling comparison-condition encodings.

8.122.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; floating comparison condition is not implemented

8.123. CMP.EQ.PH

8.123.1. Purpose

DSP packed operation CMP.EQ.PH.

8.123.2. Formats

MIPS

cmp.eq.ph

Assembly

CMP.EQ.PH {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

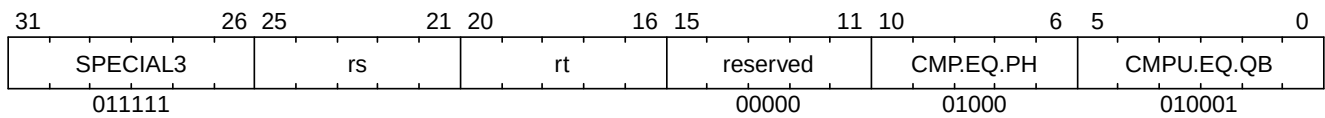


Figure 320. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

microMIPS

cmp.eq.ph.32

Assembly

CMP.EQ.PH {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

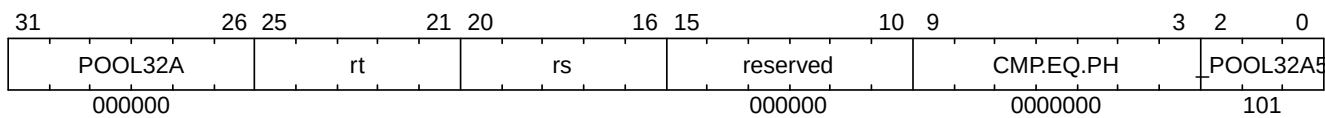


Figure 321. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

nanoMIPS

cmp.eq.ph.32

Assembly

CMP.EQ.PH {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

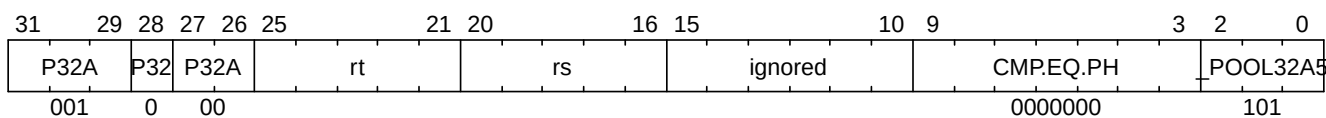


Figure 322. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

8.123.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.DSPControl.Ccond = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, "eq", signed_values=True
)
```

8.123.4. Description

CMP.EQ.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.123.5. Exceptions

None.

8.124. CMP.LE.PH

8.124.1. Purpose

DSP packed operation CMP.LE.PH.

8.124.2. Formats

MIPS

cmp.le.ph

Assembly

CMP.LE.PH {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

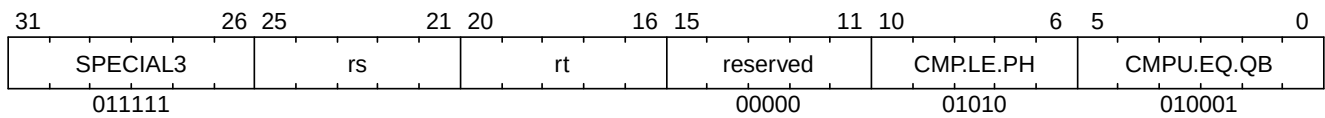


Figure 323. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

microMIPS

cmp.le.ph.32

Assembly

CMP.LE.PH {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

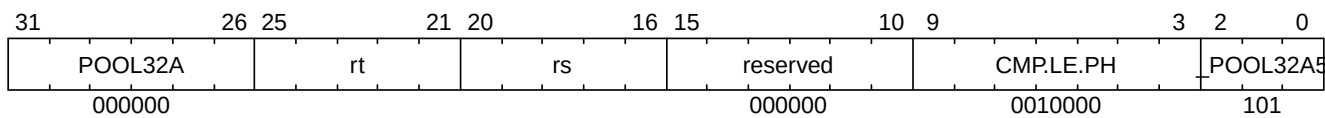


Figure 324. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

nanoMIPS

`cmp.le.ph.32`

Assembly

`CMP.LE.PH {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

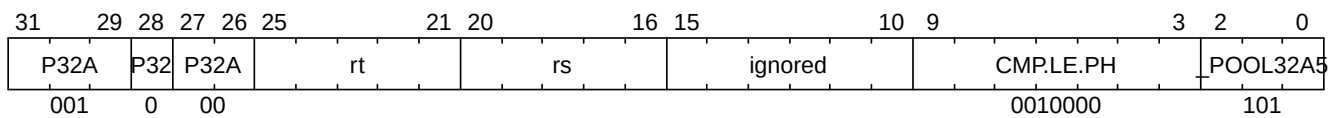


Figure 325. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

8.124.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.DSPControl.Ccond = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, "le", signed_values=True
)
```

8.124.4. Description

CMP.LE.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.124.5. Exceptions

None.

8.125. CMP.LT.PH

8.125.1. Purpose

DSP packed operation CMP.LT.PH.

8.125.2. Formats

MIPS

cmp.lt.ph

Assembly

CMP.LT.PH {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

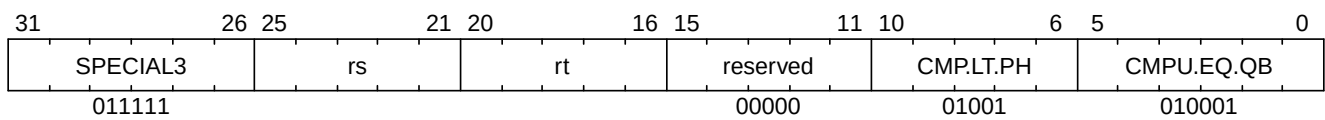


Figure 326. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

microMIPS

cmp.lt.ph.32

Assembly

CMP.LT.PH {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

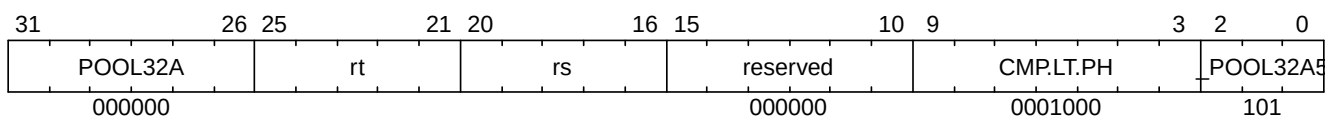


Figure 327. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

nanoMIPS

`cmp.lt.ph.32`

Assembly

`CMP.LT.PH {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

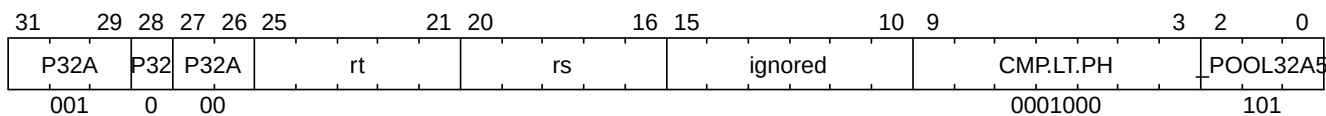


Figure 328. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

8.125.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.DSPControl.Ccond = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, "lt", signed_values=True
)
```

8.125.4. Description

CMP.LT.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.125.5. Exceptions

None.

8.126. CMP.S

8.126.1. Purpose

Compare two S-format values and write a Boolean mask to an FPR.

8.126.2. Formats

MIPS

cmp.s

Assembly

CMP.S {fd:fpr}, {fs:fpr}, {ft:fpr}, {condition:u5}

Availability

AR >= R6

Decode priority

0

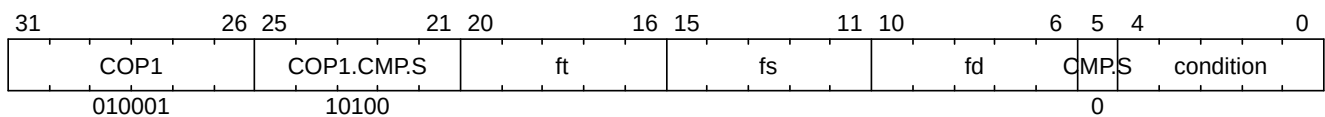


Figure 329. Encoding (32 bits)

Decode

```
return FpuCompareOperands(  
    fd=fd, fs=fs, ft=ft, condition=zero_extend(condition, from_nbits=5)  
)
```

microMIPS

cmp.s

Assembly

CMP.S {fd:fpr}, {fs:fpr}, {ft:fpr}, {condition:u5}

Availability

MMAR >= R6

Decode priority

0

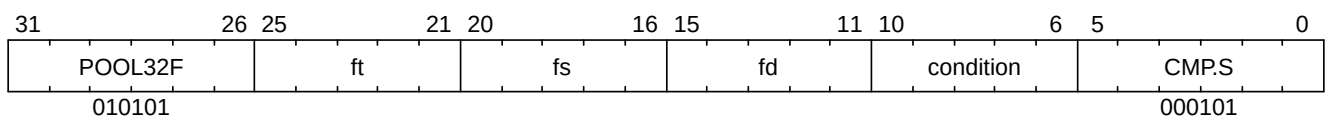


Figure 330. Encoding (32 bits)

Decode

```
return FpuCompareOperands(
    fd=fd, fs=fs, ft=ft, condition=zero_extend(condition, from_nbits=5)
)
```

nanoMIPS

cmp.s

Assembly

CMP.S {fd:fpr}, {fs:fpr}, {ft:fpr}, {condition:u5}

Availability

true

Decode priority

0

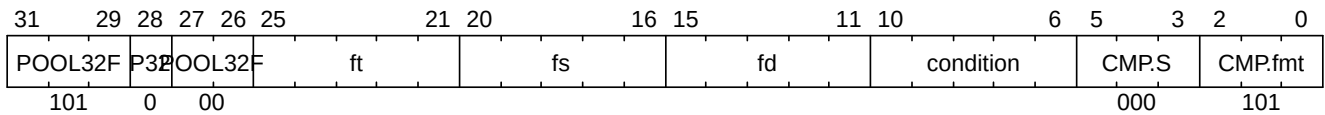


Figure 331. Encoding (32 bits)

Decode

```
return FpuCompareOperands(
    fd=fd, fs=fs, ft=ft, condition=zero_extend(condition, from_nbits=5)
)
```

8.126.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
if operands.condition in {
    0x10,
    0x14,
    0x15,
    0x16,
    0x17,
    0x18,
    0x1C,
    0x1D,
    0x1E,
    0x1F,
}:
    raise ReservedInstruction("floating comparison condition is not implemented")
condition, signaling = interpret_compare_condition(operands.condition)
matched, flags = ieee_compare(
    "S",
```

```

    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    condition,
    signaling=signaling,
)
write_fpu_result(ctx, operands.fd, 0xFFFF_FFFF if matched else 0, flags, "S")

```

8.126.4. Description

CMP.S supports the Release 6 quiet and signaling comparison-condition encodings.

8.126.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; floating comparison condition is not implemented

8.127. CMPGDU.EQ.QB

8.127.1. Purpose

DSP packed operation CMPGDU.EQ.QB.

8.127.2. Formats

MIPS

cmpgdu.eq.qb

Assembly

CMPGDU.EQ.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

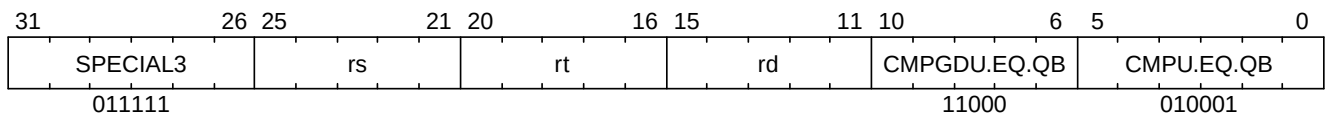


Figure 332. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

cmpgdu.eq.qb.32

Assembly

CMPGDU.EQ.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

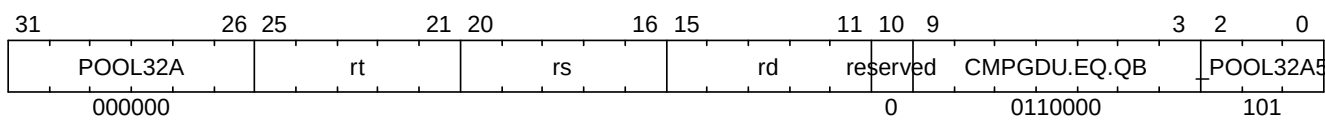


Figure 333. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

cmpgdu.eq.qb.32

Assembly

CMPGDU.EQ.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

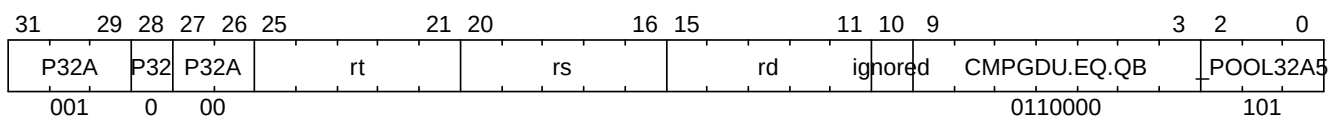


Figure 334. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.127.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value = packed_compare(ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "eq")
ctx.DSPControl.Ccond = value
ctx.GPR[operands.rd] = value
```

8.127.4. Description

CMPGDU.EQ.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.127.5. Exceptions

None.

8.128. CMPGDU.LE.QB

8.128.1. Purpose

DSP packed operation CMPGDU.LE.QB.

8.128.2. Formats

MIPS

cmpgdu.le.qb

Assembly

CMPGDU.LE.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

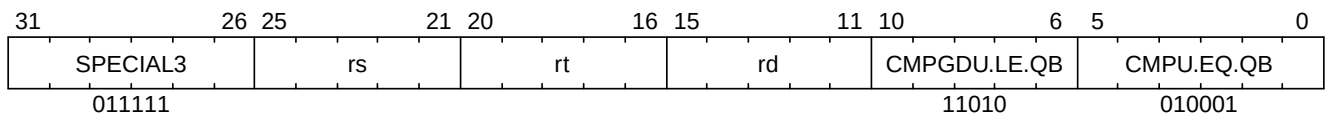


Figure 335. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

cmpgdu.le.qb.32

Assembly

CMPGDU.LE.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

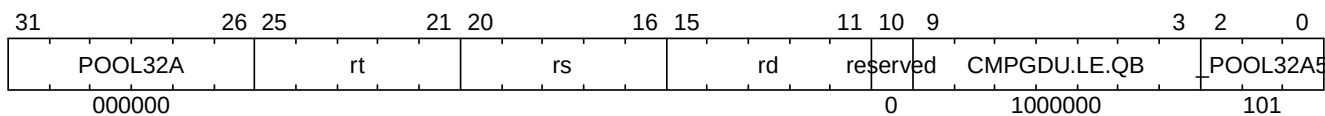


Figure 336. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

cmpgdu.le.qb.32

Assembly

CMPGDU.LE.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

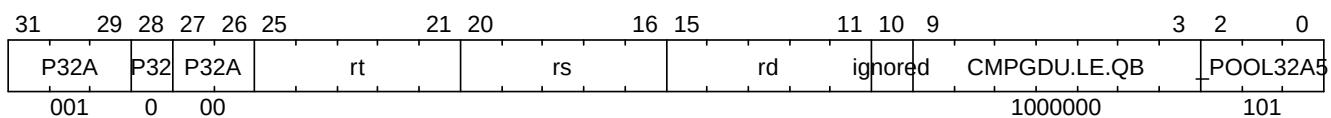


Figure 337. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.128.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value = packed_compare(ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "le")
ctx.DSPControl.Ccond = value
ctx.GPR[operands.rd] = value
```

8.128.4. Description

CMPGDU.LE.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.128.5. Exceptions

None.

8.129. CMPGDU.LT.QB

8.129.1. Purpose

DSP packed operation CMPGDU.LT.QB.

8.129.2. Formats

MIPS

cmpgdu.lt.qb

Assembly

CMPGDU.LT.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

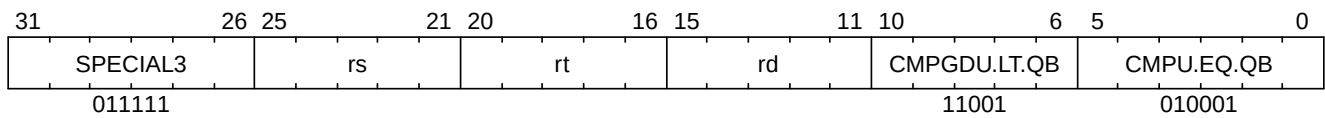


Figure 338. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

cmpgdu.lt.qb.32

Assembly

CMPGDU.LT.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

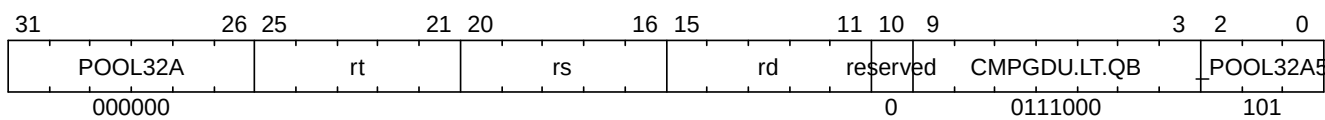


Figure 339. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

cmpgdu.lt.qb.32

Assembly

CMPGDU.LT.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

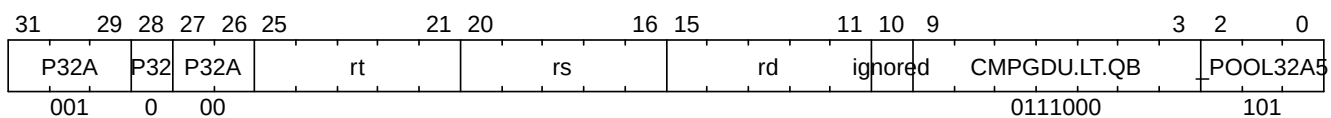


Figure 340. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.129.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value = packed_compare(ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "lt")
ctx.DSPControl.Ccond = value
ctx.GPR[operands.rd] = value
```

8.129.4. Description

CMPGDU.LT.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.129.5. Exceptions

None.

8.130. CMPGU.EQ.QB

8.130.1. Purpose

DSP packed operation CMPGU.EQ.QB.

8.130.2. Formats

MIPS

cmpgu.eq.qb

Assembly

CMPGU.EQ.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

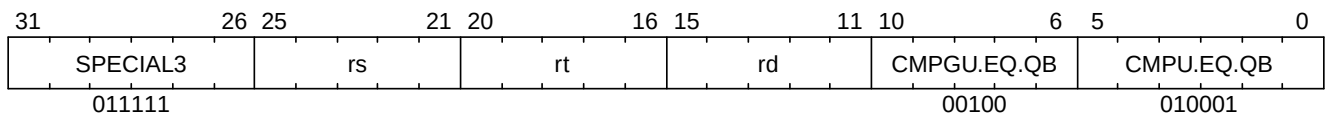


Figure 341. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

cmpgu.eq.qb.32

Assembly

CMPGU.EQ.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

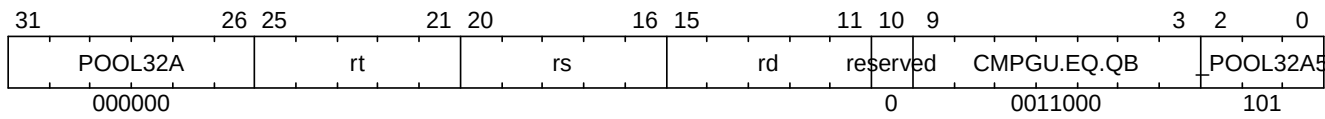


Figure 342. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

cmpgu.eq.qb.32

Assembly

CMPGU.EQ.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

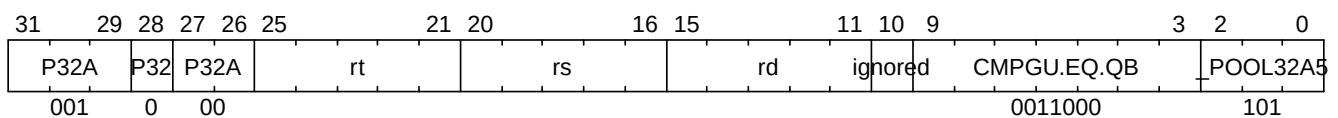


Figure 343. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.130.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.GPR[operands.rd] = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "eq"
)
```

8.130.4. Description

CMPGU.EQ.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.130.5. Exceptions

None.

8.131. CMPGU.LE.QB

8.131.1. Purpose

DSP packed operation CMPGU.LE.QB.

8.131.2. Formats

MIPS

`cmpgu.le.qb`

Assembly

`CMPGU.LE.QB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

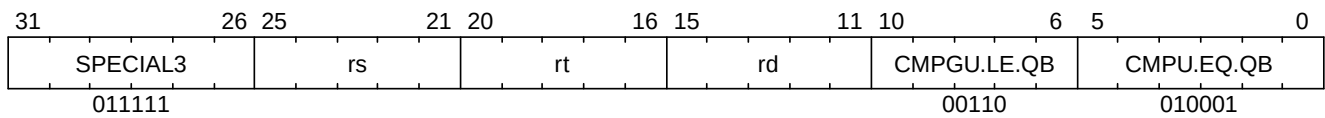


Figure 344. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`cmpgu.le.qb.32`

Assembly

`CMPGU.LE.QB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

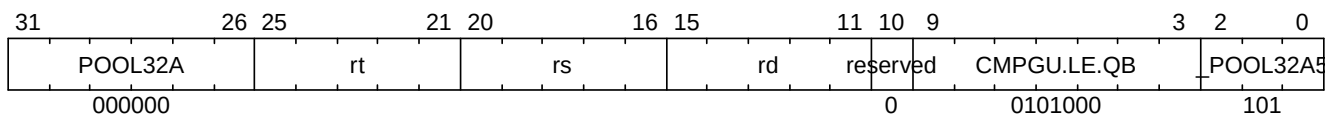


Figure 345. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`cmpgu.le.qb.32`

Assembly

`CMPGU.LE.QB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

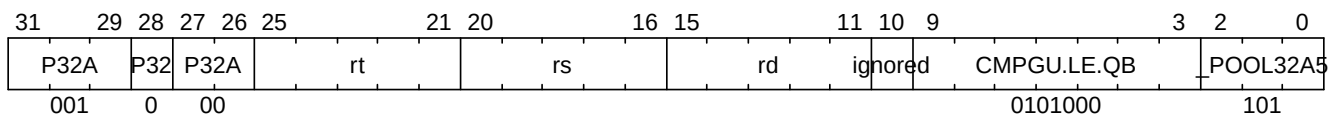


Figure 346. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.131.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.GPR[operands.rd] = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "le"
)
```

8.131.4. Description

CMPGU.LE.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.131.5. Exceptions

None.

8.132. CMPGU.LT.QB

8.132.1. Purpose

DSP packed operation CMPGU.LT.QB.

8.132.2. Formats

MIPS

cmpgu.lt.qb

Assembly

CMPGU.LT.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

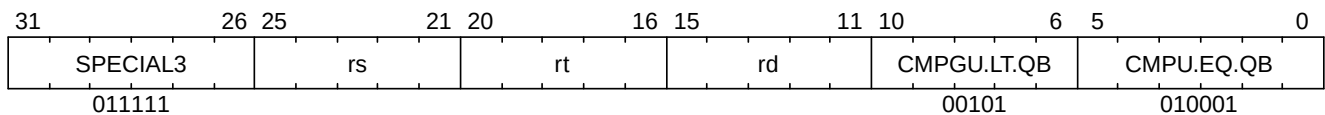


Figure 347. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

cmpgu.lt.qb.32

Assembly

CMPGU.LT.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

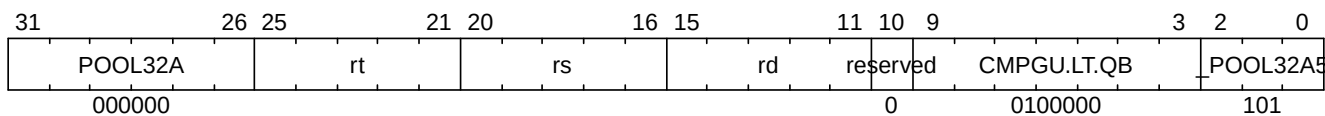


Figure 348. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

cmpgu.lt.qb.32

Assembly

CMPGU.LT.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

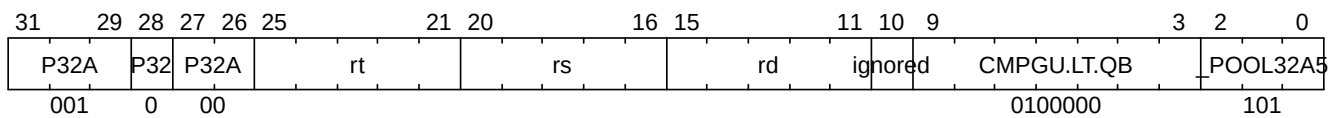


Figure 349. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.132.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.GPR[operands.rd] = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "lt"
)
```

8.132.4. Description

CMPGU.LT.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.132.5. Exceptions

None.

8.133. CMPU.EQ.QB

8.133.1. Purpose

DSP packed operation CMPU.EQ.QB.

8.133.2. Formats

MIPS

`cmпу.eq.qb`

Assembly

`CMPU.EQ.QB {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

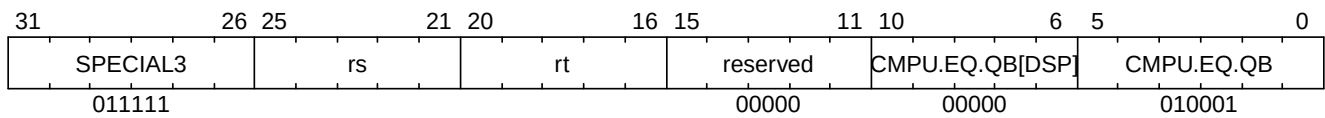


Figure 350. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

microMIPS

`cmпу.eq.qb.32`

Assembly

`CMPU.EQ.QB {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

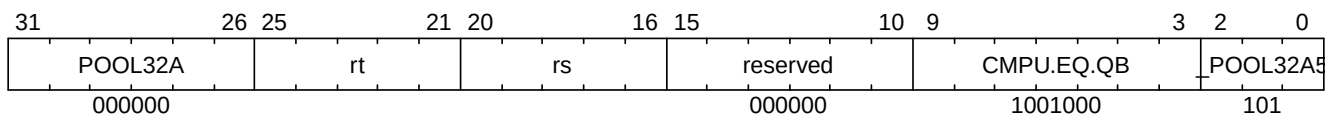


Figure 351. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

nanoMIPS

cmputeq.qb.32

Assembly

CMPU.EQ.QB {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

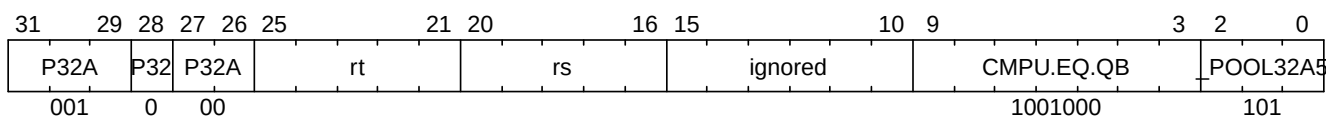


Figure 352. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

8.133.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.DSPControl.Ccond = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "eq"
)
```

8.133.4. Description

CMPU.EQ.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.133.5. Exceptions

None.

8.134. CMPU.LE.QB

8.134.1. Purpose

DSP packed operation CMPU.LE.QB.

8.134.2. Formats

MIPS

`cmпу.1e.qb`

Assembly

`CMPU.LE.QB {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

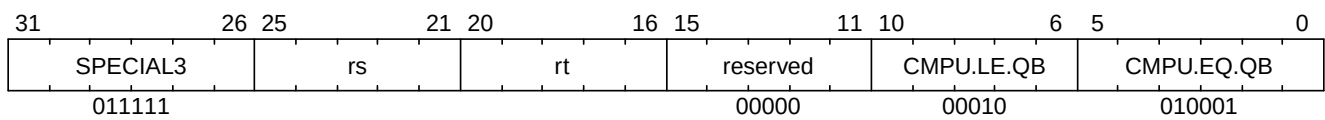


Figure 353. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

microMIPS

`cmпу.1e.qb.32`

Assembly

`CMPU.LE.QB {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

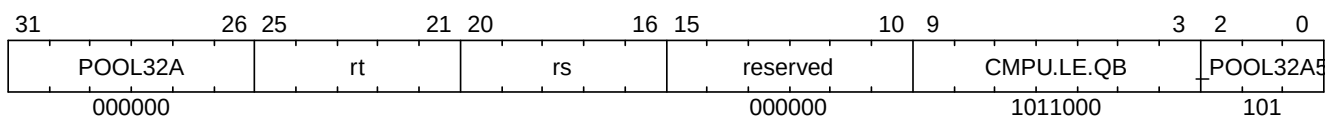


Figure 354. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

nanoMIPS

`cmput.le.qb.32`

Assembly

`CMPU.LE.QB {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

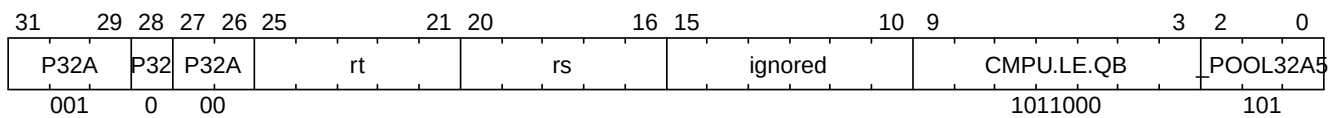


Figure 355. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

8.134.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.DSPControl.Ccond = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "le"
)
```

8.134.4. Description

CMPU.LE.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.134.5. Exceptions

None.

8.135. CMPU.LT.QB

8.135.1. Purpose

DSP packed operation CMPU.LT.QB.

8.135.2. Formats

MIPS

`cmpu.lt.qb`

Assembly

`CMPU.LT.QB {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

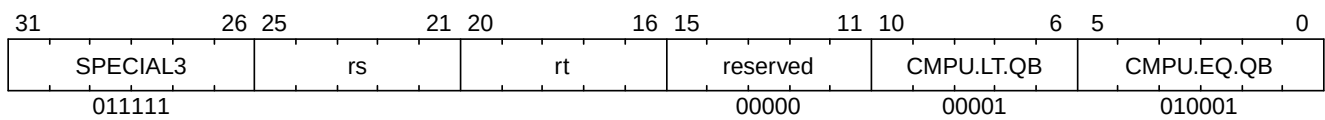


Figure 356. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

microMIPS

`cmpu.lt.qb.32`

Assembly

`CMPU.LT.QB {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

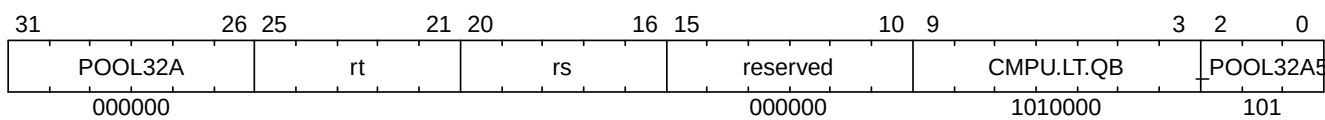


Figure 357. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

nanoMIPS

cmpu.lt.qb.32

Assembly

CMPU.LT.QB {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

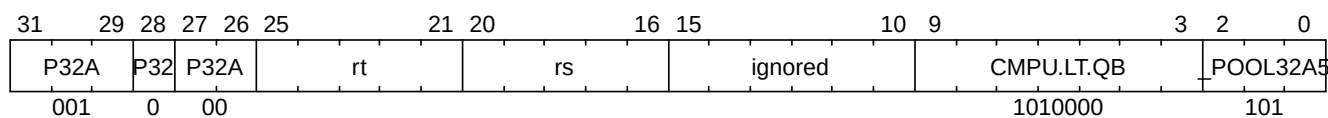


Figure 358. Encoding (32 bits)

Decode

```
return DspCompareOperands(rs=rs, rt=rt)
```

8.135.3. Operation

```
validate_access_to_dsp_resources(ctx)
ctx.DSPControl.Ccond = packed_compare(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, "lt"
)
```

8.135.4. Description

CMPU.LT.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.135.5. Exceptions

None.

implemented

8.137. CRC32B

8.137.1. Purpose

Update a cumulative IEEE CRC-32 value with the low byte of `rs`.

8.137.2. Formats

nanoMIPS

`crc32b.32`

Assembly

`CRC32B {rt:gpr}, {rs:gpr}`

Availability

CRCP

Decode priority

0

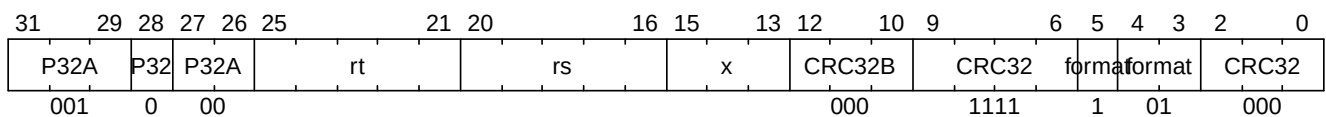


Figure 360. Encoding (32 bits)

Decode

```
del ctx, exec
return Crc32Operands(rt=rt, rs=rs)
```

8.137.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
message = int(zero_extend(ctx.GPR[operands.rs], from_nbits=8))
ctx.GPR[operands.rt] = sign_extend(
    crc32(value, message, 8, 0xEDB8_8320), from_nbits=32
)
```

8.137.4. Description

CRC32B[32] XORs the low byte of `rs` into the low 32-bit cumulative CRC in `rt`, then applies the reflected IEEE polynomial 0xEDB88320. The 32-bit result is sign-extended to the implemented GPR width.

8.137.5. Exceptions

- **RI** (Reserved Instruction) — CRC32B requires CP0 Config5.CRCP; CRC32 instructions are reserved when CP0 Config5.CRCP is clear.

8.138. CRC32CB

8.138.1. Purpose

Update a cumulative CRC-32C value with the low byte of `rs`.

8.138.2. Formats

nanoMIPS

`crc32cb.32`

Assembly

`CRC32CB {rt:gpr}, {rs:gpr}`

Availability

CRCP

Decode priority

0

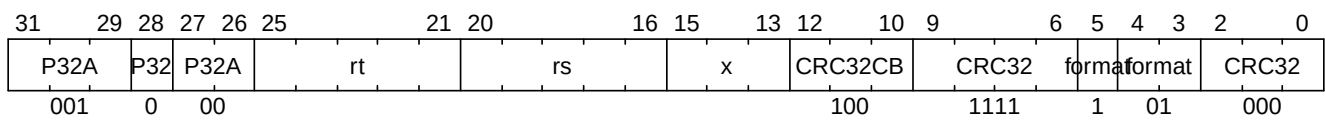


Figure 361. Encoding (32 bits)

Decode

```
del ctx, exec
return Crc32Operands(rt=rt, rs=rs)
```

8.138.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
message = int(zero_extend(ctx.GPR[operands.rs], from_nbits=8))
ctx.GPR[operands.rt] = sign_extend(
    crc32(value, message, 8, 0x82F6_3B78), from_nbits=32
)
```

8.138.4. Description

CRC32CB[32] XORs the low byte of `rs` into the low 32-bit cumulative CRC in `rt`, then applies the reflected Castagnoli polynomial 0x82F63B78. The result is sign-extended from 32 bits.

8.138.5. Exceptions

- **RI** (Reserved Instruction) — CRC32CB requires CP0 Config5.CRCP; CRC32 instructions are reserved when CP0 Config5.CRCP is clear.

8.139. CRC32CD

8.139.1. Purpose

Update a Castagnoli CRC-32C value with a 64-bit message.

8.139.2. Formats

MIPS

`crc32cd`

Assembly

`CRC32CD {rs:gpr}`

Availability

AR >= R6 && AT >= MIPS64_32 && CRCP

Decode priority

0

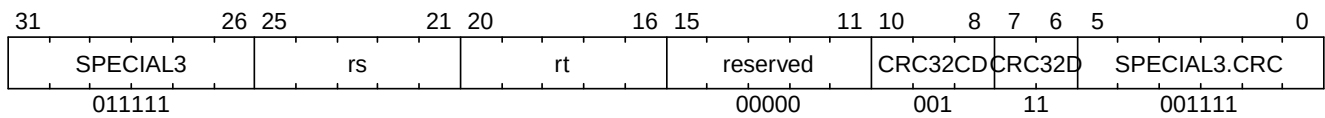


Figure 362. Encoding (32 bits)

Decode

```
return DoubleCrc320perands(rt=rt, rs=rs)
```

nanoMIPS

`crc32cd.32`

Assembly

`CRC32CD {rt:gpr}, {rs:gpr}`

Availability

AT >= MIPS64_32 && CRCP

Decode priority

0

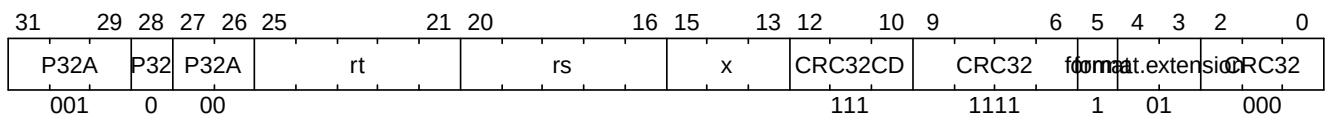


Figure 363. Encoding (32 bits)

Decode

```
return DoubleCrc32Operands(rt=rt, rs=rs)
```

8.139.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_crc32(ctx, operands, poly=0x82F6_3B78)
```

8.139.4. Description

CRC32CD[32] processes the right-justified doubleword in rs using the cumulative CRC in rt.

8.139.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled; CRC32D instructions are reserved when CP0 Config5.CRCP is clear.

8.140. CRC32CH

8.140.1. Purpose

Update a cumulative CRC-32C value with the low halfword of `rs`.

8.140.2. Formats

nanoMIPS

`crc32ch.32`

Assembly

`CRC32CH {rt:gpr}, {rs:gpr}`

Availability

CRCP

Decode priority

0

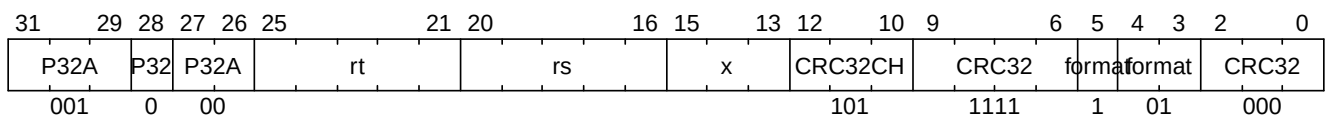


Figure 364. Encoding (32 bits)

Decode

```
del ctx, exec
return Crc32Operands(rt=rt, rs=rs)
```

8.140.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
message = int(zero_extend(ctx.GPR[operands.rs], from_nbits=16))
ctx.GPR[operands.rt] = sign_extend(
    crc32(value, message, 16, 0x82F6_3B78), from_nbits=32
)
```

8.140.4. Description

CRC32CH[32] XORs the low halfword of `rs` into the low 32-bit cumulative CRC in `rt`, then applies the reflected Castagnoli polynomial 0x82F63B78. The result is sign-extended from 32 bits.

8.140.5. Exceptions

- RI** (Reserved Instruction) — CRC32CH requires CP0 Config5.CRCP; CRC32 instructions are reserved when CP0 Config5.CRCP is clear.

8.141. CRC32CW

8.141.1. Purpose

Update a cumulative CRC-32C value with the low word of `rs`.

8.141.2. Formats

nanoMIPS

`crc32cw.32`

Assembly

`CRC32CW {rt:gpr}, {rs:gpr}`

Availability

CRCP

Decode priority

0

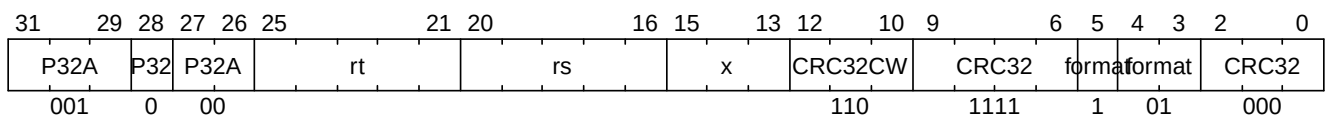


Figure 365. Encoding (32 bits)

Decode

```
del ctx, exec
return Crc32Operands(rt=rt, rs=rs)
```

8.141.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
message = int(zero_extend(ctx.GPR[operands.rs], from_nbits=32))
ctx.GPR[operands.rt] = sign_extend(
    crc32(value, message, 32, 0x82F6_3B78), from_nbits=32
)
```

8.141.4. Description

`CRC32CW[32]` XORs the low word of `rs` into the low 32-bit cumulative CRC in `rt`, then applies the reflected Castagnoli polynomial `0x82F63B78`. The result is sign-extended from 32 bits.

8.141.5. Exceptions

- **RI** (Reserved Instruction) — `CRC32CW` requires `CP0 Config5.CRCP`; CRC32 instructions are reserved when `CP0 Config5.CRCP` is clear.

8.142. CRC32D

8.142.1. Purpose

Update an IEEE CRC-32 value with a 64-bit message.

8.142.2. Formats

MIPS

`crc32d`

Assembly

`CRC32D {rs:gpr}`

Availability

AR >= R6 && AT >= MIPS64_32 && CRCP

Decode priority

0

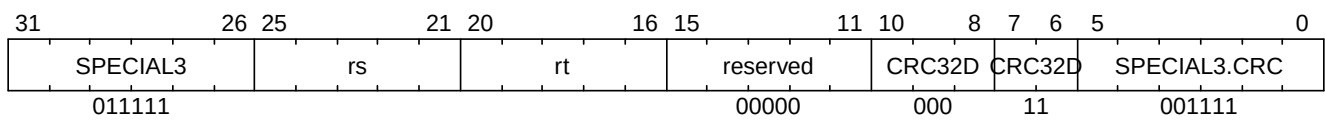


Figure 366. Encoding (32 bits)

Decode

```
return DoubleCrc32operands(rt=rt, rs=rs)
```

nanoMIPS

`crc32d.32`

Assembly

`CRC32D {rt:gpr}, {rs:gpr}`

Availability

AT >= MIPS64_32 && CRCP

Decode priority

0

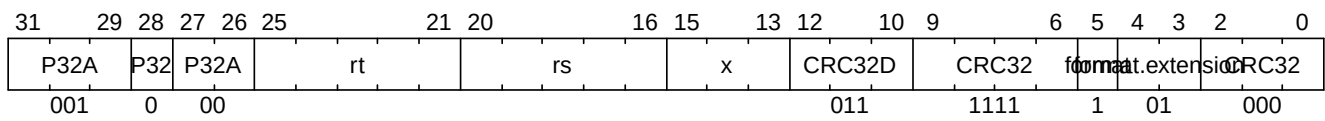


Figure 367. Encoding (32 bits)

Decode

```
return DoubleCrc32Operands(rt=rt, rs=rs)
```

8.142.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_crc32(ctx, operands, poly=0xEDB8_8320)
```

8.142.4. Description

CRC32D[32] processes the right-justified doubleword in rs using the cumulative CRC in rt.

8.142.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled; CRC32D instructions are reserved when CP0 Config5.CRCP is clear.

8.143. CRC32H

8.143.1. Purpose

Update a cumulative IEEE CRC-32 value with the low halfword of `rs`.

8.143.2. Formats

nanoMIPS

`crc32h.32`

Assembly

`CRC32H {rt:gpr}, {rs:gpr}`

Availability

CRCP

Decode priority

0

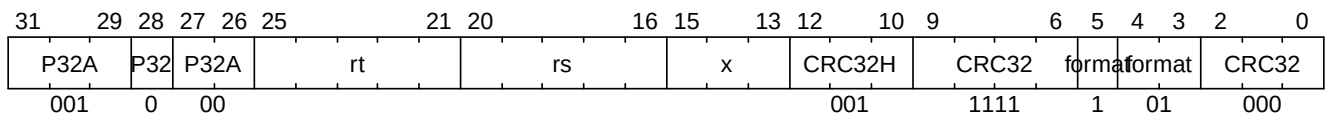


Figure 368. Encoding (32 bits)

Decode

```
del ctx, exec
return Crc32Operands(rt=rt, rs=rs)
```

8.143.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
message = int(zero_extend(ctx.GPR[operands.rs], from_nbits=16))
ctx.GPR[operands.rt] = sign_extend(
    crc32(value, message, 16, 0xEDB8_8320), from_nbits=32
)
```

8.143.4. Description

CRC32H[32] XORs the low halfword of `rs` into the low 32-bit cumulative CRC in `rt`, then applies the reflected IEEE polynomial 0xEDB88320. The result is sign-extended from 32 bits.

8.143.5. Exceptions

- **RI** (Reserved Instruction) — CRC32H requires CP0 Config5.CRCP; CRC32 instructions are reserved when CP0 Config5.CRCP is clear.

8.144. CRC32W

8.144.1. Purpose

Update a cumulative IEEE CRC-32 value with the low word of `rs`.

8.144.2. Formats

nanoMIPS

`crc32w.32`

Assembly

`CRC32W {rt:gpr}, {rs:gpr}`

Availability

CRCP

Decode priority

0

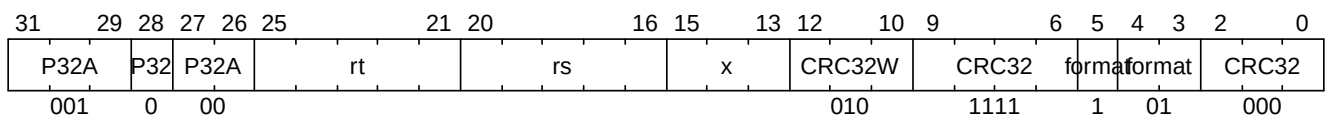


Figure 369. Encoding (32 bits)

Decode

```
del ctx, exec
return Crc32Operands(rt=rt, rs=rs)
```

8.144.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
message = int(zero_extend(ctx.GPR[operands.rs], from_nbits=32))
ctx.GPR[operands.rt] = sign_extend(
    crc32(value, message, 32, 0xEDB8_8320), from_nbits=32
)
```

8.144.4. Description

CRC32W[32] XORs the low word of `rs` into the low 32-bit cumulative CRC in `rt`, then applies the reflected IEEE polynomial 0xEDB88320. The result is sign-extended from 32 bits.

8.144.5. Exceptions

- **RI** (Reserved Instruction) — CRC32W requires CP0 Config5.CRCP; CRC32 instructions are reserved when CP0 Config5.CRCP is clear.

8.145. CTC1

8.145.1. Purpose

Transfer a value through Coprocessor 1.

8.145.2. Formats

MIPS

ctc1

Assembly

CTC1 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

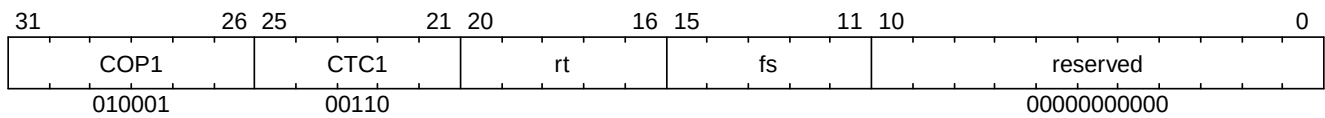


Figure 370. Encoding (32 bits)

Decode

```
return CFC10perands(rt=rt, cs=fs)
```

microMIPS

ctc1.32

Assembly

CTC1 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

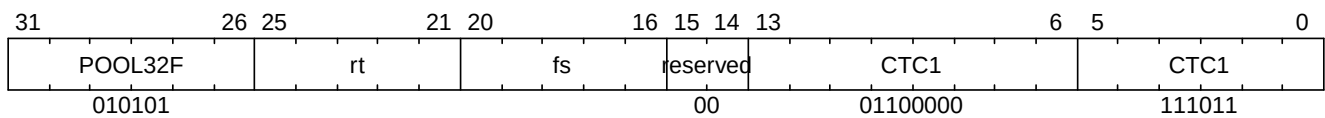


Figure 371. Encoding (32 bits)

Decode

```
return CFC10perands(rt=rt, cs=fs)
```

nanoMIPS

ctc1.32

Assembly

CTC1 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

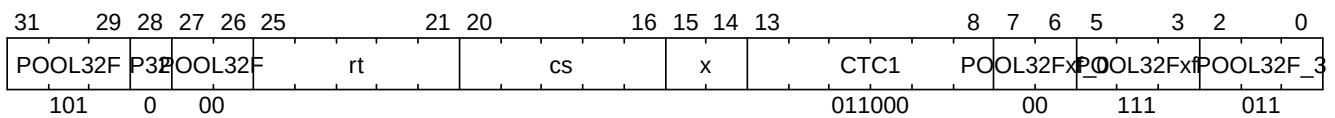


Figure 372. Encoding (32 bits)

Decode

```
return CFC10perands(rt=rt, cs=cs)
```

8.145.3. Operation

```
validate_access_to_cp1_resources(ctx)
write_cp1_control(
    ctx,
    operands.cs,
    ctx.GPR[operands.rt],
    source_register=operands.rt,
)
```

8.145.4. Description

CTC1 performs its CP1 access checks before transferring data.

8.145.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit

operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; C1 mode aliases require GPR0 as the CTC1 source; The instruction or selected behavior is reserved in the current state.; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.

8.146. CTC2

8.146.1. Purpose

Move a GPR to a CP2 control register.

8.146.2. Formats

microMIPS

ctc2.32

Assembly

CTC2 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

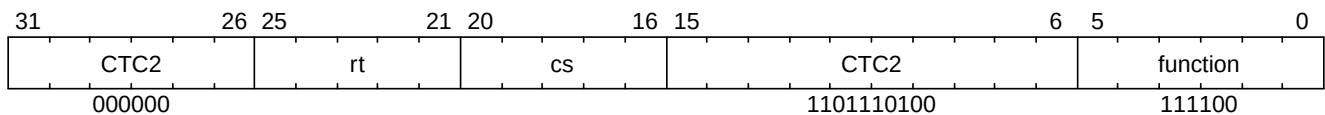


Figure 373. Encoding (32 bits)

Decode

```
return Cop2MoveOperands(rt=rt, cs=cs)
```

8.146.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("CTC2 execution is not implemented")
```

8.146.4. Description

CTC2 checks CP2 availability before accessing the register.

8.146.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; CTC2 execution is not

implemented

8.147. CVT.D.L

8.147.1. Purpose

Convert a L-format value to D format.

8.147.2. Formats

MIPS

`cvt.d.l`

Assembly

`CVT.D.L {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

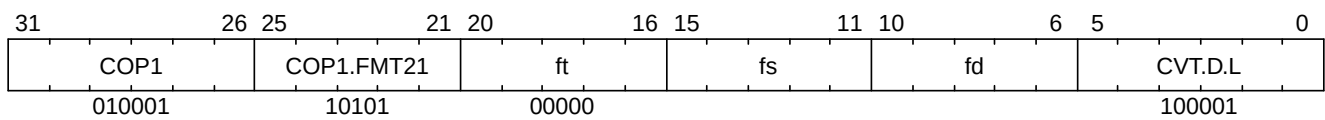


Figure 374. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.d.l`

Assembly

`CVT.D.L {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

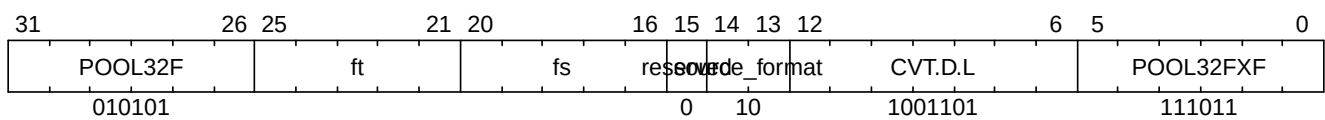


Figure 375. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.d.l

Assembly

CVT.D.L {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

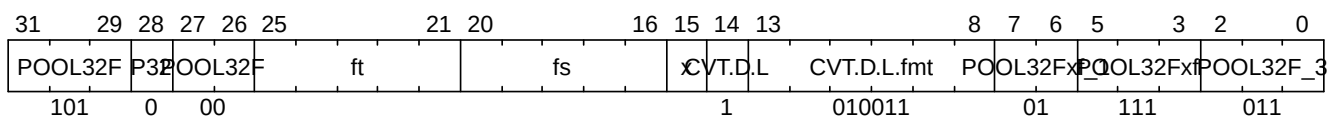


Figure 376. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.147.3. Operation

```
validate_access_to_cp1_resources(ctx, "D", "L")
result, flags = ieee_int_to_float(
    "D", "L", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "L")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.147.4. Description

CVT.D.L uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.147.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.148. CVT.D.S

8.148.1. Purpose

Convert a S-format value to D format.

8.148.2. Formats

MIPS

cvt.d.s

Assembly

CVT.D.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

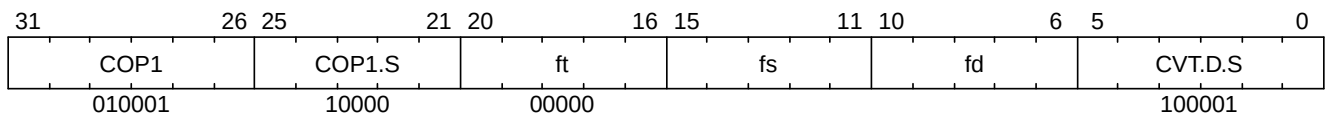


Figure 377. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

cvt.d.s

Assembly

CVT.D.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

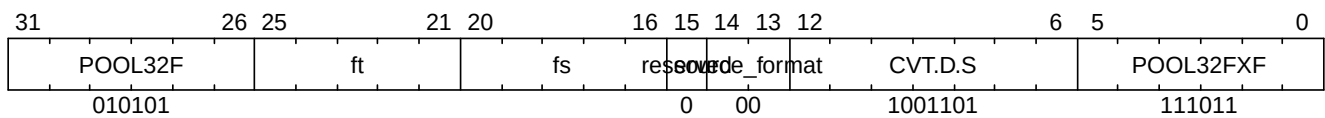


Figure 378. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.d.s

Assembly

CVT.D.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

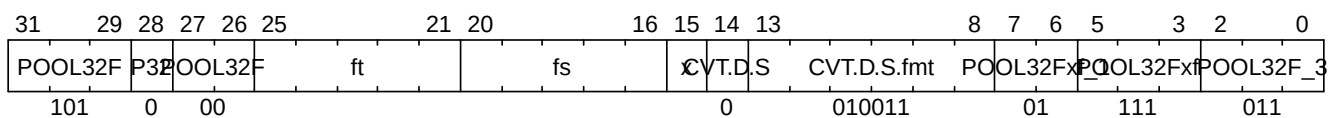


Figure 379. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.148.3. Operation

```
validate_access_to_cp1_resources(ctx, "D", "S")
result, flags = ieee_float_to_float(
    "D", "S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.148.4. Description

CVT.D.S uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.148.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.149. CVT.D.W

8.149.1. Purpose

Convert a W-format value to D format.

8.149.2. Formats

MIPS

`cvt.d.w`

Assembly

`CVT.D.W {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

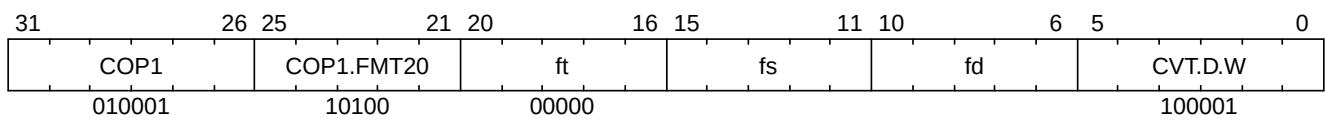


Figure 380. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.d.w`

Assembly

`CVT.D.W {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

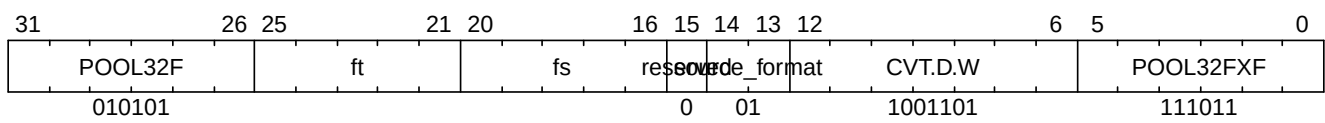


Figure 381. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.d.w

Assembly

CVT.D.W {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

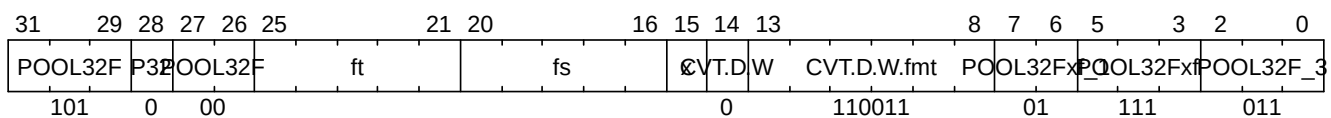


Figure 382. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.149.3. Operation

```
validate_access_to_cp1_resources(ctx, "D", "W")
result, flags = ieee_int_to_float(
    "D", "W", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "W")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.149.4. Description

CVT.D.W uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.149.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.150. CVT.L.D

8.150.1. Purpose

Convert a D-format value to L format.

8.150.2. Formats

MIPS

`cvt.l.d`

Assembly

`CVT.L.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

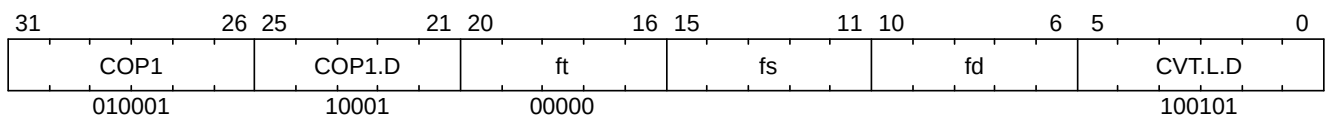


Figure 383. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.l.d`

Assembly

`CVT.L.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

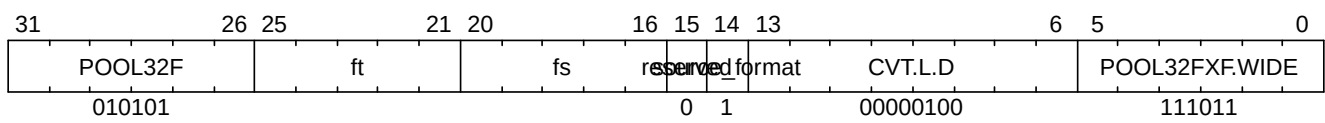


Figure 384. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.l.d

Assembly

CVT.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

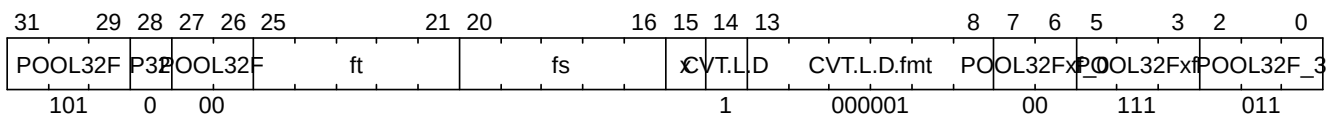


Figure 385. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.150.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "D")
result, flags = ieee_float_to_int(
    "D", "L", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.150.4. Description

CVT.L.D uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.150.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.151. CVT.L.S

8.151.1. Purpose

Convert a S-format value to L format.

8.151.2. Formats

MIPS

`cvt.l.s`

Assembly

`CVT.L.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

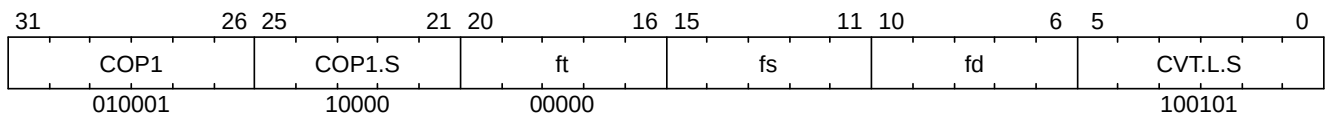


Figure 386. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.l.s`

Assembly

`CVT.L.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

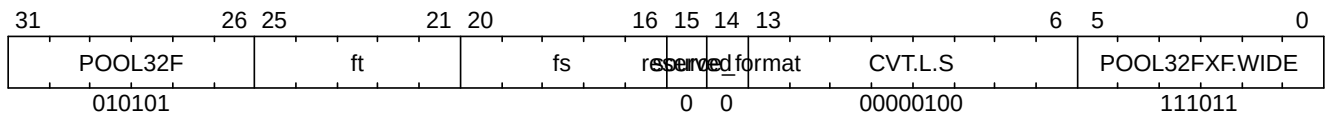


Figure 387. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.l.s

Assembly

CVT.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

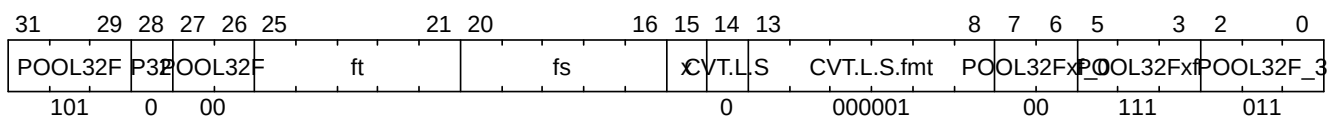


Figure 388. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.151.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "S")
result, flags = ieee_float_to_int(
    "S", "L", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.151.4. Description

CVT.L.S uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.151.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.152. CVT.PS.S

8.152.1. Purpose

Pack two S-format values into one paired-single value.

8.152.2. Formats

MIPS

`cvt.ps.s`

Assembly

`CVT.PS.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR < R6

Decode priority

0

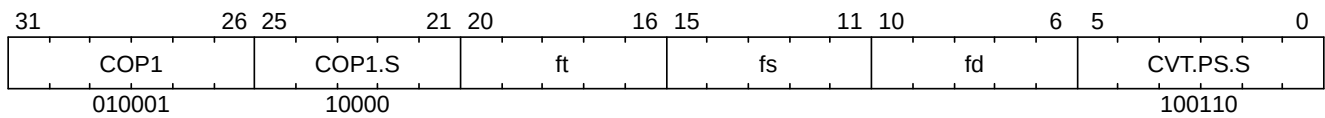


Figure 389. Encoding (32 bits)

Decode

```
return FpuPSPackOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`cvt.ps.s`

Assembly

`CVT.PS.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR < R6

Decode priority

0

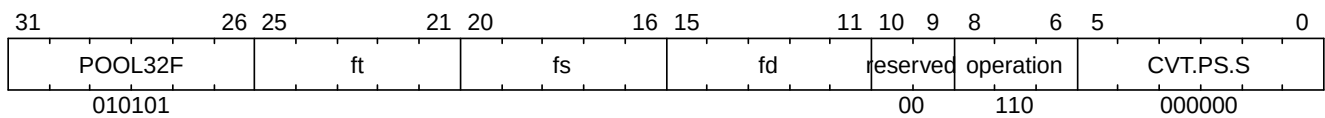


Figure 390. Encoding (32 bits)

```
return FpuPSPackOperands(fd=fd, fs=fs, ft=ft)
```

8.152.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS", "S")
upper = get_fpr(ctx, operands.fs, "S")
lower = get_fpr(ctx, operands.ft, "S")
set_fpr(ctx, operands.fd, lower | (upper << 32), "PS")
```

8.152.4. Description

CVT.PS.S places **fs** in the upper lane and **ft** in the lower lane.

8.152.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.153. CVT.S.D

8.153.1. Purpose

Convert a D-format value to S format.

8.153.2. Formats

MIPS

`cvt.s.d`

Assembly

`CVT.S.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

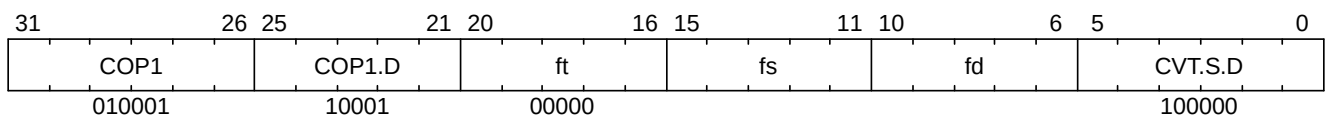


Figure 391. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.s.d`

Assembly

`CVT.S.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

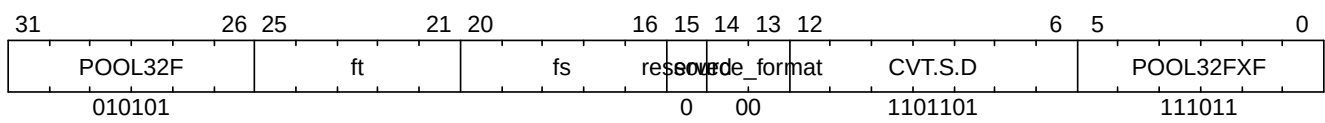


Figure 392. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.s.d

Assembly

CVT.S.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

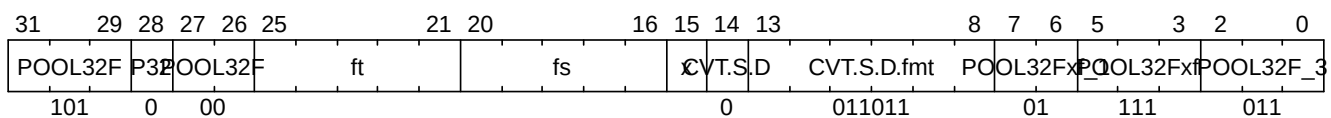


Figure 393. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.153.3. Operation

```
validate_access_to_cp1_resources(ctx, "S", "D")
result, flags = ieee_float_to_float(
    "S", "D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.153.4. Description

CVT.S.D uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.153.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.154. Cvt.S.L

8.154.1. Purpose

Convert a L-format value to S format.

8.154.2. Formats

MIPS

`cvt.s.l`

Assembly

`CVT.S.L {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

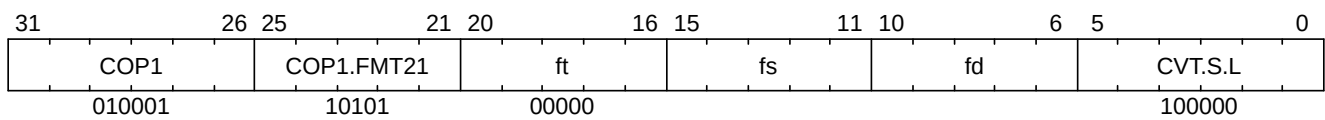


Figure 394. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.s.l`

Assembly

`CVT.S.L {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

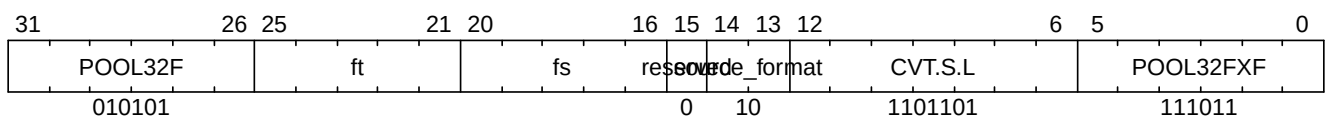


Figure 395. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.s.l

Assembly

CVT.S.L {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

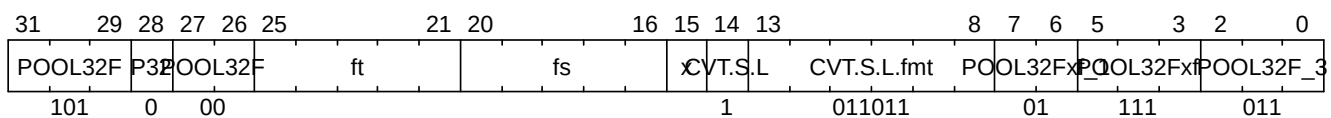


Figure 396. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.154.3. Operation

```
validate_access_to_cp1_resources(ctx, "S", "L")
result, flags = ieee_int_to_float(
    "S", "L", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "L")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.154.4. Description

CVT.S.L uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.154.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.155. CVT.S.PL

8.155.1. Purpose

Extract the lower S-format lane from a paired-single value.

8.155.2. Formats

MIPS

`cvt.s.pl`

Assembly

`CVT.S.PL {fd:fpr}, {fs:fpr}`

Availability

AR < R6

Decode priority

0

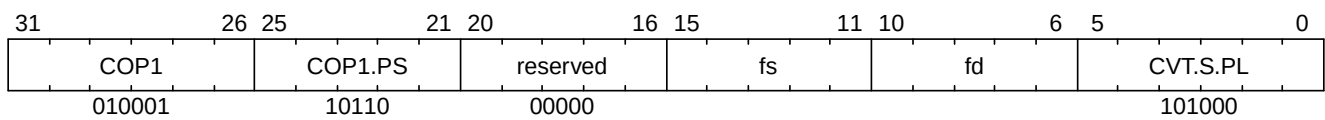


Figure 397. Encoding (32 bits)

Decode

```
return FpuPSExtractOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.s.pl`

Assembly

`CVT.S.PL {fd:fpr}, {fs:fpr}`

Availability

MMAR < R6

Decode priority

0

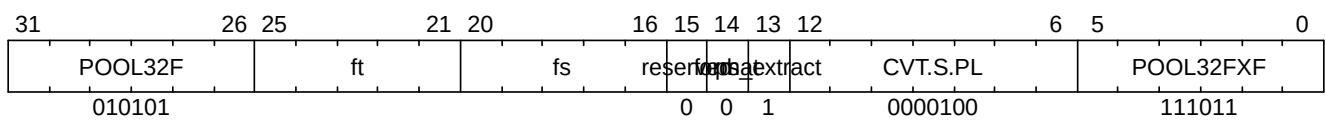


Figure 398. Encoding (32 bits)

```
return FpuPSExtractOperands(fd=ft, fs=fs)
```

8.155.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS", "S")  
set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "PS") & 0xFFFF_FFFF, "S")
```

8.155.4. Description

CVT.S.PL copies the lower 32-bit lane of **fs** to **fd**.

8.155.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.156. CVT.S.PU

8.156.1. Purpose

Extract the upper S-format lane from a paired-single value.

8.156.2. Formats

MIPS

`cvt.s.pu`

Assembly

`CVT.S.PU {fd:fpr}, {fs:fpr}`

Availability

AR < R6

Decode priority

0

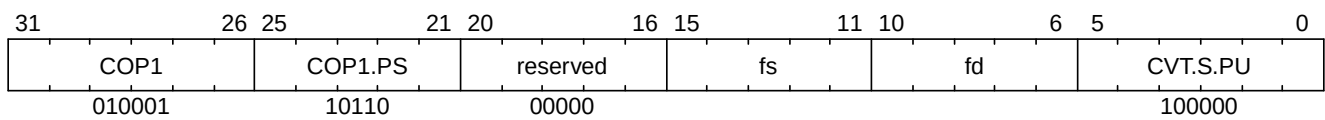


Figure 399. Encoding (32 bits)

Decode

```
return FpuPSExtractOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.s.pu`

Assembly

`CVT.S.PU {fd:fpr}, {fs:fpr}`

Availability

MMAR < R6

Decode priority

0

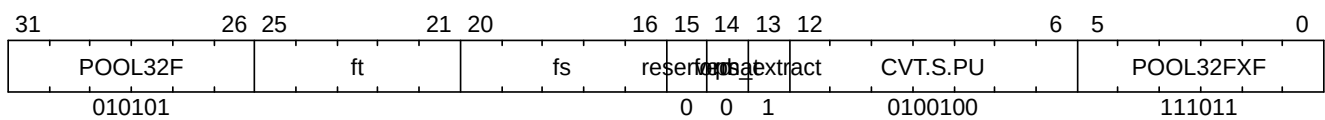


Figure 400. Encoding (32 bits)

```
return FpuPSExtractOperands(fd=ft, fs=fs)
```

8.156.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS", "S")  
set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "PS") >> 32, "S")
```

8.156.4. Description

CVT.S.PU copies the upper 32-bit lane of **fs** to **fd**.

8.156.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.157. CVT.S.W

8.157.1. Purpose

Convert a W-format value to S format.

8.157.2. Formats

MIPS

`cvt.s.w`

Assembly

`CVT.S.W {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

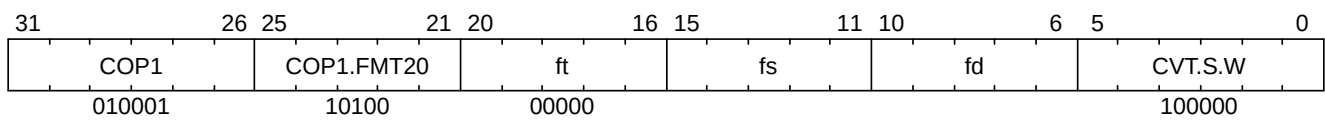


Figure 401. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.s.w`

Assembly

`CVT.S.W {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

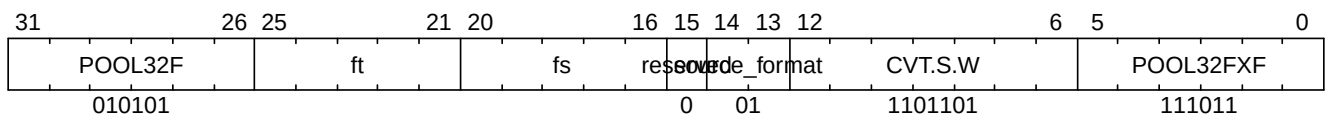


Figure 402. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.s.w

Assembly

CVT.S.W {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

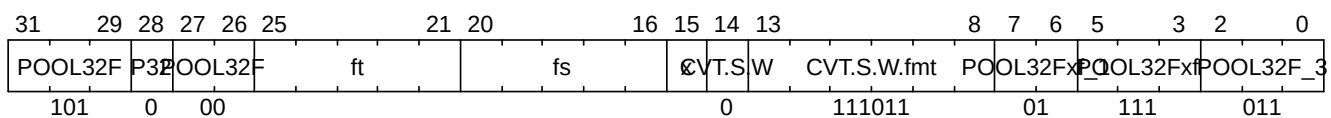


Figure 403. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.157.3. Operation

```
validate_access_to_cp1_resources(ctx, "S", "W")
result, flags = ieee_int_to_float(
    "S", "W", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "W")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.157.4. Description

CVT.S.W uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.157.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.158. CVT.W.D

8.158.1. Purpose

Convert a D-format value to W format.

8.158.2. Formats

MIPS

`cvt.w.d`

Assembly

`CVT.W.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

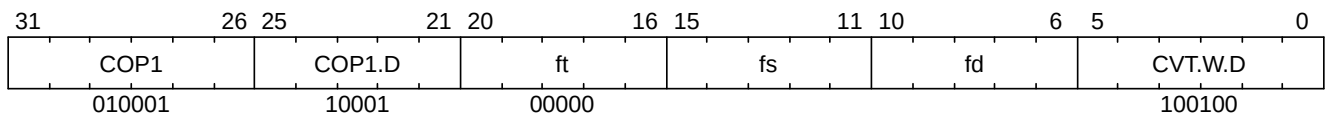


Figure 404. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.w.d`

Assembly

`CVT.W.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

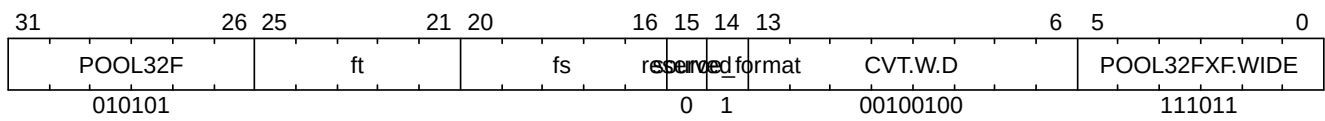


Figure 405. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.w.d

Assembly

CVT.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

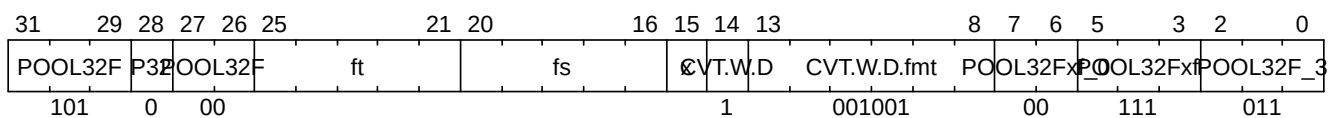


Figure 406. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.158.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "D")
result, flags = ieee_float_to_int(
    "D", "W", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.158.4. Description

CVT.W.D uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.158.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.159. CVT.W.S

8.159.1. Purpose

Convert a S-format value to W format.

8.159.2. Formats

MIPS

`cvt.w.s`

Assembly

`CVT.W.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

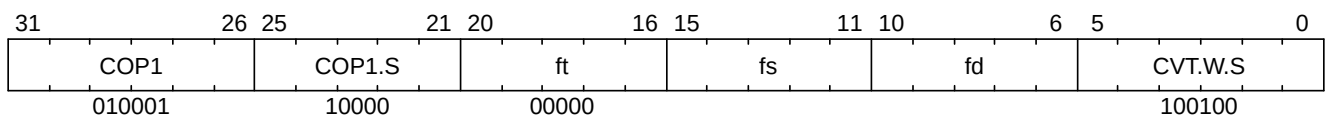


Figure 407. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`cvt.w.s`

Assembly

`CVT.W.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

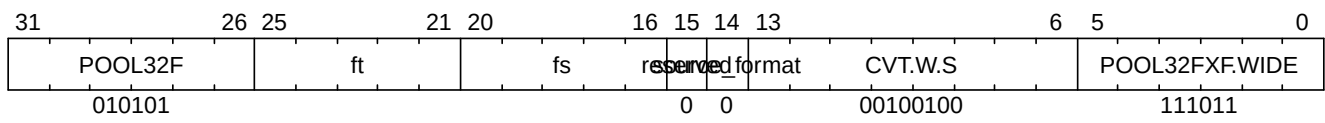


Figure 408. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

cvt.w.s

Assembly

CVT.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

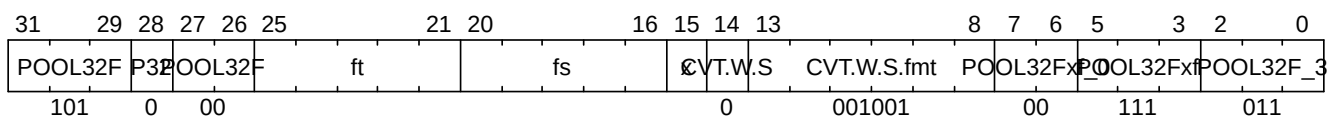


Figure 409. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.159.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "S")
result, flags = ieee_float_to_int(
    "S", "W", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.159.4. Description

CVT.W.S uses the active FCSR rounding mode and records IEEE exceptions before writing the destination.

8.159.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.160. DADD

8.160.1. Purpose

Add two signed 64-bit GPR values and trap on overflow.

8.160.2. Formats

MIPS

dadd

Assembly

DADD {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

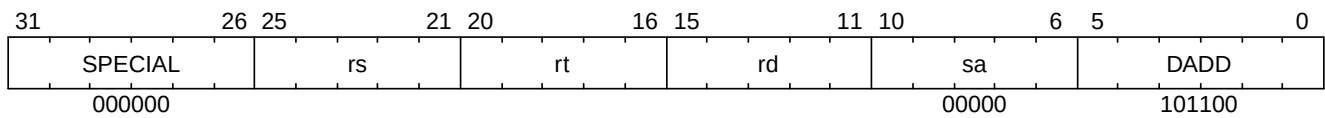


Figure 410. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

dadd

Assembly

DADD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

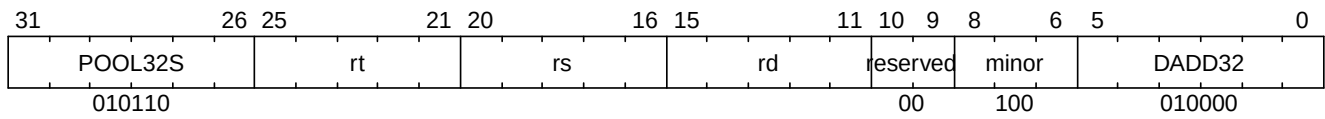


Figure 411. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dadd.32

Assembly

DADD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

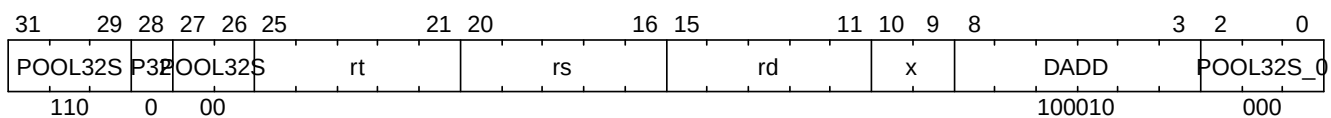


Figure 412. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

8.160.3. Operation

```
validate_access_to_64bit_operations(ctx)
result = double_arithmetic_sum(ctx, operands)
check_signed_doubleword_overflow(result, "DADD")
write_doubleword(ctx, operands.rd, result)
```

8.160.4. Description

The mathematical signed sum is checked for 64-bit overflow before the low 64 bits are written.

8.160.5. Exceptions

- **OV** (Integer Overflow) — A trapping arithmetic result is not representable at its specified width.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.161. DADDI

8.161.1. Purpose

Add a signed immediate to a signed 64-bit GPR and trap on overflow.

8.161.2. Formats

MIPS

`daddi`

Assembly

`DADDI {rd:gpr}, {rs:gpr}, {imm:s16}`

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

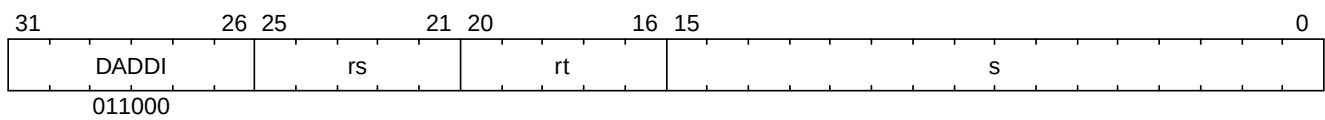


Figure 413. Encoding (32 bits)

Decode

```
return DoubleImmediateOperands(rd=rt, rs=rs, imm=sign_extend(s, from_nbits=16))
```

microMIPS

`daddi`

Assembly

`DADDI {rd:gpr}, {rs:gpr}, {imm:s10}`

Availability

AT >= MIPS64_32 && MMAR < R6

Decode priority

0

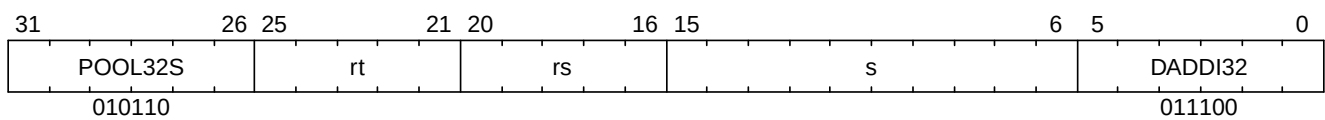


Figure 414. Encoding (32 bits)

```
return DoubleImmediateOperands(rd=rt, rs=rs, imm=sign_extend(s, from_nbits=10))
```

8.161.3. Operation

```
validate_access_to_64bit_operations(ctx)
result = double_arithmetic_sum(ctx, operands)
check_signed_doubleword_overflow(result, "DADDI")
write_doubleword(ctx, operands.rd, result)
```

8.161.4. Description

DADDI is present in MIPS64 before Release 6 and sign-extends its 16-bit immediate.

8.161.5. Exceptions

- **OV** (Integer Overflow) — A trapping arithmetic result is not representable at its specified width.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.162. DADDIU

8.162.1. Purpose

Add an immediate to a 64-bit GPR without trapping on overflow.

8.162.2. Formats

MIPS

`daddiu`

Assembly

`DADDIU {rd:gpr}, {rs:gpr}, {imm:s16}`

Availability

AT >= MIPS64_32

Decode priority

0

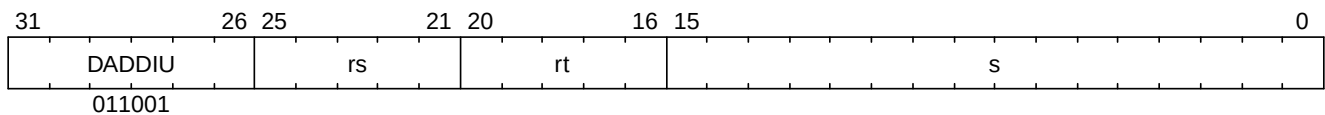


Figure 415. Encoding (32 bits)

Decode

```
return DoubleImmediateOperands(rd=rt, rs=rs, imm=sign_extend(s, from_nbits=16))
```

microMIPS

`daddiu`

Assembly

`DADDIU {rd:gpr}, {rs:gpr}, {imm:s16}`

Availability

AT >= MIPS64_32

Decode priority

0

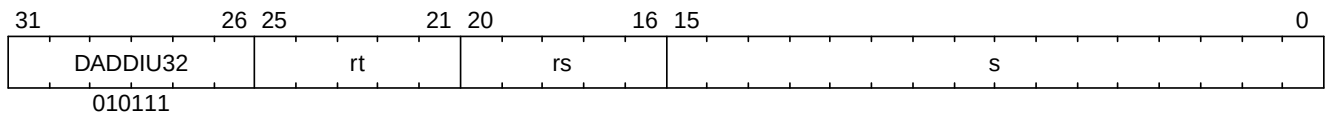


Figure 416. Encoding (32 bits)

Decode

```
return DoubleImmediateOperands(rd=rt, rs=rs, imm=sign_extend(s, from_nbits=16))
```

nanoMIPS

daddiu.48

Assembly

DADDIU {rd:gpr}, {imm:s32}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

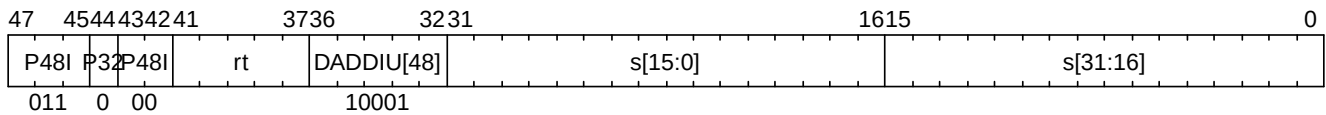


Figure 417. Encoding (48 bits)

Decode

```
return DoubleImmediateOperands(rd=rt, rs=rt, imm=sign_extend(s, from_nbits=32))
```

daddiu.gp_b

Assembly

DADDIU {rd:gpr}, \$gp, {imm:u18}

Availability

AT >= MIPS64_32

Decode priority

0

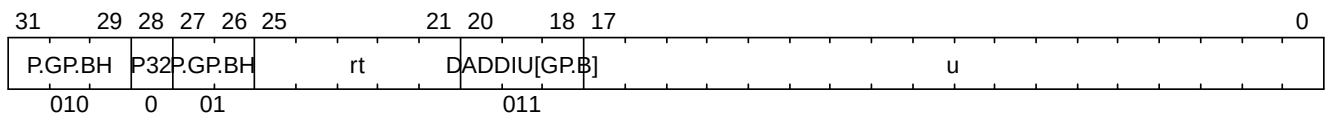


Figure 418. Encoding (32 bits)

Decode

```
if not P64_Mode(ctx):  
    raise behaves_like("addiu.gp_b")  
return DoubleImmediateOperands(rd=rt, rs=28, imm=zero_extend(u, from_nbits=18))
```

daddiu.gp_w

Assembly

DADDIU {rd:gpr}, \$gp, {imm:u21}

Availability

AT >= MIPS64_32

Decode priority

0

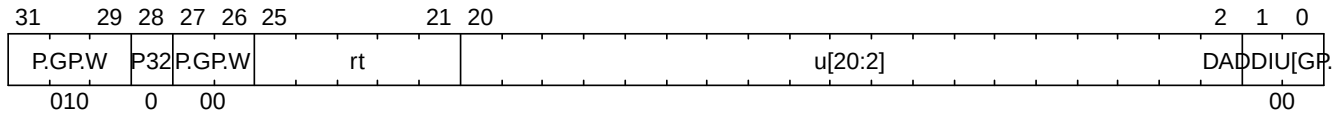


Figure 419. Encoding (32 bits)

Decode

```
if not P64_Mode(ctx):
    raise behaves_like("addiu.gp_w")
return DoubleImmediateOperands(rd=rt, rs=28, imm=zero_extend(u, from_nbits=21))
```

daddiu.gp48

Assembly

DADDIU {rd:gpr}, \$gp, {imm:s32}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

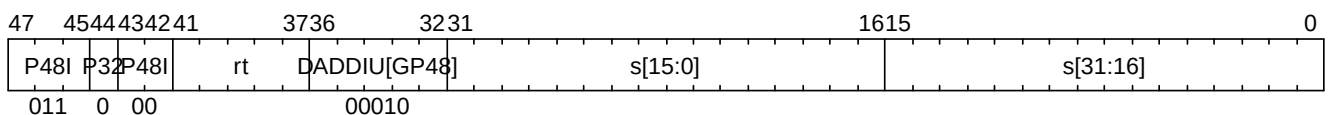


Figure 420. Encoding (48 bits)

Decode

```
if not P64_Mode(ctx):
    raise behaves_like("addiu.gp48")
return DoubleImmediateOperands(rd=rt, rs=28, imm=sign_extend(s, from_nbits=32))
```

daddiu.neg

Assembly

DADDIU {rd:gpr}, {rs:gpr}, -{u:u12}

Availability

AT >= MIPS64_32

Decode priority

0

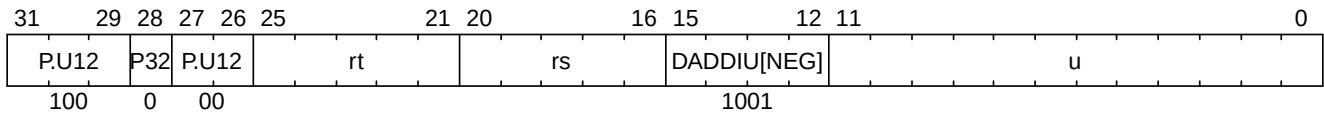


Figure 421. Encoding (32 bits)

Decode

```
return DoubleImmediateOperands(rd=rt, rs=rs, imm=sign_extend(-u, from_nbits=13))
```

daddiu.r1_sp

Assembly

```
DADDIU {rd:gpr3}, $sp, {imm:u8}
```

Availability

AT >= MIPS64_32

Decode priority

0

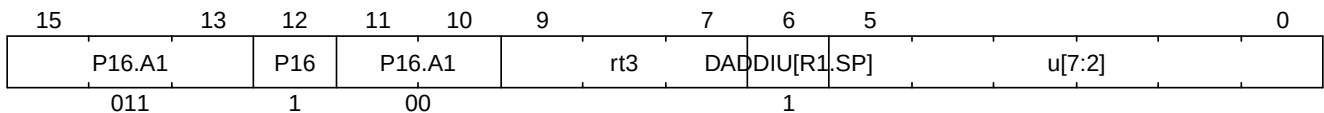


Figure 422. Encoding (16 bits)

Decode

```
if not P64_Mode(ctx):
    raise behaves_like("addiu.r1_sp")
return DoubleImmediateOperands(
    rd=GPR3.decode(rt3), rs=29, imm=zero_extend(u, from_nbits=8)
)
```

daddiu.u12

Assembly

```
DADDIU {rd:gpr}, {rs:gpr}, {imm:u12}
```

Availability

AT >= MIPS64_32

Decode priority

0

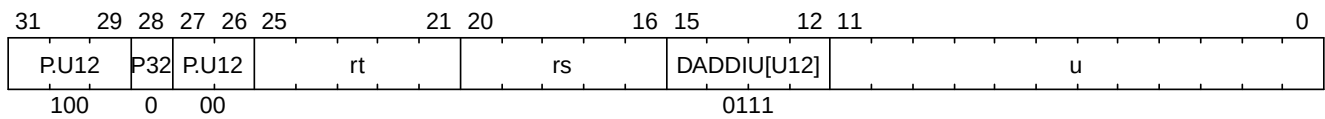


Figure 423. Encoding (32 bits)

Decode

```
return DoubleImmediateOperands(rd=rt, rs=rs, imm=zero_extend(u, from_nbits=12))
```

8.162.3. Operation

```
validate_access_to_64bit_operations(ctx)
write_doubleword(ctx, operands.rd, double_arithmetic_sum(ctx, operands))
```

8.162.4. Description

The result is reduced modulo 2^{64} . GP.B, GP.W, GP48, and R1.SP share encodings with ADDIU and select DADDIU only in P64 mode.

8.162.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.163. DADDIUPC

8.163.1. Purpose

Form a 64-bit PC-relative address in P64 mode.

8.163.2. Formats

nanoMIPS

daddiupc.32

Assembly

DADDIUPC {rt:gpr}, {imm:s22}

Availability

AT >= MIPS64_32

Decode priority

0

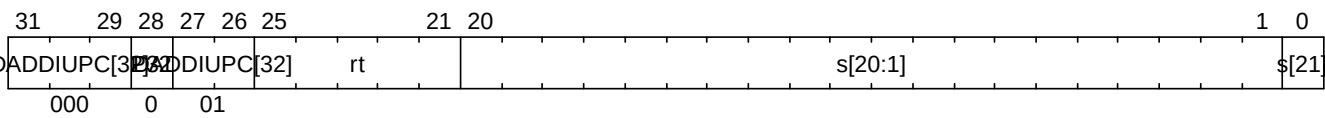


Figure 424. Encoding (32 bits)

Decode

```
if not P64_Mode(ctx):
    raise behaves_like("daddiupc.32")
imm = sign_extend(s, from_nbits=22)
return DAddiupcOperands(rt=rt, imm=imm)
```

daddiupc.48

Assembly

DADDIUPC {rt:gpr}, {imm:s32}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

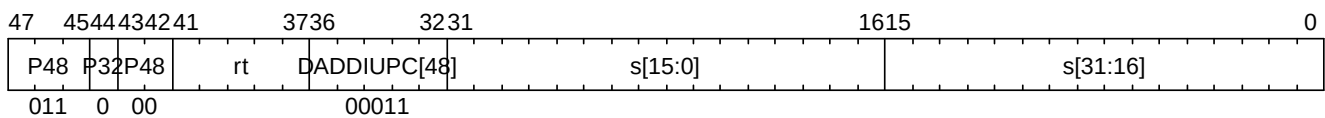


Figure 425. Encoding (48 bits)

```
if not P64_Mode(ctx):  
    raise behaves_like("addiupc.48")  
imm = sign_extend(s, from_nbits=32)  
return DAddiupcOperands(rt=rt, imm=imm)
```

8.163.3. Operation

```
ctx.GPR[operands.rt] = sign_extend(  
    effective_address(ctx, ctx.next_pc, operands.imm), from_nbits=64  
)
```

8.163.4. Description

The encoding is shared with ADDIUPC. In P64 mode the signed displacement is added to `next_pc`, which already denotes the sequential address after the current 32- or 48-bit instruction.

8.163.5. Exceptions

None.

8.164. DADDU

8.164.1. Purpose

Add two 64-bit GPR values without trapping on overflow.

8.164.2. Formats

MIPS

daddu

Assembly

DADDU {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

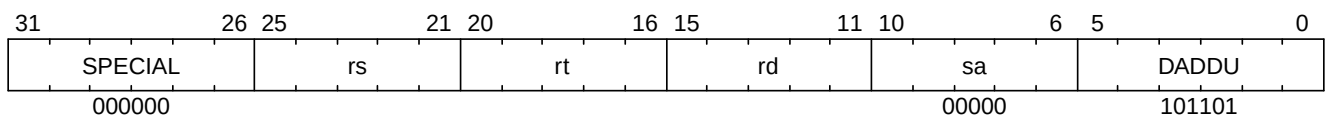


Figure 426. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

daddu

Assembly

DADDU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

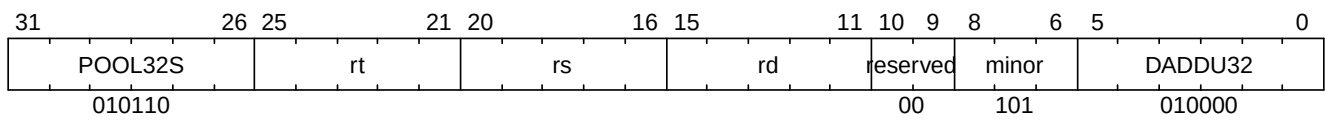


Figure 427. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

daddu.32

Assembly

DADDU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

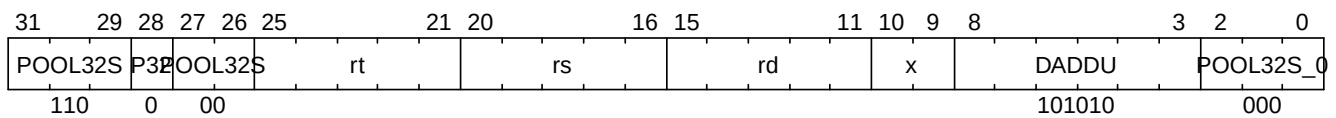


Figure 428. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

8.164.3. Operation

```
validate_access_to_64bit_operations(ctx)
write_doubleword(ctx, operands.rd, double_arithmetic_sum(ctx, operands))
```

8.164.4. Description

The low 64 bits of the sum are written to the destination GPR.

8.164.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.165. DAHI

8.165.1. Purpose

Add a signed upper immediate shifted left by 32 bits to a GPR.

8.165.2. Formats

MIPS

dahi

Assembly

DAHI {imm:s16}

Availability

AR >= R6 && AT >= MIPS64_32

Decode priority

0

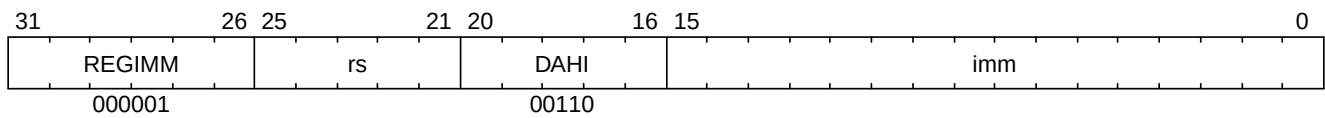


Figure 429. Encoding (32 bits)

Decode

```
imm = sign_extend(imm, from_nbits=16)
return DahiOperands(rs=rs, imm=imm)
```

microMIPS

dahi.32.micromips

Assembly

DAHI {rs:gpr}, {imm:s16}

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

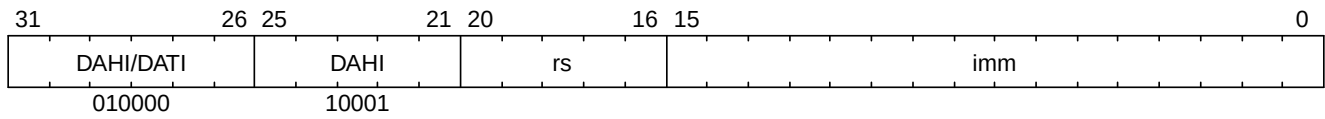


Figure 430. Encoding (32 bits)

Decode

```
imm = sign_extend(imm, from_nbits=16)
return DahiOperands(rs=rs, imm=imm)
```

8.165.3. Operation

```
validate_access_to_64bit_operations(ctx)
ctx.GPR[operands.rs] = double_upper_add(ctx, operands.rs, operands.imm, 32)
```

8.165.4. Description

DAHI adds sign-extended imm16 << 32 to GPR rs and writes the low 64-bit result back to rs.

8.165.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.166. DALIGN

8.166.1. Purpose

Extract a doubleword at a byte position from a pair of GPRs.

8.166.2. Formats

MIPS

`dalign`

Assembly

`DALIGN {rd:gpr}, {rs:gpr}, {rt:gpr}, {bp:u3}`

Availability

AR >= R6 && AT >= MIPS64_32

Decode priority

0

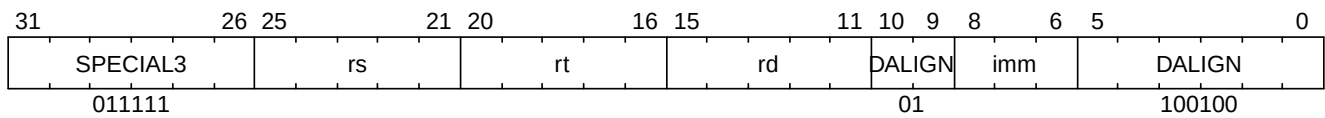


Figure 431. Encoding (32 bits)

Decode

```
return DAlignOperands(rd=rd, rs=rs, rt=rt, bp=zero_extend(imm, from_nbits=3))
```

microMIPS

`dalign.32`

Assembly

`DALIGN {rd:gpr}, {rs:gpr}, {rt:gpr}, {bp:u3}`

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

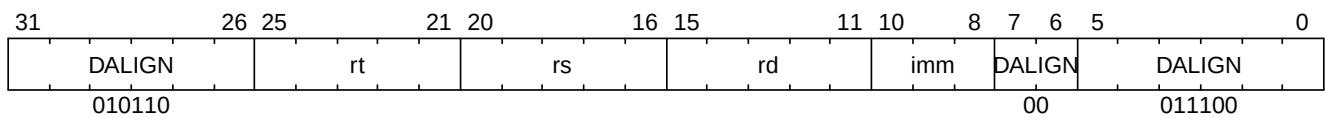


Figure 432. Encoding (32 bits)


```
return DAlignOperands(rd=rd, rs=rs, rt=rt, bp=zero_extend(imm, from_nbits=3))
```

8.166.3. Operation

```
validate_access_to_64bit_operations(ctx)
shift = (8 - int(operands.bp)) * 8
pair = (ctx.GPR[operands.rt] << 64) | ctx.GPR[operands.rs]
ctx.GPR[operands.rd] = sign_extend(pair >> shift, from_nbits=64)
```

8.166.4. Description

DALIGN concatenates rt and rs and selects the 64-bit value beginning at the specified byte position. A byte position of zero returns rt.

8.166.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.167. DATI

8.167.1. Purpose

Add a signed upper immediate shifted left by 48 bits to a GPR.

8.167.2. Formats

MIPS

dati

Assembly

DATI {imm:s16}

Availability

AR >= R6 && AT >= MIPS64_32

Decode priority

0

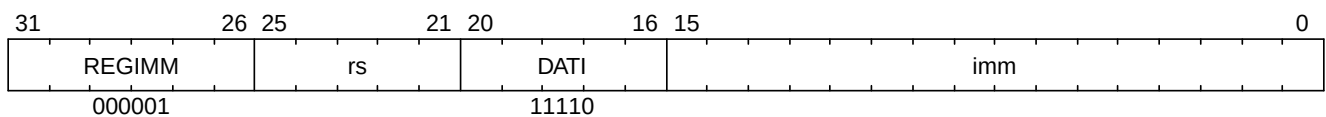


Figure 433. Encoding (32 bits)

Decode

```
imm = sign_extend(imm, from_nbits=16)
return DahiOperands(rs=rs, imm=imm)
```

microMIPS

dati.32.micromips

Assembly

DATI {rs:gpr}, {imm:s16}

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

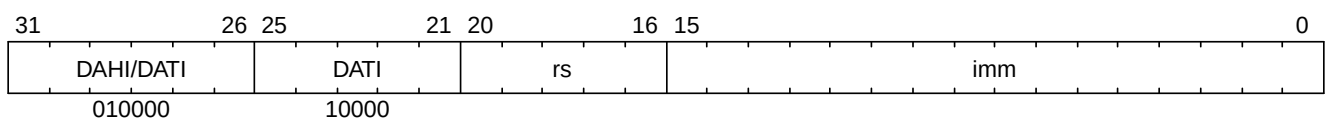


Figure 434. Encoding (32 bits)

Decode

```
imm = sign_extend(imm, from_nbits=16)
return DahiOperands(rs=rs, imm=imm)
```

8.167.3. Operation

```
validate_access_to_64bit_operations(ctx)
ctx.GPR[operands.rs] = double_upper_add(ctx, operands.rs, operands.imm, 48)
```

8.167.4. Description

DATI adds sign-extended imm16 << 48 to GPR rs and writes the low 64-bit result back to rs.

8.167.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.168. DAUI

8.168.1. Purpose

Add a signed upper immediate shifted left by 16 bits to a GPR.

8.168.2. Formats

MIPS

daui

Assembly

DAUI {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

AR >= R6 && AT >= MIPS64_32

Decode priority

0

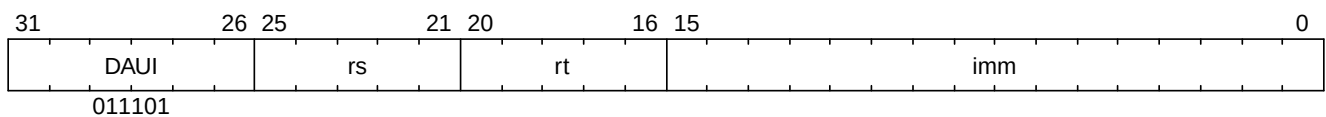


Figure 435. Encoding (32 bits)

Encoding constraints

- **rs** != 0

Decode

```
imm = sign_extend(imm, from_nbits=16)
return DauOperands(rt=rt, rs=rs, imm=imm)
```

microMIPS

daui.32.micromips

Assembly

DAUI {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

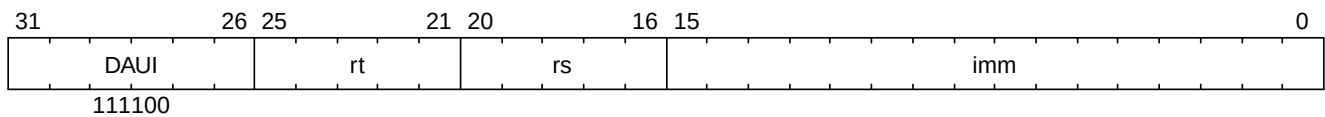


Figure 436. Encoding (32 bits)

Decode

```
imm = sign_extend(imm, from_nbits=16)
return DauIOperands(rt=rt, rs=rs, imm=imm)
```

8.168.3. Operation

```
validate_access_to_64bit_operations(ctx)
ctx.GPR[operands.rt] = double_upper_add(ctx, operands.rs, operands.imm, 16)
```

8.168.4. Description

DAUI adds sign-extended imm16 << 16 from GPR rs and writes the low 64-bit result to GPR rt.

8.168.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.169. DBITSWAP

8.169.1. Purpose

Reverse bits within each byte of a doubleword.

8.169.2. Formats

MIPS

`dbitswap`

Assembly

`DBITSWAP {rs:gpr}`

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

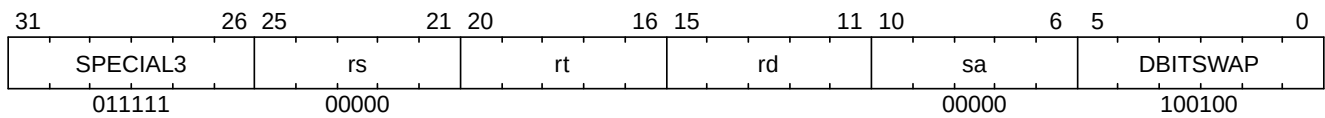


Figure 437. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitOrderOperands(rt=rd, rs=rt)
```

microMIPS

`dbitswap.32.micromips`

Assembly

`DBITSWAP {rs:gpr}`

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

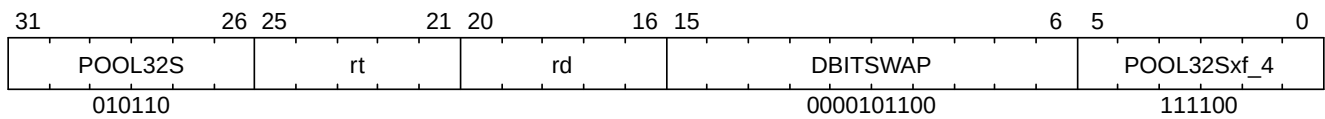


Figure 438. Encoding (32 bits)

Decode

```
del ctx, exec  
return DoubleBitOrderOperands(rt=rd, rs=rt)
```

8.169.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_byte_order(ctx, operands, kind="dbitswap")
```

8.169.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.169.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.170. DCLO

8.170.1. Purpose

Count leading ones in a 64-bit GPR value.

8.170.2. Formats

MIPS

dclo

Assembly

DCLO {rs:gpr}

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

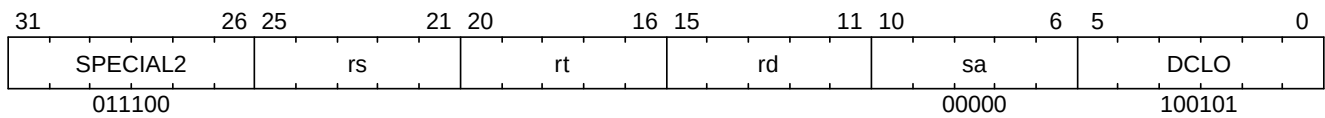


Figure 439. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rd, rs=rs)
```

dclo.r6

Assembly

DCLO {rs:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

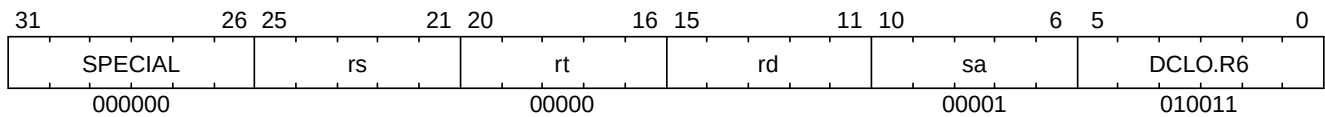


Figure 440. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rd, rs=rs)
```


microMIPS

dclo.32

Assembly

DCLO {rt:gpr}, {rs:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

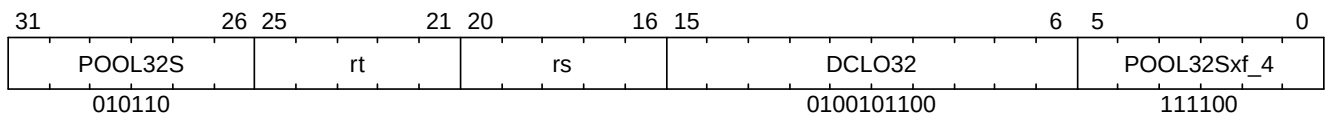


Figure 441. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rt, rs=rs)
```

nanoMIPS

dclo.32

Assembly

DCLO {rt:gpr}, {rs:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

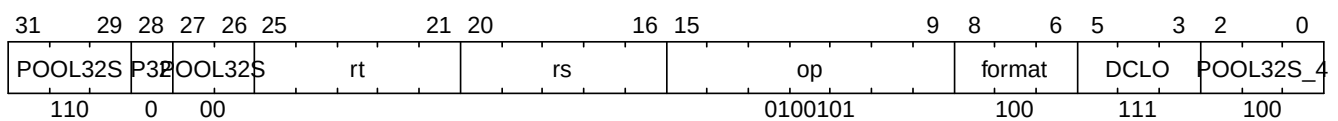


Figure 442. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rt, rs=rs)
```

8.170.3. Operation

```
validate_access_to_64bit_operations(ctx)
value = int(zero_extend(ctx.GPR[operands.rs], from_nbits=64))
ctx.GPR[operands.rt] = count_leading_bits(value, nbits=64, bit=1)
```

8.170.4. Description

The result is an integer from zero through 64.

8.170.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.171. DCLZ

8.171.1. Purpose

Count leading zeroes in a 64-bit GPR value.

8.171.2. Formats

MIPS

dclz

Assembly

DCLZ {rs:gpr}

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

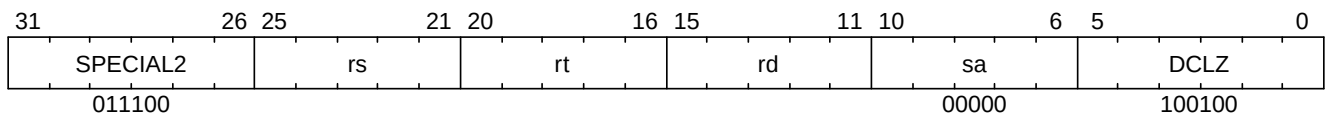


Figure 443. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rd, rs=rs)
```

dclz.r6

Assembly

DCLZ {rs:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

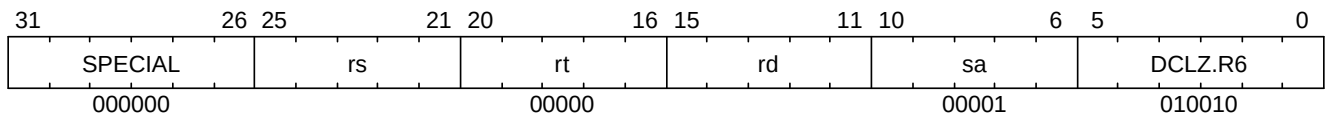


Figure 444. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rd, rs=rs)
```

microMIPS

dclz.32

Assembly

DCLZ {rt:gpr}, {rs:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

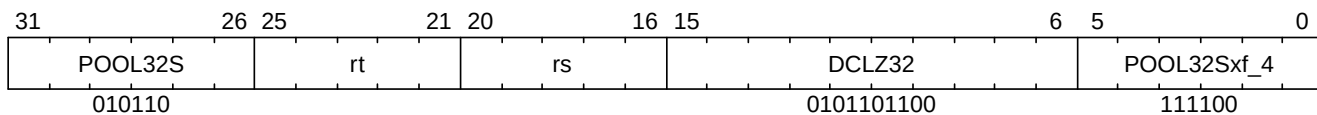


Figure 445. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rt, rs=rs)
```

nanoMIPS

dclz.32

Assembly

DCLZ {rt:gpr}, {rs:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

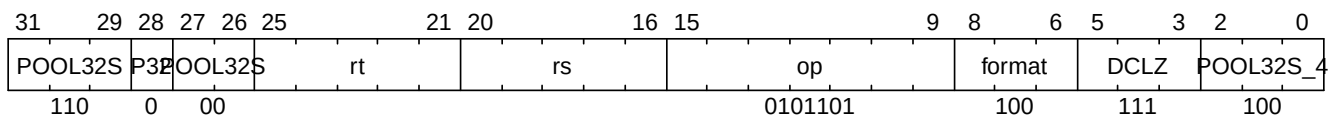


Figure 446. Encoding (32 bits)

Decode

```
return DCountOperands(rt=rt, rs=rs)
```

8.171.3. Operation

```
validate_access_to_64bit_operations(ctx)
value = int(zero_extend(ctx.GPR[operands.rs], from_nbits=64))
ctx.GPR[operands.rt] = count_leading_bits(value, nbits=64, bit=0)
```

8.171.4. Description

The result is an integer from zero through 64.

8.171.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.172. DDIV

8.172.1. Purpose

Divide signed 64-bit GPR values.

8.172.2. Formats

MIPS

`ddiv`

Assembly

`DDIV {rt:gpr}`

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

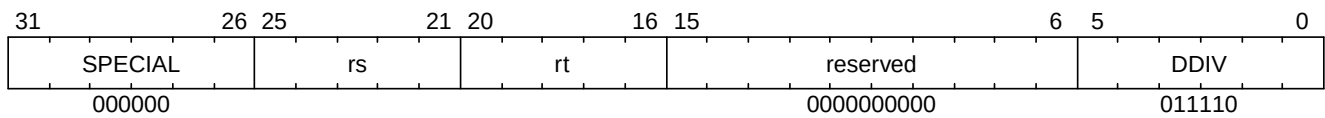


Figure 447. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

`ddiv.r6`

Assembly

`DDIV {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

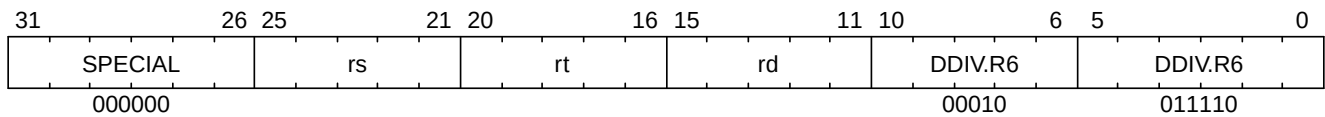


Figure 448. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

ddiv.32

Assembly

DDIV {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR < R6

Decode priority

0

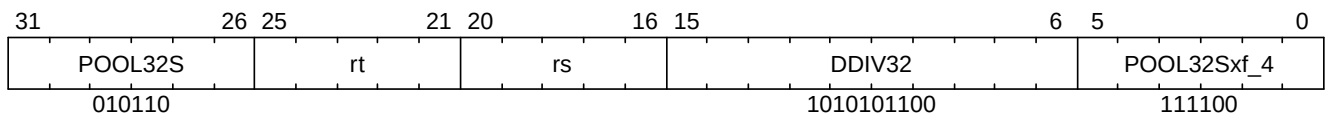


Figure 449. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

ddiv.r6.32

Assembly

DDIV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

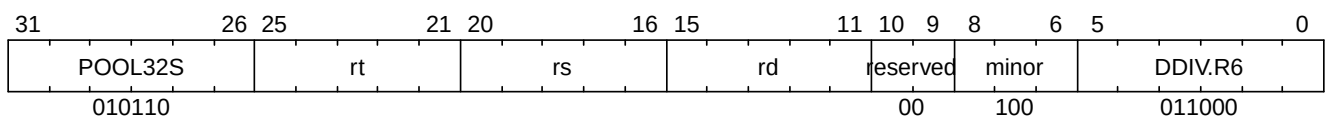


Figure 450. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

ddiv.32

Assembly

DDIV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

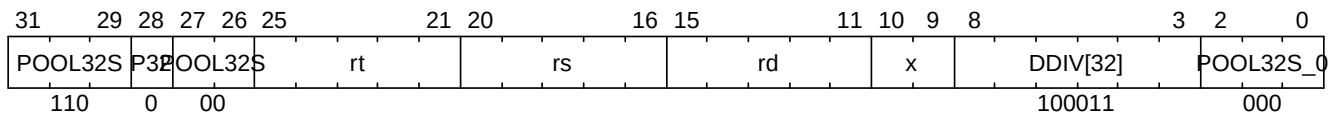


Figure 451. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.172.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_signed_divide(ctx, operands)
```

8.172.4. Description

Pre-Release 6 writes the quotient to LO and remainder to HI; Release 6 and nanoMIPS write the quotient to rd.

8.172.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.173. DDIVU

8.173.1. Purpose

Divide unsigned 64-bit GPR values.

8.173.2. Formats

MIPS

ddivu

Assembly

DDIVU {rt:gpr}

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

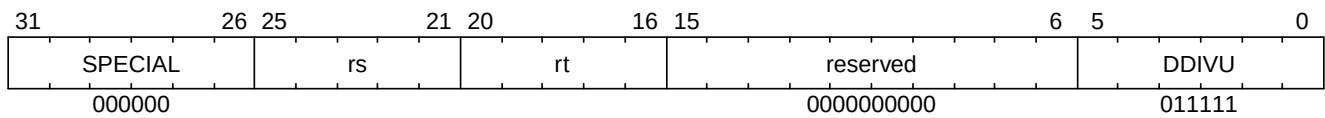


Figure 452. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

ddivu.r6

Assembly

DDIVU {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

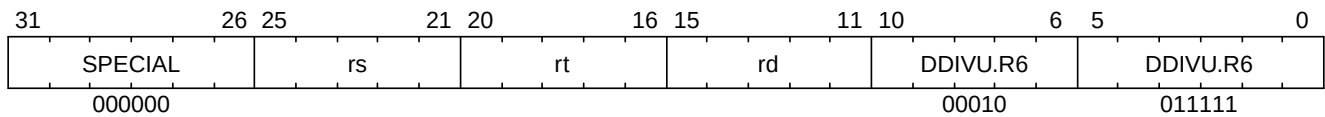


Figure 453. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

ddivu.32

Assembly

DDIVU {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR < R6

Decode priority

0

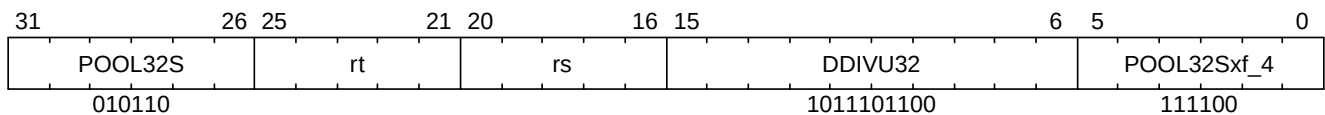


Figure 454. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

ddivu.r6.32

Assembly

DDIVU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

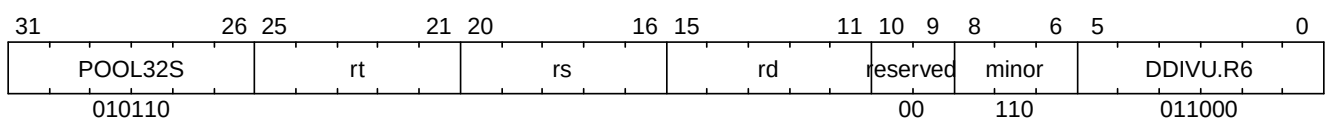


Figure 455. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

ddivu.32

Assembly

DDIVU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

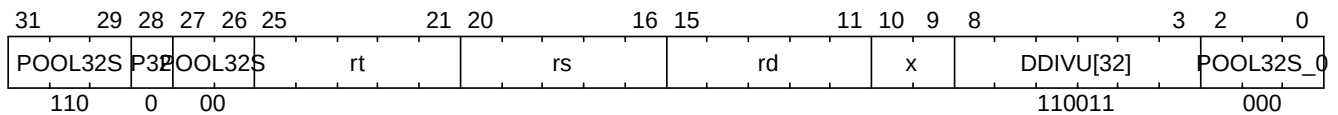


Figure 456. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.173.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_unsigned_divide(ctx, operands)
```

8.173.4. Description

Pre-Release 6 writes the quotient to LO and remainder to HI; Release 6 and nanoMIPS write the quotient to rd.

8.173.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.174. DERET

8.174.1. Purpose

Return from a debug exception.

8.174.2. Formats

MIPS

deret

Assembly

DERET

Availability

true

Decode priority

0

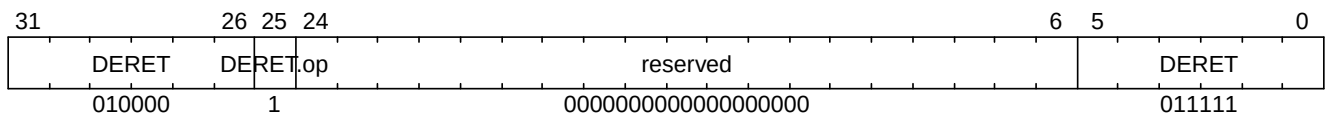


Figure 457. Encoding (32 bits)

Decode

```
exec.isCTI = True
return DeretOperands()
```

nanoMIPS

deret.32

Assembly

DERET

Availability

true

Decode priority

0

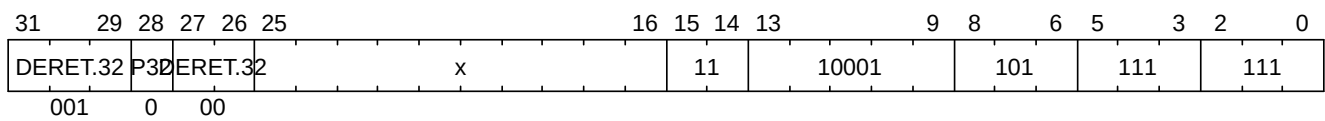


Figure 458. Encoding (32 bits)

```
exec.isCTI = True
return DeretOperands()
```

8.174.3. Operation

```
if not IsCoprocesor0Enabled(ctx):
    raise CoprocessorUnusable(0)
if not ctx.C0.Config1.EP or not ctx.C0.Debug.DM:
    raise ReservedInstruction("DERET requires debug mode")
target = read_cp0_register(ctx, 24, 0)
compact_branch(ctx, target)
clear_debug_mode(ctx)
clear_execution_hazards(ctx)
clear_instruction_hazards(ctx)
```

8.174.4. Description

DERET branches to DEPC and leaves debug mode.

8.174.5. Exceptions

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — DERET requires debug mode

8.175. DEXT

8.175.1. Purpose

Extract a low-half doubleword bitfield.

8.175.2. Formats

MIPS

dext

Assembly

DEXT {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

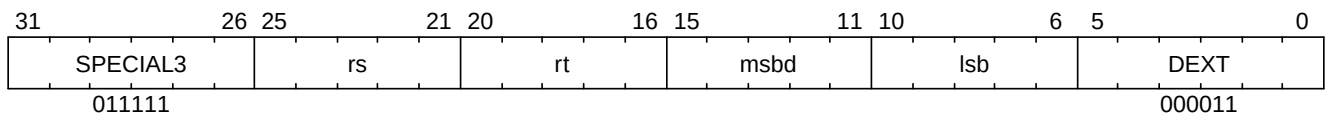


Figure 459. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfield0operands(rt=rt, rs=rs, pos=lsb, size=msbd + 1)
```

microMIPS

dext

Assembly

DEXT {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

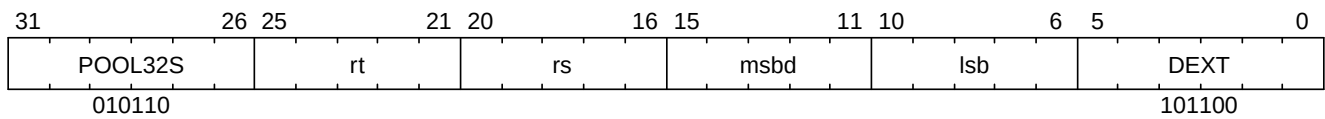


Figure 460. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msbd + 1)
```

nanoMIPS

dext.32

Assembly

DEXT {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

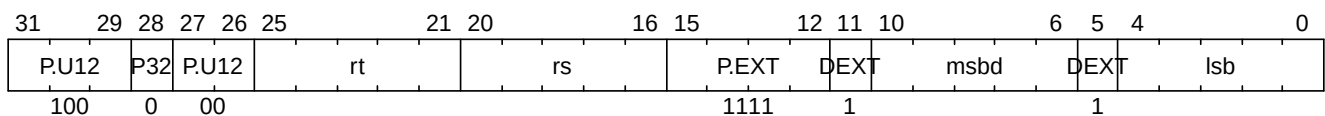


Figure 461. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msbd + 1)
```

8.175.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_extract(ctx, operands)
```

8.175.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.175.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.176. DEXTM

8.176.1. Purpose

Extract a middle doubleword bitfield.

8.176.2. Formats

MIPS

dextm

Assembly

DEXTM {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

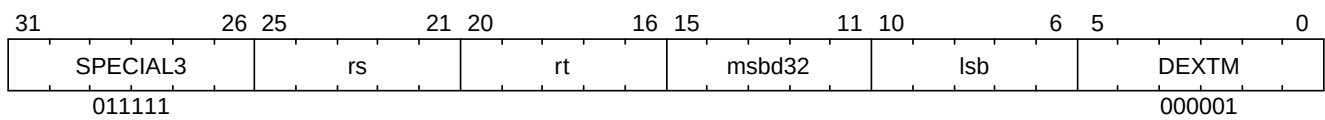


Figure 462. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfield0perands(rt=rt, rs=rs, pos=lsb, size=msbd32 + 33)
```

microMIPS

dextm

Assembly

DEXTM {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

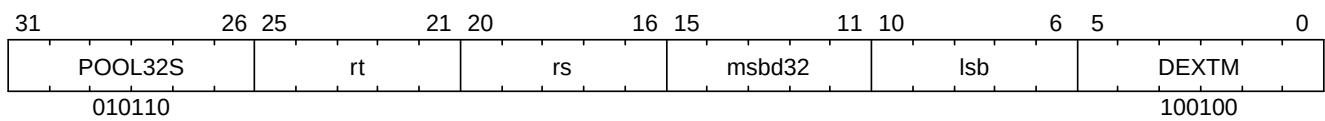


Figure 463. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msbd32 + 33)
```

nanoMIPS

dextm.32

Assembly

DEXTM {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

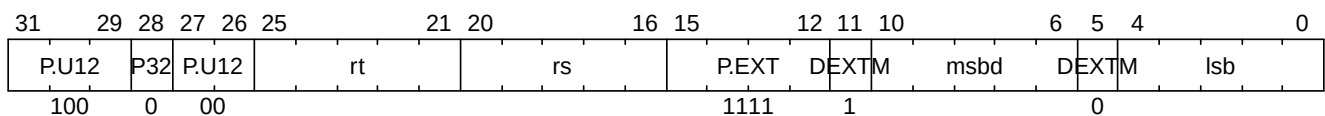


Figure 464. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msbd + 33)
```

8.176.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_extract(ctx, operands)
```

8.176.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.176.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.177. DXTU

8.177.1. Purpose

Extract an upper doubleword bitfield.

8.177.2. Formats

MIPS

dextu

Assembly

DEXTU {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

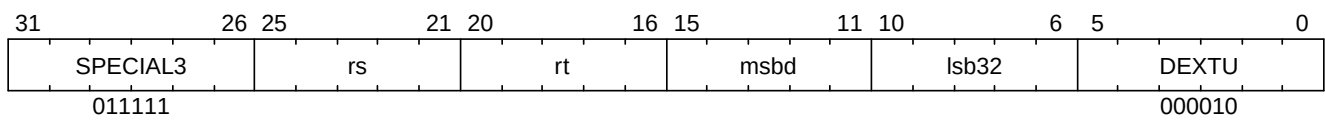


Figure 465. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfield0operands(rt=rt, rs=rs, pos=lsb32 + 32, size=msbd + 1)
```

microMIPS

dextu

Assembly

DEXTU {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

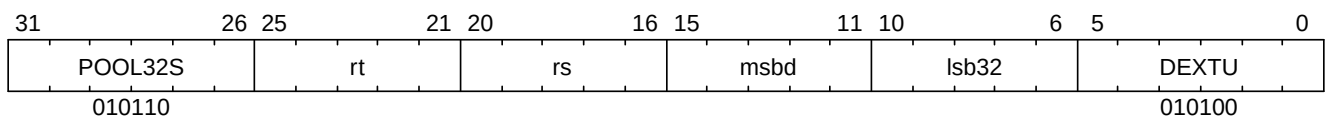


Figure 466. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb32 + 32, size=msbd + 1)
```

nanoMIPS

dextu.32

Assembly

DEXTU {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

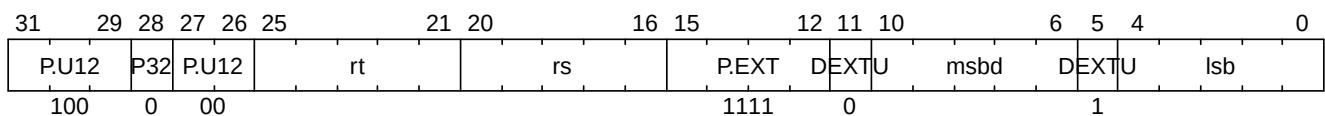


Figure 467. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb + 32, size=msbd + 1)
```

8.177.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_extract(ctx, operands)
```

8.177.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.177.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.178. DI

8.178.1. Purpose

Disable interrupts and return prior Status.

8.178.2. Formats

MIPS

di

Assembly

DI {rt:gpr}

Availability

AR >= R2_3_5

Decode priority

0

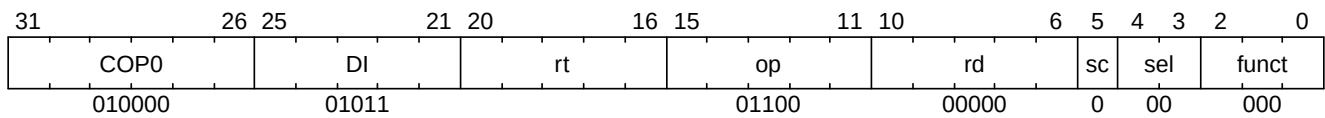


Figure 468. Encoding (32 bits)

Decode

```
return Di0perands(rt=rt)
```

microMIPS

di.32

Assembly

DI {rt:gpr}

Availability

true

Decode priority

0

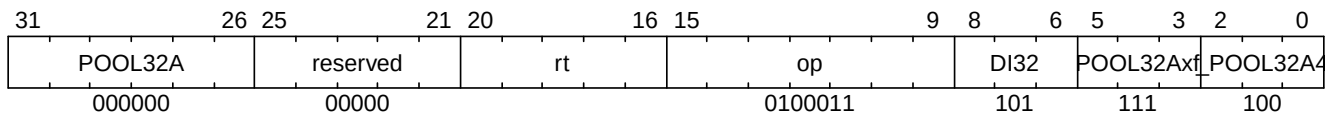


Figure 469. Encoding (32 bits)

Decode

```
return DiOperands(rt=rt)
```

nanoMIPS

di.32

Assembly

DI {rt:gpr}

Availability

true

Decode priority

0

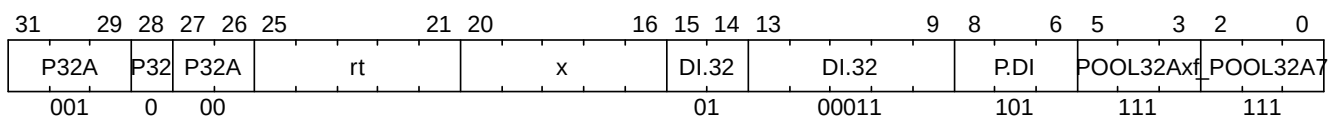


Figure 470. Encoding (32 bits)

Decode

```
return DiOperands(rt=rt)
```

8.178.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
ctx.GPR[operands.rt] = sign_extend(read_cp0_register(ctx, 12, 0), from_nbits=32)  
set_interrupt_enabled(ctx, False)
```

8.178.4. Description

DI saves CP0.Status in rt and clears Status.IE.

8.178.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.179. DINS

8.179.1. Purpose

Insert a low-half doubleword bitfield.

8.179.2. Formats

MIPS

dins

Assembly

DINS {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

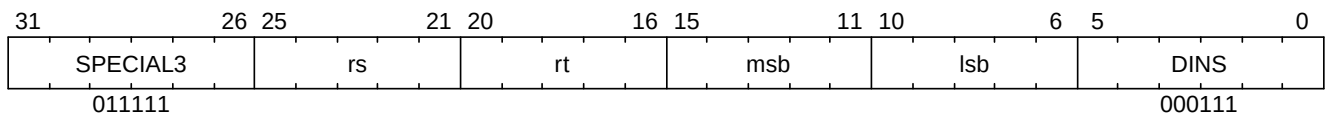


Figure 471. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfield0operands(rt=rt, rs=rs, pos=lsb, size=msb - lsb + 1)
```

microMIPS

dins

Assembly

DINS {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

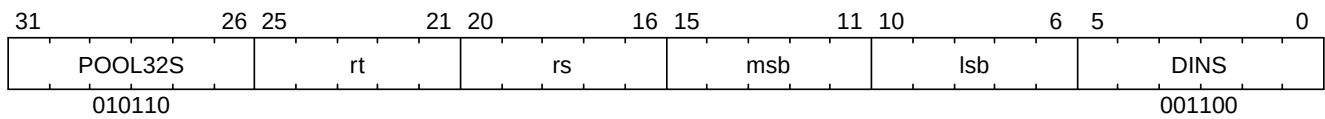


Figure 472. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msb - lsb + 1)
```

nanoMIPS

dins.32

Assembly

DINS {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

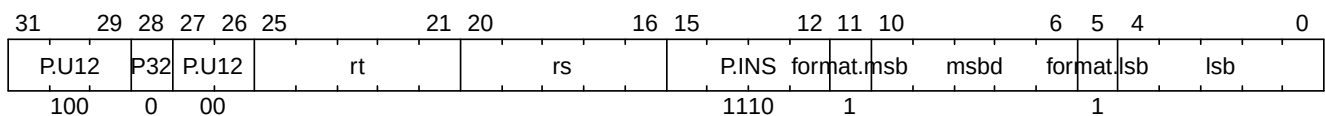


Figure 473. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msbd - lsb + 1)
```

8.179.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_insert(ctx, operands)
```

8.179.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.179.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.180. DINSM

8.180.1. Purpose

Insert a middle doubleword bitfield.

8.180.2. Formats

MIPS

dinsm

Assembly

DINSM {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

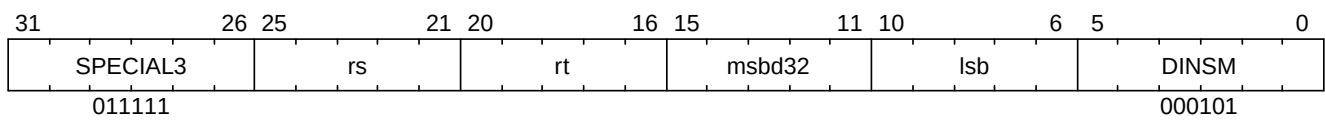


Figure 474. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfield0operands(rt=rt, rs=rs, pos=lsb, size=msbd32 - lsb + 33)
```

microMIPS

dinsm

Assembly

DINSM {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

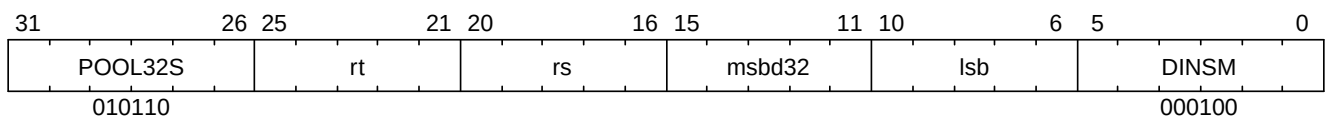


Figure 475. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msbd32 - lsb + 33)
```

nanoMIPS

dinsm.32

Assembly

DINSM {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

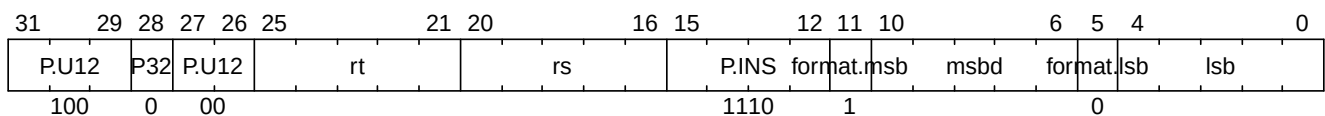


Figure 476. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb, size=msbd - lsb + 33)
```

8.180.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_insert(ctx, operands)
```

8.180.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.180.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.181. DINSU

8.181.1. Purpose

Insert an upper doubleword bitfield.

8.181.2. Formats

MIPS

dinsu

Assembly

DINSU {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

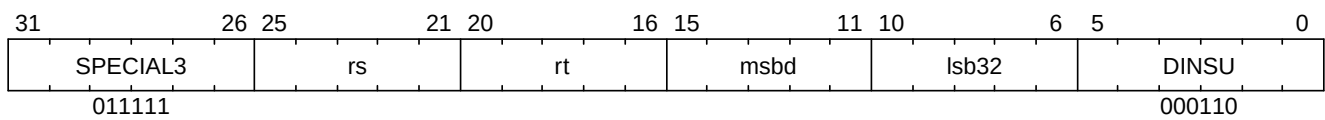


Figure 477. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfield0operands(rt=rt, rs=rs, pos=lsb32 + 32, size=msbd - lsb32 + 1)
```

microMIPS

dinsu

Assembly

DINSU {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

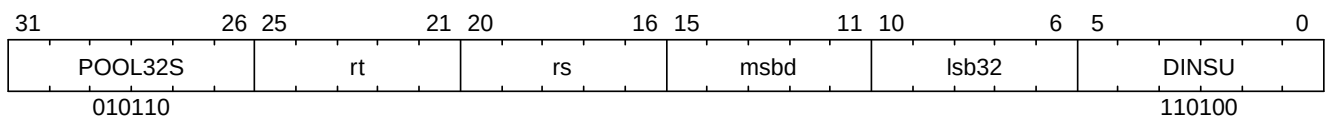


Figure 478. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb32 + 32, size=msbd - lsb32 + 1)
```

nanoMIPS

dinsu.32

Assembly

DINSU {rt:gpr}, {rs:gpr}, {pos:u6}, {size:u6}

Availability

AT >= MIPS64_32

Decode priority

0

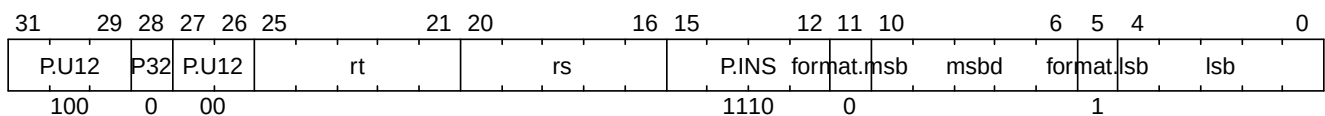


Figure 479. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitfieldOperands(rt=rt, rs=rs, pos=lsb + 32, size=msbd - lsb + 1)
```

8.181.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_insert(ctx, operands)
```

8.181.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.181.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.182. DIV

8.182.1. Purpose

Divide the signed 32-bit value in `rs` by the signed 32-bit value in `rt`, and place the quotient in `rd`.

8.182.2. Formats

MIPS

`div`

Assembly

`DIV {rs:gpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

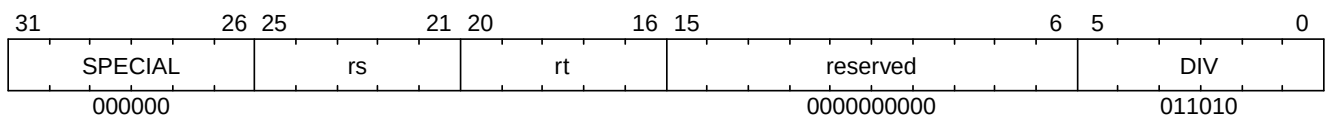


Figure 480. Encoding (32 bits)

Decode

```
return DivOperands(rd=None, rs=rs, rt=rt)
```

`div.r6`

Assembly

`DIV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R6

Decode priority

0

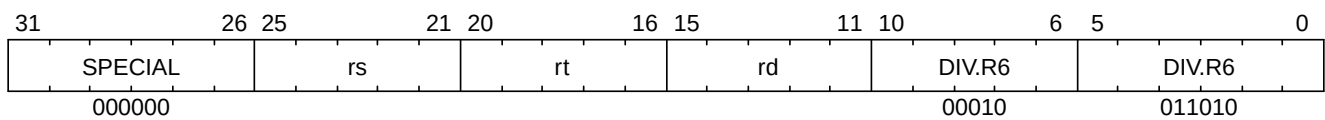


Figure 481. Encoding (32 bits)

Decode

```
return DivOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

div.32

Assembly

DIV {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

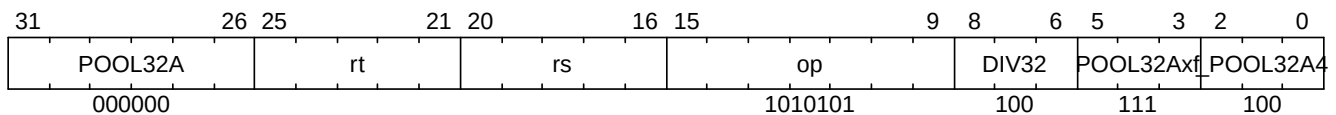


Figure 482. Encoding (32 bits)

Decode

```
return DivOperands(rd=None, rs=rs, rt=rt)
```

div.r6.32

Assembly

DIV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

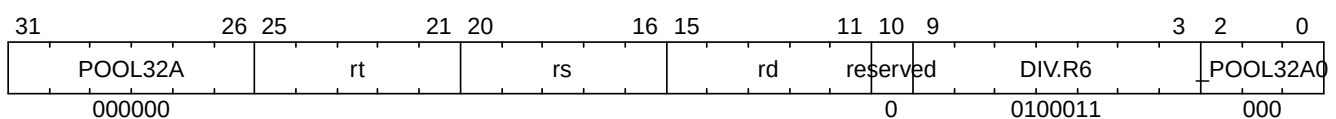


Figure 483. Encoding (32 bits)

Decode

```
return DivOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

div.32

Assembly

DIV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

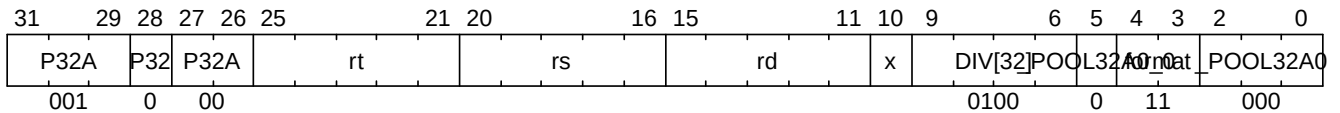


Figure 484. Encoding (32 bits)

Decode

```
del ctx, exec
return DivOperands(rd=rd, rs=rs, rt=rt)
```

8.182.3. Operation

```
quotient = divide_signed_word(
    ctx,
    ctx.GPR[operands.rs],
    ctx.GPR[operands.rt],
    operation="DIV[32]",
    result="quotient",
)
if operands.rd is None:
    remainder = divide_signed_word(
        ctx,
        ctx.GPR[operands.rs],
        ctx.GPR[operands.rt],
        operation="DIV",
        result="remainder",
    )
    if quotient is not None and remainder is not None:
        ctx.LO = int(sign_extend(quotient, from_nbits=32))
        ctx.HI = int(sign_extend(remainder, from_nbits=32))
elif quotient is not None:
    ctx.GPR[operands.rd] = sign_extend(quotient, from_nbits=32)
```

8.182.4. Description

DIV[32] interprets the low 32 bits of both source GPRs as signed values. The quotient is sign-extended from 32 bits when written to the implemented-width destination GPR. A zero divisor has an unspecified result and is routed through the replaceable division handler.

8.182.5. Exceptions

None.

8.183. DIV.D

8.183.1. Purpose

Divide two D-format floating-point values.

8.183.2. Formats

MIPS

div.d

Assembly

DIV.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

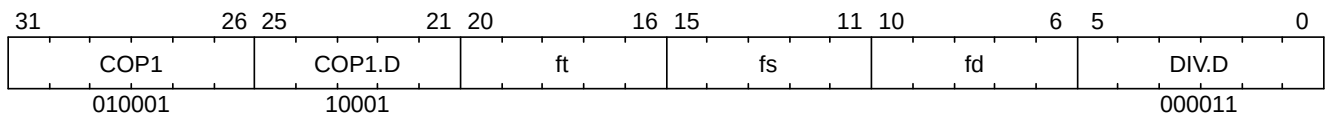


Figure 485. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

div.d

Assembly

DIV.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

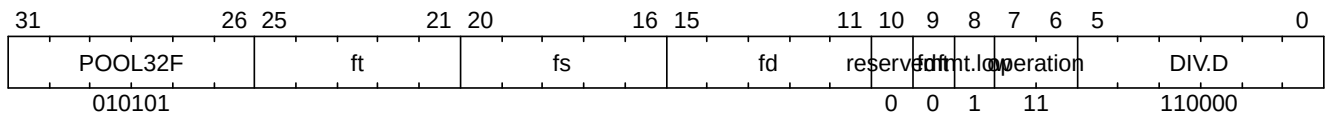


Figure 486. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

div.d

Assembly

DIV.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

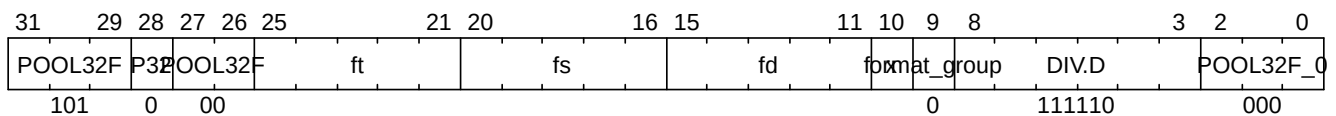


Figure 487. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.183.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_div(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.183.4. Description

DIV.D validates CP1 and D-format support before updating the destination FPR.

8.183.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.184. DIV.S

8.184.1. Purpose

Divide two S-format floating-point values.

8.184.2. Formats

MIPS

div.s

Assembly

DIV.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

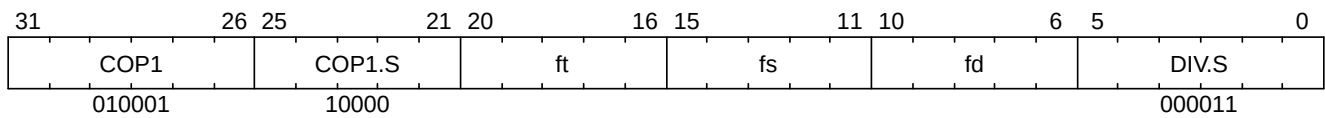


Figure 488. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

div.s

Assembly

DIV.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

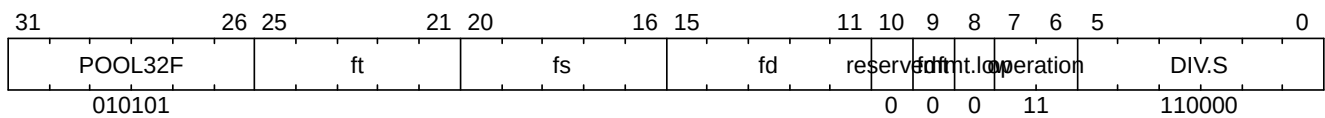


Figure 489. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

div.s

Assembly

DIV.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

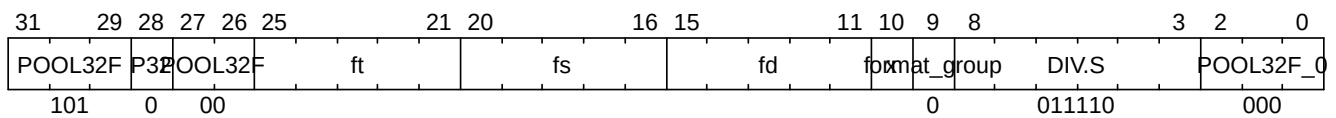


Figure 490. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.184.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_div(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.184.4. Description

DIV.S validates CP1 and S-format support before updating the destination FPR.

8.184.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported.

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.185. DIVU

8.185.1. Purpose

Divide the unsigned 32-bit value in `rs` by the unsigned 32-bit value in `rt`, and place the quotient in `rd`.

8.185.2. Formats

MIPS

`divu`

Assembly

`DIVU {rs:gpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

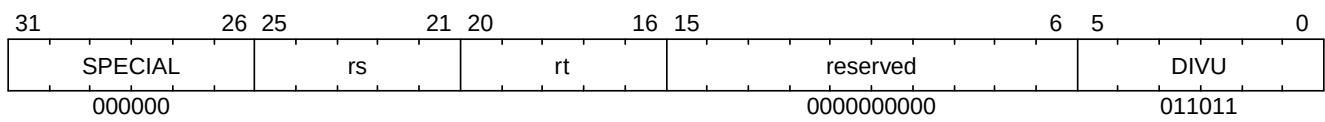


Figure 491. Encoding (32 bits)

Decode

```
return DivuOperands(rd=None, rs=rs, rt=rt)
```

`divu.r6`

Assembly

`DIVU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R6

Decode priority

0

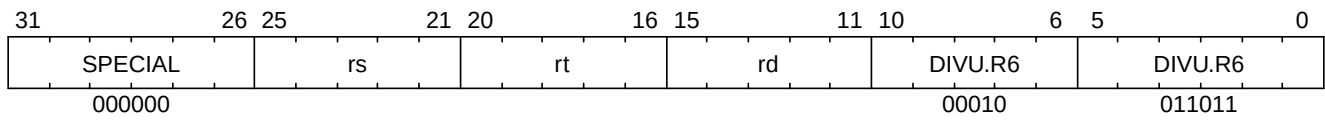


Figure 492. Encoding (32 bits)

Decode

```
return DivuOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

divu.32

Assembly

```
DIVU {rs:gpr}, {rt:gpr}
```

Availability

MMAR < R6

Decode priority

0

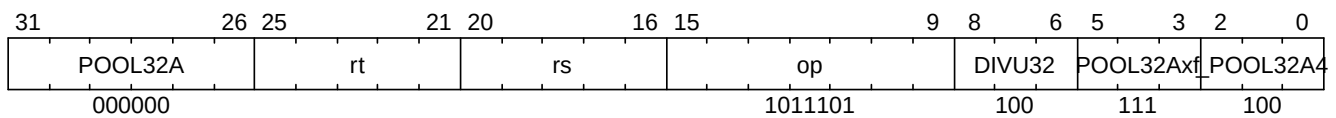


Figure 493. Encoding (32 bits)

Decode

```
return DivuOperands(rd=None, rs=rs, rt=rt)
```

divu.r6.32

Assembly

```
DIVU {rd:gpr}, {rs:gpr}, {rt:gpr}
```

Availability

MMAR >= R6

Decode priority

0

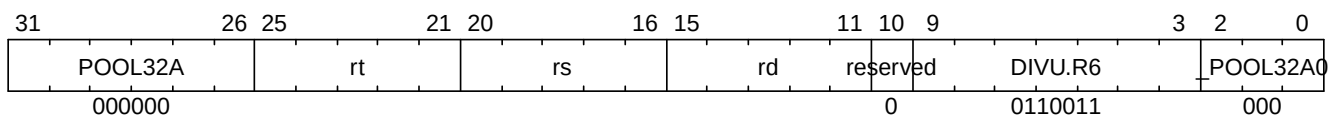


Figure 494. Encoding (32 bits)

Decode

```
return DivuOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

divu.32

Assembly

DIVU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

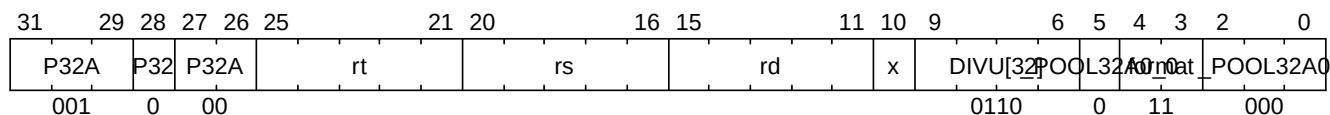


Figure 495. Encoding (32 bits)

Decode

```
del ctx, exec
return DivuOperands(rd=rd, rs=rs, rt=rt)
```

8.185.3. Operation

```
numerator = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
denominator = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
quotient = divide_unsigned_word(
    ctx,
    numerator,
    denominator,
    operation="DIVU[32]",
    result="quotient",
)
if operands.rd is None:
    remainder = divide_unsigned_word(
        ctx,
        numerator,
        denominator,
        operation="DIVU",
        result="remainder",
    )
    if quotient is not None and remainder is not None:
        ctx.LO = int(sign_extend(quotient, from_nbits=32))
        ctx.HI = int(sign_extend(remainder, from_nbits=32))
elif quotient is not None:
    ctx.GPR[operands.rd] = sign_extend(quotient, from_nbits=32)
```

8.185.4. Description

DIVU[32] zero-extends both source GPRs from 32 bits before division. The quotient is sign-extended

from 32 bits when written to the implemented-width destination GPR. A zero divisor has an unspecified result; the execution context may provide a replaceable handler. The simulator reports an unspecified-result boundary when none is supplied.

8.185.5. Exceptions

None.

8.186. DLSA

8.186.1. Purpose

Add a scaled 64-bit GPR value to another 64-bit GPR value.

8.186.2. Formats

MIPS

`dlsa`

Assembly

`DLSA {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u3}`

Availability

`AT >= MIPS64_32 && AR >= R6`

Decode priority

`0`

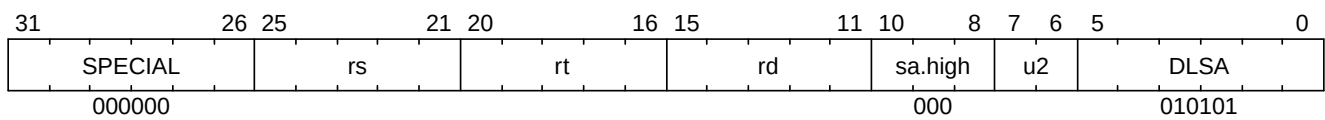


Figure 496. Encoding (32 bits)

Decode

```
shift = zero_extend(u2 + 1, from_nbits=3)
return DlsaOperands(rd=rd, rs=rs, rt=rt, shift=shift)
```

microMIPS

`dlsa.32.micromips`

Assembly

`DLSA {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u3}`

Availability

`MMAR >= R6 && AT >= MIPS64_32`

Decode priority

`0`

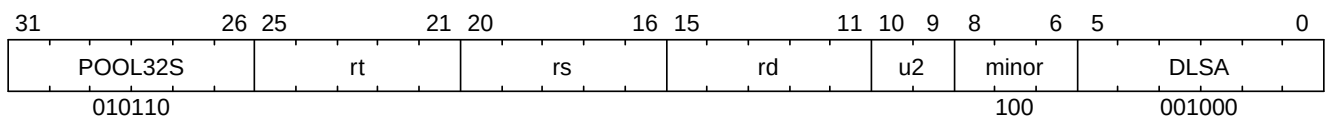


Figure 497. Encoding (32 bits)

Decode

```
shift = zero_extend(u2 + 1, from_nbits=3)
return DlsaOperands(rd=rd, rs=rs, rt=rt, shift=shift)
```

nanoMIPS

dlsa.32

Assembly

DLSA {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u2}

Availability

AT >= MIPS64_32

Decode priority

0

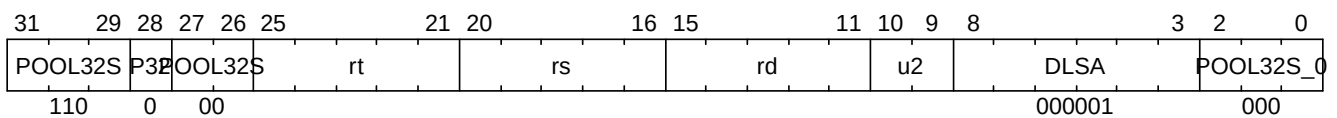


Figure 498. Encoding (32 bits)

Decode

```
shift = zero_extend(u2, from_nbits=2)
return DlsaOperands(rd=rd, rs=rs, rt=rt, shift=shift)
```

8.186.3. Operation

```
validate_access_to_64bit_operations(ctx)
scaled = int(zero_extend(ctx.GPR[operands.rs], from_nbits=64)) << int(
    operands.shift
)
result = scaled + int(zero_extend(ctx.GPR[operands.rt], from_nbits=64))
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=64)
```

8.186.4. Description

Classic MIPS encodes shifts 1 through 4; nanoMIPS64 encodes shifts 0 through 3.

8.186.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.187. DLUI

8.187.1. Purpose

Load a 32-bit immediate into the upper half of a 64-bit GPR.

8.187.2. Formats

nanoMIPS

dlui.48

Assembly

DLUI {rt:gpr}, {imm:u32}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

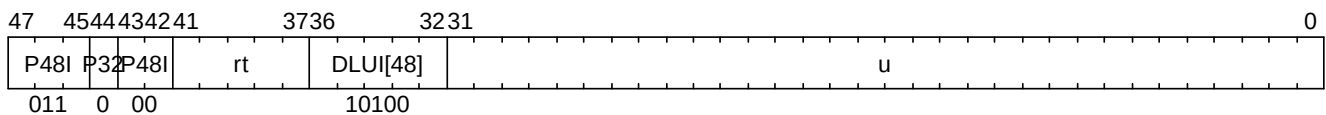


Figure 499. Encoding (48 bits)

Decode

```
imm = zero_extend(u, from_nbits=32)
return DluiOperands(rt=rt, imm=imm)
```

8.187.3. Operation

```
validate_access_to_64bit_operations(ctx)
ctx.GPR[operands.rt] = sign_extend(int(operands.imm) << 32, from_nbits=64)
```

8.187.4. Description

The lower 32 bits are cleared and the resulting 64-bit bit vector is written to the GPR.

8.187.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.188. DMFC0

8.188.1. Purpose

Move a 64-bit CP0 register value to a GPR.

8.188.2. Formats

MIPS

dmfc0

Assembly

DMFC0 {c0s:u5}, {sel:u3}

Availability

AT >= MIPS64_32

Decode priority

0

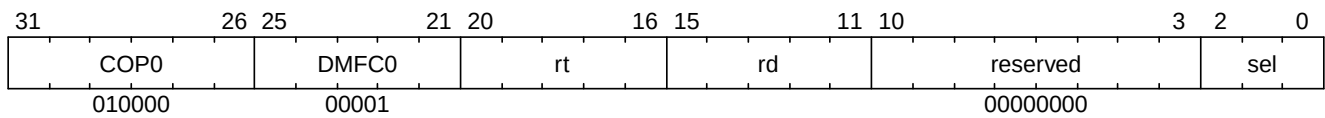


Figure 500. Encoding (32 bits)

Decode

```
selector = zero_extend(sel, from_nbits=3)
return DoubleCp0operands(rt=rt, c0s=rd, sel=selector)
```

microMIPS

dmfc0

Assembly

DMFC0 {rt:gpr}, {c0s:u5}, {sel:u3}

Availability

AT >= MIPS64_32

Decode priority

0

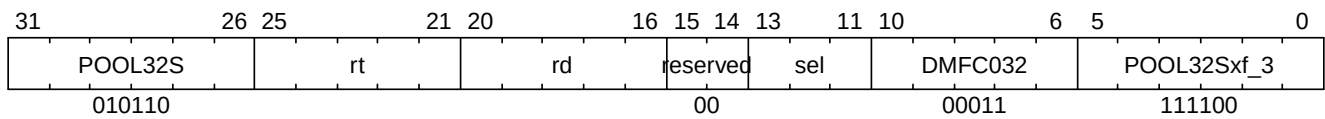


Figure 501. Encoding (32 bits)

Decode

```
selector = zero_extend(sel, from_nbits=3)
return DoubleCp0Operands(rt=rt, c0s=rd, sel=selector)
```

nanoMIPS

dmfc0.32

Assembly

DMFC0 {rt:gpr}, {c0s:u5}, {sel:u5}

Availability

AT >= MIPS64_32

Decode priority

0

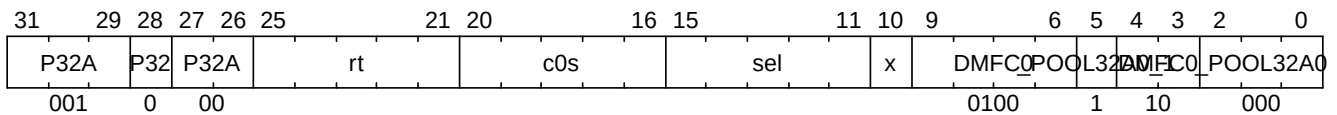


Figure 502. Encoding (32 bits)

Decode

```
selector = zero_extend(sel, from_nbits=5)
return DoubleCp0Operands(rt=rt, c0s=c0s, sel=selector)
```

8.188.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
validate_access_to_64bit_operations(ctx)
value = read_cp0_register(ctx, operands.c0s, operands.sel)
ctx.GPR[operands.rt] = sign_extend(value, from_nbits=64)
```

8.188.4. Description

The encoded CP0 register is resolved through the active register-bank definition.

8.188.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.189. DMFC1

8.189.1. Purpose

Transfer a value through Coprocessor 1.

8.189.2. Formats

MIPS

dmfc1

Assembly

DMFC1 {rt:gpr}, {fs:fpr}

Availability

AT >= MIPS64_32

Decode priority

0

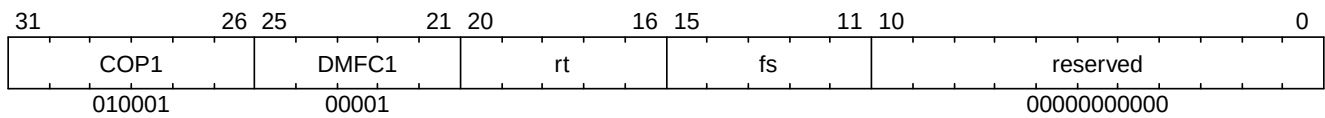


Figure 503. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

microMIPS

dmfc1.32

Assembly

DMFC1 {rt:gpr}, {fs:fpr}

Availability

AT >= MIPS64_32

Decode priority

0

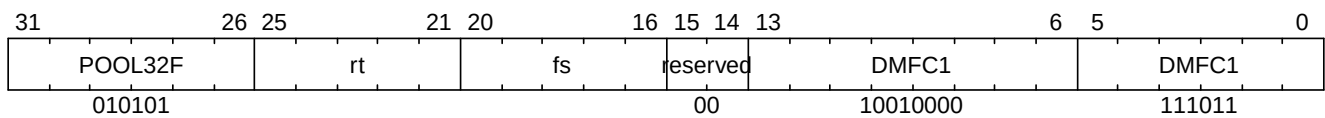


Figure 504. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

nanoMIPS

dmfc1.32

Assembly

DMFC1 {rt:gpr}, {fs:fpr}

Availability

AT >= MIPS64_32

Decode priority

0

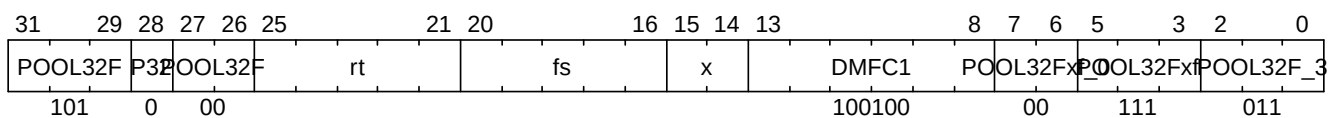


Figure 505. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

8.189.3. Operation

```
validate_access_to_cp1_resources(ctx, "64", require_mips64=True)
ctx.GPR[operands.rt] = sign_extend(get_fpr(ctx, operands.fs, "64"), from_nbits=64)
```

8.189.4. Description

DMFC1 performs its CP1 access checks before transferring data.

8.189.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.

8.190. DMOD

8.190.1. Purpose

Compute the signed 64-bit remainder.

8.190.2. Formats

MIPS

dmod.r6

Assembly

DMOD {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

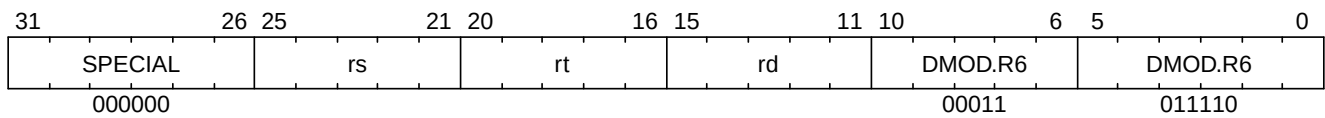


Figure 506. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

dmod.r6.32

Assembly

DMOD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

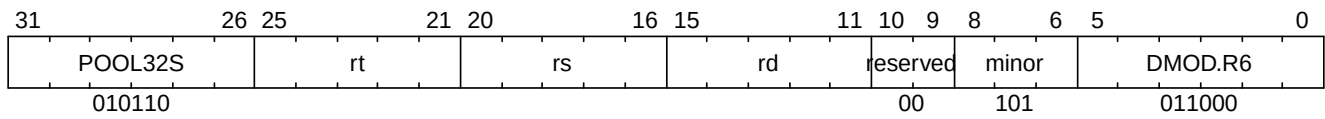


Figure 507. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dmod.32

Assembly

DMOD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

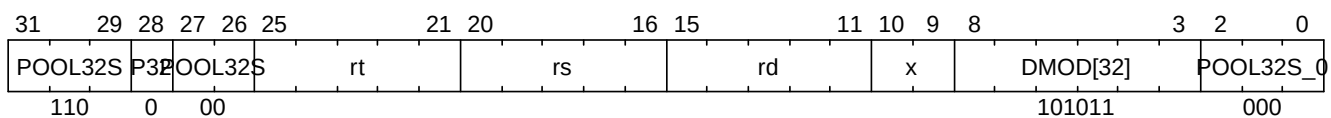


Figure 508. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.190.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_signed_modulo(ctx, operands)
```

8.190.4. Description

The remainder is written to rd; a zero divisor produces an unspecified result selected by the division policy.

8.190.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.191. DMODU

8.191.1. Purpose

Compute the unsigned 64-bit remainder.

8.191.2. Formats

MIPS

`dmodu.r6`

Assembly

`DMODU {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

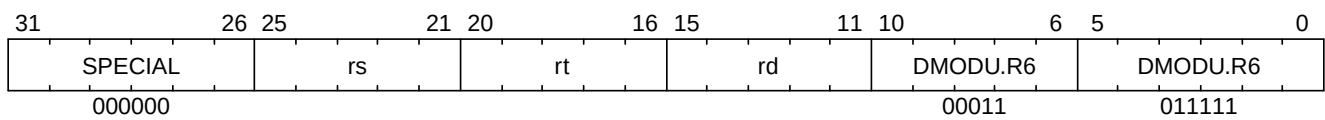


Figure 509. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`dmodu.r6.32`

Assembly

`DMODU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

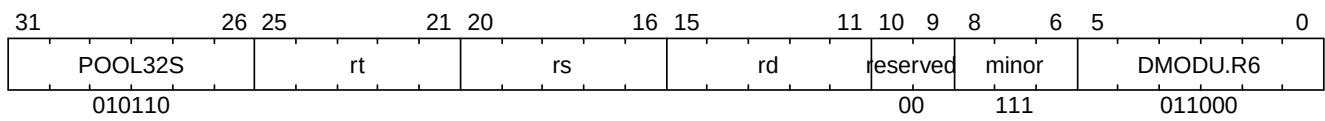


Figure 510. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dmodu.32

Assembly

DMODU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

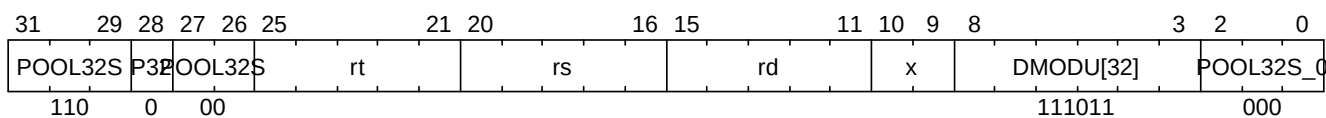


Figure 511. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.191.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_unsigned_modulo(ctx, operands)
```

8.191.4. Description

The remainder is written to rd; a zero divisor produces an unspecified result selected by the division policy.

8.191.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.192. DMTC0

8.192.1. Purpose

Move a 64-bit GPR value to a CP0 register.

8.192.2. Formats

MIPS

`dmtc0`

Assembly

`DMTC0 {c0s:u5}, {sel:u3}`

Availability

AT >= MIPS64_32

Decode priority

0

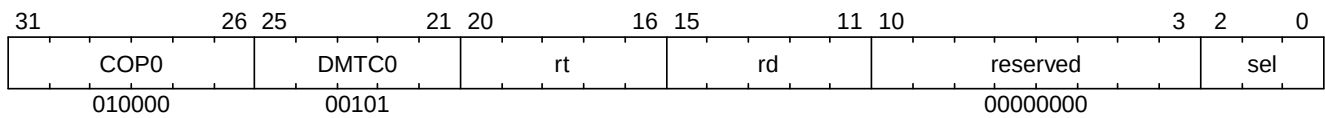


Figure 512. Encoding (32 bits)

Decode

```
selector = zero_extend(sel, from_nbits=3)
return DoubleCp0operands(rt=rt, c0s=rd, sel=selector)
```

microMIPS

`dmtc0`

Assembly

`DMTC0 {rt:gpr}, {c0s:u5}, {sel:u3}`

Availability

AT >= MIPS64_32

Decode priority

0

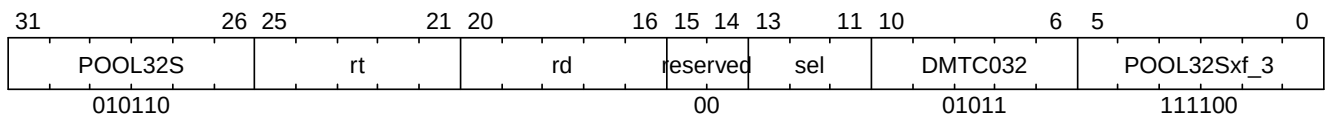


Figure 513. Encoding (32 bits)

Decode

```
selector = zero_extend(sel, from_nbits=3)
return DoubleCp0Operands(rt=rt, c0s=rd, sel=selector)
```

nanoMIPS

dmtc0.32

Assembly

DMTC0 {rt:gpr}, {c0s:u5}, {sel:u5}

Availability

AT >= MIPS64_32

Decode priority

0

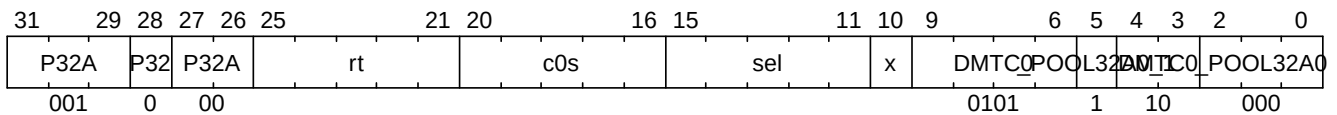


Figure 514. Encoding (32 bits)

Decode

```
selector = zero_extend(sel, from_nbits=5)
return DoubleCp0Operands(rt=rt, c0s=c0s, sel=selector)
```

8.192.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
validate_access_to_64bit_operations(ctx)
value = zero_extend(ctx.GPR[operands.rt], from_nbits=64)
write_cp0_register(ctx, value, operands.c0s, operands.sel, nbits=64)
```

8.192.4. Description

Writable fields of the encoded CP0 register receive their corresponding source bits.

8.192.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.193. DMTC1

8.193.1. Purpose

Transfer a value through Coprocessor 1.

8.193.2. Formats

MIPS

dmtc1

Assembly

DMTC1 {rt:gpr}, {fs:fpr}

Availability

AT >= MIPS64_32

Decode priority

0

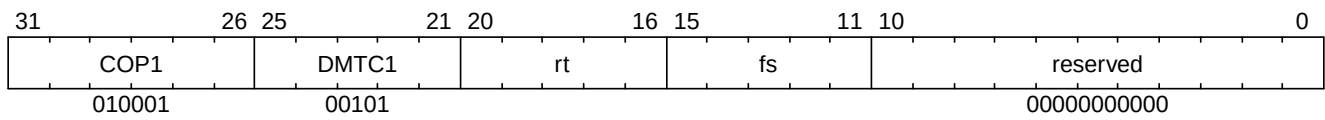


Figure 515. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

microMIPS

dmtc1.32

Assembly

DMTC1 {rt:gpr}, {fs:fpr}

Availability

AT >= MIPS64_32

Decode priority

0

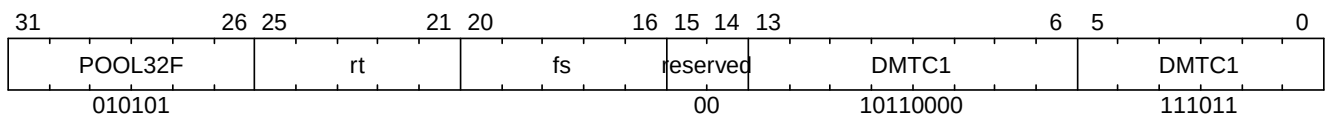


Figure 516. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

nanoMIPS

dmtc1.32

Assembly

DMTC1 {rt:gpr}, {fs:fpr}

Availability

AT >= MIPS64_32

Decode priority

0

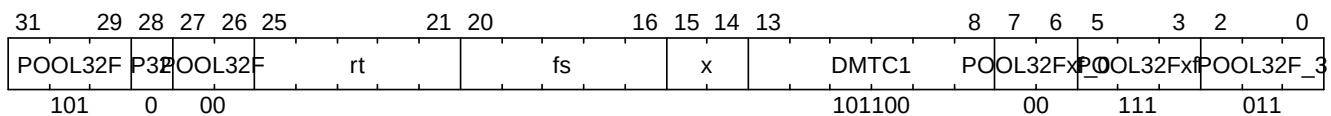


Figure 517. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

8.193.3. Operation

```
validate_access_to_cp1_resources(ctx, "64", require_mips64=True)
set_fpr(ctx, operands.fs, ctx.GPR[operands.rt], "64")
```

8.193.4. Description

DMTC1 performs its CP1 access checks before transferring data.

8.193.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.

8.194. DMUH

8.194.1. Purpose

Multiply signed 64-bit GPR values and return the high half.

8.194.2. Formats

MIPS

dmuh.r6

Assembly

DMUH {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

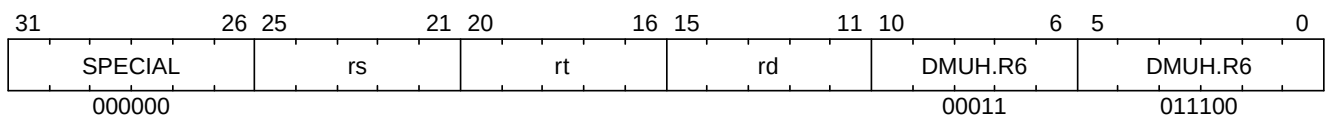


Figure 518. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

dmuh.r6.32

Assembly

DMUH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

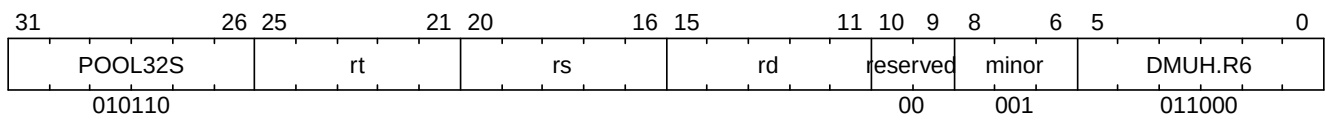


Figure 519. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dmuh.32

Assembly

DMUH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

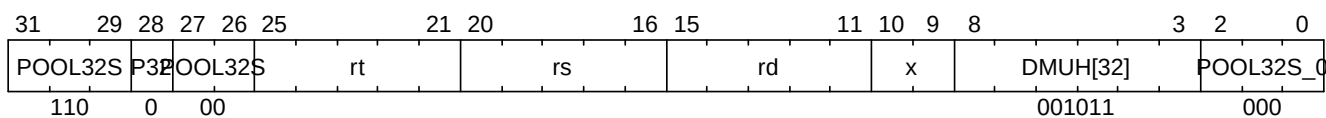


Figure 520. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.194.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_signed_multiply_high(ctx, operands)
```

8.194.4. Description

Bits 127:64 of the product are written to rd.

8.194.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.195. DMUHU

8.195.1. Purpose

Multiply unsigned 64-bit GPR values and return the high half.

8.195.2. Formats

MIPS

dmuhu.r6

Assembly

DMUHU {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

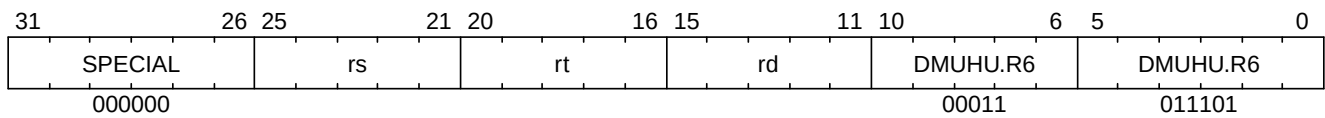


Figure 521. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

dmuhu.r6.32

Assembly

DMUHU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

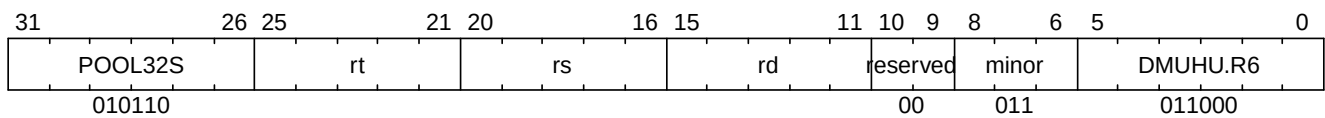


Figure 522. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dmuhu.32

Assembly

DMUHU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

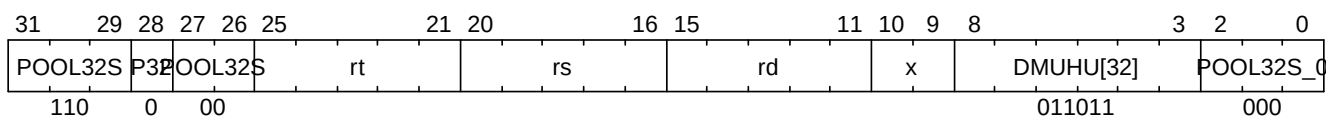


Figure 523. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.195.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_unsigned_multiply_high(ctx, operands)
```

8.195.4. Description

Bits 127:64 of the product are written to rd.

8.195.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.196. DMUL

8.196.1. Purpose

Multiply signed 64-bit GPR values.

8.196.2. Formats

MIPS

dmul.r6

Assembly

DMUL {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

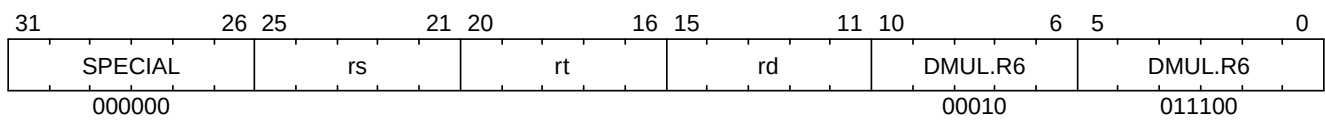


Figure 524. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

dmul.r6.32

Assembly

DMUL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

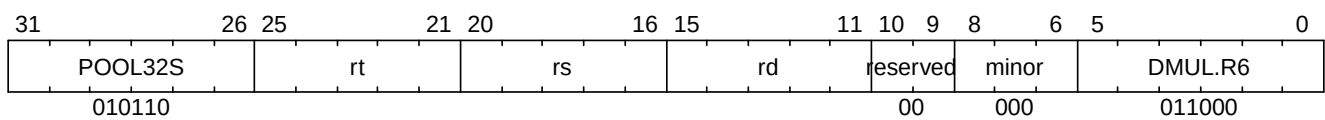


Figure 525. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dmul.32

Assembly

DMUL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

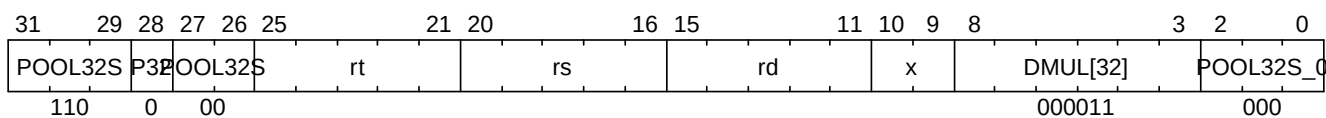


Figure 526. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.196.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_signed_multiply_low(ctx, operands)
```

8.196.4. Description

The low 64 bits of the product are written to rd.

8.196.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.197. DMULT

8.197.1. Purpose

Multiply signed 64-bit GPR values.

8.197.2. Formats

MIPS

`dmult`

Assembly

`DMULT {rt:gpr}`

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

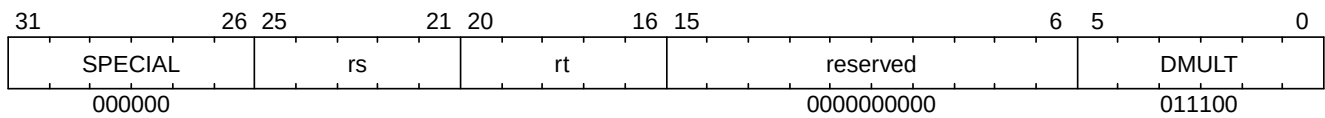


Figure 527. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

microMIPS

`dmult.32`

Assembly

`DMULT {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32 && MMAR < R6

Decode priority

0

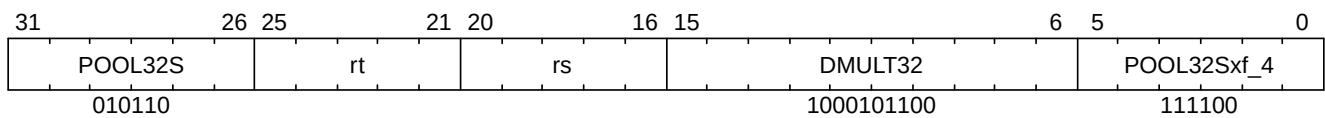


Figure 528. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

8.197.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_signed_multiply_hilo(ctx, operands)
```

8.197.4. Description

The 128-bit product is split between the 64-bit HI and LO registers.

8.197.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.198. DMULTU

8.198.1. Purpose

Multiply unsigned 64-bit GPR values.

8.198.2. Formats

MIPS

`dmultu`

Assembly

`DMULTU {rt:gpr}`

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

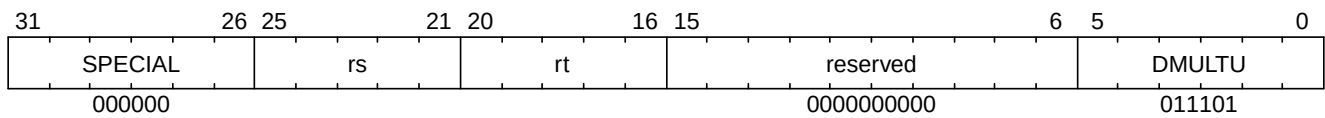


Figure 529. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

microMIPS

`dmultu.32`

Assembly

`DMULTU {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32 && MMAR < R6

Decode priority

0

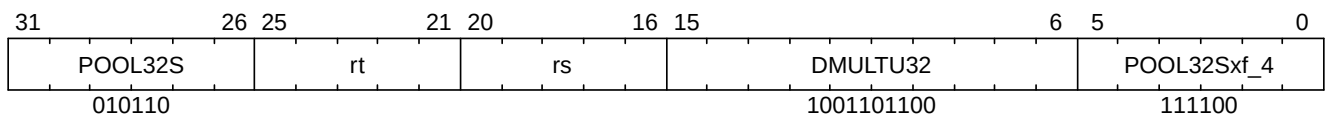


Figure 530. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=None, rs=rs, rt=rt)
```

8.198.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_unsigned_multiply_hilo(ctx, operands)
```

8.198.4. Description

The 128-bit product is split between the 64-bit HI and LO registers.

8.198.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.199. DMULU

8.199.1. Purpose

Multiply unsigned 64-bit GPR values.

8.199.2. Formats

MIPS

dmulu.r6

Assembly

DMULU {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

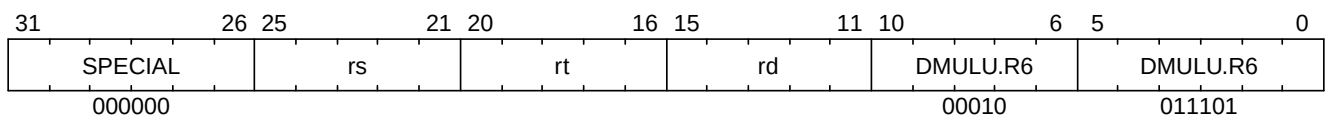


Figure 531. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

dmulu.r6.32

Assembly

DMULU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && MMAR >= R6

Decode priority

0

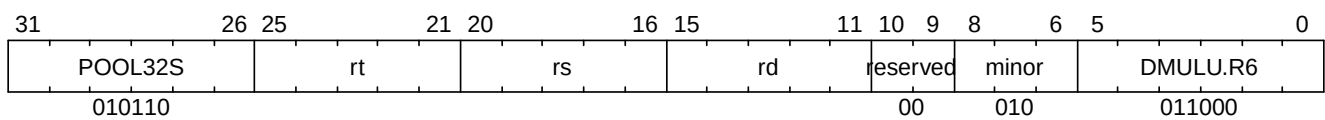


Figure 532. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dmulu.32

Assembly

DMULU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

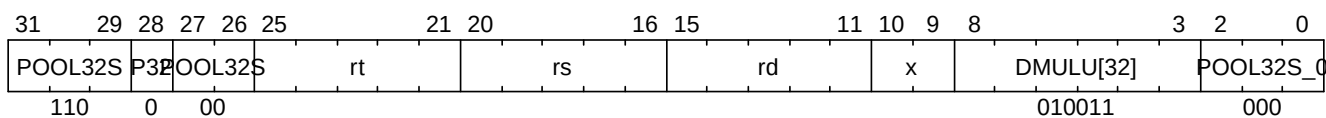


Figure 533. Encoding (32 bits)

Decode

```
return DoubleMultiplyDivideOperands(rd=rd, rs=rs, rt=rt)
```

8.199.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_unsigned_multiply_low(ctx, operands)
```

8.199.4. Description

The low 64 bits of the product are written to rd.

8.199.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.200. DPA.W.PH

8.200.1. Purpose

DSP multiply or accumulate operation DPA.W.PH.

8.200.2. Formats

MIPS

dpa.w.ph

Assembly

DPA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

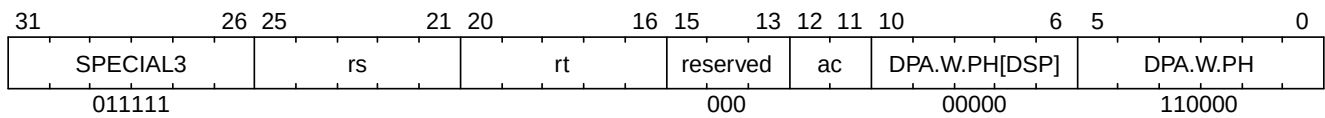


Figure 534. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpa.w.ph.32

Assembly

DPA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

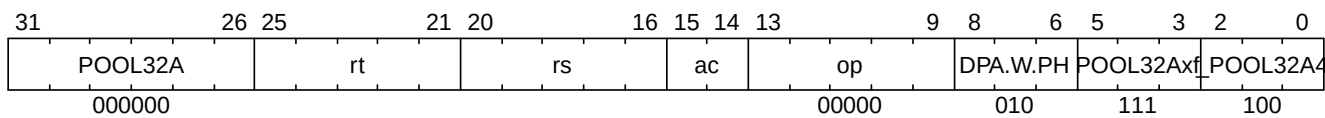


Figure 535. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpa.w.ph.32

Assembly

DPA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

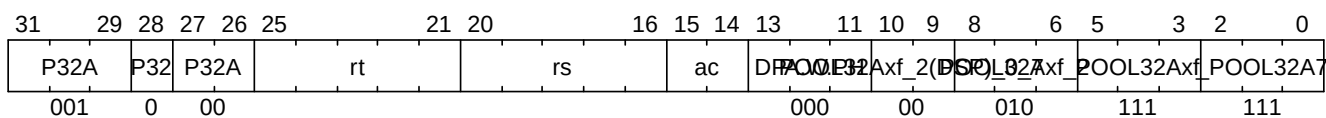


Figure 536. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.200.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_accumulator(
    ctx,
    operands.ac,
    accumulator_value(ctx, operands.ac) + half_products(ctx, operands),
)
```

8.200.4. Description

DPA.W.PH is a DSP multiply or accumulator operation.

8.200.5. Exceptions

None.

8.201. DPAQ_S.W.PH

8.201.1. Purpose

DSP multiply or accumulate operation DPAQ_S.W.PH.

8.201.2. Formats

MIPS

dpaq_s.w.ph

Assembly

DPAQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

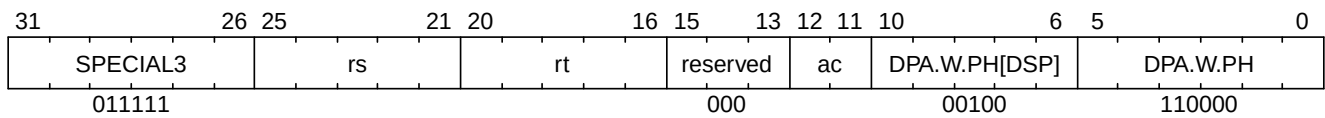


Figure 537. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpaq_s.w.ph.32

Assembly

DPAQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

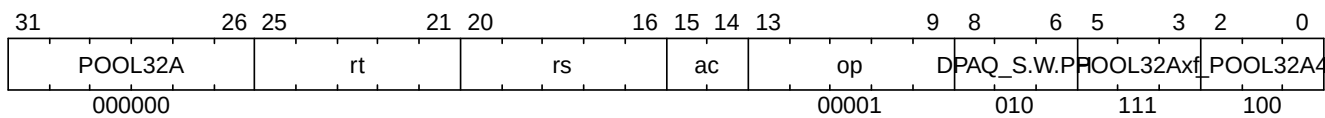


Figure 538. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpaq_s.w.ph.32

Assembly

DPAQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

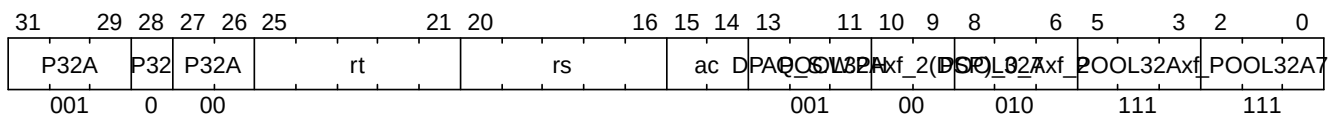


Figure 539. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.201.3. Operation

```
validate_access_to_dsp_resources(ctx)
dotp, overflow = q15_dot(ctx, operands)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, accumulator_value(ctx, operands.ac) + dotp)
```

8.201.4. Description

DPAQ_S.W.PH is a DSP multiply or accumulator operation.

8.201.5. Exceptions

None.

8.202. DPAQ_SA.L.W

8.202.1. Purpose

DSP multiply or accumulate operation DPAQ_SA.L.W.

8.202.2. Formats

MIPS

dpaq_sa.l.w

Assembly

DPAQ_SA.L.W {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

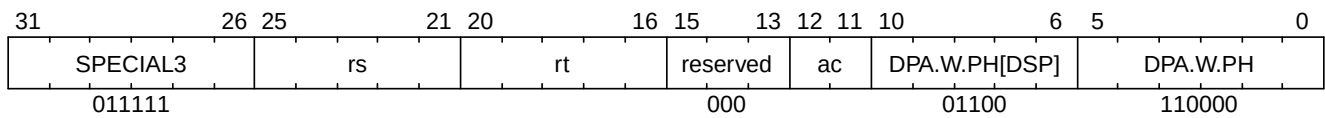


Figure 540. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpaq_sa.l.w.32

Assembly

DPAQ_SA.L.W {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

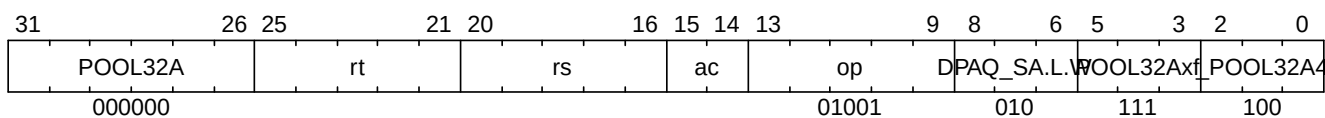


Figure 541. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpaq_sa.l.w.32

Assembly

DPAQ_SA.L.W {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

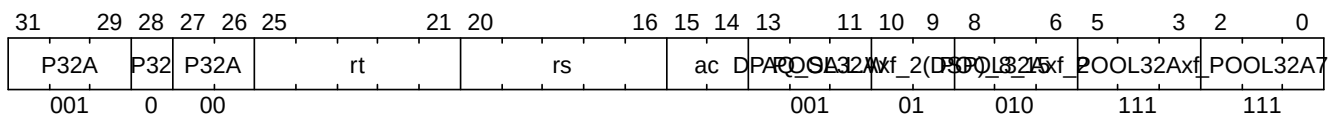


Figure 542. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.202.3. Operation

```
validate_access_to_dsp_resources(ctx)
product, overflow = product_q31(ctx.GPR[operands.rs], ctx.GPR[operands.rt])
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
result, saturated = saturate_accumulator(
    signed(accumulator_value(ctx, operands.ac), 64) + signed(product, 64)
)
if saturated:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, result)
```

8.202.4. Description

DPAQ_SA.L.W is a DSP multiply or accumulator operation.

8.202.5. Exceptions

None.

8.203. DPAQX_S.W.PH

8.203.1. Purpose

DSP multiply or accumulate operation DPAQX_S.W.PH.

8.203.2. Formats

MIPS

dpaqx_s.w.ph

Assembly

DPAQX_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

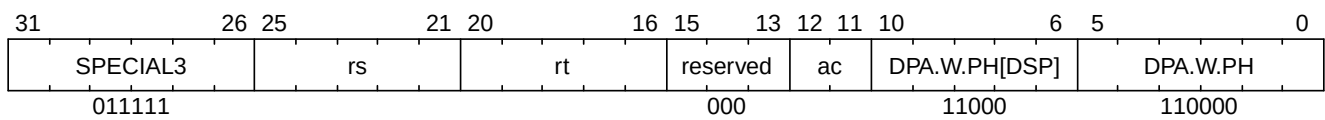


Figure 543. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpaqx_s.w.ph.32

Assembly

DPAQX_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

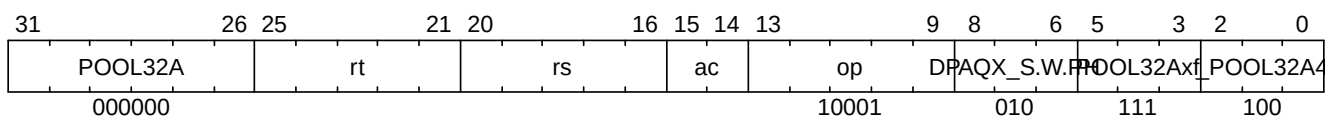


Figure 544. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpaqx_s.w.ph.32

Assembly

DPAQX_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

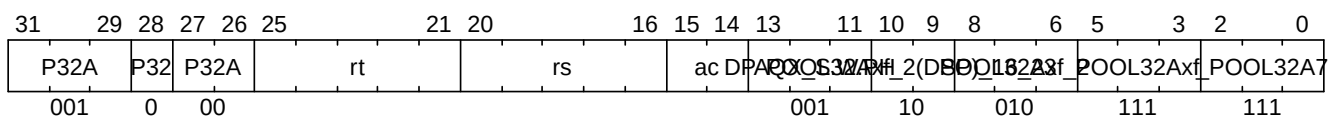


Figure 545. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.203.3. Operation

```
validate_access_to_dsp_resources(ctx)
dotp, overflow = q15_dot(ctx, operands, cross=True)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, accumulator_value(ctx, operands.ac) + dotp)
```

8.203.4. Description

DPAQX_S.W.PH is a DSP multiply or accumulator operation.

8.203.5. Exceptions

None.

8.204. DPAQX_SA.W.PH

8.204.1. Purpose

DSP multiply or accumulate operation DPAQX_SA.W.PH.

8.204.2. Formats

MIPS

dpaqx_sa.w.ph

Assembly

DPAQX_SA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

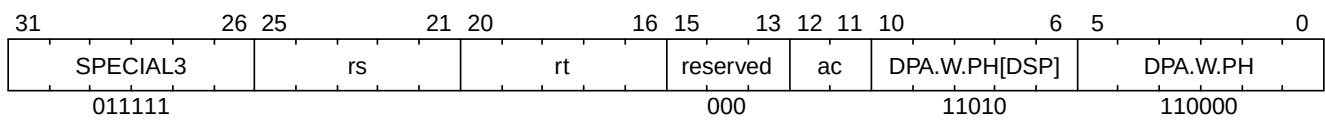


Figure 546. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpaqx_sa.w.ph.32

Assembly

DPAQX_SA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

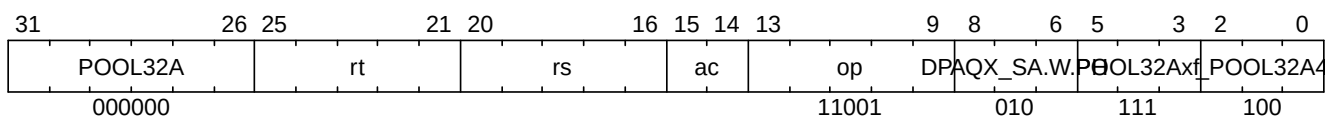


Figure 547. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpaqx_sa.w.ph.32

Assembly

DPAQX_SA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

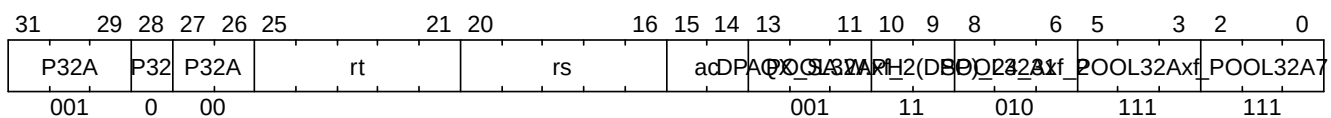


Figure 548. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.204.3. Operation

```
validate_access_to_dsp_resources(ctx)
dotp, overflow = q15_dot(ctx, operands, cross=True)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
result, saturated = saturate_accumulator_q31(
    signed(accumulator_value(ctx, operands.ac), 64) + dotp
)
if saturated:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, result)
```

8.204.4. Description

DPAQX_SA.W.PH is a DSP multiply or accumulator operation.

8.204.5. Exceptions

None.

8.205. DPAU.H.QBL

8.205.1. Purpose

DSP multiply or accumulate operation DPAU.H.QBL.

8.205.2. Formats

MIPS

dpau.h.qbl

Assembly

DPAU.H.QBL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

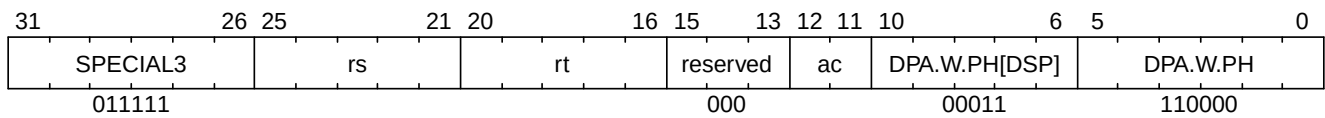


Figure 549. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpau.h.qbl.32

Assembly

DPAU.H.QBL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

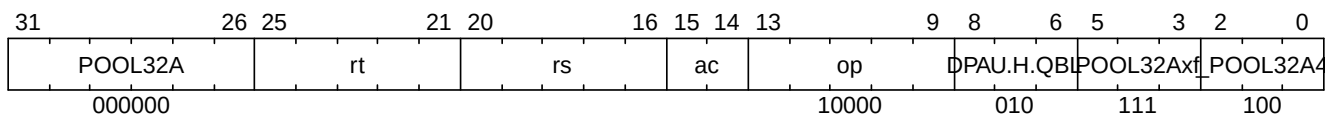


Figure 550. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

`dpau.h.qbl.32`

Assembly

`DPAU.H.QBL {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

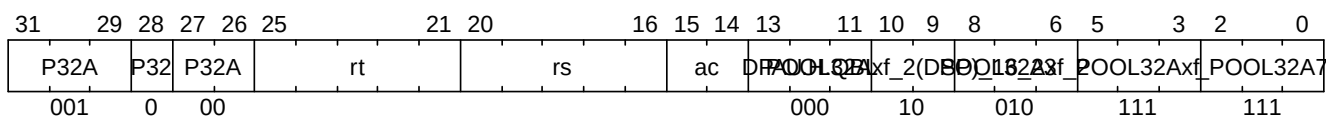


Figure 551. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.205.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_accumulator(
    ctx, operands.ac, accumulator_value(ctx, operands.ac) + byte_dot(ctx, operands)
)
```

8.205.4. Description

DPAU.H.QBL is a DSP multiply or accumulator operation.

8.205.5. Exceptions

None.

8.206. DPAU.H.QBR

8.206.1. Purpose

DSP multiply or accumulate operation DPAU.H.QBR.

8.206.2. Formats

MIPS

`dpau.h.qbr`

Assembly

`DPAU.H.QBR {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

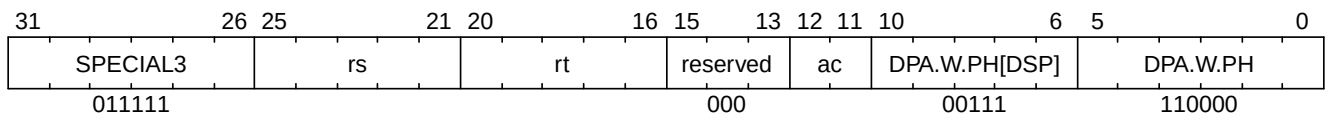


Figure 552. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

`dpau.h.qbr.32`

Assembly

`DPAU.H.QBR {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

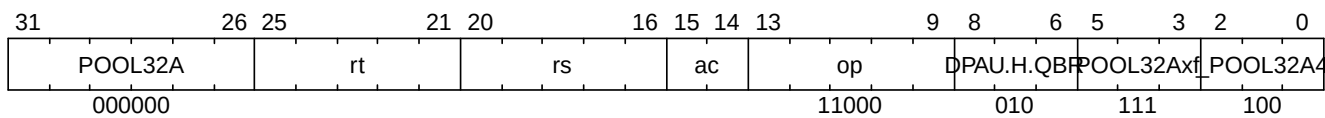


Figure 553. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

`dpau.h.qbr.32`

Assembly

`DPAU.H.QBR {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

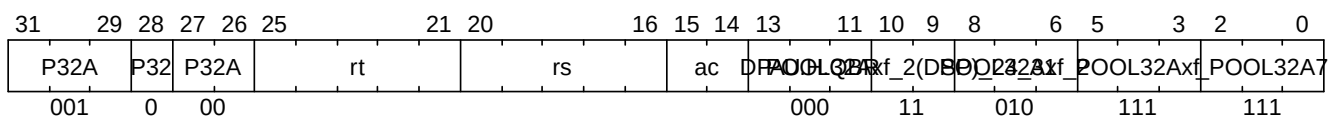


Figure 554. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.206.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_accumulator(
    ctx,
    operands.ac,
    accumulator_value(ctx, operands.ac) + byte_dot(ctx, operands, right=True),
)
```

8.206.4. Description

DPAU.H.QBR is a DSP multiply or accumulator operation.

8.206.5. Exceptions

None.

8.207. DPAX.W.PH

8.207.1. Purpose

DSP multiply or accumulate operation DPAX.W.PH.

8.207.2. Formats

MIPS

dpax.w.ph

Assembly

DPAX.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

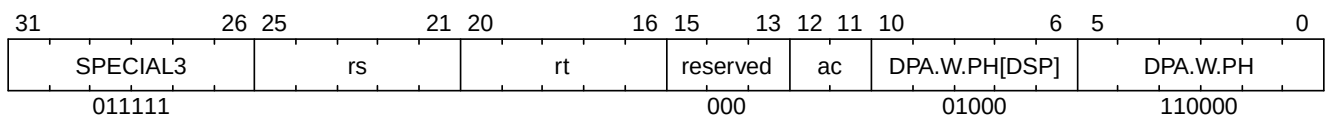


Figure 555. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpax.w.ph.32

Assembly

DPAX.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

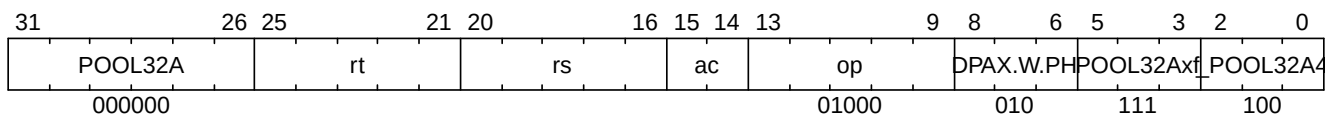


Figure 556. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpax.w.ph.32

Assembly

DPAX.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

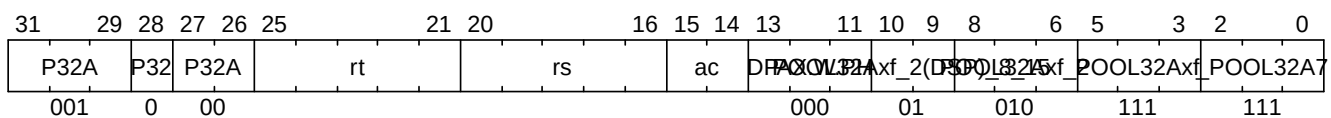


Figure 557. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.207.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_accumulator(
    ctx,
    operands.ac,
    accumulator_value(ctx, operands.ac) + half_products(ctx, operands, cross=True),
)
```

8.207.4. Description

DPAX.W.PH is a DSP multiply or accumulator operation.

8.207.5. Exceptions

None.

8.208. DPS.W.PH

8.208.1. Purpose

DSP multiply or accumulate operation DPS.W.PH.

8.208.2. Formats

MIPS

dps.w.ph

Assembly

DPS.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

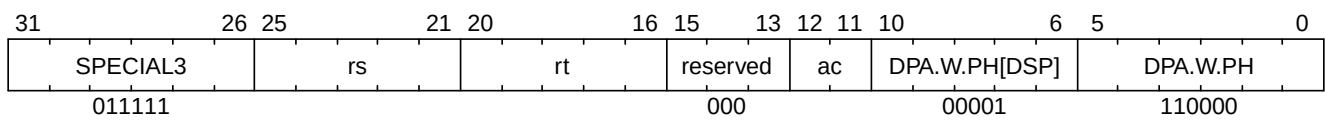


Figure 558. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dps.w.ph.32

Assembly

DPS.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

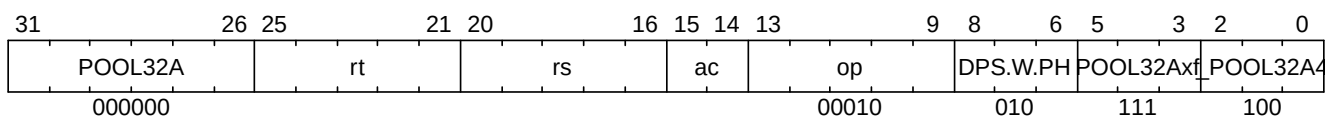


Figure 559. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dps.w.ph.32

Assembly

DPS.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

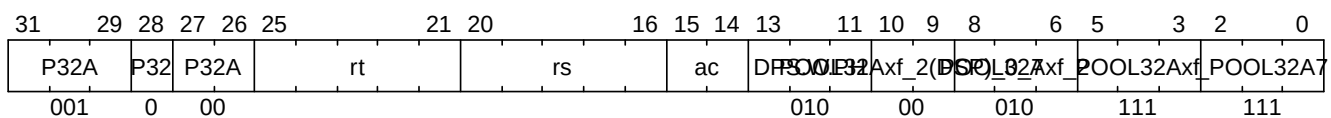


Figure 560. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.208.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_accumulator(
    ctx,
    operands.ac,
    accumulator_value(ctx, operands.ac) - half_products(ctx, operands),
)
```

8.208.4. Description

DPS.W.PH is a DSP multiply or accumulator operation.

8.208.5. Exceptions

None.

8.209. DPSQ_S.W.PH

8.209.1. Purpose

DSP multiply or accumulate operation DPSQ_S.W.PH.

8.209.2. Formats

MIPS

dpsq_s.w.ph

Assembly

DPSQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

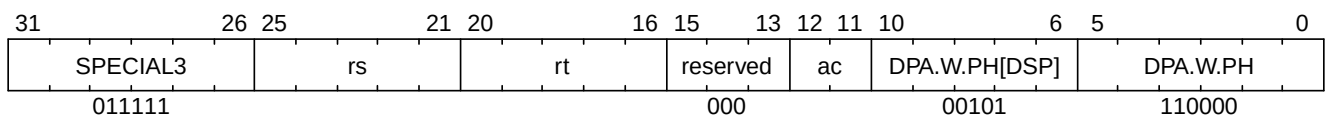


Figure 561. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpsq_s.w.ph.32

Assembly

DPSQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

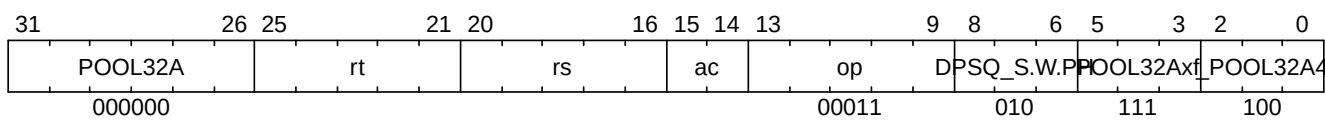


Figure 562. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpsq_s.w.ph.32

Assembly

DPSQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

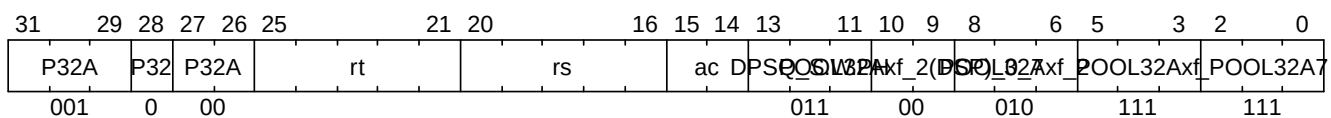


Figure 563. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.209.3. Operation

```
validate_access_to_dsp_resources(ctx)
dotp, overflow = q15_dot(ctx, operands)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, accumulator_value(ctx, operands.ac) - dotp)
```

8.209.4. Description

DPSQ_S.W.PH is a DSP multiply or accumulator operation.

8.209.5. Exceptions

None.

8.210. DPSQ_SA.L.W

8.210.1. Purpose

DSP multiply or accumulate operation DPSQ_SA.L.W.

8.210.2. Formats

MIPS

dpsq_sa.l.w

Assembly

DPSQ_SA.L.W {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

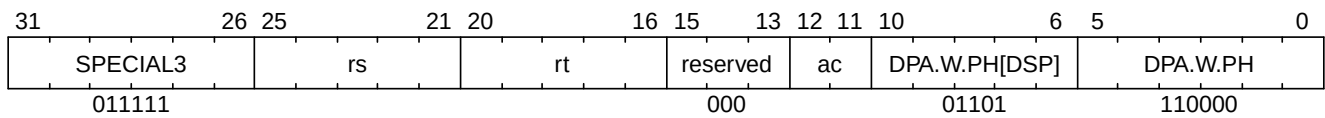


Figure 564. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpsq_sa.l.w.32

Assembly

DPSQ_SA.L.W {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

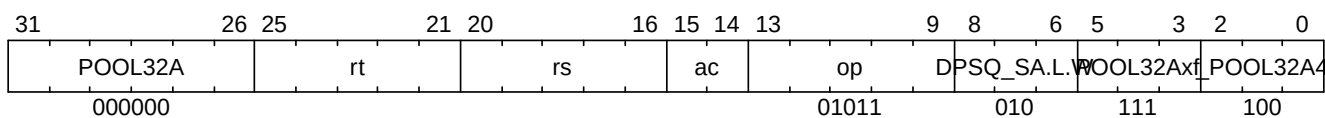


Figure 565. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpsq_sa.l.w.32

Assembly

DPSQ_SA.L.W {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

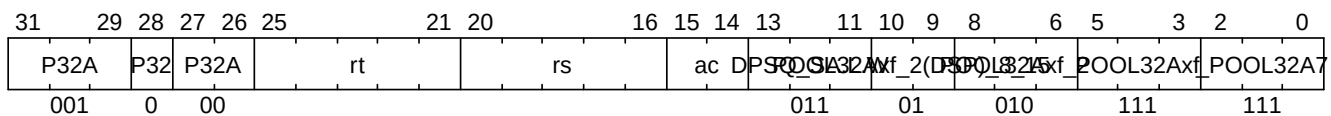


Figure 566. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.210.3. Operation

```
validate_access_to_dsp_resources(ctx)
product, overflow = product_q31(ctx.GPR[operands.rs], ctx.GPR[operands.rt])
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
result, saturated = saturate_accumulator(
    signed(accumulator_value(ctx, operands.ac), 64) - signed(product, 64)
)
if saturated:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, result)
```

8.210.4. Description

DPSQ_SA.L.W is a DSP multiply or accumulator operation.

8.210.5. Exceptions

None.

8.211. DPSQX_S.W.PH

8.211.1. Purpose

DSP multiply or accumulate operation DPSQX_S.W.PH.

8.211.2. Formats

MIPS

dpsqx_s.w.ph

Assembly

DPSQX_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

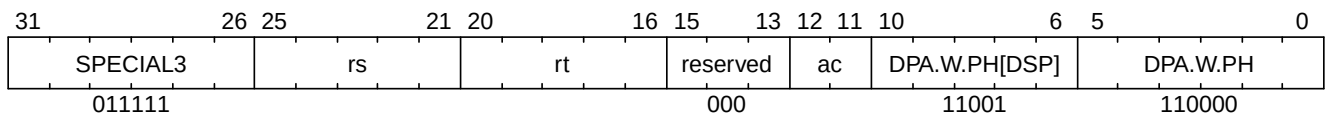


Figure 567. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpsqx_s.w.ph.32

Assembly

DPSQX_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

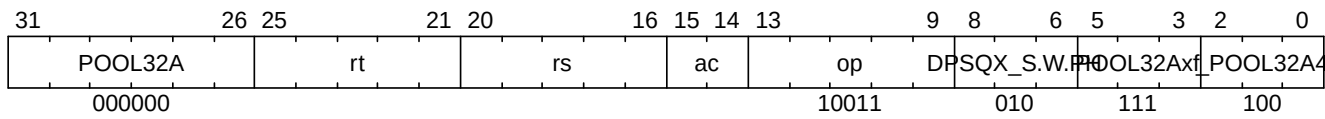


Figure 568. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpsqx_s.w.ph.32

Assembly

DPSQX_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

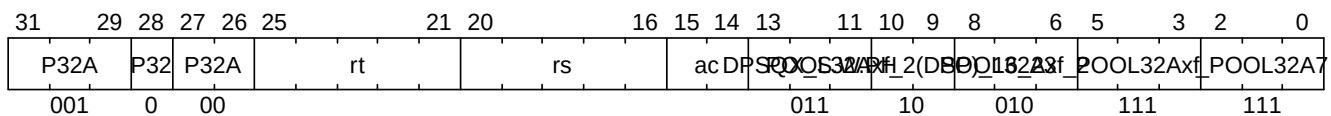


Figure 569. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.211.3. Operation

```
validate_access_to_dsp_resources(ctx)
dotp, overflow = q15_dot(ctx, operands, cross=True)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, accumulator_value(ctx, operands.ac) - dotp)
```

8.211.4. Description

DPSQX_S.W.PH is a DSP multiply or accumulator operation.

8.211.5. Exceptions

None.

8.212. DPSQX_SA.W.PH

8.212.1. Purpose

DSP multiply or accumulate operation DPSQX_SA.W.PH.

8.212.2. Formats

MIPS

dpsqx_sa.w.ph

Assembly

DPSQX_SA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

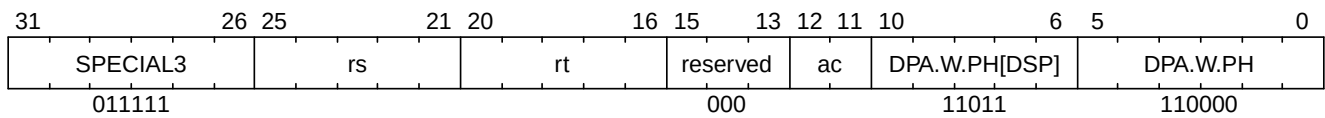


Figure 570. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpsqx_sa.w.ph.32

Assembly

DPSQX_SA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

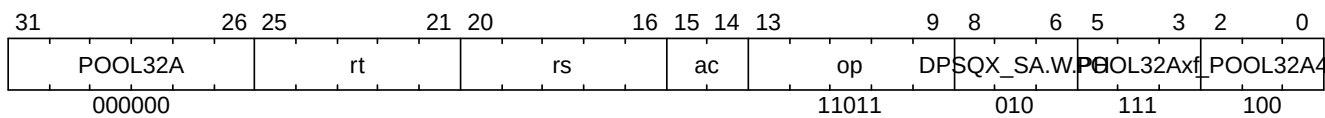


Figure 571. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpsqx_sa.w.ph.32

Assembly

DPSQX_SA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

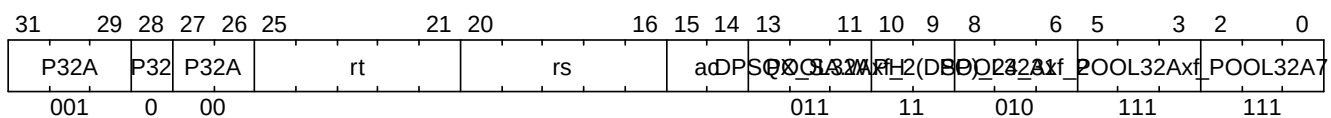


Figure 572. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.212.3. Operation

```
validate_access_to_dsp_resources(ctx)
dotp, overflow = q15_dot(ctx, operands, cross=True)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
result, saturated = saturate_accumulator_q31(
    signed(accumulator_value(ctx, operands.ac), 64) - dotp
)
if saturated:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, result)
```

8.212.4. Description

DPSQX_SA.W.PH is a DSP multiply or accumulator operation.

8.212.5. Exceptions

None.

8.213. DPSU.H.QBL

8.213.1. Purpose

DSP multiply or accumulate operation DPSU.H.QBL.

8.213.2. Formats

MIPS

dpsu.h.qbl

Assembly

DPSU.H.QBL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

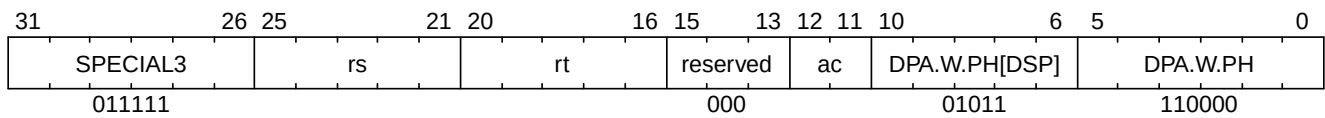


Figure 573. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpsu.h.qbl.32

Assembly

DPSU.H.QBL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

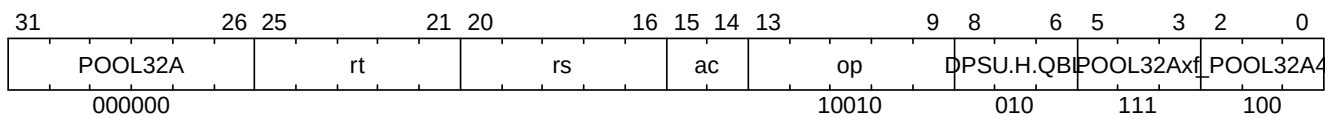


Figure 574. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpsu.h.qbl.32

Assembly

DPSU.H.QBL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

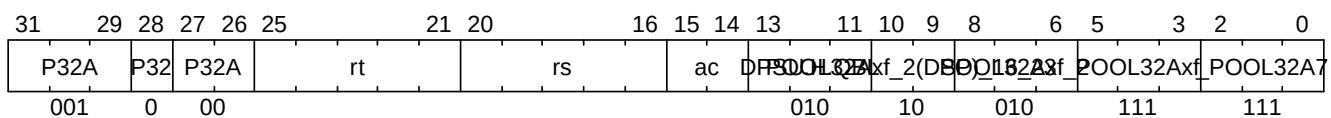


Figure 575. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.213.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_accumulator(
    ctx, operands.ac, accumulator_value(ctx, operands.ac) - byte_dot(ctx, operands)
)
```

8.213.4. Description

DPSU.H.QBL is a DSP multiply or accumulator operation.

8.213.5. Exceptions

None.

8.214. DPSU.H.QBR

8.214.1. Purpose

DSP multiply or accumulate operation DPSU.H.QBR.

8.214.2. Formats

MIPS

dpsu.h.qbr

Assembly

DPSU.H.QBR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

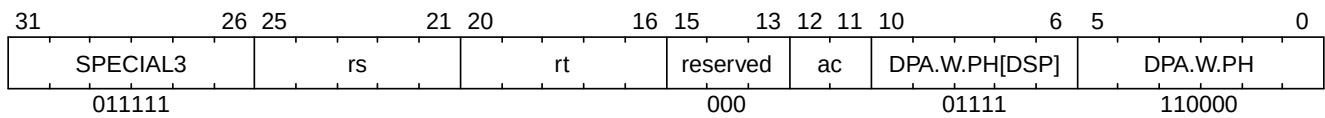


Figure 576. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpsu.h.qbr.32

Assembly

DPSU.H.QBR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

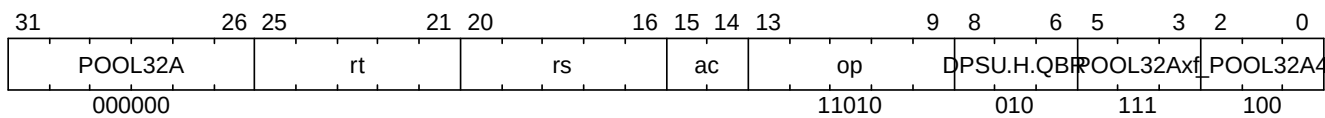


Figure 577. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpsu.h.qbr.32

Assembly

DPSU.H.QBR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

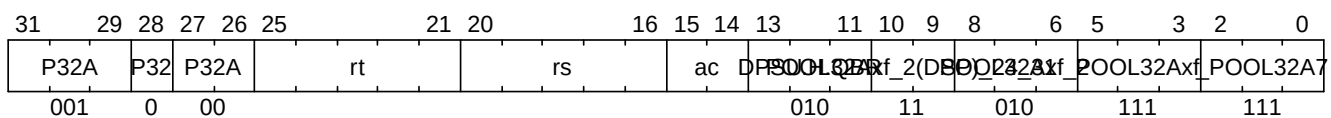


Figure 578. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.214.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_accumulator(
    ctx,
    operands.ac,
    accumulator_value(ctx, operands.ac) - byte_dot(ctx, operands, right=True),
)
```

8.214.4. Description

DPSU.H.QBR is a DSP multiply or accumulator operation.

8.214.5. Exceptions

None.

8.215. DPSX.W.PH

8.215.1. Purpose

DSP multiply or accumulate operation DPSX.W.PH.

8.215.2. Formats

MIPS

dpsx.w.ph

Assembly

DPSX.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

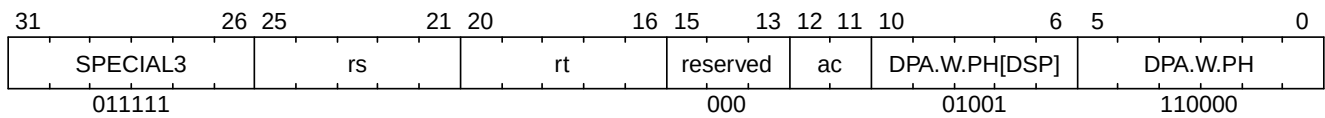


Figure 579. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

dpsx.w.ph.32

Assembly

DPSX.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

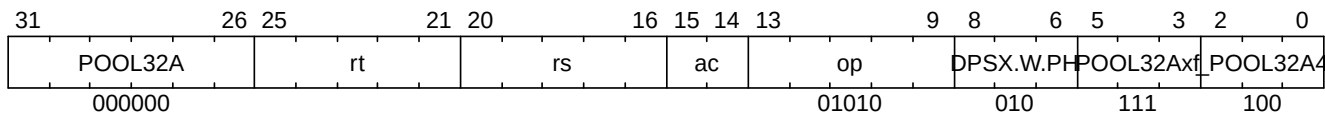


Figure 580. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

dpsx.w.ph.32

Assembly

DPSX.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

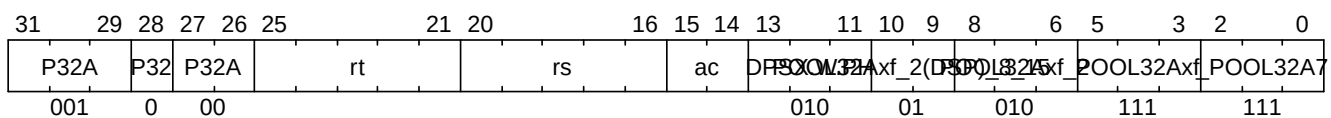


Figure 581. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.215.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_accumulator(
    ctx,
    operands.ac,
    accumulator_value(ctx, operands.ac) - half_products(ctx, operands, cross=True),
)
```

8.215.4. Description

DPSX.W.PH is a DSP multiply or accumulator operation.

8.215.5. Exceptions

None.

8.216. DROTR

8.216.1. Purpose

DROTR

8.216.2. Formats

MIPS

`drotr`

Assembly

`DROTR {rs:gpr}, {shift:u6}`

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

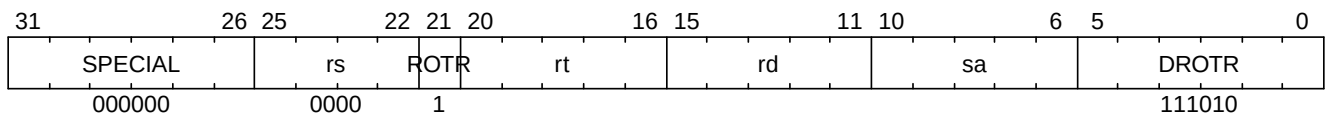


Figure 582. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa, from_nbits=5))
```

microMIPS

`drotr`

Assembly

`DROTR {rd:gpr}, {rs:gpr}, {shift:u6}`

Availability

AT >= MIPS64_32

Decode priority

0

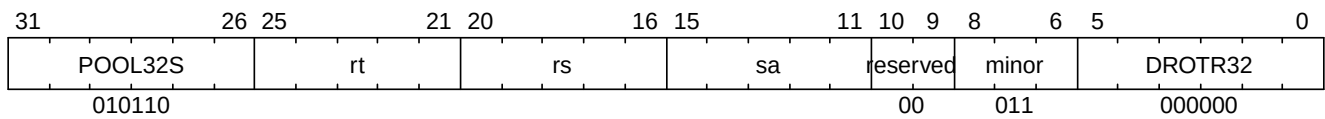


Figure 583. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa, from_nbits=5))
```

nanoMIPS

drotr.32

Assembly

DROTR {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

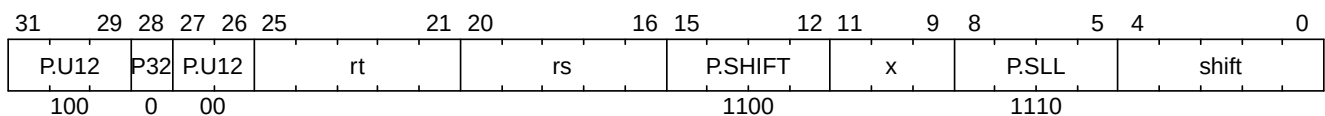


Figure 584. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(shift, from_nbits=5))
```

8.216.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="right", rotate=True)
```

8.216.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.216.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.217. DROTR32

8.217.1. Purpose

DROTR32

8.217.2. Formats

MIPS

`drotr32`

Assembly

`DROTR32 {rs:gpr}, {shift:u6}`

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

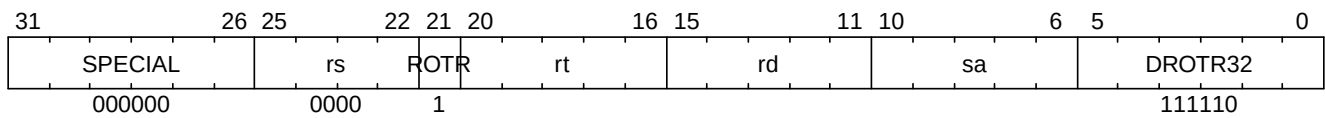


Figure 585. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa + 32, from_nbits=6))
```

microMIPS

`drotr32`

Assembly

`DROTR32 {rd:gpr}, {rs:gpr}, {shift:u6}`

Availability

AT >= MIPS64_32

Decode priority

0

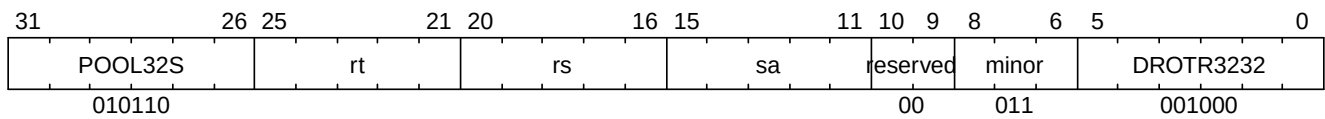


Figure 586. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa + 32, from_nbits=6))
```

nanoMIPS

drotr32.32

Assembly

```
DROTR32 {rd:gpr}, {rs:gpr}, {shift:u6}
```

Availability

AT >= MIPS64_32

Decode priority

0

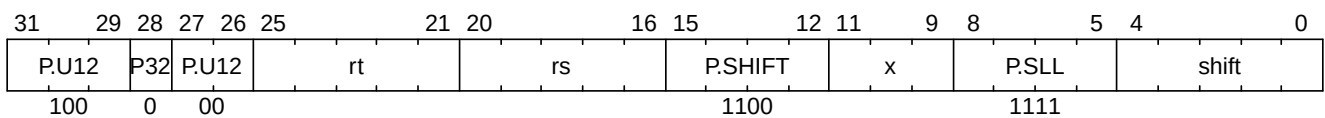


Figure 587. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(
    rd=rt, rs=rs, shift=zero_extend(shift + 32, from_nbits=6)
)
```

8.217.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="right", rotate=True)
```

8.217.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.217.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.218. DROTRV

8.218.1. Purpose

DROTRV

8.218.2. Formats

MIPS

drotrv

Assembly

DROTRV {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

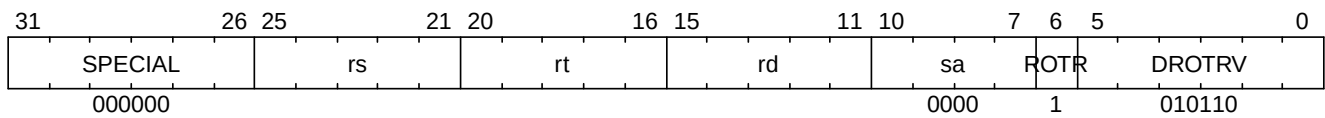


Figure 588. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

microMIPS

drotrv

Assembly

DROTRV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

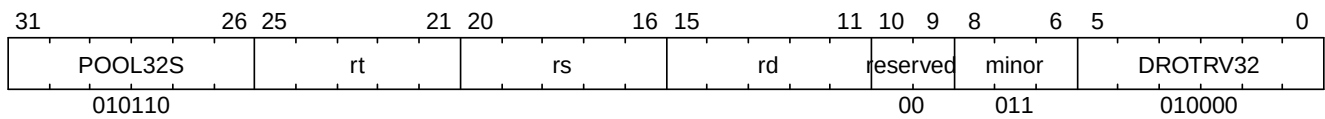


Figure 589. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

drotrv.32

Assembly

DROTRV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

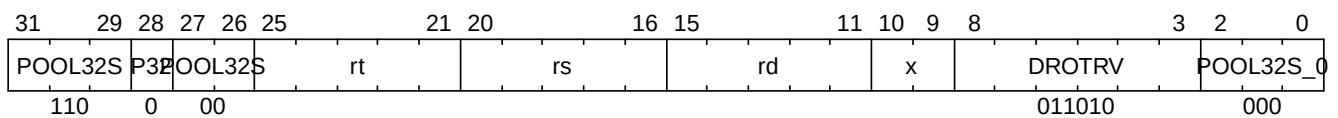


Figure 590. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rs, rt=rt)
```

8.218.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="right", rotate=True)
```

8.218.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.218.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.219. DROTX

8.219.1. Purpose

Rotate and exchange bits in a 64-bit GPR.

8.219.2. Formats

nanoMIPS

drotx.32

Assembly

```
DROTX {rt:gpr}, {rs:gpr}, {shift:u6}, {shiftx:u6}, {stripe:u1}, {flip:u1}
```

Availability

AT >= MIPS64_32

Decode priority

0

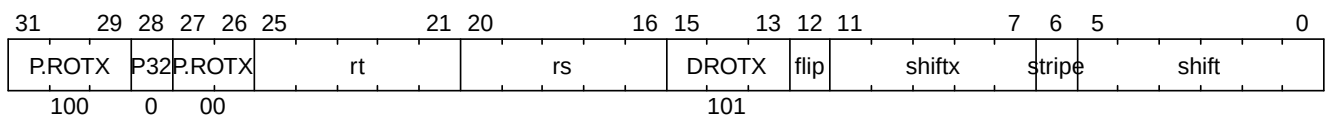


Figure 591. Encoding (32 bits)

Decode

```
shift = zero_extend(shift, from_nbits=6)
shiftx = zero_extend(shiftx << 1, from_nbits=6)
return Drotx0Operands(
    rt=rt, rs=rs, shift=shift, shiftx=shiftx, stripe=stripe, flip=flip
)
```

8.219.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_rotate_extended(ctx, operands)
```

8.219.4. Description

DROTX implements the staged bit exchange network specified by the nanoMIPS64 architecture.

8.219.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.220. DSBH

8.220.1. Purpose

Swap bytes within each 16-bit halfword.

8.220.2. Formats

MIPS

dsbh

Assembly

DSBH {rs:gpr}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

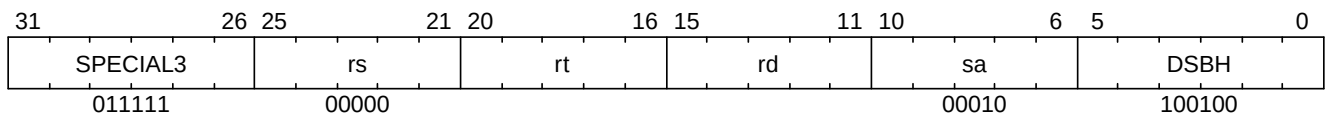


Figure 592. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitOrderOperands(rt=rd, rs=rt)
```

microMIPS

dsbh.32

Assembly

DSBH {rt:gpr}, {rs:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

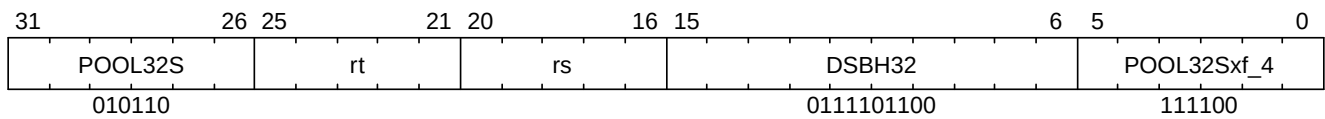


Figure 593. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitOrderOperands(rt=rt, rs=rs)
```

8.220.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_byte_order(ctx, operands, kind="dsbh")
```

8.220.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.220.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.221. DSHD

8.221.1. Purpose

Swap 16-bit halfwords within a doubleword.

8.221.2. Formats

MIPS

dshd

Assembly

DSHD {rs:gpr}

Availability

AT >= MIPS64_32 && AR >= R2_3_5

Decode priority

0

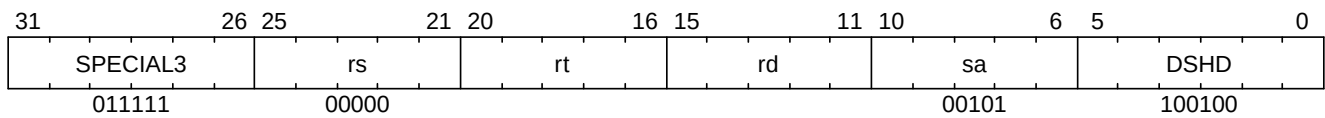


Figure 594. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleBitOrderOperands(rt=rd, rs=rt)
```

microMIPS

dshd.32

Assembly

DSHD {rt:gpr}, {rs:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

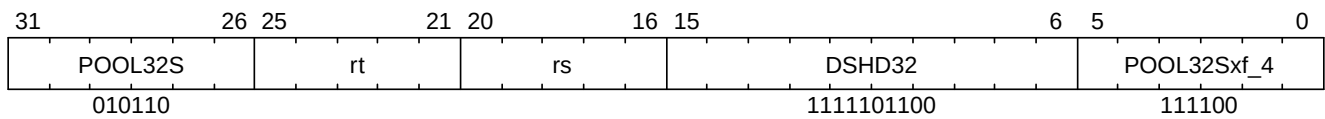


Figure 595. Encoding (32 bits)

Decode

```
del ctx, exec  
return DoubleBitOrderOperands(rt=rt, rs=rs)
```

8.221.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_byte_order(ctx, operands, kind="dshd")
```

8.221.4. Description

The operation uses 64-bit GPR bitfields and is unavailable when 64-bit operations are disabled.

8.221.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.222. DSLL

8.222.1. Purpose

DSLL

8.222.2. Formats

MIPS

dsll

Assembly

DSLL {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

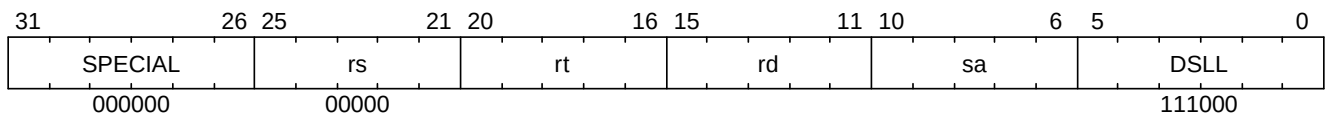


Figure 596. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa, from_nbits=5))
```

microMIPS

dsll

Assembly

DSLL {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

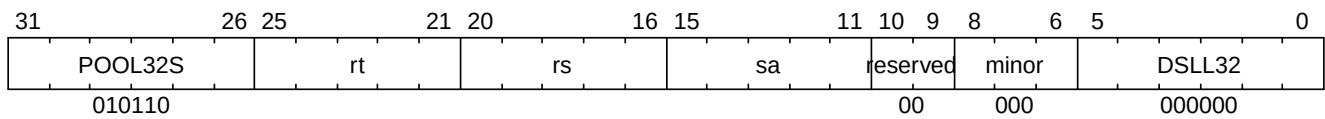


Figure 597. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa, from_nbits=5))
```

nanoMIPS

dsll.32

Assembly

DSLL {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

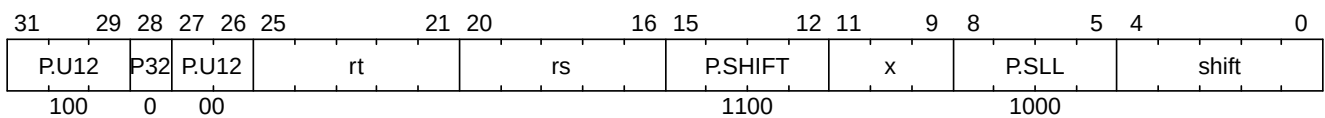


Figure 598. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(shift, from_nbits=5))
```

8.222.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="left")
```

8.222.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.222.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.223. DSLL32

8.223.1. Purpose

DSLL32

8.223.2. Formats

MIPS

dsll32

Assembly

DSLL32 {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

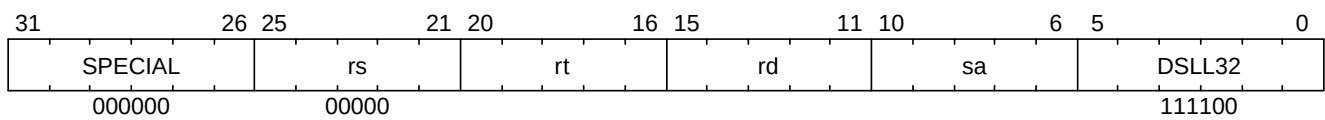


Figure 599. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa + 32, from_nbits=6))
```

microMIPS

dsll32

Assembly

DSLL32 {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

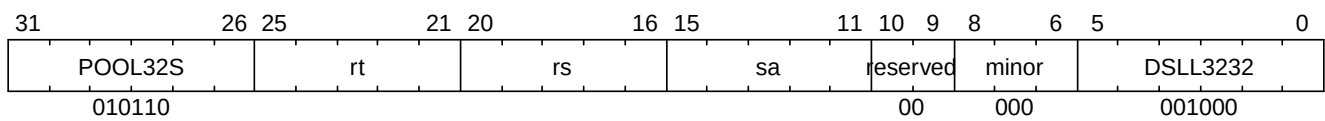


Figure 600. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa + 32, from_nbits=6))
```

nanoMIPS

dsll32.32

Assembly

DSLL32 {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

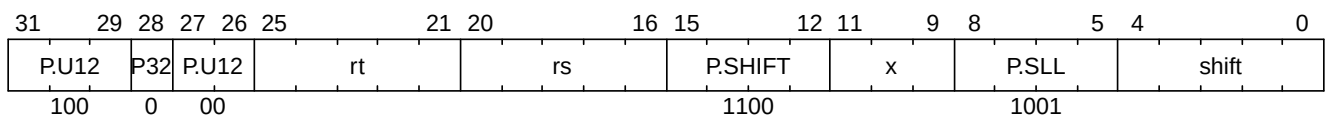


Figure 601. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(
    rd=rt, rs=rs, shift=zero_extend(shift + 32, from_nbits=6)
)
```

8.223.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="left")
```

8.223.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.223.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.224. DSLLV

8.224.1. Purpose

DSLLV

8.224.2. Formats

MIPS

dsllv

Assembly

DSLLV {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

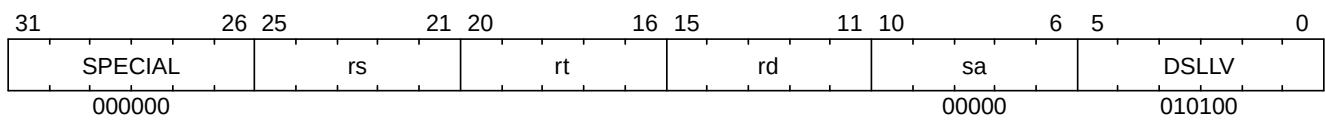


Figure 602. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

microMIPS

dsllv

Assembly

DSLLV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

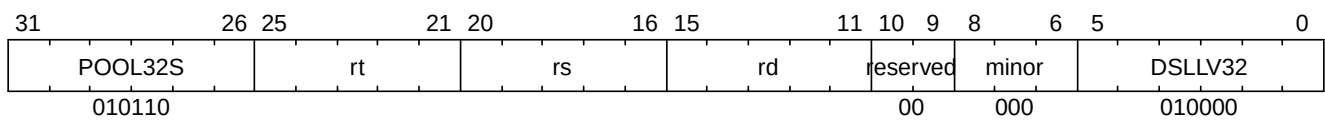


Figure 603. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

dslv.32

Assembly

DSLVLV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

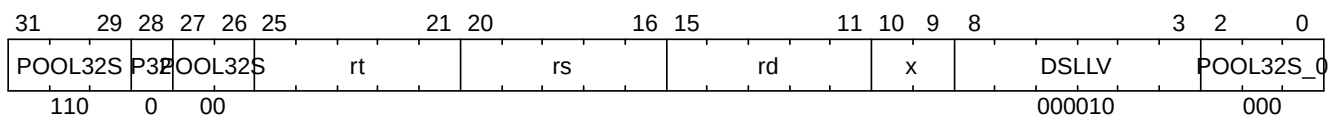


Figure 604. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rs, rt=rt)
```

8.224.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="left")
```

8.224.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.224.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.225. DSRA

8.225.1. Purpose

DSRA

8.225.2. Formats

MIPS

dsra

Assembly

DSRA {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

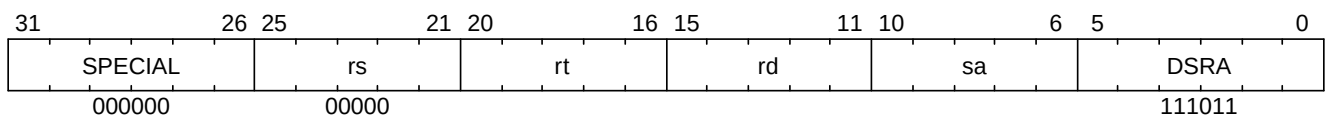


Figure 605. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa, from_nbits=5))
```

microMIPS

dsra

Assembly

DSRA {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

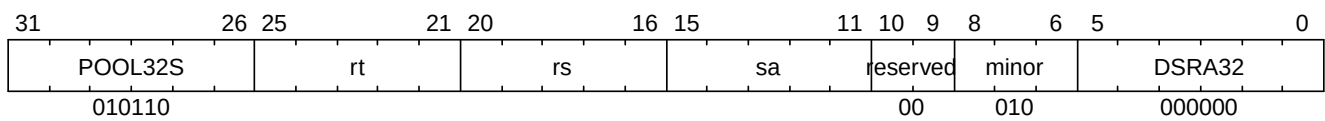


Figure 606. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa, from_nbits=5))
```

nanoMIPS

dsra.32

Assembly

DSRA {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

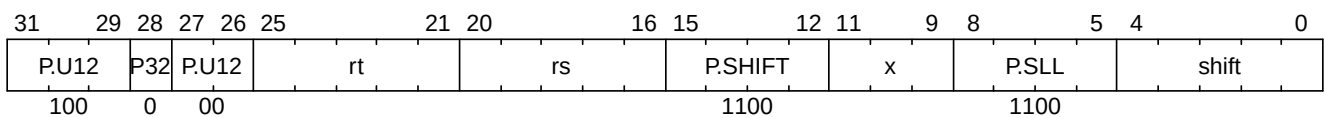


Figure 607. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(shift, from_nbits=5))
```

8.225.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="arithmetic-right")
```

8.225.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.225.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.226. DSRA32

8.226.1. Purpose

DSRA32

8.226.2. Formats

MIPS

dsra32

Assembly

DSRA32 {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

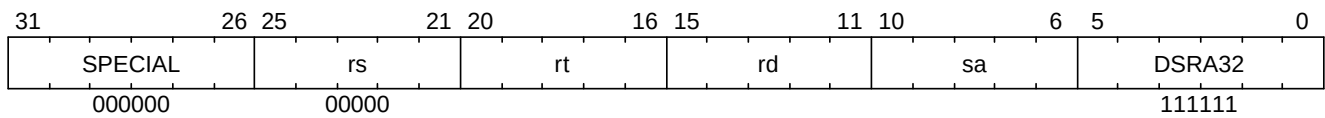


Figure 608. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa + 32, from_nbits=6))
```

microMIPS

dsra32

Assembly

DSRA32 {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

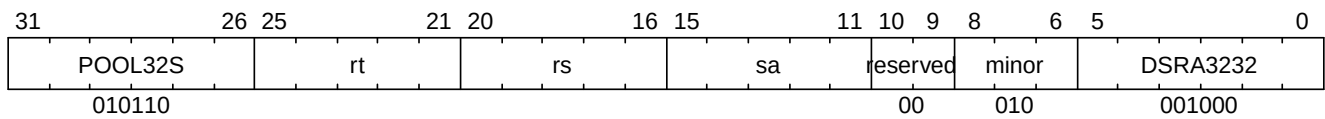


Figure 609. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa + 32, from_nbits=6))
```

nanoMIPS

dsra32.32

Assembly

DSRA32 {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

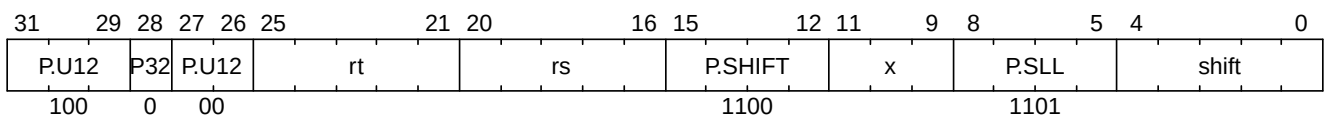


Figure 610. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(
    rd=rt, rs=rs, shift=zero_extend(shift + 32, from_nbits=6)
)
```

8.226.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="arithmetic-right")
```

8.226.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.226.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.227. DSRAV

8.227.1. Purpose

DSRAV

8.227.2. Formats

MIPS

dsrav

Assembly

DSRAV {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

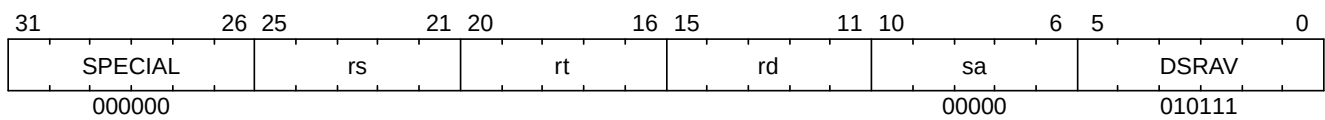


Figure 611. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

microMIPS

dsrav

Assembly

DSRAV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

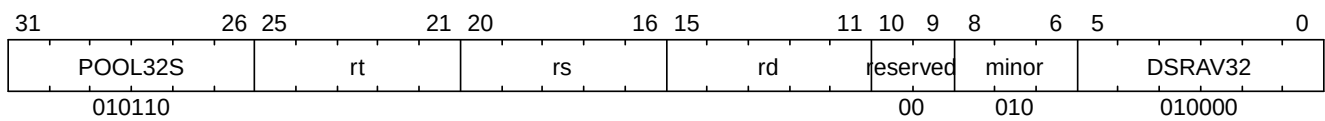


Figure 612. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

dsrav.32

Assembly

DSRAV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

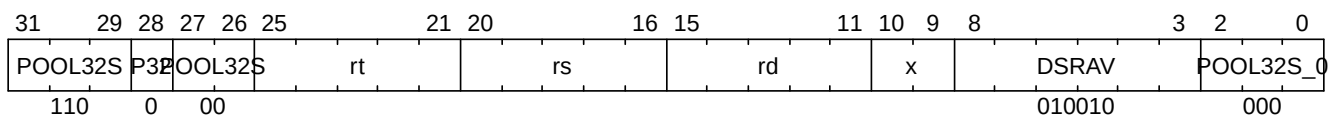


Figure 613. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rs, rt=rt)
```

8.227.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="arithmetic-right")
```

8.227.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.227.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.228. DSRL

8.228.1. Purpose

DSRL

8.228.2. Formats

MIPS

`dsrl`

Assembly

`DSRL {rs:gpr}, {shift:u6}`

Availability

AT >= MIPS64_32

Decode priority

0

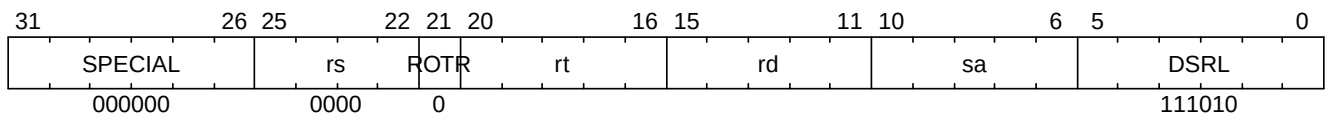


Figure 614. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa, from_nbits=5))
```

microMIPS

`dsrl`

Assembly

`DSRL {rd:gpr}, {rs:gpr}, {shift:u6}`

Availability

AT >= MIPS64_32

Decode priority

0

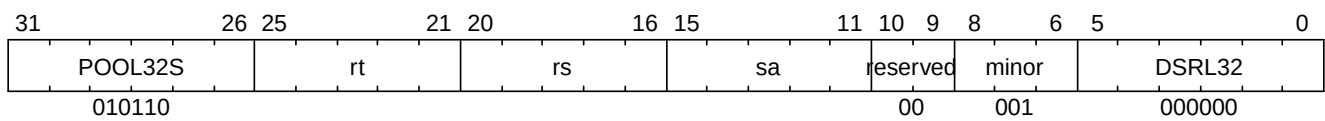


Figure 615. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa, from_nbits=5))
```

nanoMIPS

dsl.32

Assembly

DSRL {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

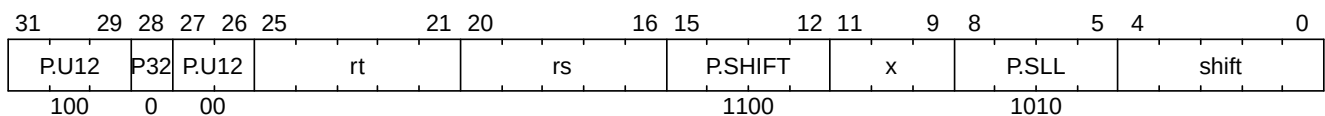


Figure 616. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(shift, from_nbits=5))
```

8.228.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="right")
```

8.228.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.228.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.229. DSRL32

8.229.1. Purpose

DSRL32

8.229.2. Formats

MIPS

dsrl32

Assembly

DSRL32 {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

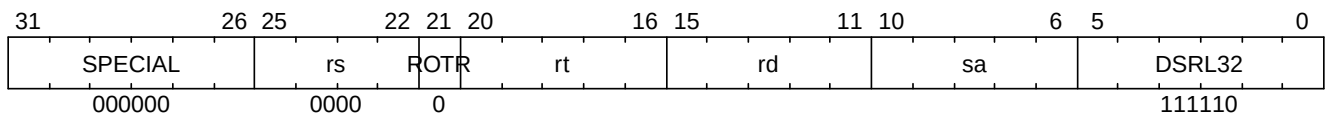


Figure 617. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, shift=zero_extend(sa + 32, from_nbits=6))
```

microMIPS

dsrl32

Assembly

DSRL32 {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

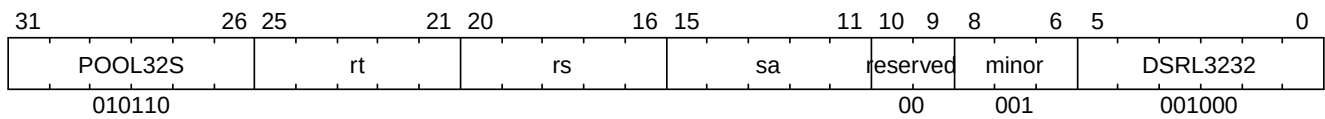


Figure 618. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rt, rs=rs, shift=zero_extend(sa + 32, from_nbits=6))
```

nanoMIPS

dsl32.32

Assembly

DSRL32 {rd:gpr}, {rs:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

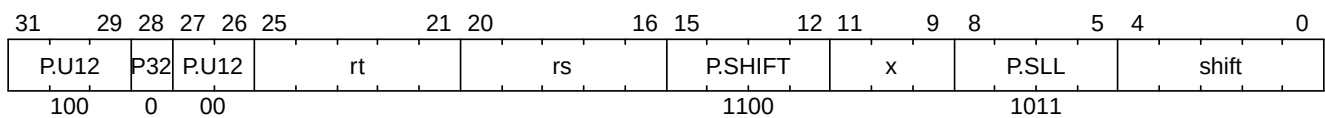


Figure 619. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(
    rd=rt, rs=rs, shift=zero_extend(shift + 32, from_nbits=6)
)
```

8.229.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="right")
```

8.229.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.229.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.230. DSRLV

8.230.1. Purpose

DSRLV

8.230.2. Formats

MIPS

`dsrlv`

Assembly

`DSRLV {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32

Decode priority

0

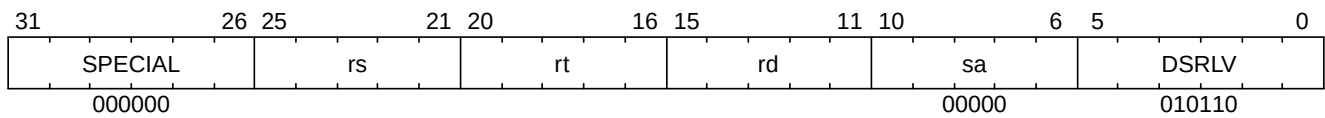


Figure 620. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

microMIPS

`dsrlv`

Assembly

`DSRLV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32

Decode priority

0

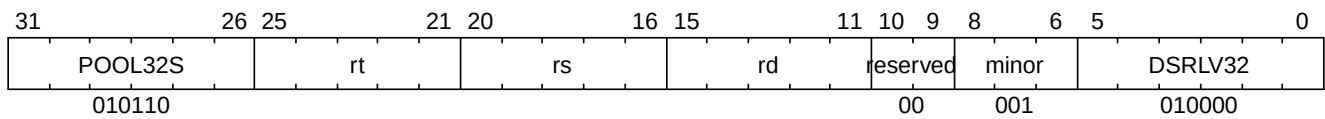


Figure 621. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

dsrlv.32

Assembly

DSRLV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

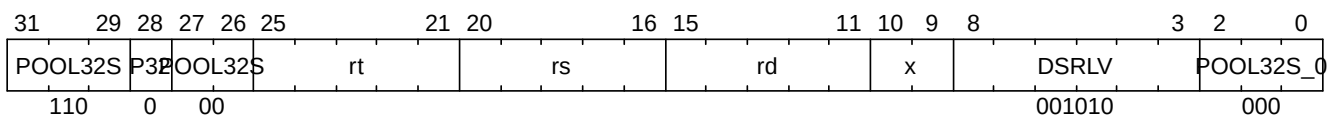


Figure 622. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleShiftOperands(rd=rd, rs=rs, rt=rt)
```

8.230.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_shift(ctx, operands, direction="right")
```

8.230.4. Description

The operation is performed on 64-bit GPR values and the result is sign-extended from 64 bits.

8.230.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.231. DSUB

8.231.1. Purpose

Subtract signed 64-bit GPR values and trap on overflow.

8.231.2. Formats

MIPS

`dsub`

Assembly

`DSUB {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32

Decode priority

0

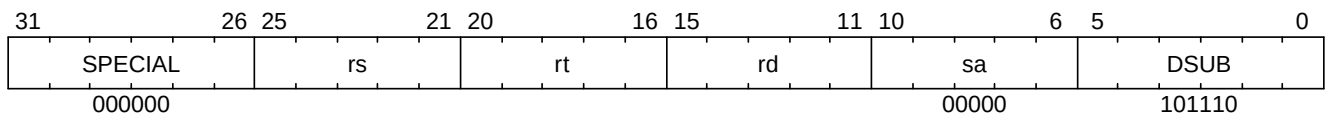


Figure 623. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`dsub`

Assembly

`DSUB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32

Decode priority

0

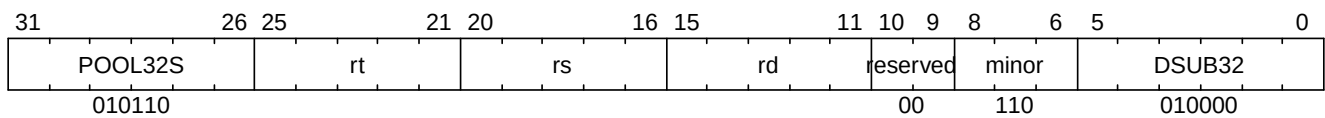


Figure 624. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dsub.32

Assembly

DSUB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

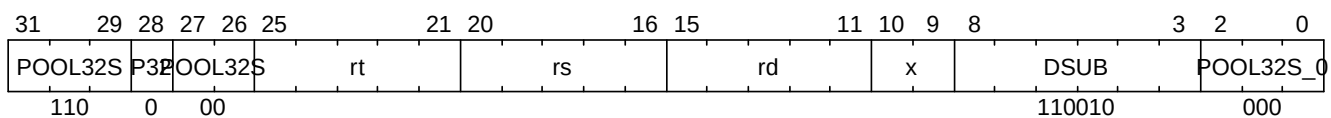


Figure 625. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

8.231.3. Operation

```
validate_access_to_64bit_operations(ctx)
result = double_arithmetic_difference(ctx, operands)
check_signed_doubleword_overflow(result, "DSUB")
write_doubleword(ctx, operands.rd, result)
```

8.231.4. Description

The mathematical signed difference is checked before the 64-bit result is written.

8.231.5. Exceptions

- **OV** (Integer Overflow) — A trapping arithmetic result is not representable at its specified width.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.232. DSUBU

8.232.1. Purpose

Subtract 64-bit GPR values without trapping on overflow.

8.232.2. Formats

MIPS

`dsubu`

Assembly

`DSUBU {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32

Decode priority

0

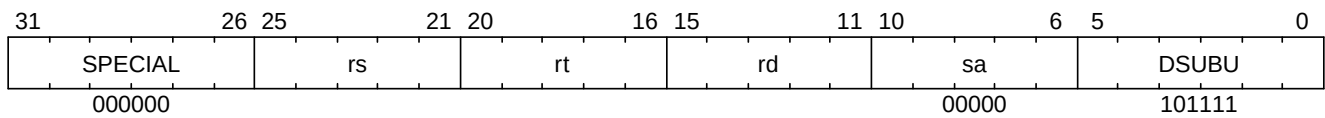


Figure 626. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`dsubu`

Assembly

`DSUBU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AT >= MIPS64_32

Decode priority

0

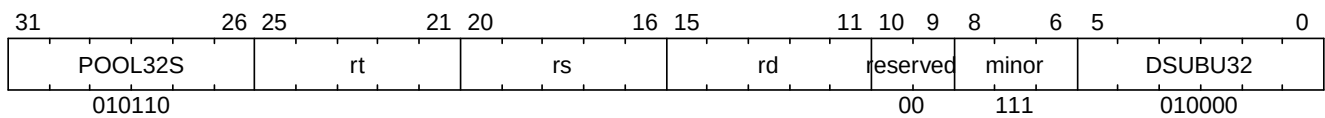


Figure 627. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

dsubu.32

Assembly

DSUBU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AT >= MIPS64_32

Decode priority

0

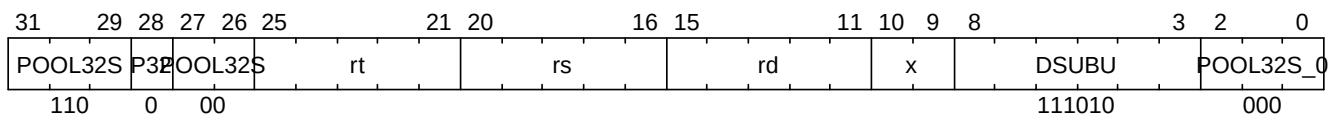


Figure 628. Encoding (32 bits)

Decode

```
return DoubleRegisterOperands(rd=rd, rs=rs, rt=rt)
```

8.232.3. Operation

```
validate_access_to_64bit_operations(ctx)
write_doubleword(ctx, operands.rd, double_arithmetic_difference(ctx, operands))
```

8.232.4. Description

The low 64 bits of the difference are written to the destination GPR.

8.232.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.233. DVP

8.233.1. Purpose

Disable virtual processors.

8.233.2. Formats

MIPS

dpv

Assembly

DVP {rt:gpr}

Availability

AR >= R6

Decode priority

0

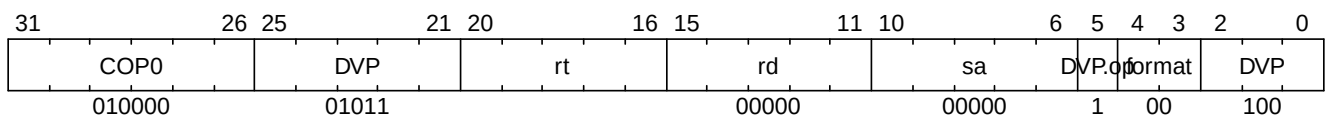


Figure 629. Encoding (32 bits)

Decode

```
return DvpOperands(rt=rt)
```

microMIPS

dpv.32.micromips

Assembly

DVP {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

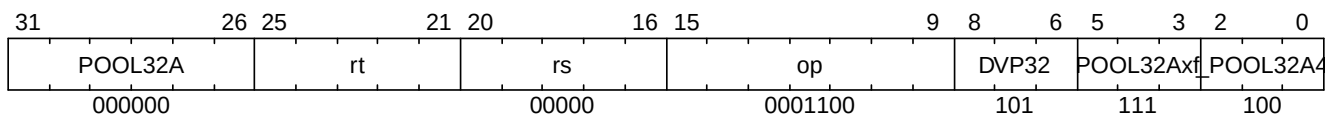


Figure 630. Encoding (32 bits)

Decode

```
return DvpOperands(rt=rt)
```

nanoMIPS

dvb.32

Assembly

DVP {rt:gpr}

Availability

true

Decode priority

0

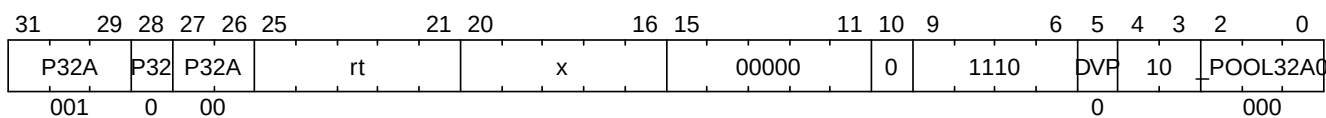


Figure 631. Encoding (32 bits)

Decode

```
return DvpOperands(rt=rt)
```

8.233.3. Operation

```
if not ctx.C0.Config5.VP:
    return
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
ctx.GPR[operands.rt] = sign_extend(read_cp0_register(ctx, 0, 4), from_nbits=32)
disable_virtual_processors(ctx)
```

8.233.4. Description

DVP saves VPControl and disables other virtual processors when VP support exists.

8.233.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.234. EHB

8.234.1. Purpose

Clear execution hazards.

8.234.2. Formats

nanoMIPS

ehb.32

Assembly

EHB

Availability

true

Decode priority

1

Overrides lower-priority formats

- SLL (sll.32)

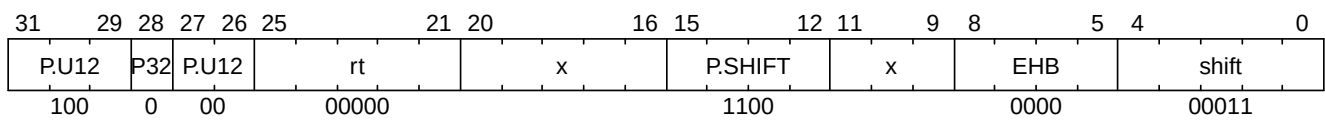


Figure 632. Encoding (32 bits)

Decode

```
return Ehb0perands()
```

8.234.3. Operation

```
clear_execution_hazards(ctx)
```

8.234.4. Description

EHB orders prior CP0 updates before subsequent instructions.

8.234.5. Exceptions

None.

8.235. EI

8.235.1. Purpose

Enable interrupts and return prior Status.

8.235.2. Formats

MIPS

ei

Assembly

EI {rt:gpr}

Availability

AR >= R2_3_5

Decode priority

0

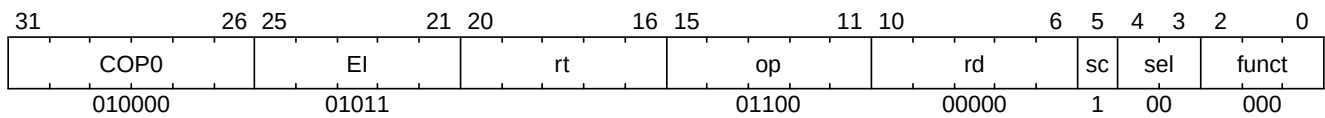


Figure 633. Encoding (32 bits)

Decode

```
return Ei0perands(rt=rt)
```

microMIPS

ei.32

Assembly

EI {rt:gpr}

Availability

true

Decode priority

0

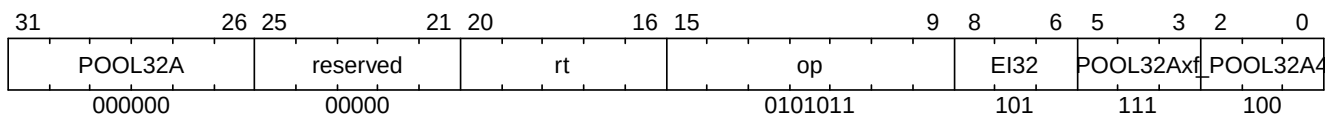


Figure 634. Encoding (32 bits)

Decode

```
return EiOperands(rt=rt)
```

nanoMIPS

ei.32

Assembly

EI {rt:gpr}

Availability

true

Decode priority

0

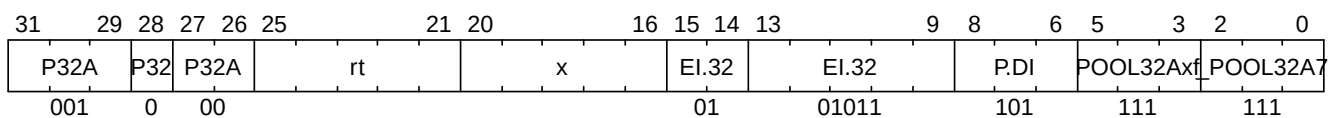


Figure 635. Encoding (32 bits)

Decode

```
return EiOperands(rt=rt)
```

8.235.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
ctx.GPR[operands.rt] = sign_extend(read_cp0_register(ctx, 12, 0), from_nbits=32)  
set_interrupt_enabled(ctx, True)
```

8.235.4. Description

EI saves CP0.Status in rt and sets Status.IE.

8.235.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.236. ERET

8.236.1. Purpose

Return from an exception, restore the interrupted PC, and clear LL state.

8.236.2. Formats

MIPS

eret

Assembly

ERET

Availability

true

Decode priority

0

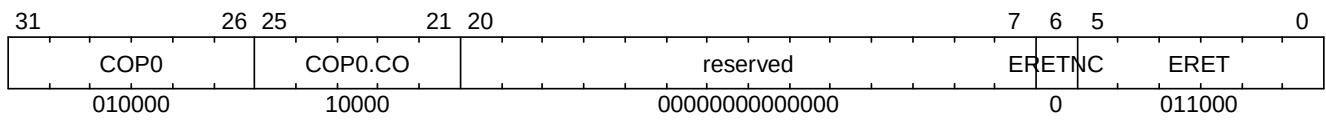


Figure 636. Encoding (32 bits)

Decode

```
exec.isCTI = True
return EretOperands(no_clear_llbit=False)
```

microMIPS

eret.32

Assembly

ERET

Availability

true

Decode priority

0

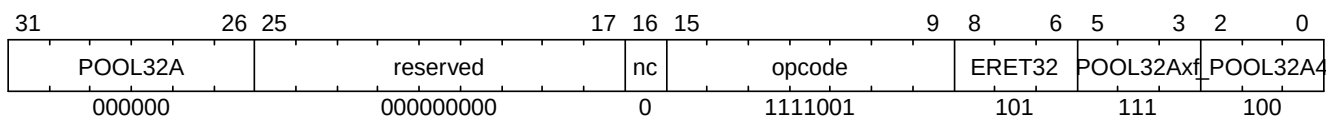


Figure 637. Encoding (32 bits)

Decode

```
exec.isCTI = True
return EretOperands(no_clear_llbit=False)
```

nanoMIPS

eret.32

Assembly

ERET

Availability

true

Decode priority

0

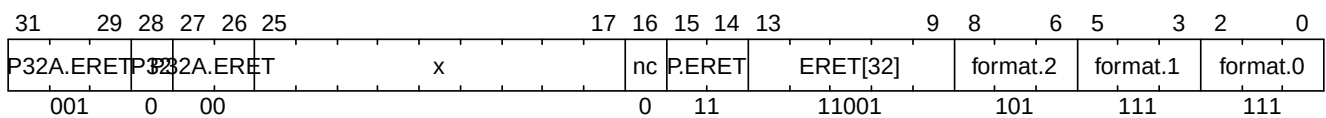


Figure 638. Encoding (32 bits)

Decode

```
exec.isCTI = True
return EretOperands(no_clear_llbit=False)
```

8.236.3. Operation

```
exception_return(ctx, no_clear_llbit=False)
```

8.236.4. Description

ERET returns through ErrorEPC while Status.ERL is set and through EPC otherwise, then clears the LL reservation.

8.236.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.237. ERETNC

8.237.1. Purpose

Return from an exception while preserving the LL reservation.

8.237.2. Formats

MIPS

eretnc

Assembly

ERETNC

Availability

LLB

Decode priority

0

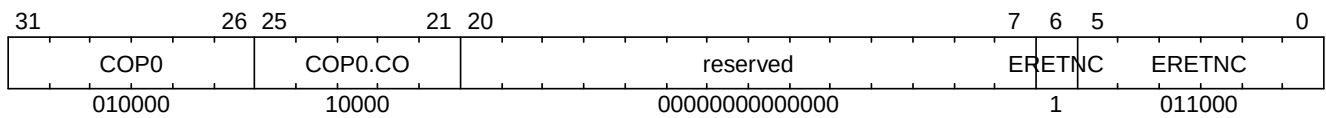


Figure 639. Encoding (32 bits)

Decode

```
exec.isCTI = True
return EretOperands(no_clear_llbit=True)
```

microMIPS

eretnc.32

Assembly

ERETNC

Availability

MMAR >= R6

Decode priority

0

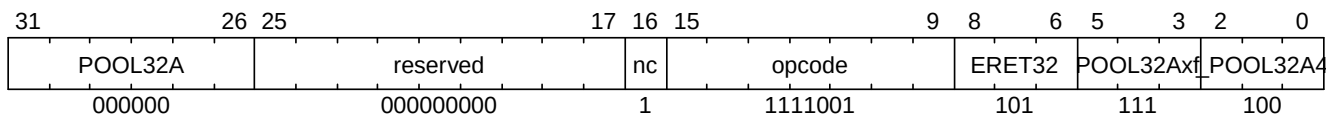


Figure 640. Encoding (32 bits)

Decode

```
exec.isCTI = True
return EretOperands(no_clear_llbit=True)
```

nanoMIPS

eretnc.32

Assembly

ERETNC

Availability

true

Decode priority

0

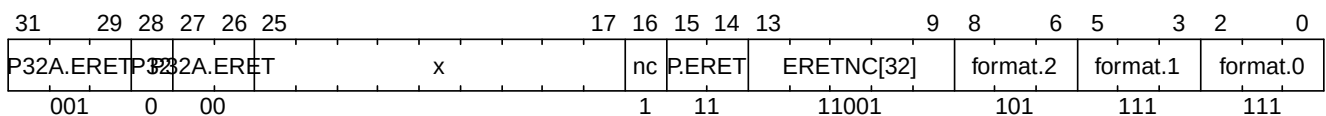


Figure 641. Encoding (32 bits)

Decode

```
exec.isCTI = True
return EretOperands(no_clear_llbit=True)
```

8.237.3. Operation

```
exception_return(ctx, no_clear_llbit=True)
```

8.237.4. Description

ERETNC returns through ErrorEPC while Status.ERL is set and through EPC otherwise, without clearing the LL reservation.

8.237.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — ERETNC requires Config5.LLB

8.238. EVP

8.238.1. Purpose

Enable virtual processors.

8.238.2. Formats

MIPS

evp

Assembly

EVP {rt:gpr}

Availability

AR >= R6

Decode priority

0

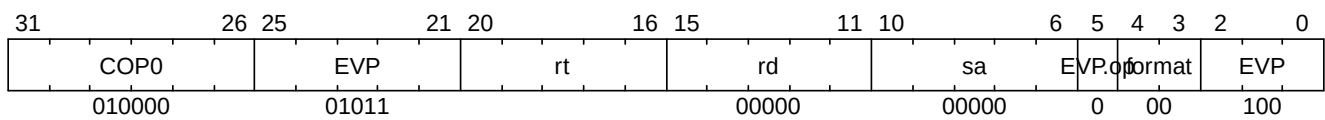


Figure 642. Encoding (32 bits)

Decode

```
return Evp0operands(rt=rt)
```

microMIPS

evp.32.micromips

Assembly

EVP {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

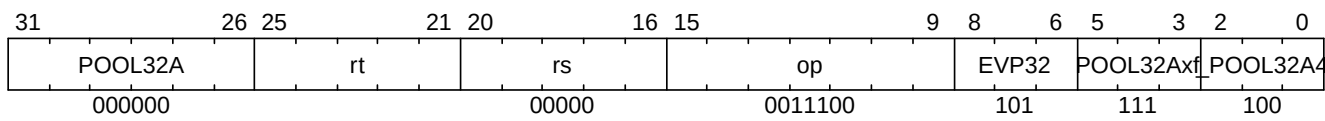


Figure 643. Encoding (32 bits)

Decode

```
return EvpOperands(rt=rt)
```

nanoMIPS

evp.32

Assembly

EVP {rt:gpr}

Availability

true

Decode priority

0

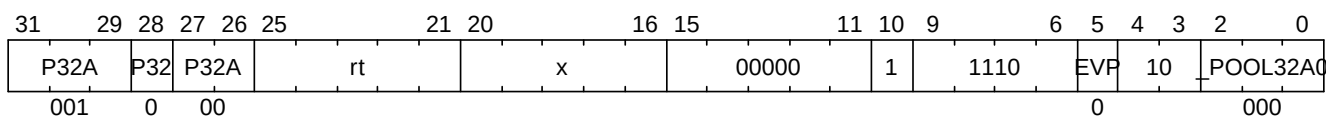


Figure 644. Encoding (32 bits)

Decode

```
return EvpOperands(rt=rt)
```

8.238.3. Operation

```
if not ctx.C0.Config5.VP:
    return
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
ctx.GPR[operands.rt] = sign_extend(read_cp0_register(ctx, 0, 4), from_nbits=32)
enable_virtual_processors(ctx)
```

8.238.4. Description

EVP saves VPControl and enables other virtual processors when VP support exists.

8.238.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.239. EXT

8.239.1. Purpose

Extract a bit field from `rs`, right-justify it, and place the sign-extended 32-bit result in `rt`.

8.239.2. Formats

MIPS

`ext`

Assembly

`EXT {rt:gpr}, {rs:gpr}, {pos:u5}, {size:u6}`

Availability

AR >= R2_3_5

Decode priority

0

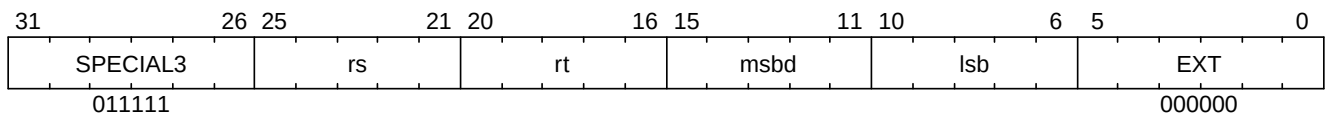


Figure 645. Encoding (32 bits)

Decode

```
return Ext0perands(rt=rt, rs=rs, pos=lsb, size=msbd + 1)
```

microMIPS

`ext.32`

Assembly

`EXT {rt:gpr}, {rs:gpr}, {pos:u5}, {size:u6}`

Availability

true

Decode priority

0

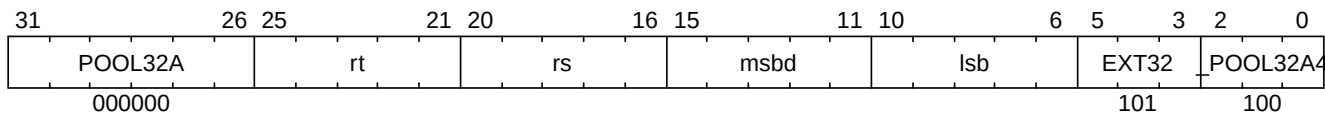


Figure 646. Encoding (32 bits)

Decode

```
pos = zero_extend(lsb, from_nbits=5)
size = zero_extend(msbd + 1, from_nbits=6)
return ExtOperands(rt=rt, rs=rs, pos=pos, size=size)
```

nanoMIPS

ext.32

Assembly

EXT {rt:gpr}, {rs:gpr}, {pos:u5}, {size:u6}

Availability

!(NMS)

Decode priority

0

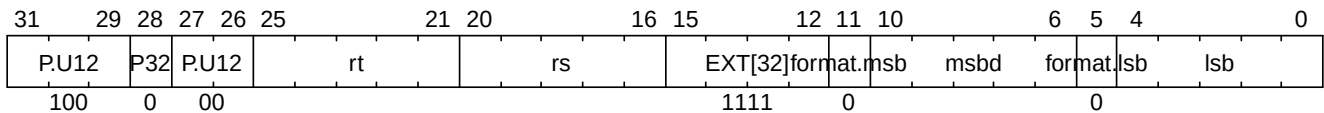


Figure 647. Encoding (32 bits)

Decode

```
pos = zero_extend(lsb, from_nbits=5)
size = zero_extend(msbd + 1, from_nbits=6)
return ExtOperands(rt=rt, rs=rs, pos=pos, size=size)
```

8.239.3. Operation

```
if int(operands.pos) + int(operands.size) > 32:
    raise ValueError("EXT with pos + size above 32 is UNPREDICTABLE")
extracted = zero_extend(
    ctx.GPR[operands.rs] >> int(operands.pos),
    from_nbits=int(operands.size),
)
ctx.GPR[operands.rt] = sign_extend(extracted, from_nbits=32)
```

8.239.4. Description

EXT[32] extracts **size** bits beginning at bit position **pos** from the low 32-bit word in **rs**. The field is zero-extended to a word, then the word result is sign-extended to the implemented GPR width. The encoding stores the inclusive most-significant bit as **msbd**, so **size** is **msbd + 1**; encodings with **pos + size > 32** are UNPREDICTABLE.

8.239.5. Exceptions

- **RI** (Reserved Instruction) — EXT[32] is reserved when CP0 Config5.NMS is set.

8.240. EXT

8.240.1. Purpose

Extract a doubleword from the concatenation of two 64-bit GPRs.

8.240.2. Formats

nanoMIPS

extd.32

Assembly

EXTD {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

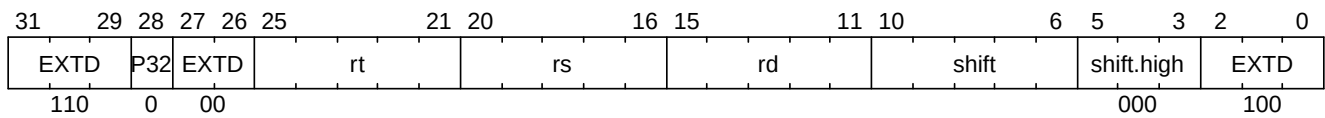


Figure 648. Encoding (32 bits)

Decode

```
shift = zero_extend(shift, from_nbits=5)
return ExtdOperands(rd=rd, rs=rs, rt=rt, shift=shift)
```

extd32.32

Assembly

EXTD32 {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u6}

Availability

AT >= MIPS64_32

Decode priority

0

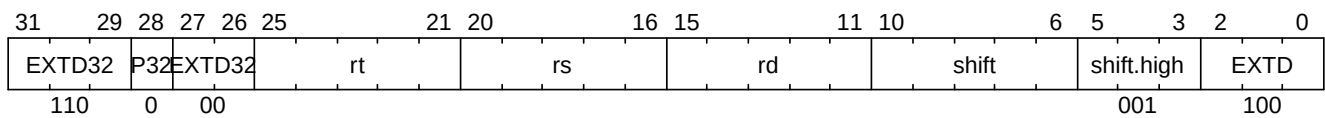


Figure 649. Encoding (32 bits)

Decode

```
shift = zero_extend(shift + 32, from_nbits=6)
```

```
return ExtdOperands(rd=rd, rs=rs, rt=rt, shift=shift)
```

8.240.3. Operation

```
validate_access_to_64bit_operations(ctx)
upper = int(zero_extend(ctx.GPR[operands.rt], from_nbits=64))
lower = int(zero_extend(ctx.GPR[operands.rs], from_nbits=64))
source = (upper << 64) | lower
result = (source >> int(operands.shift)) & MASK64
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=64)
```

8.240.4. Description

EXTD right-shifts the concatenation of rt and rs by shift and writes the low 64-bit result to rd.

8.240.5. Exceptions

- **RI** (Reserved Instruction) — 64-bit operations are disabled

8.241. EFTP

8.241.1. Purpose

DSP bitstream operation EFTP.

8.241.2. Formats

MIPS

extp

Assembly

EFTP {rt:gpr}, {ac:u2}, {size:u5}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

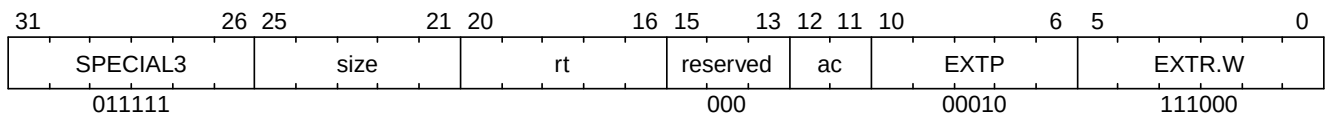


Figure 650. Encoding (32 bits)

Decode

```
return DspExtractOperands(rt=rt, ac=ac, size=size)
```

microMIPS

extp.32

Assembly

EFTP {rt:gpr}, {ac:u2}, {size:u5}

Availability

DSP

Decode priority

0

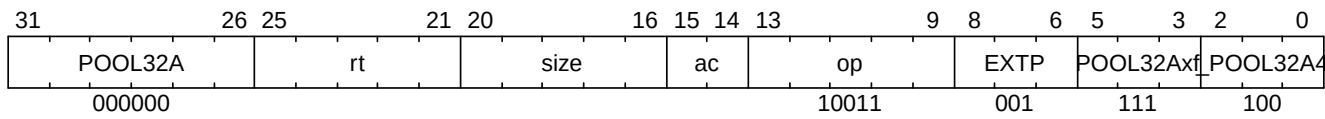


Figure 651. Encoding (32 bits)

Decode

```
return DspExtractOperands(rt=rt, ac=ac, size=size)
```

nanoMIPS

extp.32

Assembly

EXTP {rt:gpr}, {ac:u2}, {size:u5}

Availability

DSP

Decode priority

0

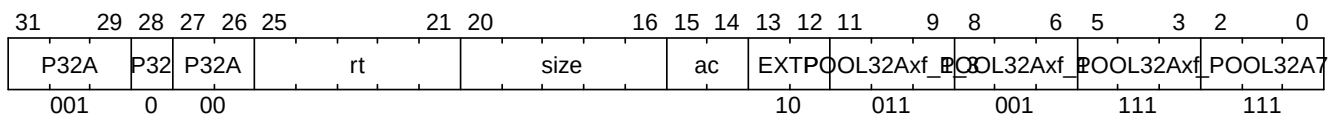


Figure 652. Encoding (32 bits)

Decode

```
return DspExtractOperands(rt=rt, ac=ac, size=size)
```

8.241.3. Operation

```
validate_access_to_dsp_resources(ctx)
extract(ctx, operands.ac, ctx.DSPControl.Pos, operands.size, false, operands.rt)
```

8.241.4. Description

EXTP is a DSP bitstream or accumulator extraction operation.

8.241.5. Exceptions

None.

8.242. EXTPDP

8.242.1. Purpose

DSP bitstream operation EXTPDP.

8.242.2. Formats

MIPS

extpdp

Assembly

EXTPDP {rt:gpr}, {ac:u2}, {size:u5}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

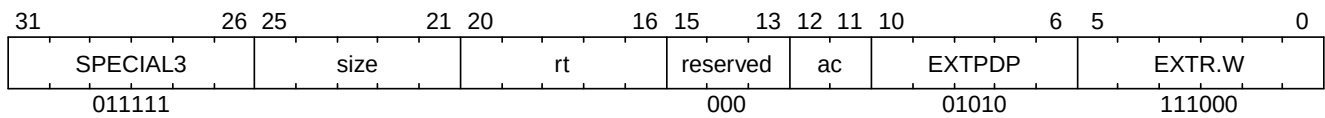


Figure 653. Encoding (32 bits)

Decode

```
return DspExtractOperands(rt=rt, ac=ac, size=size)
```

microMIPS

extpdp.32

Assembly

EXTPDP {rt:gpr}, {ac:u2}, {size:u5}

Availability

DSP

Decode priority

0

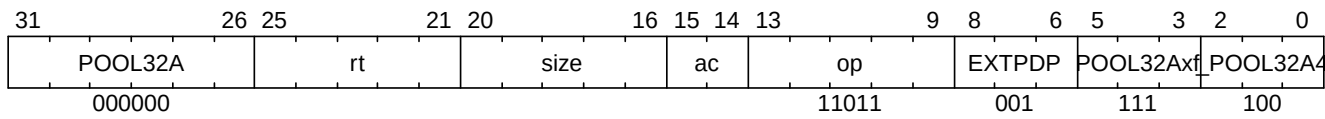


Figure 654. Encoding (32 bits)

Decode

```
return DspExtractOperands(rt=rt, ac=ac, size=size)
```

nanoMIPS

extpdp.32

Assembly

EXTPDP {rt:gpr}, {ac:u2}, {size:u5}

Availability

DSP

Decode priority

0

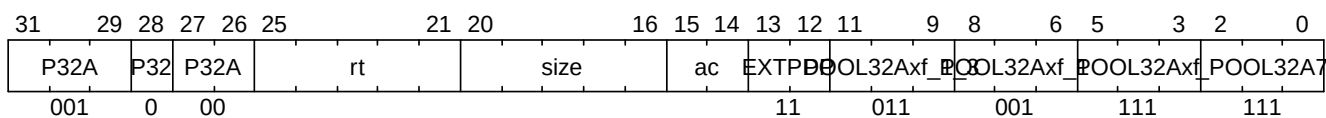


Figure 655. Encoding (32 bits)

Decode

```
return DspExtractOperands(rt=rt, ac=ac, size=size)
```

8.242.3. Operation

```
validate_access_to_dsp_resources(ctx)  
extract(ctx, operands.ac, ctx.DSPControl.Pos, operands.size, True, operands.rt)
```

8.242.4. Description

EXTPDP is a DSP bitstream or accumulator extraction operation.

8.242.5. Exceptions

None.

8.243. EXTPDPV

8.243.1. Purpose

DSP bitstream operation EXTPDPV.

8.243.2. Formats

MIPS

extpdpv

Assembly

EXTPDPV {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

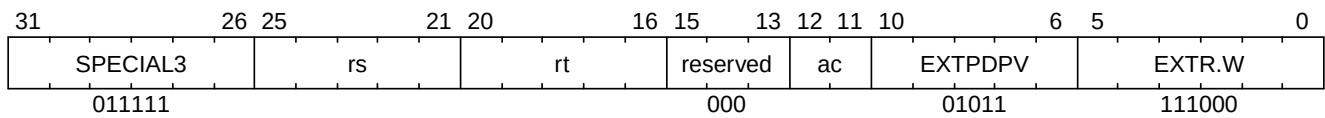


Figure 656. Encoding (32 bits)

Decode

```
return DspExtractVariable0perands(rt=rt, ac=ac, rs=rs)
```

microMIPS

extpdpv.32

Assembly

EXTPDPV {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

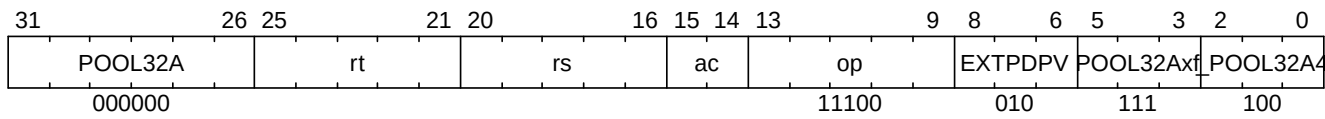


Figure 657. Encoding (32 bits)

Decode

```
return DspExtractVariableOperands(rt=rt, ac=ac, rs=rs)
```

nanoMIPS

extpdpv.32

Assembly

EXTPDPV {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

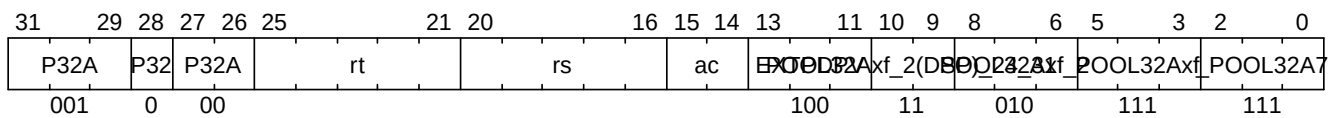


Figure 658. Encoding (32 bits)

Decode

```
return DspExtractVariableOperands(rt=rt, ac=ac, rs=rs)
```

8.243.3. Operation

```
validate_access_to_dsp_resources(ctx)
extract(
    ctx,
    operands.ac,
    ctx.DSPControl.Pos,
    ctx.GPR[operands.rs] & 0x1F,
    True,
    operands.rt,
)
```

8.243.4. Description

EXTPDPV is a DSP bitstream or accumulator extraction operation.

8.243.5. Exceptions

None.

8.244. EXTPV

8.244.1. Purpose

DSP bitstream operation EXTPV.

8.244.2. Formats

MIPS

extpv

Assembly

EXTPV {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

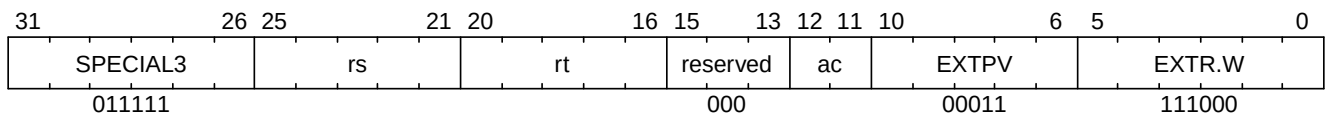


Figure 659. Encoding (32 bits)

Decode

```
return DspExtractVariable0perands(rt=rt, ac=ac, rs=rs)
```

microMIPS

extpv.32

Assembly

EXTPV {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

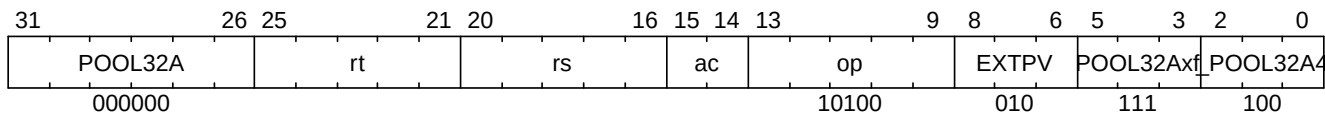


Figure 660. Encoding (32 bits)

Decode

```
return DspExtractVariableOperands(rt=rt, ac=ac, rs=rs)
```

nanoMIPS

extpv.32

Assembly

EXTPV {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

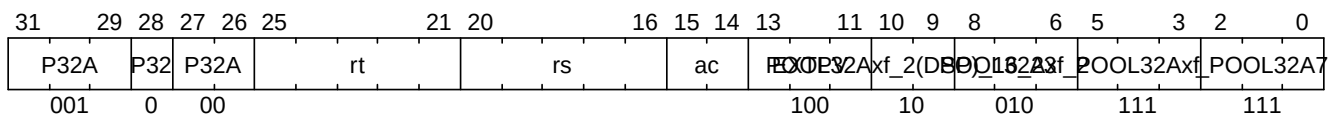


Figure 661. Encoding (32 bits)

Decode

```
return DspExtractVariableOperands(rt=rt, ac=ac, rs=rs)
```

8.244.3. Operation

```
validate_access_to_dsp_resources(ctx)
extract(
    ctx,
    operands.ac,
    ctx.DSPControl.Pos,
    ctx.GPR[operands.rs] & 0x1F,
    False,
    operands.rt,
)
```

8.244.4. Description

EXTPV is a DSP bitstream or accumulator extraction operation.

8.244.5. Exceptions

None.

8.245. EXTR.W

8.245.1. Purpose

DSP bitstream operation EXTR.W.

8.245.2. Formats

MIPS

`extr.w`

Assembly

`EXTR.W {rt:gpr}, {ac:u2}, {shift:u5}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

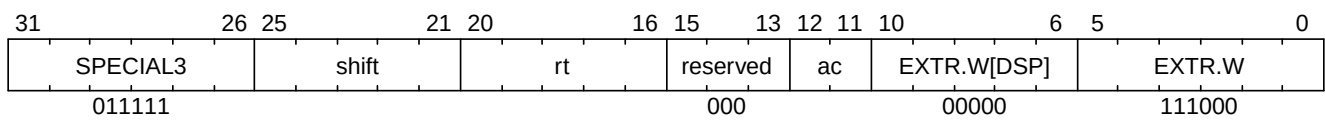


Figure 662. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

microMIPS

`extr.w.32`

Assembly

`EXTR.W {rt:gpr}, {ac:u2}, {shift:u5}`

Availability

DSP

Decode priority

0

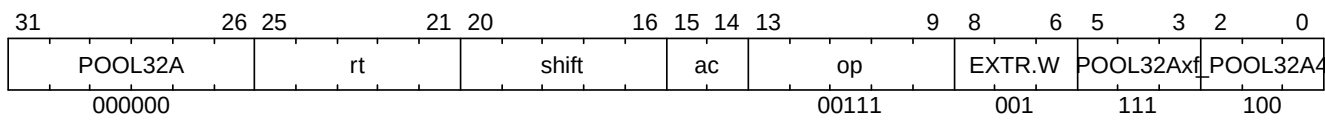


Figure 663. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

nanoMIPS

extr.w.32

Assembly

EXTR.W {rt:gpr}, {ac:u2}, {shift:u5}

Availability

DSP

Decode priority

0

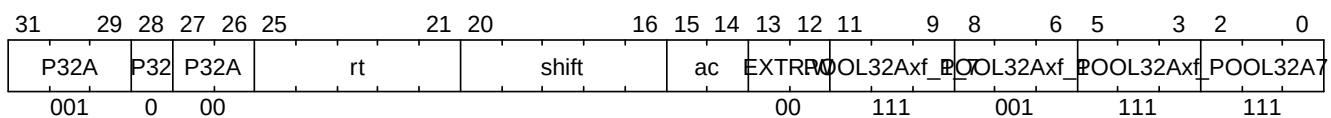


Figure 664. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

8.245.3. Operation

```
validate_access_to_dsp_resources(ctx)
extract_word(
    ctx,
    operands.ac,
    operands.shift,
    rounded=False,
    saturating=False,
    halfword=False,
    target=operands.rt,
)
```

8.245.4. Description

EXTR.W is a DSP bitstream or accumulator extraction operation.

8.245.5. Exceptions

None.

8.246. EXTR_R.W

8.246.1. Purpose

DSP bitstream operation EXTR_R.W.

8.246.2. Formats

MIPS

extr_r.w

Assembly

EXTR_R.W {rt:gpr}, {ac:u2}, {shift:u5}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

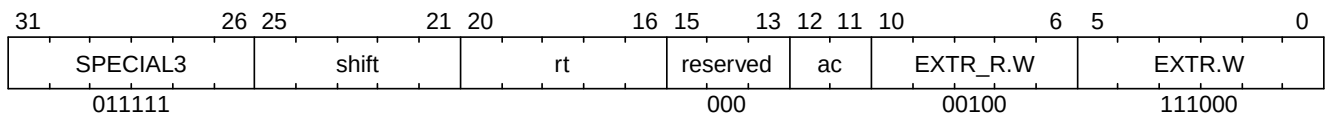


Figure 665. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

microMIPS

extr_r.w.32

Assembly

EXTR_R.W {rt:gpr}, {ac:u2}, {shift:u5}

Availability

DSP

Decode priority

0

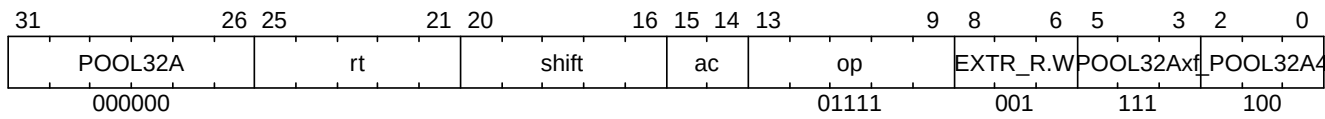


Figure 666. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

nanoMIPS

extr_r.w.32

Assembly

EXTR_R.W {rt:gpr}, {ac:u2}, {shift:u5}

Availability

DSP

Decode priority

0

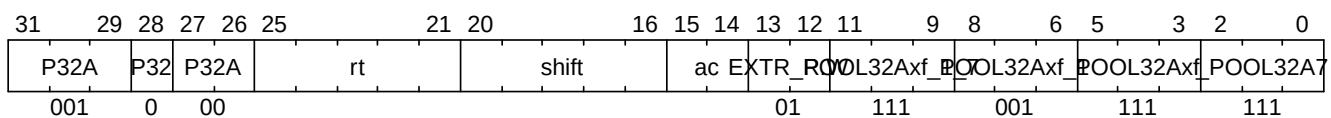


Figure 667. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

8.246.3. Operation

```
validate_access_to_dsp_resources(ctx)
extract_word(
    ctx,
    operands.ac,
    operands.shift,
    rounded=True,
    saturating=False,
    halfword=False,
    target=operands.rt,
)
```

8.246.4. Description

EXTR_R.W is a DSP bitstream or accumulator extraction operation.

8.246.5. Exceptions

None.

8.247. EXTR_RS.W

8.247.1. Purpose

DSP bitstream operation EXTR_RS.W.

8.247.2. Formats

MIPS

extr_rs.w

Assembly

EXTR_RS.W {rt:gpr}, {ac:u2}, {shift:u5}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

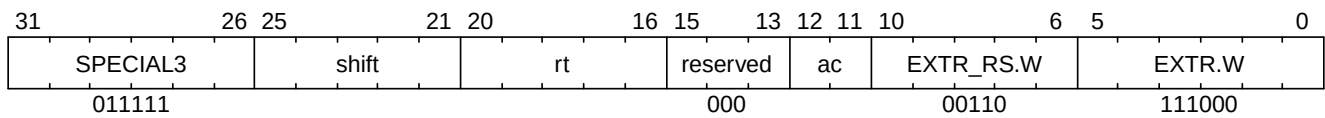


Figure 668. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

microMIPS

extr_rs.w.32

Assembly

EXTR_RS.W {rt:gpr}, {ac:u2}, {shift:u5}

Availability

DSP

Decode priority

0

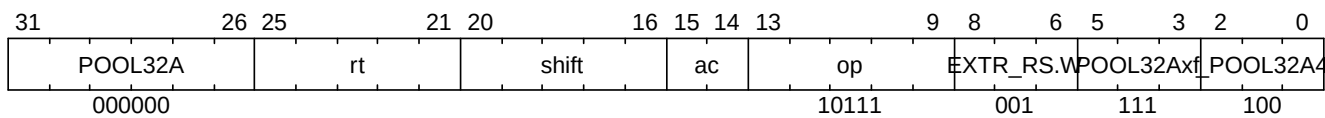


Figure 669. Encoding (32 bits)

Decode

```
return DspExtractShift0perands(rt=rt, ac=ac, shift=shift)
```

nanoMIPS

extr_rs.w.32

Assembly

EXTR_RS.W {rt:gpr}, {ac:u2}, {shift:u5}

Availability

DSP

Decode priority

0

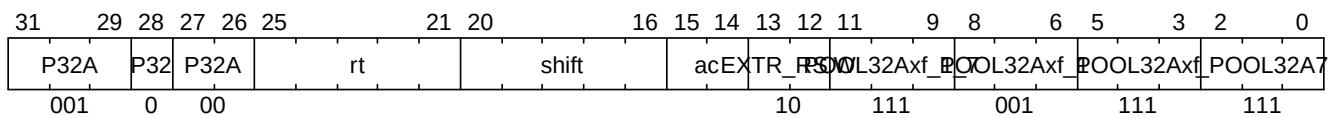


Figure 670. Encoding (32 bits)

Decode

```
return DspExtractShift0perands(rt=rt, ac=ac, shift=shift)
```

8.247.3. Operation

```
validate_access_to_dsp_resources(ctx)
extract_word(
    ctx,
    operands.ac,
    operands.shift,
    rounded=True,
    saturating=True,
    halfword=False,
    target=operands.rt,
)
```

8.247.4. Description

EXTR_RS.W is a DSP bitstream or accumulator extraction operation.

8.247.5. Exceptions

None.

8.248. EXTR_S.H

8.248.1. Purpose

DSP bitstream operation EXTR_S.H.

8.248.2. Formats

MIPS

extr_s.h

Assembly

EXTR_S.H {rt:gpr}, {ac:u2}, {shift:u5}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

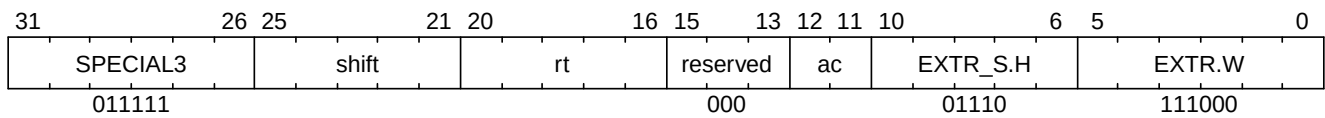


Figure 671. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

microMIPS

extr_s.h.32

Assembly

EXTR_S.H {rt:gpr}, {ac:u2}, {shift:u5}

Availability

DSP

Decode priority

0

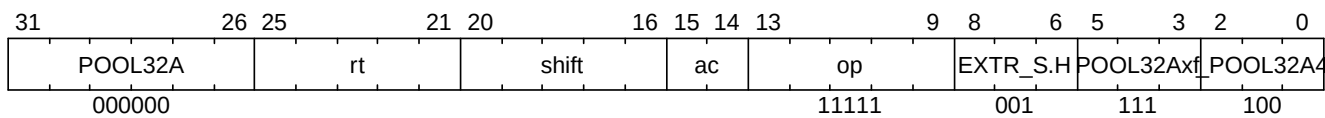


Figure 672. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

nanoMIPS

`extr_s.h.32`

Assembly

`EXTR_S.H {rt:gpr}, {ac:u2}, {shift:u5}`

Availability

DSP

Decode priority

0

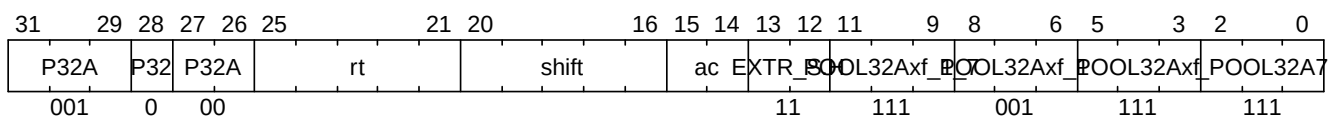


Figure 673. Encoding (32 bits)

Decode

```
return DspExtractShiftOperands(rt=rt, ac=ac, shift=shift)
```

8.248.3. Operation

```
validate_access_to_dsp_resources(ctx)
extract_word(
    ctx,
    operands.ac,
    operands.shift,
    rounded=False,
    saturating=True,
    halfword=True,
    target=operands.rt,
)
```

8.248.4. Description

EXTR_S.H is a DSP bitstream or accumulator extraction operation.

8.248.5. Exceptions

None.

8.249. EXTRV.W

8.249.1. Purpose

DSP bitstream operation EXTRV.W.

8.249.2. Formats

MIPS

extrv.w

Assembly

EXTRV.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

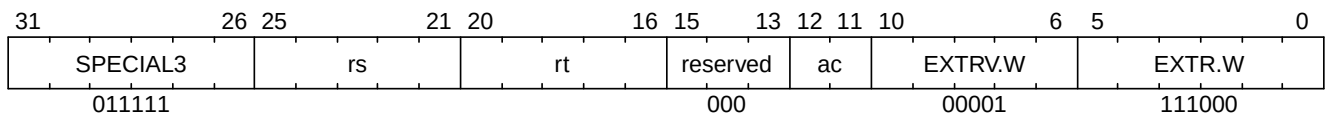


Figure 674. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

microMIPS

extrv.w.32

Assembly

EXTRV.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

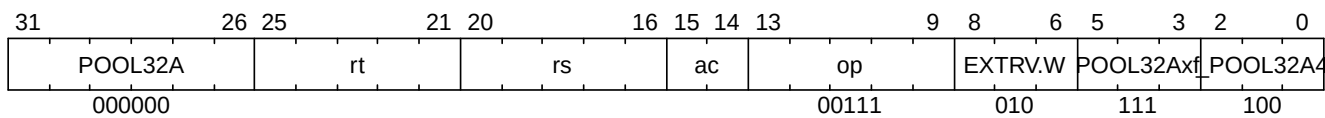


Figure 675. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

nanoMIPS

extrv.w.32

Assembly

EXTRV.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

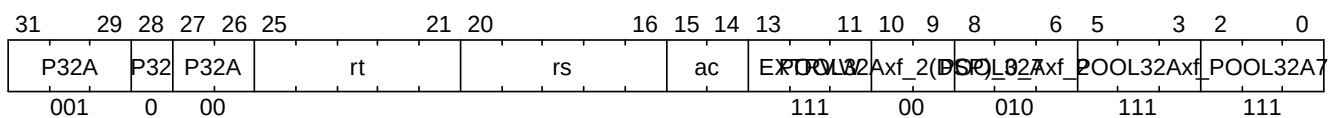


Figure 676. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

8.249.3. Operation

```
validate_access_to_dsp_resources(ctx)
shift = ctx.GPR[operands.rs] & 0x1F
extract_word(
    ctx,
    operands.ac,
    shift,
    rounded=False,
    saturating=False,
    halfword=False,
    target=operands.rt,
)
```

8.249.4. Description

EXTRV.W is a DSP bitstream or accumulator extraction operation.

8.249.5. Exceptions

None.

8.250. EXTRV_R.W

8.250.1. Purpose

DSP bitstream operation EXTRV_R.W.

8.250.2. Formats

MIPS

extrv_r.w

Assembly

EXTRV_R.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

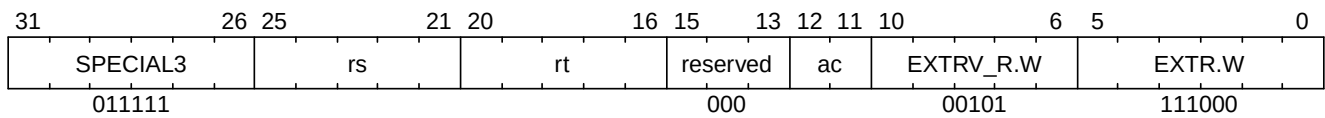


Figure 677. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

microMIPS

extrv_r.w.32

Assembly

EXTRV_R.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

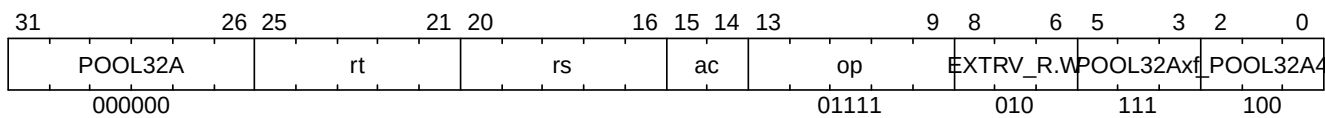


Figure 678. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

nanoMIPS

extrv_r.w.32

Assembly

EXTRV_R.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

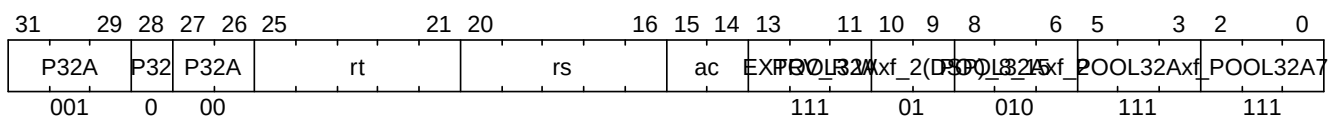


Figure 679. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

8.250.3. Operation

```
validate_access_to_dsp_resources(ctx)
shift = ctx.GPR[operands.rs] & 0x1F
extract_word(
    ctx,
    operands.ac,
    shift,
    rounded=True,
    saturating=False,
    halfword=False,
    target=operands.rt,
)
```

8.250.4. Description

EXTRV_R.W is a DSP bitstream or accumulator extraction operation.

8.250.5. Exceptions

None.

8.251. EXTRV_RS.W

8.251.1. Purpose

DSP bitstream operation EXTRV_RS.W.

8.251.2. Formats

MIPS

extrv_rs.w

Assembly

EXTRV_RS.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

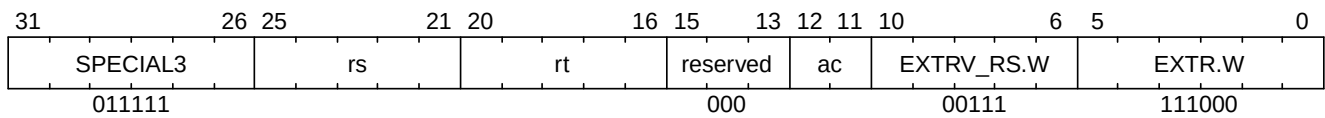


Figure 680. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

microMIPS

extrv_rs.w.32

Assembly

EXTRV_RS.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

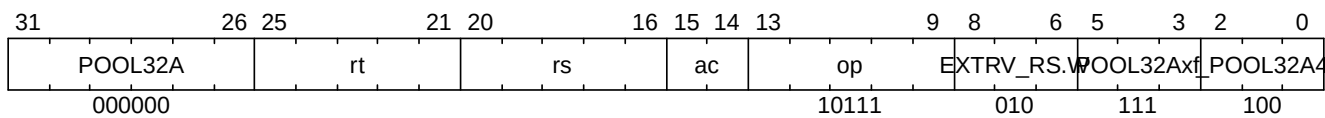


Figure 681. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

nanoMIPS

extrv_rs.w.32

Assembly

EXTRV_RS.W {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

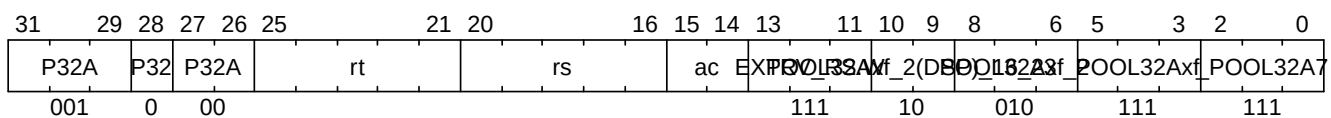


Figure 682. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

8.251.3. Operation

```
validate_access_to_dsp_resources(ctx)
shift = ctx.GPR[operands.rs] & 0x1F
extract_word(
    ctx,
    operands.ac,
    shift,
    rounded=True,
    saturating=True,
    halfword=False,
    target=operands.rt,
)
```

8.251.4. Description

EXTRV_RS.W is a DSP bitstream or accumulator extraction operation.

8.251.5. Exceptions

None.

8.252. EXTRV_S.H

8.252.1. Purpose

DSP bitstream operation EXTRV_S.H.

8.252.2. Formats

MIPS

extrv_s.h

Assembly

EXTRV_S.H {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

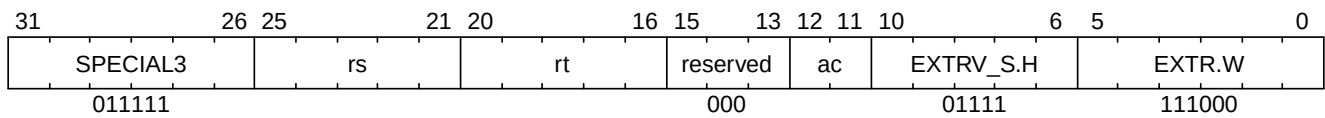


Figure 683. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

microMIPS

extrv_s.h.32

Assembly

EXTRV_S.H {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

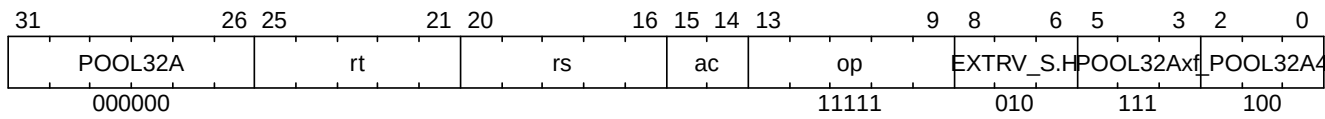


Figure 684. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

nanoMIPS

extrv_s.h.32

Assembly

EXTRV_S.H {rt:gpr}, {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

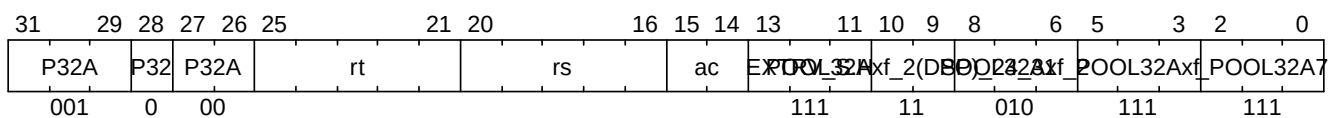


Figure 685. Encoding (32 bits)

Decode

```
return DspExtractShiftVariableOperands(rt=rt, ac=ac, rs=rs)
```

8.252.3. Operation

```
validate_access_to_dsp_resources(ctx)
shift = ctx.GPR[operands.rs] & 0x1F
extract_word(
    ctx,
    operands.ac,
    shift,
    rounded=False,
    saturating=True,
    halfword=True,
    target=operands.rt,
)
```

8.252.4. Description

EXTRV_S.H is a DSP bitstream or accumulator extraction operation.

8.252.5. Exceptions

None.

8.253. EXTW

8.253.1. Purpose

Extract a 32-bit word from the concatenation of two source GPRs at a constant bit position.

8.253.2. Formats

nanoMIPS

`extw.32`

Assembly

`EXTW {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u5}`

Availability

true

Decode priority

0

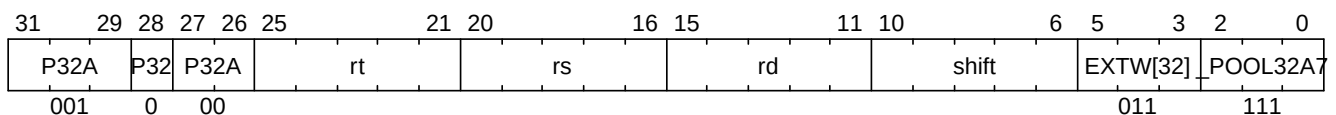


Figure 686. Encoding (32 bits)

Decode

```
del ctx, exec
return ExtwOperands(
    rd=rd,
    rs=rs,
    rt=rt,
    shift=zero_extend(shift, from_nbits=5),
)
```

8.253.3. Operation

```
high = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
low = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
concatenated = (int(high) << 32) | int(low)
result = concatenated >> int(operands.shift)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.253.4. Description

EXTW concatenates the low 32-bit words of rt and rs, shifts the 64-bit result right by shift, and sign-extends the low 32-bit result to the implemented GPR width.

8.253.5. Exceptions

None.

8.254. FLOOR.L.D

8.254.1. Purpose

Convert D format to L format using floor rounding.

8.254.2. Formats

MIPS

floor.l.d

Assembly

FLOOR.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

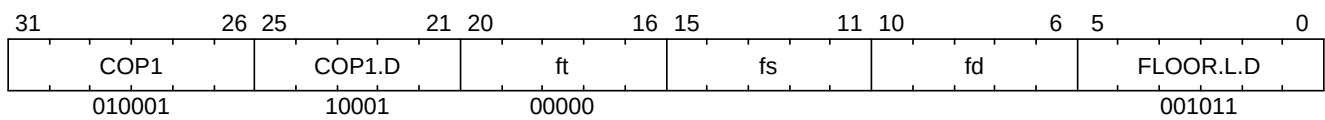


Figure 687. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

floor.l.d

Assembly

FLOOR.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

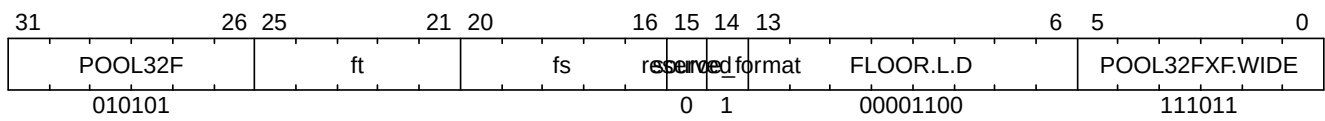


Figure 688. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

`floor.l.d`

Assembly

`FLOOR.L.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

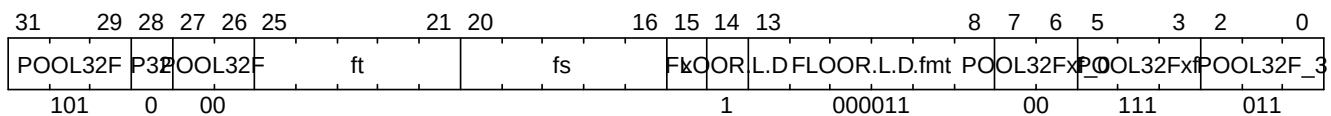


Figure 689. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.254.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "D")
result, flags = ieee_float_to_int(
    "D", "L", fpu_rounding_mode(ctx, 3), get_fpr(ctx, operands.fs, "D"), rm=3
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.254.4. Description

FLOOR.L.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.254.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.255. FLOOR.L.S

8.255.1. Purpose

Convert S format to L format using floor rounding.

8.255.2. Formats

MIPS

floor.l.s

Assembly

FLOOR.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

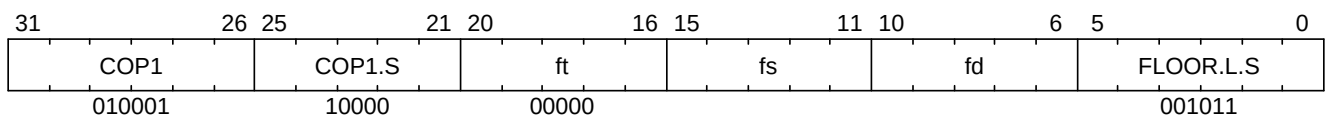


Figure 690. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

floor.l.s

Assembly

FLOOR.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

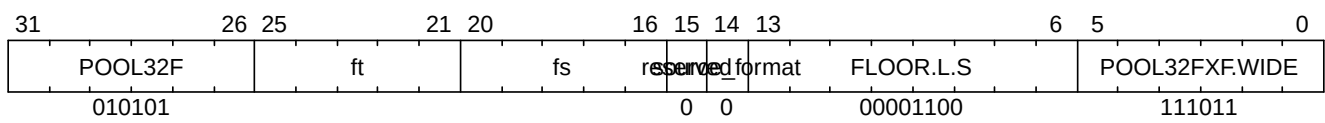


Figure 691. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

`floor.l.s`

Assembly

`FLOOR.L.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

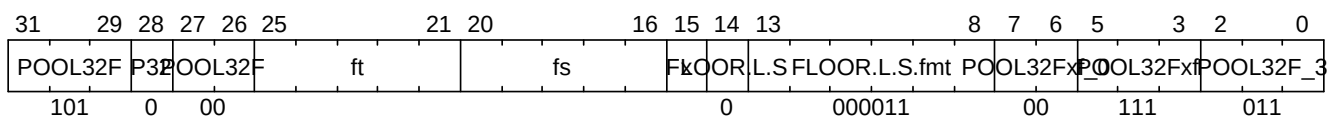


Figure 692. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.255.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "S")
result, flags = ieee_float_to_int(
    "S", "L", fpu_rounding_mode(ctx, 3), get_fpr(ctx, operands.fs, "S"), rm=3
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.255.4. Description

FLOOR.L.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.255.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.256. FLOOR.W.D

8.256.1. Purpose

Convert D format to W format using floor rounding.

8.256.2. Formats

MIPS

floor.w.d

Assembly

FLOOR.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

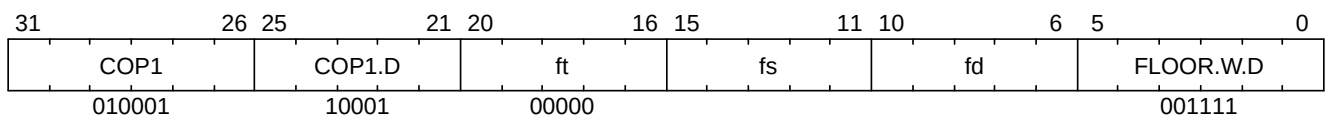


Figure 693. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

floor.w.d

Assembly

FLOOR.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

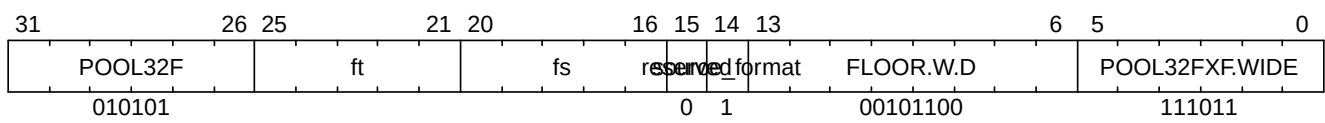


Figure 694. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

floor.w.d

Assembly

FLOOR.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

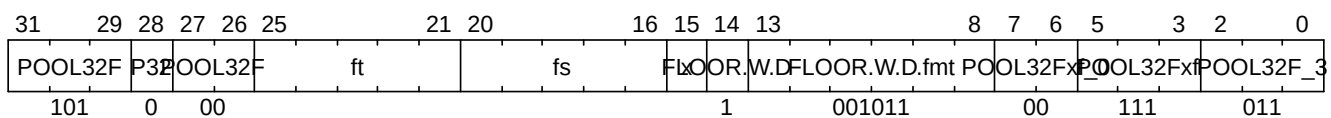


Figure 695. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.256.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "D")
result, flags = ieee_float_to_int(
    "D", "W", fpu_rounding_mode(ctx, 3), get_fpr(ctx, operands.fs, "D"), rm=3
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.256.4. Description

FLOOR.W.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.256.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.257. FLOOR.W.S

8.257.1. Purpose

Convert S format to W format using floor rounding.

8.257.2. Formats

MIPS

floor.w.s

Assembly

FLOOR.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

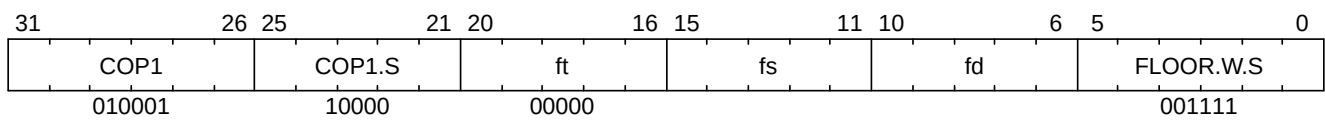


Figure 696. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

floor.w.s

Assembly

FLOOR.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

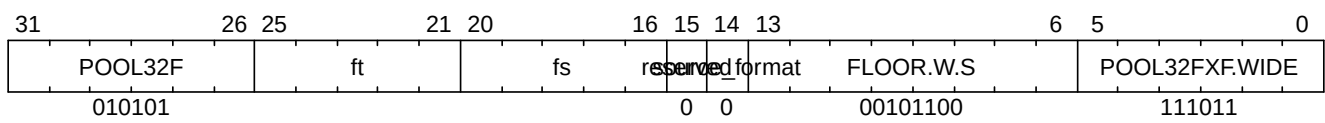


Figure 697. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

floor.w.s

Assembly

FLOOR.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

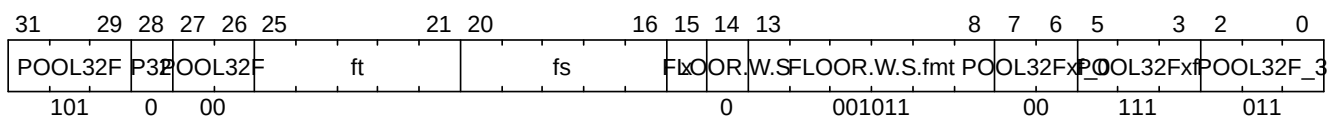


Figure 698. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.257.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "S")
result, flags = ieee_float_to_int(
    "S", "W", fpu_rounding_mode(ctx, 3), get_fpr(ctx, operands.fs, "S"), rm=3
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.257.4. Description

FLOOR.W.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.257.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.258. GINVI

8.258.1. Purpose

Globally invalidate instruction caches.

8.258.2. Formats

nanoMIPS

ginvi.32

Assembly

GINVI {rs:gpr}

Availability

true

Decode priority

0

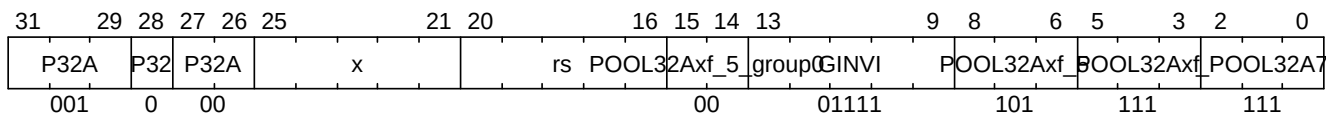


Figure 699. Encoding (32 bits)

Decode

```
return Ginvi0operands(rs=rs)
```

8.258.3. Operation

```
if ctx.C0.Config5.GI < 2:
    raise ReservedInstruction("GINVI requires GI >= 2")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
global_invalidate_instruction(ctx, ctx.GPR[operands.rs])
```

8.258.4. Description

GINVI requests global instruction-cache invalidation for all cores or an implementation-selected core.

8.258.5. Exceptions

- CPU (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **RI** (Reserved Instruction) — GINVI requires $GI \geq 2$

8.259. GINVT

8.259.1. Purpose

Globally invalidate TLBs.

8.259.2. Formats

nanoMIPS

ginvnt.32

Assembly

GINVT {rs:gpr}, {type:u2}

Availability

true

Decode priority

0

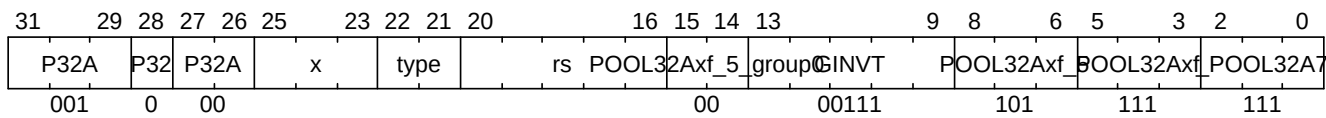


Figure 700. Encoding (32 bits)

Decode

```
return Ginvnt0operands(rs=rs, type=type)
```

8.259.3. Operation

```
if ctx.C0.Config5.GI != 3 or not ctx.C0.Config5.MI:
    raise ReservedInstruction("GINVT requires GI=3 and MI")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
global_invalidate_tlb(ctx, operands.type, ctx.GPR[operands.rs])
```

8.259.4. Description

GINVT requests global TLB invalidation by type and address.

8.259.5. Exceptions

- CPU (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- RI (Reserved Instruction) — GINVT requires GI=3 and MI

8.260. INS

8.260.1. Purpose

Insert a right-justified bit field from `rs` into `rt`.

8.260.2. Formats

MIPS

`ins`

Assembly

`INS {rt:gpr}, {rs:gpr}, {pos:u5}, {size:u6}`

Availability

AR >= R2_3_5

Decode priority

0

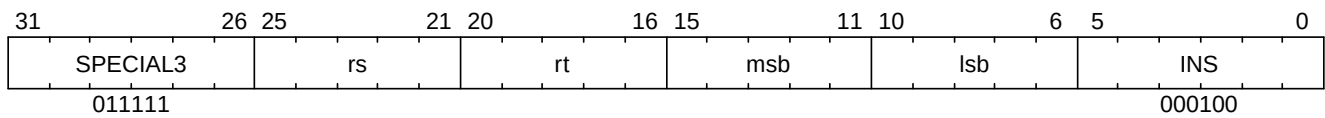


Figure 701. Encoding (32 bits)

Decode

```
return Ins0perands(rt=rt, rs=rs, pos=lsb, size=1 + msb - lsb)
```

microMIPS

`ins.32`

Assembly

`INS {rt:gpr}, {rs:gpr}, {pos:u5}, {size:u6}`

Availability

true

Decode priority

0

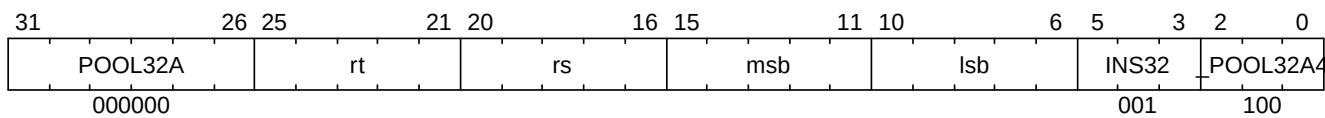


Figure 702. Encoding (32 bits)

Decode

```
return InsOperands(rt=rt, rs=rs, pos=lsb, size=1 + msb - lsb)
```

nanoMIPS

ins.32

Assembly

INS {rt:gpr}, {rs:gpr}, {pos:u5}, {size:u6}

Availability

!(NMS)

Decode priority

0

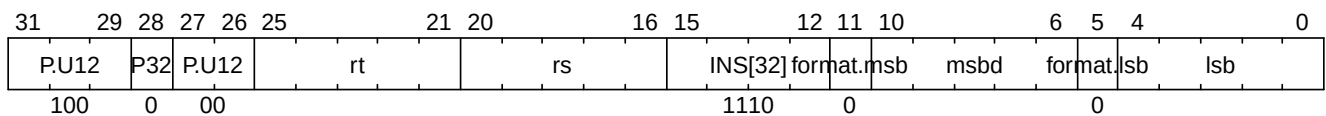


Figure 703. Encoding (32 bits)

Decode

```
pos = zero_extend(lsb, from_nbits=5)
size = 1 + msbd - lsb
return InsOperands(rt=rt, rs=rs, pos=pos, size=size)
```

8.260.3. Operation

```
if operands.size < 1:
    raise ValueError("INS with msbd below lsb is UNPREDICTABLE")
merge_mask = ((1 << operands.size) - 1) << operands.pos
destination = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
source = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
result = (int(destination) & ~merge_mask) | (
    (int(source) << operands.pos) & merge_mask
)
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.260.4. Description

INS[32] replaces **size** bits of the low 32-bit destination beginning at **pos**. The merged word is sign-extended to the implemented GPR width. The encoded **msbd** field is the inclusive destination end bit.

8.260.5. Exceptions

- **RI** (Reserved Instruction) — INS is reserved when CP0 Config5.NMS is set.; INS[32] is reserved

when CP0 Config5.NMS is set.

8.261. INSV

8.261.1. Purpose

DSP bitstream operation INSV.

8.261.2. Formats

MIPS

insv

Assembly

INSV {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

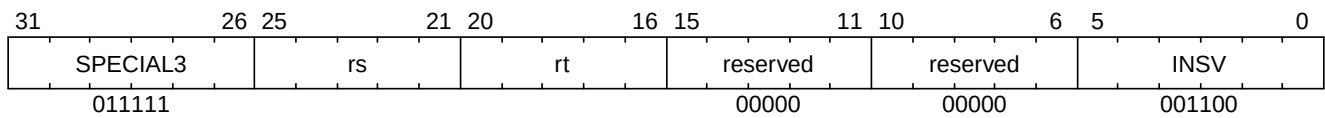


Figure 704. Encoding (32 bits)

Decode

```
return DspInsertOperands(rt=rt, rs=rs)
```

microMIPS

insv.32

Assembly

INSV {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

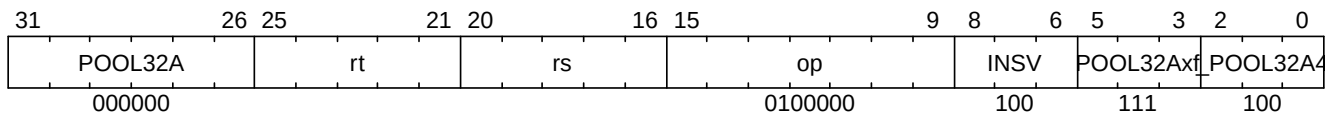


Figure 705. Encoding (32 bits)

Decode

```
return DspInsertOperands(rt=rt, rs=rs)
```

nanoMIPS

insv.32

Assembly

INSV {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

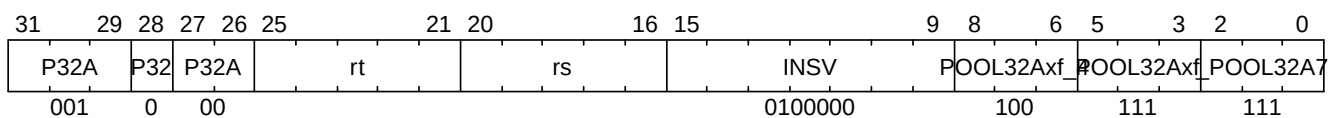


Figure 706. Encoding (32 bits)

Decode

```
return DspInsertOperands(rt=rt, rs=rs)
```

8.261.3. Operation

```
validate_access_to_dsp_resources(ctx)
pos, size = ctx.DSPControl.Pos, ctx.DSPControl.Scount
if pos >= 32 or size == 0 or pos + size > 32:
    ctx.GPR[operands.rt] = 0
    return
mask = (1 << size) - 1
result = (ctx.GPR[operands.rt] & ~(mask << pos)) | (
    (ctx.GPR[operands.rs] & mask) << pos
)
ctx.GPR[operands.rt] = sign_extend(result & 0xFFFF_FFFF, from_nbits=32)
```

8.261.4. Description

INSV is a DSP bitstream or accumulator extraction operation.

8.261.5. Exceptions

None.

8.262. J

8.262.1. Purpose

Jump within the current 256 MiB aligned region after one delay slot.

8.262.2. Formats

MIPS

j

Assembly

J {index:u28}

Availability

true

Decode priority

0

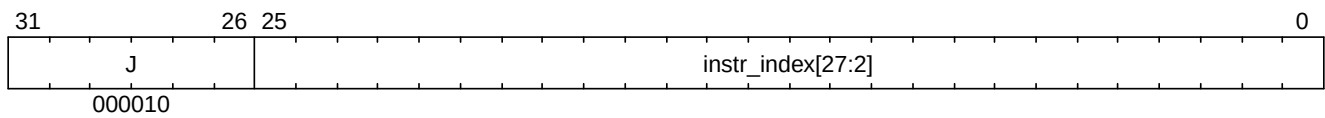


Figure 707. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return J0perands(index=zero_extend(instr_index, from_nbits=28))
```

microMIPS

j.32

Assembly

J {index:u27}

Availability

MMAR < R6

Decode priority

0

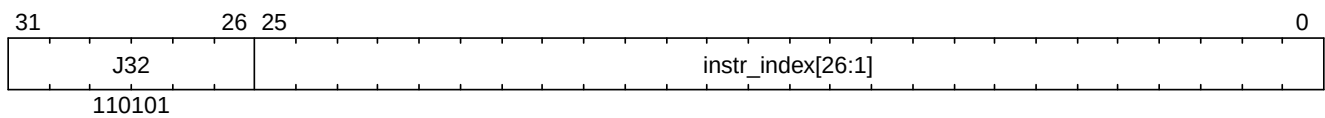


Figure 708. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return J0perands(index=zero_extend(instr_index, from_nbits=27))
```

8.262.3. Operation

```
region_mask = 0x07FF_FFFF if ctx.ISAMode is ISAMode.MICROMIPS else 0x0FFF_FFFF
target = (ctx.next_pc & ~region_mask) | int(operands.index)
delayed_branch(ctx, target)
```

8.262.4. Description

J concatenates the high address bits of the instruction following the jump with the encoded, word-aligned 28-bit index. The instruction in the delay slot executes before control reaches the target.

8.262.5. Exceptions

None.

8.263. JAL

8.263.1. Purpose

Jump within the current region and write the return address to GPR 31.

8.263.2. Formats

MIPS

jal

Assembly

JAL {index:u28}

Availability

true

Decode priority

0

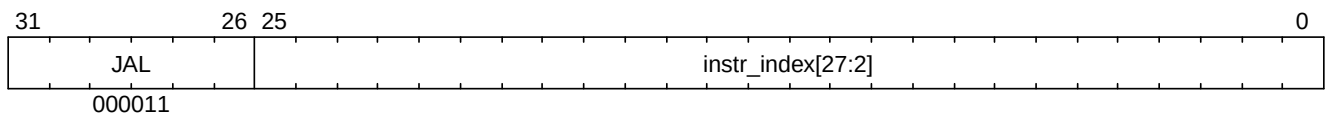


Figure 709. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Jal0operands(index=zero_extend(instr_index, from_nbits=28))
```

microMIPS

jal.32

Assembly

JAL {index:u27}

Availability

MMAR < R6

Decode priority

0

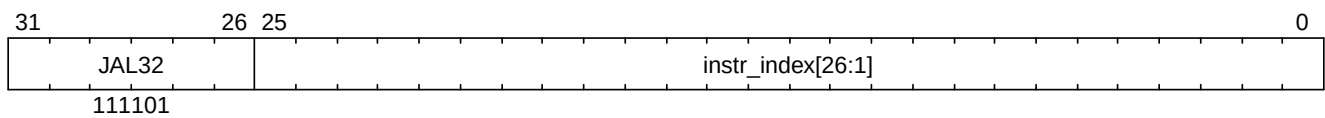


Figure 710. Encoding (32 bits)

```

exec.isCTI = True
exec.hasDelaySlot = True
return JalOperands(index=zero_extend(instr_index, from_nbits=27))

```

8.263.3. Operation

```

region_mask = 0x07FF_FFFF if ctx.ISAMode is ISAMode.MICROMIPS else 0x0FFF_FFFF
target = (ctx.next_pc & ~region_mask) | int(operands.index)
# The link is the full sequential PC. In MIPS64 this may be an XKPHYS
# address, so truncating it through a 32-bit sign extension loses the
# region bits needed by JR/JALR in the delay-slot return path.
ctx.GPR[31] = ModeTaggedAddress(
    ctx,
    effective_address(ctx, ctx.pc, 8, access_type="Fetch"),
)
delayed_branch(ctx, target)

```

8.263.4. Description

JAL writes the address following its delay slot to GPR 31, then transfers to the region-relative target after that slot executes.

8.263.5. Exceptions

None.

8.264. JALR

8.264.1. Purpose

Jump to a GPR address and write a selectable link GPR.

8.264.2. Formats

MIPS

jalr

Assembly

JALR {rd:gpr}, {rs:gpr}

Availability

true

Decode priority

0

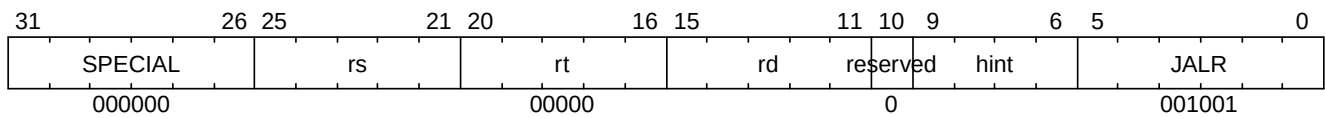


Figure 711. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalrOperands(rd=rd, rs=rs, hint=hint)
```

jalr.hb

Assembly

JALR {rd:gpr}, {rs:gpr}

Availability

true

Decode priority

0

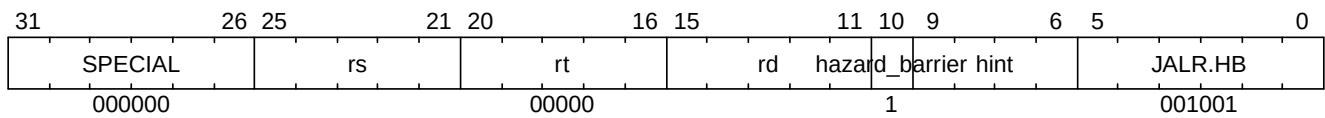


Figure 712. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalrOperands(rd=rd, rs=rs, hint=hint)
```

microMIPS

jalr.16

Assembly

JALR {rs:gpr}

Availability

MMAR < R6

Decode priority

0

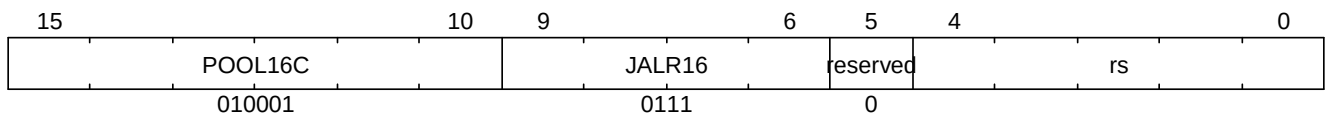


Figure 713. Encoding (16 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalrOperands(rd=31, rs=rs, hint=0)
```

jalr.32

Assembly

JALR {rd:gpr}, {rs:gpr}

Availability

MMAR < R6

Decode priority

0

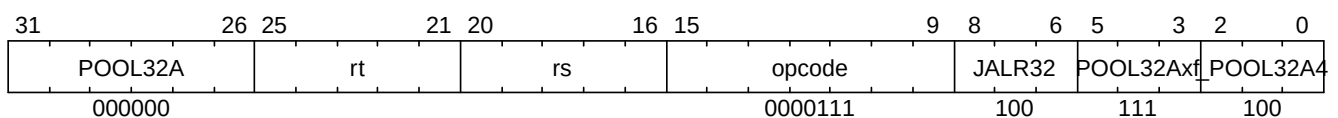


Figure 714. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
```

```
return JalrOperands(rd=rt, rs=rs, hint=0)
```

jalr.hb.32

Assembly

JALR {rd:gpr}, {rs:gpr}

Availability

MMAR < R6

Decode priority

0

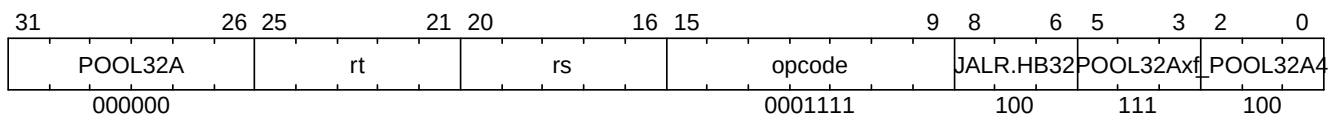


Figure 715. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalrOperands(rd=rt, rs=rs, hint=0)
```

8.264.3. Operation

```
target = ctx.GPR[operands.rs]
ctx.GPR[operands.rd] = ModeTaggedAddress(
    ctx,
    effective_address(ctx, ctx.next_pc, 4, access_type="Fetch"),
)
delayed_jump_register(ctx, target)
```

8.264.4. Description

JALR captures the target before writing the link value, so using the same register as source and destination has well-defined behavior.

8.264.5. Exceptions

None.

8.265. JALRC

8.265.1. Purpose

Perform an unconditional compact jump through a GPR and save the address of the following instruction in a destination GPR.

8.265.2. Formats

microMIPS

jalrc.16

Assembly

JALRC {src:gpr}

Availability

MMAR >= R6

Decode priority

0

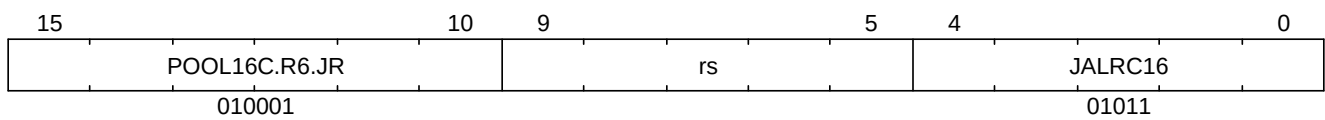


Figure 716. Encoding (16 bits)

Decode

```
exec.isCTI = True
return JalrcOperands(dst=31, src=rs)
```

jalrc.32

Assembly

JALRC {dst:gpr}, {src:gpr}

Availability

MMAR >= R6

Decode priority

0

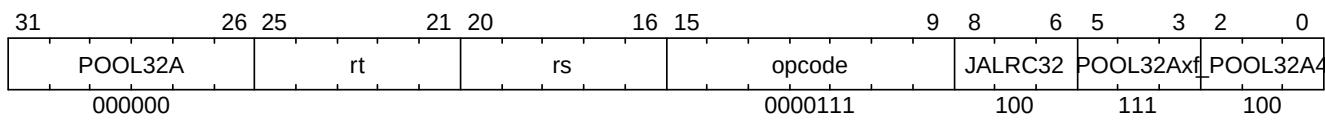


Figure 717. Encoding (32 bits)

Decode

```
exec.isCTI = True
return JalrcOperands(dst=rt, src=rs)
```

jalrc.hb.32

Assembly

JALRC {dst:gpr}, {src:gpr}

Availability

MMAR >= R6

Decode priority

0

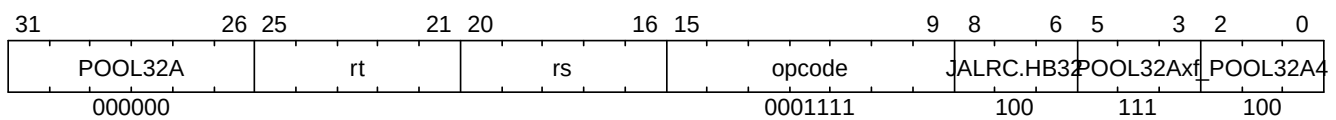


Figure 718. Encoding (32 bits)

Decode

```
exec.isCTI = True
return JalrcOperands(dst=rt, src=rs)
```

nanoMIPS

jalrc.16

Assembly

JALRC {src:gpr}

Availability

true

Decode priority

0

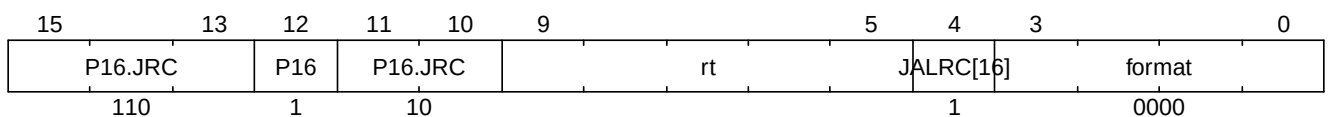


Figure 719. Encoding (16 bits)

Decode

```
exec.isCTI = True
return JalrcOperands(dst=31, src=rt)
```

jalrc.32

Assembly

JALRC {dst:gpr}, {src:gpr}

Availability

true

Decode priority

0

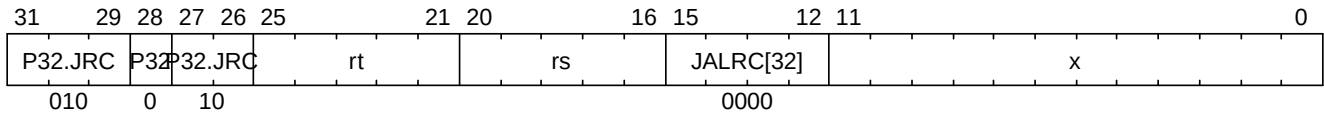


Figure 720. Encoding (32 bits)

Decode

```
exec.isCTI = True
return JalrcOperands(dst=rt, src=rs)
```

jalrc.hb.32

Assembly

JALRC {dst:gpr}, {src:gpr}

Availability

true

Decode priority

0

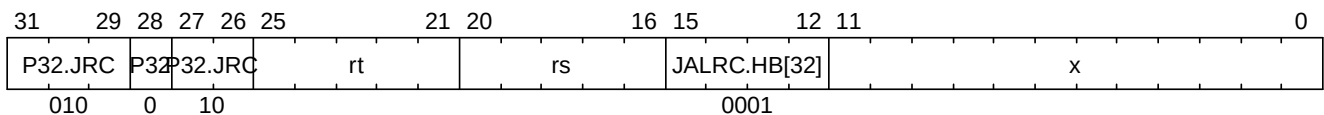


Figure 721. Encoding (32 bits)

Decode

```
exec.isCTI = True
return JalrcOperands(dst=rt, src=rs)
```

8.265.3. Operation

```
address = effective_address(ctx, ctx.GPR[operands.src], 0)
link_address = ModeTaggedAddress(ctx, ctx.next_pc)
compact_jump_register(ctx, address)
ctx.GPR[operands.dst] = link_address
```


8.265.4. Description

The 16-bit forms use GPR 31 as the destination and the encoded GPR as the source. The 32-bit forms encode both destination and source GPRs. JALRC.HB[32] has the same control-flow and link behavior while also serving as an instruction and execution hazard barrier. All forms are compact transfers without a delay slot.

8.265.5. Exceptions

None.

8.266. JALRS

8.266.1. Purpose

Jump through a GPR and link across a short microMIPS delay slot.

8.266.2. Formats

microMIPS

jalrs.16

Assembly

JALRS {rs:gpr}

Availability

MMAR < R6

Decode priority

0

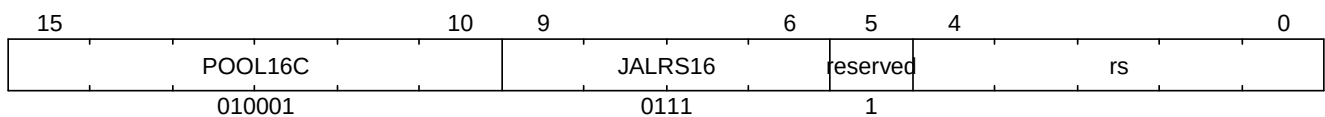


Figure 722. Encoding (16 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalrOperands(rd=31, rs=rs, hint=0)
```

jalrs.32

Assembly

JALRS {rd:gpr}, {rs:gpr}

Availability

MMAR < R6

Decode priority

0

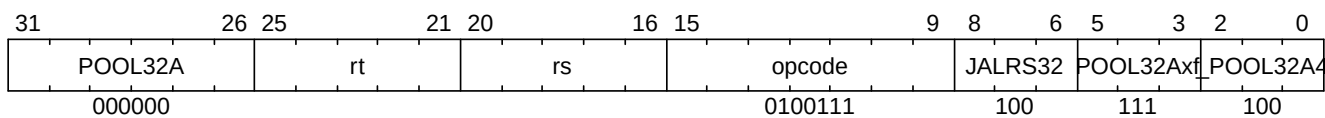


Figure 723. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalrOperands(rd=rt, rs=rs, hint=0)
```

`jalrs.hb.32`

Assembly

`JALRS {rd:gpr}, {rs:gpr}`

Availability

MMAR < R6

Decode priority

0

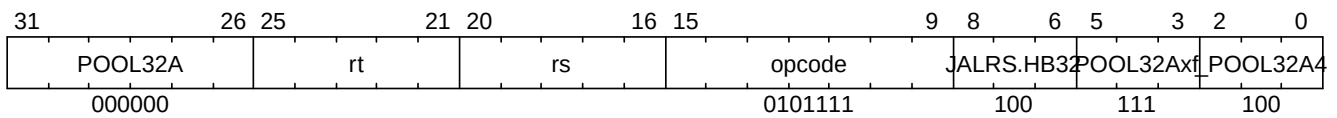


Figure 724. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalrOperands(rd=rt, rs=rs, hint=0)
```

8.266.3. Operation

```
target = ctx.GPR[operands.rs]
ctx.GPR[operands.rd] = ModeTaggedAddress(
    ctx,
    effective_address(ctx, ctx.next_pc, 2, access_type="Fetch"),
)
delayed_jump_register(ctx, target)
```

8.266.4. Description

JALRS writes the mode-tagged address following its required 16-bit delay-slot instruction, then jumps to the mode-tagged GPR target.

8.266.5. Exceptions

None.

8.267. JALS

8.267.1. Purpose

Jump within the current microMIPS region and link across a short delay slot.

8.267.2. Formats

microMIPS

jals.32

Assembly

JALS {index:u27}

Availability

MMAR < R6

Decode priority

0

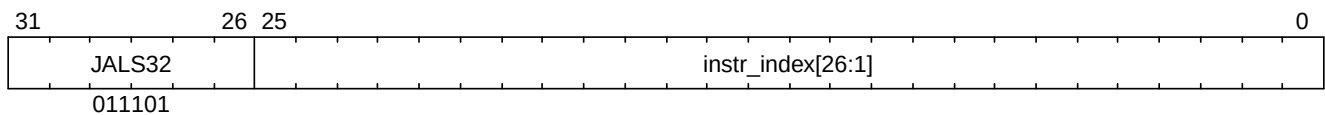


Figure 725. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalOperands(index=zero_extend(instr_index, from_nbits=27))
```

8.267.3. Operation

```
target = (ctx.next_pc & ~0x07FF_FFFF) | int(operands.index)
ctx.GPR[31] = ModeTaggedAddress(
    ctx, effective_address(ctx, ctx.next_pc, 2, access_type="Fetch")
)
delayed_branch(ctx, target)
```

8.267.4. Description

JALS writes the mode-tagged address following its required 16-bit delay-slot instruction to GPR 31.

8.267.5. Exceptions

None.

8.268. JALX

8.268.1. Purpose

Call a target in the current 256 MB region and exchange between MIPS mode and the implemented compressed ISA mode.

8.268.2. Formats

MIPS

jalx

Assembly

JALX {index:u28}

Availability

AR < R6 && (MMAR >= R3_5 && MMAR < nanoMIPS_R6 || MIPS16E)

Decode priority

0

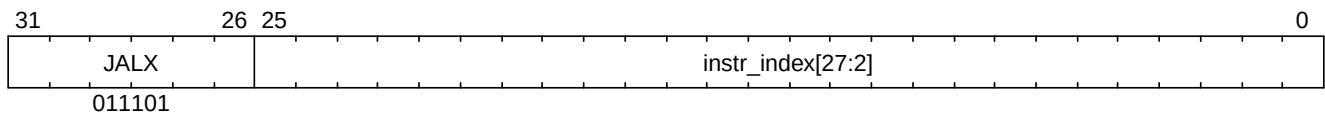


Figure 726. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JalxOperands(index=zero_extend(instr_index, from_nbits=28))
```

microMIPS

jalx.32

Assembly

JALX {index:u28}

Availability

MMAR < R6

Decode priority

0

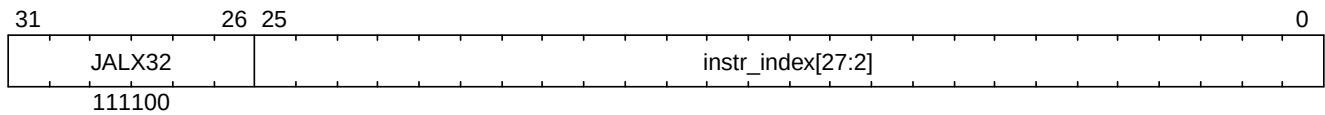


Figure 727. Encoding (32 bits)

```

exec.isCTI = True
exec.hasDelaySlot = True
return JalxOperands(index=zero_extend(instr_index, from_nbits=28))

```

8.268.3. Operation

```

target = (ctx.next_pc & ~0xFFFF_FFFF) | int(operands.index)
ctx.GPR[31] = effective_address(ctx, ctx.pc, 8, access_type="Fetch") | int(
    ctx.ISAMode
)

def exchange_mode_and_jump() -> None:
    ctx.next_pc = target
    ctx.ISAMode = ISAMode(1 - int(ctx.ISAMode))

defer_after_delay_slot(ctx, exchange_mode_and_jump)

```

8.268.4. Description

JALX writes the address following its delay slot to GPR 31, with bit zero recording the caller's ISA mode. After the delay slot it jumps to the region-relative target and toggles the ISA Mode bit. It requires microMIPS or MIPS16E and was removed in Release 6.

8.268.5. Exceptions

None.

8.269. JIALC

8.269.1. Purpose

Perform a compact indirect call through a GPR.

8.269.2. Formats

MIPS

jialc

Assembly

JIALC {offset:s16}

Availability

AR >= R6

Decode priority

0

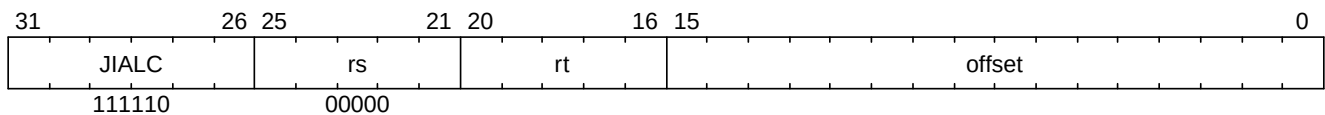


Figure 728. Encoding (32 bits)

Decode

```
exec.isCTI = True
return IndirectCompactOperands(rt=rt, offset=sign_extend(offset, from_nbits=16))
```

microMIPS

jialc.32.micromips

Assembly

JIALC {rt:gpr}, {offset:s16}

Availability

MMAR >= R6

Decode priority

0

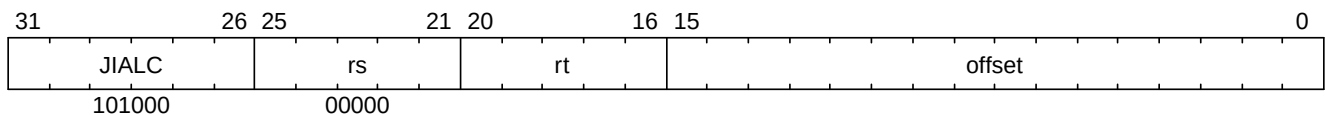


Figure 729. Encoding (32 bits)

Decode

```
exec.isCTI = True
return IndirectCompactOperands(rt=rt, offset=sign_extend(offset, from_nbits=16))
```

8.269.3. Operation

```
try:
    target = effective_address(ctx, ctx.GPR[operands.rt], operands.offset)
except AddressErrorLoad as exception:
    exception.epc = exception.badva
    raise
ctx.GPR[31] = ModeTaggedAddress(ctx, ctx.next_pc)
compact_jump_register(ctx, target)
```

8.269.4. Description

The signed byte displacement is added to the base GPR. The instruction has no delay slot; JIALC writes the sequential PC to GPR 31.

8.269.5. Exceptions

None.

8.270. JIC

8.270.1. Purpose

Perform a compact indirect jump through a GPR.

8.270.2. Formats

MIPS

`jic`

Assembly

`JIC {offset:s16}`

Availability

AR >= R6

Decode priority

0

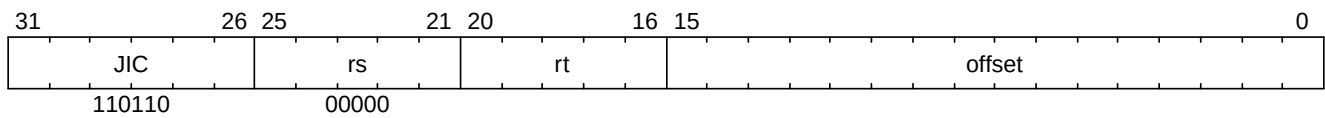


Figure 730. Encoding (32 bits)

Decode

```
exec.isCTI = True
return IndirectCompactOperands(rt=rt, offset=sign_extend(offset, from_nbits=16))
```

microMIPS

`jic.32.micromips`

Assembly

`JIC {rt:gpr}, {offset:s16}`

Availability

MMAR >= R6

Decode priority

0

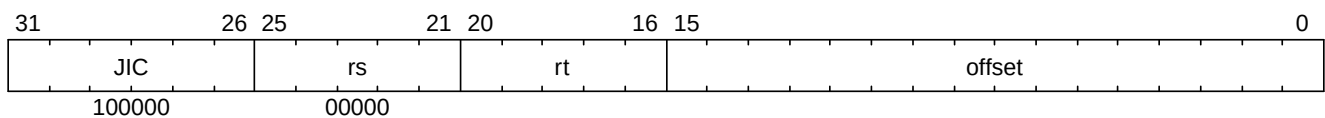


Figure 731. Encoding (32 bits)

Decode

```
exec.isCTI = True
return IndirectCompactOperands(rt=rt, offset=sign_extend(offset, from_nbits=16))
```

8.270.3. Operation

```
try:
    target = effective_address(ctx, ctx.GPR[operands.rt], operands.offset)
except AddressErrorLoad as exception:
    # JIC validates the computed target before committing the compact
    # transfer. The R6 AVP contract reports that invalid target in EPC
    # (and BadVAddr), rather than retaining the JIC instruction address.
    exception.epc = exception.badva
    raise
compact_jump_register(ctx, target)
```

8.270.4. Description

The signed byte displacement is added to the base GPR. The instruction has no delay slot; JIALC writes the sequential PC to GPR 31.

8.270.5. Exceptions

None.

8.271. JR

8.271.1. Purpose

Jump to the address in a GPR after one delay slot.

8.271.2. Formats

MIPS

jr

Assembly

JR {rs:gpr}

Availability

AR < R6

Decode priority

0

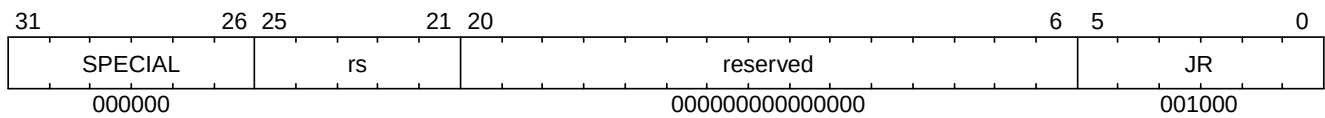


Figure 732. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return Jr0perands(rs=rs)
```

jr.hb

Assembly

JR {rs:gpr}

Availability

AR < R6

Decode priority

0

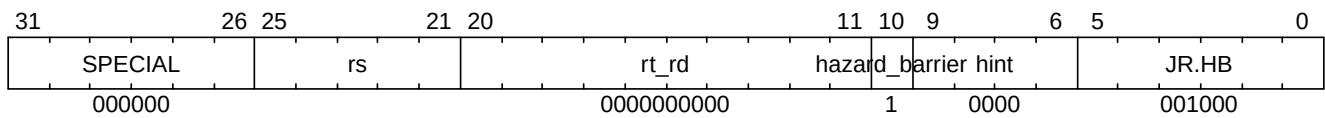


Figure 733. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JrOperands(rs=rs)
```

microMIPS

jr.16

Assembly

JR {rs:gpr}

Availability

MMAR < R6

Decode priority

0

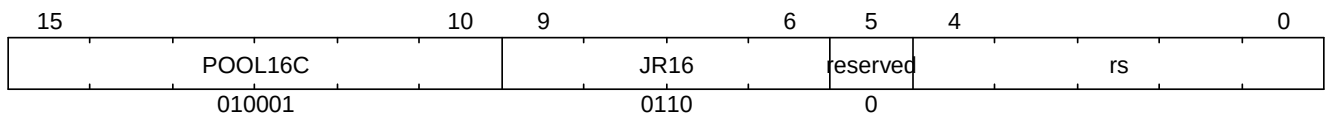


Figure 734. Encoding (16 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return JrOperands(rs=rs)
```

8.271.3. Operation

```
delayed_jump_register(ctx, ctx.GPR[operands.rs])
```

8.271.4. Description

JR is the pre-Release-6 register-indirect delayed jump.

8.271.5. Exceptions

None.

8.272. JRADDIUSP

8.272.1. Purpose

Add a nonnegative immediate to the stack pointer and jump through GPR 31.

8.272.2. Formats

microMIPS

jraddiusp.16

Assembly

JRADDIUSP {imm}

Availability

MMAR < R6

Decode priority

0

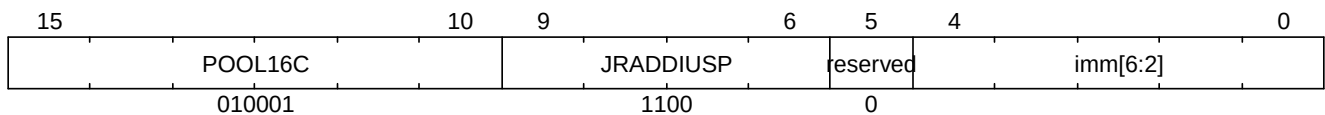


Figure 735. Encoding (16 bits)

Decode

```
exec.isCTI = True
return JraddiuspOperands(imm=zero_extend(imm, from_nbits=7))
```

8.272.3. Operation

```
target = ctx.GPR[31]
ctx.GPR[29] = effective_address(ctx, ctx.GPR[29], int(operands.imm))
compact_jump_register(ctx, target)
```

8.272.4. Description

JRADDIUSP is a compact microMIPS return instruction. It updates GPR 29 by the encoded word-scaled immediate and transfers control to the address held in GPR 31 without executing a delay-slot instruction.

8.272.5. Exceptions

None.

8.273. JRC

8.273.1. Purpose

Perform an unconditional compact jump to the address held in a GPR.

8.273.2. Formats

microMIPS

jrc.16

Assembly

JRC {rt:gpr}

Availability

MMAR < R6

Decode priority

0

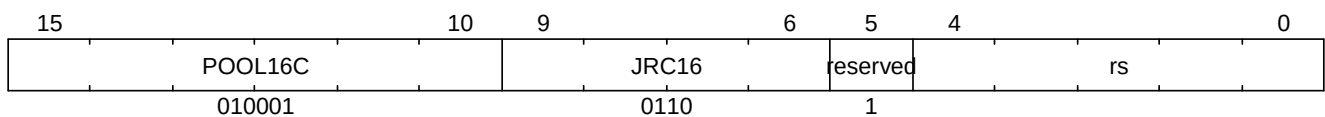


Figure 736. Encoding (16 bits)

Decode

```
exec.isCTI = True
return JrcOperands(rt=rs)
```

jrc.r6.16

Assembly

JRC {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

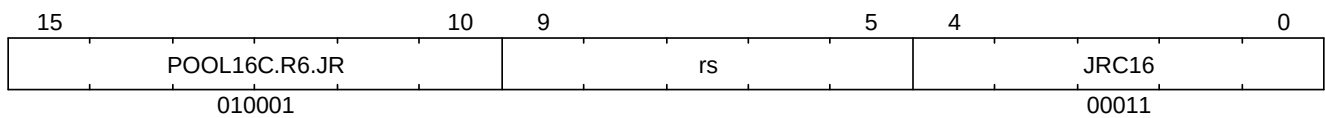


Figure 737. Encoding (16 bits)

Decode

```
exec.isCTI = True
```

```
return JrcOperands(rt=rs)
```

nanoMIPS

jrc.16

Assembly

JRC {rt:gpr}

Availability

true

Decode priority

0

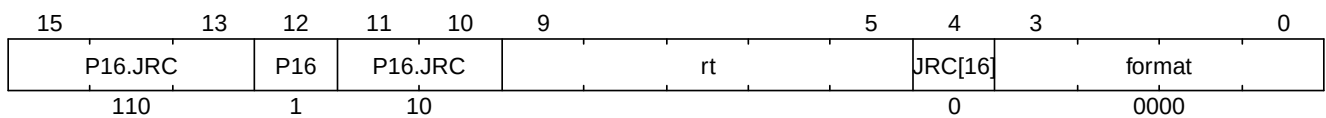


Figure 738. Encoding (16 bits)

Decode

```
exec.isCTI = True
return JrcOperands(rt=rt)
```

8.273.3. Operation

```
address = effective_address(ctx, ctx.GPR[operands.rt], 0)
compact_jump_register(ctx, address)
```

8.273.4. Description

JRC[16] copies the source GPR address to the next-PC state and has no delay slot. The instruction is the compact nanoMIPS return/jump form; JRC32 is an assembler alias for JALRC[32] with destination GPR zero.

8.273.5. Exceptions

None.

8.274. JRCADDIUSP

8.274.1. Purpose

Adjust the stack pointer and compact-jump through the adjusted address.

8.274.2. Formats

microMIPS

`jrcaddiusp.16`

Assembly

`JRCADDIUSP {imm:s5}`

Availability

MMAR >= R6

Decode priority

0

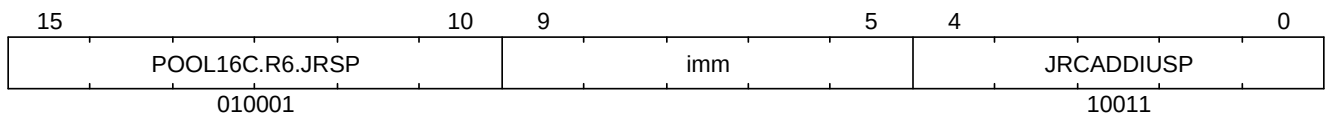


Figure 739. Encoding (16 bits)

Decode

```
exec.isCTI = True
# The microMIPS R6 form encodes a non-negative word count (the AVP
# spelling exposes the resulting byte displacement, 0..124).
return JrcAddiuSpOperands(imm=zero_extend(imm, from_nbits=5))
```

8.274.3. Operation

```
adjustment = operands.imm << 2
ctx.GPR[29] = effective_address(ctx, ctx.GPR[29], adjustment)
compact_jump_register(ctx, ctx.GPR[31])
```

8.274.4. Description

JRCADDIUSP adds a signed five-bit immediate to SP and jumps to the resulting address without a delay slot.

8.274.5. Exceptions

None.

8.275. LB

8.275.1. Purpose

Load a signed byte from memory into a GPR.

8.275.2. Formats

MIPS

lb

Assembly

LB {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

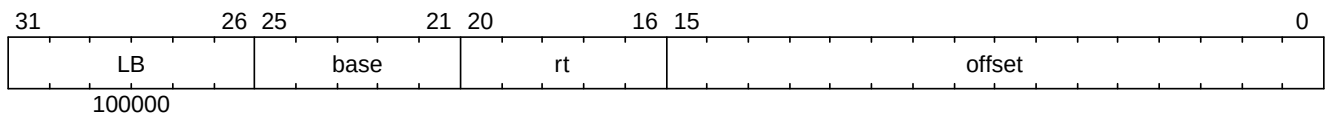


Figure 740. Encoding (32 bits)

Decode

```
return Lb0perands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
)
```

microMIPS

lb.32

Assembly

LB {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

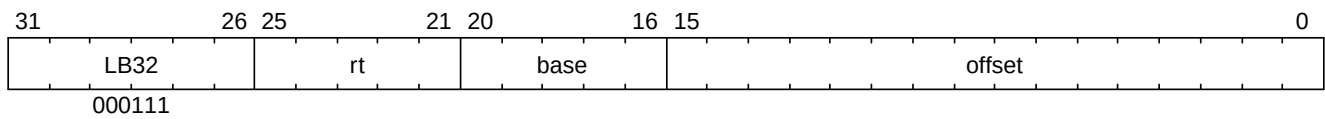


Figure 741. Encoding (32 bits)

Decode

```
return Lb0perands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

nanoMIPS

lb.16

Assembly

```
LB {rt:gpr3}, {offset:u2}({rs:gpr3})
```

Availability

true

Decode priority

0

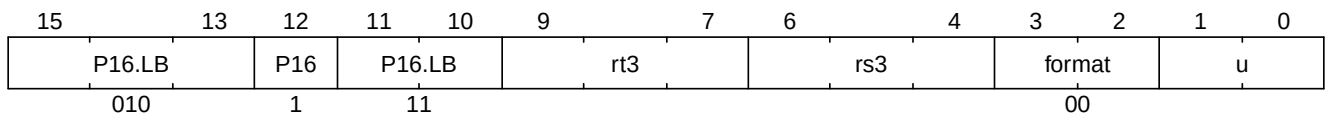


Figure 742. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
offset = zero_extend(u, from_nbits=2)
return Lb0perands(rt=rt, rs=rs, offset=offset)
```

lb.gp

Assembly

```
LB {rt:gpr}, {offset:u18}(gp)
```

Availability

true

Decode priority

0

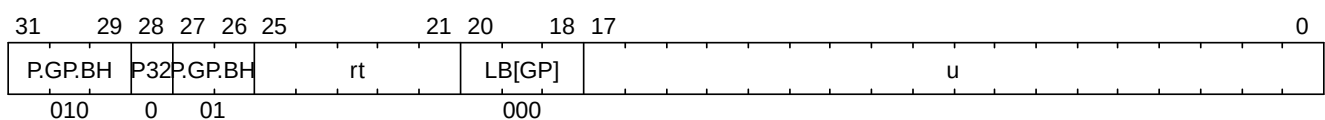


Figure 743. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=18)
return Lb0perands(rt=rt, rs=28, offset=offset)
```

lb.s9

Assembly

LB {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

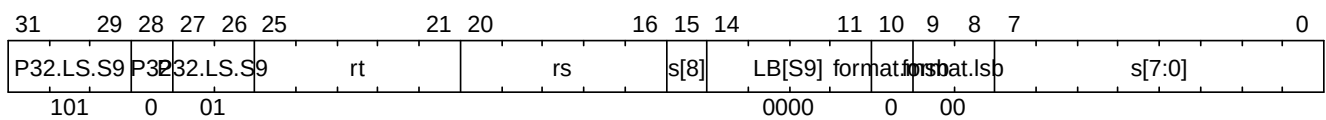


Figure 744. Encoding (32 bits)

Decode

```
offset = sign_extend(s, from_nbits=9)
return Lb0perands(rt=rt, rs=rs, offset=offset)
```

lb.u12

Assembly

LB {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

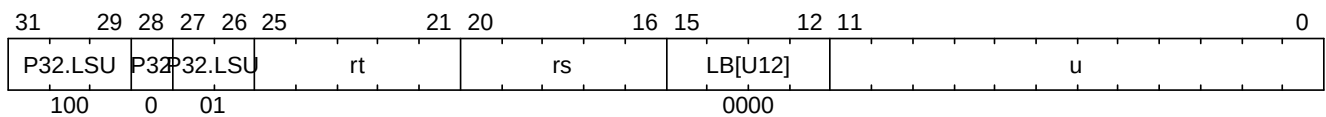


Figure 745. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return Lb0perands(rt=rt, rs=rs, offset=offset)
```

8.275.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
data = read_memory_at_va(ctx, va, nbytes=1)
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=8)
```

8.275.4. Description

LB adds the encoded byte offset to the base GPR, reads one byte through the active memory backend, and sign-extends that byte to the implemented GPR width.

8.275.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.276. LBE

8.276.1. Purpose

Load a signed byte using EVA addressing.

8.276.2. Formats

nanoMIPS

lbe.32

Assembly

LBE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

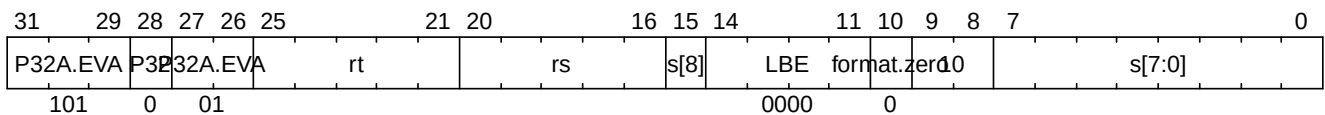


Figure 746. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.276.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load", eva=True)
data = read_memory_at_va(ctx, va, nbytes=1, eva=True)
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=8)
```

8.276.4. Description

This optional privileged form uses EVA user-mode address translation.

8.276.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.277. LBU

8.277.1. Purpose

Load an unsigned byte from memory into a GPR.

8.277.2. Formats

MIPS

`lbu`

Assembly

`LBU {rt:gpr}, {offset:s16}({rs:gpr})`

Availability

true

Decode priority

0

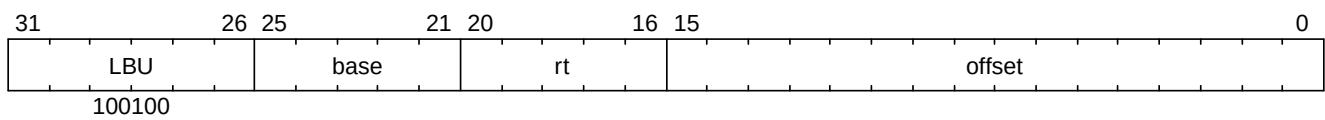


Figure 747. Encoding (32 bits)

Decode

```
return LbuOperands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
)
```

microMIPS

`lbu.32`

Assembly

`LBU {rt:gpr}, {offset:s16}({rs:gpr})`

Availability

true

Decode priority

0

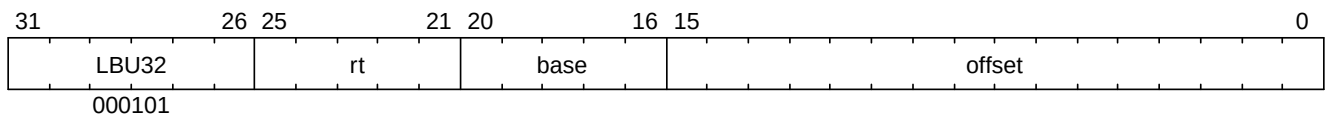


Figure 748. Encoding (32 bits)

Decode

```
return LbuOperands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

lbu.16

Assembly

LBU {rt:gpr3}, {offset:u4}({rs:gpr3})

Availability

true

Decode priority

0

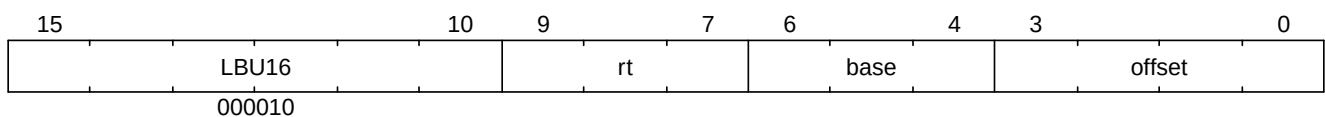


Figure 749. Encoding (16 bits)

Decode

```
return LbuOperands(
    rt=MICROMIPS_GPR3.decode(rt),
    rs=MICROMIPS_GPR3.decode(base),
    offset=-1 if offset == 0xF else zero_extend(offset, from_nbits=4),
)
```

nanoMIPS

lbu.16

Assembly

LBU {rt:gpr3}, {offset:u2}({rs:gpr3})

Availability

true

Decode priority

0

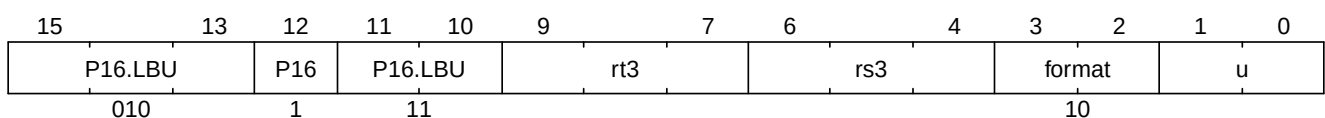


Figure 750. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
offset = zero_extend(u, from_nbits=2)
return Lbu0Operands(rt=rt, rs=rs, offset=offset)
```

lbu.gp

Assembly

LBU {rt:gpr}, {offset:u18}(gp)

Availability

true

Decode priority

0

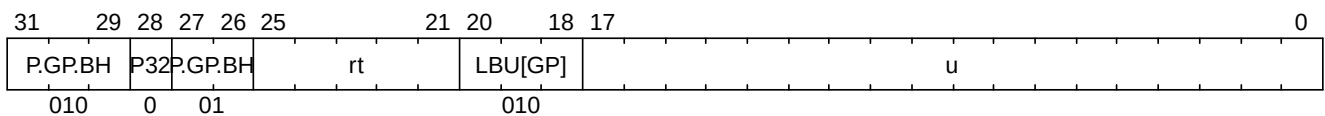


Figure 751. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=18)
return Lbu0Operands(rt=rt, rs=28, offset=offset)
```

lbu.s9

Assembly

LBU {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

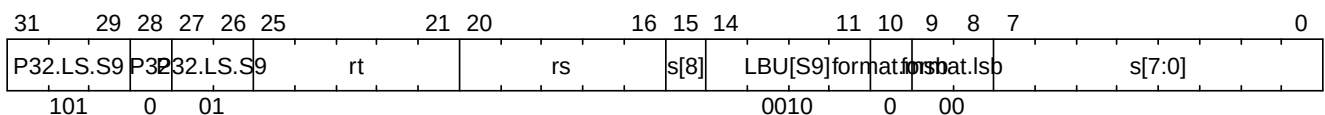


Figure 752. Encoding (32 bits)

Decode

```
offset = sign_extend(s, from_nbits=9)
return Lbu0Operands(rt=rt, rs=rs, offset=offset)
```

lbu.u12

Assembly

LBU {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

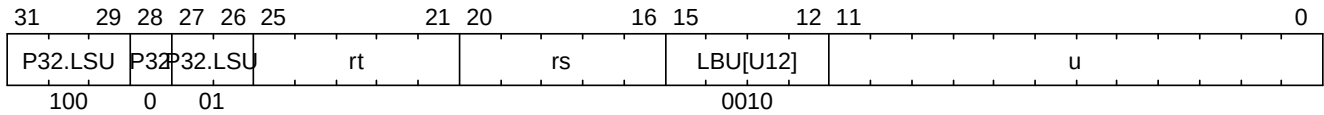


Figure 753. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return Lbu0Operands(rt=rt, rs=rs, offset=offset)
```

8.277.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
data = read_memory_at_va(ctx, va, nbytes=1)
ctx.GPR[operands.rt] = zero_extend(data, from_nbits=8)
```

8.277.4. Description

LBU adds the encoded byte offset to the base GPR, reads one byte through the active memory backend, and zero-extends that byte to the implemented GPR width.

8.277.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.278. LBUE

8.278.1. Purpose

Load an unsigned byte using EVA addressing.

8.278.2. Formats

nanoMIPS

lbue.32

Assembly

LBUE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

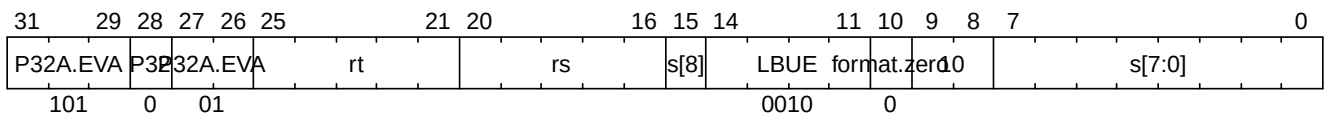


Figure 754. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.278.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load", eva=True)
ctx.GPR[operands.rt] = read_memory_at_va(ctx, va, nbytes=1, eva=True)
```

8.278.4. Description

This optional privileged form uses EVA user-mode address translation.

8.278.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.279. LBUX

8.279.1. Purpose

Load an unsigned byte using a register-indexed address.

8.279.2. Formats

MIPS

`lbux.dsp`

Assembly

`LBUX {rs:gpr}({rt:gpr})`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

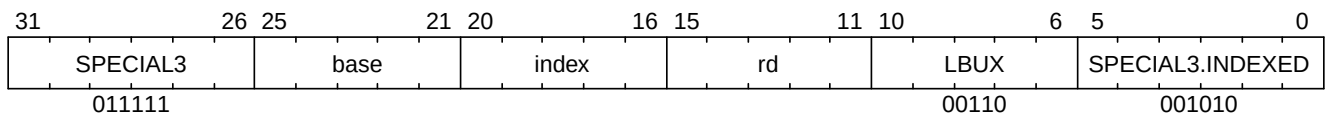


Figure 755. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd=rd, rs=index, rt=base)
```

microMIPS

`lbux.dsp.32`

Assembly

`LBUX {rs:gpr}({rt:gpr})`

Availability

DSP

Decode priority

0

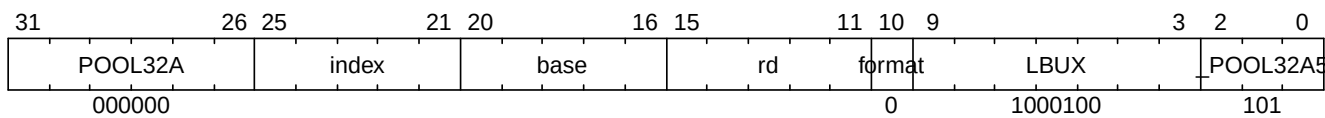


Figure 756. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd=rd, rs=index, rt=base)
```

nanoMIPS

lbux.32

Assembly

LBUX {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

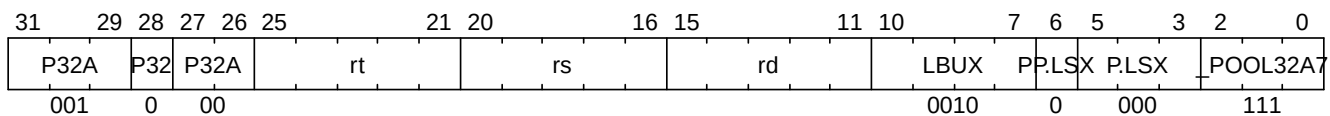


Figure 757. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.279.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Load")
ctx.GPR[operands.rd] = read_memory_at_va(ctx, va, nbytes=1)
```

8.279.4. Description

The effective address is formed from two full-width GPR values.

8.279.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **DSPDIS** (DSP Module State Disabled) — DSP Module resource access is disabled by Status.MX.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.279.6. Programming Notes

The MIPS64 DSP Module Revision 3.02 operation pseudocode describes `vAddr[31:0]` as the sum of `GPR[index][31:0]` and `GPR[base][31:0]`. That truncation is a documentation error. In a 64-bit GPR mode, the effective virtual address is formed by adding the full 64-bit GPR values.

8.280. LBX

8.280.1. Purpose

Load a signed byte using a register-indexed address.

8.280.2. Formats

nanoMIPS

lbx.32

Assembly

LBX {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

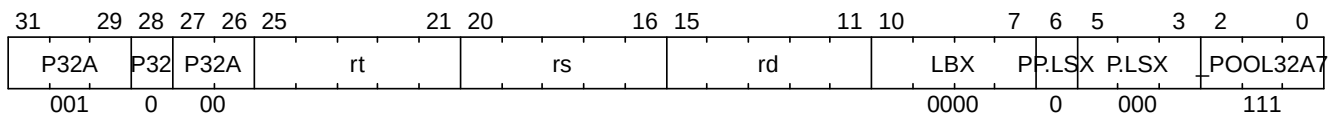


Figure 758. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.280.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Load")
data = read_memory_at_va(ctx, va, nbytes=1)
ctx.GPR[operands.rd] = sign_extend(data, from_nbits=8)
```

8.280.4. Description

The effective address is formed from two full-width GPR values.

8.280.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.281. LD

8.281.1. Purpose

Load a 64-bit doubleword from memory.

8.281.2. Formats

MIPS

ld

Assembly

LD {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

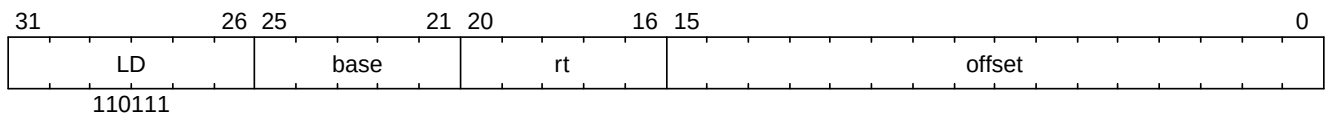


Figure 759. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, base, sign_extend(offset, from_nbits=16))
```

microMIPS

ld

Assembly

LD {rt:gpr}, {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

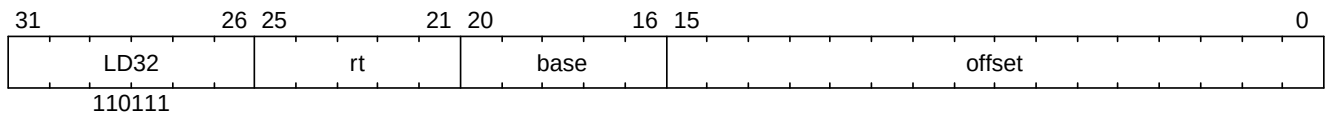


Figure 760. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, base, sign_extend(offset, from_nbits=16))
```

nanoMIPS

ld.gp

Assembly

```
LD {rt:gpr}, {offset:u21}(gp)
```

Availability

AT >= MIPS64_32

Decode priority

0



Figure 761. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, 28, zero_extend(u, from_nbits=21))
```

ld.s9

Assembly

```
LD {rt:gpr}, {offset:s9}({rs:gpr})
```

Availability

AT >= MIPS64_32

Decode priority

0

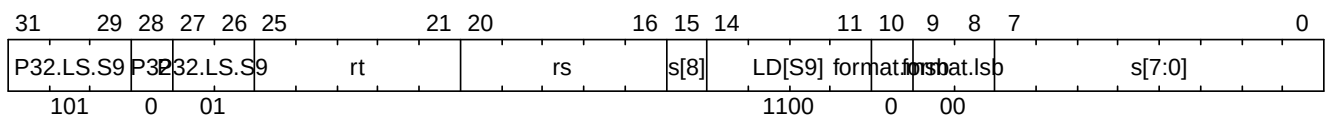


Figure 762. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

ld.u12

Assembly

```
LD {rt:gpr}, {offset:u12}({rs:gpr})
```

Availability

AT >= MIPS64_32

Decode priority

0

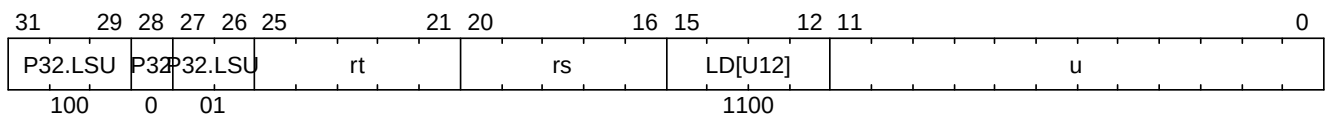


Figure 763. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, rs, zero_extend(u, from_nbits=12))
```

8.281.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_memory_access(ctx, operands, nbytes=8, signed_load=True, store=False)
```

8.281.4. Description

LD forms a full-width effective address and sign-extends the loaded doubleword to the GPR width.

8.281.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.282. LDC1

8.282.1. Purpose

LDC1 a CP1 register to or from memory.

8.282.2. Formats

MIPS

`ldc1`

Assembly

`LDC1 {ft:fpr}, {offset:s16}({base:gpr})`

Availability

true

Decode priority

0

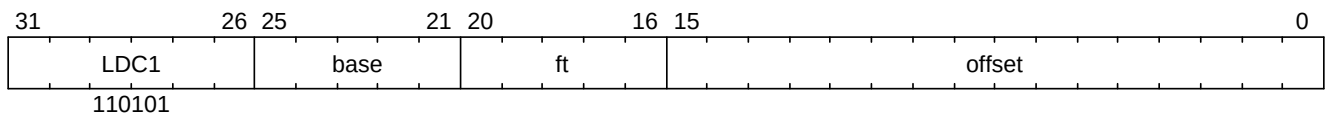


Figure 764. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(  
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)  
)
```

microMIPS

`ldc1.32`

Assembly

`LDC1 {ft:fpr}, {offset:s16}({base:gpr})`

Availability

true

Decode priority

0

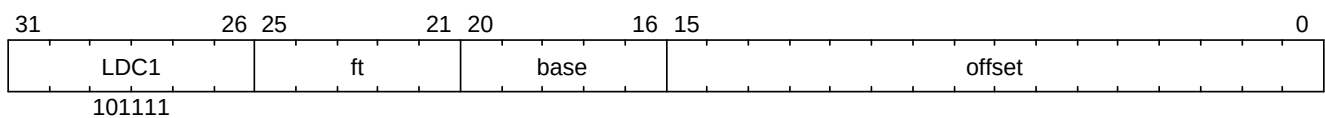


Figure 765. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)
)
```

nanoMIPS

ldc1.gp

Assembly

LDC1 {ft:fpr}, ...

Availability

true

Decode priority

0

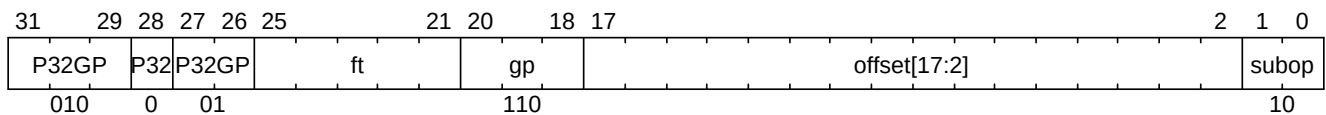


Figure 766. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(
    ft=ft, base=28, offset=zero_extend(offset, from_nbits=16) << 2
)
```

1dc1.s9

Assembly

LDC1 {ft:fpr}, ...

Availability

true

Decode priority

0

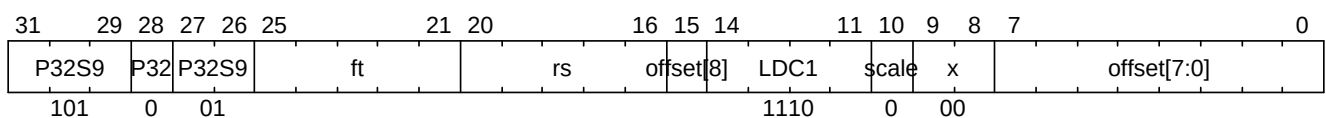


Figure 767. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=sign_extend(offset, from_nbits=9))
```

ldc1.u12

Assembly

LDC1 {ft:fpr}, ...

Availability

true

Decode priority

0

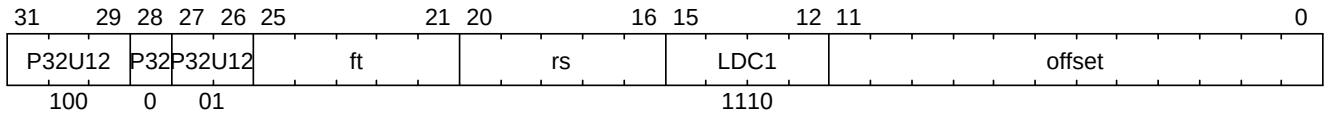


Figure 768. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=zero_extend(offset, from_nbits=12))
```

8.282.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
set_fpr(ctx, operands.ft, read_memory_at_va(ctx, va, nbytes=8), "64")
```

8.282.4. Description

LDC1 performs CP1 access checks before its memory access.

8.282.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.; The CP1 load address is not valid.
- **ADES** (Address Error) — The CP1 store address is not valid.
- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the required FPU resource is unavailable.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.; The CP1

load address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.283. LDC2

8.283.1. Purpose

Load a doubleword from memory into a CP2 register.

8.283.2. Formats

microMIPS

ldc2.32

Assembly

LDC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR < R6

Decode priority

0

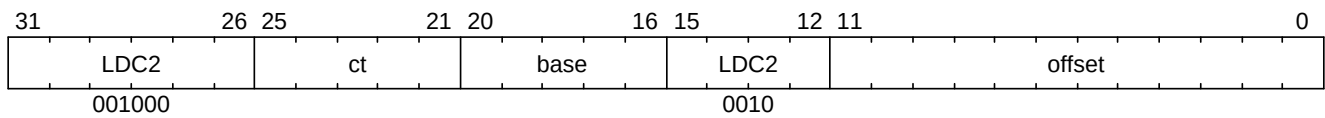


Figure 769. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

ldc2.r6.32

Assembly

LDC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR >= R6

Decode priority

0

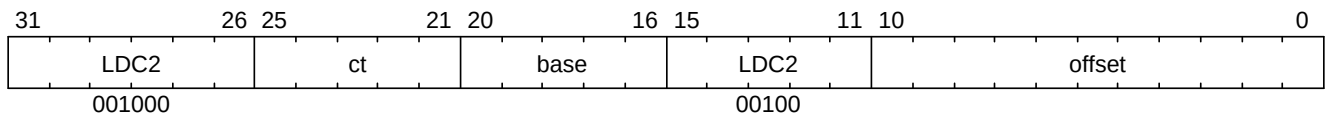


Figure 770. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

8.283.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("LDC2 execution is not implemented")
```

8.283.4. Description

LDC2 performs the CP2 access check before any CP2 operation is attempted.

8.283.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; LDC2 execution is not implemented

8.284. LDL

8.284.1. Purpose

Perform the legacy unaligned doubleword LDL operation.

8.284.2. Formats

MIPS

ldl

Assembly

LDL {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

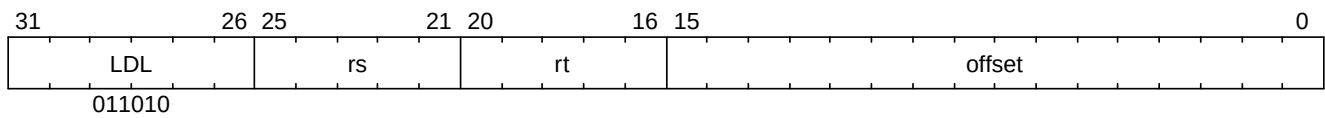


Figure 771. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=16)
)
```

microMIPS

ldl.32

Assembly

LDL {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6 && AT >= MIPS64_32

Decode priority

0

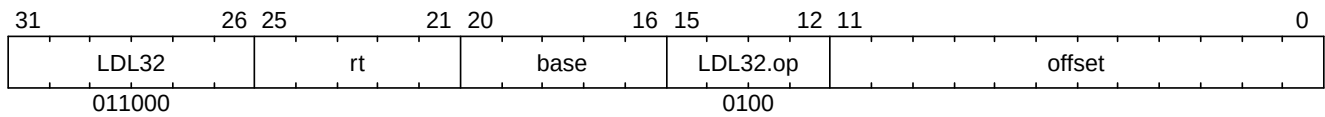


Figure 772. Encoding (32 bits)

```
del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=base, offset=sign_extend(offset, from_nbits=12)
)
```

8.284.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_legacy_load(ctx, operands, left=True)
```

8.284.4. Description

The naturally aligned doubleword containing the effective address is merged according to the active memory endianness. This legacy MIPS64 instruction is unavailable in Release 6.

8.284.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.285. LDM

8.285.1. Purpose

Load multiple doublewords into sequential GPRs.

8.285.2. Formats

nanoMIPS

ldm.32

Assembly

LDM {rt:gpr}, {offset:s9}({rs:gpr}), {count:u3}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

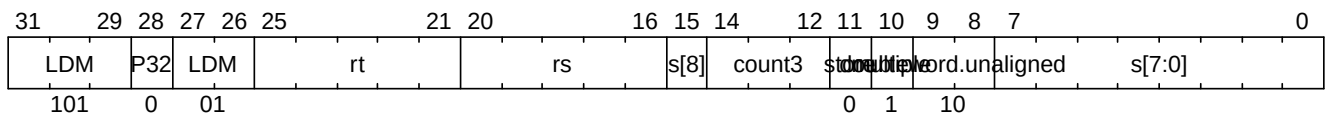


Figure 773. Encoding (32 bits)

Decode

```
return DoubleMultipleOperands(  
    rt=rt,  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    count=8 if count3 == 0 else count3,  
)
```

8.285.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_multiple_access(ctx, operands, store=False, unaligned=False)
```

8.285.4. Description

LDM transfers signed 64-bit elements from consecutive aligned addresses, wrapping register numbers after r31 to r16.

8.285.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.286. LDPC

8.286.1. Purpose

Load a doubleword from a PC-relative address.

8.286.2. Formats

MIPS

ldpc

Assembly

LDPC {rt:gpr}, {offset:s21}

Availability

AR >= R6 && AT >= MIPS64_32

Decode priority

0

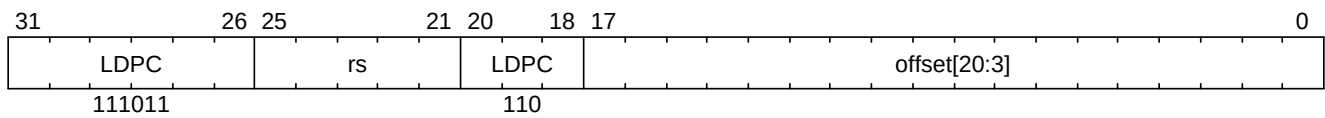


Figure 774. Encoding (32 bits)

Decode

```
return DoublePcRelativeOperands(  
    rs,  
    sign_extend(offset, from_nbits=21),  
    True,  
)
```

microMIPS

ldpc.32.micromips

Assembly

LDPC {rt:gpr}, {offset:s21}

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

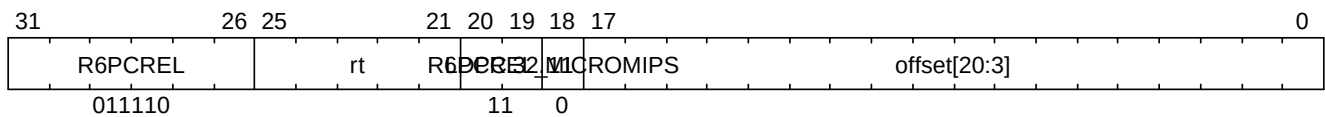


Figure 775. Encoding (32 bits)

Decode

```
del ctx, exec
return DoublePcRelativeOperands(
    rt,
    sign_extend(offset, from_nbits=21),
    True,
)
```

nanoMIPS

ldpc.48

Assembly

LDPC {rt:gpr}, {offset:s32}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

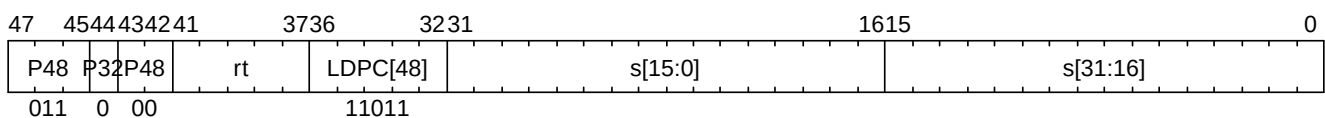


Figure 776. Encoding (48 bits)

Decode

```
return DoublePcRelativeOperands(rt, sign_extend(s, from_nbits=32), False)
```

8.286.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_pc_relative_access(ctx, operands, store=False)
```

8.286.4. Description

The MIPS form uses a signed, eight-byte-scaled displacement from the current eight-byte-aligned PC; LDPC[48] uses a signed 32-bit displacement from the following-instruction address.

8.286.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.287. LDR

8.287.1. Purpose

Perform the legacy unaligned doubleword LDR operation.

8.287.2. Formats

MIPS

`ldr`

Assembly

`LDR {offset:s16}({rs:gpr})`

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

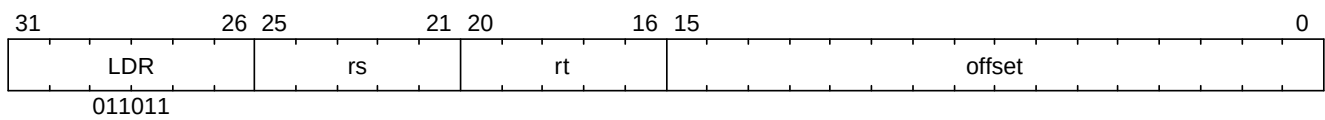


Figure 777. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=16)
)
```

microMIPS

`ldr.32`

Assembly

`LDR {rt:gpr}, {offset:s12}({rs:gpr})`

Availability

MMAR < R6 && AT >= MIPS64_32

Decode priority

0

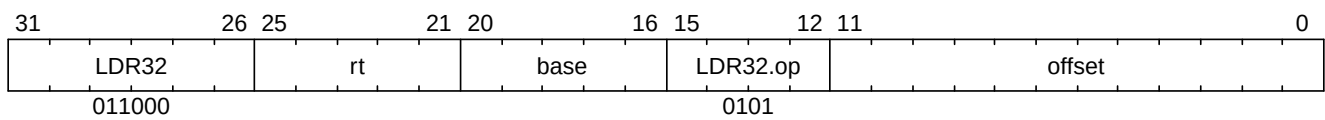


Figure 778. Encoding (32 bits)

```
del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=base, offset=sign_extend(offset, from_nbits=12)
)
```

8.287.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_legacy_load(ctx, operands, left=False)
```

8.287.4. Description

The naturally aligned doubleword containing the effective address is merged according to the active memory endianness. This legacy MIPS64 instruction is unavailable in Release 6.

8.287.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.288. LDX

8.288.1. Purpose

Load a doubleword using a register-indexed address.

8.288.2. Formats

nanoMIPS

ldx

Assembly

LDX {rs:gpr}({rt:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

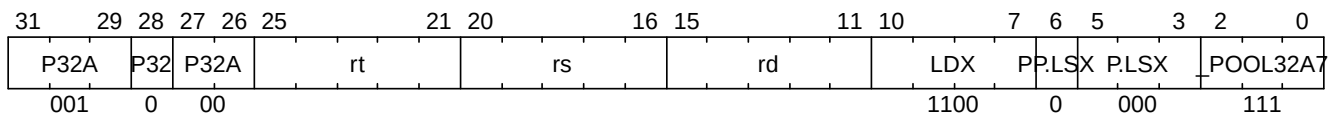


Figure 779. Encoding (32 bits)

Decode

```
return IndexedDoubleMemoryOperands(rd, rs, rt)
```

8.288.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_indexed_double_memory_access(
    ctx, operands, nbytes=8, signed_load=True, scale=0, store=False
)
```

8.288.4. Description

LDX adds two full-width GPR values to form the effective address.

8.288.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.289. LDXC1

8.289.1. Purpose

Indexed LDXC1 access through Coprocessor 1.

8.289.2. Formats

MIPS

`ldxc1`

Assembly

`LDXC1 {fd:fpr}, {index:gpr}({base:gpr})`

Availability

AR >= R2_3_5 && AR < R6

Decode priority

0

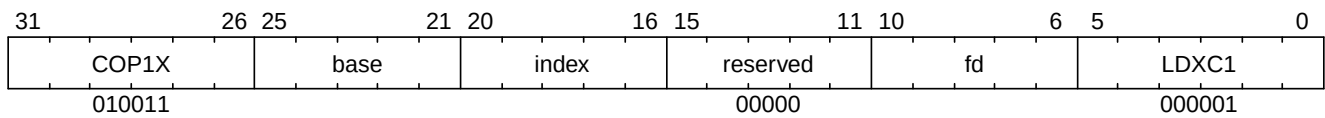


Figure 780. Encoding (32 bits)

Decode

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

microMIPS

`ldxc1.32`

Assembly

`LDXC1 {fd:fpr}, {index:gpr}({base:gpr})`

Availability

MMAR < R6

Decode priority

0

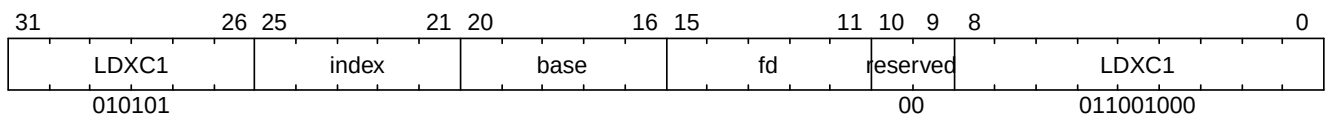


Figure 781. Encoding (32 bits)

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

8.289.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
index = sign_extend(ctx.GPR[operands.index], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, index, "Load")
set_fpr(ctx, operands.fd, read_memory_at_va(ctx, va, nbytes=8), "64")
```

8.289.4. Description

LDXC1 performs its CP1 access checks before transferring data.

8.289.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.; The CP1 load address is not valid.
- **ADES** (Address Error) — The CP1 store address is not valid.
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.; The CP1 load address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.290. LDXS

8.290.1. Purpose

Load a doubleword using a scaled register-indexed address.

8.290.2. Formats

nanoMIPS

ldxs

Assembly

LDXS {rs:gpr}({rt:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

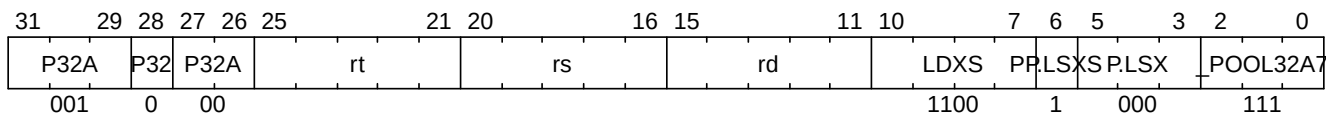


Figure 782. Encoding (32 bits)

Decode

```
return IndexedDoubleMemoryOperands(rd, rs, rt)
```

8.290.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_indexed_double_memory_access(
    ctx, operands, nbytes=8, signed_load=True, scale=3, store=False
)
```

8.290.4. Description

LDXS adds the base GPR to eight times the index GPR.

8.290.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.291. LH

8.291.1. Purpose

Load a signed 16-bit halfword from memory into a GPR.

8.291.2. Formats

MIPS

lh

Assembly

LH {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

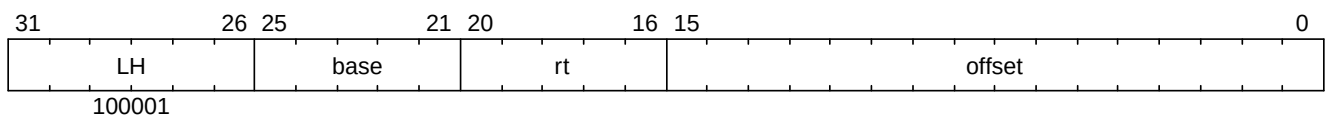


Figure 783. Encoding (32 bits)

Decode

```
return Lh0perands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
)
```

microMIPS

lh.32

Assembly

LH {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

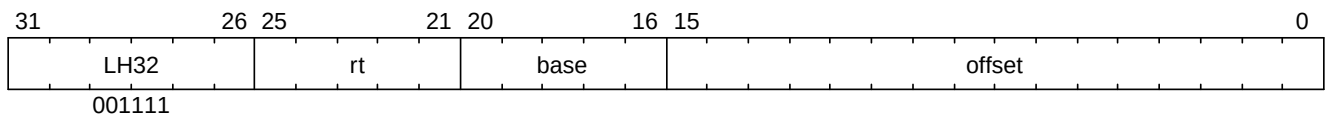


Figure 784. Encoding (32 bits)

Decode

```
return LhOperands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

nanoMIPS

lh.16

Assembly

```
LH {rt:gpr3}, {offset:u3}({rs:gpr3})
```

Availability

true

Decode priority

0

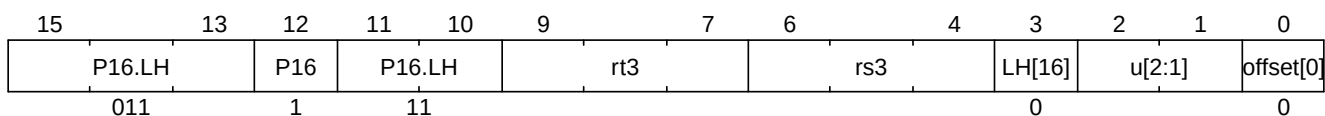


Figure 785. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
offset = zero_extend(u, from_nbits=3)
return LhOperands(rt=rt, rs=rs, offset=offset)
```

lh.gp

Assembly

```
LH {rt:gpr}, {offset:u18}($gp)
```

Availability

true

Decode priority

0

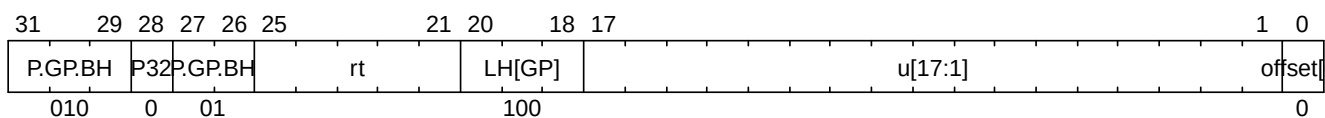


Figure 786. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=18)
return Lh0perands(rt=rt, rs=28, offset=offset)
```

lh.s9

Assembly

LH {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

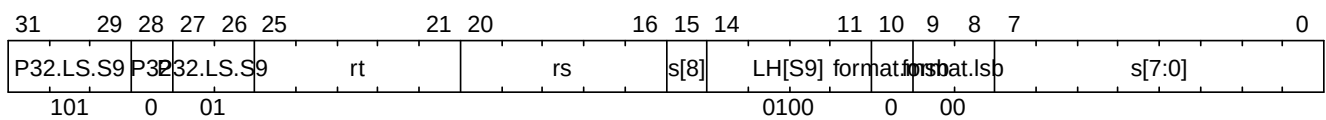


Figure 787. Encoding (32 bits)

Decode

```
offset = sign_extend(s, from_nbits=9)
return Lh0perands(rt=rt, rs=rs, offset=offset)
```

lh.u12

Assembly

LH {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

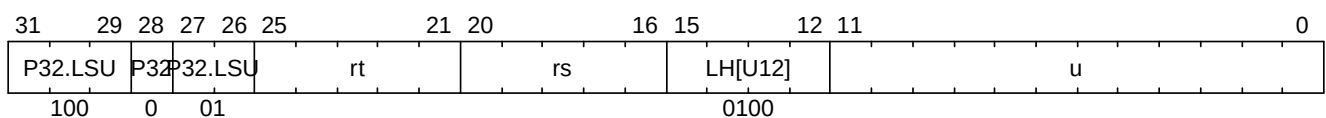


Figure 788. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return Lh0perands(rt=rt, rs=rs, offset=offset)
```

8.291.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
data = read_memory_at_va(ctx, va, nbytes=2)
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=16)
```

8.291.4. Description

The effective address is the full-width base GPR plus the decoded offset. Two naturally aligned bytes are read through the active memory backend and sign-extended to the implemented GPR width.

8.291.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.292. LHE

8.292.1. Purpose

Load a signed halfword using EVA addressing.

8.292.2. Formats

nanoMIPS

lhe.32

Assembly

LHE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

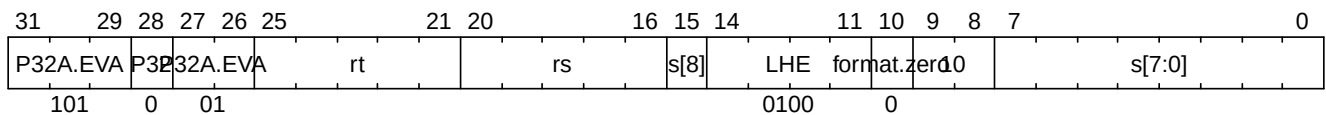


Figure 789. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.292.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load", eva=True)
data = read_memory_at_va(ctx, va, nbytes=2, eva=True)
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=16)
```

8.292.4. Description

This optional privileged form uses EVA user-mode address translation.

8.292.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.293. LHU

8.293.1. Purpose

Load an unsigned 16-bit halfword from memory into a GPR.

8.293.2. Formats

MIPS

lhu

Assembly

LHU {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

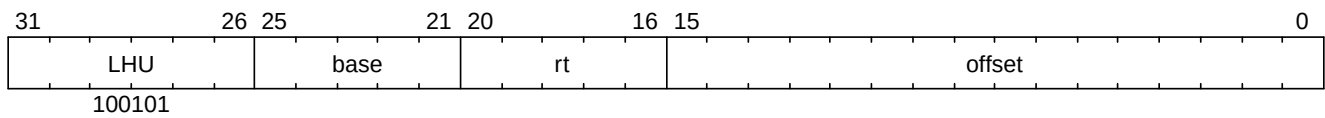


Figure 790. Encoding (32 bits)

Decode

```
return LhuOperands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
)
```

microMIPS

lhu.32

Assembly

LHU {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

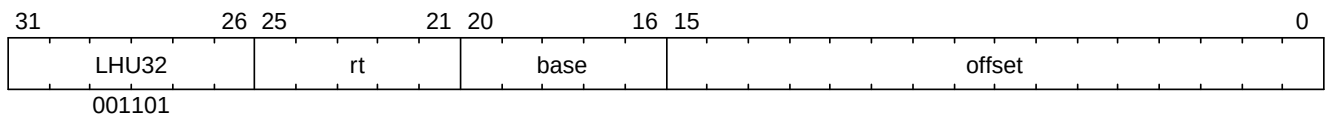


Figure 791. Encoding (32 bits)

Decode

```
return LhuOperands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

lhu.16

Assembly

LHU {rt:gpr3}, {offset:u4}({rs:gpr3})

Availability

true

Decode priority

0

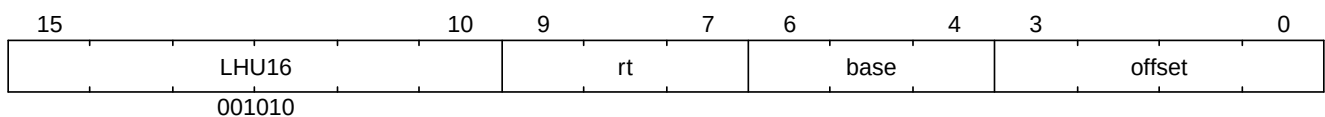


Figure 792. Encoding (16 bits)

Decode

```
return LhuOperands(
    rt=MICROMIPS_GPR3.decode(rt),
    rs=MICROMIPS_GPR3.decode(base),
    offset=zero_extend(offset, from_nbits=4) << 1,
)
```

nanoMIPS

lhu.16

Assembly

LHU {rt:gpr3}, {offset:u3}({rs:gpr3})

Availability

true

Decode priority

0

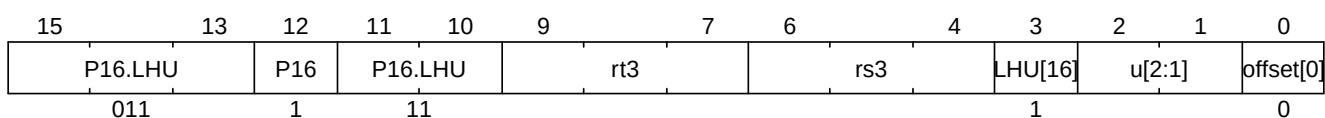


Figure 793. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
offset = zero_extend(u, from_nbits=3)
return Lhu0Operands(rt=rt, rs=rs, offset=offset)
```

lhu.gp

Assembly

LHU {rt:gpr}, {offset:u18}(\$gp)

Availability

true

Decode priority

0



Figure 794. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=18)
return Lhu0Operands(rt=rt, rs=28, offset=offset)
```

lhu.s9

Assembly

LHU {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

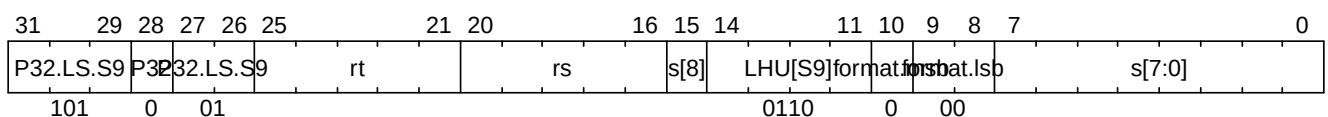


Figure 795. Encoding (32 bits)

Decode

```
offset = sign_extend(s, from_nbits=9)
return Lhu0Operands(rt=rt, rs=rs, offset=offset)
```

lhu.u12

Assembly

LHU {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

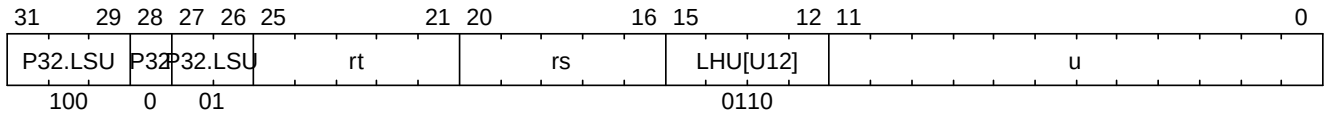


Figure 796. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return Lhu0Operands(rt=rt, rs=rs, offset=offset)
```

8.293.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
data = read_memory_at_va(ctx, va, nbytes=2)
ctx.GPR[operands.rt] = zero_extend(data, from_nbits=16)
```

8.293.4. Description

The effective address is the full-width base GPR plus the decoded offset. Two naturally aligned bytes are read through the active memory backend and zero-extended to the implemented GPR width.

8.293.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.294. LHUE

8.294.1. Purpose

Load an unsigned halfword using EVA addressing.

8.294.2. Formats

nanoMIPS

lhue.32

Assembly

LHUE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

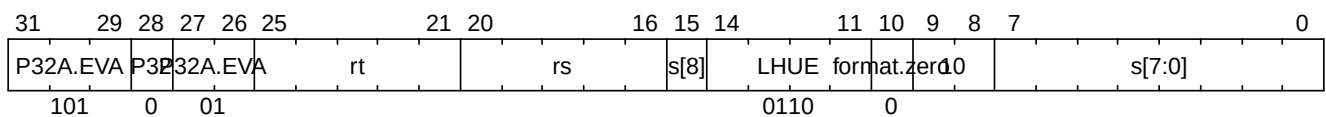


Figure 797. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.294.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load", eva=True)
ctx.GPR[operands.rt] = read_memory_at_va(ctx, va, nbytes=2, eva=True)
```

8.294.4. Description

This optional privileged form uses EVA user-mode address translation.

8.294.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.295. LHUX

8.295.1. Purpose

Load an unsigned halfword using a register-indexed address.

8.295.2. Formats

nanoMIPS

lhux.32

Assembly

LHUX {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

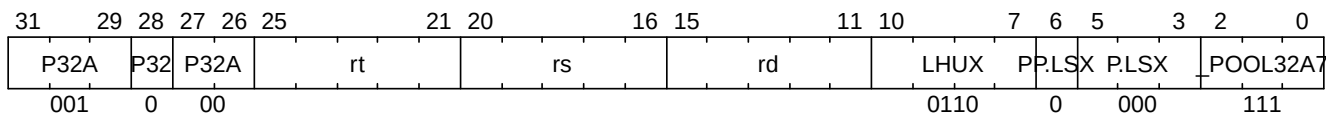


Figure 798. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.295.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Load")
ctx.GPR[operands.rd] = read_memory_at_va(ctx, va, nbytes=2)
```

8.295.4. Description

The effective address is formed from two full-width GPR values.

8.295.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.296. LHUXS

8.296.1. Purpose

Load an unsigned halfword using a scaled register-indexed address.

8.296.2. Formats

nanoMIPS

lhuxs.32

Assembly

LHUXS {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

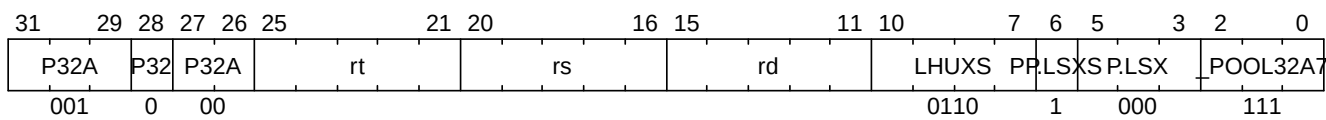


Figure 799. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.296.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)) << 1
va = effective_address(ctx, base, index, "Load")
ctx.GPR[operands.rd] = read_memory_at_va(ctx, va, nbytes=2)
```

8.296.4. Description

The effective address is formed from two full-width GPR values.

8.296.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.297. LHX

8.297.1. Purpose

Load a signed halfword using a register-indexed address.

8.297.2. Formats

MIPS

lhx.dsp

Assembly

LHX {rs:gpr}({rt:gpr})

Availability

AR >= R2_3_5 && DSP

Decode priority

0

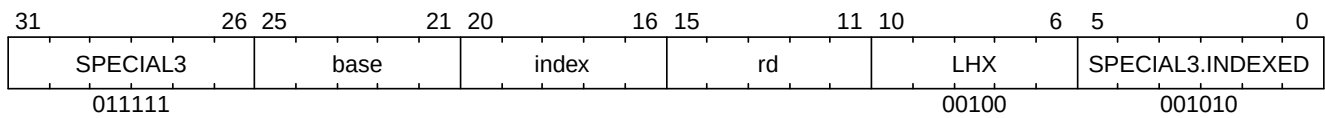


Figure 800. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd=rd, rs=index, rt=base)
```

microMIPS

lhx.dsp.32

Assembly

LHX {rs:gpr}({rt:gpr})

Availability

DSP

Decode priority

0

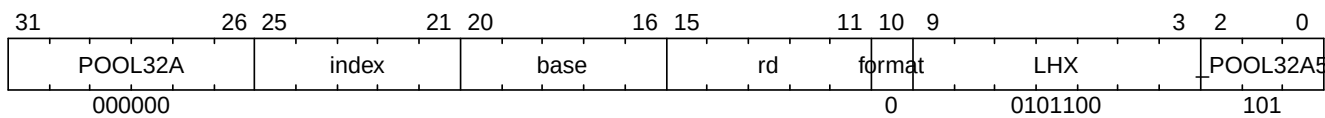


Figure 801. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd=rd, rs=index, rt=base)
```

nanoMIPS

lhx.32

Assembly

LHX {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

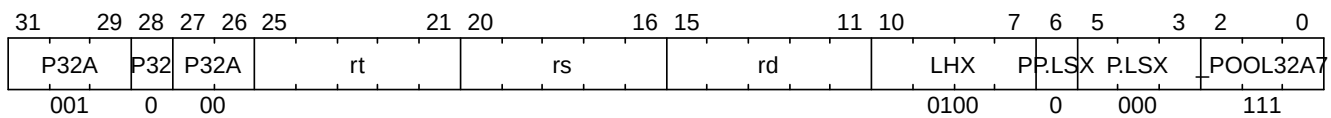


Figure 802. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.297.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Load")
data = read_memory_at_va(ctx, va, nbytes=2)
ctx.GPR[operands.rd] = sign_extend(data, from_nbits=16)
```

8.297.4. Description

The effective address is formed from two full-width GPR values.

8.297.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **DSPDIS** (DSP Module State Disabled) — DSP Module resource access is disabled by Status.MX.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.297.6. Programming Notes

The MIPS64 DSP Module Revision 3.02 operation pseudocode describes `vAddr[31:0]` as the sum of `GPR[index][31:0]` and `GPR[base][31:0]`. That truncation is a documentation error. In a 64-bit GPR mode, the effective virtual address is formed by adding the full 64-bit GPR values.

8.298. LHXS

8.298.1. Purpose

Load a signed halfword using a scaled register-indexed address.

8.298.2. Formats

nanoMIPS

lhxs.32

Assembly

LHXS {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

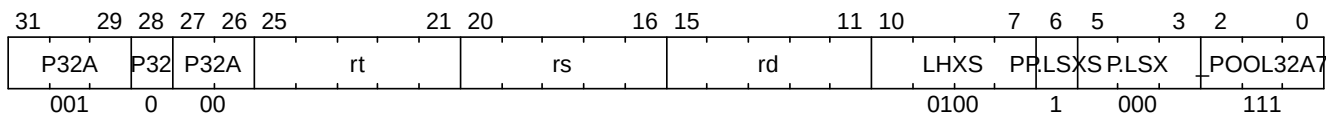


Figure 803. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.298.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)) << 1
va = effective_address(ctx, base, index, "Load")
data = read_memory_at_va(ctx, va, nbytes=2)
ctx.GPR[operands.rd] = sign_extend(data, from_nbits=16)
```

8.298.4. Description

The effective address is formed from two full-width GPR values.

8.298.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.299. LI

8.299.1. Purpose

Load an immediate value into a GPR.

8.299.2. Formats

microMIPS

li.16

Assembly

LI {rt:gpr3}, {imm:u7}

Availability

true

Decode priority

0

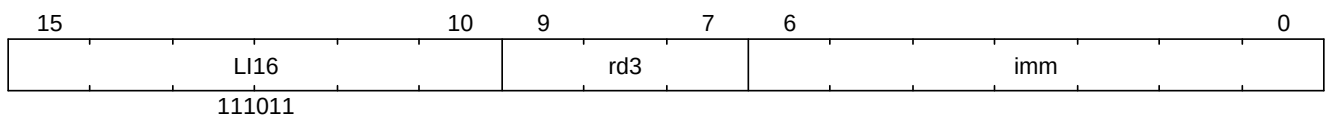


Figure 804. Encoding (16 bits)

Decode

```
return LiOperands(  
    rt=MICROMIPS_GPR3.decode(rd3),  
    imm=sign_extend(imm, from_nbits=7)  
    if imm == 0x7F  
    else zero_extend(imm, from_nbits=7),  
)
```

nanoMIPS

li.16

Assembly

LI {rt:gpr3}, {imm:s7}

Availability

true

Decode priority

0

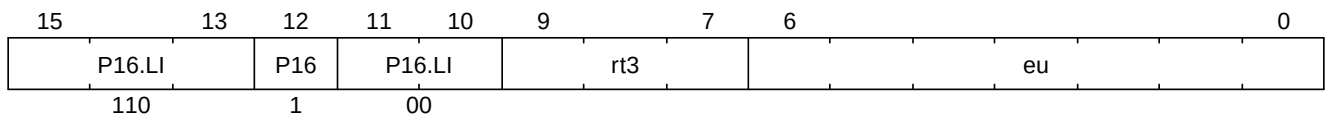


Figure 805. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
imm = sign_extend(-1 if eu == 0x7F else eu, from_nbits=8)
return LiOperands(rt=rt, imm=imm)
```

li.48

Assembly

LI {rt:gpr}, {imm:s32}

Availability

!(NMS)

Decode priority

0

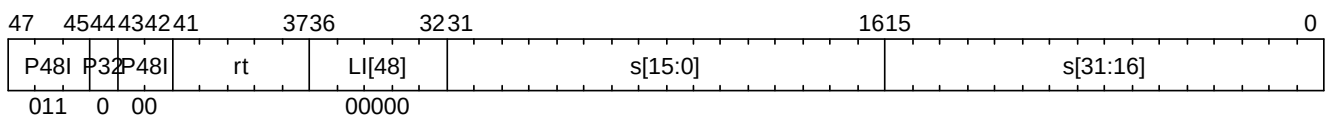


Figure 806. Encoding (48 bits)

Decode

```
imm = sign_extend(s, from_nbits=32)
return LiOperands(rt=rt, imm=imm)
```

8.299.3. Operation

```
ctx.GPR[operands.rt] = operands.imm
```

8.299.4. Description

LI[16] uses compact GPR3 encoding and represents immediate values 0 through 126 directly; encoded eu=127 represents -1. LI[48] encodes a 32-bit immediate across two 16-bit parcels and sign-extends it to the implemented GPR width. LI[48] is not available when the NMS architecture feature is enabled.

8.299.5. Exceptions

- **RI** (Reserved Instruction) — LI[48] is reserved on NMS cores.

8.300. LL

8.300.1. Purpose

Load a word and establish a load-linked reservation.

8.300.2. Formats

MIPS

ll

Assembly

LL {rt:gpr}, {offset:s16}({rs:gpr})

Availability

AR < R6

Decode priority

0

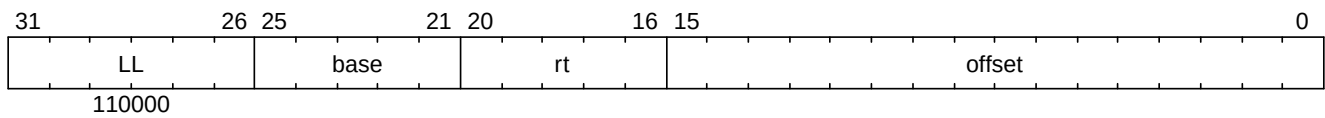


Figure 807. Encoding (32 bits)

Decode

```
return Ll0perands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
    ru=None,  
)
```

ll.r6

Assembly

LL {rt:gpr}, {offset:s9}({rs:gpr})

Availability

AR >= R6

Decode priority

0

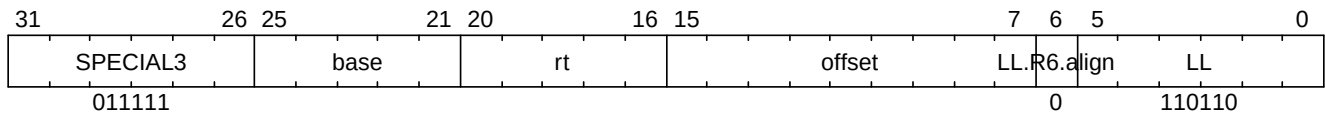


Figure 808. Encoding (32 bits)

Decode

```
return Ll0perands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=9),  
    ru=None,  
)
```

microMIPS

11.32

Assembly

LL {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6

Decode priority

0

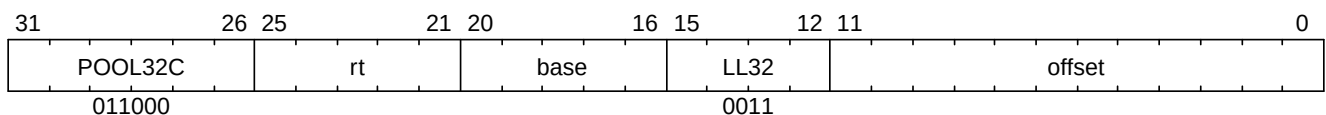


Figure 809. Encoding (32 bits)

Decode

```
return Ll0perands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=12),  
    ru=None,  
)
```

11.r6.32

Assembly

LL {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR >= R6

Decode priority

0

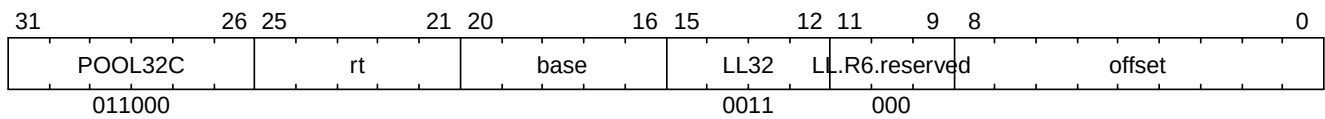


Figure 810. Encoding (32 bits)

Decode

```
return Ll0operands(
    rt=rt,
    rs=base,
    offset=sign_extend(offset, from_nbits=9),
    ru=None,
)
```

nanoMIPS

ll.32

Assembly

LL {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

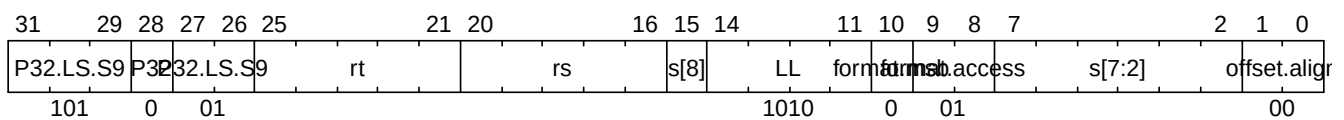


Figure 811. Encoding (32 bits)

Decode

```
return Ll0operands(
    rt=rt,
    rs=rs,
    offset=sign_extend(s, from_nbits=9),
    ru=None,
)
```

8.300.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
if va & 3:
    raise AddressErrorLoad(va, "unaligned linked load")
record_linked_address(ctx, va, nbytes=4)
first = read_memory_at_va(ctx, va, nbytes=4)
```

```
ctx.GPR[operands.rt] = sign_extend(first, from_nbits=32)
```

8.300.4. Description

LL computes a sign-extended effective address, loads a 32-bit word, sign-extends it to the active GPR width, sets CP0 LLAddr.LLB, and records the translated address in implementation-dependent state.

8.300.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.301. LLD

8.301.1. Purpose

Load a doubleword and establish a linked reservation.

8.301.2. Formats

MIPS

lld

Assembly

LLD {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

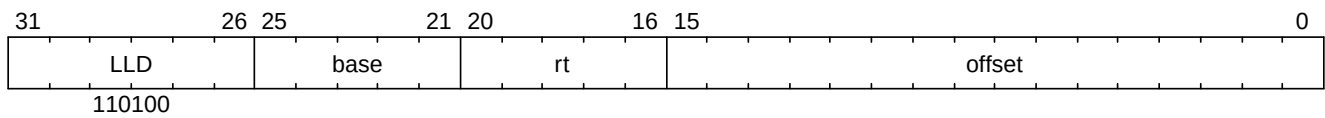


Figure 812. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=16), None)
```

lld.r6

Assembly

LLD {offset:s9}({rs:gpr})

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

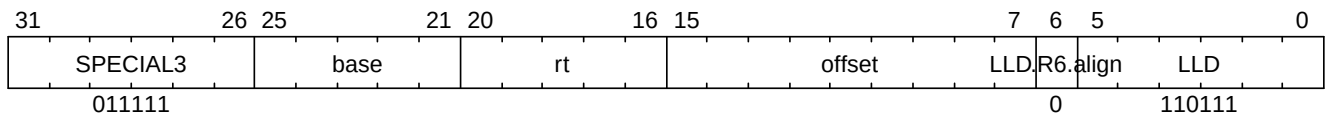


Figure 813. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=9), None)
```

microMIPS

lld.32

Assembly

LLD {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6 && AT >= MIPS64_32

Decode priority

0

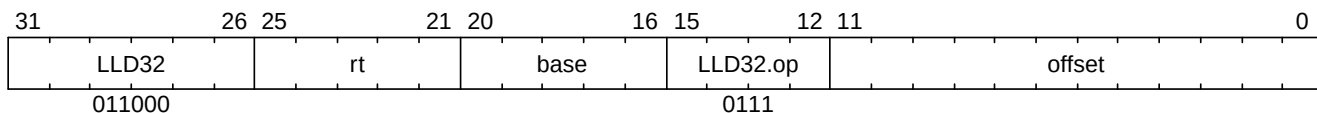


Figure 814. Encoding (32 bits)

Decode

```
del ctx, exec
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=12), None)
```

lld.r6.32

Assembly

LLD {rt:gpr}, {offset:s9}({rs:gpr})

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

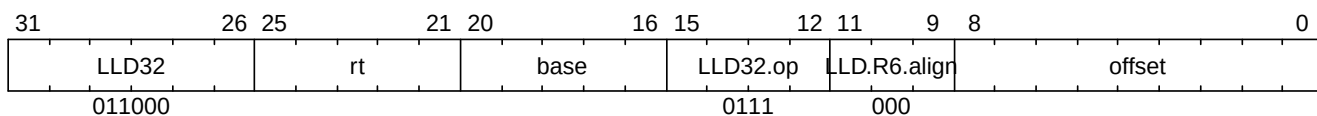


Figure 815. Encoding (32 bits)

Decode

```
del ctx, exec
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=9), None)
```

nanoMIPS

lld.32

Assembly

LLD {rt:gpr}, {offset:s9}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

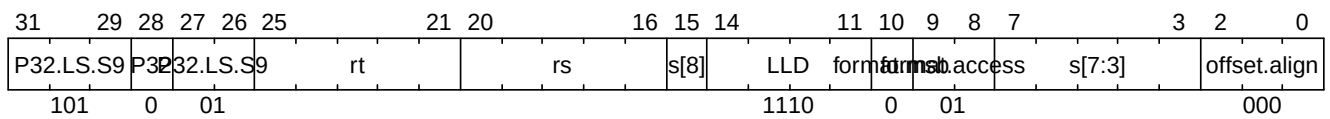


Figure 816. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, rs, sign_extend(s, from_nbits=9), None)
```

8.301.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_atomic_memory_access(ctx, operands, store=False, pair=False)
```

8.301.4. Description

LLD performs an aligned 64-bit linked load and records an eight-byte reservation.

8.301.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.302. LLDP

8.302.1. Purpose

Load a pair of doublewords and establish a linked reservation.

8.302.2. Formats

MIPS

`lldp`

Assembly

`LLDP {ru:gpr}, ({rs:gpr})`

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

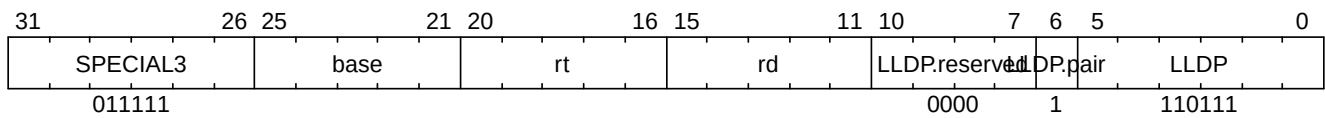


Figure 817. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(0, from_nbits=1), rd)
```

microMIPS

`lldp.32`

Assembly

`LLDP {rt:gpr}, {ru:gpr}, ({rs:gpr})`

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

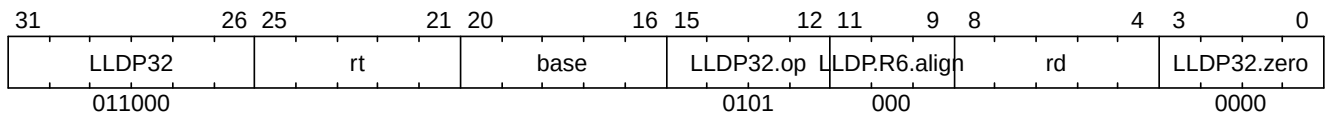


Figure 818. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(0, from_nbits=1), rd)
```

nanoMIPS

lldp.32

Assembly

LLDP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

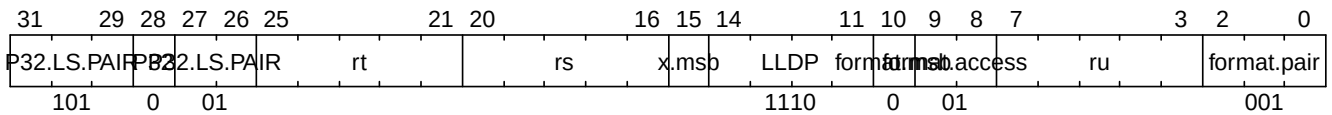


Figure 819. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, rs, sign_extend(0, from_nbits=1), ru)
```

8.302.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_atomic_memory_access(ctx, operands, store=False, pair=True)
```

8.302.4. Description

LLDP performs an aligned 128-bit linked load and records a sixteen-byte reservation.

8.302.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled; paired LL/SC operations are unavailable
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.303. LLE

8.303.1. Purpose

Load-linked word using EVA addressing.

8.303.2. Formats

nanoMIPS

lle.32

Assembly

LLE {rt:gpr}, {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

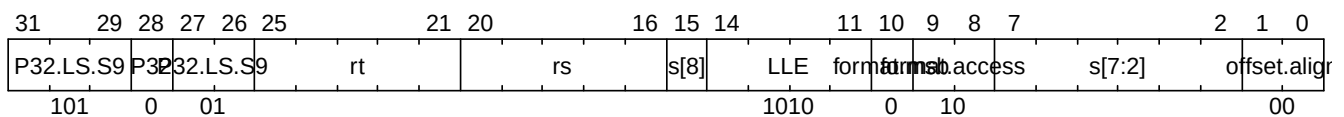


Figure 820. Encoding (32 bits)

Decode

```
return Ll0perands(  
    rt=rt,  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    ru=None,  
)
```

8.303.3. Operation

```
if not ctx.C0.Config5.EVA:  
    raise ReservedInstruction("LLE requires EVA")  
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)  
va = effective_address(ctx, base, operands.offset, "Load", eva=True)  
if va & 3:  
    raise AddressErrorLoad(va, "unaligned linked load")  
record_linked_address(ctx, va, nbytes=4)  
first = read_memory_at_va(ctx, va, nbytes=4, eva=True)  
ctx.GPR[operands.rt] = sign_extend(first, from_nbits=32)
```

8.303.4. Description

LLE performs a privileged EVA linked word load.

8.303.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — LLE requires EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.304. LLWP

8.304.1. Purpose

Load-linked word pair.

8.304.2. Formats

MIPS

llwp

Assembly

LLWP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

AR >= R6

Decode priority

0

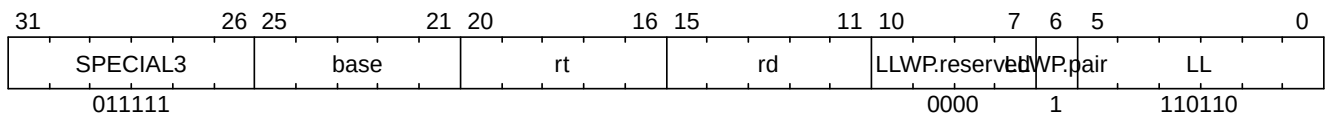


Figure 821. Encoding (32 bits)

Decode

```
return Ll0perands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(0, from_nbits=1),  
    ru=rd,  
)
```

microMIPS

llwp.32

Assembly

LLWP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

MMAR >= R6

Decode priority

0

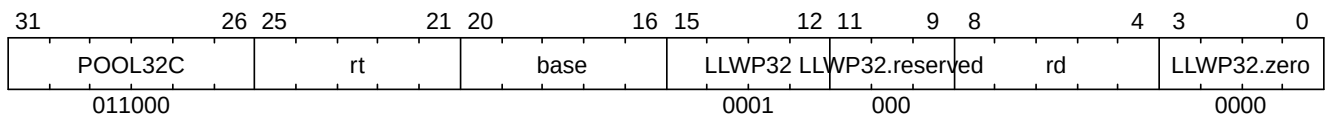


Figure 822. Encoding (32 bits)

Decode

```
return LlOperands(
    rt=rt,
    rs=base,
    offset=sign_extend(0, from_nbits=1),
    ru=rd,
)
```

nanoMIPS

llwp.32

Assembly

LLWP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

!(NMS)

Decode priority

0

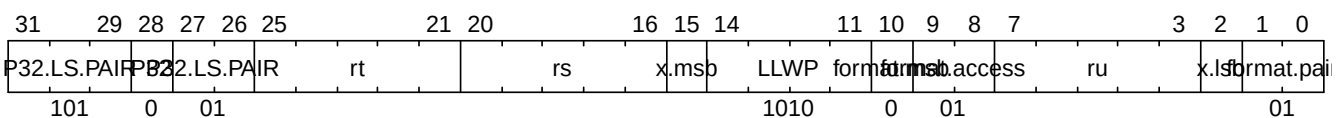


Figure 823. Encoding (32 bits)

Decode

```
return LlOperands(
    rt=rt,
    rs=rs,
    offset=sign_extend(0, from_nbits=1),
    ru=ru,
)
```

8.304.3. Operation

```
if ctx.C0.Config5.XNP:
    raise ReservedInstruction("LLWP requires word-paired support")
if operands.ru is None or operands.rt == operands.ru:
    raise ValueError("LLWP with identical destination registers is UNPREDICTABLE")
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
```

```

if va & 7:
    raise AddressErrorLoad(va, "unaligned linked load pair")
record_linked_address(ctx, va, nbytes=8)
first = read_memory_at_va(ctx, va, nbytes=4)
second = read_memory_at_va(ctx, va + 4, nbytes=4)
ctx.GPR[operands.rt] = sign_extend(first, from_nbits=32)
ctx.GPR[operands.ru] = sign_extend(second, from_nbits=32)

```

8.304.4. Description

LLWP loads two adjacent words and establishes one eight-byte reservation.

8.304.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — LLWP requires word-paired support
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.305. LLWPE

8.305.1. Purpose

Load-linked word pair using EVA addressing.

8.305.2. Formats

nanoMIPS

llwpe.32

Assembly

LLWPE {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

EVA

Decode priority

0

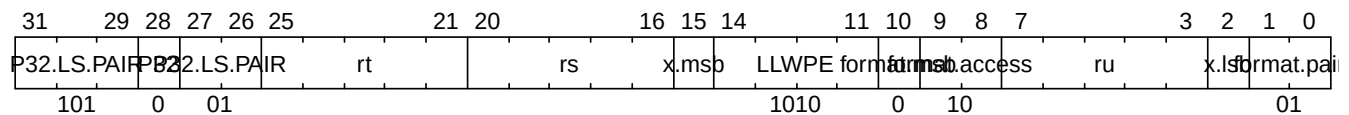


Figure 824. Encoding (32 bits)

Decode

```
return Ll0perands(  
    rt=rt,  
    rs=rs,  
    offset=sign_extend(0, from_nbits=1),  
    ru=ru,  
)
```

8.305.3. Operation

```
if ctx.C0.Config5.XNP:  
    raise ReservedInstruction("LLWPE requires word-paired support")  
if not ctx.C0.Config5.EVA:  
    raise ReservedInstruction("LLWPE requires EVA")  
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
if operands.ru is None or operands.rt == operands.ru:  
    raise ValueError("LLWPE with identical destination registers is UNPREDICTABLE")  
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)  
va = effective_address(ctx, base, operands.offset, "Load", eva=True)  
if va & 7:  
    raise AddressErrorLoad(va, "unaligned linked load pair")
```

```
record_linked_address(ctx, va, nbytes=8)
first = read_memory_at_va(ctx, va, nbytes=4, eva=True)
second = read_memory_at_va(ctx, va + 4, nbytes=4, eva=True)
ctx.GPR[operands.rt] = sign_extend(first, from_nbits=32)
ctx.GPR[operands.ru] = sign_extend(second, from_nbits=32)
```

8.305.4. Description

LLWPE performs a privileged EVA paired linked load.

8.305.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — LLWPE requires word-paired support; LLWPE requires EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.306. LSA

8.306.1. Purpose

Add register `rs` scaled by a left shift to register `rt` and place the 32-bit result in register `rd`.

8.306.2. Formats

MIPS

`lsa`

Assembly

`LSA {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u2}`

Availability

AR >= R6

Decode priority

0

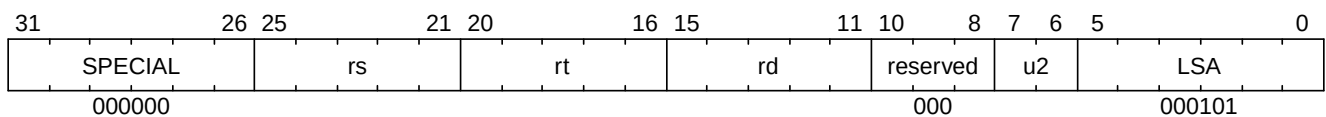


Figure 825. Encoding (32 bits)

Decode

```
shift = zero_extend(u2, from_nbits=2) + 1
return Lsa0operands(rd=rd, rs=rs, rt=rt, shift=shift)
```

microMIPS

`lsa.32.micromips`

Assembly

`LSA {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u2}`

Availability

MMAR >= R6

Decode priority

0

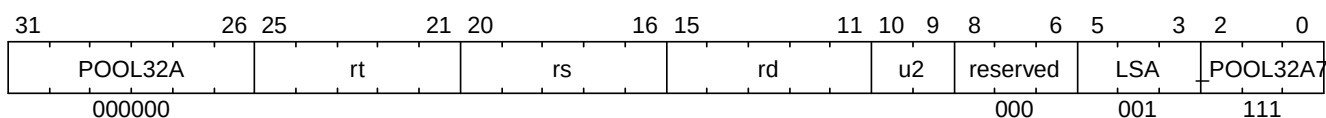


Figure 826. Encoding (32 bits)

Decode

```
shift = zero_extend(u2, from_nbits=2) + 1
return LsaOperands(rd=rd, rs=rs, rt=rt, shift=shift)
```

nanoMIPS

lsa.32

Assembly

LSA {rd:gpr}, {rs:gpr}, {rt:gpr}, {shift:u2}

Availability

true

Decode priority

0

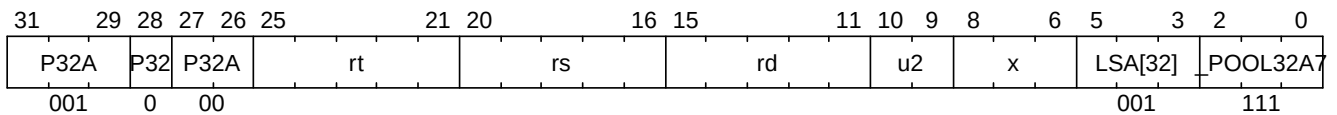


Figure 827. Encoding (32 bits)

Decode

```
del ctx, exec
shift = zero_extend(u2, from_nbits=2)
return LsaOperands(rd=rd, rs=rs, rt=rt, shift=shift)
```

8.306.3. Operation

```
source = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
base = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
sum = (int(source) << int(operands.shift)) + int(base)
ctx.GPR[operands.rd] = sign_extend(sum, from_nbits=32)
```

8.306.4. Description

LSA shifts the 32-bit value in *rs* by the encoded *u2* amount, adds it to *rt*, and sign-extends the 32-bit result to the active GPR width. In nanoMIPS, *u2* directly encodes shifts 0 through 3.

8.306.5. Exceptions

None.

8.307. LUI

8.307.1. Purpose

Load an upper immediate value into a GPR, with the lower twelve bits set to zero.

8.307.2. Formats

MIPS

lui

Assembly

LUI {rt:gpr}, {imm:u16}

Availability

AR < R6

Decode priority

0

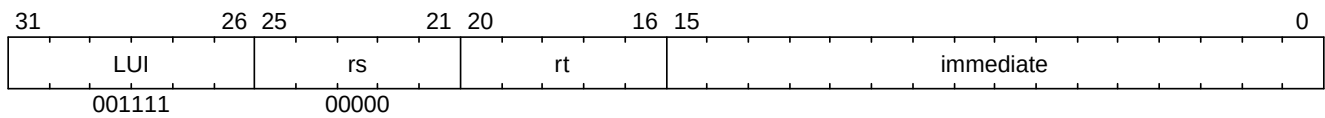


Figure 828. Encoding (32 bits)

Decode

```
del ctx, exec
value = zero_extend(immediate, from_nbits=16) << 16
return LuiOperands(rt=rt, imm=sign_extend(value, from_nbits=32))
```

microMIPS

lui.32

Assembly

LUI {rt:gpr}, {imm:u16}

Availability

MMAR < R6

Decode priority

0

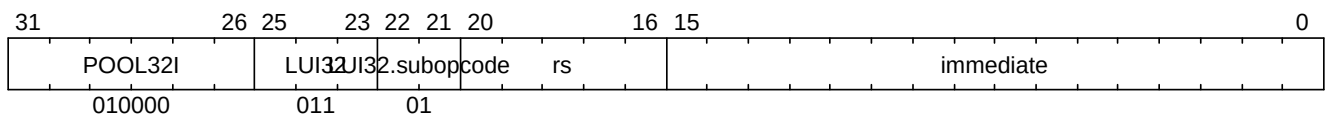


Figure 829. Encoding (32 bits)

Decode

```
value = zero_extend(immediate, from_nbits=16) << 16
return LuiOperands(rt=rs, imm=sign_extend(value, from_nbits=32))
```

nanoMIPS

lui.32

Assembly

LUI {rt:gpr}, {imm:s32}

Availability

true

Decode priority

0

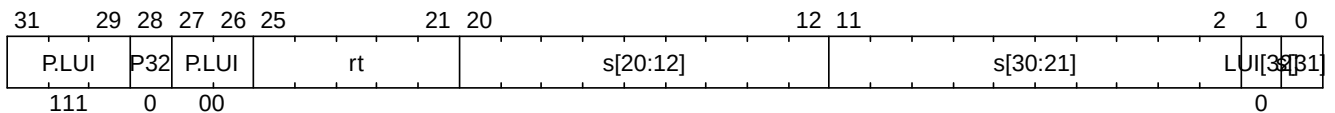


Figure 830. Encoding (32 bits)

Decode

```
imm = sign_extend(s, from_nbits=32)
return LuiOperands(rt=rt, imm=imm)
```

8.307.3. Operation

```
ctx.GPR[operands.rt] = operands.imm
```

8.307.4. Description

LUI[32] reconstructs a 32-bit value from the split signed s field. The resulting word is sign-extended when written to a wider GPR.

8.307.5. Exceptions

None.

8.308. LUXC1

8.308.1. Purpose

Indexed LUXC1 access through Coprocessor 1.

8.308.2. Formats

MIPS

luxc1

Assembly

LUXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

AR >= R2_3_5 && AR < R6

Decode priority

0

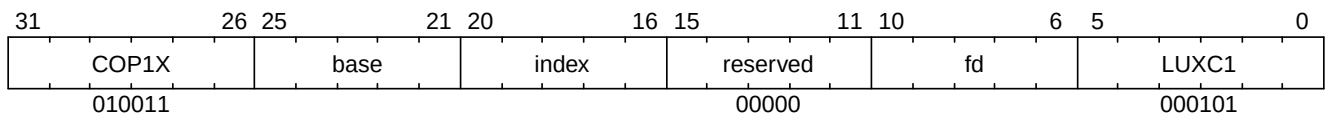


Figure 831. Encoding (32 bits)

Decode

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

microMIPS

luxc1.32

Assembly

LUXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

MMAR < R6

Decode priority

0

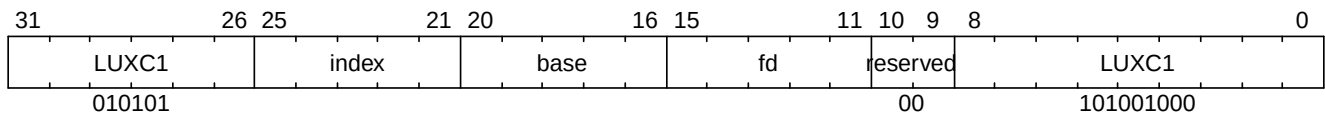


Figure 832. Encoding (32 bits)

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

8.308.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
index = sign_extend(ctx.GPR[operands.index], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, index, "Load")
# LUXC1 discards the low address bits, as specified by the AVP model's
# ``ldxc1(..., -align)`` pseudocode. It is not a byte-concatenating
# unaligned load: the effective address is rounded down to a dword.
va &= -8
set_fpr(ctx, operands.fd, read_memory_at_va(ctx, va, nbytes=8), "64")
```

8.308.4. Description

LUXC1 performs its CP1 access checks before transferring data.

8.308.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.; The CP1 load address is not valid.
- **ADES** (Address Error) — The CP1 store address is not valid.
- **CPU** (Coprorocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.; The CP1 load address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.309. LW

8.309.1. Purpose

Load a 32-bit word from memory into a GPR.

8.309.2. Formats

MIPS

`lw`

Assembly

`LW {rt:gpr}, {offset:s16}({rs:gpr})`

Availability

true

Decode priority

0

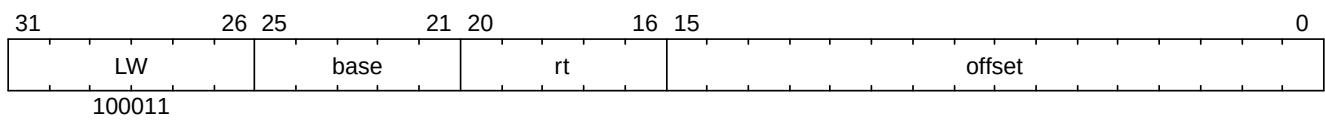


Figure 833. Encoding (32 bits)

Decode

```
return LwOperands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
)
```

microMIPS

`lw.sp.16`

Assembly

`LW {rt:gpr}, {offset:u7}({rs:gpr})`

Availability

true

Decode priority

0

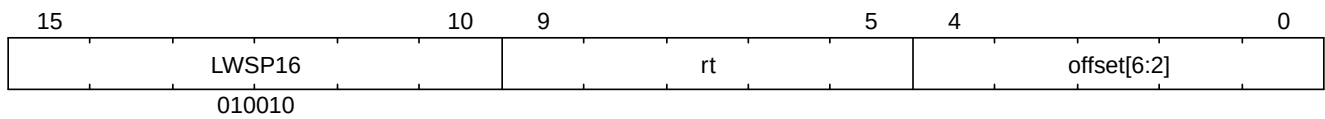


Figure 834. Encoding (16 bits)

Decode

```
return LwOperands(
    rt=rt,
    rs=29,
    offset=zero_extend(offset, from_nbits=7),
)
```

lw.16

Assembly

LW {rt:gpr3}, {offset:u6}({rs:gpr3})

Availability

true

Decode priority

0

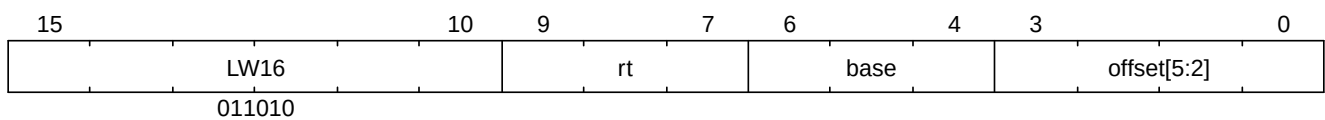


Figure 835. Encoding (16 bits)

Decode

```
return LwOperands(
    rt=MICROMIPS_GPR3.decode(rt),
    rs=MICROMIPS_GPR3.decode(base),
    offset=zero_extend(offset, from_nbits=6),
)
```

lw.32

Assembly

LW {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

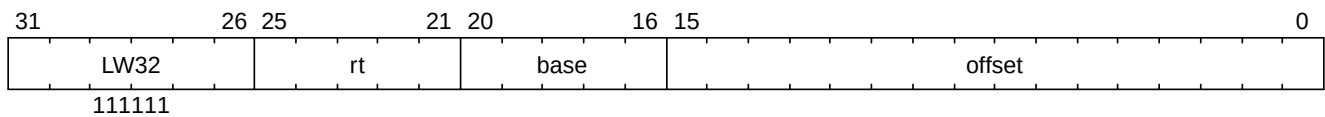


Figure 836. Encoding (32 bits)

Decode

```
return LwOperands(
    rt=rt,
    rs=base,
    offset=sign_extend(offset, from_nbits=16),
)
```

lw.gp.16

Assembly

LW {rt:gpr3}, {offset:u9}(gp)

Availability

true

Decode priority

0

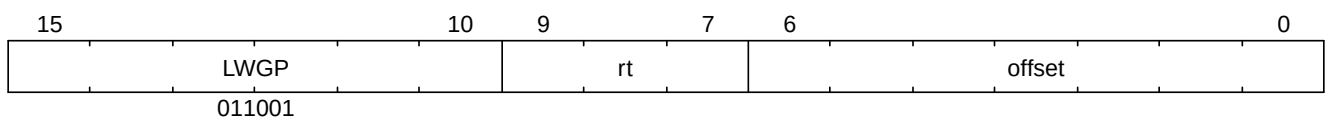


Figure 837. Encoding (16 bits)

Decode

```
return LwOperands(
    rt=MICROMIPS_GPR3.decode(rt),
    rs=28,
    offset=sign_extend(offset, from_nbits=7) << 2,
)
```

nanoMIPS

lw.16

Assembly

LW {rt:gpr3}, {offset:u6}({rs:gpr3})

Availability

true

Decode priority

0

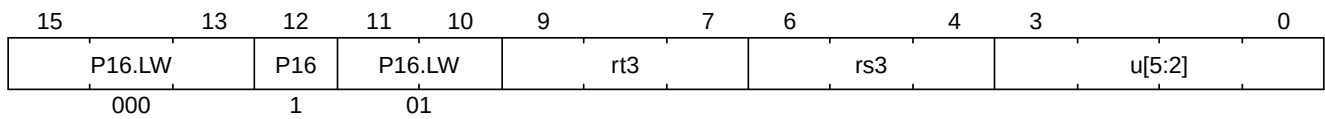


Figure 838. Encoding (16 bits)

Decode

```
if P64_Mode(ctx, "Load"):
    raise ReservedInstruction("LW[16] is unavailable in P64 mode")
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
offset = zero_extend(u, from_nbits=6)
return LwOperands(rt=rt, rs=rs, offset=offset)
```

Lw.4x4

Assembly

LW {rt:gpr4}, {offset:u4}({rs:gpr4})

Availability

!(NMS)

Decode priority

0

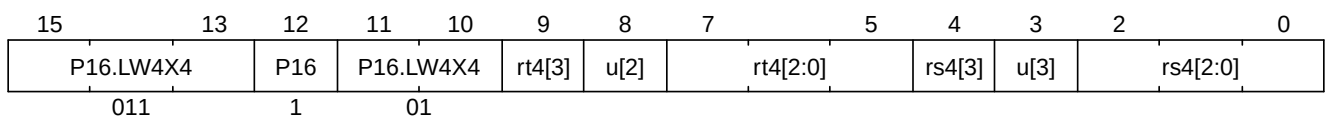


Figure 839. Encoding (16 bits)

Decode

```
if P64_Mode(ctx, "Load"):
    raise ReservedInstruction("LW[4X4] is unavailable in P64 mode")
rt = GPR4.decode(rt4)
rs = GPR4.decode(rs4)
offset = zero_extend(u, from_nbits=4)
return LwOperands(rt=rt, rs=rs, offset=offset)
```

Lw.gp16

Assembly

LW {rt:gpr3}, {offset:u9}(gp)

Availability

true

Decode priority

0

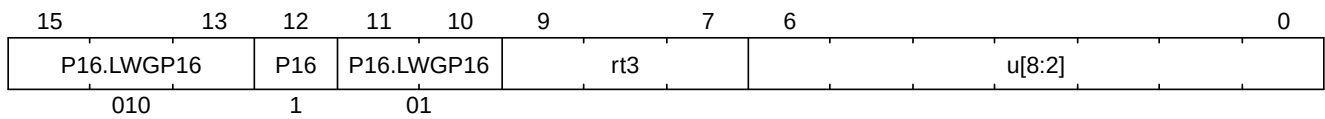


Figure 840. Encoding (16 bits)

Decode

```

if P64_Mode(ctx, "Load"):
    raise ReservedInstruction("LW[GP16] is unavailable in P64 mode")
rt = GPR3.decode(rt3)
offset = zero_extend(u, from_nbits=9)
return LwOperands(rt=rt, rs=28, offset=offset)

```

lw.gp

Assembly

LW {rt:gpr}, {offset:u21}(gp)

Availability

true

Decode priority

0

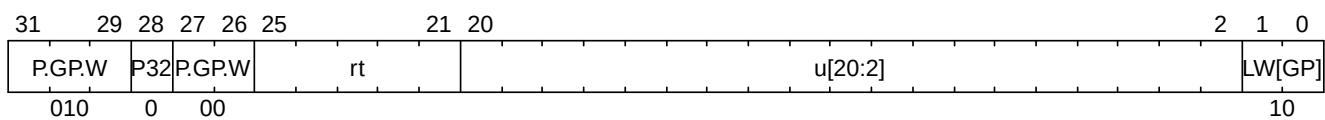


Figure 841. Encoding (32 bits)

Decode

```

offset = zero_extend(u, from_nbits=21)
return LwOperands(rt=rt, rs=28, offset=offset)

```

lw.u12

Assembly

LW {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

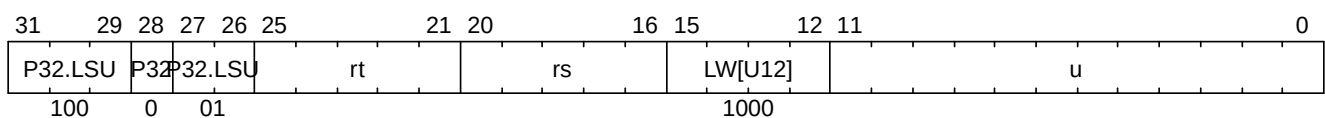


Figure 842. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return LwOperands(rt=rt, rs=rs, offset=offset)
```

lw.s9

Assembly

LW {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

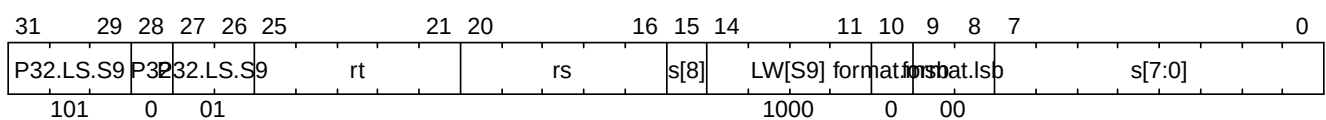


Figure 843. Encoding (32 bits)

Decode

```
if P64_Mode(ctx, "Load"):
    raise ReservedInstruction("LW[S9] is unavailable in P64 mode")
offset = sign_extend(s, from_nbits=9)
return LwOperands(rt=rt, rs=rs, offset=offset)
```

lw.sp

Assembly

LW {rt:gpr}, {offset:u7}(sp)

Availability

true

Decode priority

0

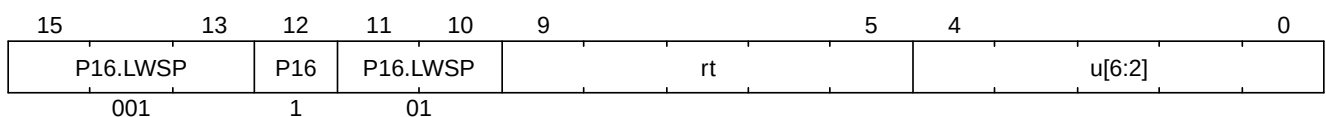


Figure 844. Encoding (16 bits)

Decode

```
offset = zero_extend(u, from_nbits=7)
return LwOperands(rt=rt, rs=29, offset=offset)
```

8.309.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
data = read_memory_at_va(ctx, va, nbytes=4)
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=32)
```

8.309.4. Description

Each format adds its decoded byte offset to the explicit or implicit base GPR, reads four bytes using the active memory backend, and sign-extends the word to the implemented GPR width.

8.309.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — LW[16] is unavailable in P64 mode; LW[4X4] is unavailable in P64 mode; LW[GP16] is unavailable in P64 mode; LW[S9] is unavailable in P64 mode; LW[4X4] is reserved when CP0 Config5.NMS is set.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.310. LWC1

8.310.1. Purpose

LWC1 a CP1 register to or from memory.

8.310.2. Formats

MIPS

`lwc1`

Assembly

`LWC1 {ft:fpr}, {offset:s16}({base:gpr})`

Availability

true

Decode priority

0

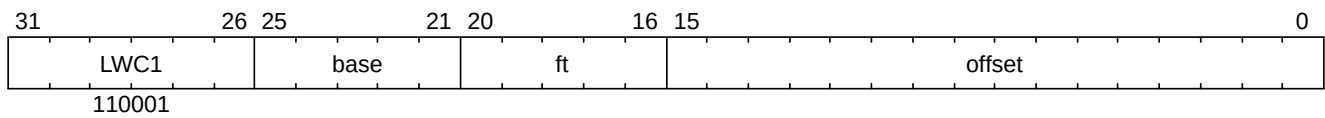


Figure 845. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(  
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)  
)
```

microMIPS

`lwc1.32`

Assembly

`LWC1 {ft:fpr}, {offset:s16}({base:gpr})`

Availability

true

Decode priority

0

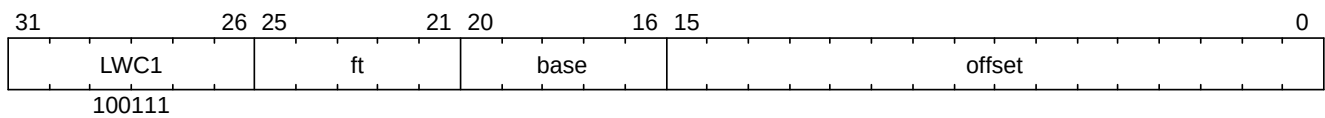


Figure 846. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)
)
```

nanoMIPS

lwc1.gp

Assembly

LWC1 {ft:fpr}, ...

Availability

true

Decode priority

0

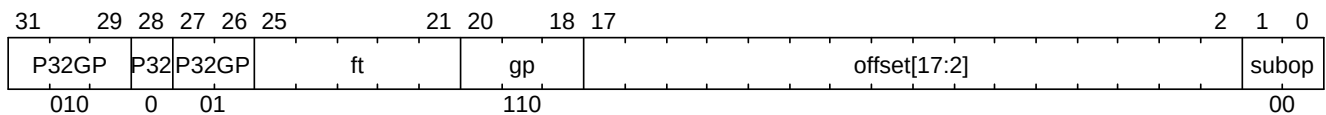


Figure 847. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(
    ft=ft, base=28, offset=zero_extend(offset, from_nbits=16) << 2
)
```

lwc1.s9

Assembly

LWC1 {ft:fpr}, ...

Availability

true

Decode priority

0

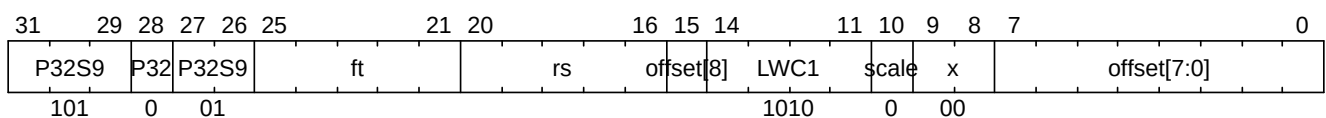


Figure 848. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=sign_extend(offset, from_nbits=9))
```

lwc1.u12

Assembly

LWC1 {ft:fpr}, ...

Availability

true

Decode priority

0

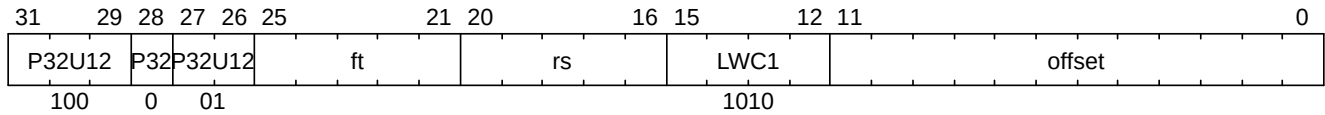


Figure 849. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=zero_extend(offset, from_nbits=12))
```

8.310.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
set_fpr(ctx, operands.ft, read_memory_at_va(ctx, va, nbytes=4), "32")
```

8.310.4. Description

LWC1 performs CP1 access checks before its memory access.

8.310.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.; The CP1 load address is not valid.
- **ADES** (Address Error) — The CP1 store address is not valid.
- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the required FPU resource is unavailable.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.; The CP1

load address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.311. LWC2

8.311.1. Purpose

Load a word from memory into a CP2 register.

8.311.2. Formats

microMIPS

lwc2.32

Assembly

LWC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR < R6

Decode priority

0

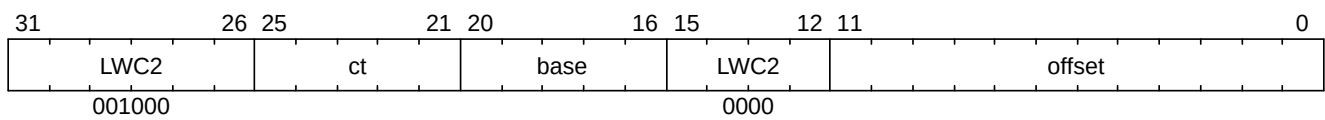


Figure 850. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

lwc2.r6.32

Assembly

LWC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR >= R6

Decode priority

0

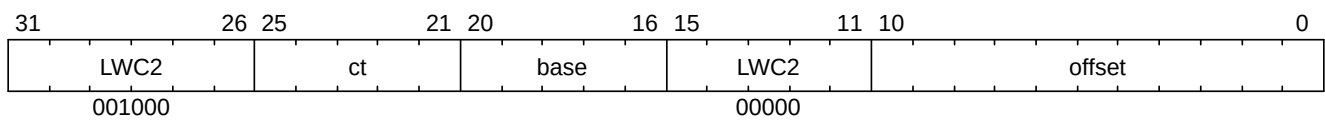


Figure 851. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

8.311.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("LWC2 execution is not implemented")
```

8.311.4. Description

LWC2 performs the CP2 access check before any CP2 operation is attempted.

8.311.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; LWC2 execution is not implemented

8.312. LWE

8.312.1. Purpose

Load a word using EVA addressing.

8.312.2. Formats

nanoMIPS

lwe.32

Assembly

LWE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

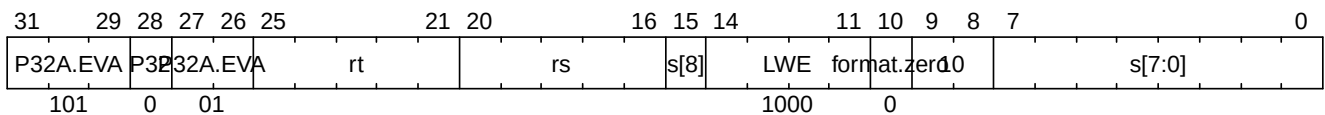


Figure 852. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.312.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load", eva=True)
data = read_memory_at_va(ctx, va, nbytes=4, eva=True)
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=32)
```

8.312.4. Description

This optional privileged form uses EVA user-mode address translation.

8.312.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.313. LWL

8.313.1. Purpose

Load the left portion of an unaligned word into a GPR.

8.313.2. Formats

MIPS

lw1

Assembly

LWL {offset:s16}({rs:gpr})

Availability

AR < R6

Decode priority

0

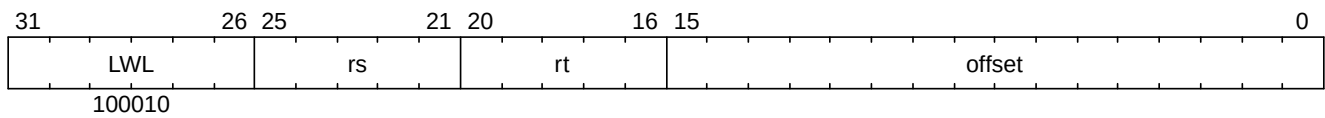


Figure 853. Encoding (32 bits)

Decode

```
return LegacyUnalignedOperands(  
    rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=16)  
)
```

microMIPS

lw1.32

Assembly

LWL {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6

Decode priority

0

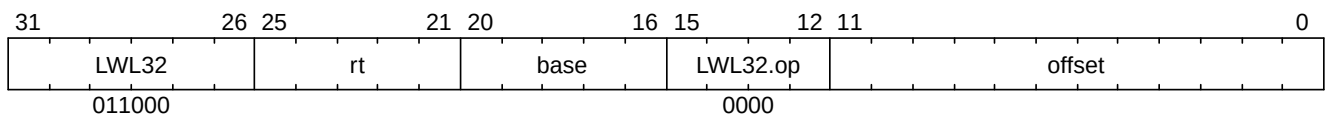


Figure 854. Encoding (32 bits)

```

del ctx, exec
return LegacyUnalignedOperands(
    rt=rt, rs=base, offset=sign_extend(offset, from_nbits=12)
)

```

8.313.3. Operation

```

base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
word = read_memory_at_va(
    ctx,
    va,
    nbytes=4,
    unaligned_ok=True,
    partial_ok=True,
)
byte = va & 3
order = memory_endianness(ctx)
shift = (3 - byte) * 8 if order == "little" else byte * 8
preserve = (1 << shift) - 1
merged = ((word << shift) & 0xFFFF_FFFF) | (ctx.GPR[operands.rt] & preserve)
ctx.GPR[operands.rt] = sign_extend(merged, from_nbits=32)

```

8.313.4. Description

The naturally aligned word containing the effective byte address is merged according to the active memory endianness. This legacy instruction is unavailable in Release 6.

8.313.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.314. LWM

8.314.1. Purpose

Load multiple words into sequential GPRs.

8.314.2. Formats

microMIPS

`lwm.16`

Assembly

`LWM {offset:u4}(sp)`

Availability

MMAR < R6

Decode priority

0

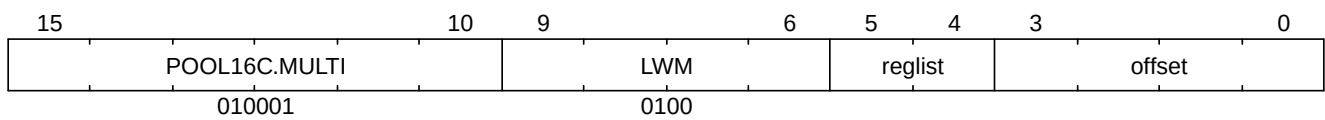


Figure 855. Encoding (16 bits)

Decode

```
return MicroMultipleOperands(_MICRO_LWM16[reglist], 29, zero_extend(offset, 4) << 2)
```

`lwm.r6.16`

Assembly

`LWM {offset:u4}(sp)`

Availability

MMAR >= R6

Decode priority

0

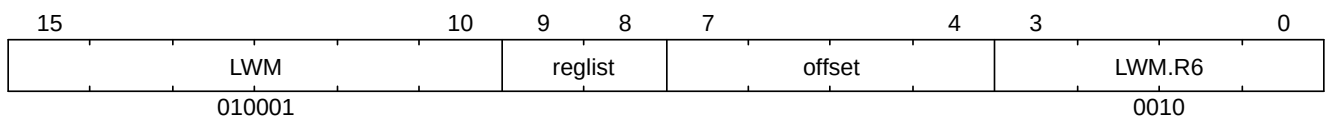


Figure 856. Encoding (16 bits)

Decode

```
return MicroMultipleOperands(_MICRO_LWM16[reglist], 29, zero_extend(offset, 4) << 2)
```


Lwm.32

Assembly

LWM {offset:s12}({rs:gpr})

Availability

true

Decode priority

0

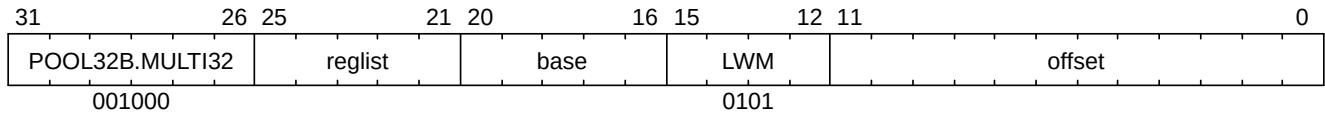


Figure 857. Encoding (32 bits)

Decode

```
if reglist >= len(_MICRO_LWM32) or _MICRO_LWM32[reglist] is None:
    raise ReservedInstruction("reserved microMIPS multiple register list")
return MicroMultipleOperands(_MICRO_LWM32[reglist], base, sign_extend(offset, 12))
```

nanoMIPS

Lwm.32

Assembly

LWM {offset:s9}({rs:gpr}), {count:u3}

Availability

!(NMS)

Decode priority

0

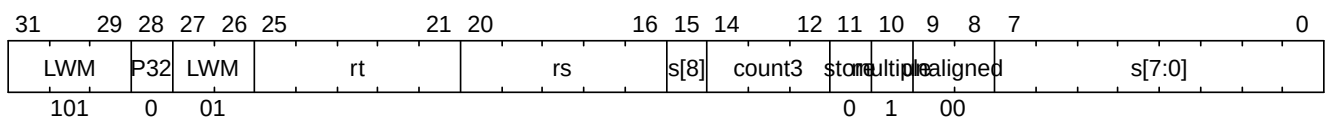


Figure 858. Encoding (32 bits)

Decode

```
return MultipleMemoryOperands(
    rt, rs, sign_extend(s, from_nbits=9), 8 if count3 == 0 else count3
)
```

8.314.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
registers = load_multiple_registers(operands)
```

```

for index in multiple_transfer_order(operands):
    register = registers[index]
    va = effective_address(ctx, base, int(operands.offset) + index * 4, "Load")
    data = read_memory_at_va(ctx, va, nbytes=4)
    ctx.GPR[register] = sign_extend(data, from_nbits=32)
    if register == operands.rs and index != len(registers) - 1:
        raise ValueError("multiple load updating its base early is UNPREDICTABLE")

```

8.314.4. Description

This full-nano form is unavailable in the nanoMIPS subset.

8.314.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — reserved microMIPS multiple register list
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.315. LWP

8.315.1. Purpose

LWP microMIPS register pair memory operation.

8.315.2. Formats

microMIPS

`lwp.32`

Assembly

`LWP {rd:gpr}, {offset:s12}({rs:gpr})`

Availability

true

Decode priority

0

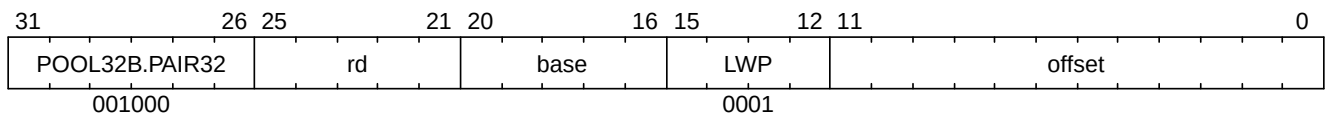


Figure 859. Encoding (32 bits)

Decode

```
return MicroMultipleOperands((rd, rd + 1), base, sign_extend(offset, 12))
```

8.315.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
for index, register in enumerate(operands regs):
    va = effective_address(ctx, base, operands.offset + index * 4, "Load")
    ctx.GPR[register] = sign_extend(
        read_memory_at_va(ctx, va, nbytes=4), from_nbits=32
    )
```

8.315.4. Description

microMIPS register-pair memory transfer.

8.315.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.

- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.316. LWPC

8.316.1. Purpose

Load a word from a PC-relative address.

8.316.2. Formats

MIPS

`lwpc`

Assembly

`LWPC {offset:s21}`

Availability

AR >= R6

Decode priority

0

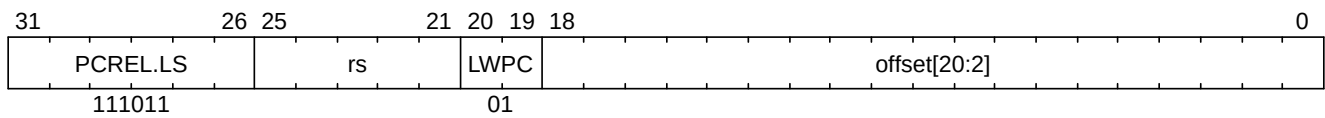


Figure 860. Encoding (32 bits)

Decode

```
return PcRelativeMemoryOperands(rs, sign_extend(offset, from_nbits=21))
```

microMIPS

`lwpc.32.micromips`

Assembly

`LWPC {rt:gpr}, {offset:s21}`

Availability

MMAR >= R6

Decode priority

0

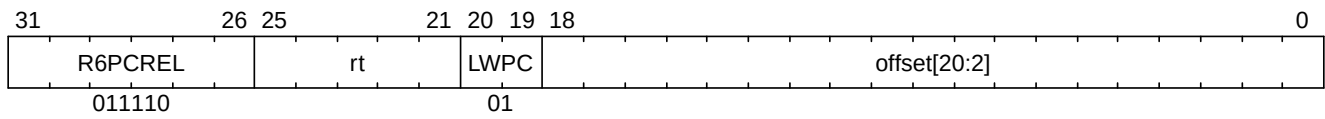


Figure 861. Encoding (32 bits)

Decode

```
del ctx, exec
return PcRelativeMemoryOperands(rt, sign_extend(offset, from_nbits=21))
```

nanoMIPS

lwpc.48

Assembly

LWPC {rt:gpr}, {offset:s32}

Availability

!(NMS)

Decode priority

0

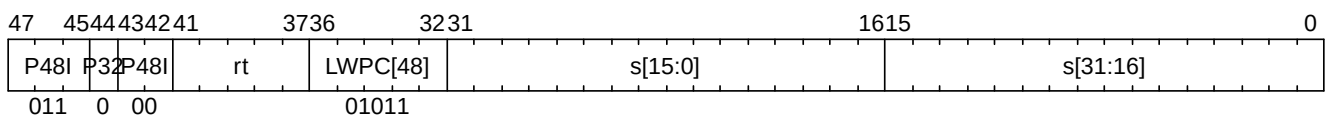


Figure 862. Encoding (48 bits)

Decode

```
return PcRelativeMemoryOperands(rt, sign_extend(s, from_nbits=32))
```

8.316.3. Operation

```
if ctx.profile.architecture is Architecture.NANOMIPS:
    base = (
        ctx.next_pc
        if P64_Mode(ctx, "Load")
        else sign_extend(ctx.next_pc, from_nbits=32)
    )
    address = effective_address(ctx, base, operands.offset, "Load")
elif ctx.ISAMode == ISAMode.MIPS:
    base = sign_extend(ctx.pc & ~0b11, from_nbits=32)
    address = effective_address(ctx, base, operands.offset, "Load")
elif ctx.ISAMode == ISAMode.MICROMIPS:
    # microMIPS32 PC-relative loads use the word-aligned address of the
    # current instruction. The low bit of the halfword address is an ISA
    # mode tag, so using next_pc here would shift every load by one parcel.
    base = sign_extend(ctx.pc & ~0b11, from_nbits=32)
    address = effective_address(ctx, base, operands.offset, "Load")
else:
    raise ReservedInstruction("LWPC is unavailable in the active ISA mode")
data = read_memory_at_va(ctx, address, nbytes=4)
```

```
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=32)
```

8.316.4. Description

The signed 19-bit displacement is shifted left two bits and added to the aligned address of the LWPC instruction.

8.316.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — LWPC is unavailable in the active ISA mode
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.317. LWR

8.317.1. Purpose

Load the right part of an unaligned word into a GPR.

8.317.2. Formats

MIPS

`lwr`

Assembly

`LWR {rt:gpr}, {offset:s16}({rs:gpr})`

Availability

AR < R6

Decode priority

0

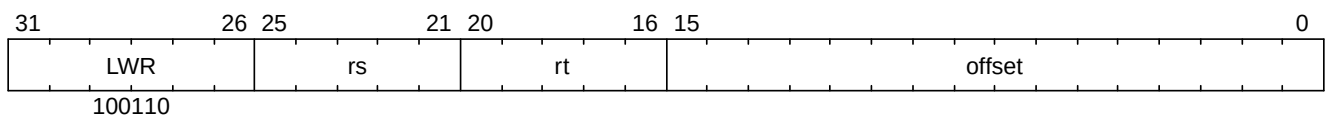


Figure 863. Encoding (32 bits)

Decode

```
del ctx, exec
return LwrOperands(
    rt=rt,
    rs=rs,
    offset=sign_extend(offset, from_nbits=16),
)
```

microMIPS

`lwr.32`

Assembly

`LWR {rt:gpr}, {offset:s12}({rs:gpr})`

Availability

MMAR < R6

Decode priority

0

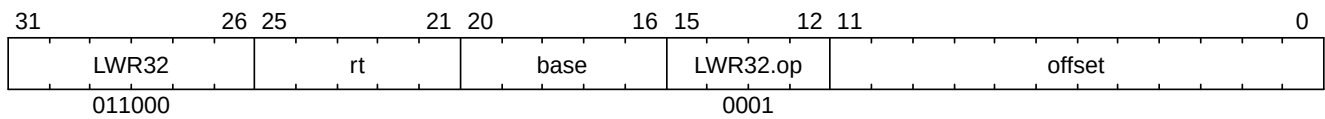


Figure 864. Encoding (32 bits)

Decode

```
del ctx, exec
return LwrOperands(
    rt=rt,
    rs=base,
    offset=sign_extend(offset, from_nbits=12),
)
```

8.317.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
word = read_memory_at_va(
    ctx,
    va,
    nbytes=4,
    unaligned_ok=True,
    partial_ok=True,
)

# LWR is a 32-bit merge operation. The byte selected by the unaligned
# address is interpreted in memory order; the remaining low-order bytes
# come from the aligned word.
byte = va & 0x3
order = memory_endianness(ctx)
shift = byte * 8 if order == "little" else (3 - byte) * 8
loaded_bits = 32 - shift
loaded_mask = (1 << loaded_bits) - 1
merged = (ctx.GPR[operands.rt] & ~loaded_mask) | ((word >> shift) & loaded_mask)
ctx.GPR[operands.rt] = sign_extend(merged, from_nbits=32)
```

8.317.4. Description

LWR reads the naturally aligned word containing the effective byte address and merges the bytes on the right side of that address with the existing low 32-bit value of `rt`. The merge direction follows memory endianness. The resulting 32-bit word is sign-extended to the implemented GPR width. LWR is a legacy MIPS32 instruction and is not available in Release 6 or nanoMIPS encodings.

8.317.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.

- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.318. LWU

8.318.1. Purpose

Load an unsigned 32-bit word into a 64-bit GPR.

8.318.2. Formats

MIPS

`lwu`

Assembly

`LWU {offset:s16}({rs:gpr})`

Availability

AT >= MIPS64_32

Decode priority

0

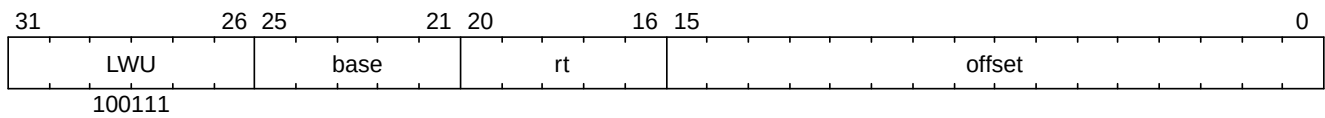


Figure 865. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, base, sign_extend(offset, from_nbits=16))
```

microMIPS

`lwu`

Assembly

`LWU {rt:gpr}, {offset:s12}({rs:gpr})`

Availability

AT >= MIPS64_32

Decode priority

0

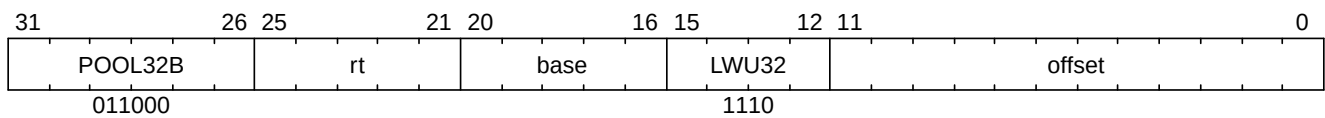


Figure 866. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, base, sign_extend(offset, from_nbits=12))
```

nanoMIPS

lwu . gp

Assembly

```
LWU {rt:gpr}, {offset:u21}(gp)
```

Availability

AT >= MIPS64_32

Decode priority

0

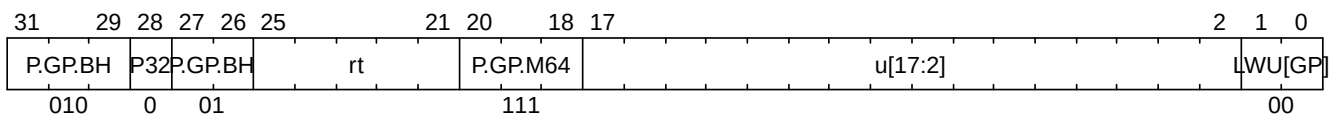


Figure 867. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, 28, zero_extend(u, from_nbits=18))
```

lwu.s9

Assembly

```
LWU {rt:gpr}, {offset:s9}({rs:gpr})
```

Availability

AT >= MIPS64_32

Decode priority

0

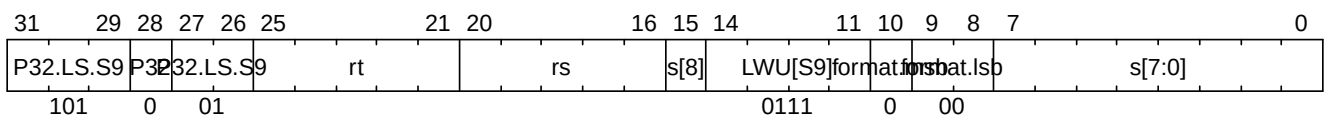


Figure 868. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

lwu.u12

Assembly

LWU {rt:gpr}, {offset:u12}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

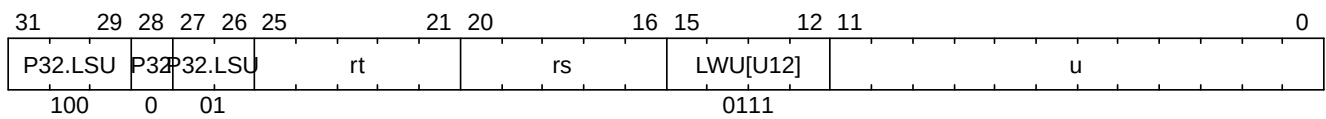


Figure 869. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, rs, zero_extend(u, from_nbits=12))
```

8.318.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_memory_access(
    ctx, operands, nbytes=4, signed_load=False, store=False
)
```

8.318.4. Description

LWU forms a full-width effective address, reads four bytes, and zero-extends the result.

8.318.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.319. LWUPC

8.319.1. Purpose

Load an unsigned word from a PC-relative address.

8.319.2. Formats

MIPS

`lwupc`

Assembly

`LWUPC {offset:s21}`

Availability

AR >= R6 && AT >= MIPS64_32

Decode priority

0

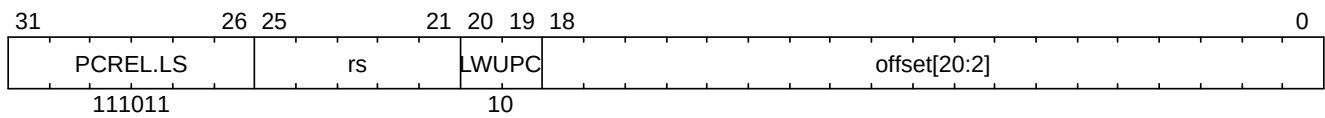


Figure 870. Encoding (32 bits)

Decode

```
return PcRelativeMemoryOperands(rs, sign_extend(offset, from_nbits=21))
```

microMIPS

`lwupc.32.micromips`

Assembly

`LWUPC {rt:gpr}, {offset:s21}`

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

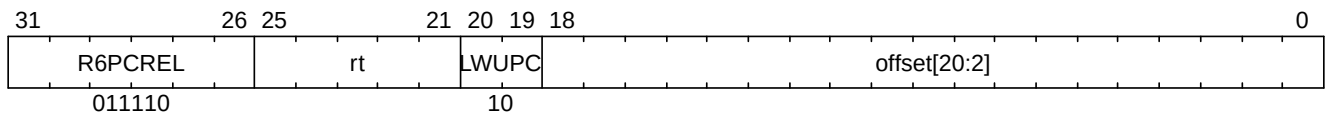


Figure 871. Encoding (32 bits)

```
del ctx, exec
return PcRelativeMemoryOperands(rt, sign_extend(offset, from_nbits=21))
```

8.319.3. Operation

```
base = sign_extend(ctx.pc & ~0b11, from_nbits=32)
address = effective_address(ctx, base, operands.offset, "Load")
data = read_memory_at_va(ctx, address, nbytes=4)
ctx.GPR[operands.rt] = zero_extend(data, from_nbits=32)
```

8.319.4. Description

The signed 19-bit displacement is shifted left two bits and added to the aligned address of the LWUPC instruction.

8.319.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.320. LWUX

8.320.1. Purpose

Load an unsigned word using a register-indexed address.

8.320.2. Formats

nanoMIPS

lwux

Assembly

LWUX {rs:gpr}({rt:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

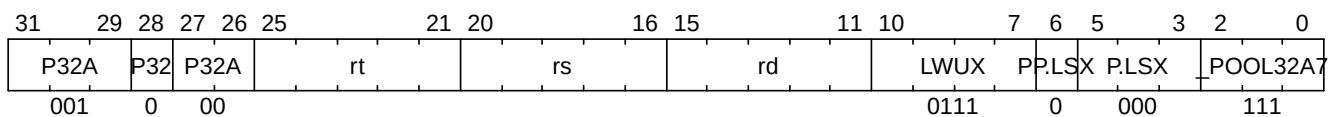


Figure 872. Encoding (32 bits)

Decode

```
return IndexedDoubleMemoryOperands(rd, rs, rt)
```

8.320.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_indexed_double_memory_access(
    ctx, operands, nbytes=4, signed_load=False, scale=0, store=False
)
```

8.320.4. Description

LWUX adds two full-width GPR values and zero-extends the loaded word.

8.320.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.321. LWUXS

8.321.1. Purpose

Load an unsigned word using a scaled register-indexed address.

8.321.2. Formats

nanoMIPS

lwuxs

Assembly

LWUXS {rs:gpr}({rt:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

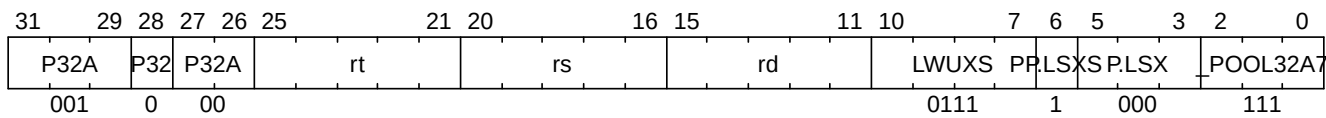


Figure 873. Encoding (32 bits)

Decode

```
return IndexedDoubleMemoryOperands(rd, rs, rt)
```

8.321.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_indexed_double_memory_access(
    ctx, operands, nbytes=4, signed_load=False, scale=2, store=False
)
```

8.321.4. Description

LWUXS adds the base GPR to four times the index GPR and zero-extends the loaded word.

8.321.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled

- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.322. LWX

8.322.1. Purpose

Load a word using a register-indexed address.

8.322.2. Formats

MIPS

`lwx.dsp`

Assembly

`LWX {rs:gpr}({rt:gpr})`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

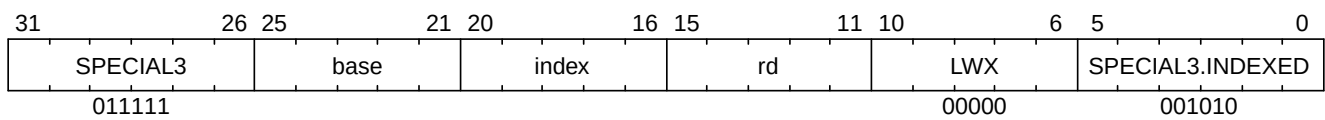


Figure 874. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd=rd, rs=index, rt=base)
```

microMIPS

`lwx.dsp.32`

Assembly

`LWX {rs:gpr}({rt:gpr})`

Availability

DSP

Decode priority

0

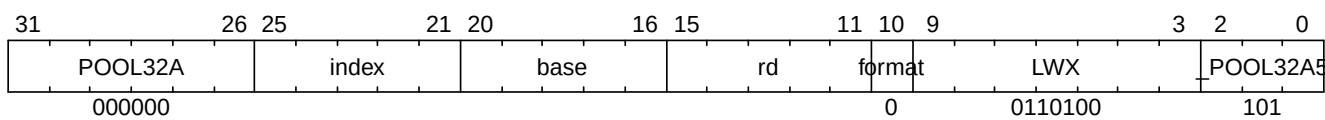


Figure 875. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd=rd, rs=index, rt=base)
```

nanoMIPS

lwx.32

Assembly

LWX {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

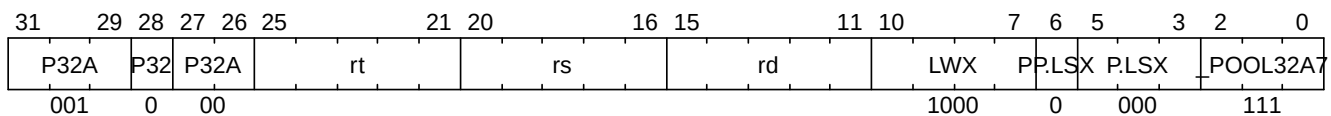


Figure 876. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.322.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Load")
data = read_memory_at_va(ctx, va, nbytes=4)
ctx.GPR[operands.rd] = sign_extend(data, from_nbits=32)
```

8.322.4. Description

The effective address is formed from two full-width GPR values.

8.322.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **DSPDIS** (DSP Module State Disabled) — DSP Module resource access is disabled by Status.MX.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.322.6. Programming Notes

The MIPS64 DSP Module Revision 3.02 operation pseudocode describes `vAddr[31:0]` as the sum of `GPR[index][31:0]` and `GPR[base][31:0]`. That truncation is a documentation error. In a 64-bit GPR mode, the effective virtual address is formed by adding the full 64-bit GPR values.

8.323. LWXC1

8.323.1. Purpose

Indexed LWXC1 access through Coprocessor 1.

8.323.2. Formats

MIPS

`lwxc1`

Assembly

`LWXC1 {fd:fpr}, {index:gpr}({base:gpr})`

Availability

`AR >= R2_3_5 && AR < R6`

Decode priority

`0`

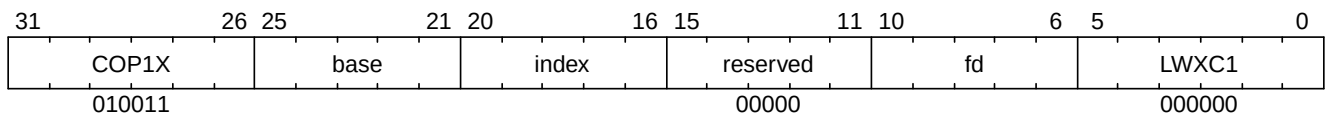


Figure 877. Encoding (32 bits)

Decode

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

microMIPS

`lwxc1.32`

Assembly

`LWXC1 {fd:fpr}, {index:gpr}({base:gpr})`

Availability

`MMAR < R6`

Decode priority

`0`

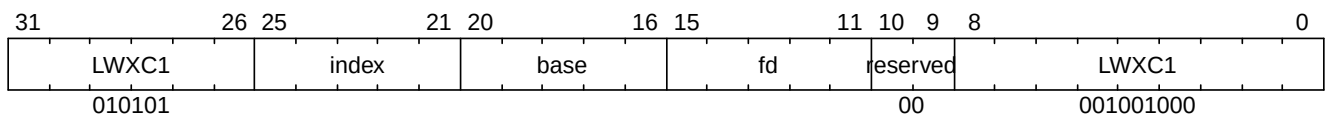


Figure 878. Encoding (32 bits)

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

8.323.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
index = sign_extend(ctx.GPR[operands.index], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, index, "Load")
set_fpr(ctx, operands.fd, read_memory_at_va(ctx, va, nbytes=4), "32")
```

8.323.4. Description

LWXC1 performs its CP1 access checks before transferring data.

8.323.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.; The CP1 load address is not valid.
- **ADES** (Address Error) — The CP1 store address is not valid.
- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.; The CP1 load address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.324. LWXS

8.324.1. Purpose

Load a word using a scaled register-indexed address.

8.324.2. Formats

microMIPS

`lwx.s.32`

Assembly

`LWXS {rd:gpr}, {rs:gpr}({rt:gpr})`

Availability

MMAR < R6

Decode priority

0

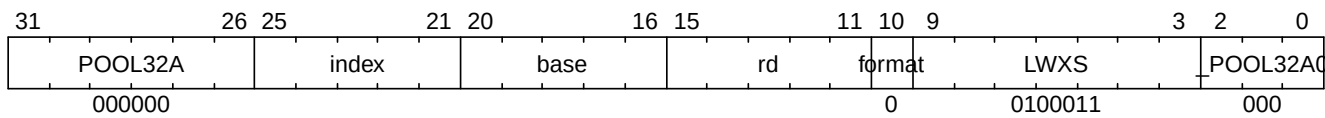


Figure 879. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd=rd, rs=index, rt=base)
```

nanoMIPS

`lwx.s.16`

Assembly

`LWXS {rd:gpr3}, {rs:gpr3}({rt:gpr3})`

Availability

true

Decode priority

0

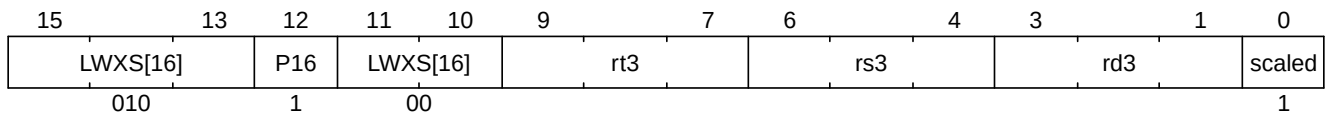


Figure 880. Encoding (16 bits)

Decode

```
return IndexedMemoryOperands(  
    GPR3.decode(rd3),  
    GPR3.decode(rs3),  
    GPR3.decode(rt3),  
)
```

lwx.s.32

Assembly

LWXS {rs:gpr}({rt:gpr})

Availability

true

Decode priority

0

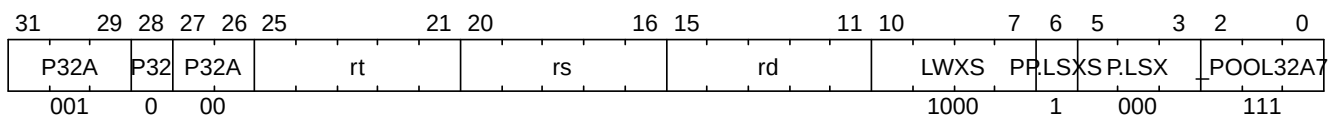


Figure 881. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.324.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)  
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)) << 2  
va = effective_address(ctx, base, index, "Load")  
data = read_memory_at_va(ctx, va, nbytes=4)  
ctx.GPR[operands.rd] = sign_extend(data, from_nbits=32)
```

8.324.4. Description

The effective address is formed from two full-width GPR values.

8.324.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.

- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.325. MADD

8.325.1. Purpose

MADD a 32-bit product with the classic HI/LO accumulator.

8.325.2. Formats

MIPS

madd

Assembly

MADD {rs:gpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

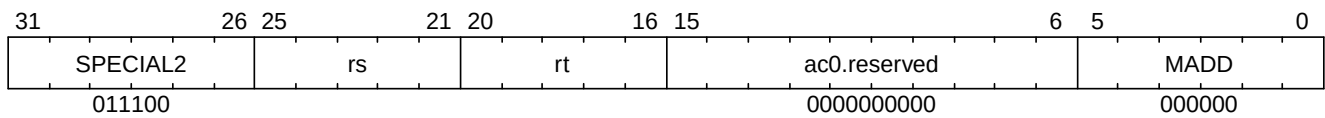


Figure 882. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

madd.dsp

Assembly

MADD {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

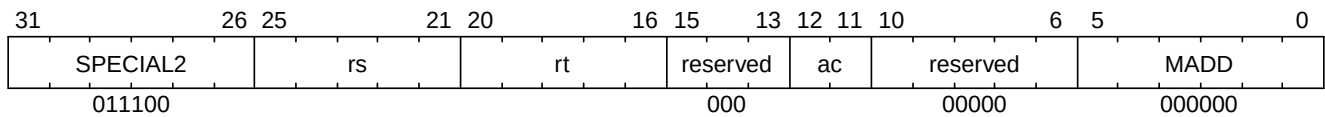


Figure 883. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

microMIPS

madd.32

Assembly

MADD {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

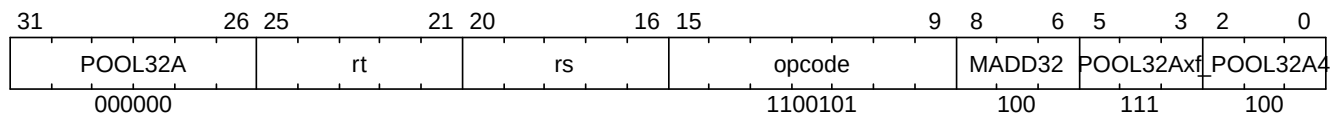


Figure 884. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

madd.dsp.32

Assembly

MADD {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

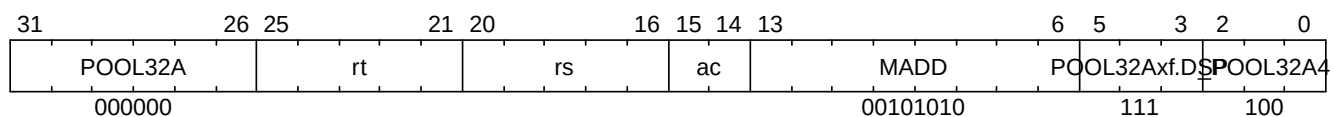


Figure 885. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

nanoMIPS

madd.dsp.r7

Assembly

MADD {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

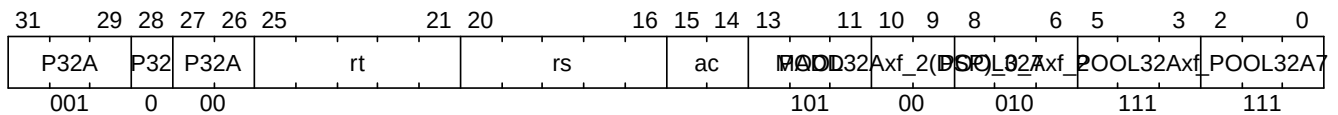


Figure 886. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

8.325.3. Operation

```
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
write_selected_accumulator(
    ctx,
    operands.ac,
    selected_accumulator_value(ctx, operands.ac)
    + classic_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False),
)
```

8.325.4. Description

The selected signed or unsigned 32-bit product is added to or subtracted from the 64-bit HI:LO value modulo 2^{64} . The base encoding selects accumulator zero.

8.325.5. Exceptions

None.

8.326. MADD.D

8.326.1. Purpose

Compute the pre-Release-6 madd of D-format values using the explicit accumulator `fr`.

8.326.2. Formats

MIPS

`madd.d`

Assembly

`MADD.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR < R6

Decode priority

0

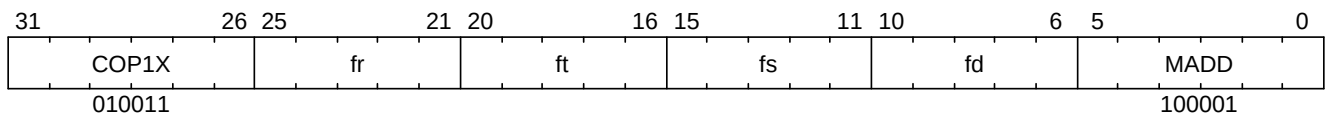


Figure 887. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

`madd.d`

Assembly

`MADD.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR < R6

Decode priority

0

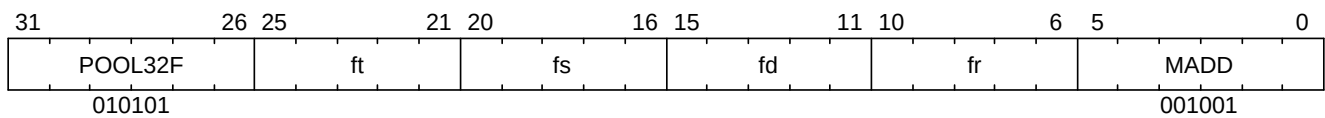


Figure 888. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.326.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_mul_add(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "D"),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.326.4. Description

MADD.D combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.326.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.327. MADD.PS

8.327.1. Purpose

Compute the paired-single madd operation.

8.327.2. Formats

MIPS

madd.ps

Assembly

MADD.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

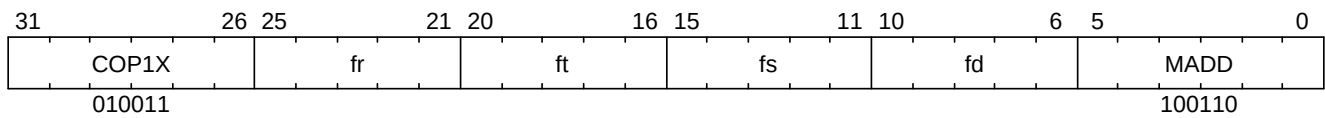


Figure 889. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

madd.ps

Assembly

MADD.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

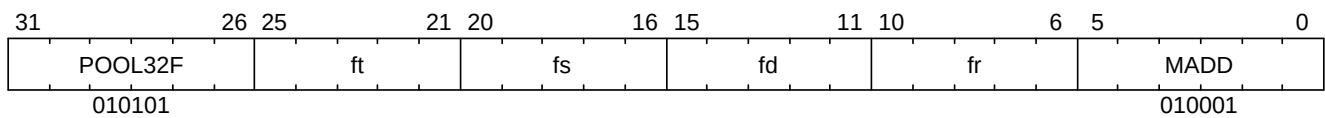


Figure 890. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.327.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
fr = get_fpr(ctx, operands.fr, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
low, low_flags = ieee_mul_add(
    "S", rounding, fr & 0xFFFF_FFFF, fs & 0xFFFF_FFFF, ft & 0xFFFF_FFFF
)
high, high_flags = ieee_mul_add("S", rounding, fr >> 32, fs >> 32, ft >> 32)
write_fpu_result(ctx, operands.fd, low | (high << 32), low_flags | high_flags, "PS")
```

8.327.4. Description

MADD.PS applies independently to the upper and lower S-format lanes.

8.327.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.328. MADD.S

8.328.1. Purpose

Compute the pre-Release-6 madd of S-format values using the explicit accumulator **fr**.

8.328.2. Formats

MIPS

madd.s

Assembly

MADD.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

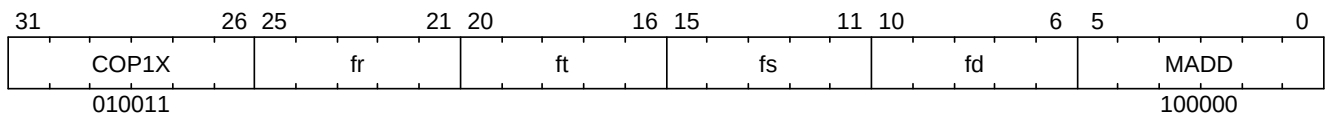


Figure 891. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

madd.s

Assembly

MADD.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

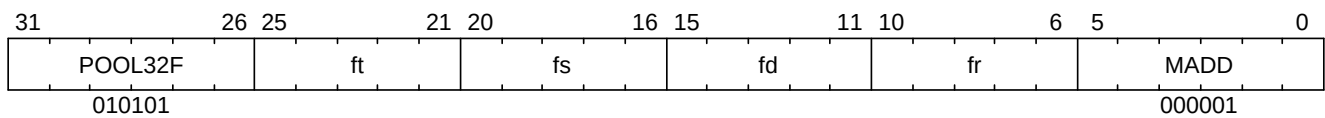


Figure 892. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.328.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_mul_add(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "S"),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.328.4. Description

MADD.S combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.328.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.329. MADDF.D

8.329.1. Purpose

Fused multiply-add D-format values.

8.329.2. Formats

MIPS

maddf.d

Assembly

MADDF.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

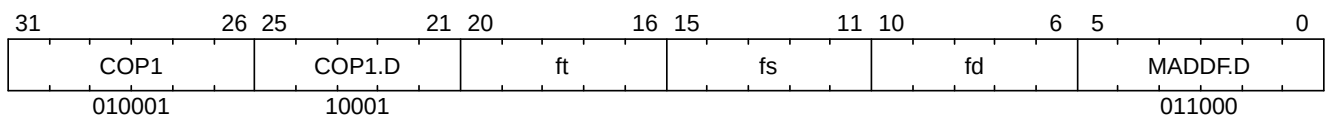


Figure 893. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

maddf.d

Assembly

MADDF.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

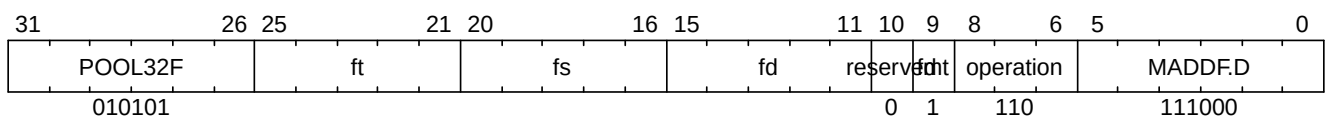


Figure 894. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

maddf.d

Assembly

MADDF.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

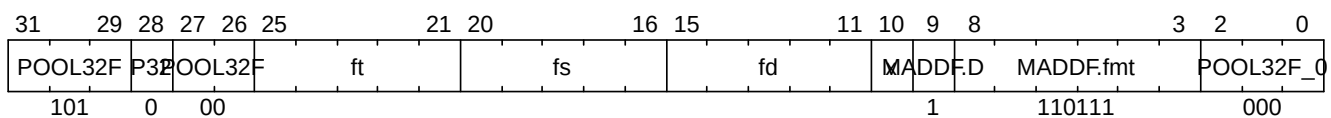


Figure 895. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.329.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_madd(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fd, "D"),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.329.4. Description

MADDF.D validates CP1 and D-format support before updating the destination FPR.

8.329.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.330. MADDF.S

8.330.1. Purpose

Fused multiply-add S-format values.

8.330.2. Formats

MIPS

maddf.s

Assembly

MADDF.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

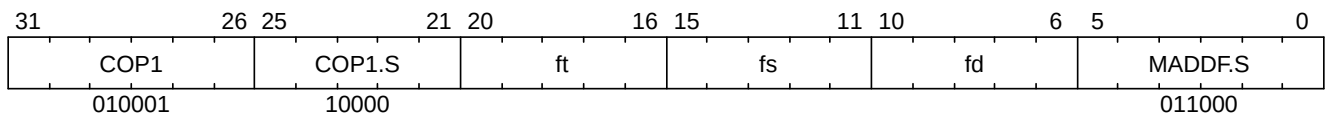


Figure 896. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

maddf.s

Assembly

MADDF.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

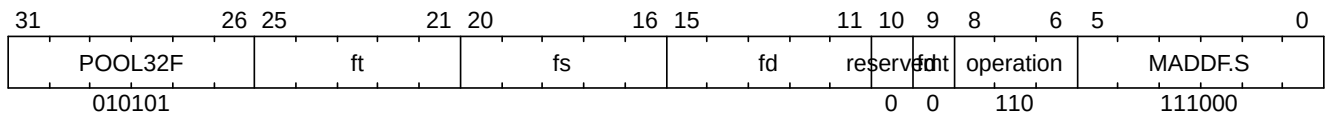


Figure 897. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

maddf.s

Assembly

MADDF.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

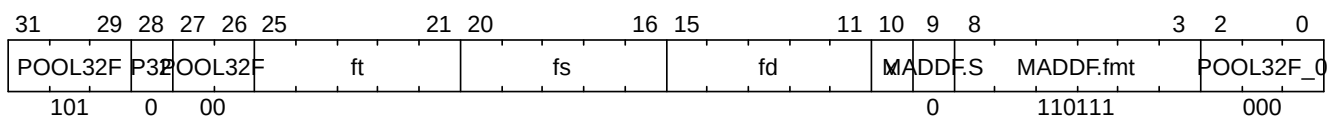


Figure 898. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.330.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_madd(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fd, "S"),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.330.4. Description

MADDF.S validates CP1 and S-format support before updating the destination FPR.

8.330.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.331. MADDU

8.331.1. Purpose

MADDU a 32-bit product with the classic HI/LO accumulator.

8.331.2. Formats

MIPS

maddu

Assembly

MADDU {rs:gpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

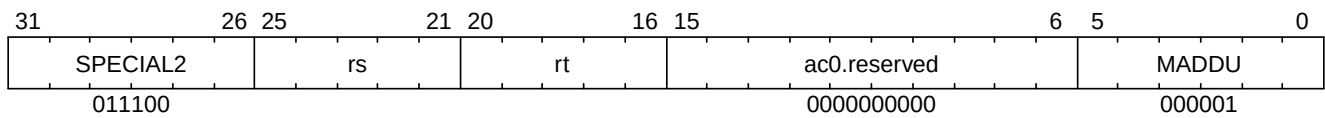


Figure 899. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

maddu.dsp

Assembly

MADDU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

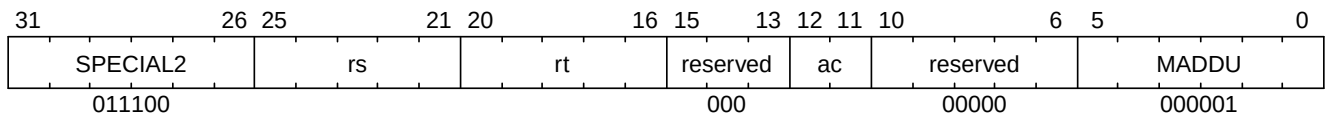


Figure 900. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

microMIPS

maddu.32

Assembly

MADDU {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

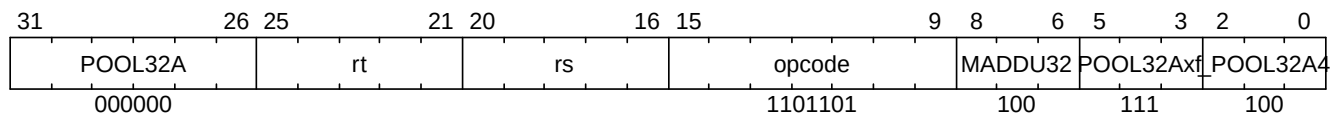


Figure 901. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

maddu.dsp.32

Assembly

MADDU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

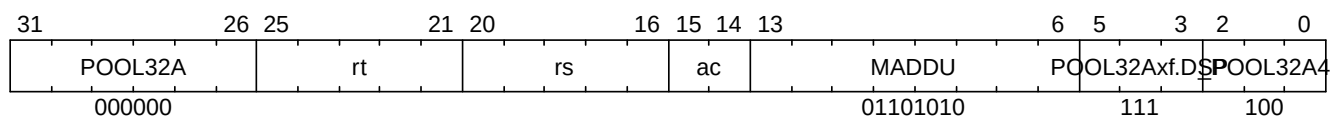


Figure 902. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

nanoMIPS

maddu.dsp.r7

Assembly

MADDU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

0

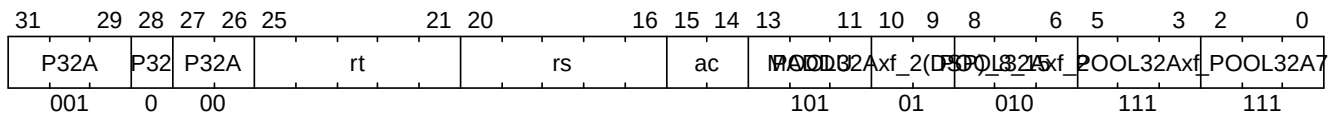


Figure 903. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

8.331.3. Operation

```

if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
write_selected_accumulator(
    ctx,
    operands.ac,
    selected_accumulator_value(ctx, operands.ac)
    + classic_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True),
)

```

8.331.4. Description

The selected signed or unsigned 32-bit product is added to or subtracted from the 64-bit HI:LO value modulo 2^{64} . The base encoding selects accumulator zero.

8.331.5. Exceptions

None.

8.332. MAQ_S.W.PHL

8.332.1. Purpose

DSP multiply or accumulate operation MAQ_S.W.PHL.

8.332.2. Formats

MIPS

maq_s.w.phl

Assembly

MAQ_S.W.PHL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

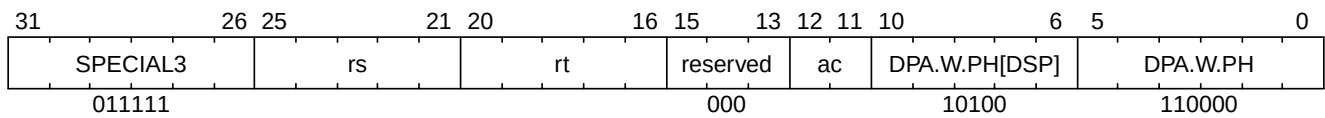


Figure 904. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

maq_s.w.phl.32

Assembly

MAQ_S.W.PHL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

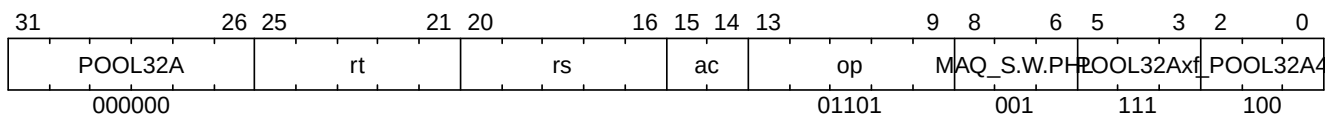


Figure 905. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

maq_s.w.phl.32

Assembly

MAQ_S.W.PHL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

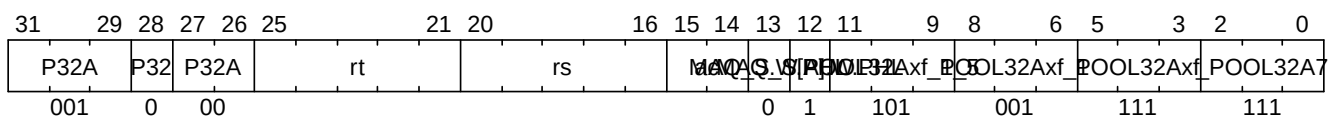


Figure 906. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.332.3. Operation

```
validate_access_to_dsp_resources(ctx)
product, overflow = muleq_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(
    ctx, operands.ac, accumulator_value(ctx, operands.ac) + signed(product, 32)
)
```

8.332.4. Description

MAQ_S.W.PHL is a DSP multiply or accumulator operation.

8.332.5. Exceptions

None.

8.333. MAQ_S.W.PHR

8.333.1. Purpose

DSP multiply or accumulate operation MAQ_S.W.PHR.

8.333.2. Formats

MIPS

maq_s.w.phr

Assembly

MAQ_S.W.PHR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

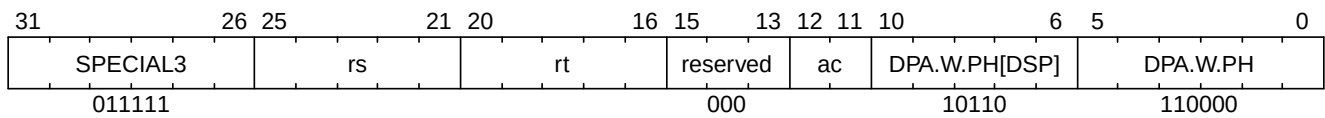


Figure 907. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

maq_s.w.phr.32

Assembly

MAQ_S.W.PHR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

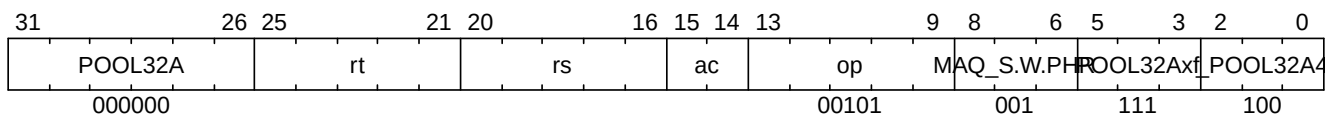


Figure 908. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

maq_s.w.phr.32

Assembly

MAQ_S.W.PHR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

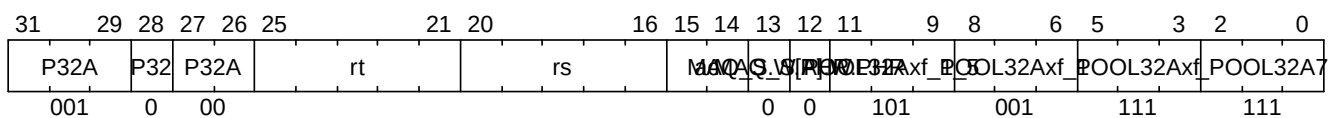


Figure 909. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.333.3. Operation

```
validate_access_to_dsp_resources(ctx)
product, overflow = muleq_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(
    ctx, operands.ac, accumulator_value(ctx, operands.ac) + signed(product, 32)
)
```

8.333.4. Description

MAQ_S.W.PHR is a DSP multiply or accumulator operation.

8.333.5. Exceptions

None.

8.334. MAQ_SA.W.PHL

8.334.1. Purpose

DSP multiply or accumulate operation MAQ_SA.W.PHL.

8.334.2. Formats

MIPS

maq_sa.w.phl

Assembly

MAQ_SA.W.PHL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

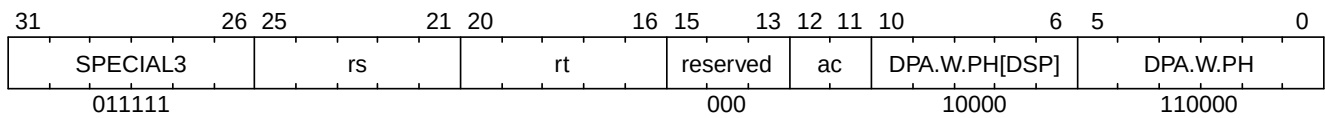


Figure 910. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

maq_sa.w.phl.32

Assembly

MAQ_SA.W.PHL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

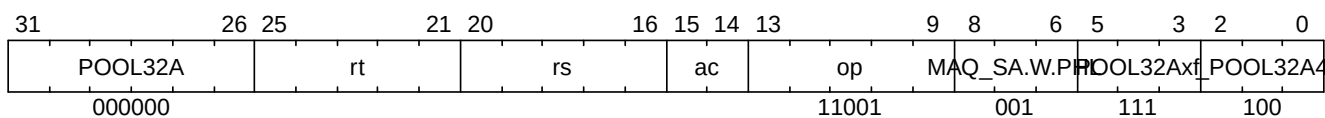


Figure 911. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

maq_sa.w.phl.32

Assembly

MAQ_SA.W.PHL {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

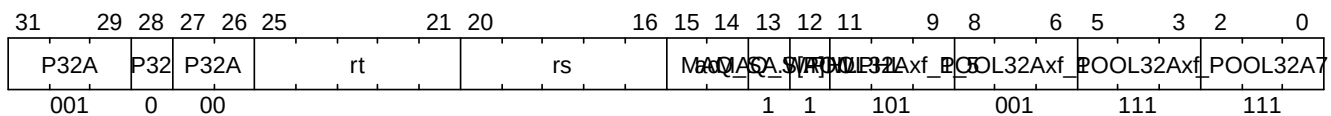


Figure 912. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.334.3. Operation

```
validate_access_to_dsp_resources(ctx)
product, overflow = muleq_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
result, saturated = saturate_accumulator_q31(
    signed(accumulator_value(ctx, operands.ac), 64) + signed(product, 32)
)
if saturated:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, result)
```

8.334.4. Description

MAQ_SA.W.PHL is a DSP multiply or accumulator operation.

8.334.5. Exceptions

None.

8.335. MAQ_SA.W.PHR

8.335.1. Purpose

DSP multiply or accumulate operation MAQ_SA.W.PHR.

8.335.2. Formats

MIPS

maq_sa.w.phr

Assembly

MAQ_SA.W.PHR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

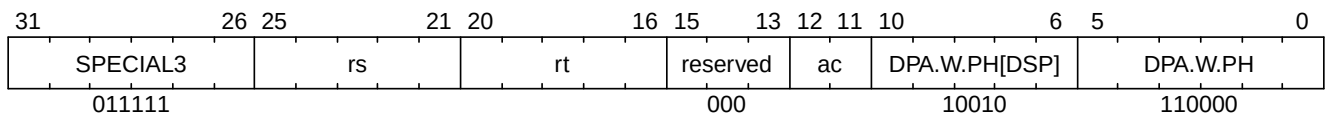


Figure 913. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

maq_sa.w.phr.32

Assembly

MAQ_SA.W.PHR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

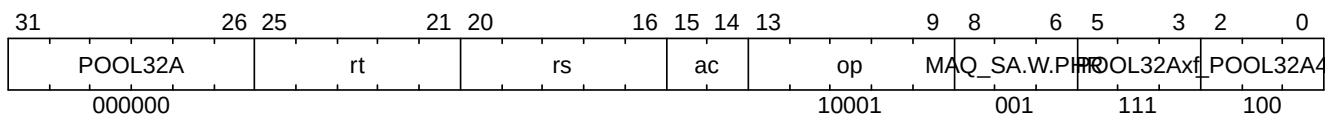


Figure 914. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

maq_sa.w.phr.32

Assembly

MAQ_SA.W.PHR {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

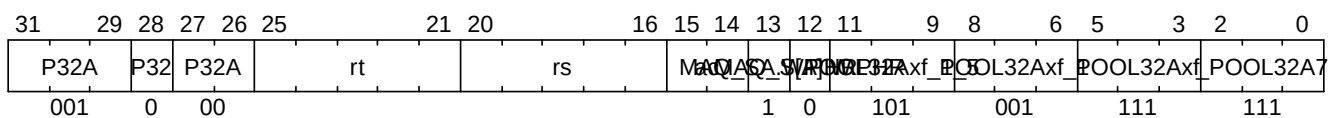


Figure 915. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.335.3. Operation

```
validate_access_to_dsp_resources(ctx)
product, overflow = muleq_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False)
if overflow:
    set_accumulator_overflow(ctx, operands.ac)
result, saturated = saturate_accumulator_q31(
    signed(accumulator_value(ctx, operands.ac), 64) + signed(product, 32)
)
if saturated:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(ctx, operands.ac, result)
```

8.335.4. Description

MAQ_SA.W.PHR is a DSP multiply or accumulator operation.

8.335.5. Exceptions

None.

8.336. MAX.D

8.336.1. Purpose

Select the larger D-format value.

8.336.2. Formats

MIPS

max.d

Assembly

MAX.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

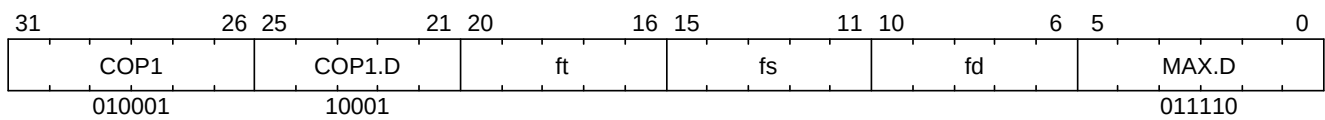


Figure 916. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

max.d

Assembly

MAX.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

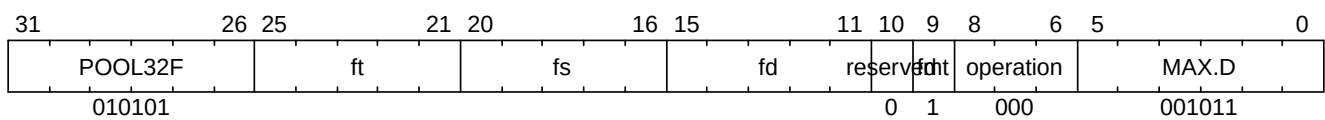


Figure 917. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

max.d

Assembly

MAX.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

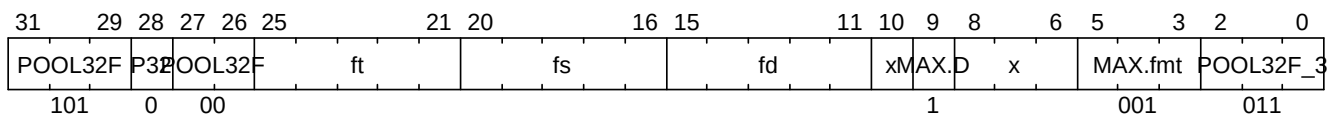


Figure 918. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.336.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_minmax(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    maximum=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.336.4. Description

MAX.D validates CP1 and D-format support before updating the destination FPR.

8.336.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.337. MAX.S

8.337.1. Purpose

Select the larger S-format value.

8.337.2. Formats

MIPS

max.s

Assembly

MAX.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

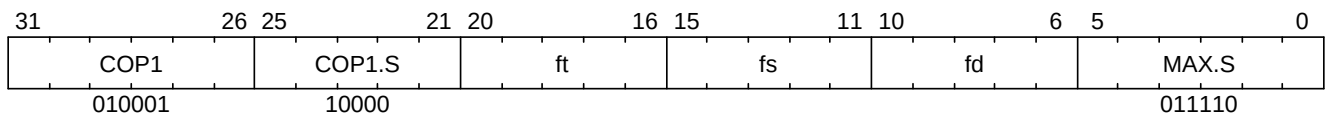


Figure 919. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

max.s

Assembly

MAX.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

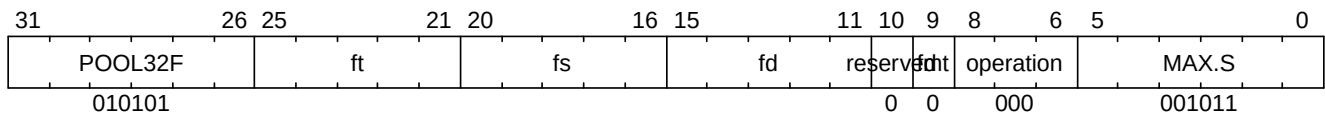


Figure 920. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

max.s

Assembly

MAX.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

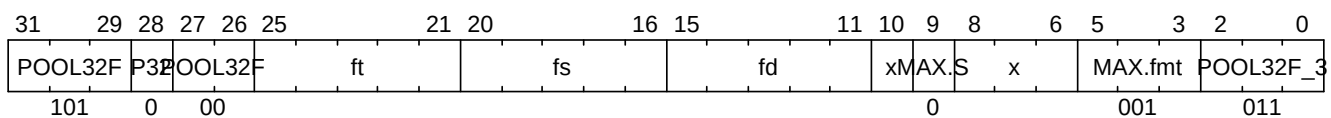


Figure 921. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.337.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_minmax(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    maximum=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.337.4. Description

MAX.S validates CP1 and S-format support before updating the destination FPR.

8.337.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.338. MAXA.D

8.338.1. Purpose

Select by D-format magnitude.

8.338.2. Formats

MIPS

maxa.d

Assembly

MAXA.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

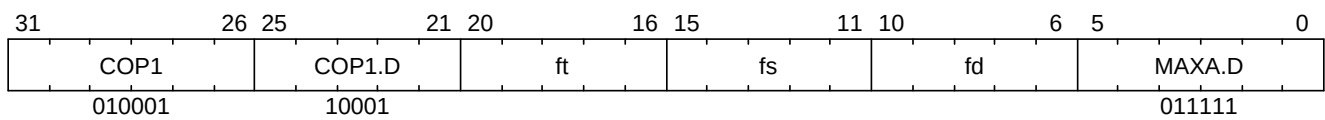


Figure 922. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

maxa.d

Assembly

MAXA.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

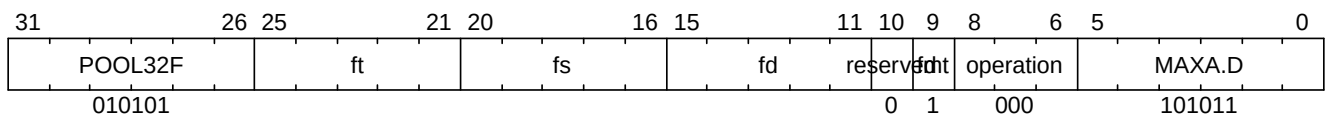


Figure 923. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

maxa.d

Assembly

MAXA.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

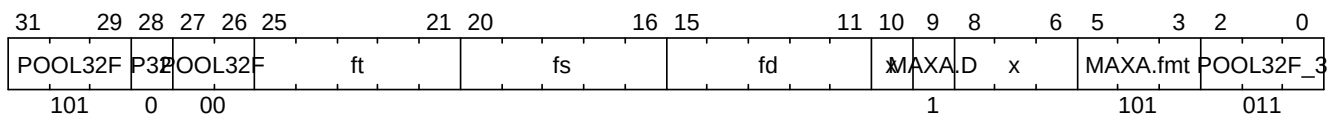


Figure 924. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.338.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_minmax(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    maximum=True,
    absolute=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.338.4. Description

MAXA.D validates CP1 and D-format support before updating the destination FPR.

8.338.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.339. MAXA.S

8.339.1. Purpose

Select by S-format magnitude.

8.339.2. Formats

MIPS

maxa.s

Assembly

MAXA.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

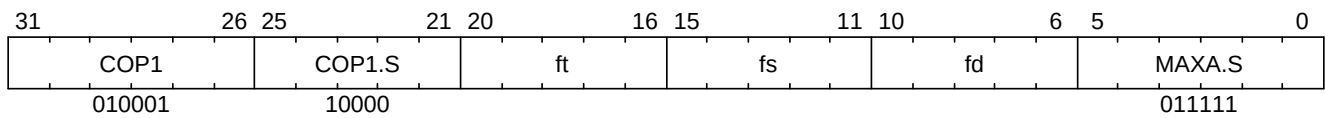


Figure 925. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

maxa.s

Assembly

MAXA.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

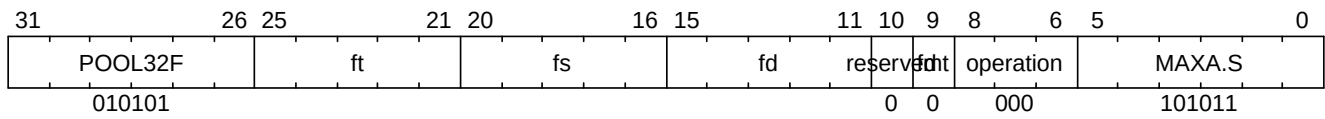


Figure 926. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

maxa.s

Assembly

MAXA.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

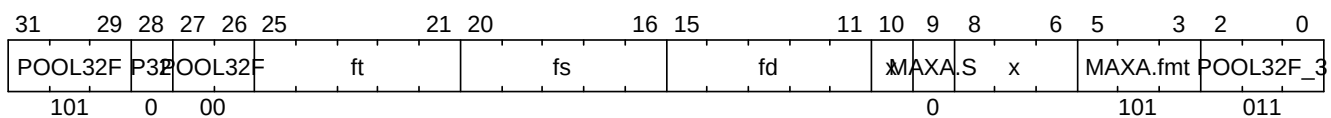


Figure 927. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.339.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_minmax(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    maximum=True,
    absolute=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.339.4. Description

MAXA.S validates CP1 and S-format support before updating the destination FPR.

8.339.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.340. MFC0

8.340.1. Purpose

Read an encoded CP0 register and place its low 32-bit value in a destination GPR.

8.340.2. Formats

MIPS

mfc0

Assembly

MFC0 {rt:gpr}, {c0s:u5}, {sel:u3}

Availability

true

Decode priority

0

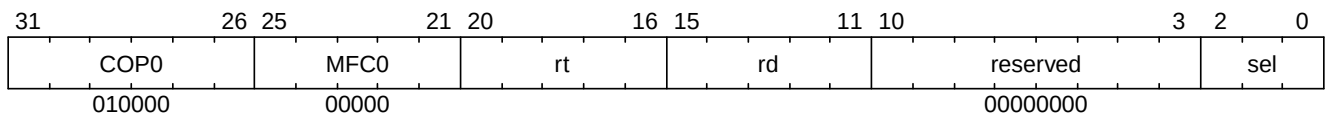


Figure 928. Encoding (32 bits)

Decode

```
return Mfc00perands(rt=rt, c0s=rd, sel=zero_extend(sel, from_nbits=3))
```

microMIPS

mfc0.32

Assembly

MFC0 {rt:gpr}, {c0s:u5}, {sel:u3}

Availability

true

Decode priority

0

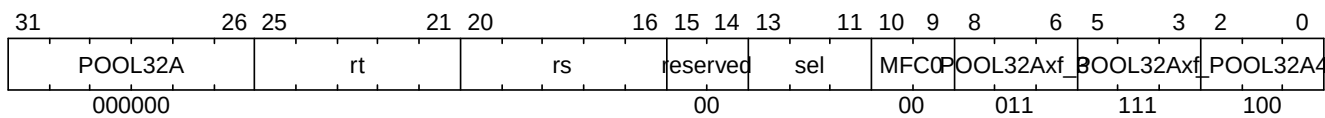


Figure 929. Encoding (32 bits)

Decode

```
return Mfc0Operands(rt=rt, c0s=rs, sel=zero_extend(sel, from_nbits=3))
```

nanoMIPS

mfc0.32

Assembly

MFC0 {rt:gpr}, {c0s:u5}, {sel:u5}

Availability

true

Decode priority

0

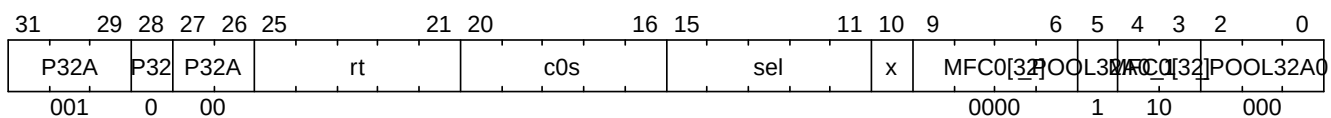


Figure 930. Encoding (32 bits)

Decode

```
return Mfc0Operands(rt=rt, c0s=c0s, sel=sel)
```

8.340.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
value = read_cp0_register(ctx, operands.c0s, operands.sel)  
ctx.GPR[operands.rt] = sign_extend(value, from_nbits=32)
```

8.340.4. Description

MFC0[32] resolves (*c0s*, *sel*) through the active CP0 register-bank definition. An encoding not used by the current core reads as zero. The 32-bit result is sign-extended to the implemented GPR width, and CP0 access requires the current privilege state to permit CP0.

8.340.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; MFC0 requires an enabled Coprocessor 0 privilege state.

8.341. MFC1

8.341.1. Purpose

Move the low 32 bits of a Coprocessor 1 register to a GPR.

8.341.2. Formats

MIPS

mfc1

Assembly

MFC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

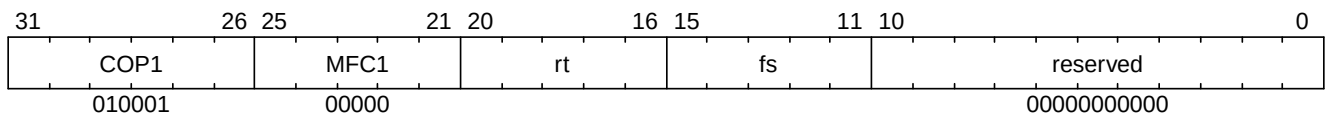


Figure 931. Encoding (32 bits)

Decode

```
return Mfc10perands(rt=rt, fs=fs)
```

microMIPS

mfc1.32

Assembly

MFC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

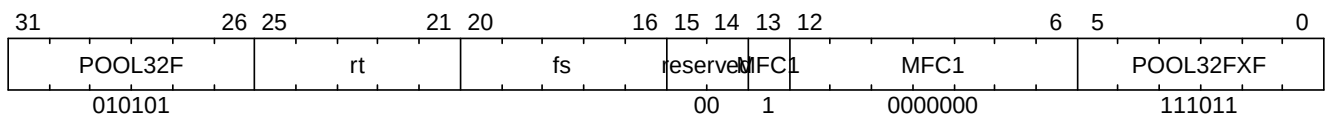


Figure 932. Encoding (32 bits)

Decode

```
return Mfc10perands(rt=rt, fs=fs)
```

nanoMIPS

mfc1.32

Assembly

MFC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

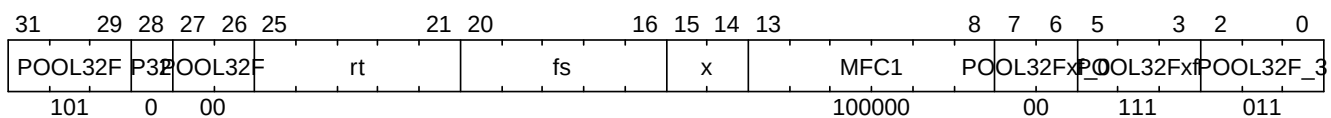


Figure 933. Encoding (32 bits)

Decode

```
return Mfc10perands(rt=rt, fs=fs)
```

8.341.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
ctx.GPR[operands.rt] = sign_extend(get_fpr(ctx, operands.fs, "32"), from_nbits=32)
```

8.341.4. Description

MFC1 copies bits 31..0 of FPR *fs* to GPR *rt* and sign-extends the word to the implemented GPR width. On classic MIPS, disabled CP1 access is checked before CP1 implementation and format checks.

8.341.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled by Status.CU1.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented.

8.342. MFC2

8.342.1. Purpose

Move a CP2 register to a GPR.

8.342.2. Formats

microMIPS

mfc2.32

Assembly

MFC2 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

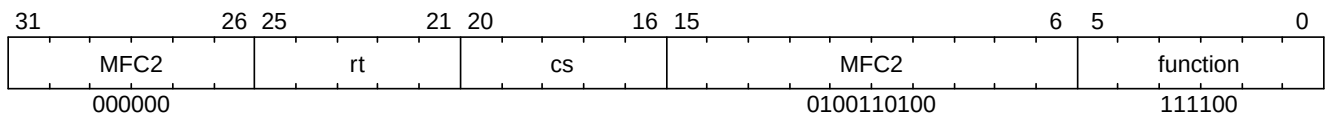


Figure 934. Encoding (32 bits)

Decode

```
return Cop2MoveOperands(rt=rt, cs=cs)
```

8.342.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("MFC2 execution is not implemented")
```

8.342.4. Description

MFC2 checks CP2 availability before accessing the register.

8.342.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; MFC2 execution is not

implemented

8.343. MFHC0

8.343.1. Purpose

Read high CP0 register bits.

8.343.2. Formats

MIPS

`mfhc0`

Assembly

`MFHC0 {rt:gpr}, {c0s:u5}, {sel:u3}`

Availability

AT >= MIPS64_32 || MVH

Decode priority

0

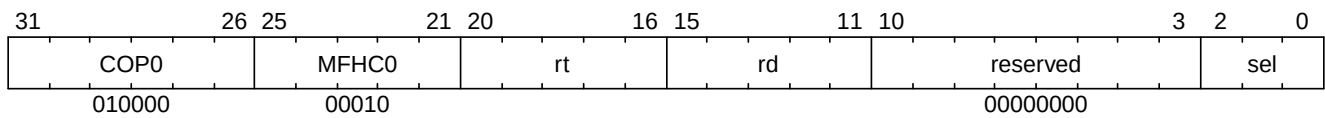


Figure 935. Encoding (32 bits)

Decode

```
return Mfhc0operands(rt=rt, c0s=rd, sel=zero_extend(sel, from_nbits=3))
```

microMIPS

`mfhc0.32`

Assembly

`MFHC0 {rt:gpr}, {c0s:u5}, {sel:u3}`

Availability

AT >= MIPS64_32 || MVH

Decode priority

0

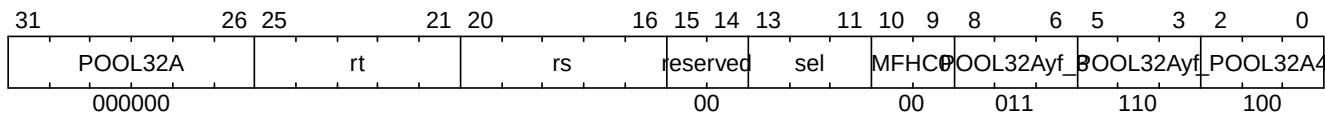


Figure 936. Encoding (32 bits)

Decode

```
return Mfhc0Operands(rt=rt, c0s=rs, sel=zero_extend(sel, from_nbits=3))
```

nanoMIPS

mfhc0.32

Assembly

MFHC0 {rt:gpr}, {c0s:u5}, {sel:u5}

Availability

true

Decode priority

0

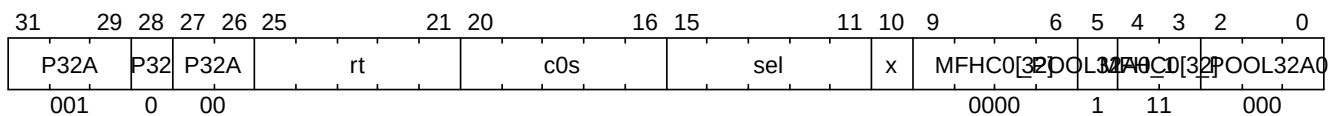


Figure 937. Encoding (32 bits)

Decode

```
return Mfhc0Operands(rt=rt, c0s=c0s, sel=sel)
```

8.343.3. Operation

```
if not ctx.C0.Config5.MVH:
    raise ReservedInstruction("MFHC0 requires MVH")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
ctx.GPR[operands.rt] = sign_extend(
    read_cp0_register_high(ctx, operands.c0s, operands.sel), from_nbits=32
)
```

8.343.4. Description

MFHC0 reads bits 63:32 of an extended CP0 register.

8.343.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — MFHC0 requires MVH

8.344. MFHC1

8.344.1. Purpose

Transfer a value through Coprocessor 1.

8.344.2. Formats

MIPS

`mfhc1`

Assembly

`MFHC1 {rt:gpr}, {fs:fpr}`

Availability

AR >= R2_3_5 && (AT >= MIPS64_32 || FPU64)

Decode priority

0

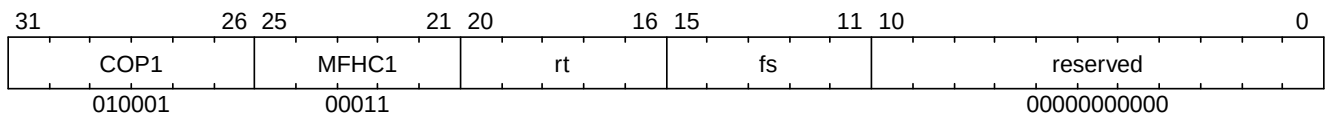


Figure 938. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

microMIPS

`mfhc1.32`

Assembly

`MFHC1 {rt:gpr}, {fs:fpr}`

Availability

AT >= MIPS64_32 || FPU64

Decode priority

0

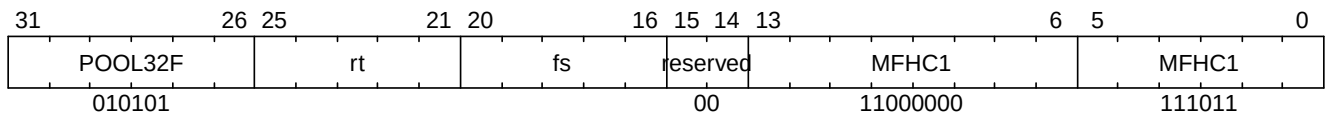


Figure 939. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

nanoMIPS

mfhc1.32

Assembly

MFHC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

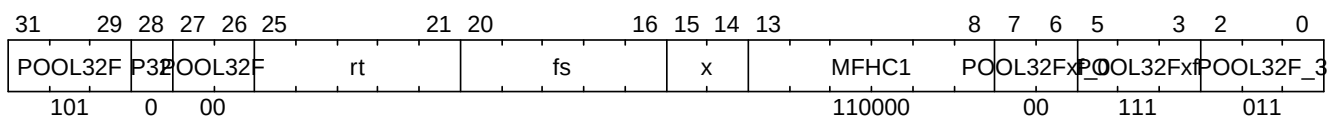


Figure 940. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

8.344.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
ctx.GPR[operands.rt] = sign_extend(
    (get_fpr(ctx, operands.fs, "64") >> 32) & 0xFFFF_FFFF, from_nbits=32
)
```

8.344.4. Description

MFHC1 performs its CP1 access checks before transferring data.

8.344.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.

8.345. MFHC2

8.345.1. Purpose

Move the high half of a CP2 register to a GPR.

8.345.2. Formats

microMIPS

`mfhc2.32`

Assembly

`MFHC2 {rt:gpr}, {cs:u5}`

Availability

true

Decode priority

0

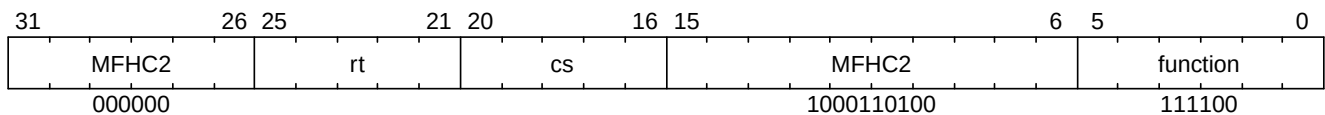


Figure 941. Encoding (32 bits)

Decode

```
return Cop2MoveOperands(rt=rt, cs=cs)
```

8.345.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("MFHC2 execution is not implemented")
```

8.345.4. Description

MFHC2 checks CP2 availability before accessing the register.

8.345.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; MFHC2 execution is not

implemented

8.346. MFHI

8.346.1. Purpose

Copy the classic multiply/divide HI register to a GPR.

8.346.2. Formats

MIPS

`mfhi.dsp`

Assembly

`MFHI {rd:gpr}, {ac:u2}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

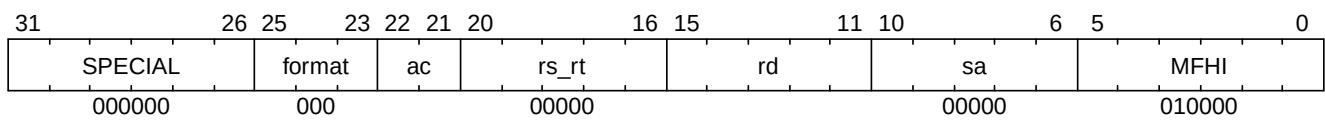


Figure 942. Encoding (32 bits)

Decode

```
return MfhiOperands(rd=rd, ac=ac)
```

`mfhi`

Assembly

`MFHI {rd:gpr}`

Availability

AR < R6

Decode priority

0

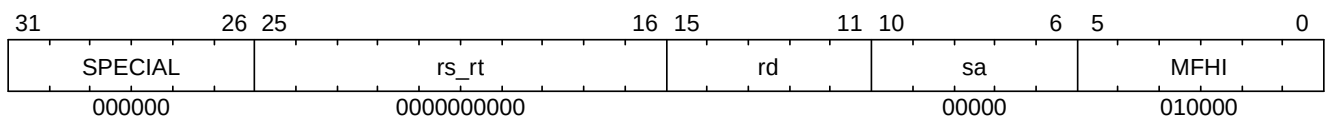


Figure 943. Encoding (32 bits)

Decode

```
return MfhiOperands(rd=rd)
```

microMIPS

mfhi.dsp.32

Assembly

MFHI {rd:gpr}, {ac:u2}

Availability

DSP

Decode priority

0

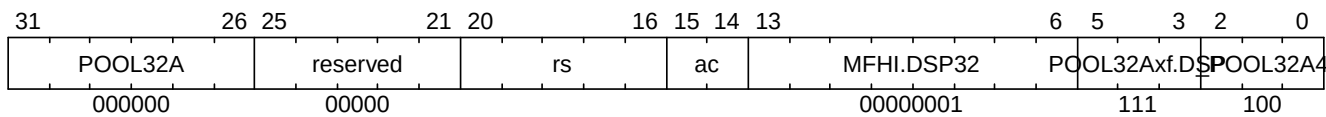


Figure 944. Encoding (32 bits)

Decode

```
return MfhiOperands(rd=rs, ac=ac)
```

mfhi.16

Assembly

MFHI {rd:gpr}

Availability

MMAR < R6

Decode priority

0

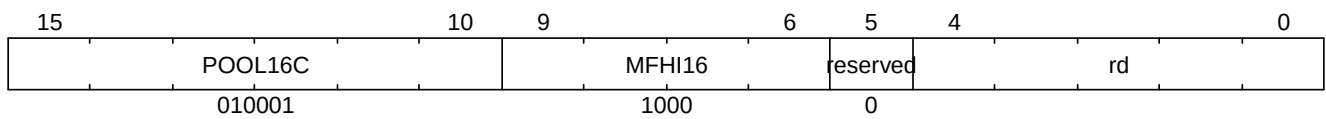


Figure 945. Encoding (16 bits)

Decode

```
return MfhiOperands(rd=rd)
```

mfhi.32

Assembly

MFHI {rd:gpr}

Availability

MMAR < R6

Decode priority

0

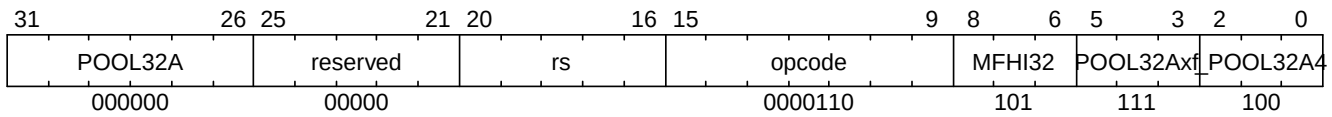


Figure 946. Encoding (32 bits)

Decode

```
return MfhiOperands(rd=rs)
```

nanoMIPS

mfhi.dsp.r7

Assembly

MFHI {rd:gpr}, {ac:u2}

Availability

DSP

Decode priority

0

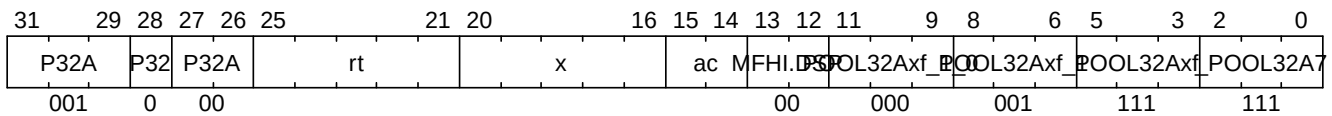


Figure 947. Encoding (32 bits)

Decode

```
return MfhiOperands(rd=rt, ac=ac)
```

8.346.3. Operation

```
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
    value = ctx.AC[operands.ac].HI
else:
    value = ctx.HI
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=ctx.GPR.width)
```

8.346.4. Description

MFHI copies HI at the implemented GPR width. In a 32-bit context the word value is sign-extended; in a 64-bit context the full 64-bit HI value is copied.

8.346.5. Exceptions

None.

8.347. MFLO

8.347.1. Purpose

Copy the classic multiply/divide LO register to a GPR.

8.347.2. Formats

MIPS

`mflo.dsp`

Assembly

`MFLO {rd:gpr}, {ac:u2}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

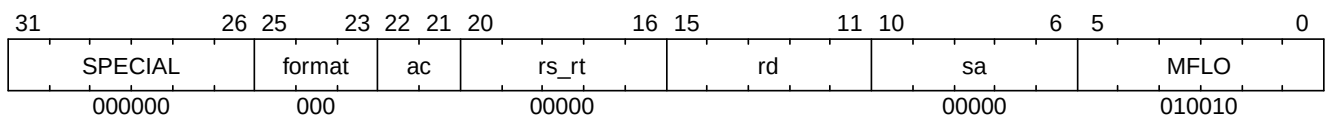


Figure 948. Encoding (32 bits)

Decode

```
return MfloOperands(rd=rd, ac=ac)
```

`mflo`

Assembly

`MFLO {rd:gpr}`

Availability

AR < R6

Decode priority

0

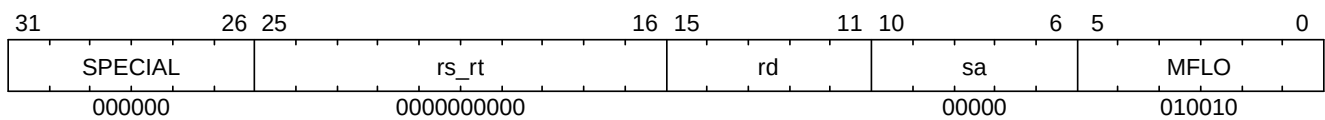


Figure 949. Encoding (32 bits)

Decode

```
return MfloOperands(rd=rd)
```

microMIPS

mflo.dsp.32

Assembly

MFLO {rd:gpr}, {ac:u2}

Availability

DSP

Decode priority

0

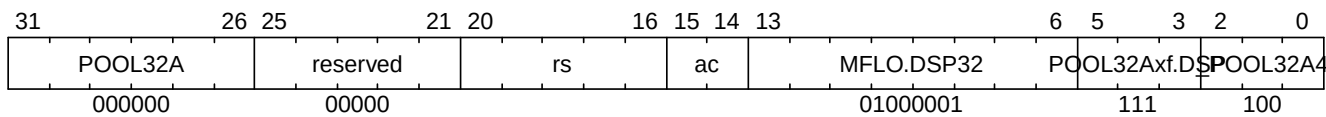


Figure 950. Encoding (32 bits)

Decode

```
return MfloOperands(rd=rs, ac=ac)
```

mflo.16

Assembly

MFLO {rd:gpr}

Availability

MMAR < R6

Decode priority

0

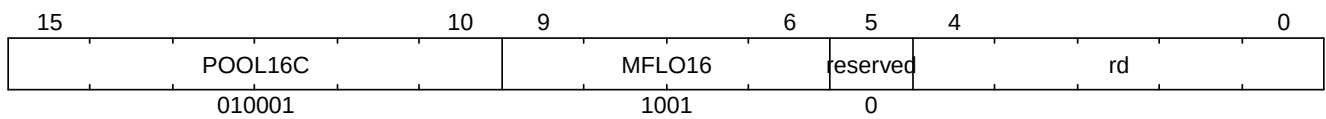


Figure 951. Encoding (16 bits)

Decode

```
return MfloOperands(rd=rd)
```

mflo.32

Assembly

MFLO {rd:gpr}

Availability

MMAR < R6

Decode priority

0

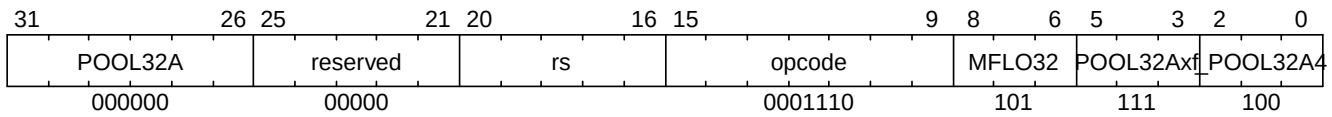


Figure 952. Encoding (32 bits)

Decode

```
return MfloOperands(rd=rs)
```

nanoMIPS

mflo.dsp.r7

Assembly

MFLO {rd:gpr}, {ac:u2}

Availability

DSP

Decode priority

0

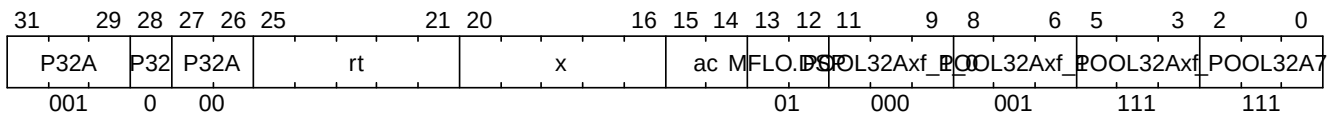


Figure 953. Encoding (32 bits)

Decode

```
return MfloOperands(rd=rt, ac=ac)
```

8.347.3. Operation

```
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
    value = ctx.AC[operands.ac].LO
else:
    value = ctx.LO
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=ctx.GPR.width)
```

8.347.4. Description

MFLO copies LO at the implemented GPR width. In a 32-bit context the word value is sign-extended; in a 64-bit context the full 64-bit LO value is copied.

8.347.5. Exceptions

None.

8.348. MIN.D

8.348.1. Purpose

Select the smaller D-format value.

8.348.2. Formats

MIPS

`min.d`

Assembly

`MIN.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

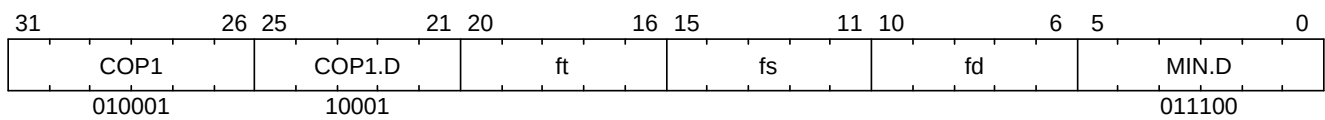


Figure 954. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`min.d`

Assembly

`MIN.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

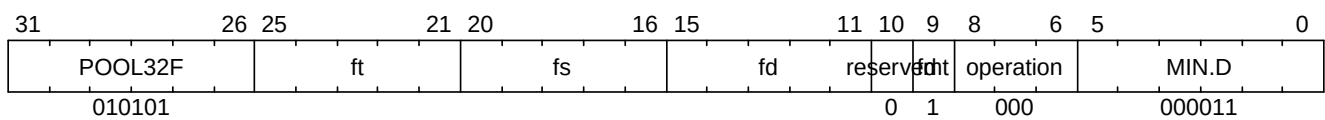


Figure 955. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

min.d

Assembly

MIN.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

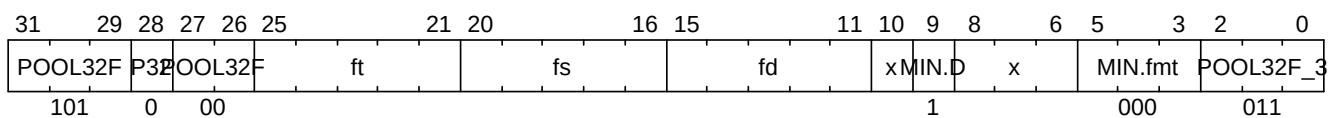


Figure 956. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.348.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_minmax(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    maximum=False,
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.348.4. Description

MIN.D validates CP1 and D-format support before updating the destination FPR.

8.348.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.349. MIN.S

8.349.1. Purpose

Select the smaller S-format value.

8.349.2. Formats

MIPS

`min.s`

Assembly

`MIN.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

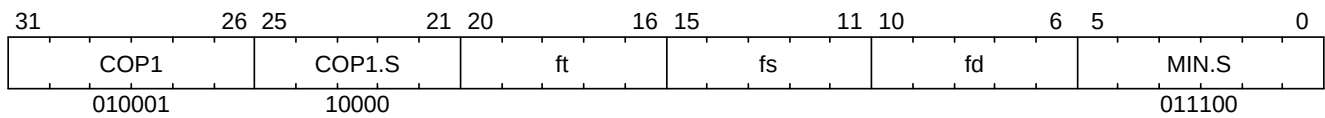


Figure 957. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`min.s`

Assembly

`MIN.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

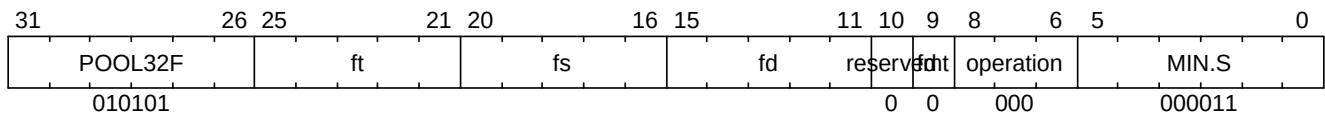


Figure 958. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

min.s

Assembly

MIN.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

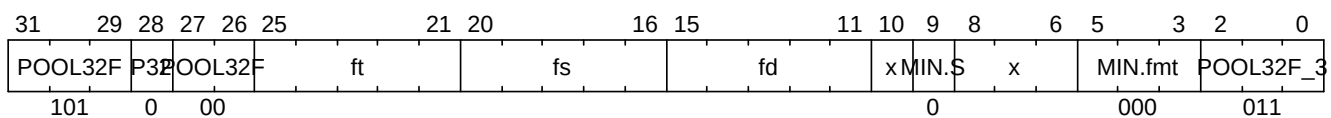


Figure 959. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.349.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_minmax(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    maximum=False,
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.349.4. Description

MIN.S validates CP1 and S-format support before updating the destination FPR.

8.349.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.350. MINA.D

8.350.1. Purpose

Select by D-format magnitude.

8.350.2. Formats

MIPS

`mina.d`

Assembly

`MINA.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

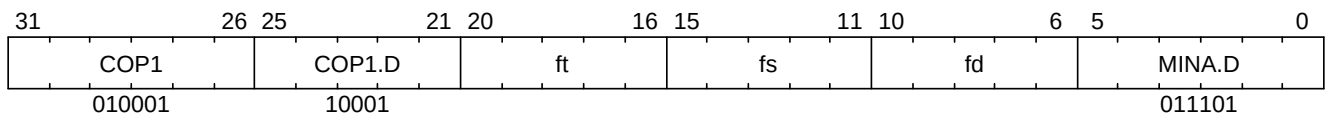


Figure 960. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`mina.d`

Assembly

`MINA.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

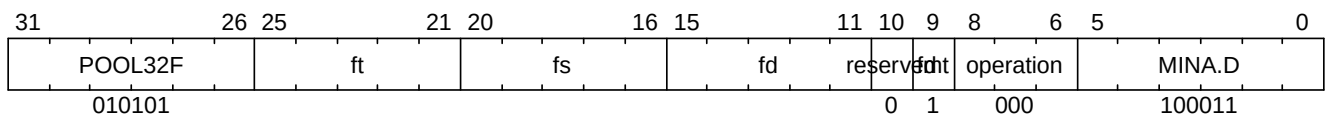


Figure 961. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

mina.d

Assembly

MINA.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

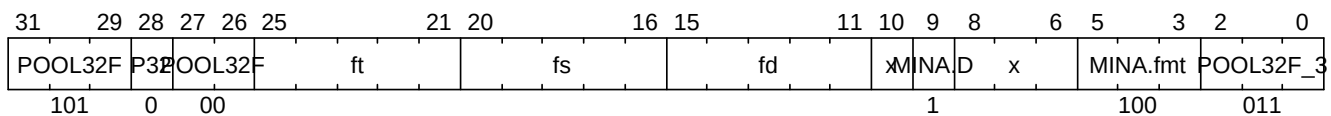


Figure 962. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.350.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_minmax(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    maximum=False,
    absolute=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.350.4. Description

MINA.D validates CP1 and D-format support before updating the destination FPR.

8.350.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.351. MINA.S

8.351.1. Purpose

Select by S-format magnitude.

8.351.2. Formats

MIPS

`mina.s`

Assembly

`MINA.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

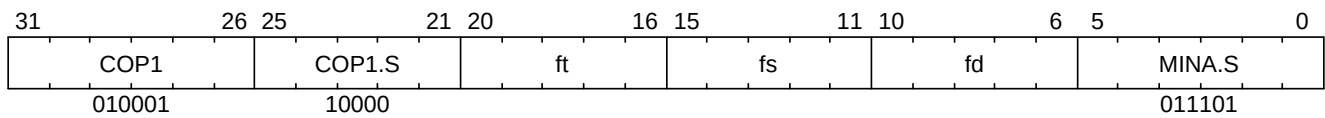


Figure 963. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`mina.s`

Assembly

`MINA.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

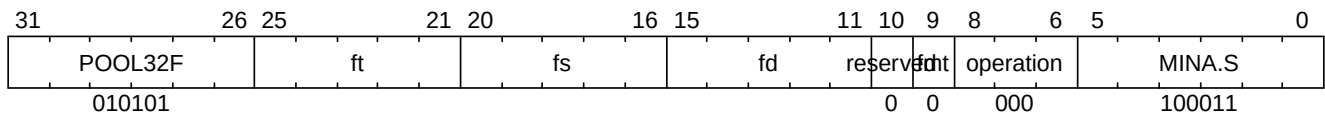


Figure 964. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

mina.s

Assembly

MINA.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

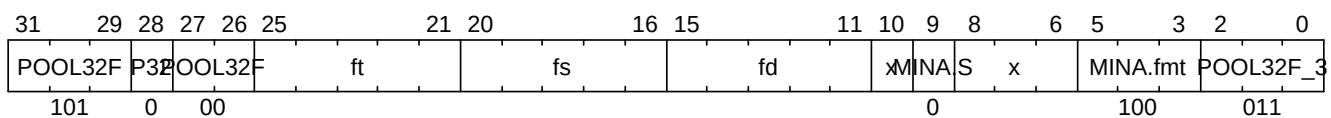


Figure 965. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.351.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_minmax(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    maximum=False,
    absolute=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.351.4. Description

MINA.S validates CP1 and S-format support before updating the destination FPR.

8.351.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.352. MOD

8.352.1. Purpose

Compute the signed 32-bit remainder of `rs` divided by `rt`, and place it in `rd`.

8.352.2. Formats

MIPS

`mod`

Assembly

`MOD {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R6

Decode priority

0

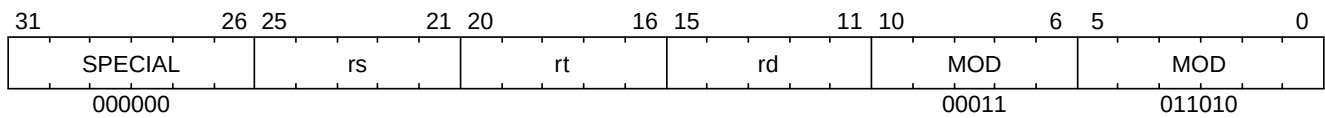


Figure 966. Encoding (32 bits)

Decode

```
return ModOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`mod.32.micromips`

Assembly

`MOD {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

MMAR >= R6

Decode priority

0

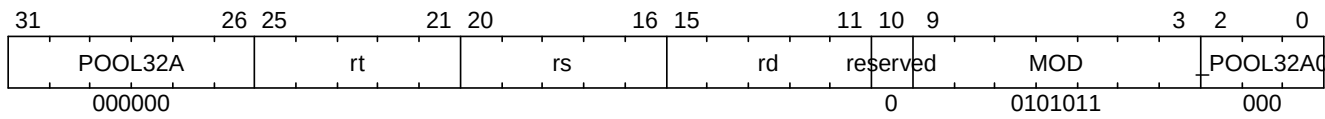


Figure 967. Encoding (32 bits)

Decode

```
return ModOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mod.32

Assembly

MOD {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

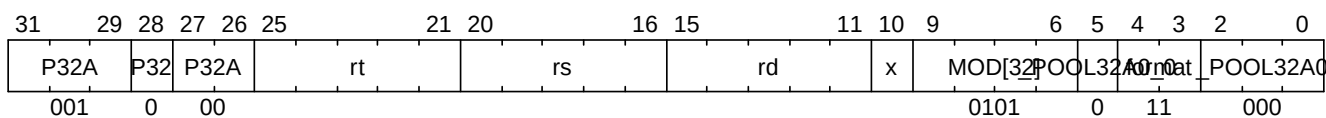


Figure 968. Encoding (32 bits)

Decode

```
del ctx, exec
return ModOperands(rd=rd, rs=rs, rt=rt)
```

8.352.3. Operation

```
remainder = divide_signed_word(
    ctx,
    ctx.GPR[operands.rs],
    ctx.GPR[operands.rt],
    operation="MOD[32]",
    result="remainder",
)
if remainder is not None:
    ctx.GPR[operands.rd] = sign_extend(remainder, from_nbits=32)
```

8.352.4. Description

MOD[32] interprets the low 32 bits of both source GPRs as signed values. The remainder is sign-extended from 32 bits when written to the implemented-width destination GPR. A zero divisor has an unspecified result and is routed through the replaceable division handler.

8.352.5. Exceptions

None.

8.353. MODSUB

8.353.1. Purpose

DSP packed operation MODSUB.

8.353.2. Formats

MIPS

modsub

Assembly

MODSUB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

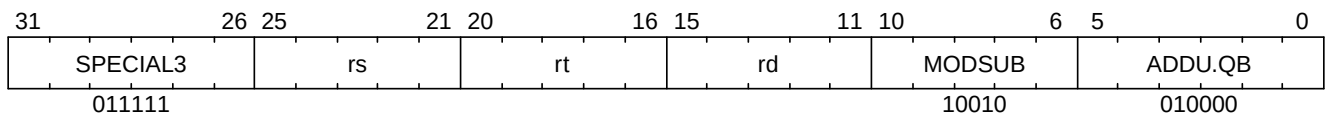


Figure 969. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

modsub.32

Assembly

MODSUB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

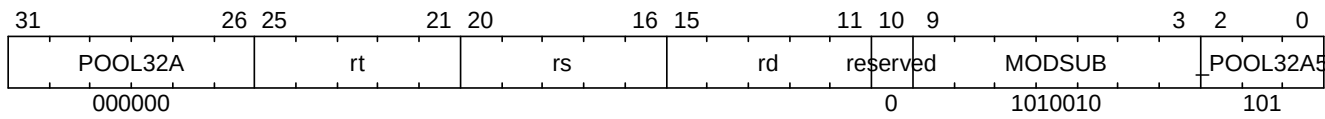


Figure 970. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

modsub.32

Assembly

MODSUB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

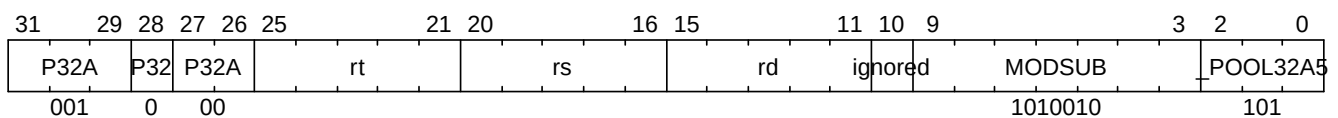


Figure 971. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.353.3. Operation

```
validate_access_to_dsp_resources(ctx)
decrement = ctx.GPR[operands.rt] & 0xFF
last_index = (ctx.GPR[operands.rt] >> 8) & 0xFFFF
if (ctx.GPR[operands.rs] & 0xFFFF_FFFF) == 0:
    ctx.GPR[operands.rd] = zero_extend(last_index, from_nbits=16)
else:
    ctx.GPR[operands.rd] = sign_extend(
        (ctx.GPR[operands.rs] - decrement) & 0xFFFF_FFFF, from_nbits=32
    )
```

8.353.4. Description

MODSUB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.353.5. Exceptions

None.

8.354. MODU

8.354.1. Purpose

Compute the unsigned 32-bit remainder of `rs` divided by `rt`, and place it in `rd`.

8.354.2. Formats

MIPS

`modu`

Assembly

`MODU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R6

Decode priority

0

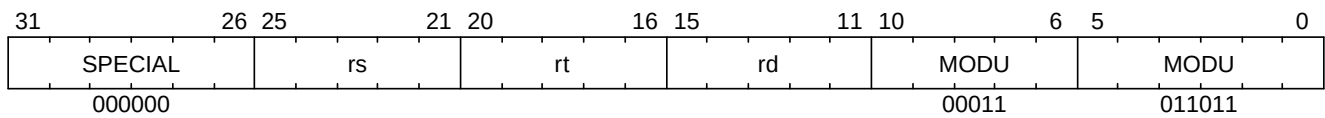


Figure 972. Encoding (32 bits)

Decode

```
return ModuOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`modu.32.micromips`

Assembly

`MODU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

MMAR >= R6

Decode priority

0

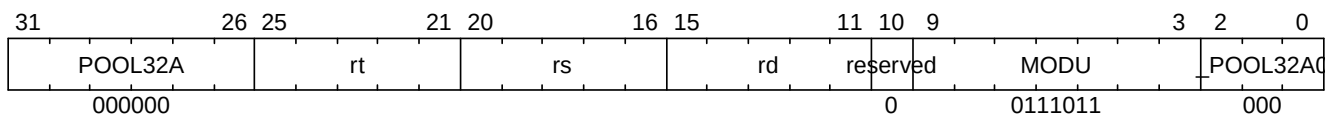


Figure 973. Encoding (32 bits)

Decode

```
return ModuOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

modu.32

Assembly

MODU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

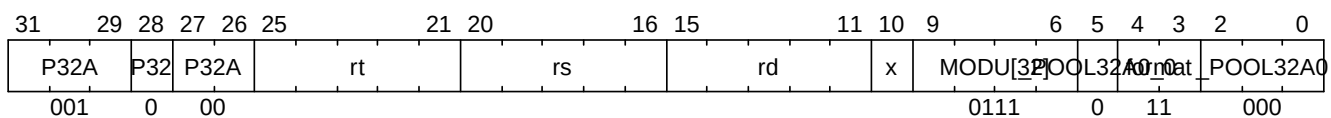


Figure 974. Encoding (32 bits)

Decode

```
del ctx, exec
return ModuOperands(rd=rd, rs=rs, rt=rt)
```

8.354.3. Operation

```
numerator = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
denominator = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
remainder = divide_unsigned_word(
    ctx,
    numerator,
    denominator,
    operation="MODU[32]",
    result="remainder",
)
if remainder is not None:
    ctx.GPR[operands.rd] = sign_extend(remainder, from_nbits=32)
```

8.354.4. Description

MODU[32] zero-extends both source GPRs from 32 bits before division. The remainder is sign-extended from 32 bits when written to the implemented-width destination GPR. A zero divisor has an unspecified result; the execution context may provide a replaceable handler. When none is supplied, the default implementation preserves the destination.

8.354.5. Exceptions

None.

8.355. MOV.D

8.355.1. Purpose

Copy a D-format value without arithmetic interpretation.

8.355.2. Formats

MIPS

mov.d

Assembly

MOV.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

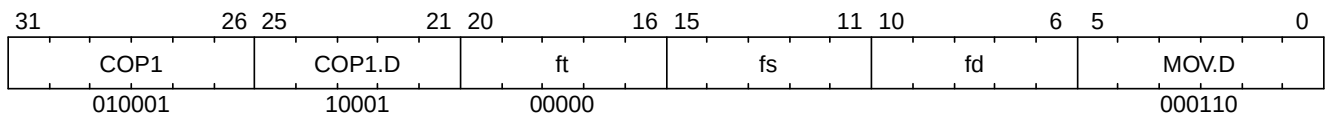


Figure 975. Encoding (32 bits)

Decode

```
return FpuUnary0perands(fd=fd, fs=fs)
```

microMIPS

mov.d

Assembly

MOV.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

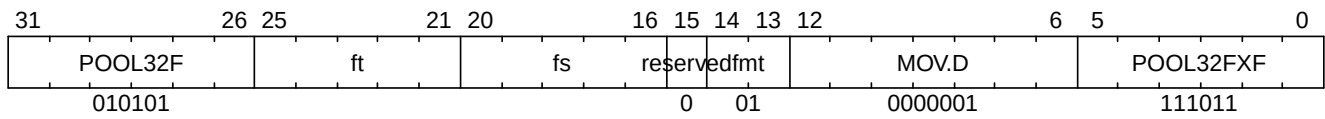


Figure 976. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

mov.d

Assembly

MOV.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

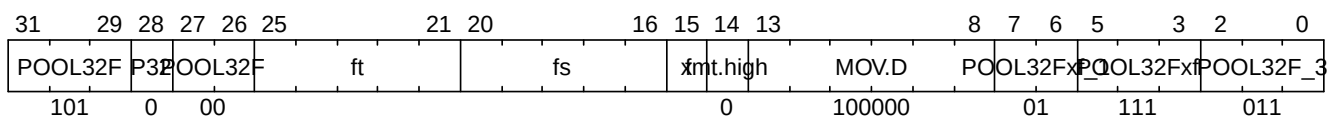


Figure 977. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.355.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "D"), "D")
```

8.355.4. Description

MOV.D validates CP1 and D-format support before updating the destination FPR.

8.355.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.356. MOV.PS

8.356.1. Purpose

Perform the paired-single mov operation.

8.356.2. Formats

MIPS

mov.ps

Assembly

MOV.PS {fd:fpr}, {fs:fpr}

Availability

AR < R6

Decode priority

0

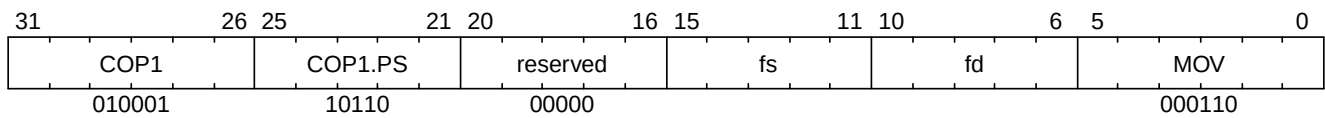


Figure 978. Encoding (32 bits)

Decode

```
return FpuPSUnaryOperands(fd=fd, fs=fs)
```

microMIPS

mov.ps

Assembly

MOV.PS {fd:fpr}, {fs:fpr}

Availability

MMAR < R6

Decode priority

0

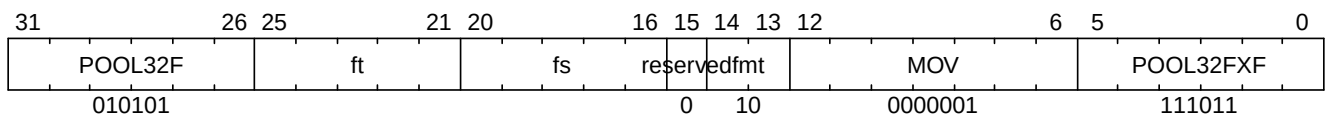


Figure 979. Encoding (32 bits)

```
return FpuPSUnaryOperands(fd=ft, fs=fs)
```

8.356.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")  
set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "PS"), "PS")
```

8.356.4. Description

MOV.PS applies independently to the upper and lower S-format lanes.

8.356.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.357. MOV.S

8.357.1. Purpose

Copy an S-format value without arithmetic interpretation.

8.357.2. Formats

MIPS

mov.s

Assembly

MOV.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

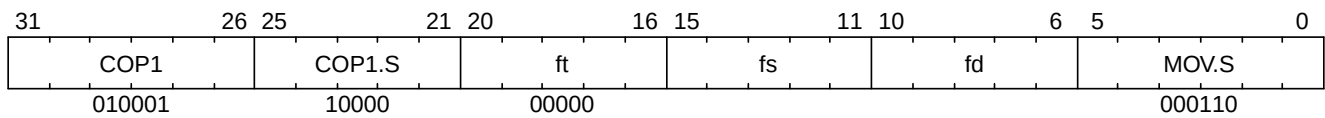


Figure 980. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

mov.s

Assembly

MOV.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

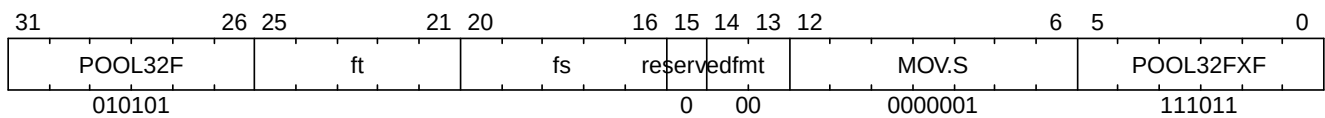


Figure 981. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

mov.s

Assembly

MOV.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

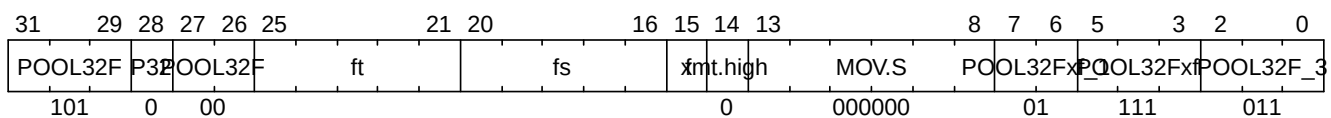


Figure 982. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.357.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "S"), "S")
```

8.357.4. Description

MOV.S validates CP1 and S-format support before updating the destination FPR.

8.357.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.358. MOVE

8.358.1. Purpose

Copy the value of `rs` to `rt`.

8.358.2. Formats

microMIPS

`move.16`

Assembly

`MOVE {rt:gpr}, {rs:gpr}`

Availability

true

Decode priority

0

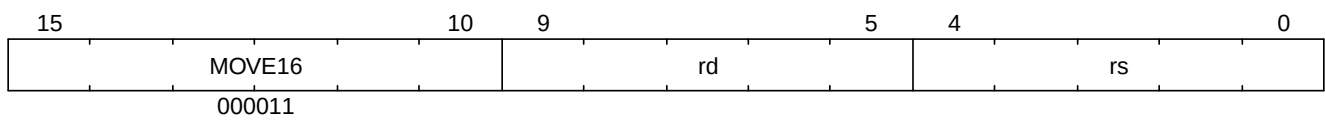


Figure 983. Encoding (16 bits)

Decode

```
return MoveOperands(rt=rd, rs=rs)
```

nanoMIPS

`move.16`

Assembly

`MOVE {rt:gpr}, {rs:gpr}`

Availability

true

Decode priority

0

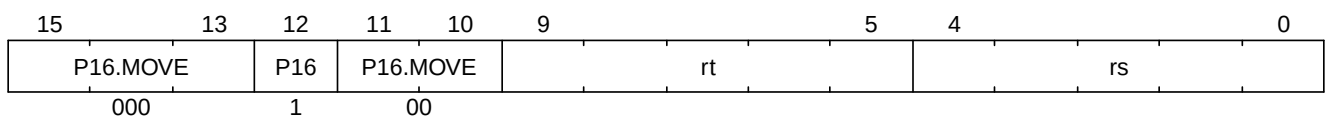


Figure 984. Encoding (16 bits)

Encoding constraints

- `rt != 0`

Decode

```
return MoveOperands(rt=rt, rs=rs)
```

8.358.3. Operation

```
value = zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
ctx.GPR[operands.rt] = value
```

8.358.4. Description

MOVE[16] copies the complete implemented-width value of the source GPR to the destination GPR. The encoded destination register must not be GPR zero.

8.358.5. Exceptions

None.

8.359. MOVE.BALC

8.359.1. Purpose

Copy a register and perform an unconditional compact PC-relative call.

8.359.2. Formats

nanoMIPS

move.balc.32

Assembly

MOVE.BALC {rd:gpr1}, {rt:gpr4}, {offset:s22}

Availability

!(NMS)

Decode priority

0

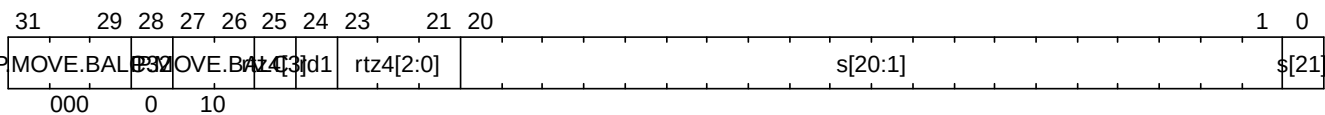


Figure 985. Encoding (32 bits)

Decode

```
exec.isCTI = True
return MoveBalcOperands(
    rd=GPR1.decode(rd1),
    rt=GPR4_ZERO.decode(rtz4),
    offset=sign_extend(s, from_nbits=22),
)
```

8.359.3. Operation

```
link_address = ctx.next_pc
address = effective_address(ctx, link_address, operands.offset)
ctx.GPR[operands.rd] = ctx.GPR[operands.rt]
compact_branch(ctx, address)
ctx.GPR[31] = link_address
```

8.359.4. Description

MOVE.BALC[32] copies rt to rd, branches relative to next_pc, and writes the sequential next_pc to GPR 31. It has no delay slot and is unavailable in NMS.

8.359.5. Exceptions

- **RI** (Reserved Instruction) — nanoMIPS register-pair forms are reserved when CP0 Config5.NMS is set.

8.360. MOVEP

8.360.1. Purpose

Copy two source GPRs to two destination GPRs.

8.360.2. Formats

microMIPS

movep.16

Assembly

MOVEP {dst1:gpr3}, {dst2:gpr3}, {src1:gpr3}, {src2:gpr3}

Availability

MMAR < R6

Decode priority

0

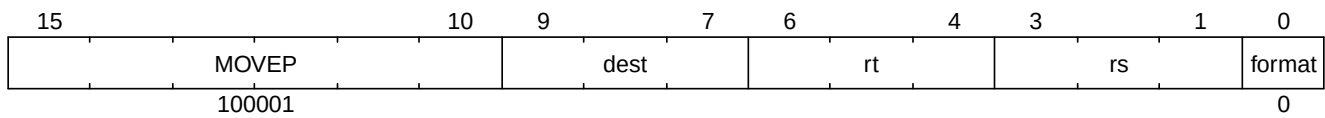


Figure 986. Encoding (16 bits)

Decode

```
destination1 = (5, 5, 6, 4, 4, 4, 4, 4)[dest]
destination2 = (6, 7, 7, 21, 22, 5, 6, 7)[dest]
source = (0, 17, 2, 3, 16, 18, 19, 20)
return MovepOperands(
    dst1=destination1,
    dst2=destination2,
    src1=source[rs],
    src2=source[rt],
)
```

movep.r6.16

Assembly

MOVEP {dst1:gpr3}, {dst2:gpr3}, {src1:gpr3}, {src2:gpr3}

Availability

MMAR >= R6

Decode priority

0

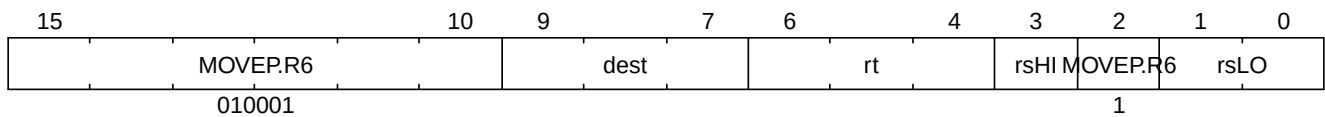


Figure 987. Encoding (16 bits)

Decode

```
source_index = (rsHI << 2) | rsLO
return MovepOperands(
    dst1=MICROMIPS_MOVEP_DEST1.decode(dest),
    dst2=MICROMIPS_MOVEP_DEST2.decode(dest),
    src1=MICROMIPS_MOVEP_SOURCE.decode(source_index),
    src2=MICROMIPS_MOVEP_SOURCE.decode(rt),
)
```

nanoMIPS

movep.16

Assembly

MOVEP {dst1:gpr2}, {dst2:gpr2}, {src1:gpr4}, {src2:gpr4}

Availability

!(NMS)

Decode priority

0

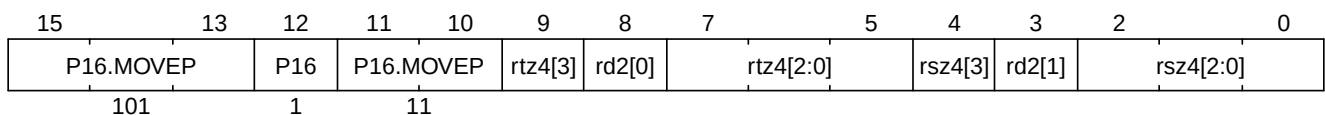


Figure 988. Encoding (16 bits)

Decode

```
return MovepOperands(
    dst1=GPR2_REG1.decode(rd2),
    dst2=GPR2_REG2.decode(rd2),
    src1=GPR4_ZERO.decode(rsz4),
    src2=GPR4_ZERO.decode(rtz4),
)
```

movep.rev

Assembly

MOVEP {dst1:gpr4}, {dst2:gpr4}, {src1:gpr2}, {src2:gpr2}

Availability

!(NMS)

Decode priority

0

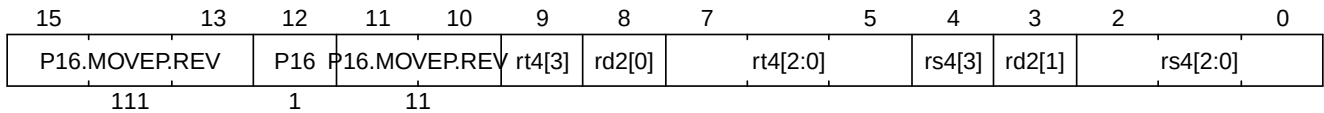


Figure 989. Encoding (16 bits)

Decode

```
return MovepOperands(  
    dst1=GPR4.decode(rs4),  
    dst2=GPR4.decode(rt4),  
    src1=GPR2_REG1.decode(rd2),  
    src2=GPR2_REG2.decode(rd2),  
)
```

8.360.3. Operation

```
value1 = ctx.GPR[operands.src1]  
value2 = ctx.GPR[operands.src2]  
if operands.dst1 in (operands.src1, operands.src2) or operands.dst2 in (  
    operands.src1,  
    operands.src2,  
):  
    # The architecture specifies UNKNOWN results for overlap. Preserve a  
    # deterministic implementation while making both destinations invalid.  
    ctx.GPR[operands.dst1] = 0  
    ctx.GPR[operands.dst2] = 0  
    return  
ctx.GPR[operands.dst1] = value1  
ctx.GPR[operands.dst2] = value2
```

8.360.4. Description

MOVEP[16] and MOVEP[REV] copy a pair of registers and are unavailable in NMS.

8.360.5. Exceptions

- **RI** (Reserved Instruction) — nanoMIPS register-pair forms are reserved when CP0 Config5.NMS is set.

8.361. MOVF

8.361.1. Purpose

Copy *rs* to *rd* when the selected floating-point condition is false.

8.361.2. Formats

MIPS

movf

Assembly

MOVF {rd:gpr}, {rs:gpr}, {cc:u3}

Availability

AR < R6

Decode priority

0

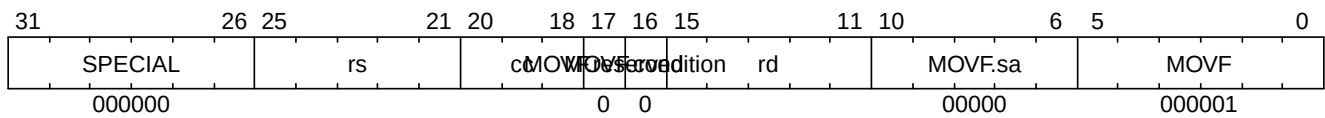


Figure 990. Encoding (32 bits)

Decode

```
return Movf0perands(rd=rd, rs=rs, cc=cc)
```

microMIPS

movf

Assembly

MOVF {rd:gpr}, {rs:gpr}, {cc:u3}

Availability

MMAR < R6

Decode priority

0

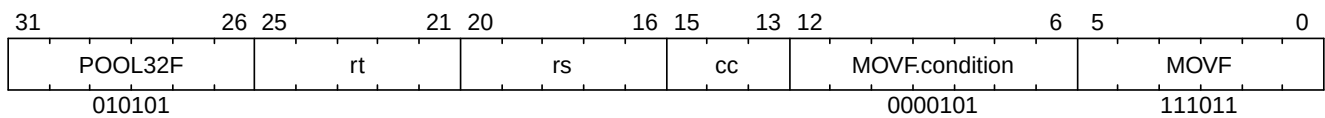


Figure 991. Encoding (32 bits)

```
return MovfOperands(rd=rt, rs=rs, cc=cc)
```

8.361.3. Operation

```
if not ctx.C0.Status.CU1:
    raise CoprocessorUnusable(1)
if not ctx.C0.Config1.FP:
    raise ReservedInstruction("Coprocessor 1 is not implemented")
if operands.cc == 0:
    condition = ctx.C1.FCSR.FCC0
elif operands.cc == 1:
    condition = ctx.C1.FCSR.FCC1
elif operands.cc == 2:
    condition = ctx.C1.FCSR.FCC2
elif operands.cc == 3:
    condition = ctx.C1.FCSR.FCC3
elif operands.cc == 4:
    condition = ctx.C1.FCSR.FCC4
elif operands.cc == 5:
    condition = ctx.C1.FCSR.FCC5
elif operands.cc == 6:
    condition = ctx.C1.FCSR.FCC6
else:
    condition = ctx.C1.FCSR.FCC7
if condition == 0:
    ctx.GPR[operands.rd] = ctx.GPR[operands.rs]
```

8.361.4. Description

MOVF uses FCC[cc] as the condition for a GPR conditional move.

8.361.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 1 is not implemented

8.362. MOVF.D

8.362.1. Purpose

Copy the D-format value from `fs` to `fd` when the selected floating-point condition is false.

8.362.2. Formats

MIPS

`movf.d`

Assembly

`MOVF.D {fd:fpr}, {fs:fpr}, {cc:u3}`

Availability

AR < R6

Decode priority

0

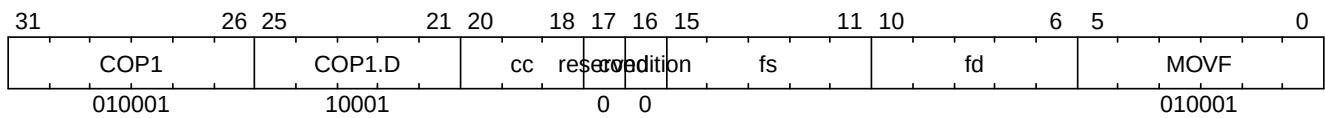


Figure 992. Encoding (32 bits)

Decode

```
return FpuConditionalOperands(fd=fd, fs=fs, cc=cc)
```

microMIPS

`movf.d.32`

Assembly

`MOVF.D {fd:fpr}, {fs:fpr}, {cc:u3}`

Availability

MMAR < R6

Decode priority

0

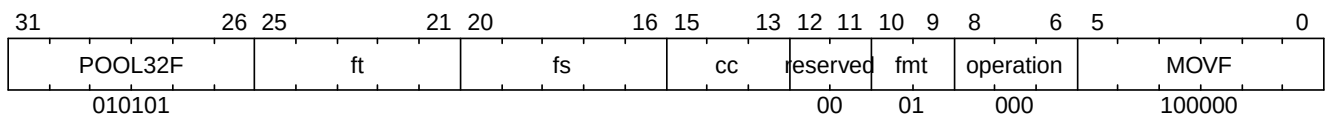


Figure 993. Encoding (32 bits)


```
return FpuConditionalOperands(fd=ft, fs=fs, cc=cc)
```

8.362.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")  
if read_fcc(ctx, operands.cc) == 0:  
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "D"), "D")
```

8.362.4. Description

MOV.F.D tests FCC[cc] and conditionally writes the D-format value to the destination FPR.

8.362.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.363. MOVF.PS

8.363.1. Purpose

Copy the PS-format value from **fs** to **fd** when the selected floating-point condition is false.

8.363.2. Formats

MIPS

movf.ps

Assembly

MOVF.PS {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

AR < R6

Decode priority

0

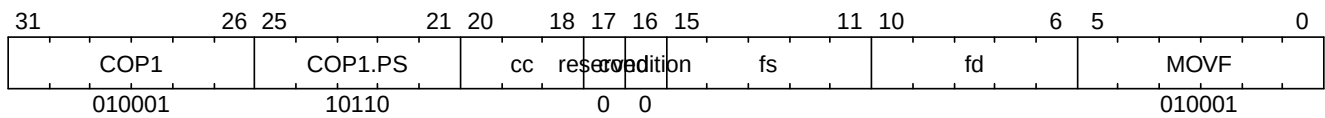


Figure 994. Encoding (32 bits)

Decode

```
return FpuConditionalOperands(fd=fd, fs=fs, cc=cc)
```

microMIPS

movf.ps.32

Assembly

MOVF.PS {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

MMAR < R6

Decode priority

0

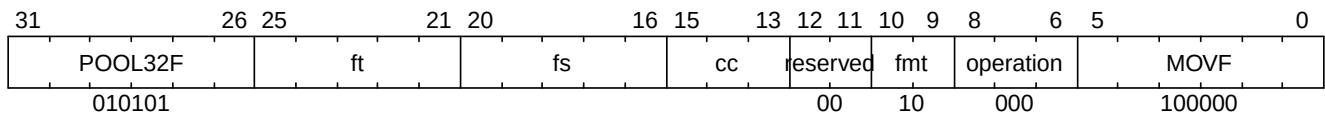


Figure 995. Encoding (32 bits)

```
return FpuConditionalOperands(fd=ft, fs=fs, cc=cc)
```

8.363.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
cc = int(operands.cc)
if cc & 1 or cc > 6:
    raise ReservedInstruction("MOV.F.PS requires an even FCC pair")
source = get_fpr(ctx, operands.fs, "PS")
destination = get_fpr(ctx, operands.fd, "PS")
if read_fcc(ctx, cc) == 0:
    destination = (destination & 0xFFFF_FFFF_0000_0000) | (source & 0xFFFF_FFFF)
if read_fcc(ctx, cc + 1) == 0:
    destination = (destination & 0x0000_0000_FFFF_FFFF) | (
        source & 0xFFFF_FFFF_0000_0000
    )
set_fpr(ctx, operands.fd, destination, "PS")
```

8.363.4. Description

MOV.F.PS tests FCC[cc] and conditionally writes the PS-format value to the destination FPR.

8.363.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; MOV.F.PS requires an even FCC pair; Coprocessor 1 or the required floating-point format is unavailable.

8.364. MOVF.S

8.364.1. Purpose

Copy the S-format value from **fs** to **fd** when the selected floating-point condition is false.

8.364.2. Formats

MIPS

movf.s

Assembly

MOVF.S {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

AR < R6

Decode priority

0

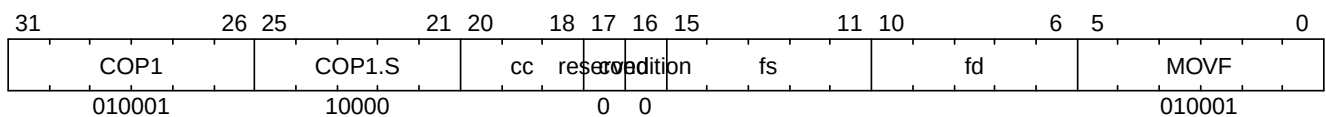


Figure 996. Encoding (32 bits)

Decode

```
return FpuConditionalOperands(fd=fd, fs=fs, cc=cc)
```

microMIPS

movf.s.32

Assembly

MOVF.S {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

MMAR < R6

Decode priority

0

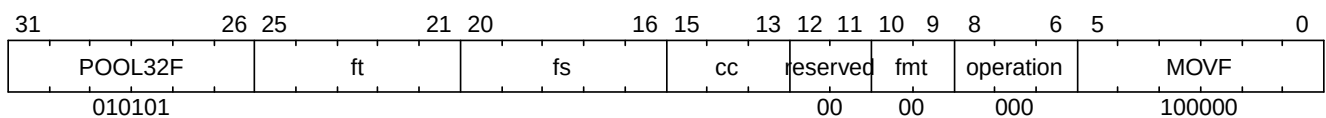


Figure 997. Encoding (32 bits)

```
return FpuConditionalOperands(fd=ft, fs=fs, cc=cc)
```

8.364.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
if read_fcc(ctx, operands.cc) == 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "S"), "S")
```

8.364.4. Description

MOV.F.S tests FCC[cc] and conditionally writes the S-format value to the destination FPR.

8.364.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.365. MOVN

8.365.1. Purpose

Copy *rs* to *rd* if the value in *rt* is not zero.

8.365.2. Formats

MIPS

movn

Assembly

MOVN {*rd:gpr*}, {*rs:gpr*}, {*rt:gpr*}

Availability

AR < R6

Decode priority

0

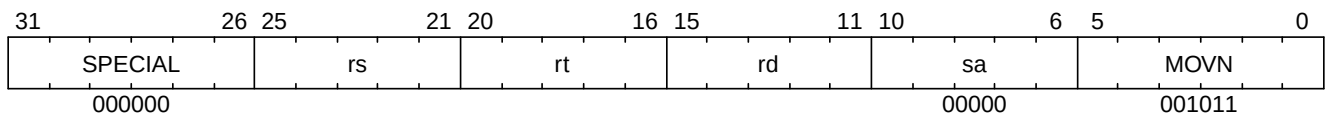


Figure 998. Encoding (32 bits)

Decode

```
return MovnOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

movn.32

Assembly

MOVN {*rd:gpr*}, {*rs:gpr*}, {*rt:gpr*}

Availability

MMAR < R6

Decode priority

0

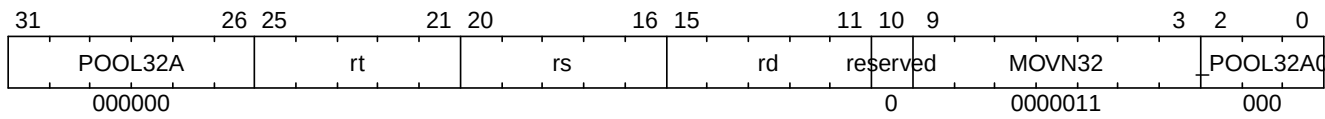


Figure 999. Encoding (32 bits)

Decode

```
return MovnOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

movn.32

Assembly

MOVN {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

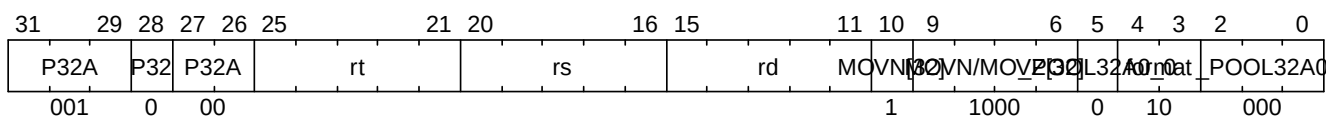


Figure 1000. Encoding (32 bits)

Decode

```
return MovnOperands(rd=rd, rs=rs, rt=rt)
```

8.365.3. Operation

```
condition = zero_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
if condition != 0:
    value = zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
    ctx.GPR[operands.rd] = value
```

8.365.4. Description

MOVN tests the complete implemented-width value in **rt**. If it is nonzero, the complete implemented-width value in **rs** is copied to **rd**; otherwise **rd** is unchanged.

8.365.5. Exceptions

None.

8.366. MOVN.D

8.366.1. Purpose

Move the D-format value conditionally on GPR *rt*.

8.366.2. Formats

MIPS

movn.d

Assembly

MOVN.D {fd:fpr}, {fs:fpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

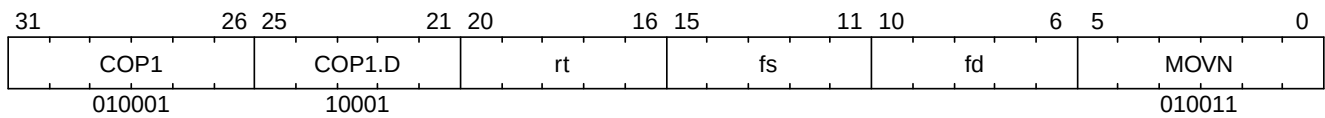


Figure 1001. Encoding (32 bits)

Decode

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

microMIPS

movn.d

Assembly

MOVN.D {fd:fpr}, {fs:fpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

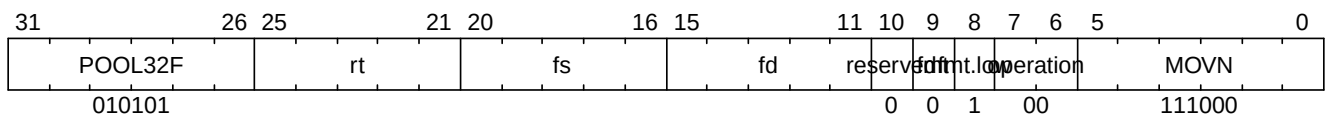


Figure 1002. Encoding (32 bits)


```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

8.366.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
if ctx.GPR[operands.rt] != 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "D"), "D")
```

8.366.4. Description

MOVN.D copies *fs* to *fd* when *rt* is nonzero.

8.366.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.367. MOVN.PS

8.367.1. Purpose

Move the PS-format value conditionally on GPR `rt`.

8.367.2. Formats

MIPS

`movn.ps`

Assembly

`MOVN.PS {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

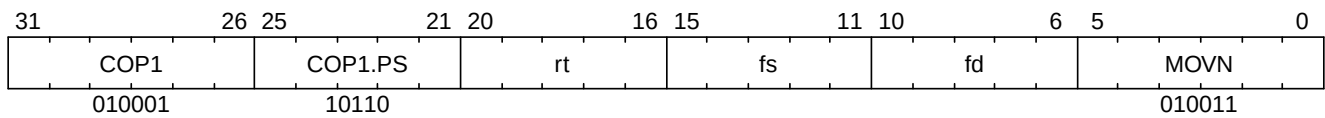


Figure 1003. Encoding (32 bits)

Decode

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

microMIPS

`movn.ps`

Assembly

`MOVN.PS {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

MMAR < R6

Decode priority

0

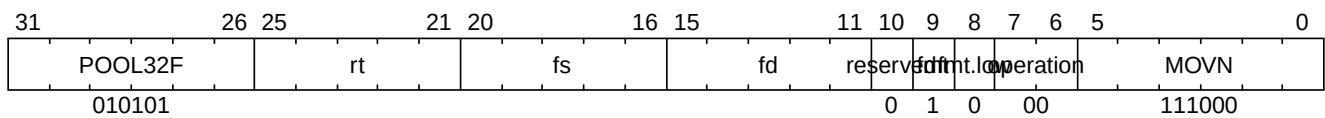


Figure 1004. Encoding (32 bits)

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

8.367.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
if ctx.GPR[operands.rt] != 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "PS"), "PS")
```

8.367.4. Description

MOVN.PS copies `fs` to `fd` when `rt` is nonzero.

8.367.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.368. MOVN.S

8.368.1. Purpose

Move the S-format value conditionally on GPR `rt`.

8.368.2. Formats

MIPS

`movn.s`

Assembly

`MOVN.S {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

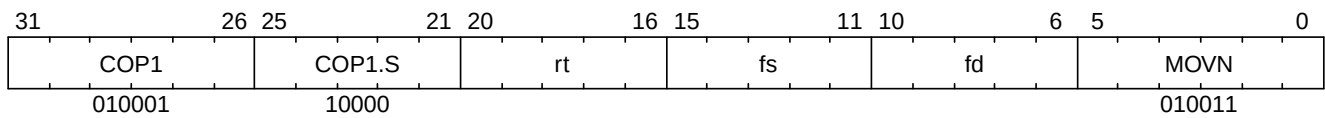


Figure 1005. Encoding (32 bits)

Decode

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

microMIPS

`movn.s`

Assembly

`MOVN.S {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

MMAR < R6

Decode priority

0

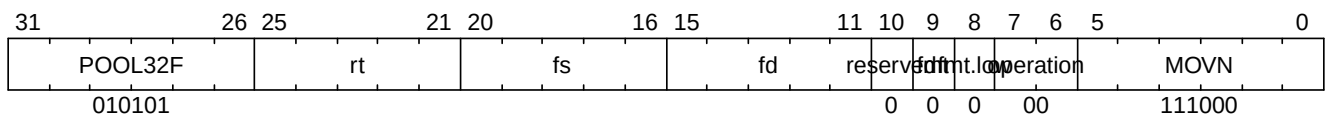


Figure 1006. Encoding (32 bits)

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

8.368.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
if ctx.GPR[operands.rt] != 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "S"), "S")
```

8.368.4. Description

MOVN.S copies `fs` to `fd` when `rt` is nonzero.

8.368.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.369. MOVT

8.369.1. Purpose

Copy *rs* to *rd* when the selected floating-point condition is true.

8.369.2. Formats

MIPS

movt

Assembly

MOVT {rd:gpr}, {rs:gpr}, {cc:u3}

Availability

AR < R6

Decode priority

0

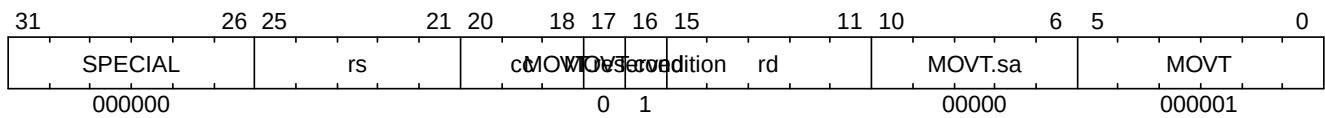


Figure 1007. Encoding (32 bits)

Decode

```
return Movf0perands(rd=rd, rs=rs, cc=cc)
```

microMIPS

movt

Assembly

MOVT {rd:gpr}, {rs:gpr}, {cc:u3}

Availability

MMAR < R6

Decode priority

0

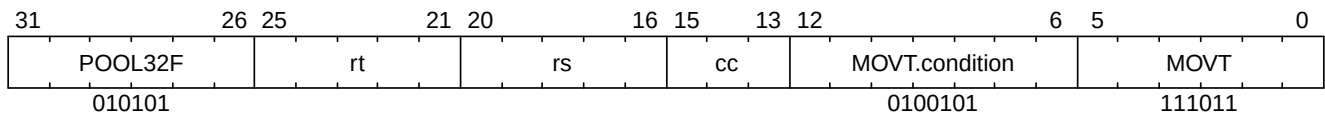


Figure 1008. Encoding (32 bits)

```
return MovfOperands(rd=rt, rs=rs, cc=cc)
```

8.369.3. Operation

```
if not ctx.C0.Status.CU1:
    raise CoprocessorUnusable(1)
if not ctx.C0.Config1.FP:
    raise ReservedInstruction("Coprocessor 1 is not implemented")
if operands.cc == 0:
    condition = ctx.C1.FCSR.FCC0
elif operands.cc == 1:
    condition = ctx.C1.FCSR.FCC1
elif operands.cc == 2:
    condition = ctx.C1.FCSR.FCC2
elif operands.cc == 3:
    condition = ctx.C1.FCSR.FCC3
elif operands.cc == 4:
    condition = ctx.C1.FCSR.FCC4
elif operands.cc == 5:
    condition = ctx.C1.FCSR.FCC5
elif operands.cc == 6:
    condition = ctx.C1.FCSR.FCC6
else:
    condition = ctx.C1.FCSR.FCC7
if condition != 0:
    ctx.GPR[operands.rd] = ctx.GPR[operands.rs]
```

8.369.4. Description

MOVT uses FCC[cc] as the condition for a GPR conditional move.

8.369.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 1 is not implemented

8.370. MOVT.D

8.370.1. Purpose

Copy the D-format value from `fs` to `fd` when the selected floating-point condition is true.

8.370.2. Formats

MIPS

`movt.d`

Assembly

`MOVT.D {fd:fpr}, {fs:fpr}, {cc:u3}`

Availability

AR < R6

Decode priority

0

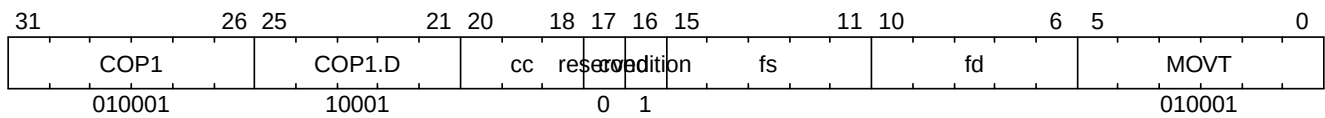


Figure 1009. Encoding (32 bits)

Decode

```
return FpuConditionalOperands(fd=fd, fs=fs, cc=cc)
```

microMIPS

`movt.d.32`

Assembly

`MOVT.D {fd:fpr}, {fs:fpr}, {cc:u3}`

Availability

MMAR < R6

Decode priority

0

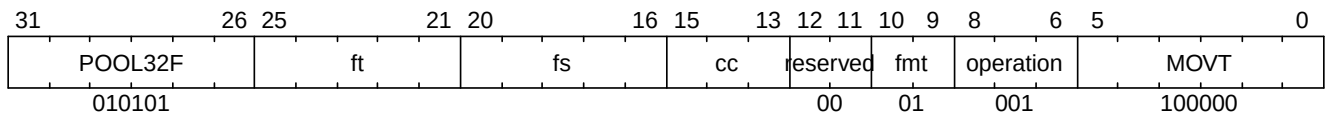


Figure 1010. Encoding (32 bits)


```
return FpuConditionalOperands(fd=ft, fs=fs, cc=cc)
```

8.370.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
if read_fcc(ctx, operands.cc) != 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "D"), "D")
```

8.370.4. Description

MOVT.D tests FCC[cc] and conditionally writes the D-format value to the destination FPR.

8.370.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.371. MOVT.PS

8.371.1. Purpose

Copy the PS-format value from **fs** to **fd** when the selected floating-point condition is true.

8.371.2. Formats

MIPS

movt.ps

Assembly

MOVT.PS {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

AR < R6

Decode priority

0

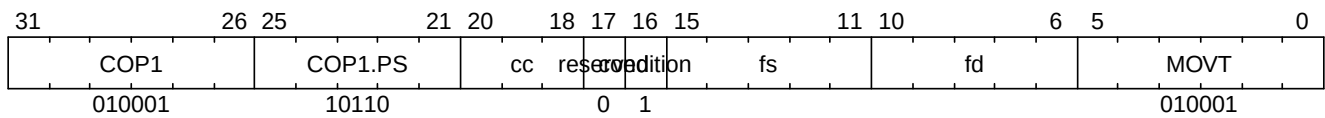


Figure 1011. Encoding (32 bits)

Decode

```
return FpuConditionalOperands(fd=fd, fs=fs, cc=cc)
```

microMIPS

movt.ps.32

Assembly

MOVT.PS {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

MMAR < R6

Decode priority

0

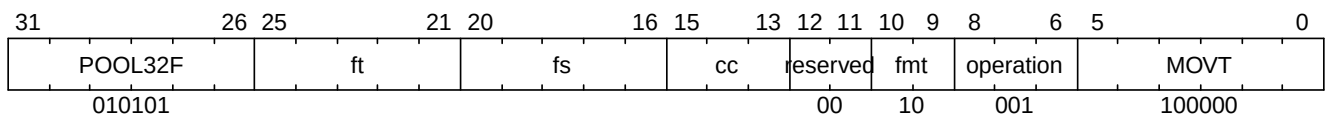


Figure 1012. Encoding (32 bits)

```
return FpuConditionalOperands(fd=ft, fs=fs, cc=cc)
```

8.371.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
cc = int(operands.cc)
if cc & 1 or cc > 6:
    raise ReservedInstruction("MOVT.PS requires an even FCC pair")
source = get_fpr(ctx, operands.fs, "PS")
destination = get_fpr(ctx, operands.fd, "PS")
if read_fcc(ctx, cc) != 0:
    destination = (destination & 0xFFFF_FFFF_0000_0000) | (source & 0xFFFF_FFFF)
if read_fcc(ctx, cc + 1) != 0:
    destination = (destination & 0x0000_0000_FFFF_FFFF) | (
        source & 0xFFFF_FFFF_0000_0000
    )
set_fpr(ctx, operands.fd, destination, "PS")
```

8.371.4. Description

MOVT.PS tests FCC[cc] and conditionally writes the PS-format value to the destination FPR.

8.371.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; MOVT.PS requires an even FCC pair; Coprocessor 1 or the required floating-point format is unavailable.

8.372. MOVT.S

8.372.1. Purpose

Copy the S-format value from **fs** to **fd** when the selected floating-point condition is true.

8.372.2. Formats

MIPS

movt.s

Assembly

MOVT.S {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

AR < R6

Decode priority

0

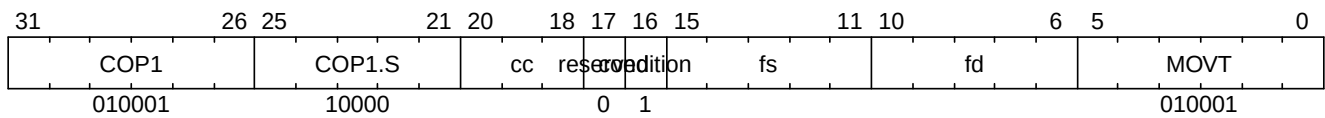


Figure 1013. Encoding (32 bits)

Decode

```
return FpuConditionalOperands(fd=fd, fs=fs, cc=cc)
```

microMIPS

movt.s.32

Assembly

MOVT.S {fd:fpr}, {fs:fpr}, {cc:u3}

Availability

MMAR < R6

Decode priority

0

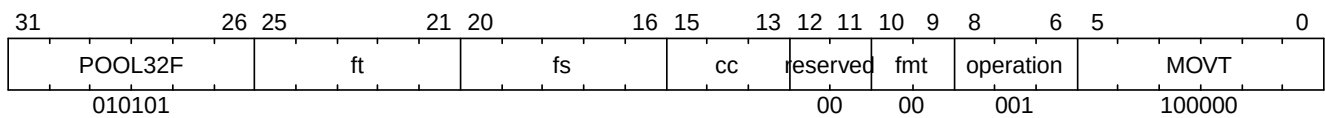


Figure 1014. Encoding (32 bits)

```
return FpuConditionalOperands(fd=ft, fs=fs, cc=cc)
```

8.372.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
if read_fcc(ctx, operands.cc) != 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "S"), "S")
```

8.372.4. Description

MOVT.S tests FCC[cc] and conditionally writes the S-format value to the destination FPR.

8.372.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.373. MOVZ

8.373.1. Purpose

Copy *rs* to *rd* if the value in *rt* is zero.

8.373.2. Formats

MIPS

movz

Assembly

MOVZ {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

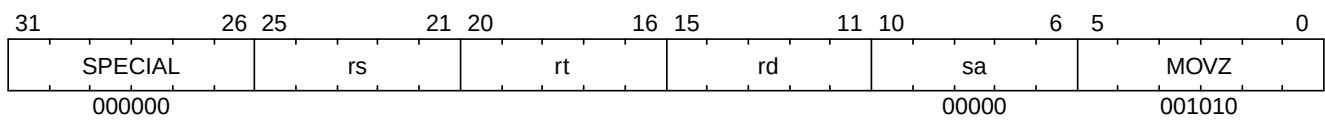


Figure 1015. Encoding (32 bits)

Decode

```
return MovzOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

movz.32

Assembly

MOVZ {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

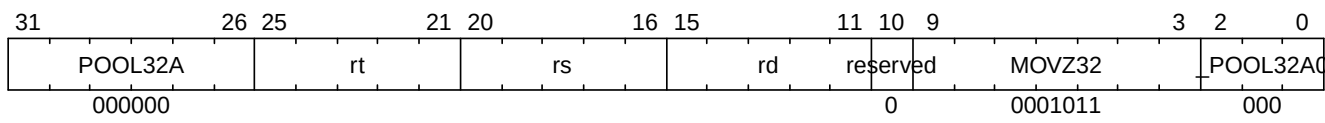


Figure 1016. Encoding (32 bits)

Decode

```
return MovzOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

movz.32

Assembly

MOVZ {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

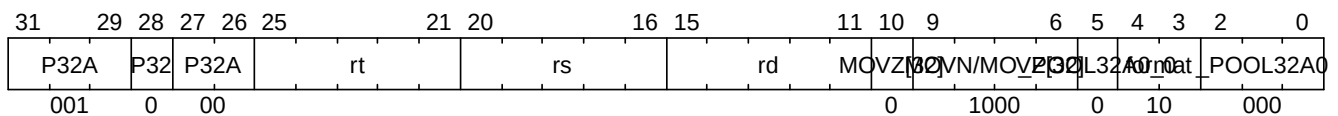


Figure 1017. Encoding (32 bits)

Decode

```
return MovzOperands(rd=rd, rs=rs, rt=rt)
```

8.373.3. Operation

```
condition = zero_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
if condition == 0:
    value = zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
    ctx.GPR[operands.rd] = value
```

8.373.4. Description

MOVZ tests the complete implemented-width value in **rt**. If it is zero, the complete implemented-width value in **rs** is copied to **rd**; otherwise **rd** is unchanged.

8.373.5. Exceptions

None.

8.374. MOVZ.D

8.374.1. Purpose

Move the D-format value conditionally on GPR *rt*.

8.374.2. Formats

MIPS

movz.d

Assembly

MOVZ.D {fd:fpr}, {fs:fpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

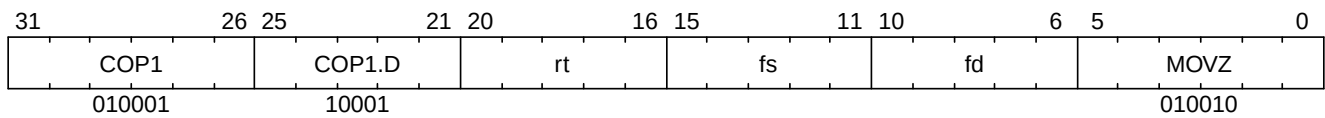


Figure 1018. Encoding (32 bits)

Decode

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

microMIPS

movz.d

Assembly

MOVZ.D {fd:fpr}, {fs:fpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

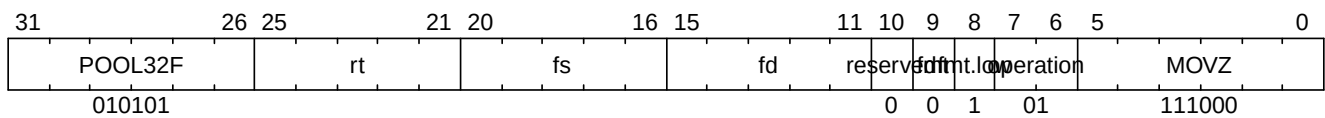


Figure 1019. Encoding (32 bits)


```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

8.374.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
if ctx.GPR[operands.rt] == 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "D"), "D")
```

8.374.4. Description

MOVZ.D copies `fs` to `fd` when `rt` is zero.

8.374.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.375. MOVZ.PS

8.375.1. Purpose

Move the PS-format value conditionally on GPR `rt`.

8.375.2. Formats

MIPS

`movz.ps`

Assembly

`MOVZ.PS {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

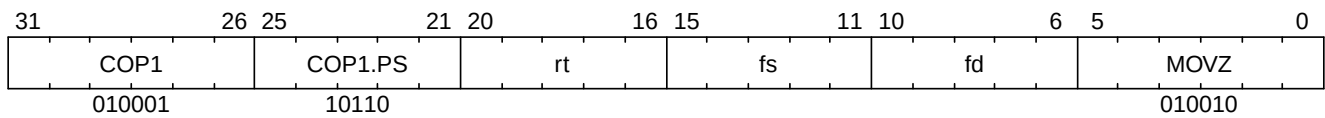


Figure 1020. Encoding (32 bits)

Decode

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

microMIPS

`movz.ps`

Assembly

`MOVZ.PS {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

MMAR < R6

Decode priority

0

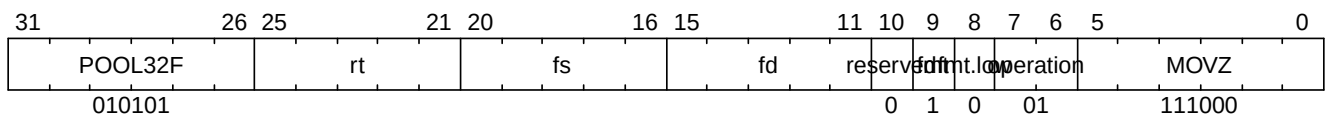


Figure 1021. Encoding (32 bits)

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

8.375.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
if ctx.GPR[operands.rt] == 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "PS"), "PS")
```

8.375.4. Description

MOVZ.PS copies *fs* to *fd* when *rt* is zero.

8.375.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.376. MOVZ.S

8.376.1. Purpose

Move the S-format value conditionally on GPR `rt`.

8.376.2. Formats

MIPS

`MOVZ.S`

Assembly

`MOVZ.S {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

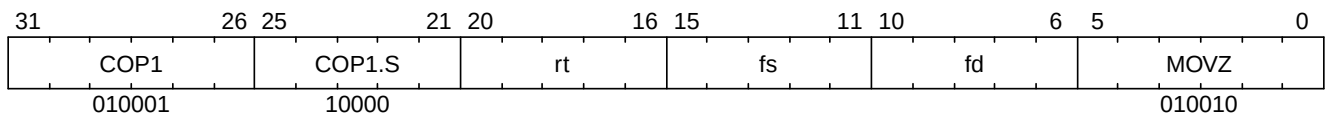


Figure 1022. Encoding (32 bits)

Decode

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

microMIPS

`MOVZ.S`

Assembly

`MOVZ.S {fd:fpr}, {fs:fpr}, {rt:gpr}`

Availability

MMAR < R6

Decode priority

0

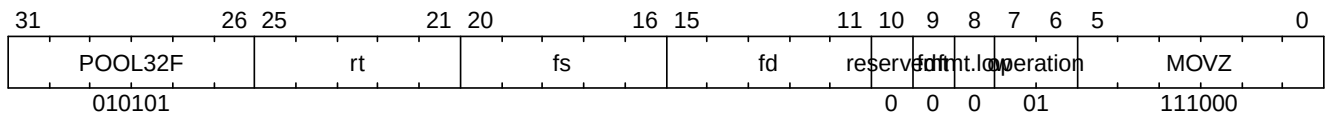


Figure 1023. Encoding (32 bits)

```
return FpuGprConditionalOperands(fd=fd, fs=fs, rt=rt)
```

8.376.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
if ctx.GPR[operands.rt] == 0:
    set_fpr(ctx, operands.fd, get_fpr(ctx, operands.fs, "S"), "S")
```

8.376.4. Description

MOVZ.S copies *fs* to *fd* when *rt* is zero.

8.376.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.377. MSUB

8.377.1. Purpose

MSUB a 32-bit product with the classic HI/LO accumulator.

8.377.2. Formats

MIPS

`msub`

Assembly

`MSUB {rs:gpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

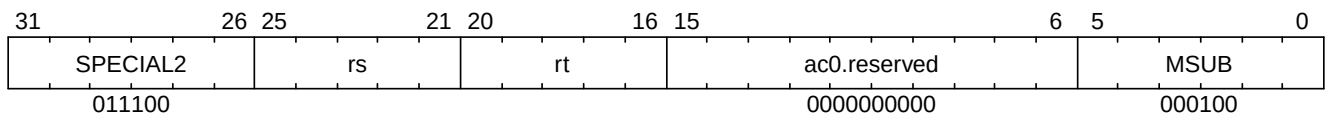


Figure 1024. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

`msub.dsp`

Assembly

`MSUB {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

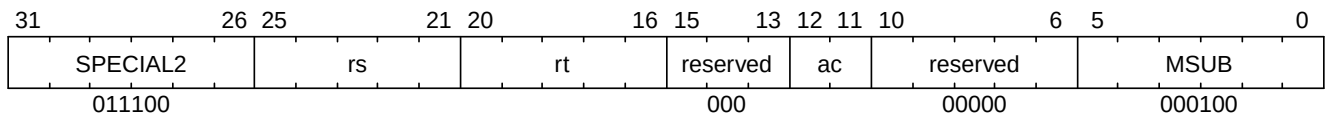


Figure 1025. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

microMIPS

msub.32

Assembly

MSUB {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

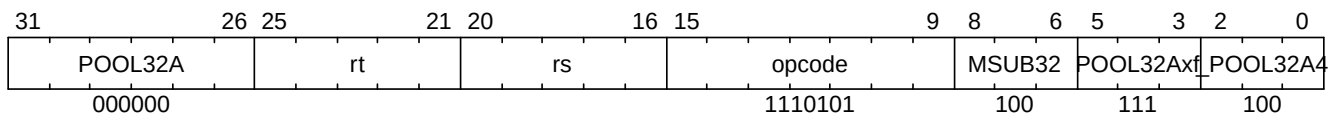


Figure 1026. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

msub.dsp.32

Assembly

MSUB {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

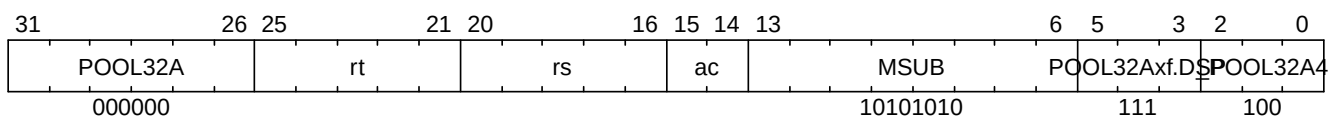


Figure 1027. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

nanoMIPS

msub.dsp.r7

Assembly

MSUB {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

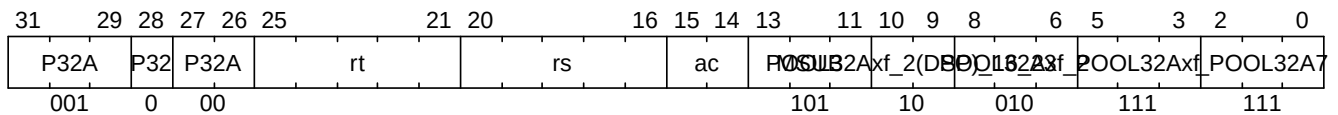


Figure 1028. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

8.377.3. Operation

```
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
write_selected_accumulator(
    ctx,
    operands.ac,
    selected_accumulator_value(ctx, operands.ac)
    - classic_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False),
)
```

8.377.4. Description

The selected signed or unsigned 32-bit product is added to or subtracted from the 64-bit HI:LO value modulo 2^{64} . The base encoding selects accumulator zero.

8.377.5. Exceptions

None.

8.378. MSUB.D

8.378.1. Purpose

Compute the pre-Release-6 msub of D-format values using the explicit accumulator **fr**.

8.378.2. Formats

MIPS

msub.d

Assembly

MSUB.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

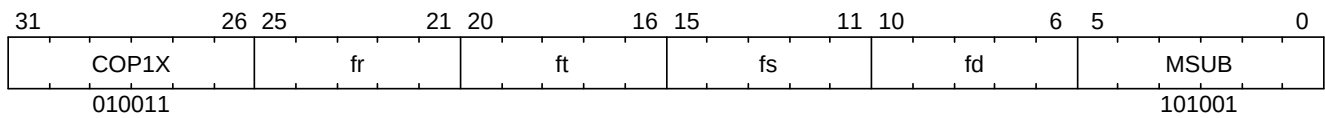


Figure 1029. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

msub.d

Assembly

MSUB.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

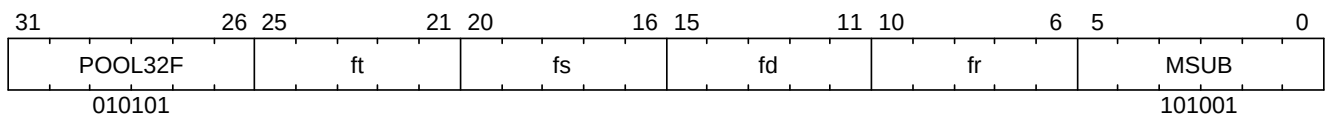


Figure 1030. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.378.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_mul_add(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "D"),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    subtract=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.378.4. Description

MSUB.D combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.378.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.379. MSUB.PS

8.379.1. Purpose

Compute the paired-single msub operation.

8.379.2. Formats

MIPS

`msub.ps`

Assembly

`MSUB.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR < R6

Decode priority

0

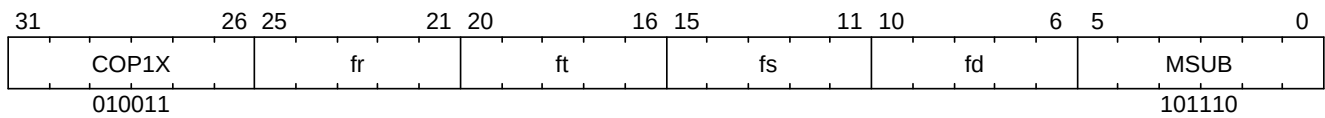


Figure 1031. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

`msub.ps`

Assembly

`MSUB.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR < R6

Decode priority

0

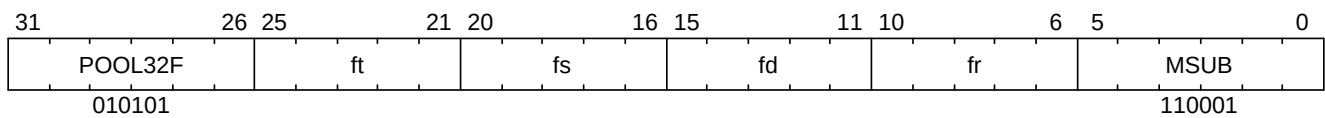


Figure 1032. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.379.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
fr = get_fpr(ctx, operands.fr, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
low, low_flags = ieee_mul_add(
    "S",
    rounding,
    fr & 0xFFFF_FFFF,
    fs & 0xFFFF_FFFF,
    ft & 0xFFFF_FFFF,
    subtract=True,
)
high, high_flags = ieee_mul_add(
    "S", rounding, fr >> 32, fs >> 32, ft >> 32, subtract=True
)
write_fpu_result(ctx, operands.fd, low | (high << 32), low_flags | high_flags, "PS")
```

8.379.4. Description

MSUB.PS applies independently to the upper and lower S-format lanes.

8.379.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.380. MSUB.S

8.380.1. Purpose

Compute the pre-Release-6 msub of S-format values using the explicit accumulator **fr**.

8.380.2. Formats

MIPS

msub.s

Assembly

MSUB.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

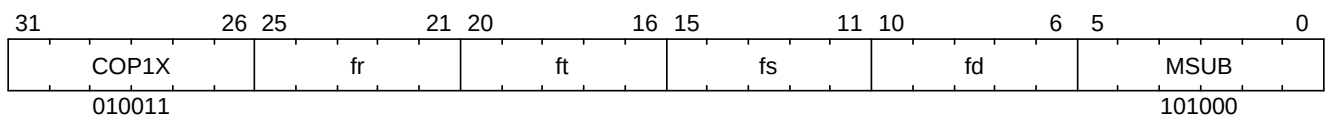


Figure 1033. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

msub.s

Assembly

MSUB.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

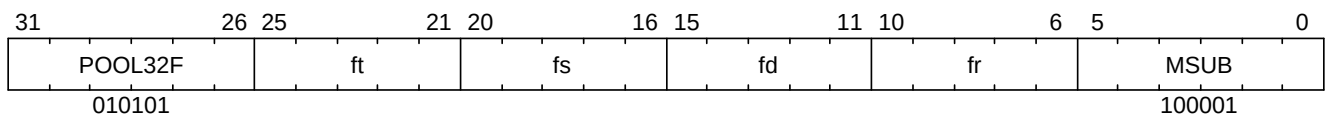


Figure 1034. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.380.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_mul_add(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "S"),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    subtract=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.380.4. Description

MSUB.S combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.380.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.381. MSUBF.D

8.381.1. Purpose

Fused multiply-subtract D-format values.

8.381.2. Formats

MIPS

`msubf.d`

Assembly

`MSUBF.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

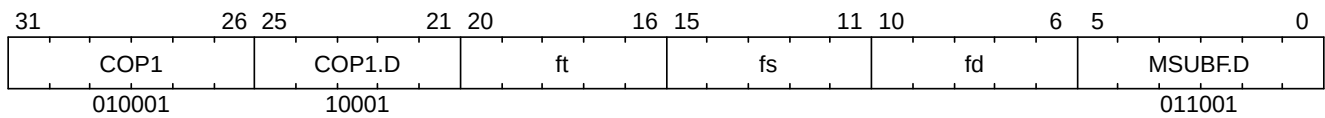


Figure 1035. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`msubf.d`

Assembly

`MSUBF.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

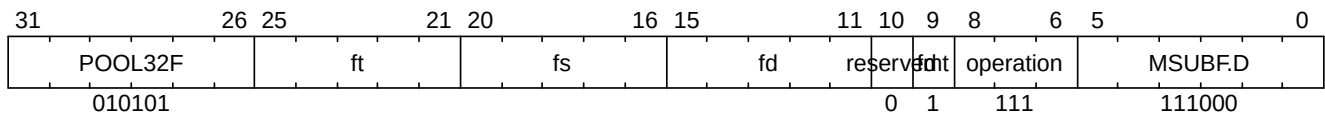


Figure 1036. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

`msubf.d`

Assembly

`MSUBF.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

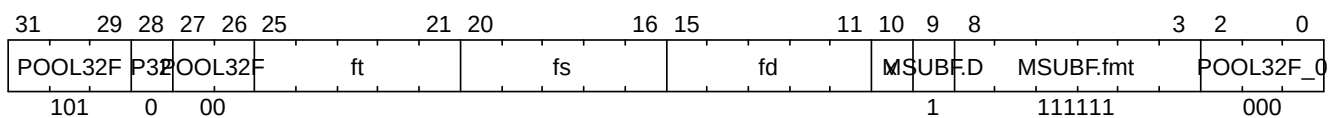


Figure 1037. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.381.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_madd(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fd, "D"),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    subtract=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.381.4. Description

MSUBF.D validates CP1 and D-format support before updating the destination FPR.

8.381.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.382. MSUBF.S

8.382.1. Purpose

Fused multiply-subtract S-format values.

8.382.2. Formats

MIPS

`msubf.s`

Assembly

`MSUBF.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

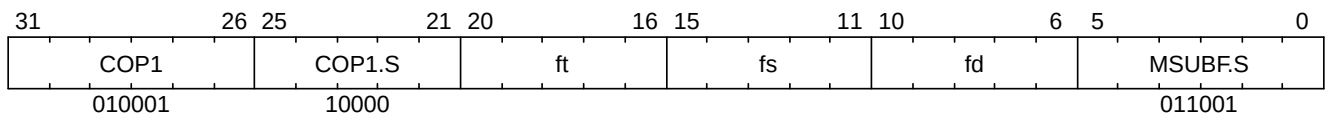


Figure 1038. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`msubf.s`

Assembly

`MSUBF.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

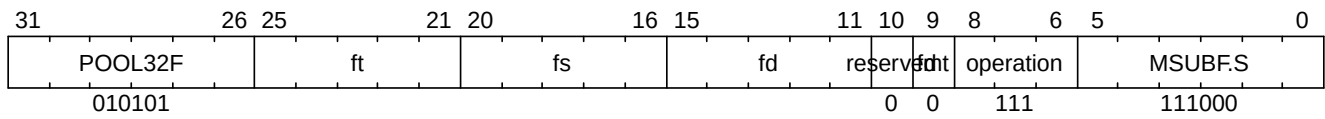


Figure 1039. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

`msubf.s`

Assembly

`MSUBF.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

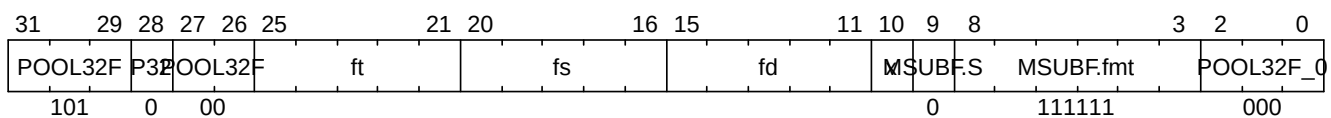


Figure 1040. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.382.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_madd(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fd, "S"),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    subtract=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.382.4. Description

MSUBF.S validates CP1 and S-format support before updating the destination FPR.

8.382.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.383. MSUBU

8.383.1. Purpose

MSUBU a 32-bit product with the classic HI/LO accumulator.

8.383.2. Formats

MIPS

`msubu`

Assembly

`MSUBU {rs:gpr}, {rt:gpr}`

Availability

AR < R6

Decode priority

0

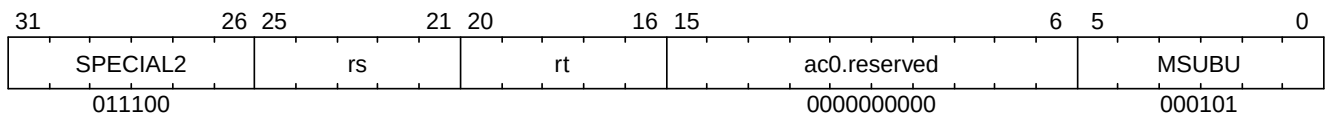


Figure 1041. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

`msubu.dsp`

Assembly

`MSUBU {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

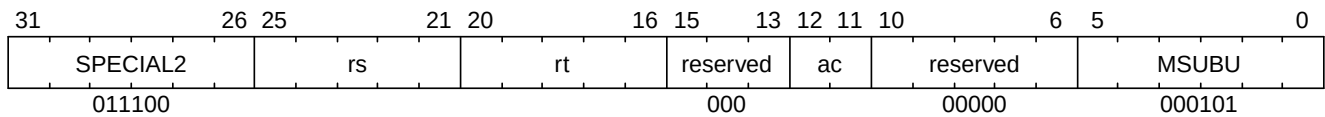


Figure 1042. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

microMIPS

msubu.32

Assembly

MSUBU {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

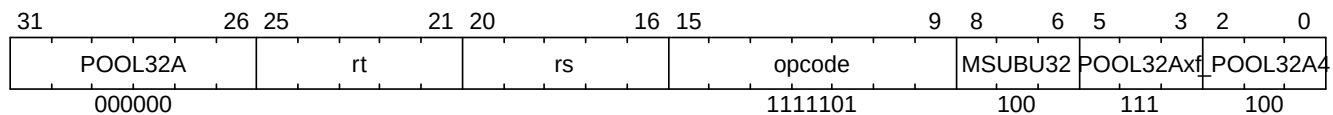


Figure 1043. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=None)
```

msubu.dsp.32

Assembly

MSUBU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

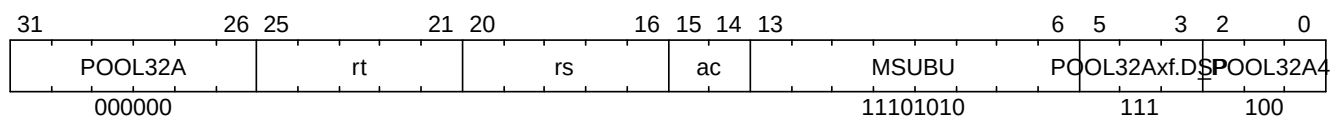


Figure 1044. Encoding (32 bits)

Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

nanoMIPS

msubu.dsp.r7

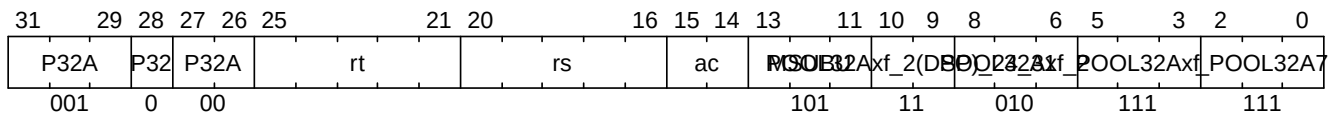
Assembly

MSUBU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

0



Decode

```
return MultiplyAccumulateOperands(rs=rs, rt=rt, ac=ac)
```

8.383.3. Operation

```

if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
write_selected_accumulator(
    ctx,
    operands.ac,
    selected_accumulator_value(ctx, operands.ac)
    - classic_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True),
)

```

8.383.4. Description

The selected signed or unsigned 32-bit product is added to or subtracted from the 64-bit HI:LO value modulo 2^{64} . The base encoding selects accumulator zero.

8.383.5. Exceptions

None.

8.384. MTC0

8.384.1. Purpose

Write the low 32 bits of a GPR to an encoded CP0 register.

8.384.2. Formats

MIPS

mtc0

Assembly

MTC0 {rt:gpr}, {c0s:u5}, {sel:u3}

Availability

true

Decode priority

0

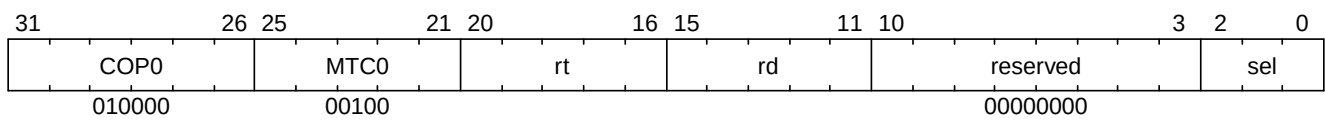


Figure 1046. Encoding (32 bits)

Decode

```
return Mtc0operands(rt=rt, c0s=rd, sel=zero_extend(sel, from_nbits=3))
```

microMIPS

mtc0.32

Assembly

MTC0 {rt:gpr}, {c0s:u5}, {sel:u3}

Availability

true

Decode priority

0

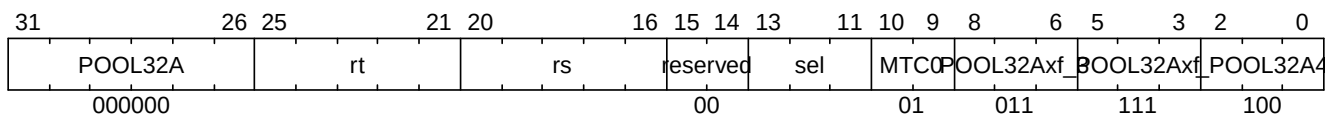


Figure 1047. Encoding (32 bits)

Decode

```
return Mtc0Operands(rt=rt, c0s=rs, sel=zero_extend(sel, from_nbits=3))
```

nanoMIPS

mtc0.32

Assembly

MTC0 {rt:gpr}, {c0s:u5}, {sel:u5}

Availability

true

Decode priority

0

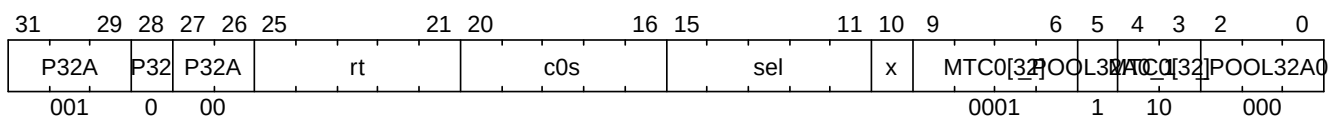


Figure 1048. Encoding (32 bits)

Decode

```
return Mtc0Operands(rt=rt, c0s=c0s, sel=sel)
```

8.384.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
nbits = 64 if ctx.GPR.width >= 64 else 32
write_cp0_register(
    ctx, ctx.GPR[operands.rt], operands.c0s, operands.sel, nbits=nbits
)
```

8.384.4. Description

MTC0 resolves (*c0s*, *sel*) through the active CP0 register bank. Writes to unused registers and read-only fields are ignored, while the CP0 privilege check remains architectural.

8.384.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.385. MTC1

8.385.1. Purpose

Transfer a value through Coprocessor 1.

8.385.2. Formats

MIPS

mtc1

Assembly

MTC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

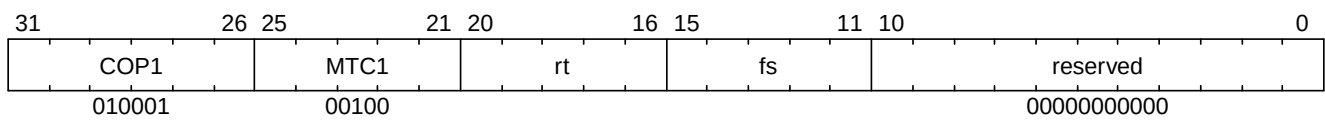


Figure 1049. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

microMIPS

mtc1.32

Assembly

MTC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

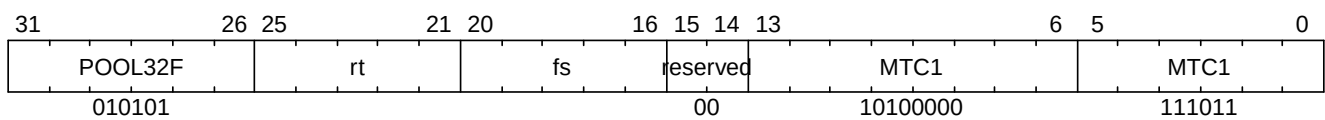


Figure 1050. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

nanoMIPS

mtc1.32

Assembly

MTC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

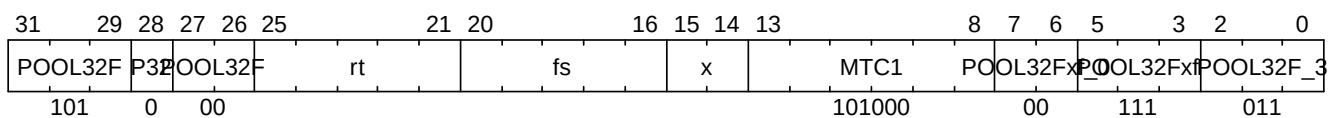


Figure 1051. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

8.385.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")  
set_fpr(ctx, operands.fs, ctx.GPR[operands.rt], "32")
```

8.385.4. Description

MTC1 performs its CP1 access checks before transferring data.

8.385.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.

8.386. MTC2

8.386.1. Purpose

Move a GPR to a CP2 register.

8.386.2. Formats

microMIPS

mtc2.32

Assembly

MTC2 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

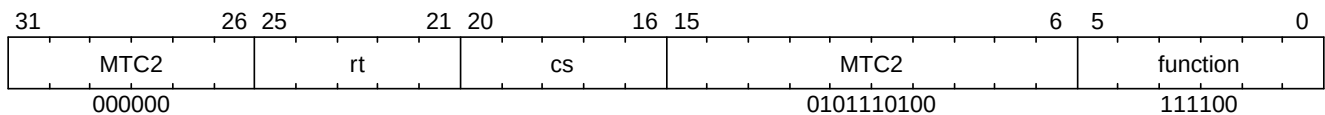


Figure 1052. Encoding (32 bits)

Decode

```
return Cop2MoveOperands(rt=rt, cs=cs)
```

8.386.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("MTC2 execution is not implemented")
```

8.386.4. Description

MTC2 checks CP2 availability before accessing the register.

8.386.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; MTC2 execution is not

implemented

8.387. MTHC0

8.387.1. Purpose

Write high CP0 register bits.

8.387.2. Formats

MIPS

`mthc0`

Assembly

`MTHC0 {rt:gpr}, {c0s:u5}, {sel:u3}`

Availability

AT >= MIPS64_32 || MVH

Decode priority

0

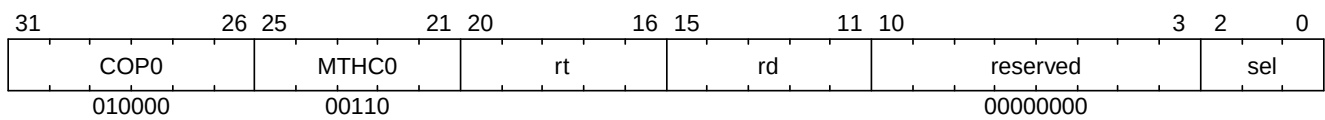


Figure 1053. Encoding (32 bits)

Decode

```
return Mthc0operands(rt=rt, c0s=rd, sel=zero_extend(sel, from_nbits=3))
```

microMIPS

`mthc0.32`

Assembly

`MTHC0 {rt:gpr}, {c0s:u5}, {sel:u3}`

Availability

AT >= MIPS64_32 || MVH

Decode priority

0

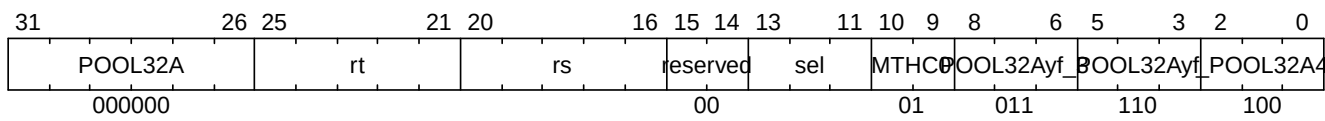


Figure 1054. Encoding (32 bits)

Decode

```
return Mthc0Operands(rt=rt, c0s=rs, sel=zero_extend(sel, from_nbits=3))
```

nanoMIPS

mthc0.32

Assembly

MTHC0 {rt:gpr}, {c0s:u5}, {sel:u5}

Availability

true

Decode priority

0

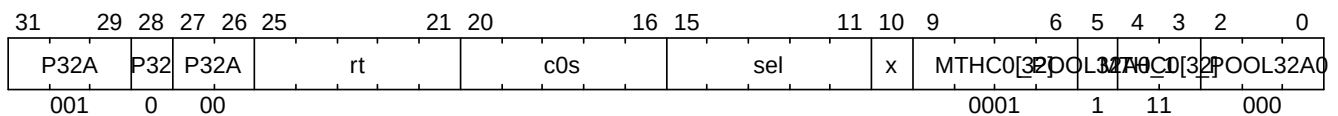


Figure 1055. Encoding (32 bits)

Decode

```
return Mthc0Operands(rt=rt, c0s=c0s, sel=sel)
```

8.387.3. Operation

```
if not ctx.C0.Config5.MVH:
    raise ReservedInstruction("MTHC0 requires MVH")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
write_cp0_register_high(ctx, ctx.GPR[operands.rt], operands.c0s, operands.sel)
```

8.387.4. Description

MTHC0 writes bits 63:32 of an extended CP0 register.

8.387.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — MTHC0 requires MVH

8.388. MTHC1

8.388.1. Purpose

Transfer a value through Coprocessor 1.

8.388.2. Formats

MIPS

`mthc1`

Assembly

`MTHC1 {rt:gpr}, {fs:fpr}`

Availability

AR >= R2_3_5 && (AT >= MIPS64_32 || FPU64)

Decode priority

0

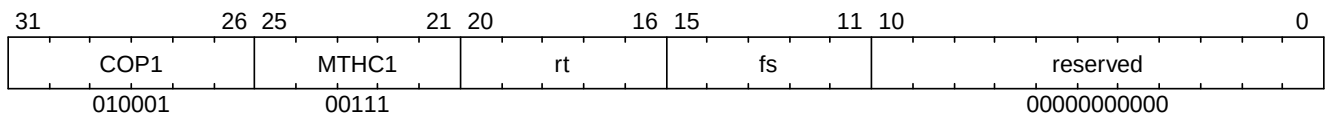


Figure 1056. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

microMIPS

`mthc1.32`

Assembly

`MTHC1 {rt:gpr}, {fs:fpr}`

Availability

AT >= MIPS64_32 || FPU64

Decode priority

0

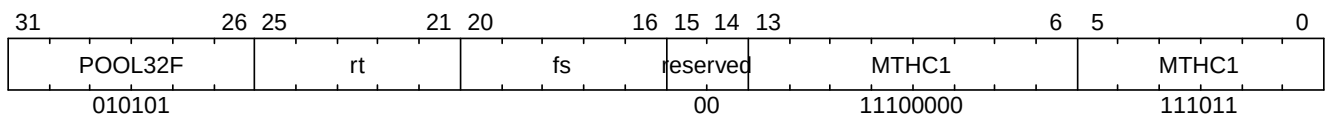


Figure 1057. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

nanoMIPS

mthc1.32

Assembly

MTHC1 {rt:gpr}, {fs:fpr}

Availability

true

Decode priority

0

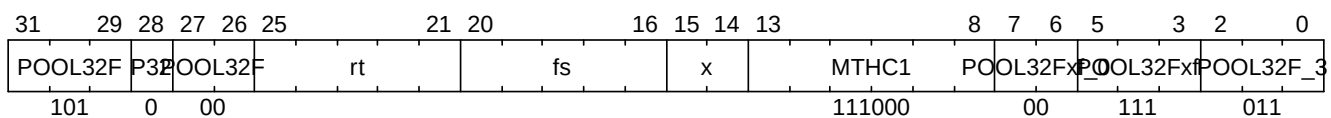


Figure 1058. Encoding (32 bits)

Decode

```
return Cp1MoveOperands(rt=rt, fs=fs)
```

8.388.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
value = get_fpr(ctx, operands.fs, "64")
set_fpr(
    ctx,
    operands.fs,
    (value & 0xFFFF_FFFF) | ((ctx.GPR[operands.rt] & 0xFFFF_FFFF) << 32),
    "64",
)
```

8.388.4. Description

MTHC1 performs its CP1 access checks before transferring data.

8.388.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit

operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.

8.389. MTHC2

8.389.1. Purpose

Move a GPR to the high half of a CP2 register.

8.389.2. Formats

microMIPS

mthc2.32

Assembly

MTHC2 {rt:gpr}, {cs:u5}

Availability

true

Decode priority

0

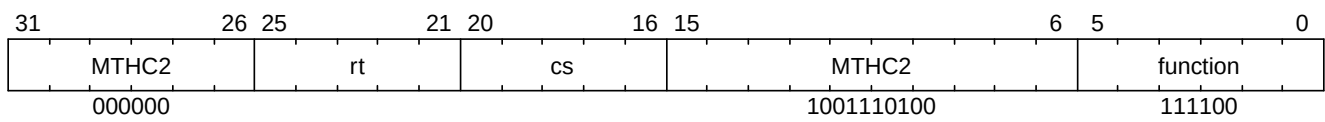


Figure 1059. Encoding (32 bits)

Decode

```
return Cop2MoveOperands(rt=rt, cs=cs)
```

8.389.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("MTHC2 execution is not implemented")
```

8.389.4. Description

MTHC2 checks CP2 availability before accessing the register.

8.389.5. Exceptions

- CPU (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- RI (Reserved Instruction) — Coprocessor 2 is not implemented; MTHC2 execution is not

implemented

8.390. MTHI

8.390.1. Purpose

Copy a GPR value to the classic multiply/divide HI register.

8.390.2. Formats

MIPS

`mthi.dsp`

Assembly

`MTHI {rs:gpr}, {ac:u2}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

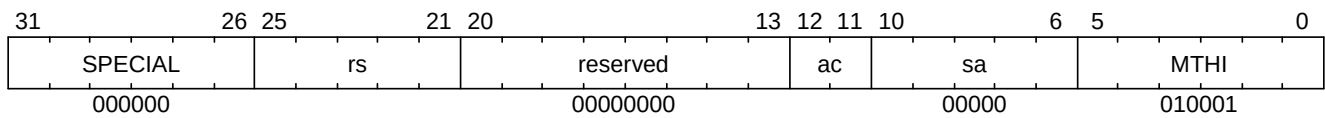


Figure 1060. Encoding (32 bits)

Decode

```
return MthiOperands(rs=rs, ac=ac)
```

`mthi`

Assembly

`MTHI {rs:gpr}`

Availability

AR < R6

Decode priority

0

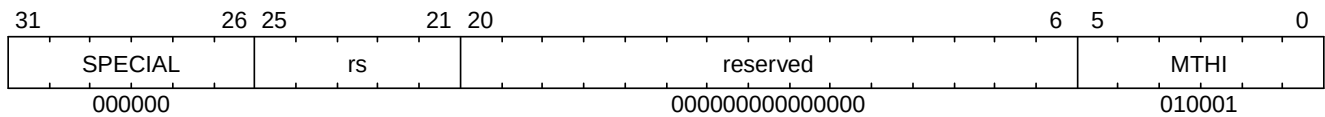


Figure 1061. Encoding (32 bits)

Decode

```
return MthiOperands(rs=rs)
```

microMIPS

`mthi.dsp.32`

Assembly

`MTHI {rs:gpr}, {ac:u2}`

Availability

DSP

Decode priority

0

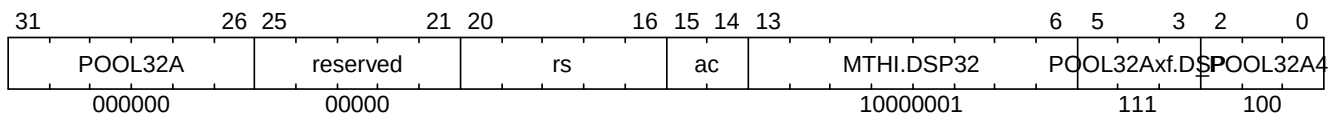


Figure 1062. Encoding (32 bits)

Decode

```
return MthiOperands(rs=rs, ac=ac)
```

`mthi.32`

Assembly

`MTHI {rs:gpr}`

Availability

MMAR < R6

Decode priority

0

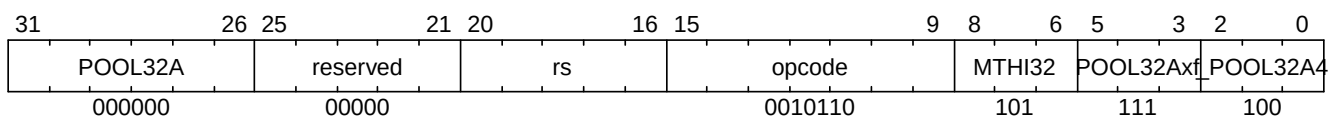


Figure 1063. Encoding (32 bits)

Decode

```
return MthiOperands(rs=rs)
```

nanoMIPS

`mthi.dsp.r7`

Assembly

`MTHI {rs:gpr}, {ac:u2}`

Availability

DSP

Decode priority

0

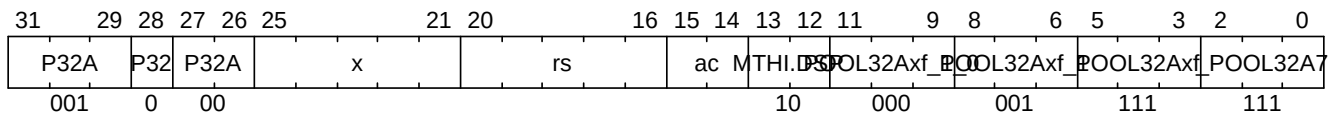


Figure 1064. Encoding (32 bits)

Decode

```
return MthiOperands(rs=rs, ac=ac)
```

8.390.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
    ctx.AC[operands.ac].HI = value
else:
    ctx.HI = value
```

8.390.4. Description

MTHI writes the selected GPR to HI at the implemented GPR width; 32-bit contexts write a 32-bit value and 64-bit contexts write all 64 bits.

8.390.5. Exceptions

None.

8.391. MTHLIP

8.391.1. Purpose

Move accumulator LO to HI, move a GPR word to LO, and advance DSPControl.Pos.

8.391.2. Formats

MIPS

`mthlip`

Assembly

`MTHLIP {rs:gpr}, {ac:u2}`

Availability

DSP && AR >= R2_3_5

Decode priority

0

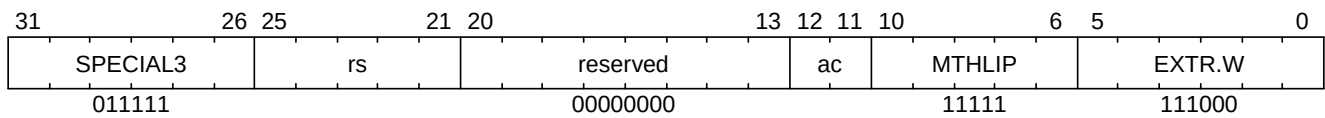


Figure 1065. Encoding (32 bits)

Decode

```
return DspMoveAccumulatorOperands(rs=rs, ac=ac)
```

microMIPS

`mthlip.32`

Assembly

`MTHLIP {rs:gpr}, {ac:u2}`

Availability

DSP

Decode priority

0

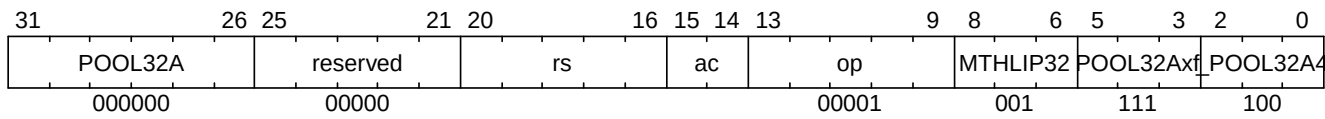


Figure 1066. Encoding (32 bits)

Decode

```
return DspMoveAccumulatorOperands(rs=rs, ac=ac)
```

nanoMIPS

`mthlip.32`

Assembly

`MTHLIP {rs:gpr}, {ac:u2}`

Availability

DSP

Decode priority

0

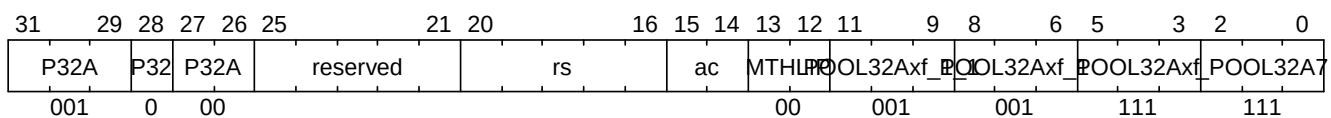


Figure 1067. Encoding (32 bits)

Decode

```
return DspMoveAccumulatorOperands(rs=rs, ac=ac)
```

8.391.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
old_lo = sign_extend(ctx.AC[operands.ac].LO, from_nbits=32)
ctx.AC[operands.ac].HI = sign_extend(old_lo, from_nbits=32)
ctx.AC[operands.ac].LO = sign_extend(source, from_nbits=32)
old_pos = ctx.DSPControl.Pos
if old_pos < 32:
    ctx.DSPControl.Pos = old_pos + 32
```

8.391.4. Description

Both accumulator parts receive sign-extended word values. Pos is increased by 32 when its old value is below 32; the architecture leaves the result unspecified for larger old positions.

8.391.5. Exceptions

None.

8.392. MTLO

8.392.1. Purpose

Copy a GPR value to the classic multiply/divide LO register.

8.392.2. Formats

MIPS

mtlo.dsp

Assembly

MTLO {rs:gpr}, {ac:u2}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

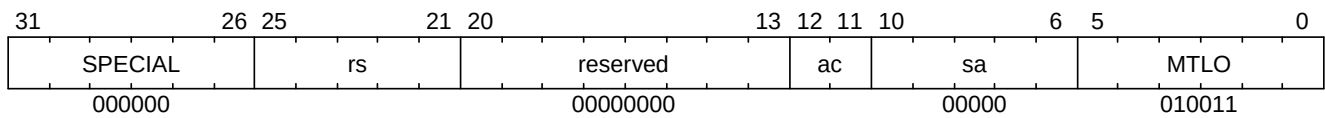


Figure 1068. Encoding (32 bits)

Decode

```
return MtloOperands(rs=rs, ac=ac)
```

mtlo

Assembly

MTLO {rs:gpr}

Availability

AR < R6

Decode priority

0

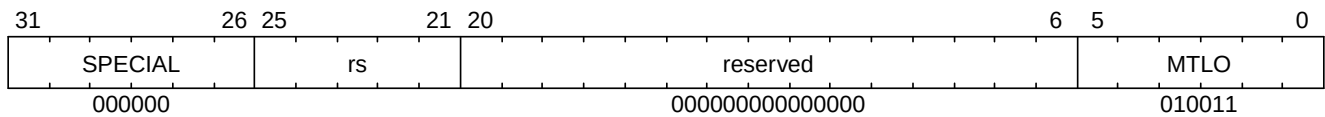


Figure 1069. Encoding (32 bits)

Decode

```
return MtloOperands(rs=rs)
```

microMIPS

mtlo.dsp.32

Assembly

MTLO {rs:gpr}, {ac:u2}

Availability

DSP

Decode priority

0

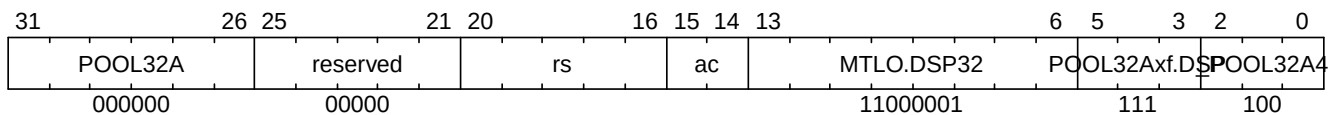


Figure 1070. Encoding (32 bits)

Decode

```
return MtloOperands(rs=rs, ac=ac)
```

mtlo.32

Assembly

MTLO {rs:gpr}

Availability

MMAR < R6

Decode priority

0

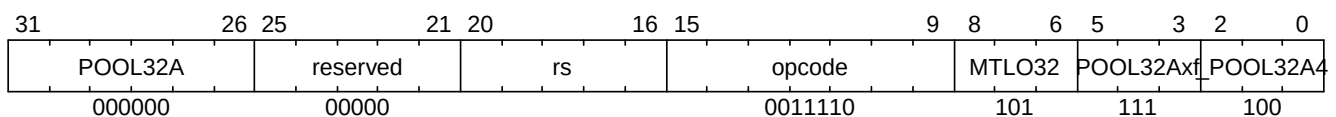


Figure 1071. Encoding (32 bits)

Decode

```
return MtloOperands(rs=rs)
```

nanoMIPS

mtlo.dsp.r7

Assembly

MTLO {rs:gpr}, {ac:u2}

Availability

DSP

Decode priority

0

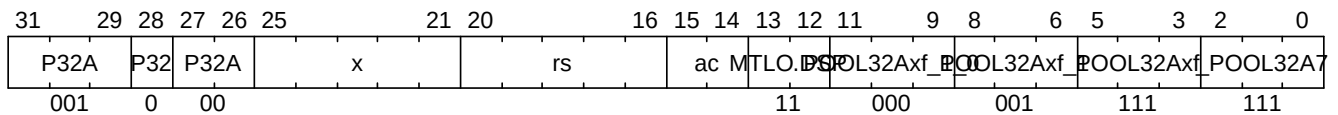


Figure 1072. Encoding (32 bits)

Decode

```
return MtloOperands(rs=rs, ac=ac)
```

8.392.3. Operation

```
value = int(zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
    ctx.AC[operands.ac].LO = value
else:
    ctx.L0 = value
```

8.392.4. Description

MTLO writes the selected GPR to LO at the implemented GPR width; 32-bit contexts write a 32-bit value and 64-bit contexts write all 64 bits.

8.392.5. Exceptions

None.

8.393. MUH

8.393.1. Purpose

Multiply signed 32-bit GPR values and write the high word.

8.393.2. Formats

MIPS

`muh`

Assembly

`MUH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R6

Decode priority

0

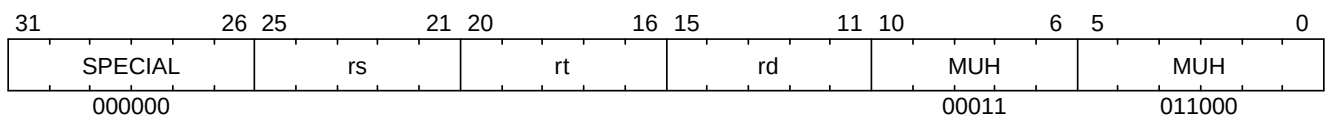


Figure 1073. Encoding (32 bits)

Decode

```
return MuhOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`muh.32.micromips`

Assembly

`MUH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

MMAR >= R6

Decode priority

0

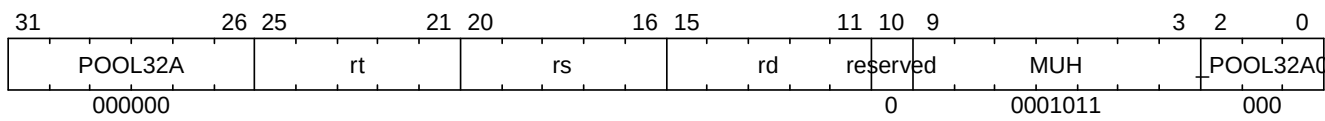


Figure 1074. Encoding (32 bits)

Decode

```
return MuhOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

muh.32

Assembly

MUH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

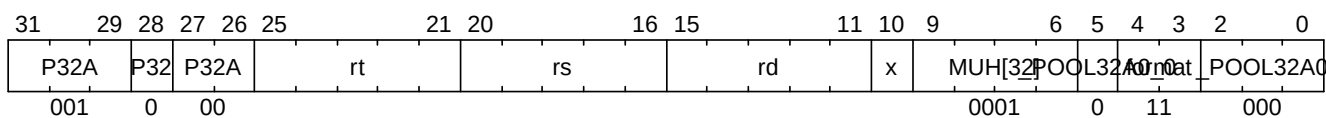


Figure 1075. Encoding (32 bits)

Decode

```
del ctx, exec
return MuhOperands(rd=rd, rs=rs, rt=rt)
```

8.393.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
result_hi = (int(lhs) * int(rhs)) >> 32
ctx.GPR[operands.rd] = sign_extend(result_hi, from_nbits=32)
```

8.393.4. Description

MUH multiplies the signed low words of rs and rt, takes product bits 63..32, and sign-extends that word to the active GPR width.

8.393.5. Exceptions

None.

8.394. MUHU

8.394.1. Purpose

Multiply unsigned 32-bit GPR values and write the high word.

8.394.2. Formats

MIPS

muhu

Assembly

MUHU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R6

Decode priority

0

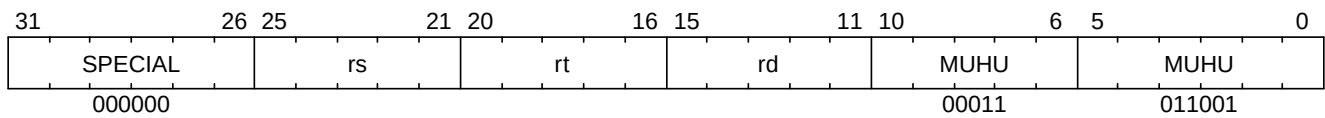


Figure 1076. Encoding (32 bits)

Decode

```
return MuhuOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

muhu.32.micromips

Assembly

MUHU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

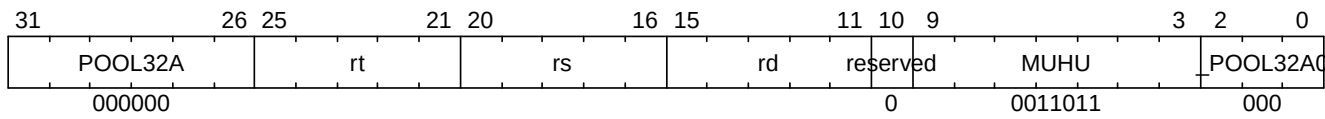


Figure 1077. Encoding (32 bits)

Decode

```
return MuhuOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

muhu.32

Assembly

MUHU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

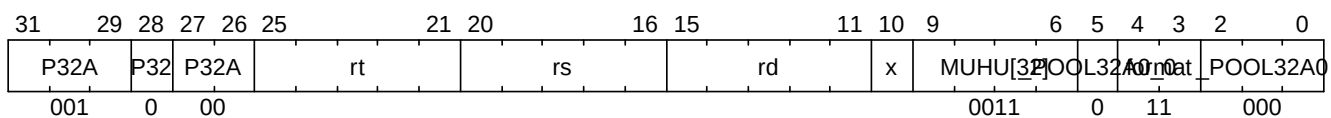


Figure 1078. Encoding (32 bits)

Decode

```
del ctx, exec
return MuhuOperands(rd=rd, rs=rs, rt=rt)
```

8.394.3. Operation

```
lhs = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
result_hi = (int(lhs) * int(rhs)) >> 32
ctx.GPR[operands.rd] = sign_extend(result_hi, from_nbits=32)
```

8.394.4. Description

MUHU multiplies the unsigned low words of rs and rt, takes product bits 63..32, and sign-extends that word to the active GPR width.

8.394.5. Exceptions

None.

8.395. MUL

8.395.1. Purpose

Multiply signed 32-bit values in *rs* and *rt*, and place the low 32 bits of the result in *rd*.

8.395.2. Formats

MIPS

mul

Assembly

MUL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

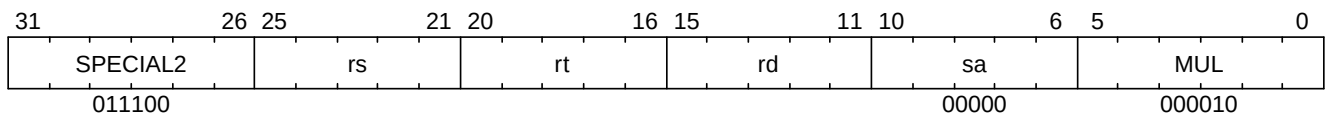


Figure 1079. Encoding (32 bits)

Decode

```
return MulOperands(rd=rd, rs=rs, rt=rt)
```

mul.r6

Assembly

MUL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R6

Decode priority

0

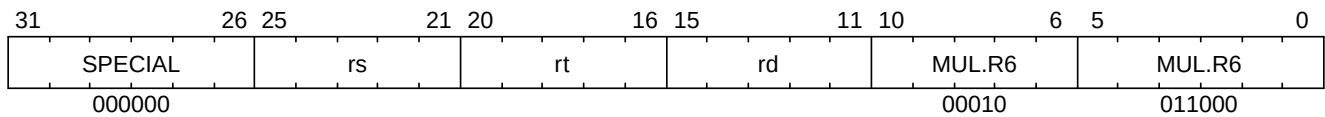


Figure 1080. Encoding (32 bits)

Decode

```
return MulOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

mul.32

Assembly

MUL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

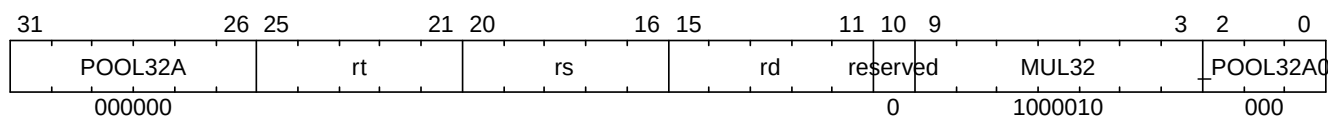


Figure 1081. Encoding (32 bits)

Decode

```
return MulOperands(rd=rd, rs=rs, rt=rt)
```

mul.r6.32

Assembly

MUL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

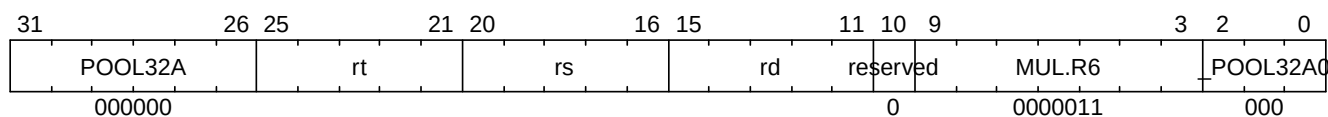


Figure 1082. Encoding (32 bits)

Decode

```
return MulOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mul.32

Assembly

MUL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

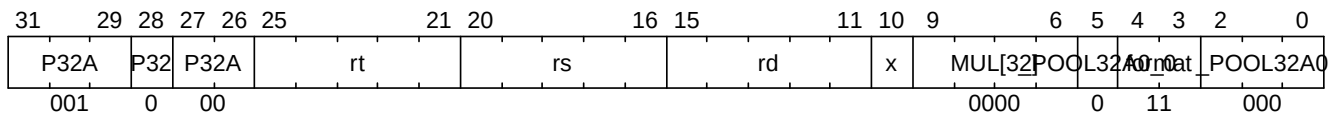


Figure 1083. Encoding (32 bits)

Decode

```
return MulOperands(rd=rd, rs=rs, rt=rt)
```

mul.4x4

Assembly

MUL {rd:gpr4}, {rt:gpr4}, {rs:gpr4}

Availability

!(NMS)

Decode priority

0

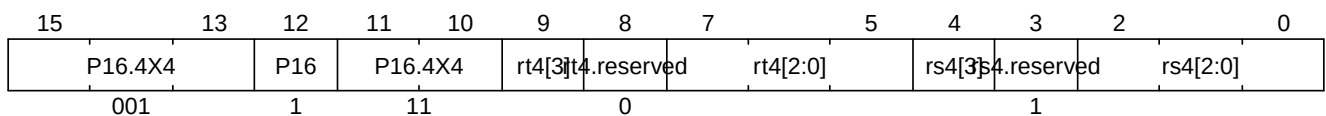


Figure 1084. Encoding (16 bits)

Decode

```
del ctx, exec
rd = GPR4.decode(rt4)
return MulOperands(rd=rd, rs=rd, rt=GPR4.decode(rs4))
```

8.395.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
result = int(lhs) * int(rhs)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.395.4. Description

MUL[32] interprets both source operands as signed 32-bit values. The low 32 bits of the product are sign-extended when written to the implemented-width destination GPR, making the operation uniform for 32-bit and 64-bit GPR files.

8.395.5. Exceptions

- **RI** (Reserved Instruction) — MUL[4X4] is reserved when CP0 Config5.NMS is set.

8.396. MUL.D

8.396.1. Purpose

Multiply two D-format floating-point values.

8.396.2. Formats

MIPS

`mul.d`

Assembly

`MUL.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

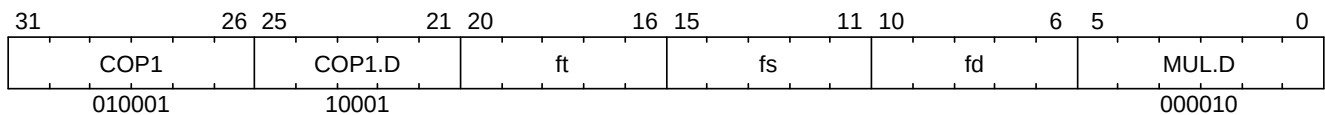


Figure 1085. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`mul.d`

Assembly

`MUL.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

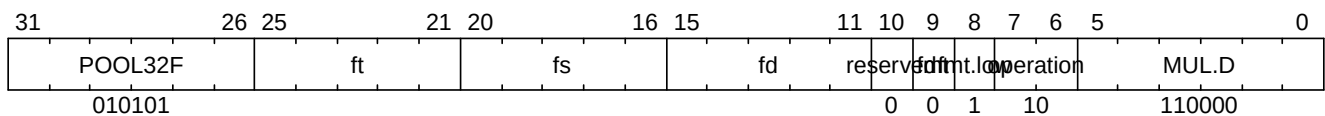


Figure 1086. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

mul.d

Assembly

MUL.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

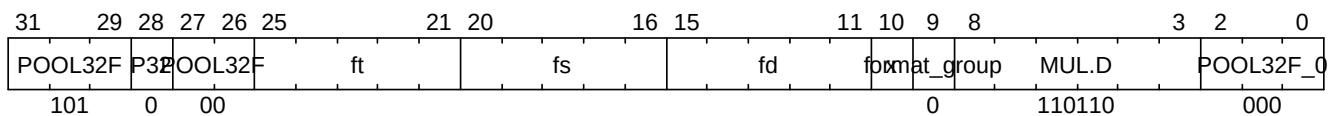


Figure 1087. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.396.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_mul(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.396.4. Description

MUL.D validates CP1 and D-format support before updating the destination FPR.

8.396.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.397. MUL.PH

8.397.1. Purpose

DSP multiply or accumulate operation MUL.PH.

8.397.2. Formats

MIPS

mul.ph

Assembly

MUL.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

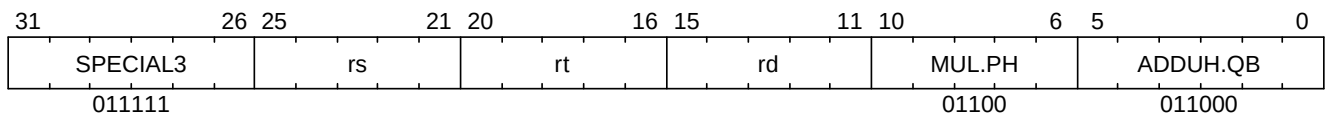


Figure 1088. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

mul.ph.32

Assembly

MUL.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

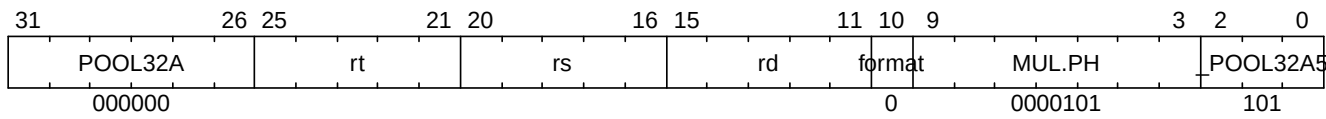


Figure 1089. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mul.ph.32

Assembly

MUL.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

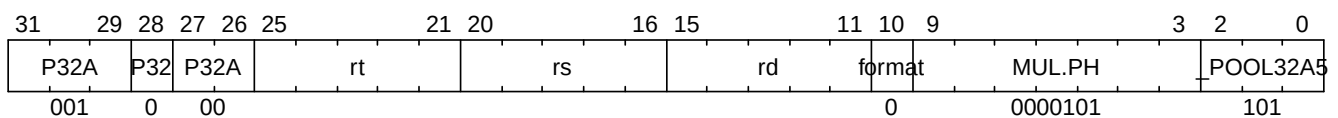


Figure 1090. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.397.3. Operation

```
validate_access_to_dsp_resources(ctx)
hi_value = signed((ctx.GPR[operands.rs] >> 16) & 0xFFFF, 16) * signed(
    (ctx.GPR[operands.rt] >> 16) & 0xFFFF, 16
)
lo_value = signed(ctx.GPR[operands.rs] & 0xFFFF, 16) * signed(
    ctx.GPR[operands.rt] & 0xFFFF, 16
)
hi, ov_hi = hi_value & 0xFFFF, hi_value < -0x8000 or hi_value > 0x7FFF
lo, ov_lo = lo_value & 0xFFFF, lo_value < -0x8000 or lo_value > 0x7FFF
if ov_hi or ov_lo:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend(
    ((hi & 0xFFFF) << 16) | (lo & 0xFFFF), from_nbits=32
)
```

8.397.4. Description

MUL.PH is a DSP multiply or accumulator operation.

8.397.5. Exceptions

None.

8.398. MUL.PS

8.398.1. Purpose

Perform the paired-single mul operation.

8.398.2. Formats

MIPS

mul.ps

Assembly

MUL.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

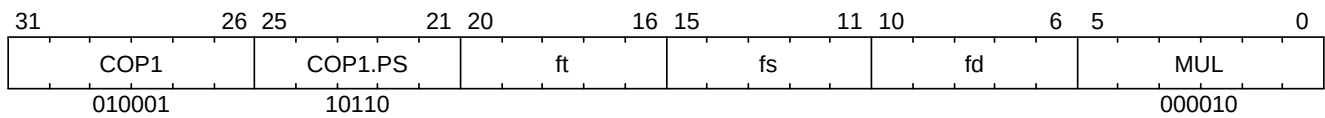


Figure 1091. Encoding (32 bits)

Decode

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

mul.ps

Assembly

MUL.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

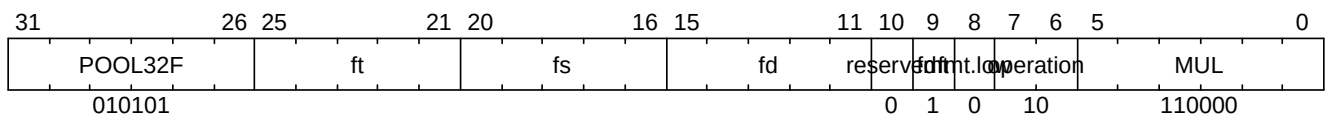


Figure 1092. Encoding (32 bits)

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.398.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
left = get_fpr(ctx, operands.fs, "PS")
right = get_fpr(ctx, operands.ft, "PS")
low, low_flags = ieee_mul("S", rounding, left & 0xFFFF_FFFF, right & 0xFFFF_FFFF)
high, high_flags = ieee_mul("S", rounding, left >> 32, right >> 32)
write_fpu_result(ctx, operands.fd, low | (high << 32), low_flags | high_flags, "PS")
```

8.398.4. Description

MUL.PS applies independently to the upper and lower S-format lanes.

8.398.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.399. MUL.S

8.399.1. Purpose

Multiply two S-format floating-point values.

8.399.2. Formats

MIPS

`mul.s`

Assembly

`MUL.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

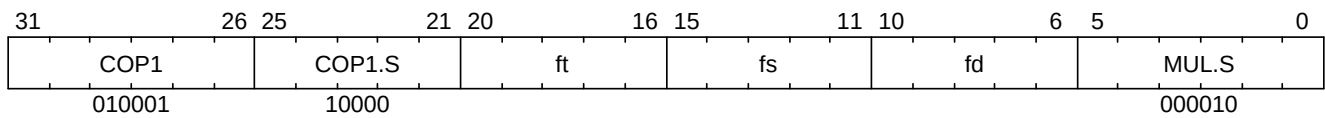


Figure 1093. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`mul.s`

Assembly

`MUL.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

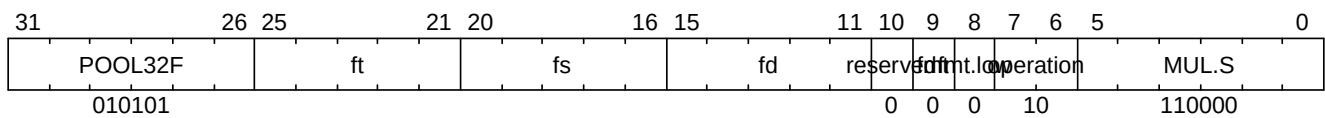


Figure 1094. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

mul.s

Assembly

MUL.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

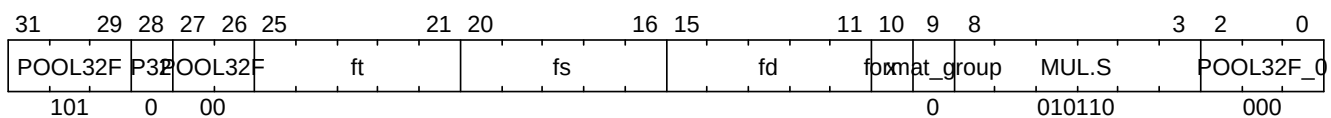


Figure 1095. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.399.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_mul(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.399.4. Description

MUL.S validates CP1 and S-format support before updating the destination FPR.

8.399.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.400. MUL_S.PH

8.400.1. Purpose

DSP multiply or accumulate operation MUL_S.PH.

8.400.2. Formats

MIPS

mul_s.ph

Assembly

MUL_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

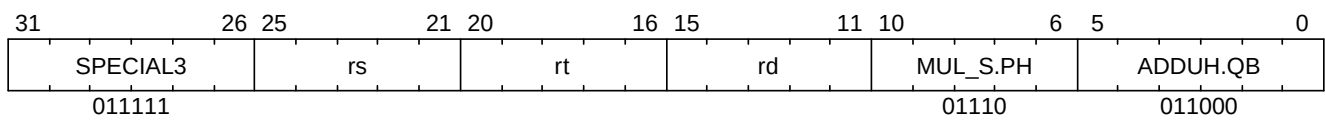


Figure 1096. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

mul_s.ph.32

Assembly

MUL_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

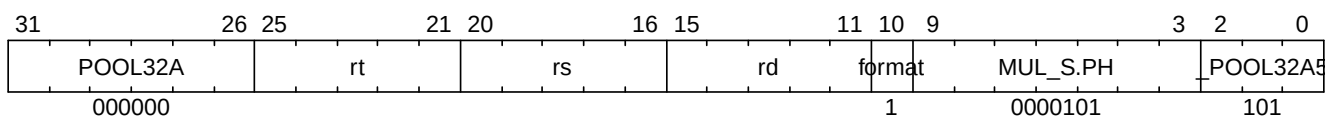


Figure 1097. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mul_s.ph.32

Assembly

MUL_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

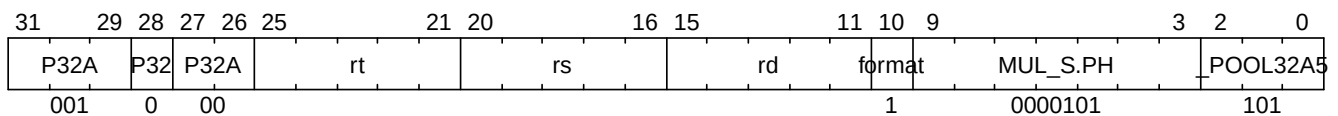


Figure 1098. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.400.3. Operation

```
validate_access_to_dsp_resources(ctx)
hi, ov_hi = sat_signed(
    signed((ctx.GPR[operands.rs] >> 16) & 0xFFFF, 16)
    * signed((ctx.GPR[operands.rt] >> 16) & 0xFFFF, 16),
    16,
)
lo, ov_lo = sat_signed(
    signed(ctx.GPR[operands.rs] & 0xFFFF, 16)
    * signed(ctx.GPR[operands.rt] & 0xFFFF, 16),
    16,
)
if ov_hi or ov_lo:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend((hi << 16) | lo, from_nbits=32)
```

8.400.4. Description

MUL_S.PH is a DSP multiply or accumulator operation.

8.400.5. Exceptions

None.

8.401. MULEQ_S.W.PHL

8.401.1. Purpose

DSP multiply or accumulate operation MULEQ_S.W.PHL.

8.401.2. Formats

MIPS

muleq_s.w.phl

Assembly

MULEQ_S.W.PHL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

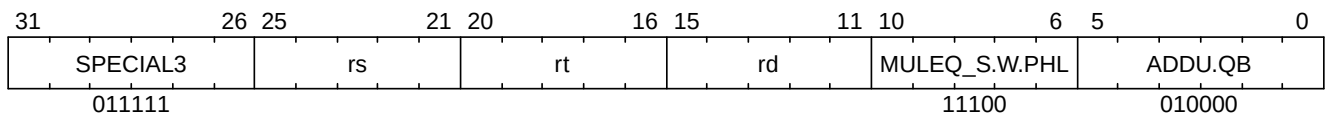


Figure 1099. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

muleq_s.w.phl.32

Assembly

MULEQ_S.W.PHL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

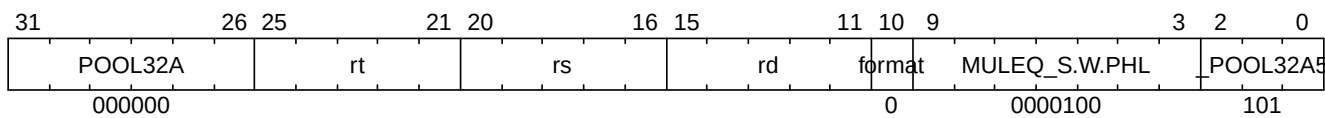


Figure 1100. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

muleq_s.w.phl.32

Assembly

MULEQ_S.W.PHL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

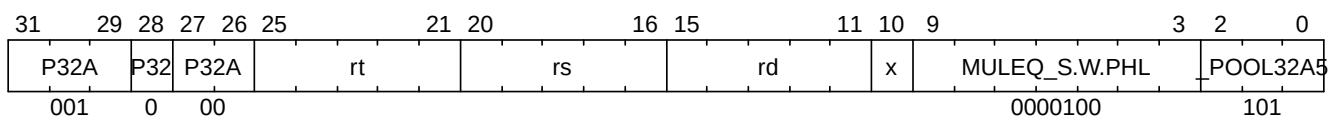


Figure 1101. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.401.3. Operation

```
validate_access_to_dsp_resources(ctx)
result, overflow = muleq_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.401.4. Description

MULEQ_S.W.PHL is a DSP multiply or accumulator operation.

8.401.5. Exceptions

None.

8.402. MULEQ_S.W.PHR

8.402.1. Purpose

DSP multiply or accumulate operation MULEQ_S.W.PHR.

8.402.2. Formats

MIPS

muleq_s.w.phr

Assembly

MULEQ_S.W.PHR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

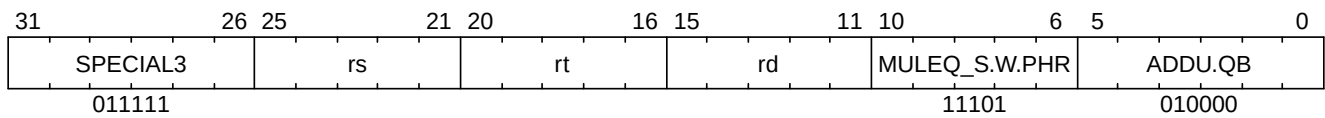


Figure 1102. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

muleq_s.w.phr.32

Assembly

MULEQ_S.W.PHR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

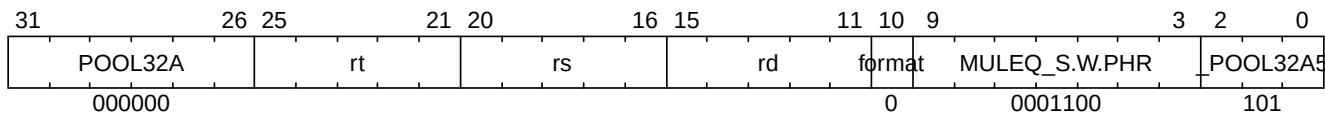


Figure 1103. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

muleq_s.w.phr.32

Assembly

MULEQ_S.W.PHR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

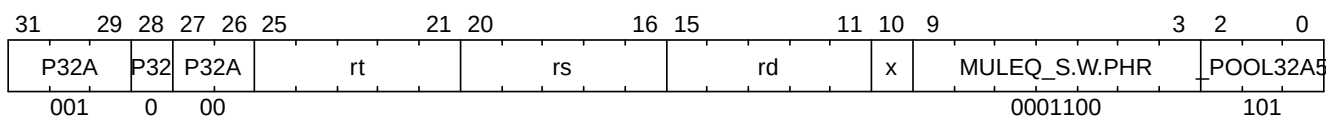


Figure 1104. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.402.3. Operation

```
validate_access_to_dsp_resources(ctx)
result, overflow = muleq_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.402.4. Description

MULEQ_S.W.PHR is a DSP multiply or accumulator operation.

8.402.5. Exceptions

None.

8.403. MULEU_S.PH.QBL

8.403.1. Purpose

DSP multiply or accumulate operation MULEU_S.PH.QBL.

8.403.2. Formats

MIPS

muleu_s.ph.qbl

Assembly

MULEU_S.PH.QBL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

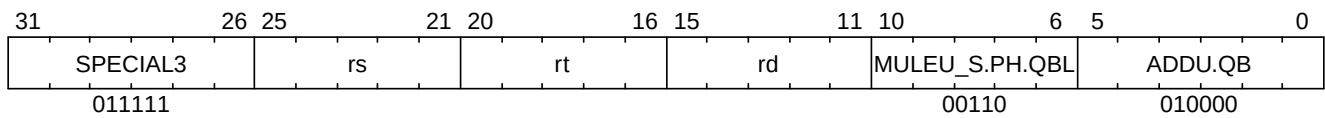


Figure 1105. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

muleu_s.ph.qbl.32

Assembly

MULEU_S.PH.QBL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

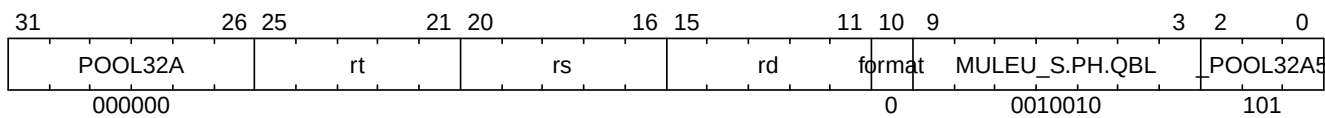


Figure 1106. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

muleu_s.ph.qbl.32

Assembly

MULEU_S.PH.QBL {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

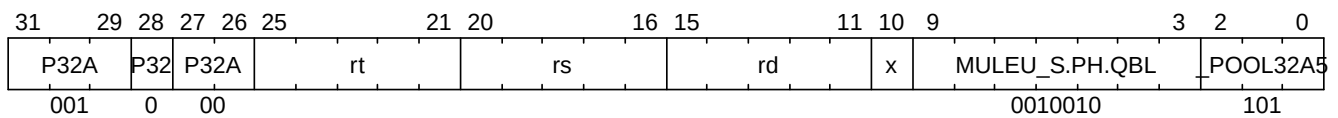


Figure 1107. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.403.3. Operation

```
validate_access_to_dsp_resources(ctx)
one, two, overflow = muleu_product(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], False
)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend((one << 16) | two, from_nbits=32)
```

8.403.4. Description

MULEU_S.PH.QBL is a DSP multiply or accumulator operation.

8.403.5. Exceptions

None.

8.404. MULEU_S.PH.QBR

8.404.1. Purpose

DSP multiply or accumulate operation MULEU_S.PH.QBR.

8.404.2. Formats

MIPS

muleu_s.ph.qbr

Assembly

MULEU_S.PH.QBR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

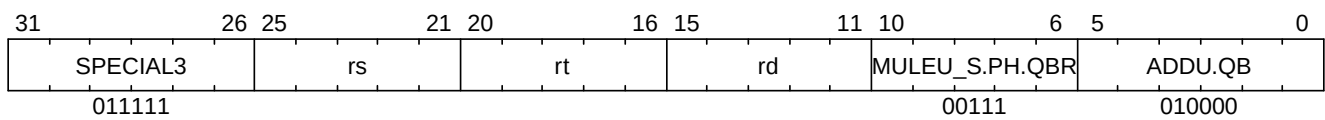


Figure 1108. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

muleu_s.ph.qbr.32

Assembly

MULEU_S.PH.QBR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

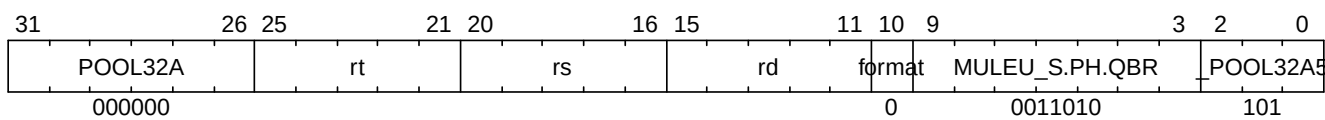


Figure 1109. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

muleu_s.ph.qbr.32

Assembly

MULEU_S.PH.QBR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

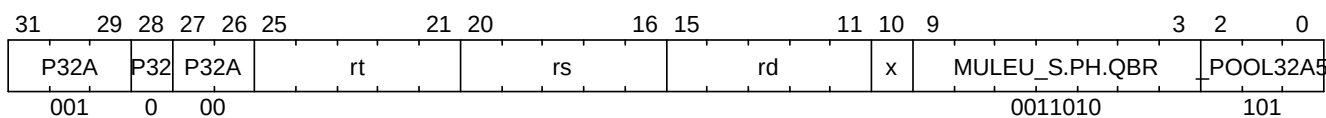


Figure 1110. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.404.3. Operation

```
validate_access_to_dsp_resources(ctx)
one, two, overflow = muleu_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend((one << 16) | two, from_nbits=32)
```

8.404.4. Description

MULEU_S.PH.QBR is a DSP multiply or accumulator operation.

8.404.5. Exceptions

None.

8.405. MULQ_RS.PH

8.405.1. Purpose

DSP multiply or accumulate operation MULQ_RS.PH.

8.405.2. Formats

MIPS

mulq_rs.ph

Assembly

MULQ_RS.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

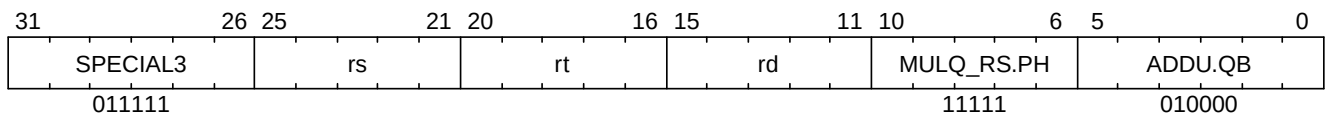


Figure 1111. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

mulq_rs.ph.32

Assembly

MULQ_RS.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

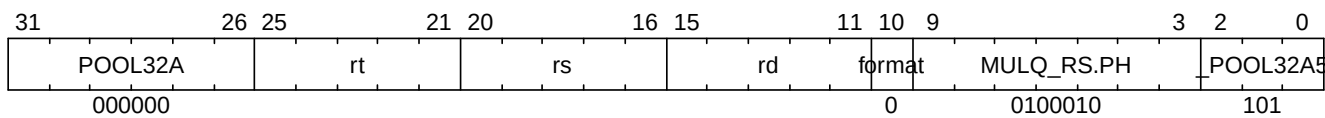


Figure 1112. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mulq_rs.ph.32

Assembly

MULQ_RS.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

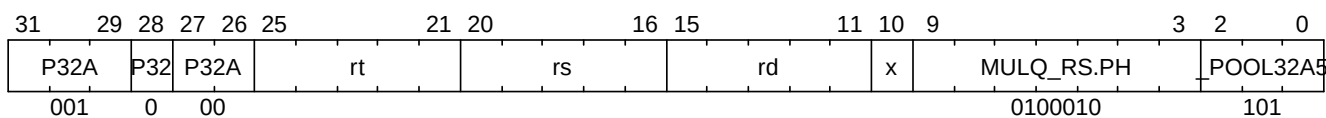


Figure 1113. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.405.3. Operation

```
validate_access_to_dsp_resources(ctx)
hi, ov_hi = q15_pair(
    ctx.GPR[operands.rs] >> 16, ctx.GPR[operands.rt] >> 16, True, True
)
lo, ov_lo = q15_pair(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True, True)
if ov_hi or ov_lo:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend((hi << 16) | lo, from_nbits=32)
```

8.405.4. Description

MULQ_RS.PH is a DSP multiply or accumulator operation.

8.405.5. Exceptions

None.

8.406. MULQ_RS.W

8.406.1. Purpose

DSP multiply or accumulate operation MULQ_RS.W.

8.406.2. Formats

MIPS

mulq_rs.w

Assembly

MULQ_RS.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

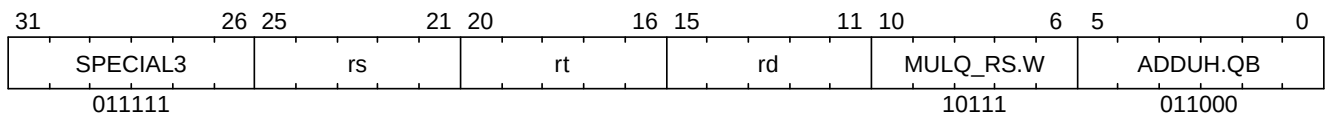


Figure 1114. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

mulq_rs.w.32

Assembly

MULQ_RS.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

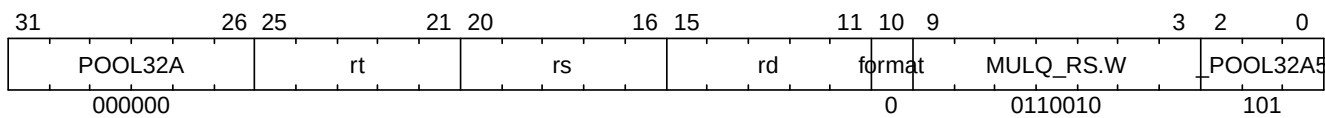


Figure 1115. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mulq_rs.w.32

Assembly

MULQ_RS.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

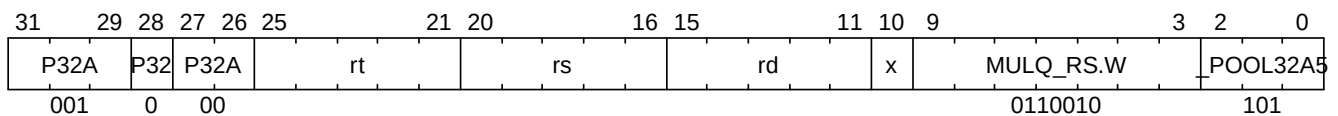


Figure 1116. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.406.3. Operation

```
validate_access_to_dsp2_resources(ctx)
product, overflow = mulq_word(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend(product, from_nbits=32)
```

8.406.4. Description

MULQ_RS.W is a DSP multiply or accumulator operation.

8.406.5. Exceptions

None.

8.407. MULQ_S.PH

8.407.1. Purpose

DSP multiply or accumulate operation MULQ_S.PH.

8.407.2. Formats

MIPS

mulq_s.ph

Assembly

MULQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

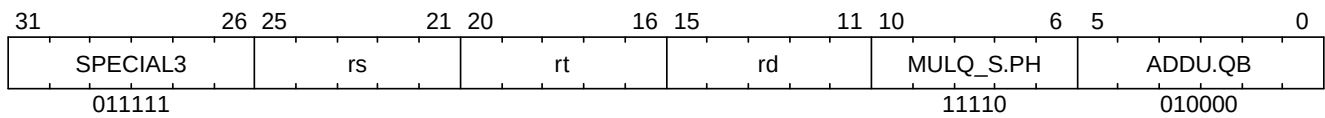


Figure 1117. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

mulq_s.ph.32

Assembly

MULQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

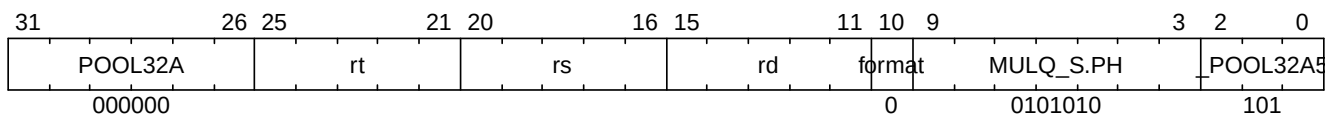


Figure 1118. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mulq_s.ph.32

Assembly

MULQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

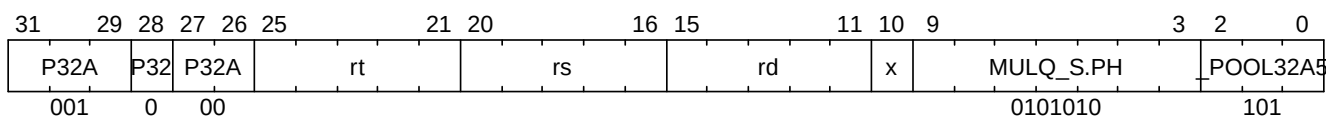


Figure 1119. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.407.3. Operation

```
validate_access_to_dsp2_resources(ctx)
hi, ov_hi = q15_pair(
    ctx.GPR[operands.rs] >> 16, ctx.GPR[operands.rt] >> 16, False, True
)
lo, ov_lo = q15_pair(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False, True)
if ov_hi or ov_lo:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend((hi << 16) | lo, from_nbits=32)
```

8.407.4. Description

MULQ_S.PH is a DSP multiply or accumulator operation.

8.407.5. Exceptions

None.

8.408. MULQ_S.W

8.408.1. Purpose

DSP multiply or accumulate operation MULQ_S.W.

8.408.2. Formats

MIPS

mulq_s.w

Assembly

MULQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

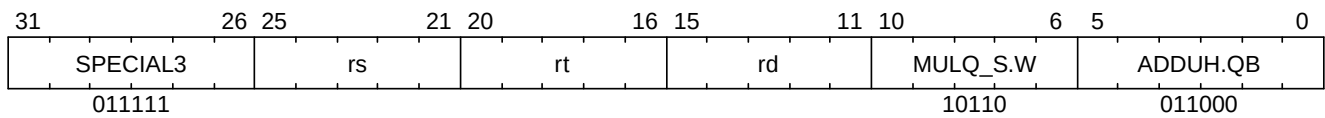


Figure 1120. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

mulq_s.w.32

Assembly

MULQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

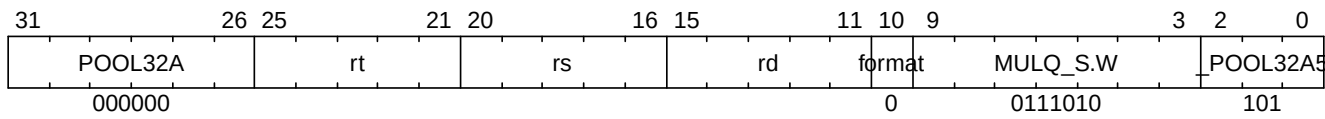


Figure 1121. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mulq_s.w.32

Assembly

MULQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

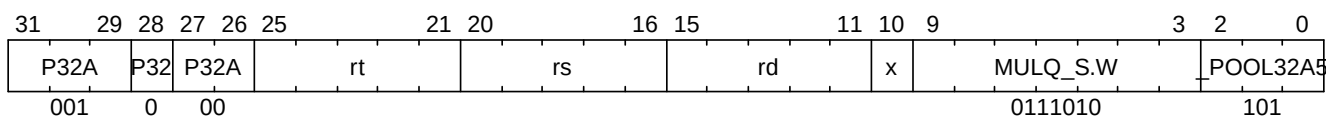


Figure 1122. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.408.3. Operation

```
validate_access_to_dsp2_resources(ctx)
product, overflow = mulq_word(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x20
ctx.GPR[operands.rd] = sign_extend(product, from_nbits=32)
```

8.408.4. Description

MULQ_S.W is a DSP multiply or accumulator operation.

8.408.5. Exceptions

None.

8.409. MULSA.W.PH

8.409.1. Purpose

DSP multiply or accumulate operation MULSA.W.PH.

8.409.2. Formats

MIPS

`mulsa.w.ph`

Assembly

`MULSA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

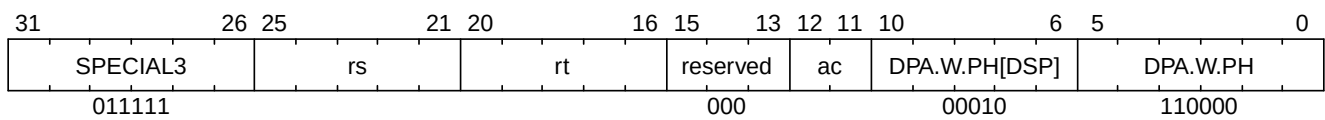


Figure 1123. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

`mulsa.w.ph.32`

Assembly

`MULSA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

DSP2

Decode priority

0

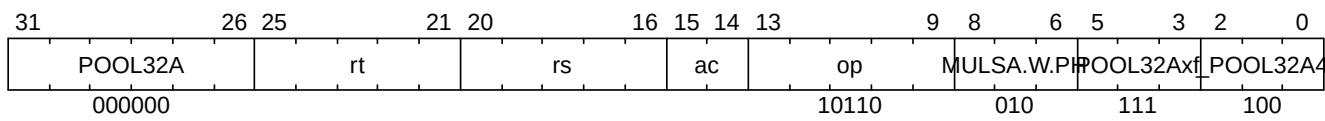


Figure 1124. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

`mulsa.w.ph.32`

Assembly

`MULSA.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

DSP2

Decode priority

0

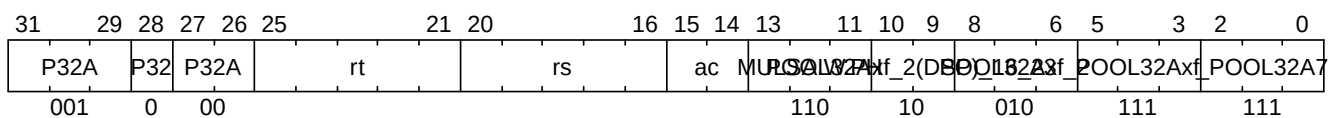


Figure 1125. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.409.3. Operation

```
validate_access_to_dsp2_resources(ctx)
high = signed((ctx.GPR[operands.rs] >> 16) & 0xFFFF, 16) * signed(
    (ctx.GPR[operands.rt] >> 16) & 0xFFFF, 16
)
low = signed(ctx.GPR[operands.rs] & 0xFFFF, 16) * signed(
    ctx.GPR[operands.rt] & 0xFFFF, 16
)
write_accumulator(
    ctx, operands.ac, accumulator_value(ctx, operands.ac) + high - low
)
```

8.409.4. Description

MULSA.W.PH is a DSP multiply or accumulator operation.

8.409.5. Exceptions

None.

8.410. MULSAQ_S.W.PH

8.410.1. Purpose

DSP multiply or accumulate operation MULSAQ_S.W.PH.

8.410.2. Formats

MIPS

`mulsaq_s.w.ph`

Assembly

`MULSAQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

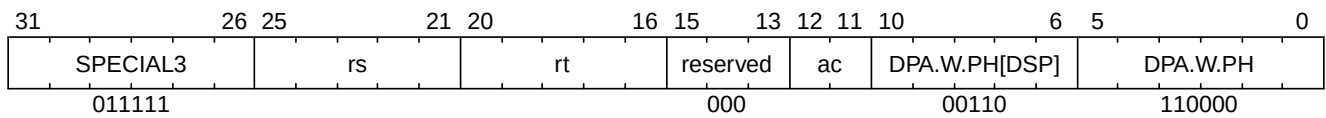


Figure 1126. Encoding (32 bits)

Decode

```
return DspAccumulator0perands(ac=ac, rs=rs, rt=rt)
```

microMIPS

`mulsaq_s.w.ph.32`

Assembly

`MULSAQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

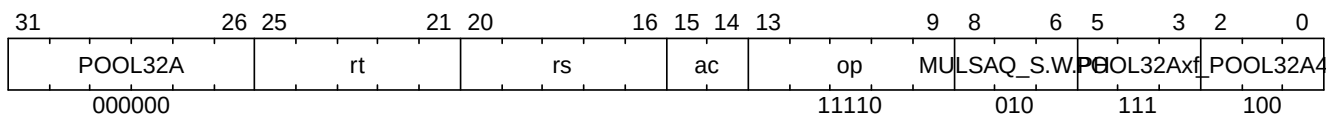


Figure 1127. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

nanoMIPS

`mulsaq_s.w.ph.32`

Assembly

`MULSAQ_S.W.PH {ac:u2}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

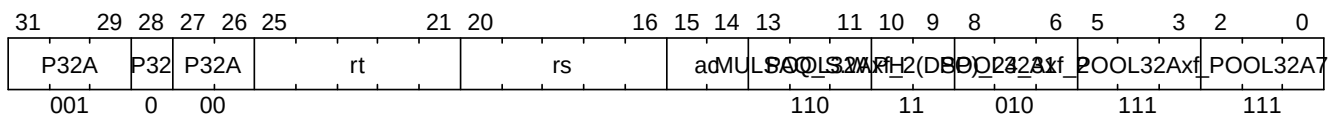


Figure 1128. Encoding (32 bits)

Decode

```
return DspAccumulatorOperands(ac=ac, rs=rs, rt=rt)
```

8.410.3. Operation

```
validate_access_to_dsp_resources(ctx)
dotp_hi, ov_hi = product_q15(
    (ctx.GPR[operands.rs] >> 16) & 0xFFFF, (ctx.GPR[operands.rt] >> 16) & 0xFFFF
)
dotp_lo, ov_lo = product_q15(
    ctx.GPR[operands.rs] & 0xFFFF, ctx.GPR[operands.rt] & 0xFFFF
)
if ov_hi or ov_lo:
    set_accumulator_overflow(ctx, operands.ac)
write_accumulator(
    ctx,
    operands.ac,
    accumulator_value(ctx, operands.ac) + signed(dotp_hi, 32) - signed(dotp_lo, 32),
)
```

8.410.4. Description

MULSAQ_S.W.PH is a DSP multiply or accumulator operation.

8.410.5. Exceptions

None.

8.411. MULT

8.411.1. Purpose

Multiply two signed 32-bit GPR values and write HI and LO.

8.411.2. Formats

MIPS

mult

Assembly

MULT {rs:gpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

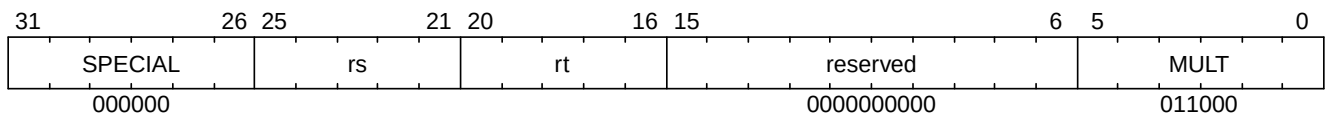


Figure 1129. Encoding (32 bits)

Decode

```
return MultOperands(rs=rs, rt=rt, ac=None)
```

mult.dsp

Assembly

MULT {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

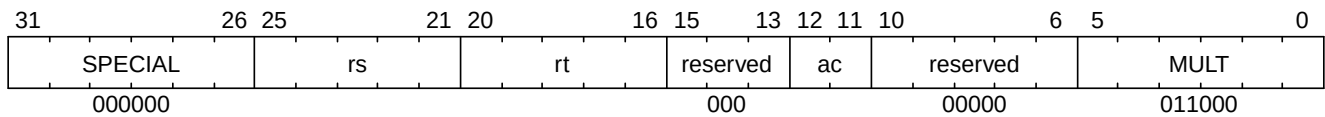


Figure 1130. Encoding (32 bits)

Decode

```
return MultOperands(rs=rs, rt=rt, ac=ac)
```


microMIPS

mult.32

Assembly

MULT {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

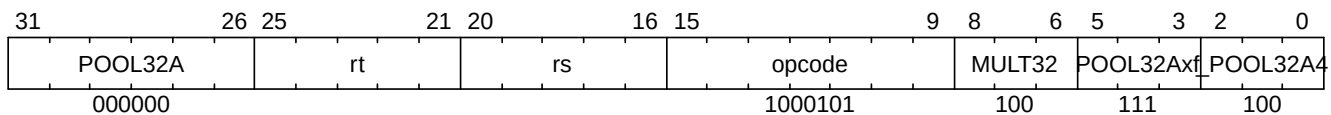


Figure 1131. Encoding (32 bits)

Decode

```
return MultOperands(rs=rs, rt=rt, ac=None)
```

mult.dsp.32

Assembly

MULT {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

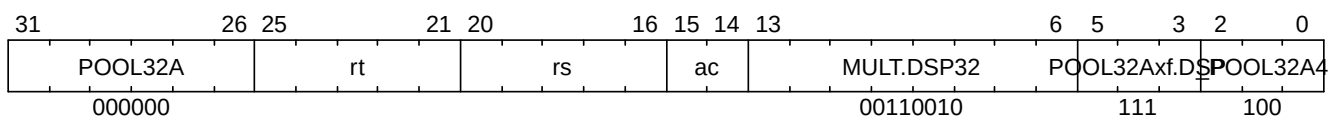


Figure 1132. Encoding (32 bits)

Decode

```
return MultOperands(rs=rs, rt=rt, ac=ac)
```

nanoMIPS

mult.dsp.r7

Assembly

MULT {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

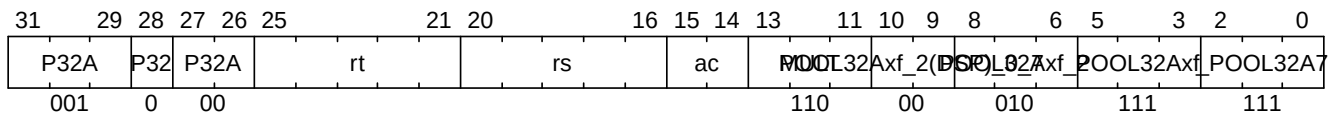


Figure 1133. Encoding (32 bits)

Decode

```
return MultOperands(rs=rs, rt=rt, ac=ac)
```

8.411.3. Operation

```
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
    product = classic_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], False)
    write_selected_accumulator(ctx, operands.ac, product)
```

8.411.4. Description

The full signed 64-bit product is split between HI and LO.

8.411.5. Exceptions

None.

8.412. MULTU

8.412.1. Purpose

Multiply two unsigned 32-bit GPR values and write HI and LO.

8.412.2. Formats

MIPS

multu

Assembly

MULTU {rs:gpr}, {rt:gpr}

Availability

AR < R6

Decode priority

0

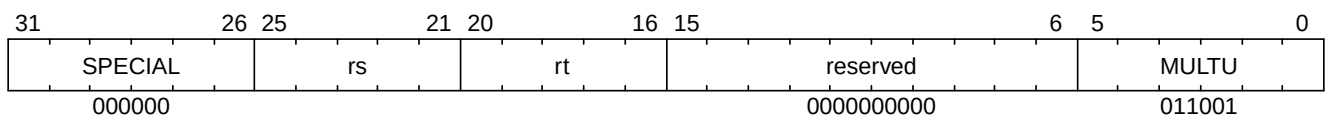


Figure 1134. Encoding (32 bits)

Decode

```
return MultuOperands(rs=rs, rt=rt, ac=None)
```

multu.dsp

Assembly

MULTU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

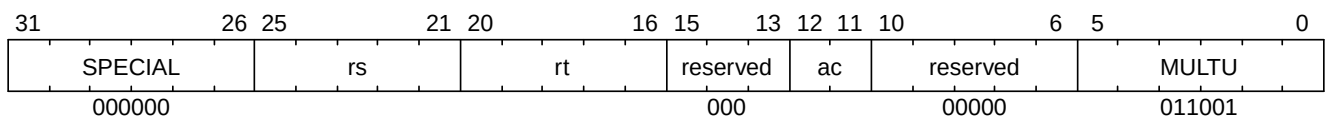


Figure 1135. Encoding (32 bits)

Decode

```
return MultuOperands(rs=rs, rt=rt, ac=ac)
```

microMIPS

multu.32

Assembly

MULTU {rs:gpr}, {rt:gpr}

Availability

MMAR < R6

Decode priority

0

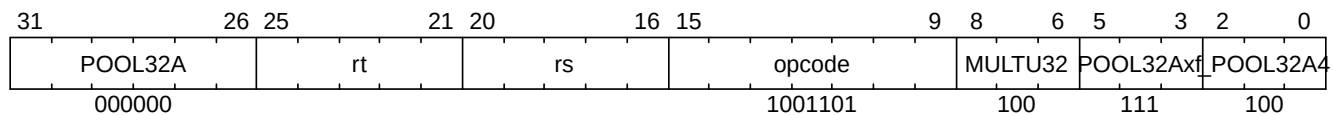


Figure 1136. Encoding (32 bits)

Decode

```
return MultuOperands(rs=rs, rt=rt, ac=None)
```

multu.dsp.32

Assembly

MULTU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

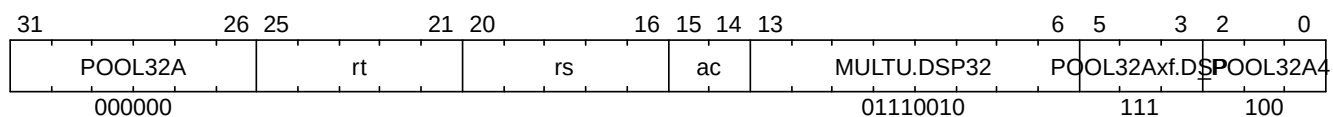


Figure 1137. Encoding (32 bits)

Decode

```
return MultuOperands(rs=rs, rt=rt, ac=ac)
```

nanoMIPS

multu.dsp.r7

Assembly

MULTU {ac:u2}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

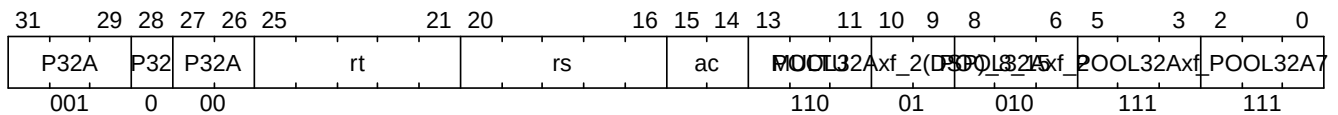


Figure 1138. Encoding (32 bits)

Decode

```
return MultuOperands(rs=rs, rt=rt, ac=ac)
```

8.412.3. Operation

```
if operands.ac is not None:
    validate_access_to_dsp_accumulator(ctx, operands.ac)
    product = classic_product(ctx.GPR[operands.rs], ctx.GPR[operands.rt], True)
    write_selected_accumulator(ctx, operands.ac, product)
```

8.412.4. Description

The full unsigned 64-bit product is split between HI and LO.

8.412.5. Exceptions

None.

8.413. Mulu

8.413.1. Purpose

Multiply unsigned 32-bit GPR values and write the low word.

8.413.2. Formats

MIPS

`mulu`

Assembly

`MULU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R6

Decode priority

0

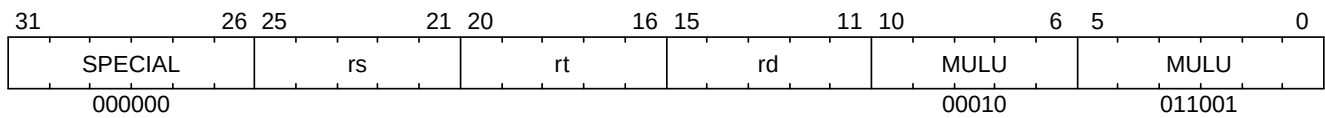


Figure 1139. Encoding (32 bits)

Decode

```
return MuluOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`mulu.32.micromips`

Assembly

`MULU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

MMAR >= R6

Decode priority

0

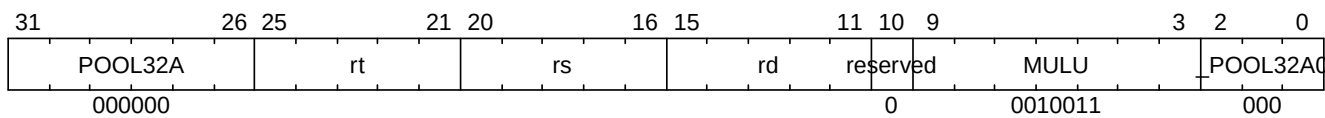


Figure 1140. Encoding (32 bits)

Decode

```
return MuluOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

mulu.32

Assembly

MULU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

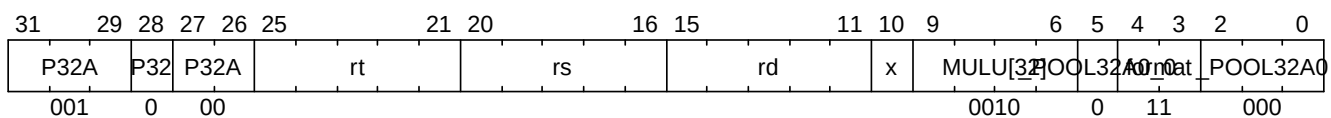


Figure 1141. Encoding (32 bits)

Decode

```
del ctx, exec
return MuluOperands(rd=rd, rs=rs, rt=rt)
```

8.413.3. Operation

```
lhs = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
result = int(lhs) * int(rhs)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.413.4. Description

MULU multiplies the unsigned low words of rs and rt and sign-extends the low 32-bit product to the active GPR width.

8.413.5. Exceptions

None.

8.414. NAL

8.414.1. Purpose

Write the address after a delay slot to GPR 31 without taking a branch.

8.414.2. Formats

MIPS

`nal`

Assembly

`NAL`

Availability

AR >= R6

Decode priority

0

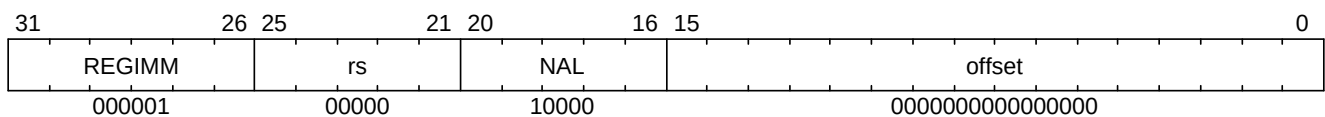


Figure 1142. Encoding (32 bits)

Decode

```
exec.isCTI = True
exec.hasDelaySlot = True
return NalOperands()
```

8.414.3. Operation

```
delayed_branch(ctx, ctx.next_pc, taken=False)
ctx.GPR[31] = effective_address(ctx, ctx.next_pc, 4)
```

8.414.4. Description

NAL behaves as an untaken delayed branch and writes PC+8 to the link register. It preserves the historical branch-and-link scheduling effect.

8.414.5. Exceptions

None.

8.415. NEG.D

8.415.1. Purpose

Negate a D-format value.

8.415.2. Formats

MIPS

neg.d

Assembly

NEG.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

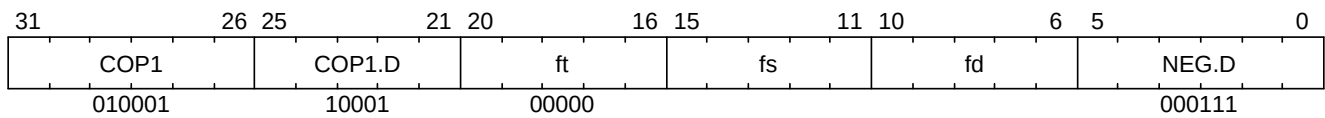


Figure 1143. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

neg.d

Assembly

NEG.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

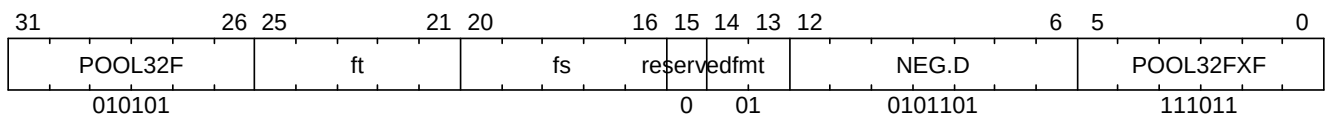


Figure 1144. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

neg.d

Assembly

NEG.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

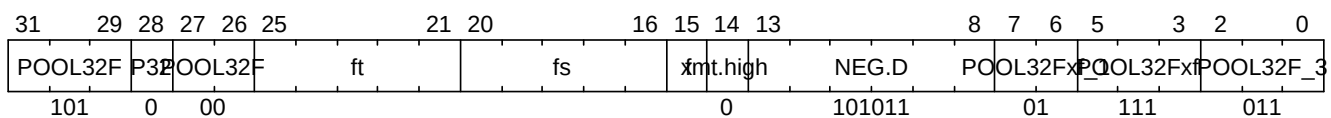


Figure 1145. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.415.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_neg(
    "D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.415.4. Description

NEG.D validates CP1 and D-format support before updating the destination FPR.

8.415.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.416. NEG.PS

8.416.1. Purpose

Perform the paired-single neg operation.

8.416.2. Formats

MIPS

neg.ps

Assembly

NEG.PS {fd:fpr}, {fs:fpr}

Availability

AR < R6

Decode priority

0

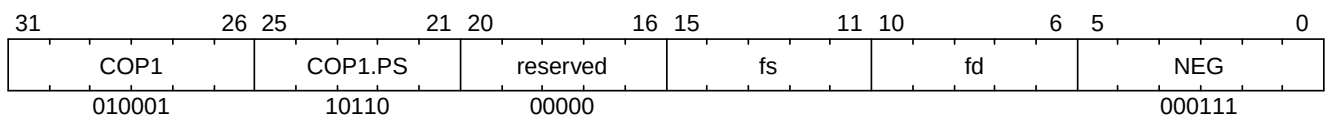


Figure 1146. Encoding (32 bits)

Decode

```
return FpuPSUnaryOperands(fd=fd, fs=fs)
```

microMIPS

neg.ps

Assembly

NEG.PS {fd:fpr}, {fs:fpr}

Availability

MMAR < R6

Decode priority

0

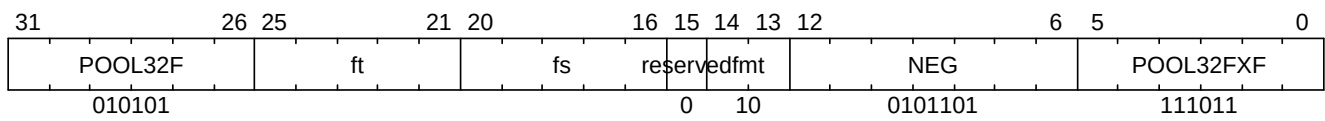


Figure 1147. Encoding (32 bits)

```
return FpuPSUnaryOperands(fd=ft, fs=fs)
```

8.416.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
source = get_fpr(ctx, operands.fs, "PS")
low, low_flags = ieee_neg("S", rounding, source & 0xFFFF_FFFF)
high, high_flags = ieee_neg("S", rounding, source >> 32)
write_fpu_result(ctx, operands.fd, low | (high << 32), low_flags | high_flags, "PS")
```

8.416.4. Description

NEG.PS applies independently to the upper and lower S-format lanes.

8.416.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.417. NEG.S

8.417.1. Purpose

Negate an S-format value.

8.417.2. Formats

MIPS

neg.s

Assembly

NEG.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

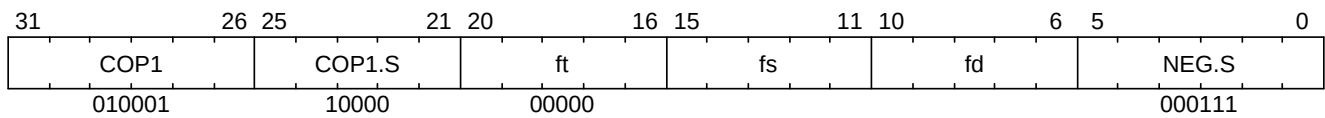


Figure 1148. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

neg.s

Assembly

NEG.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

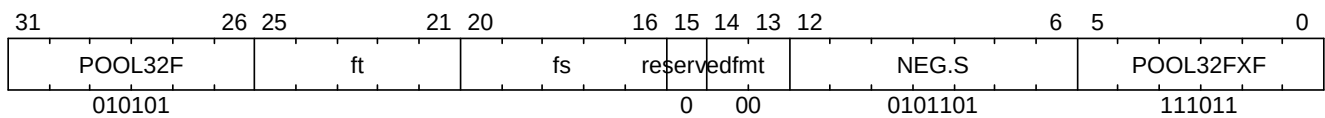


Figure 1149. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

neg.s

Assembly

NEG.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

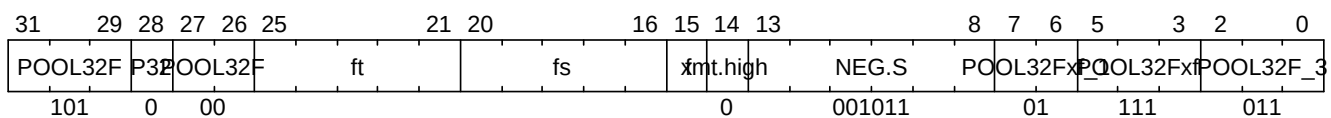


Figure 1150. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.417.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_neg(
    "S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.417.4. Description

NEG.S validates CP1 and S-format support before updating the destination FPR.

8.417.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.418. NMADD.D

8.418.1. Purpose

Compute the pre-Release-6 nmadd of D-format values using the explicit accumulator `fr`.

8.418.2. Formats

MIPS

`nmadd.d`

Assembly

`NMADD.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR < R6

Decode priority

0

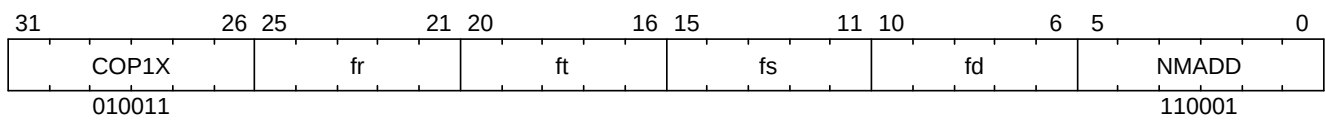


Figure 1151. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

`nmadd.d`

Assembly

`NMADD.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR < R6

Decode priority

0

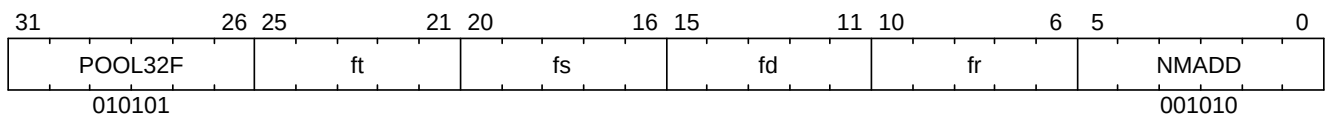


Figure 1152. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.418.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_mul_add(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "D"),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
)
result = ieee_negated_arithmetic_result("D", fpu_rounding_mode(ctx), result)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.418.4. Description

NMADD.D combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.418.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.419. NMADD.PS

8.419.1. Purpose

Compute the paired-single nmadd operation.

8.419.2. Formats

MIPS

nmadd.ps

Assembly

NMADD.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

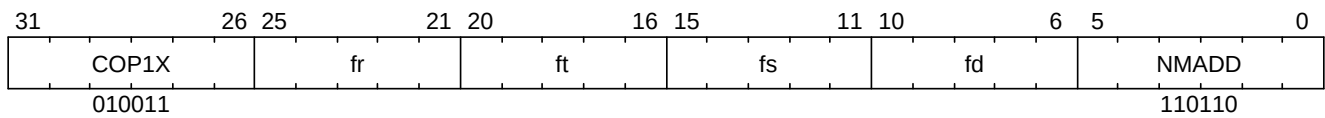


Figure 1153. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

nmadd.ps

Assembly

NMADD.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

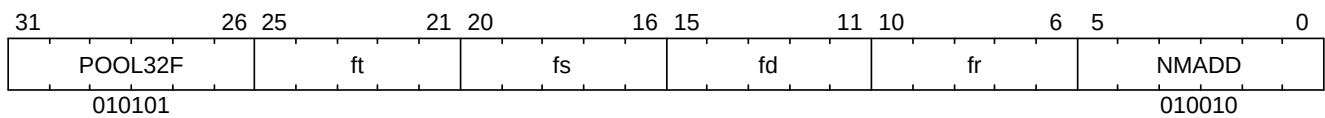


Figure 1154. Encoding (32 bits)

```
return Cop1xMultiplyAdd0operands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.419.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
fr = get_fpr(ctx, operands.fr, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
low, low_flags = ieee_mul_add(
    "S", rounding, fr & 0xFFFF_FFFF, fs & 0xFFFF_FFFF, ft & 0xFFFF_FFFF
)
high, high_flags = ieee_mul_add("S", rounding, fr >> 32, fs >> 32, ft >> 32)
low = ieee_negated_arithmetic_result("S", rounding, low)
high = ieee_negated_arithmetic_result("S", rounding, high)
write_fpu_result(
    ctx,
    operands.fd,
    low | (high << 32),
    low_flags | high_flags,
    "PS",
)
```

8.419.4. Description

NMADD.PS applies independently to the upper and lower S-format lanes.

8.419.5. Exceptions

- **CPU** (Coprocesor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.420. NMADD.S

8.420.1. Purpose

Compute the pre-Release-6 nmadd of S-format values using the explicit accumulator `fr`.

8.420.2. Formats

MIPS

`nmadd.s`

Assembly

`NMADD.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR < R6

Decode priority

0

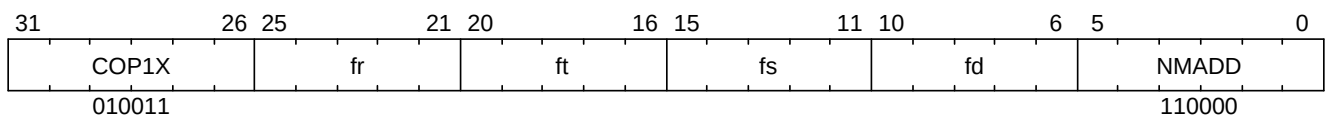


Figure 1155. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

`nmadd.s`

Assembly

`NMADD.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR < R6

Decode priority

0

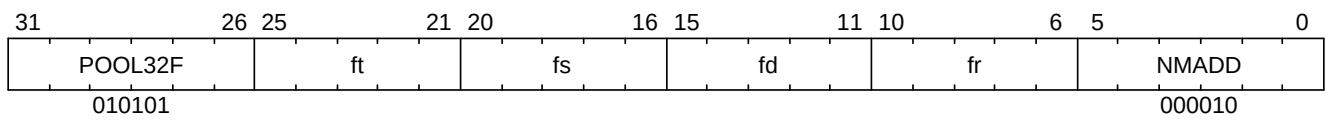


Figure 1156. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.420.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_mul_add(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "S"),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
)
result = ieee_negated_arithmetic_result("S", fpu_rounding_mode(ctx), result)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.420.4. Description

NMADD.S combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.420.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.421. NMSUB.D

8.421.1. Purpose

Compute the pre-Release-6 nmsub of D-format values using the explicit accumulator `fr`.

8.421.2. Formats

MIPS

`nmsub.d`

Assembly

`NMSUB.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR < R6

Decode priority

0

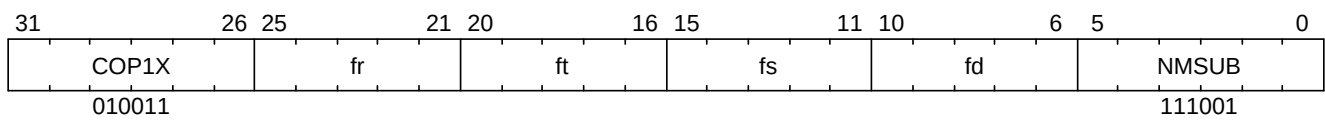


Figure 1157. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

`nmsub.d`

Assembly

`NMSUB.D {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR < R6

Decode priority

0

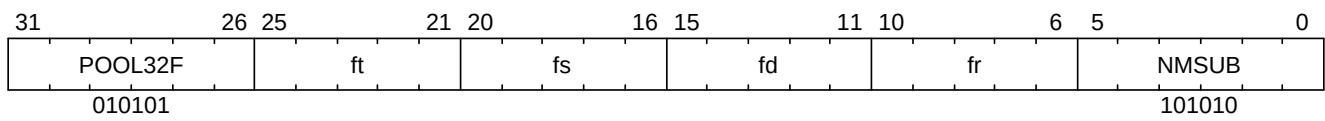


Figure 1158. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.421.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_mul_add(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "D"),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    subtract=True,
)
result = ieee_negated_arithmetic_result("D", fpu_rounding_mode(ctx), result)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.421.4. Description

NMSUB.D combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.421.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.422. NMSUB.PS

8.422.1. Purpose

Compute the paired-single nmsub operation.

8.422.2. Formats

MIPS

nmsub.ps

Assembly

NMSUB.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

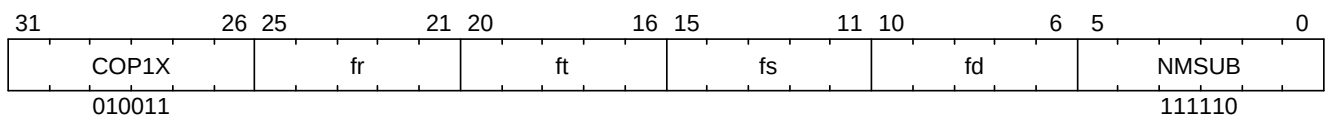


Figure 1159. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

nmsub.ps

Assembly

NMSUB.PS {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

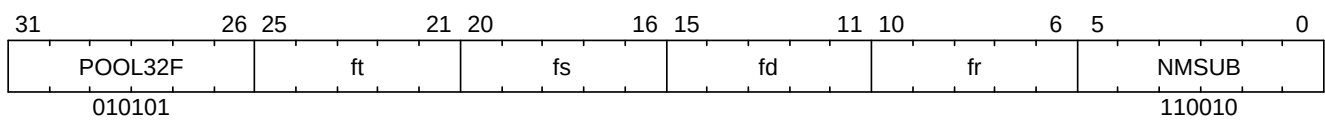


Figure 1160. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.422.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
fr = get_fpr(ctx, operands.fr, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
low, low_flags = ieee_mul_add(
    "S",
    rounding,
    fr & 0xFFFF_FFFF,
    fs & 0xFFFF_FFFF,
    ft & 0xFFFF_FFFF,
    subtract=True,
)
high, high_flags = ieee_mul_add(
    "S", rounding, fr >> 32, fs >> 32, ft >> 32, subtract=True
)
low = ieee_negated_arithmetic_result("S", rounding, low)
high = ieee_negated_arithmetic_result("S", rounding, high)
write_fpu_result(
    ctx,
    operands.fd,
    low | (high << 32),
    low_flags | high_flags,
    "PS",
)
```

8.422.4. Description

NMSUB.PS applies independently to the upper and lower S-format lanes.

8.422.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.423. NMSUB.S

8.423.1. Purpose

Compute the pre-Release-6 nmsub of S-format values using the explicit accumulator `fr`.

8.423.2. Formats

MIPS

`nmsub.s`

Assembly

`NMSUB.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR < R6

Decode priority

0

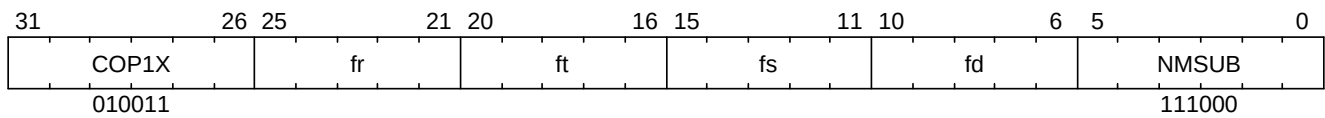


Figure 1161. Encoding (32 bits)

Decode

```
return Cop1xMultiplyAdd0perands(fd=fd, fr=fr, fs=fs, ft=ft)
```

microMIPS

`nmsub.s`

Assembly

`NMSUB.S {fd:fpr}, {fr:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR < R6

Decode priority

0

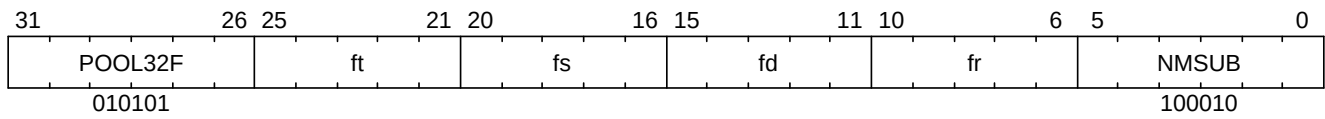


Figure 1162. Encoding (32 bits)

```
return Cop1xMultiplyAddOperands(fd=fd, fr=fr, fs=fs, ft=ft)
```

8.423.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_mul_add(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fr, "S"),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    subtract=True,
)
result = ieee_negated_arithmetic_result("S", fpu_rounding_mode(ctx), result)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.423.4. Description

NMSUB.S combines **fr** and the product of **fs** and **ft** using the architecturally defined multiply/add sequence.

8.423.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.; The operation raised an enabled floating-point condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 or the required floating-point format is unavailable.

8.424. NOP

8.424.1. Purpose

Perform no architectural operation.

8.424.2. Formats

MIPS

`nop`

Assembly

`NOP`

Availability

true

Decode priority

1

Overrides lower-priority formats

- SLL (`sll`)

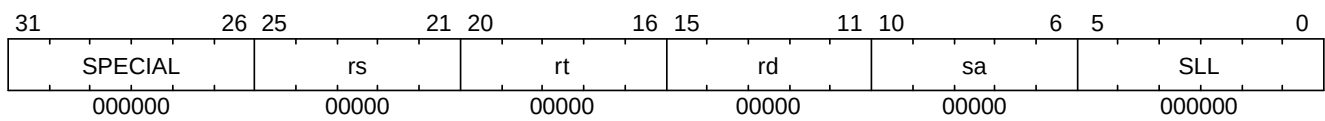


Figure 1163. Encoding (32 bits)

Decode

```
return Nop0perands()
```

MIPS16E

`nop.16`

Assembly

`NOP`

Availability

MIPS16E

Decode priority

1

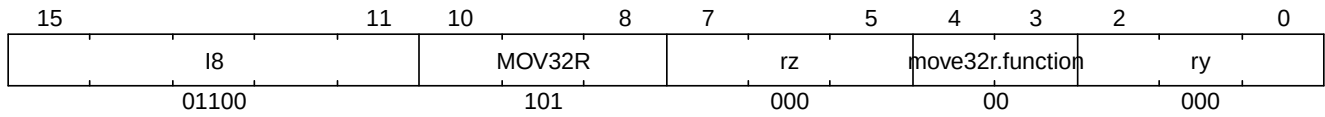


Figure 1164. Encoding (16 bits)

Decode

```
return Nop0perands()
```

microMIPS

nop.32

Assembly

NOP

Availability

true

Decode priority

1

Overrides lower-priority formats

- SLL (s11.32)

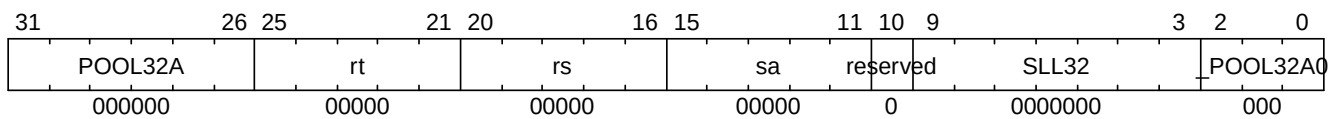


Figure 1165. Encoding (32 bits)

Decode

```
return Nop0perands()
```

nanoMIPS

nop.16

Assembly

NOP

Availability

true

Decode priority

1

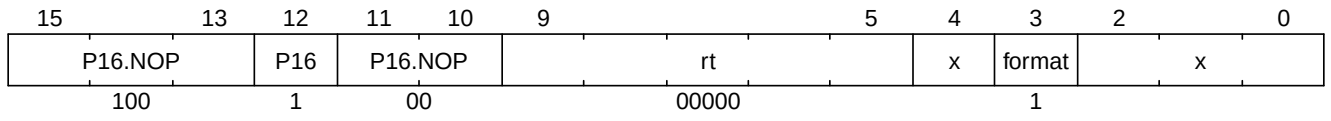


Figure 1166. Encoding (16 bits)

Decode

```
return NopOperands()
```

nop.32

Assembly

NOP

Availability

true

Decode priority

1

Overrides lower-priority formats

- SLL (**sll.32**)

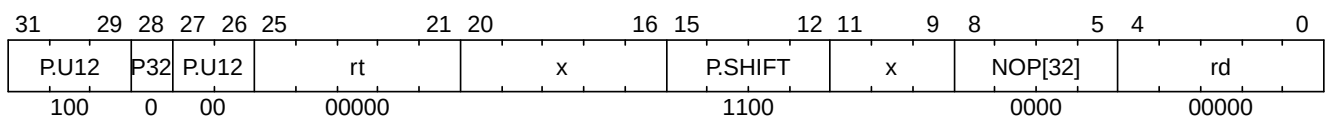


Figure 1167. Encoding (32 bits)

Decode

```
return NopOperands()
```

8.424.3. Operation

pass

8.424.4. Description

The classic MIPS and microMIPS forms are assembly idioms for the zero shift form of SLL. The MIPS16E form is an assembly idiom for MOVE GPR zero, GPR 16. nanoMIPS defines NOP[16] and NOP[32] encodings; NOP[32] aliases the zero shift form of SLL[32]. Ignored **x** fields in the nanoMIPS forms do not affect execution.

8.424.5. Exceptions

None.

8.425. NOR

8.425.1. Purpose

Compute the bitwise NOR of two source GPRs and place the result in a destination GPR.

8.425.2. Formats

MIPS

`nor`

Assembly

`NOR {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

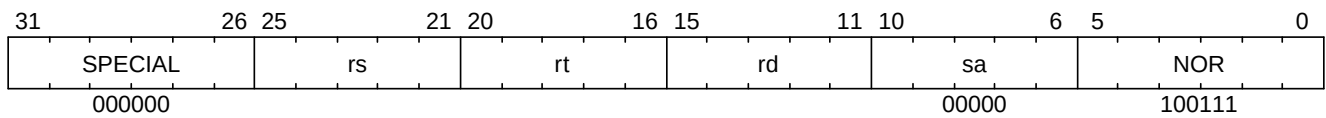


Figure 1168. Encoding (32 bits)

Decode

```
del ctx, exec
return NorOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`nor.32`

Assembly

`NOR {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

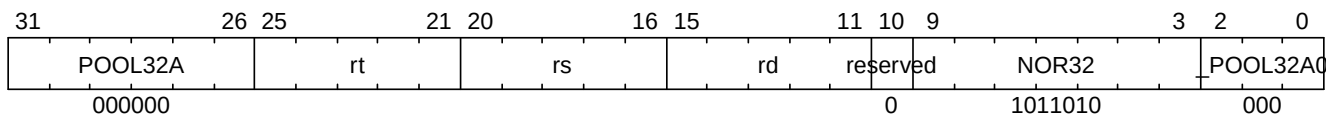


Figure 1169. Encoding (32 bits)

Decode

```
del ctx, exec
return NorOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

nor.32

Assembly

NOR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

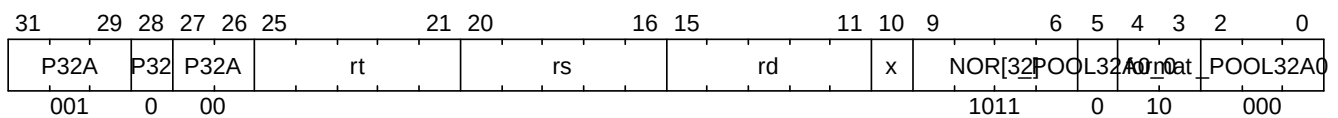


Figure 1170. Encoding (32 bits)

Decode

```
return NorOperands(rd=rd, rs=rs, rt=rt)
```

8.425.3. Operation

```
ctx.GPR[operands.rd] = ~(ctx.GPR[operands.rs] | ctx.GPR[operands.rt])
```

8.425.4. Description

NOR[32] complements the bitwise OR of **rs** and **rt**. The operation is evaluated and written at the implemented GPR width, making the definition uniform for 32-bit and 64-bit nanoMIPS profiles.

8.425.5. Exceptions

None.

8.426. NOT

8.426.1. Purpose

Invert the value of a GPR and write it to a GPR.

8.426.2. Formats

microMIPS

not.16

Assembly

NOT {rt:gpr3}, {rs:gpr3}

Availability

MMAR < R6

Decode priority

0

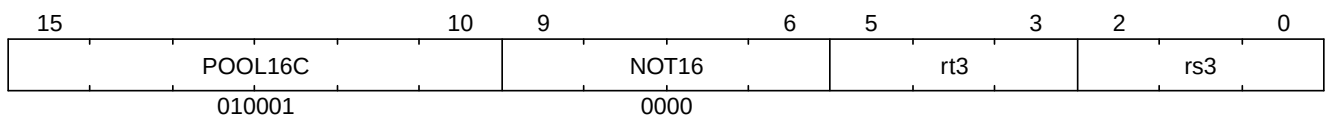


Figure 1171. Encoding (16 bits)

Decode

```
return Not0operands(rt=MICROMIPS_GPR3.decode(rt3), rs=MICROMIPS_GPR3.decode(rs3))
```

not.r6.16

Assembly

NOT {rt:gpr3}, {rs:gpr3}

Availability

MMAR >= R6

Decode priority

0

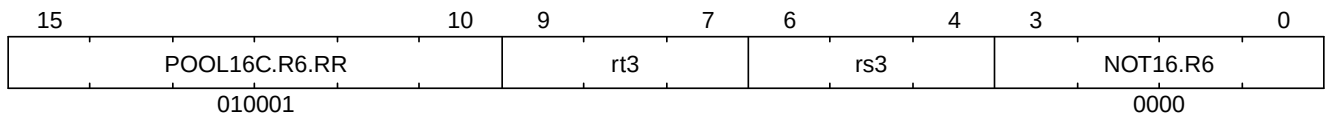


Figure 1172. Encoding (16 bits)

Decode

```
return Not0operands(rt=MICROMIPS_GPR3.decode(rt3), rs=MICROMIPS_GPR3.decode(rs3))
```


nanoMIPS

not.16

Assembly

NOT {rt:gpr3}, {rs:gpr3}

Availability

true

Decode priority

0

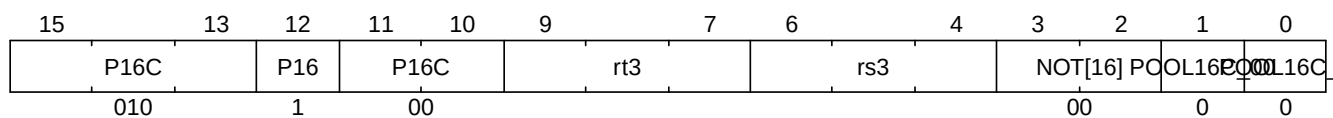


Figure 1173. Encoding (16 bits)

Decode

```
del ctx, exec
return NotOperands(rt=GPR3.decode(rt3), rs=GPR3.decode(rs3))
```

8.426.3. Operation

```
ctx.GPR[operands.rt] = ~ctx.GPR[operands.rs]
```

8.426.4. Description

NOT[16] complements the full implemented-width source GPR.

8.426.5. Exceptions

None.

8.427. OR

8.427.1. Purpose

Compute the logical OR of two GPRs and place it in the destination GPR.

8.427.2. Formats

MIPS

or

Assembly

OR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

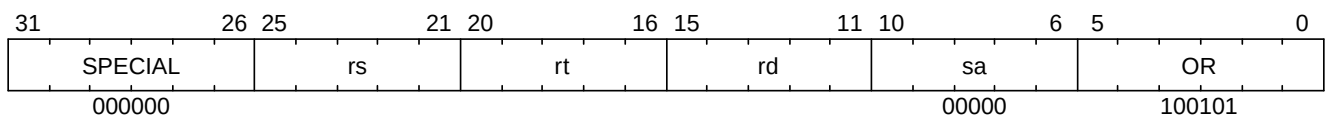


Figure 1174. Encoding (32 bits)

Decode

```
del ctx, exec
return OrOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

or.32

Assembly

OR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

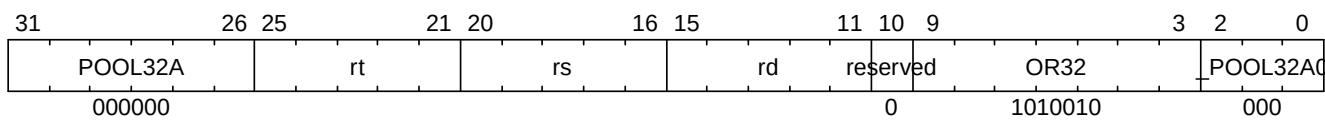


Figure 1175. Encoding (32 bits)

Decode

```
return OrOperands(rd=rd, rs=rs, rt=rt)
```

or.16

Assembly

OR {rd:gpr3}, {rs:gpr3}

Availability

MMAR < R6

Decode priority

0

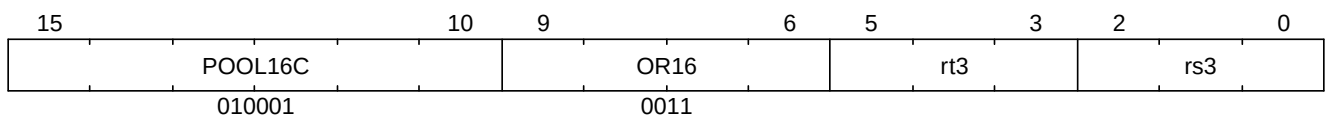


Figure 1176. Encoding (16 bits)

Decode

```
rt = MICROMIPS_GPR3.decode(rt3)
rs = MICROMIPS_GPR3.decode(rs3)
return OrOperands(rd=rt, rs=rs, rt=rt)
```

or.16.r6

Assembly

OR {rd:gpr3}, {rs:gpr3}

Availability

MMAR >= R6

Decode priority

0

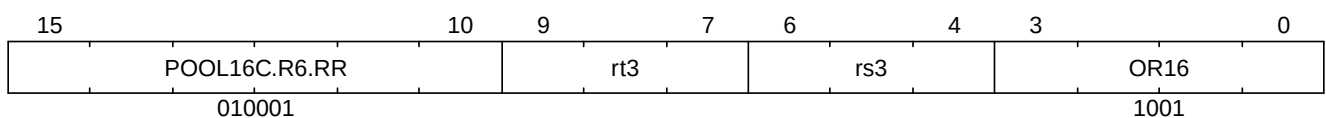


Figure 1177. Encoding (16 bits)

Decode

```
rt = MICROMIPS_GPR3.decode(rt3)
rs = MICROMIPS_GPR3.decode(rs3)
return OrOperands(rd=rt, rs=rs, rt=rt)
```

nanoMIPS

or.32

Assembly

OR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

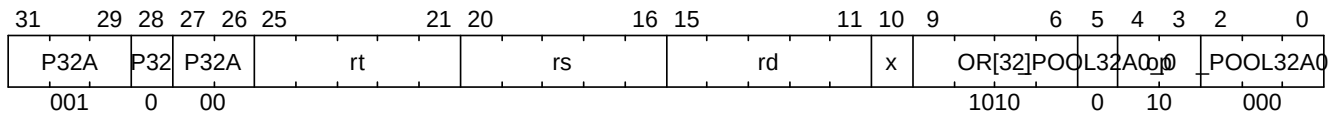


Figure 1178. Encoding (32 bits)

Decode

```
return OrOperands(rd=rd, rs=rs, rt=rt)
```

or.16

Assembly

OR {rd:gpr3}, {rs:gpr3}

Availability

true

Decode priority

0

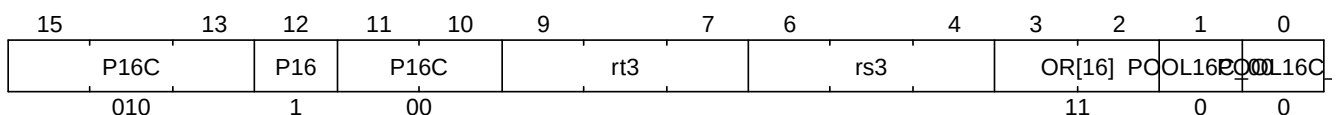


Figure 1179. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
return OrOperands(rd=rt, rs=rs, rt=rt)
```

8.427.3. Operation

```
ctx.GPR[operands.rd] = ctx.GPR[operands.rs] | ctx.GPR[operands.rt]
```

8.427.4. Description

The nanoMIPS[32] form names an independent destination. The compact nanoMIPS[16] form aliases its encoded rt3 register as both source and destination. The operation is performed across the full implemented GPR width.

8.427.5. Exceptions

None.

8.428. ORI

8.428.1. Purpose

Compute the logical OR of a GPR and an unsigned immediate.

8.428.2. Formats

MIPS

ori

Assembly

ORI {rt:gpr}, {rs:gpr}, {imm:u16}

Availability

true

Decode priority

0

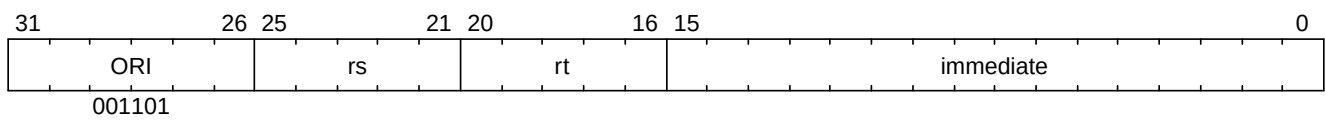


Figure 1180. Encoding (32 bits)

Decode

```
del ctx, exec
return Ori0perands(rt=rt, rs=rs, imm=zero_extend(immediate, from_nbits=16))
```

microMIPS

ori.32

Assembly

ORI {rt:gpr}, {rs:gpr}, {imm:u16}

Availability

true

Decode priority

0

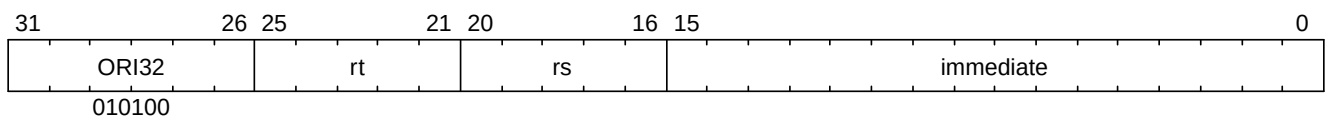


Figure 1181. Encoding (32 bits)

Decode

```
del ctx, exec
return Ori0Operands(rt=rt, rs=rs, imm=zero_extend(immediate, from_nbits=16))
```

nanoMIPS

ori.32

Assembly

ORI {rt:gpr}, {rs:gpr}, {imm:u12}

Availability

true

Decode priority

0

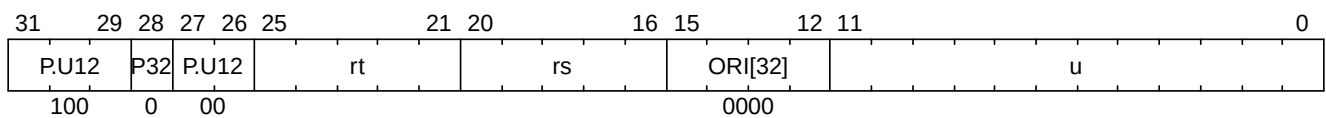


Figure 1182. Encoding (32 bits)

Decode

```
return Ori0Operands(rt=rt, rs=rs, imm=zero_extend(u, from_nbits=12))
```

8.428.3. Operation

```
ctx.GPR[operands.rt] = ctx.GPR[operands.rs] | int(operands.imm)
```

8.428.4. Description

The immediate is zero-extended and ORed with the full-width source GPR.

8.428.5. Exceptions

None.

8.429. PACKRL.PH

8.429.1. Purpose

Pack the right halfword of *rs* with the left halfword of *rt*.

8.429.2. Formats

MIPS

packrl.ph

Assembly

PACKRL.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

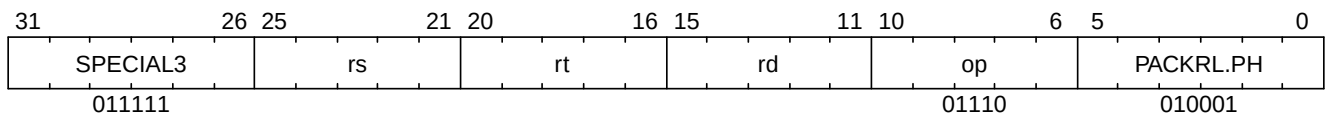


Figure 1183. Encoding (32 bits)

Decode

```
return DspPackBinary0perands(rd=rd, rs=rs, rt=rt)
```

microMIPS

packrl.ph.32

Assembly

PACKRL.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

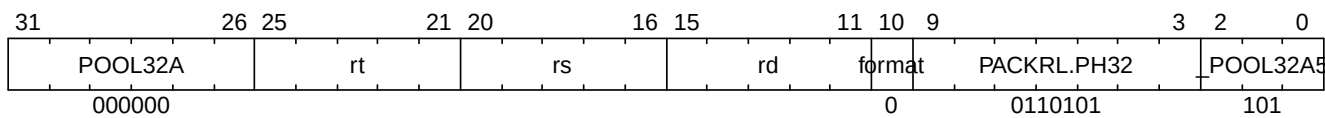


Figure 1184. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

packr1.ph.32

Assembly

PACKRL.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

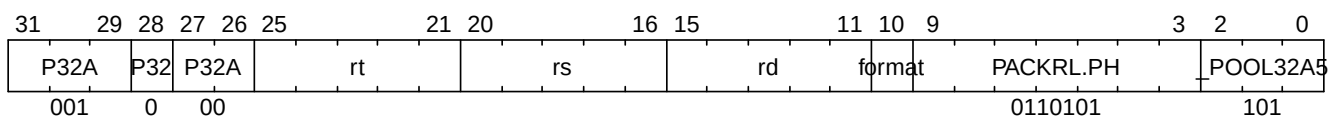


Figure 1185. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

8.429.3. Operation

```
validate_access_to_dsp_resources(ctx)
left = zero_extend(ctx.GPR[operands.rs], from_nbits=16)
right = (ctx.GPR[operands.rt] >> 16) & 0xFFFF
ctx.GPR[operands.rd] = sign_extend((left << 16) | right, from_nbits=32)
```

8.429.4. Description

The operation rearranges packed fields in the low word and sign-extends the result to the implemented GPR width.

8.429.5. Exceptions

None.

8.430. PAUSE

8.430.1. Purpose

Pause until the processor load-linked bit is cleared.

8.430.2. Formats

MIPS

pause

Assembly

PAUSE

Availability

true

Decode priority

1

Overrides lower-priority formats

- SLL (s11)

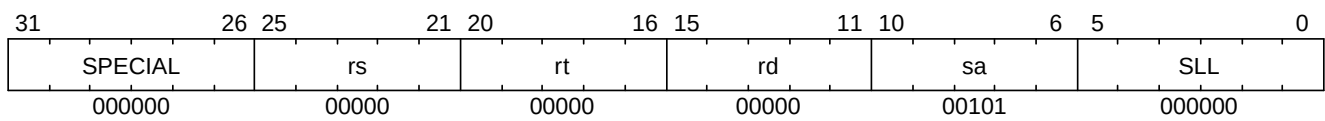


Figure 1186. Encoding (32 bits)

Decode

```
return PauseOperands()
```

nanoMIPS

pause.32

Assembly

PAUSE

Availability

true

Decode priority

1

Overrides lower-priority formats

- SLL (s11.32)

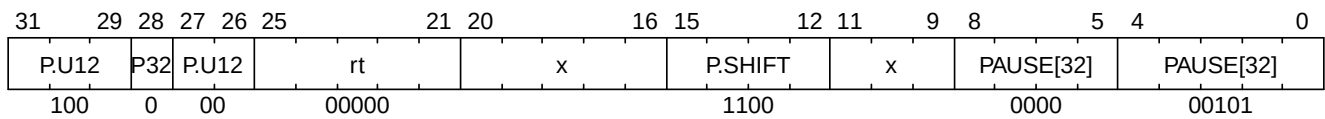


Figure 1187. Encoding (32 bits)

Decode

```
return PauseOperands()
```

8.430.3. Operation

```
if ctx.profile.architecture is Architecture.MIPS:
    raise ReservedInstruction("classic MIPS PAUSE is reserved")
pause_until_llbit_cleared(ctx)
```

8.430.4. Description

PAUSE is encoded as an SLL[32] alias with zero destination and source and shift five. When LLAddr.LLB is set, it enters the context pause state; otherwise it behaves as a no-op.

8.430.5. Exceptions

- **RI** (Reserved Instruction) — classic MIPS PAUSE is reserved

8.431. PICK.PH

8.431.1. Purpose

DSP packed operation PICK.PH.

8.431.2. Formats

MIPS

`pick.ph`

Assembly

`PICK.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

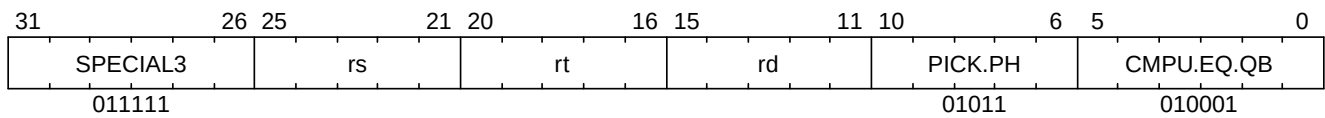


Figure 1188. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`pick.ph.32`

Assembly

`PICK.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

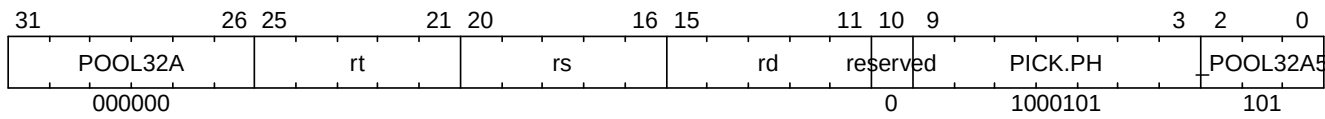


Figure 1189. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

pick.ph.32

Assembly

PICK.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

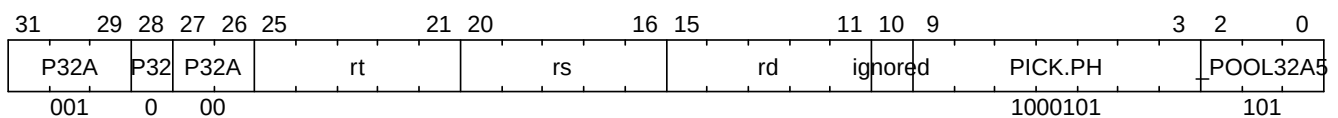


Figure 1190. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.431.3. Operation

```
validate_access_to_dsp_resources(ctx)
left, right = lanes(ctx.GPR[operands.rs], 16, 2), lanes(ctx.GPR[operands.rt], 16, 2)
value = pack_lanes(
    tuple(
        a if (ctx.DSPControl.Ccond >> bit) & 1 else b
        for bit, (a, b) in zip((1, 0), zip(left, right))
    ),
    16,
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.431.4. Description

PICK.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.431.5. Exceptions

None.

8.432. PICK.QB

8.432.1. Purpose

DSP packed operation PICK.QB.

8.432.2. Formats

MIPS

`pick.qb`

Assembly

`PICK.QB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5 && DSP

Decode priority

0

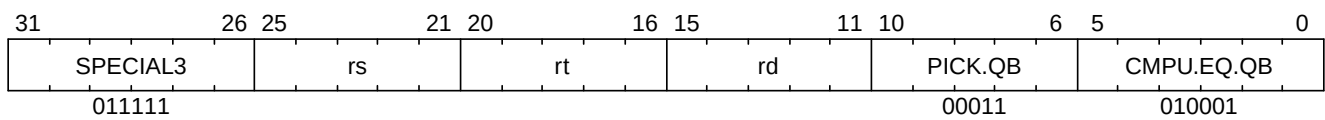


Figure 1191. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`pick.qb.32`

Assembly

`PICK.QB {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

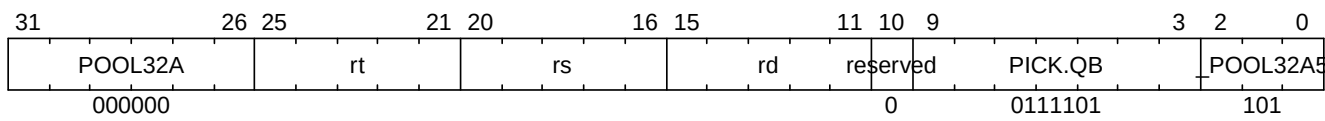


Figure 1192. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

`pick.qb.32`

Assembly

PICK.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

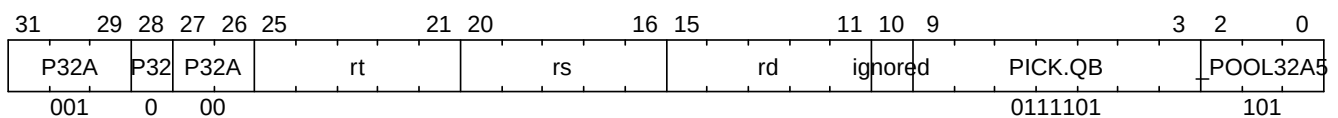


Figure 1193. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.432.3. Operation

```
validate_access_to_dsp_resources(ctx)
left, right = lanes(ctx.GPR[operands.rs], 8, 4), lanes(ctx.GPR[operands.rt], 8, 4)
value = pack_lanes(
    tuple(
        a if (ctx.DSPControl.Ccond >> bit) & 1 else b
        for bit, (a, b) in zip((3, 2, 1, 0), zip(left, right))
    ),
    8,
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.432.4. Description

PICK.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.432.5. Exceptions

None.

8.433. PLL.PS

8.433.1. Purpose

Pair lower lower.

8.433.2. Formats

MIPS

pll.ps

Assembly

PLL.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

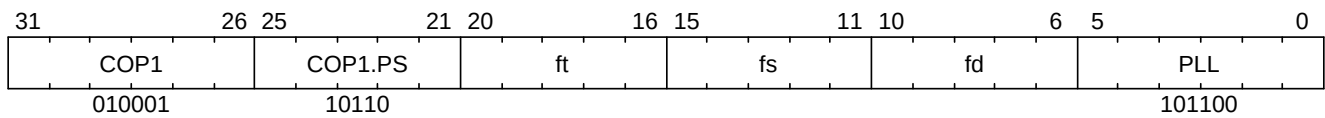


Figure 1194. Encoding (32 bits)

Decode

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

pll.ps.32

Assembly

PLL.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

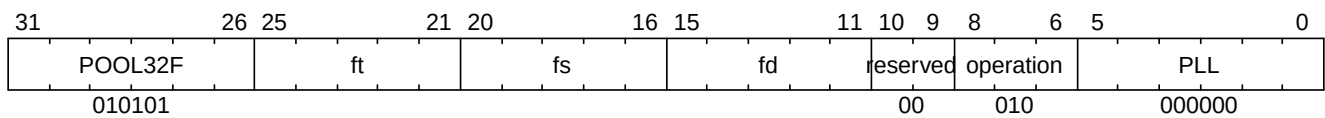


Figure 1195. Encoding (32 bits)


```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.433.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
set_fpr(ctx, operands.fd, ((fs & 0xFFFF_FFFF) << 32) | (ft & 0xFFFF_FFFF), "PS")
```

8.433.4. Description

Merge lower paired-single lanes.

8.433.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.434. PLU.PS

8.434.1. Purpose

Pair lower upper.

8.434.2. Formats

MIPS

plu.ps

Assembly

PLU.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

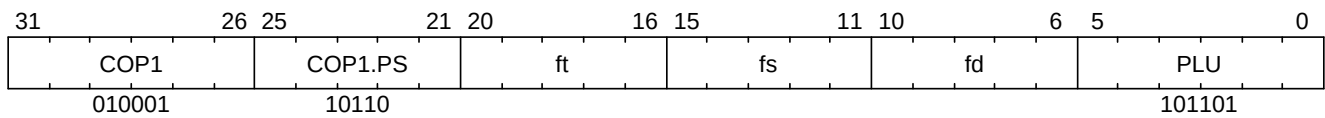


Figure 1196. Encoding (32 bits)

Decode

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

plu.ps.32

Assembly

PLU.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

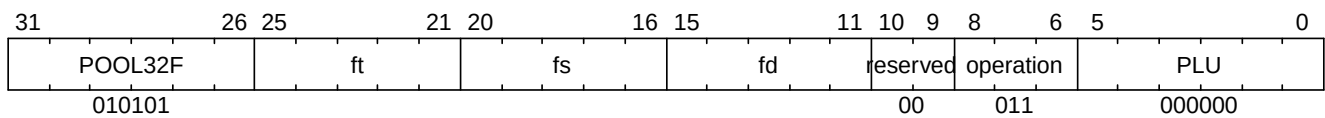


Figure 1197. Encoding (32 bits)

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.434.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
set_fpr(ctx, operands.fd, ((fs & 0xFFFF_FFFF) << 32) | (ft >> 32), "PS")
```

8.434.4. Description

Merge a lower and upper paired-single lane.

8.434.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.435. PRECEQ.W.PHL

8.435.1. Purpose

Expand the left Q15 halfword to Q31.

8.435.2. Formats

MIPS

preceq.w.phl

Assembly

PRECEQ.W.PHL {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

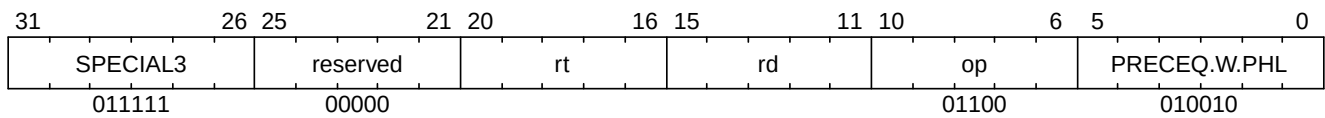


Figure 1198. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

preceq.w.phl.32

Assembly

PRECEQ.W.PHL {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

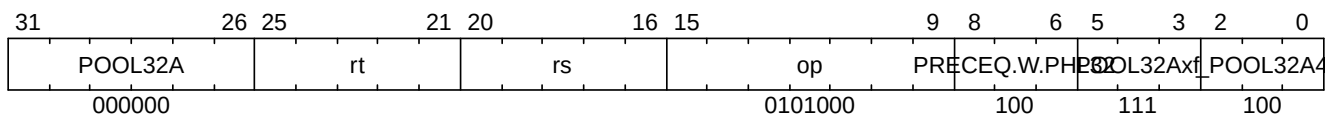


Figure 1199. Encoding (32 bits)

Decode

```
return DspPackUnary0operands(rd=rt, rt=rs)
```

nanoMIPS

preceq.w.phl.32

Assembly

PRECEQ.W.PHL {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

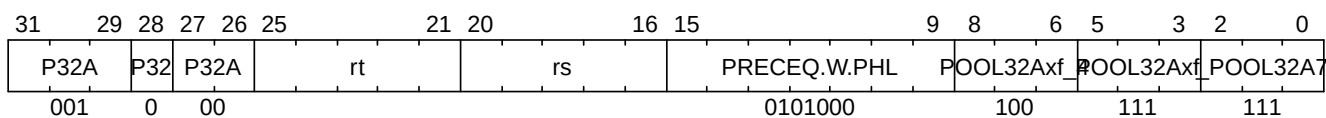


Figure 1200. Encoding (32 bits)

Decode

```
return DspPackUnary0operands(rd=rt, rt=rs)
```

8.435.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = (ctx.GPR[operands.rt] >> 16) & 0xFFFF
ctx.GPR[operands.rd] = sign_extend(source << 16, from_nbits=32)
```

8.435.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.435.5. Exceptions

None.

8.436. PRECEQ.W.PHR

8.436.1. Purpose

Expand the right Q15 halfword to Q31.

8.436.2. Formats

MIPS

preceq.w.phr

Assembly

PRECEQ.W.PHR {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

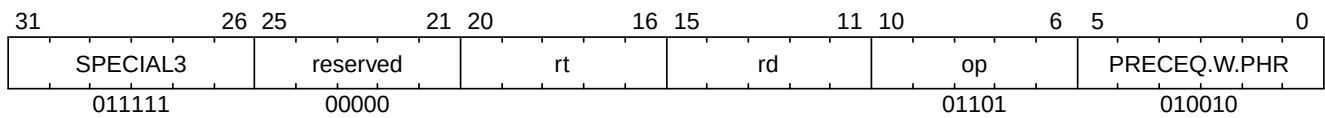


Figure 1201. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

preceq.w.phr.32

Assembly

PRECEQ.W.PHR {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

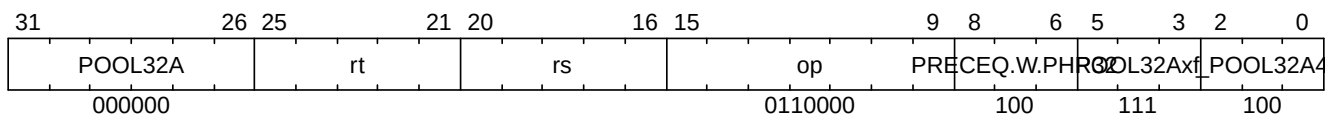


Figure 1202. Encoding (32 bits)

Decode

```
return DspPackUnary0operands(rd=rt, rt=rs)
```

nanoMIPS

preceq.w.phr.32

Assembly

PRECEQ.W.PHR {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

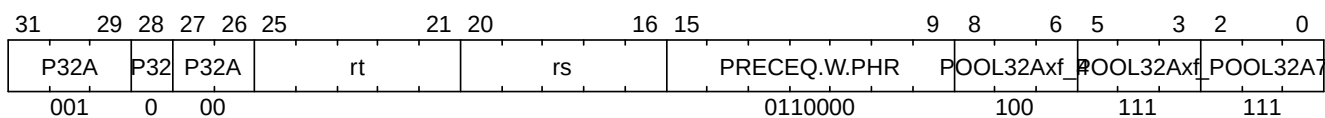


Figure 1203. Encoding (32 bits)

Decode

```
return DspPackUnary0operands(rd=rt, rt=rs)
```

8.436.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt] & 0xFFFF
ctx.GPR[operands.rd] = sign_extend(source << 16, from_nbits=32)
```

8.436.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.436.5. Exceptions

None.

8.437. PRECEQU.PH.QBL

8.437.1. Purpose

Expand the left two unsigned bytes to Q15 halfwords.

8.437.2. Formats

MIPS

precequ.ph.qbl

Assembly

PRECEQU.PH.QBL {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

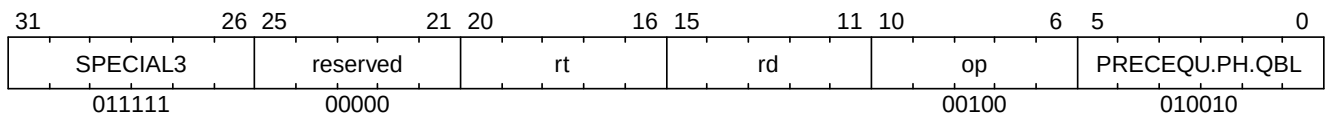


Figure 1204. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

precequ.ph.qbl.32

Assembly

PRECEQU.PH.QBL {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

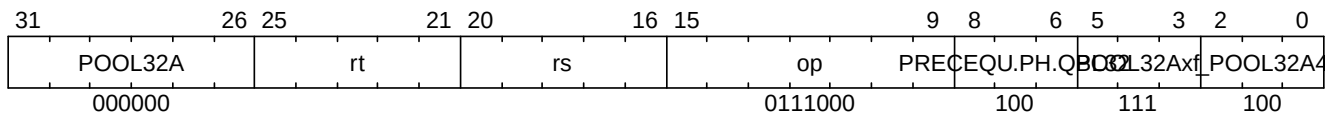


Figure 1205. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

precequ.ph.qbl.32

Assembly

PRECEQU.PH.QBL {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

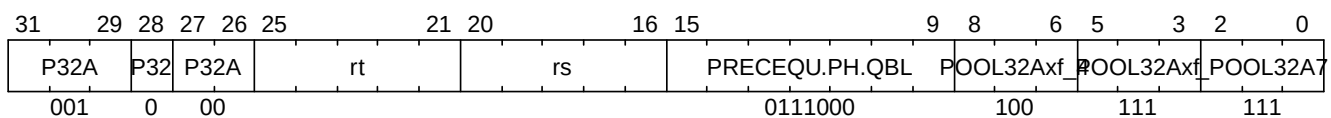


Figure 1206. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.437.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 24) & 0xFF) << 23) | (((source >> 16) & 0xFF) << 7)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.437.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.437.5. Exceptions

None.

8.438. PRECEQU.PH.QBLA

8.438.1. Purpose

Expand alternate left unsigned bytes to Q15 halfwords.

8.438.2. Formats

MIPS

precequ.ph.qbla

Assembly

PRECEQU.PH.QBLA {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

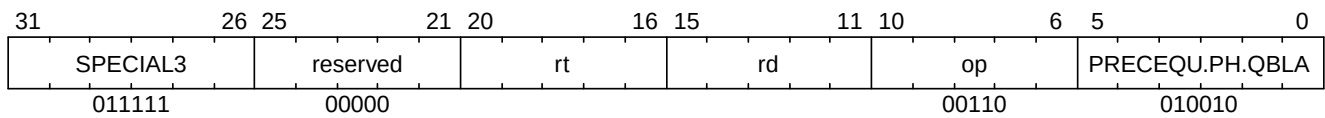


Figure 1207. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

precequ.ph.qbla.32

Assembly

PRECEQU.PH.QBLA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

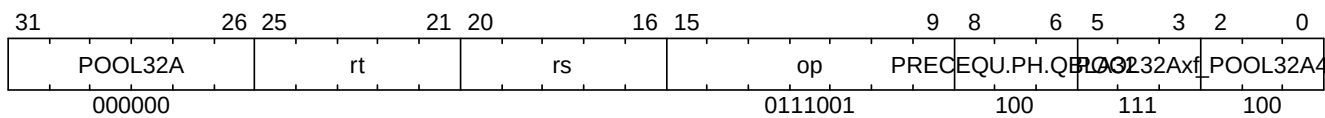


Figure 1208. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

precequ.ph.qbla.32

Assembly

PRECEQU.PH.QBLA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

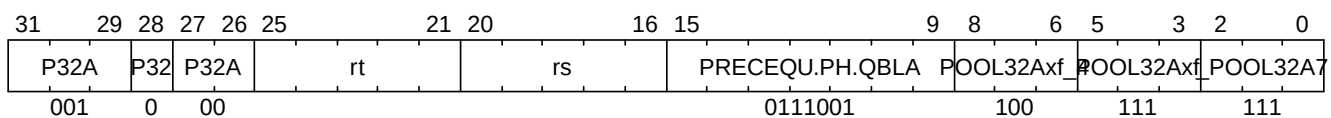


Figure 1209. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.438.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 24) & 0xFF) << 23) | (((source >> 8) & 0xFF) << 7)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.438.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.438.5. Exceptions

None.

8.439. PRECEQU.PH.QBR

8.439.1. Purpose

Expand the right two unsigned bytes to Q15 halfwords.

8.439.2. Formats

MIPS

precequ.ph.qbr

Assembly

PRECEQU.PH.QBR {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

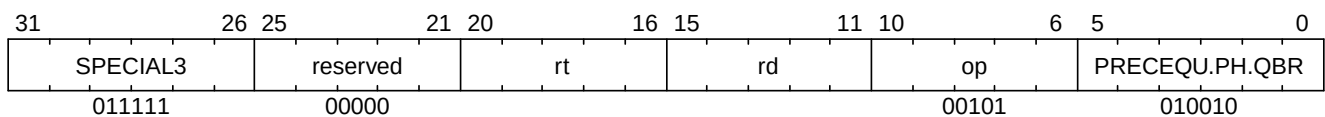


Figure 1210. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

precequ.ph.qbr.32

Assembly

PRECEQU.PH.QBR {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

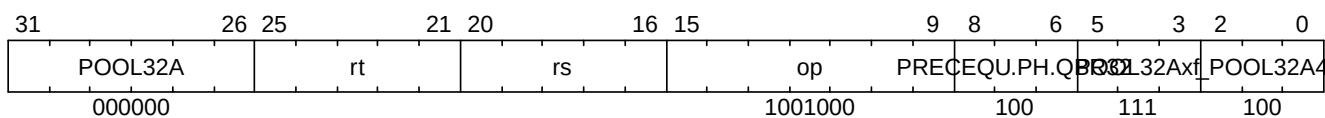


Figure 1211. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

precequ.ph.qbr.32

Assembly

PRECEQU.PH.QBR {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

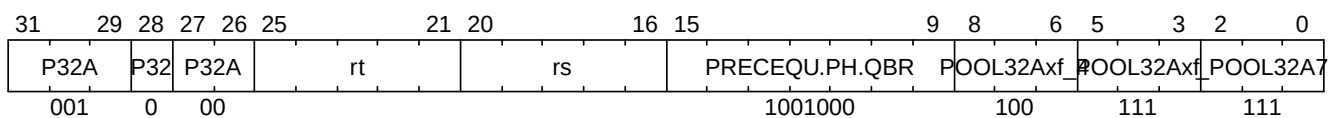


Figure 1212. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.439.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 8) & 0xFF) << 23) | ((source & 0xFF) << 7)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.439.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.439.5. Exceptions

None.

8.440. PRECEQU.PH.QBRA

8.440.1. Purpose

Expand alternate right unsigned bytes to Q15 halfwords.

8.440.2. Formats

MIPS

precequ.ph.qbra

Assembly

PRECEQU.PH.QBRA {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

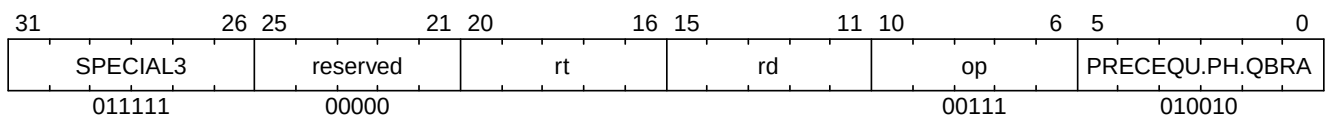


Figure 1213. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

precequ.ph.qbra.32

Assembly

PRECEQU.PH.QBRA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

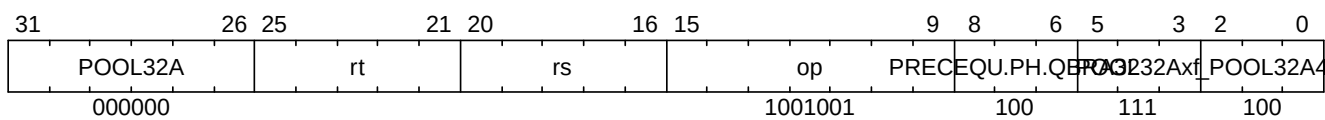


Figure 1214. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

precequ.ph.qbra.32

Assembly

PRECEQU.PH.QBRA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

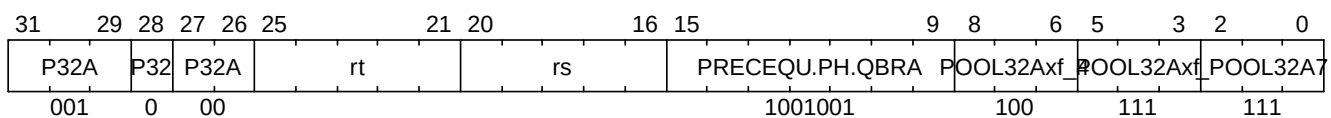


Figure 1215. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.440.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 16) & 0xFF) << 23) | ((source & 0xFF) << 7)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.440.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.440.5. Exceptions

None.

8.441. PRECEU.PH.QBL

8.441.1. Purpose

Expand the left two unsigned bytes to halfwords.

8.441.2. Formats

MIPS

preceu.ph.qbl

Assembly

PRECEU.PH.QBL {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

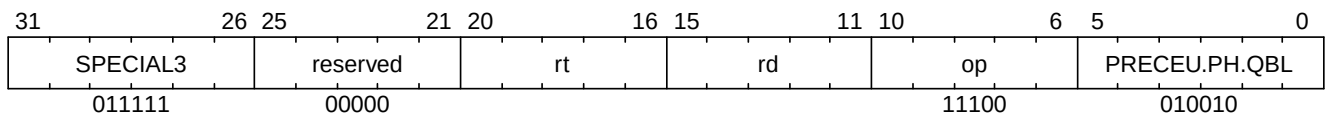


Figure 1216. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

preceu.ph.qbl.32

Assembly

PRECEU.PH.QBL {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

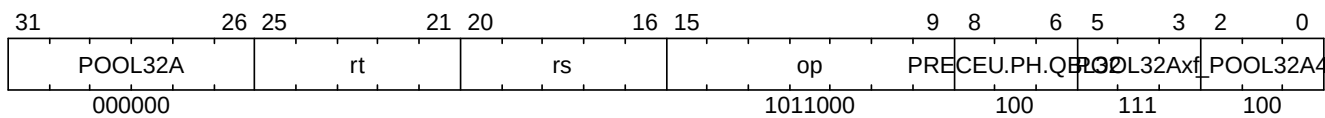


Figure 1217. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

preceu.ph.qbl.32

Assembly

PRECEU.PH.QBL {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

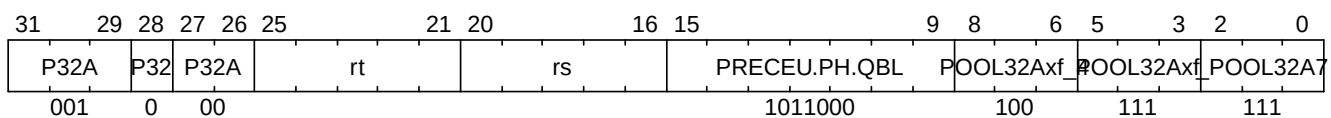


Figure 1218. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.441.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 24) & 0xFF) << 16) | ((source >> 16) & 0xFF)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.441.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.441.5. Exceptions

None.

8.442. PRECEU.PH.QBLA

8.442.1. Purpose

Expand alternate left unsigned bytes to halfwords.

8.442.2. Formats

MIPS

preceu.ph.qbla

Assembly

PRECEU.PH.QBLA {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

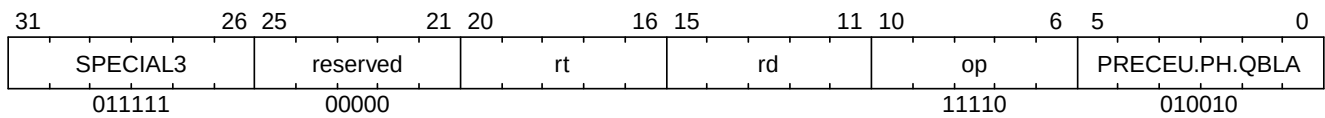


Figure 1219. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

preceu.ph.qbla.32

Assembly

PRECEU.PH.QBLA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

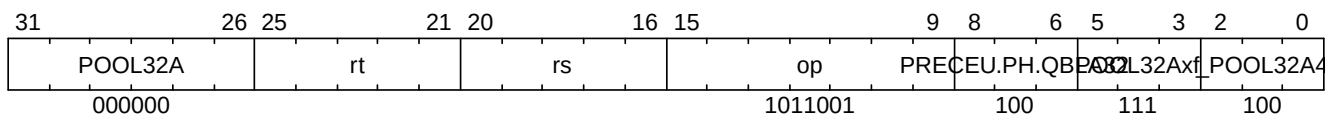


Figure 1220. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

preceu.ph.qbla.32

Assembly

PRECEU.PH.QBLA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

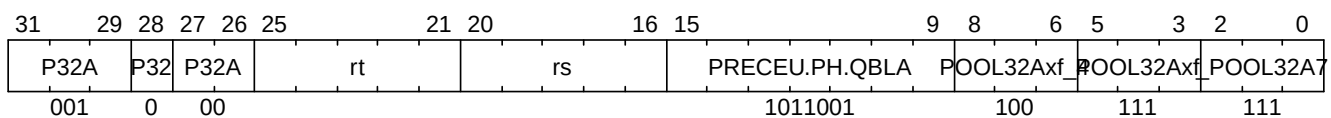


Figure 1221. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.442.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 24) & 0xFF) << 16) | ((source >> 8) & 0xFF)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.442.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.442.5. Exceptions

None.

8.443. PRECEU.PH.QBR

8.443.1. Purpose

Expand the right two unsigned bytes to halfwords.

8.443.2. Formats

MIPS

preceu.ph.qbr

Assembly

PRECEU.PH.QBR {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

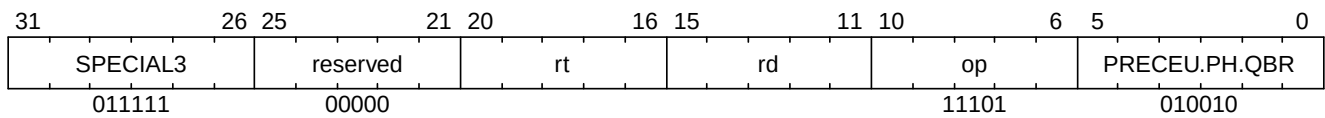


Figure 1222. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

preceu.ph.qbr.32

Assembly

PRECEU.PH.QBR {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

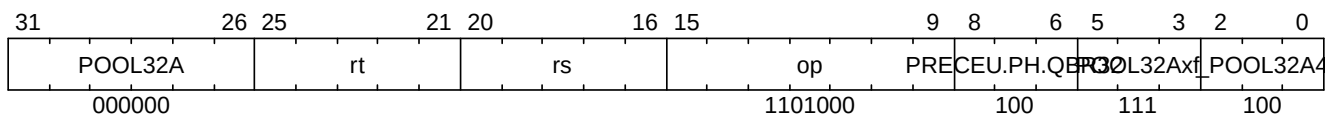


Figure 1223. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

preceu.ph.qbr.32

Assembly

PRECEU.PH.QBR {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

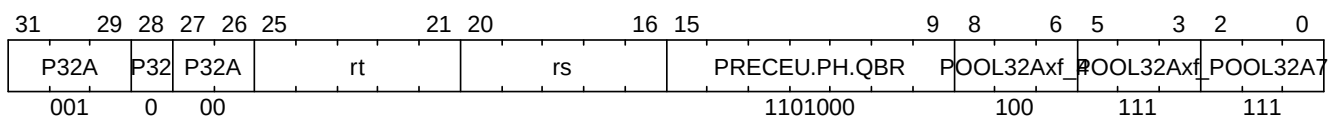


Figure 1224. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.443.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 8) & 0xFF) << 16) | (source & 0xFF)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.443.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.443.5. Exceptions

None.

8.444. PRECEU.PH.QBRA

8.444.1. Purpose

Expand alternate right unsigned bytes to halfwords.

8.444.2. Formats

MIPS

preceu.ph.qbra

Assembly

PRECEU.PH.QBRA {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

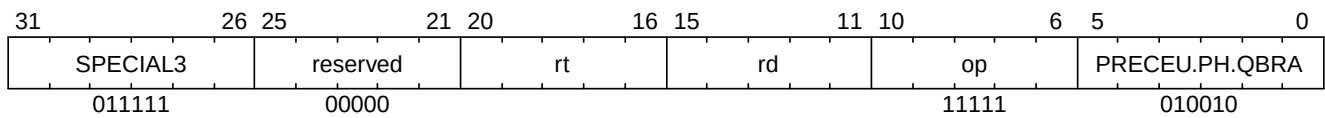


Figure 1225. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

preceu.ph.qbra.32

Assembly

PRECEU.PH.QBRA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

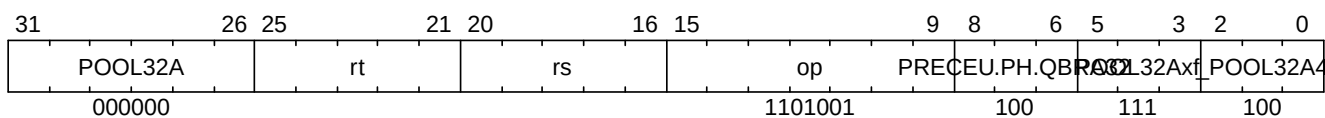


Figure 1226. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

nanoMIPS

preceu.ph.qbra.32

Assembly

PRECEU.PH.QBRA {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

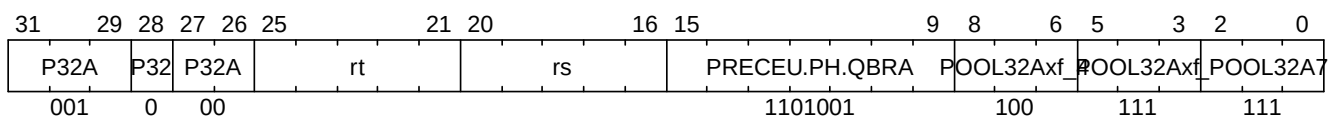


Figure 1227. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rt, rt=rs)
```

8.444.3. Operation

```
validate_access_to_dsp_resources(ctx)
source = ctx.GPR[operands.rt]
result = (((source >> 16) & 0xFF) << 16) | (source & 0xFF)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.444.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.444.5. Exceptions

None.

8.445. PRECR.QB.PH

8.445.1. Purpose

Reduce four integer halfwords to bytes.

8.445.2. Formats

MIPS

`precr.qb.ph`

Assembly

`PRECR.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP2 && AR >= R2_3_5

Decode priority

0

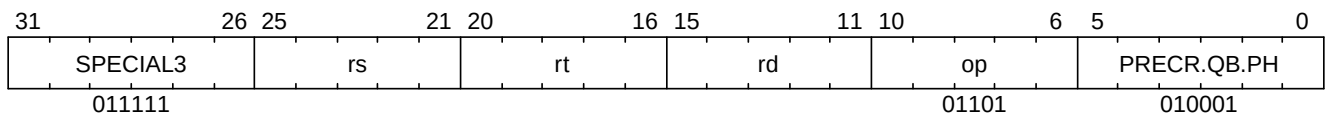


Figure 1228. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`precr.qb.ph.32`

Assembly

`PRECR.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP2

Decode priority

0

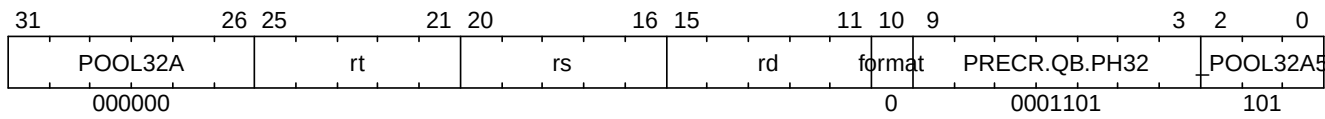


Figure 1229. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

precr.qb.ph.32

Assembly

PRECR.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

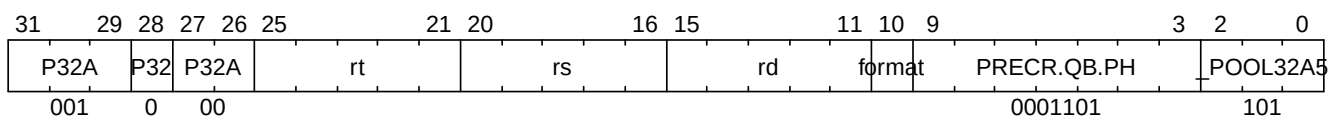


Figure 1230. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

8.445.3. Operation

```
validate_access_to_dsp2_resources(ctx)
left = ctx.GPR[operands.rs]
right = ctx.GPR[operands.rt]
result = ((left >> 16) & 0xFF) << 24
result |= (left & 0xFF) << 16
result |= ((right >> 16) & 0xFF) << 8
result |= right & 0xFF
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.445.4. Description

The operation rearranges packed fields in the low word and sign-extends the result to the implemented GPR width.

8.445.5. Exceptions

None.

8.446. PRECR_SRA.PH.W

8.446.1. Purpose

Shift two words and reduce them to packed halfwords.

8.446.2. Formats

MIPS

`precr_sra.ph.w`

Assembly

`PRECR_SRA.PH.W {rt:gpr}, {rs:gpr}, {sa:u5}`

Availability

DSP2 && AR >= R2_3_5

Decode priority

0

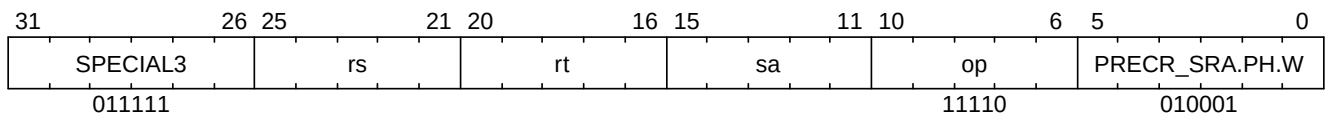


Figure 1231. Encoding (32 bits)

Decode

```
return DspPrecrShift0perands(rt=rt, rs=rs, sa=sa)
```

microMIPS

`precr_sra.ph.w.32`

Assembly

`PRECR_SRA.PH.W {rt:gpr}, {rs:gpr}, {sa:u5}`

Availability

DSP2

Decode priority

0

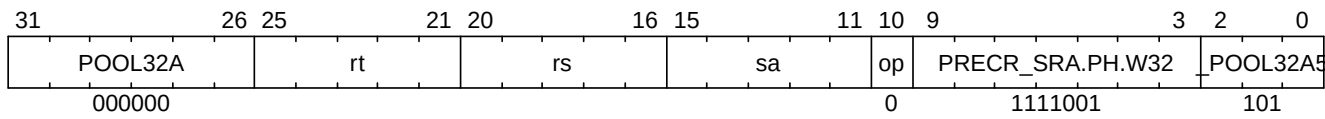


Figure 1232. Encoding (32 bits)

Decode

```
return DspPrecrShiftOperands(rt=rt, rs=rs, sa=sa)
```

nanoMIPS

precr_sra.ph.w.32

Assembly

PRECR_SRA.PH.W {rt:gpr}, {rs:gpr}, {sa:u5}

Availability

DSP2

Decode priority

0

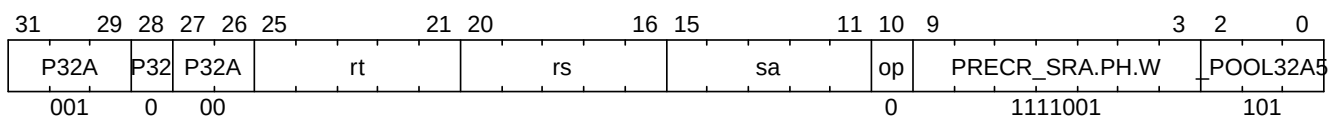


Figure 1233. Encoding (32 bits)

Decode

```
return DspPrecrShiftOperands(rt=rt, rs=rs, sa=sa)
```

8.446.3. Operation

```
validate_access_to_dsp2_resources(ctx)
left = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
right = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
if operands.sa != 0:
    left >>= operands.sa
    right >>= operands.sa
ctx.GPR[operands.rt] = sign_extend(
    ((left & 0xFFFF) << 16) | (right & 0xFFFF), from_nbits=32
)
```

8.446.4. Description

The destination is also the left input; the optional rounded form adds at the most-significant discarded bit.

8.446.5. Exceptions

None.

8.447. PRECR_SRA_R.PH.W

8.447.1. Purpose

Shift two words and reduce them to packed halfwords.

8.447.2. Formats

MIPS

`precr_sra_r.ph.w`

Assembly

`PRECR_SRA_R.PH.W {rt:gpr}, {rs:gpr}, {sa:u5}`

Availability

DSP2 && AR >= R2_3_5

Decode priority

0

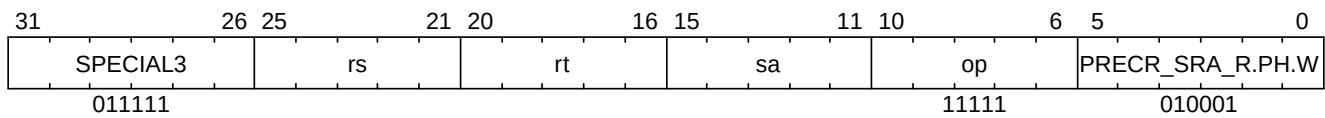


Figure 1234. Encoding (32 bits)

Decode

```
return DspPrecrShift0perands(rt=rt, rs=rs, sa=sa)
```

microMIPS

`precr_sra_r.ph.w.32`

Assembly

`PRECR_SRA_R.PH.W {rt:gpr}, {rs:gpr}, {sa:u5}`

Availability

DSP2

Decode priority

0

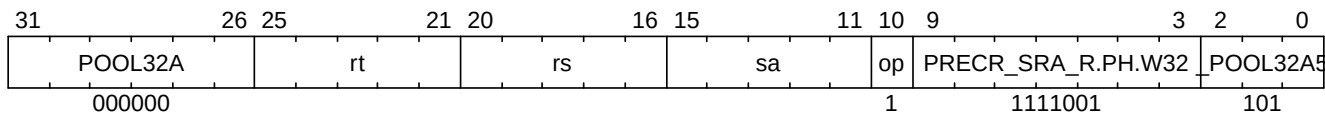


Figure 1235. Encoding (32 bits)

Decode

```
return DspPrecrShiftOperands(rt=rt, rs=rs, sa=sa)
```

nanoMIPS

precr_sra_r.ph.w.32

Assembly

PRECR_SRA_R.PH.W {rt:gpr}, {rs:gpr}, {sa:u5}

Availability

DSP2

Decode priority

0

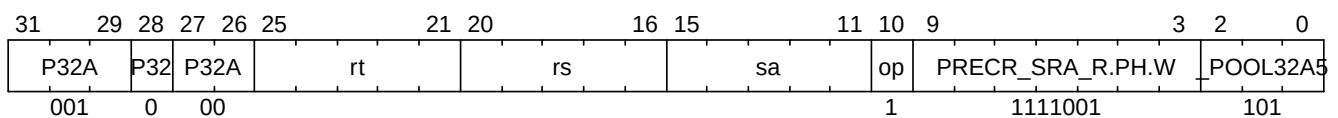


Figure 1236. Encoding (32 bits)

Decode

```
return DspPrecrShiftOperands(rt=rt, rs=rs, sa=sa)
```

8.447.3. Operation

```
validate_access_to_dsp2_resources(ctx)
left = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
right = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
if operands.sa != 0:
    left = (left + (1 << (operands.sa - 1))) >> operands.sa
    right = (right + (1 << (operands.sa - 1))) >> operands.sa
ctx.GPR[operands.rt] = sign_extend(
    ((left & 0xFFFF) << 16) | (right & 0xFFFF), from_nbits=32
)
```

8.447.4. Description

The destination is also the left input; the optional rounded form adds at the most-significant discarded bit.

8.447.5. Exceptions

None.

8.448. PRECRQ.PH.W

8.448.1. Purpose

Reduce two Q31 words to Q15 halfwords.

8.448.2. Formats

MIPS

`precq.ph.w`

Assembly

`PRECQ.PH.W {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP && AR >= R2_3_5

Decode priority

0

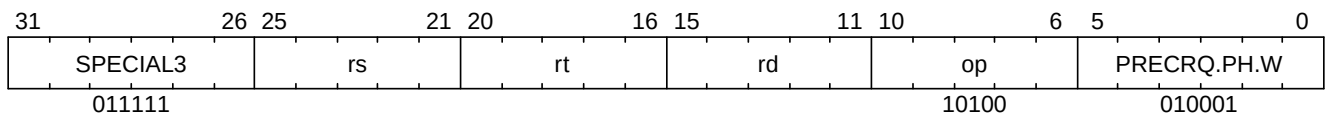


Figure 1237. Encoding (32 bits)

Decode

```
return DspPackBinary0perands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`precq.ph.w.32`

Assembly

`PRECQ.PH.W {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

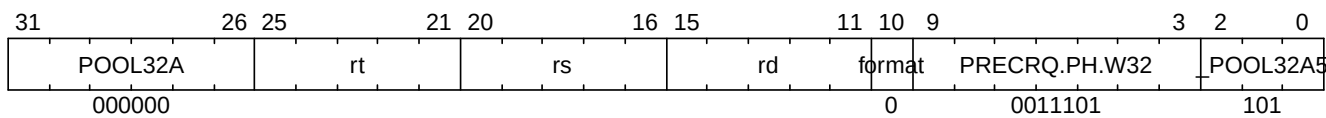


Figure 1238. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

precq.ph.w.32

Assembly

PRECQ.PH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

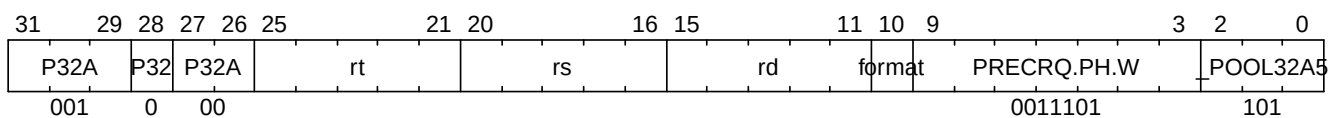


Figure 1239. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

8.448.3. Operation

```
validate_access_to_dsp_resources(ctx)
left = (ctx.GPR[operands.rs] >> 16) & 0xFFFF
right = (ctx.GPR[operands.rt] >> 16) & 0xFFFF
ctx.GPR[operands.rd] = sign_extend((left << 16) | right, from_nbits=32)
```

8.448.4. Description

The operation rearranges packed fields in the low word and sign-extends the result to the implemented GPR width.

8.448.5. Exceptions

None.

8.449. PRECRQ.QB.PH

8.449.1. Purpose

Reduce four Q15 halfwords to Q7 bytes.

8.449.2. Formats

MIPS

`precrq.qb.ph`

Assembly

`PRECRQ.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP && AR >= R2_3_5

Decode priority

0

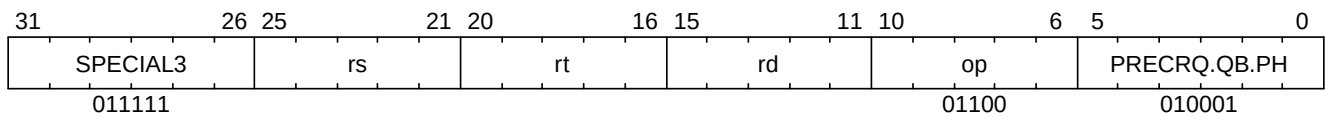


Figure 1240. Encoding (32 bits)

Decode

```
return DspPackBinary0perands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`precrq.qb.ph.32`

Assembly

`PRECRQ.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

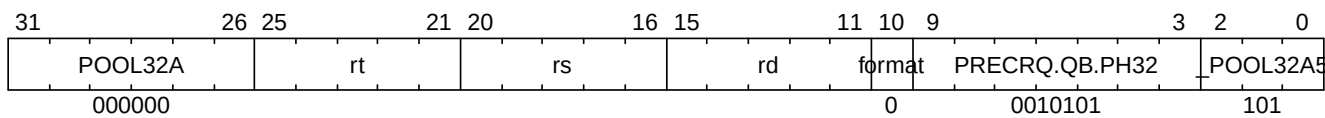


Figure 1241. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

precrq.qb.ph.32

Assembly

PRECRQ.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

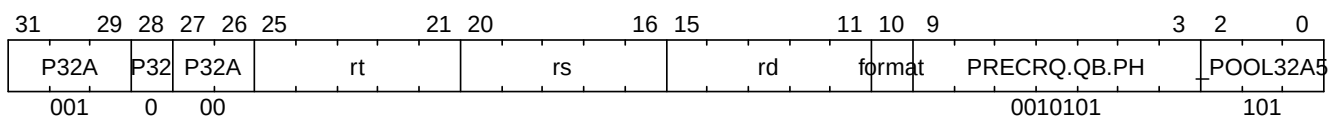


Figure 1242. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

8.449.3. Operation

```
validate_access_to_dsp_resources(ctx)
left = ctx.GPR[operands.rs]
right = ctx.GPR[operands.rt]
result = ((left >> 24) & 0xFF) << 24
result |= ((left >> 8) & 0xFF) << 16
result |= ((right >> 24) & 0xFF) << 8
result |= (right >> 8) & 0xFF
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.449.4. Description

The operation rearranges packed fields in the low word and sign-extends the result to the implemented GPR width.

8.449.5. Exceptions

None.

8.450. PRECRQ_RS.PH.W

8.450.1. Purpose

Round and saturate two Q31 words to Q15 halfwords.

8.450.2. Formats

MIPS

`precrq_rs.ph.w`

Assembly

`PRECQ_RS.PH.W {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP && AR >= R2_3_5

Decode priority

0

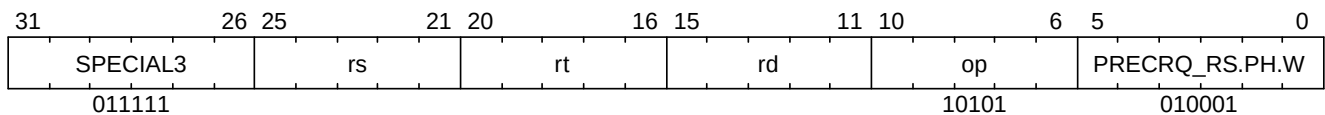


Figure 1243. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`precrq_rs.ph.w.32`

Assembly

`PRECQ_RS.PH.W {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

DSP

Decode priority

0

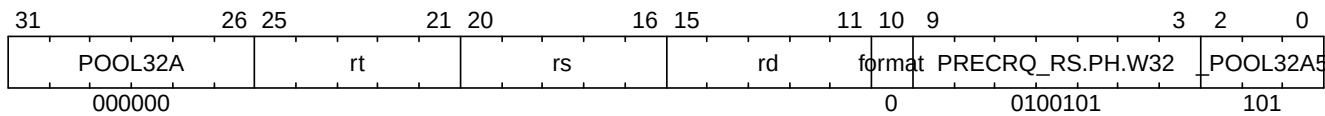


Figure 1244. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

prec_rq.rs.ph.w.32

Assembly

PREC_RQ_RS.PH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

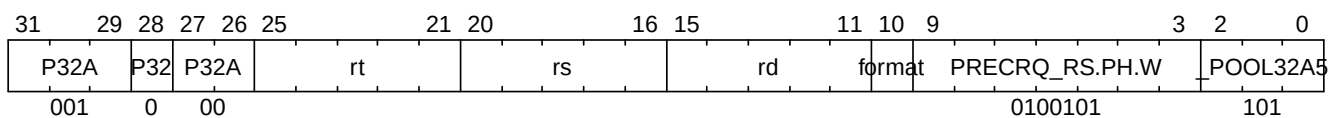


Figure 1245. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

8.450.3. Operation

```
validate_access_to_dsp_resources(ctx)
left = sign_extend(ctx.GPR[operands.rs], from_nbits=32) + 0x8000
right = sign_extend(ctx.GPR[operands.rt], from_nbits=32) + 0x8000
if left > 0x7FFF_FFFF:
    left = 0x7FFF_FFFF
    ctx.DSPControl.Ouflag |= 0x40
if right > 0x7FFF_FFFF:
    right = 0x7FFF_FFFF
    ctx.DSPControl.Ouflag |= 0x40
result = ((left >> 16) & 0xFFFF) << 16
result |= (right >> 16) & 0xFFFF
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.450.4. Description

The operation rearranges packed fields in the low word and sign-extends the result to the implemented GPR width.

8.450.5. Exceptions

None.

8.451. PRECRQU_S.QB.PH

8.451.1. Purpose

Saturate and reduce four Q15 halfwords to unsigned bytes.

8.451.2. Formats

MIPS

precrqu_s.qb.ph

Assembly

PRECRQU_S.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

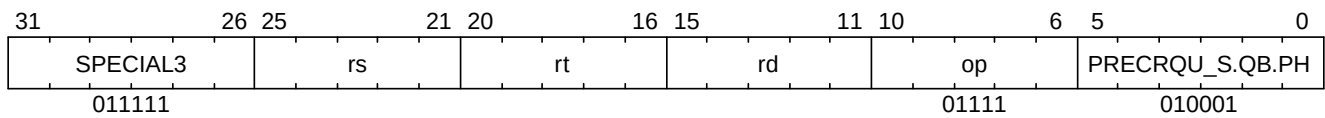


Figure 1246. Encoding (32 bits)

Decode

```
return DspPackBinary0perands(rd=rd, rs=rs, rt=rt)
```

microMIPS

precrqu_s.qb.ph.32

Assembly

PRECRQU_S.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

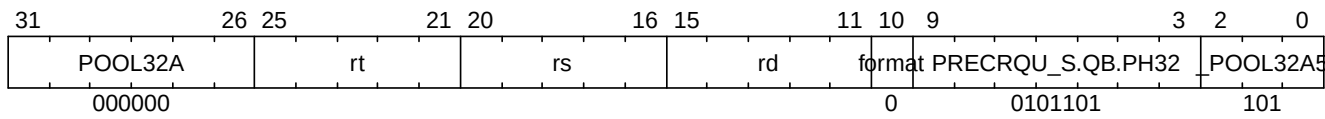


Figure 1247. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

precrcu_s.qb.ph.32

Assembly

PRECRCU_S.QB.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

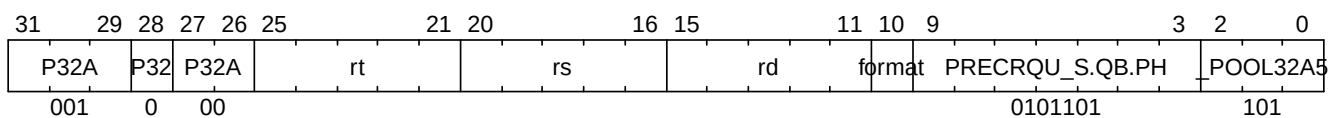


Figure 1248. Encoding (32 bits)

Decode

```
return DspPackBinaryOperands(rd=rd, rs=rs, rt=rt)
```

8.451.3. Operation

```
validate_access_to_dsp_resources(ctx)
result = 0
sources = (
    (ctx.GPR[operands.rs] >> 16) & 0xFFFF,
    ctx.GPR[operands.rs] & 0xFFFF,
    (ctx.GPR[operands.rt] >> 16) & 0xFFFF,
    ctx.GPR[operands.rt] & 0xFFFF,
)
for source in sources:
    signed_source = sign_extend(source, from_nbits=16)
    if signed_source < 0:
        lane = 0
        ctx.DSPControl.Ouflag |= 0x40
    elif source > 0x7F80:
        lane = 0xFF
        ctx.DSPControl.Ouflag |= 0x40
    else:
        lane = source >> 7
    result = (result << 8) | lane
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.451.4. Description

The operation rearranges packed fields in the low word and sign-extends the result to the implemented GPR width.

8.451.5. Exceptions

None.

8.452. PREF

8.452.1. Purpose

Request an implementation-defined prefetch for an address and hint.

8.452.2. Formats

MIPS

`pref`

Assembly

`PREF {hint:u5}, {offset:s16}({rs:gpr})`

Availability

`AR < R6`

Decode priority

0

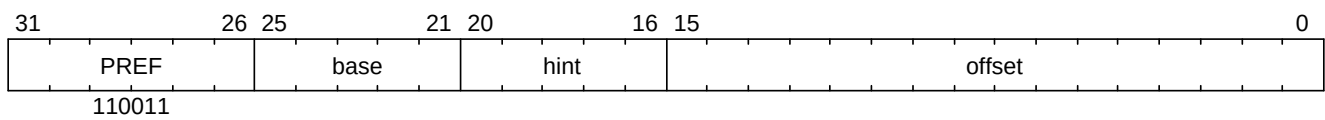


Figure 1249. Encoding (32 bits)

Decode

```
return PrefOperands(  
    hint=zero_extend(hint, from_nbits=5),  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
    eva=False,  
)
```

`pref.r6`

Assembly

`PREF {hint:u5}, {offset:s9}({rs:gpr})`

Availability

`AR >= R6`

Decode priority

0

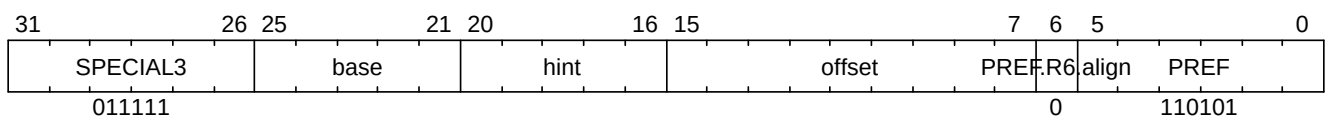


Figure 1250. Encoding (32 bits)

Decode

```
if hint >= 24:
    raise ReservedInstruction("reserved PREF hint in Release 6")
return PrefOperands(
    hint=zero_extend(hint, from_nbits=5),
    rs=base,
    offset=sign_extend(offset, from_nbits=9),
    eva=False,
)
```

microMIPS

pref.32

Assembly

PREF {hint:u5}, {offset:s12}({rs:gpr})

Availability

MMAR < R6

Decode priority

0

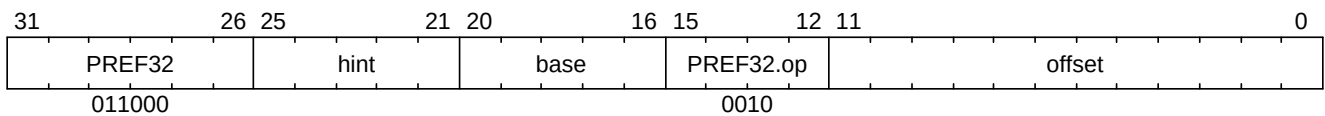


Figure 1251. Encoding (32 bits)

Decode

```
del ctx, exec
return PrefOperands(
    hint=zero_extend(hint, from_nbits=5),
    rs=base,
    offset=sign_extend(offset, from_nbits=12),
    eva=False,
)
```

pref.32.r6

Assembly

PREF {hint:u5}, {offset:s9}({rs:gpr})

Availability

MMAR >= R6

Decode priority

0

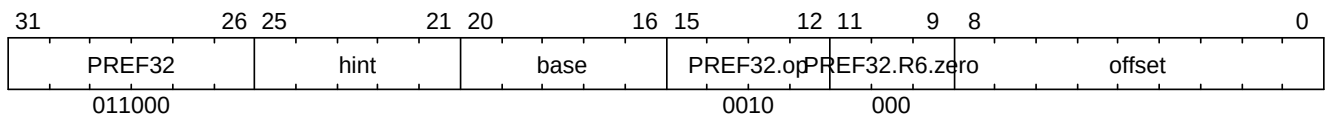


Figure 1252. Encoding (32 bits)

Decode

```
del ctx, exec
if hint >= 24:
    raise ReservedInstruction("reserved PREF hint in Release 6")
return PrefOperands(
    hint=zero_extend(hint, from_nbits=5),
    rs=base,
    offset=sign_extend(offset, from_nbits=9),
    eva=False,
)
```

prefe.32

Assembly

PREF {hint:u5}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

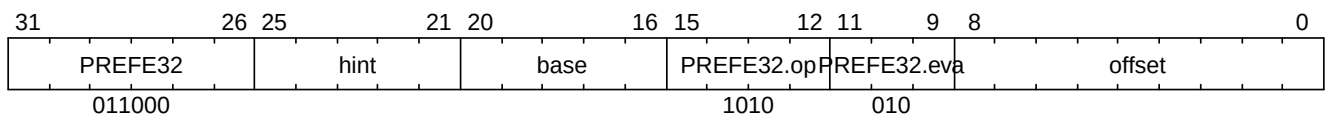


Figure 1253. Encoding (32 bits)

Decode

```
del ctx, exec
return PrefOperands(
    hint=zero_extend(hint, from_nbits=5),
    rs=base,
    offset=sign_extend(offset, from_nbits=9),
    eva=True,
)
```

nanoMIPS

pref.eva

Assembly

PREF {hint:u5}, {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

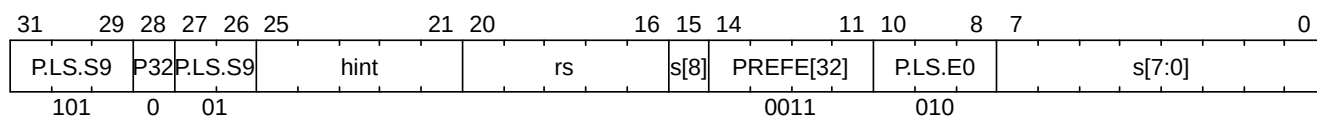


Figure 1254. Encoding (32 bits)

Encoding constraints

- hint != 31

Decode

```
return PrefOperands(  
    hint=zero_extend(hint, from_nbits=5),  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    eva=True,  
)
```

pref.s9

Assembly

PREF {hint:u5}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

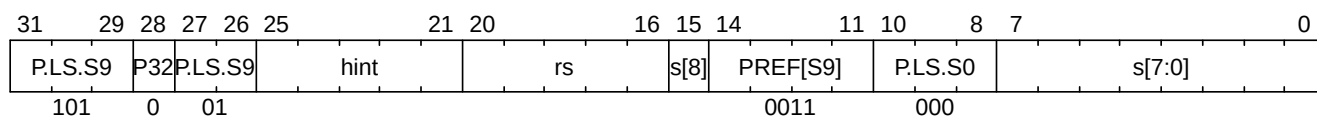


Figure 1255. Encoding (32 bits)

Encoding constraints

- hint != 31

Decode

```
return PrefOperands(  
    hint=zero_extend(hint, from_nbits=5),  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    eva=False,
```

```
)
```

pref.u12

Assembly

PREF {hint:u5}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

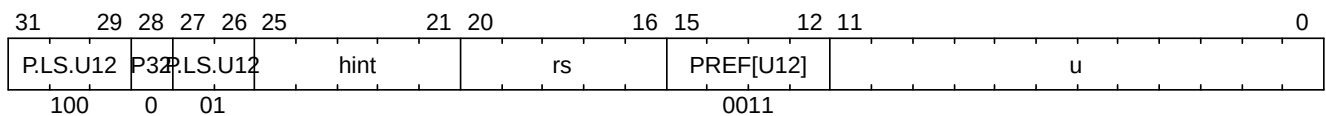


Figure 1256. Encoding (32 bits)

Encoding constraints

- **hint != 31**

Decode

```
return PrefOperands(  
    hint=zero_extend(hint, from_nbits=5),  
    rs=rs,  
    offset=zero_extend(u, from_nbits=12),  
    eva=False,  
)
```

8.452.3. Operation

```
if operands.eva:  
    if not ctx.C0.Config5.EVA:  
        raise ReservedInstruction("PREFE requires Config5.EVA")  
    if not IsCoproprocessor0Enabled(ctx):  
        raise CoprocessorUnusable(0)  
  
# nanoMIPS reserves hints 24..30 as NOPs. Encoding excludes hint 31,  
# which is the separate SYNCI operation.  
if int(operands.hint) >= 24:  
    return  
  
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)  
advisory_prefetch(ctx, base, operands.offset, operands.hint, eva=operands.eva)
```

8.452.4. Description

PREF[S9] and PREF[U12] form an effective address from a GPR and a signed or unsigned byte offset. PREFE[32] uses EVA translation and requires Config5.EVA and usable CP0. The request is delivered to the optional memory-backend prefetch hook; absent a hook, it is a no-op, as permitted for an implementation-dependent performance operation.

8.452.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — PREFE requires Config5.EVA; reserved PREF hint in Release 6

8.453. PREFIX

8.453.1. Purpose

Request a prefetch using a two-GPR indexed address.

8.453.2. Formats

MIPS

prefix

Assembly

PREFIX {hint:u5}, {index:gpr}({base:gpr})

Availability

AR >= R2_3_5 && AR < R6

Decode priority

0

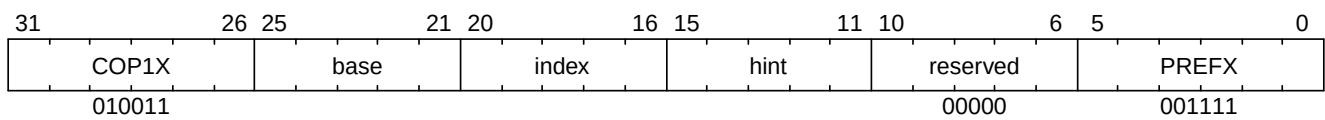


Figure 1257. Encoding (32 bits)

Decode

```
return Prefx0operands(  
    hint=zero_extend(hint, from_nbits=5),  
    base=base,  
    index=index,  
)
```

microMIPS

prefix

Assembly

PREFIX {hint:u5}, {index:gpr}({base:gpr})

Availability

MMAR < R6

Decode priority

0

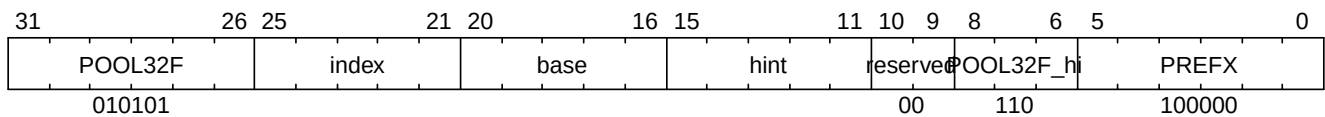


Figure 1258. Encoding (32 bits)

Decode

```
return PrefxOperands(
    hint=zero_extend(hint, from_nbits=5),
    base=base,
    index=index,
)
```

8.453.3. Operation

```
if not ctx.C0.Status.CU1:
    raise CoprocessorUnusable(1)
if not ctx.C0.Config1.FP:
    raise CoprocessorUnusable(1)

base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
index = sign_extend(ctx.GPR[operands.index], from_nbits=ctx.GPR.width)
advisory_prefetch(ctx, base, index, operands.hint)
```

8.453.4. Description

PREFIX adds the full-width values in base and index and supplies the result and five-bit hint to the implementation-dependent prefetch interface. It does not change architecturally visible data state.

8.453.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.454. PREPEND

8.454.1. Purpose

DSP bitstream operation PREPEND.

8.454.2. Formats

MIPS

prepend

Assembly

PREPEND {rt:gpr}, {rs:gpr}, {sa:u5}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

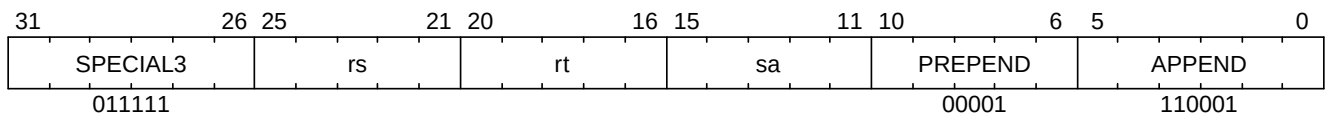


Figure 1259. Encoding (32 bits)

Decode

```
return DspAppendOperands(rt=rt, rs=rs, sa=sa)
```

microMIPS

prepend.32

Assembly

PREPEND {rt:gpr}, {rs:gpr}, {sa:u5}

Availability

DSP2

Decode priority

0

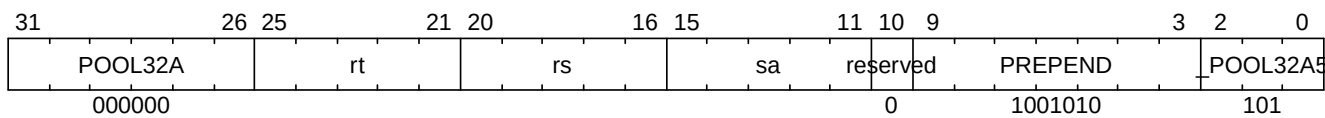


Figure 1260. Encoding (32 bits)


```
return DspAppendOperands(rt=rt, rs=rs, sa=sa)
```

8.454.3. Operation

```
validate_access_to_dsp2_resources(ctx)
amount = operands.sa & 0x1F
if amount:
    source = ctx.GPR[operands.rs] << (32 - amount)
    target = ctx.GPR[operands.rt] >> amount
    ctx.GPR[operands.rt] = sign_extend(
        (source | target) & 0xFFFF_FFFF,
        from_nbits=32,
    )
```

8.454.4. Description

PREPEND is a DSP bitstream operation.

8.454.5. Exceptions

None.

8.455. PUL.PS

8.455.1. Purpose

Pair upper lower.

8.455.2. Formats

MIPS

pul.ps

Assembly

PUL.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

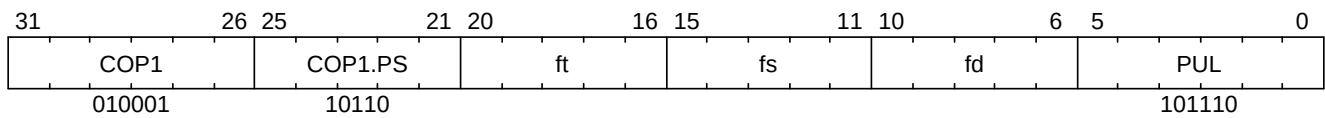


Figure 1261. Encoding (32 bits)

Decode

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

pul.ps.32

Assembly

PUL.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

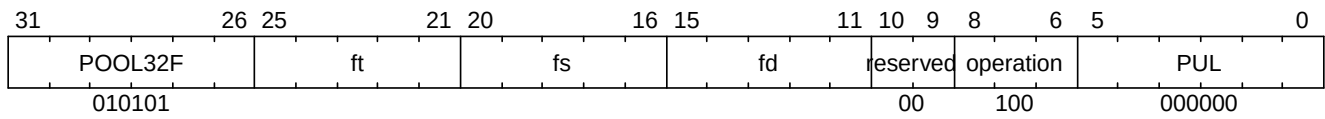


Figure 1262. Encoding (32 bits)

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.455.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
set_fpr(ctx, operands.fd, ((fs >> 32) << 32) | (ft & 0xFFFF_FFFF), "PS")
```

8.455.4. Description

Merge an upper and lower paired-single lane.

8.455.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.456. PUU.PS

8.456.1. Purpose

Pair upper upper.

8.456.2. Formats

MIPS

puu.ps

Assembly

PUU.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

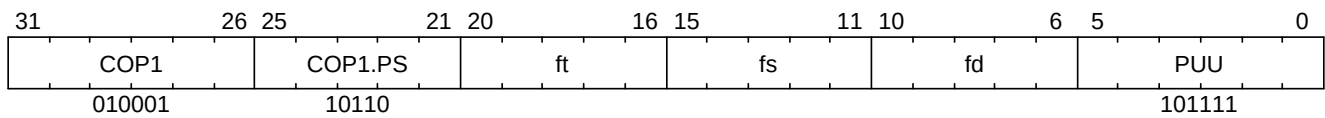


Figure 1263. Encoding (32 bits)

Decode

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

puu.ps.32

Assembly

PUU.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

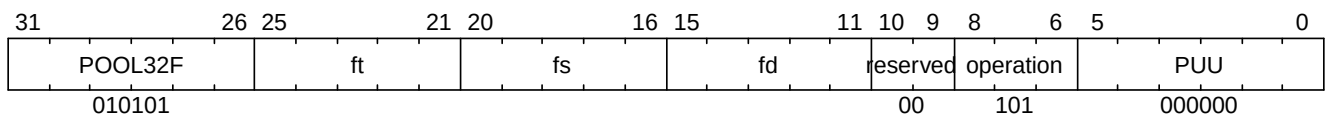


Figure 1264. Encoding (32 bits)

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.456.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
fs = get_fpr(ctx, operands.fs, "PS")
ft = get_fpr(ctx, operands.ft, "PS")
set_fpr(ctx, operands.fd, ((fs >> 32) << 32) | (ft >> 32), "PS")
```

8.456.4. Description

Merge upper paired-single lanes.

8.456.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.457. RADDU.W.QB

8.457.1. Purpose

DSP packed operation RADDU.W.QB.

8.457.2. Formats

MIPS

raddu.w.qb

Assembly

RADDU.W.QB {rd:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

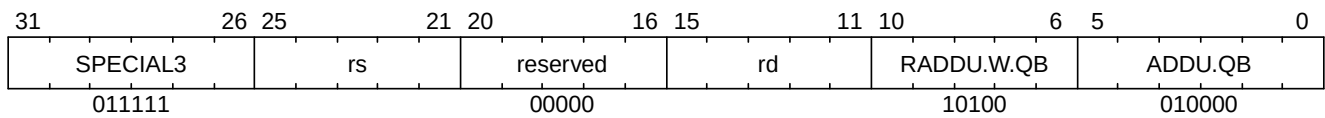


Figure 1265. Encoding (32 bits)

Decode

```
return DspUnary0perands(rd=rd, rs=rs)
```

microMIPS

raddu.w.qb.32

Assembly

RADDU.W.QB {rd:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

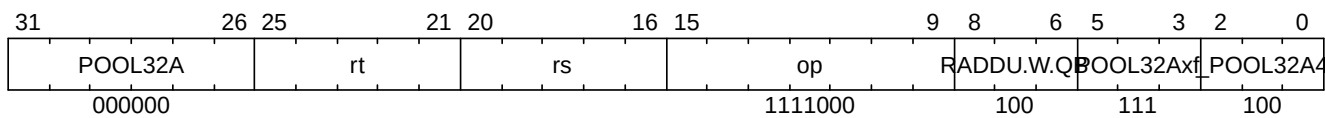


Figure 1266. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

nanoMIPS

raddu.w.qb.32

Assembly

RADDU.W.QB {rd:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

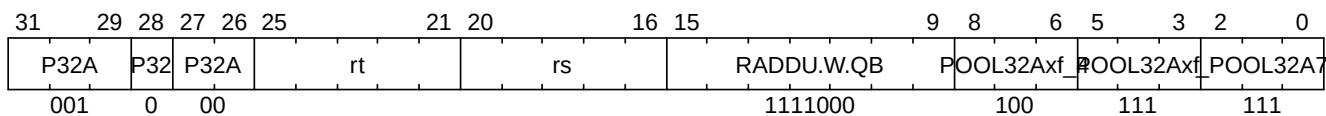


Figure 1267. Encoding (32 bits)

Decode

```
return DspUnaryOperands(rd=rt, rs=rs)
```

8.457.3. Operation

```
validate_access_to_dsp_resources(ctx)
total = sum(lanes(ctx.GPR[operands.rs], 8, 4))
ctx.GPR[operands.rd] = zero_extend(total & 0x3FF, from_nbits=10)
```

8.457.4. Description

RADDU.W.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.457.5. Exceptions

None.

8.458. RDDSP

8.458.1. Purpose

Copy selected DSPControl fields to a GPR.

8.458.2. Formats

MIPS

rddsp

Assembly

RDDSP {rd:gpr}, {mask:u10}

Availability

DSP && AR >= R2_3_5

Decode priority

0

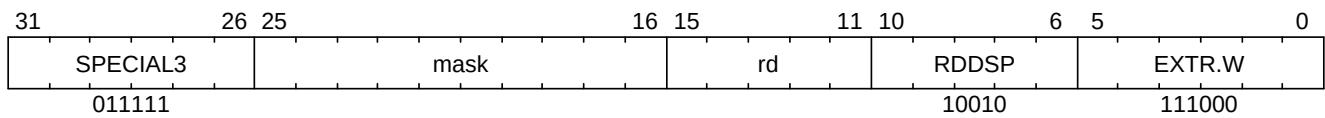


Figure 1268. Encoding (32 bits)

Decode

```
return DspControlRead0perands(rd=rd, mask=mask)
```

microMIPS

rddsp.32

Assembly

RDDSP {rd:gpr}, {mask:u7}

Availability

DSP

Decode priority

0

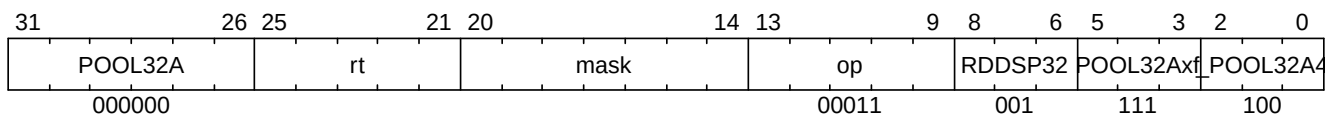


Figure 1269. Encoding (32 bits)

Decode

```
return DspControlReadOperands(rd=rt, mask=mask)
```

nanoMIPS

rddsp.32

Assembly

RDDSP {rd:gpr}, {mask:u7}

Availability

DSP

Decode priority

0

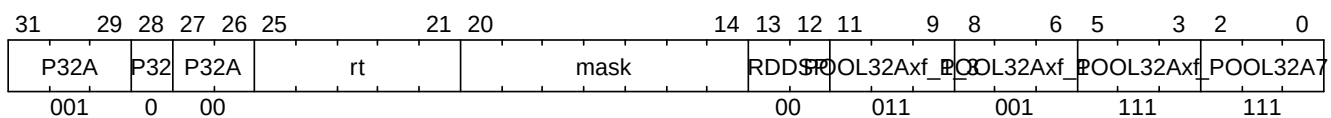


Figure 1270. Encoding (32 bits)

Decode

```
return DspControlReadOperands(rd=rt, mask=mask)
```

8.458.3. Operation

```
validate_access_to_dsp_resources(ctx)
value = 0
if operands.mask & 0x01:
    value |= ctx.DSPControl.Pos
if operands.mask & 0x02:
    value |= ctx.DSPControl.Scound << 7
if operands.mask & 0x04:
    value |= ctx.DSPControl.C << 13
if operands.mask & 0x08:
    value |= ctx.DSPControl.Ouflag << 16
if operands.mask & 0x10:
    value |= ctx.DSPControl.Ccond << 24
if operands.mask & 0x20:
    value |= ctx.DSPControl.EFI << 14
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.458.4. Description

The mask selects Pos, Scound, C, Ouflag, Ccond, and EFI. Unselected and reserved bits are returned as zero.

8.458.5. Exceptions

None.

8.459. RDHWR

8.459.1. Purpose

Read a hardware register.

8.459.2. Formats

MIPS

`rdhwr`

Assembly

`RDHWR {rt:gpr}, {hs:u5}, {sel:u3}`

Availability

AR >= R2_3_5

Decode priority

0

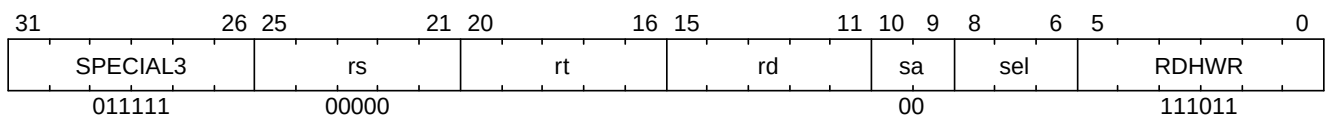


Figure 1271. Encoding (32 bits)

Decode

```
return RdhwrOperands(rt=rt, hs=rd, sel=sel)
```

microMIPS

`rdhwr.32`

Assembly

`RDHWR {rt:gpr}, {hs:u5}`

Availability

MMAR < R6

Decode priority

0

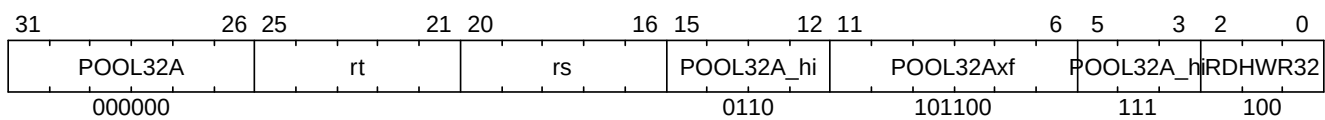


Figure 1272. Encoding (32 bits)

Decode

```
return RdhwrOperands(rt=rt, hs=rs, sel=0)
```

rdhwr.r6.32

Assembly

RDHWR {rt:gpr}, {hs:u5}, {sel:u3}

Availability

MMAR >= R6

Decode priority

0

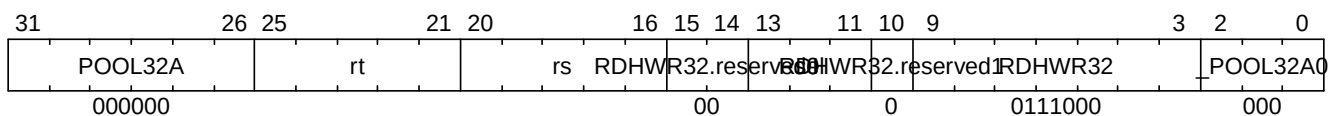


Figure 1273. Encoding (32 bits)

Decode

```
return RdhwrOperands(rt=rt, hs=rs, sel=sel)
```

nanoMIPS

rdhwr.32

Assembly

RDHWR {rt:gpr}, {hs:u5}, {sel:u5}

Availability

!(NMS)

Decode priority

0

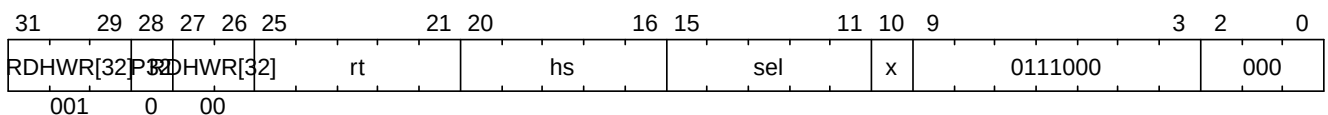


Figure 1274. Encoding (32 bits)

Decode

```
return RdhwrOperands(rt=rt, hs=hs, sel=sel)
```

8.459.3. Operation

```
if operands.sel and operands.hs != 4:
```

```

    raise ReservedInstruction("RDHWR select is unsupported for this register")
if not IsCoproprocessor0Enabled(ctx) and not (
    read_cp0_register(ctx, 7, 0) & (1 << operands.hs)
):
    raise ReservedInstruction("RDHWR is not enabled")
if operands.hs == 0:
    value = ctx.C0.EBase.CPUNum
elif operands.hs == 1:
    data_line = ctx.C0.Config1.DL
    instruction_line = ctx.C0.Config1.IL
    line = min(filter(None, (data_line, instruction_line)), default=0)
    value = 1 << (line + 1) if line else 0
elif operands.hs == 2:
    value = read_cp0_register(ctx, 9, 0)
elif operands.hs == 3:
    value = 1
elif operands.hs == 4:
    raise ReservedInstruction("performance counters are not implemented")
elif operands.hs == 5:
    if ctx.C0.Config.AR < 2:
        raise ReservedInstruction("RDHWR register 5 requires Release 6")
    value = ctx.C0.Config5.XNP
elif operands.hs == 29:
    if not ctx.C0.Config3.ULRI:
        raise ReservedInstruction("UserLocal is not implemented")
    value = read_cp0_register(ctx, 4, 2)
else:
    raise ReservedInstruction("RDHWR register is unsupported")
ctx.GPR[operands.rt] = sign_extend(value, from_nbits=32)

```

8.459.4. Description

RDHWR exposes selected CP0 hardware state subject to HWREna.

8.459.5. Exceptions

- **RI** (Reserved Instruction) — RDHWR select is unsupported for this register; RDHWR is not enabled; performance counters are not implemented; RDHWR register 5 requires Release 6; UserLocal is not implemented; RDHWR register is unsupported

8.460. RDPGPR

8.460.1. Purpose

Read a GPR in the previous shadow register set.

8.460.2. Formats

MIPS

rdpgpr

Assembly

RDPGPR {dst:gpr}, {src:gpr}

Availability

AR >= R2_3_5

Decode priority

0

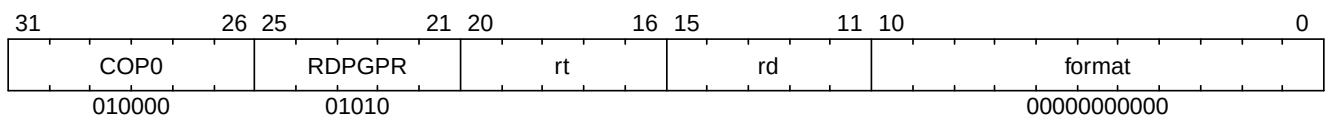


Figure 1275. Encoding (32 bits)

Decode

```
return RdpgprOperands(dst=rd, src=rt)
```

microMIPS

rdpgpr.32

Assembly

RDPGPR {dst:gpr}, {src:gpr}

Availability

true

Decode priority

0

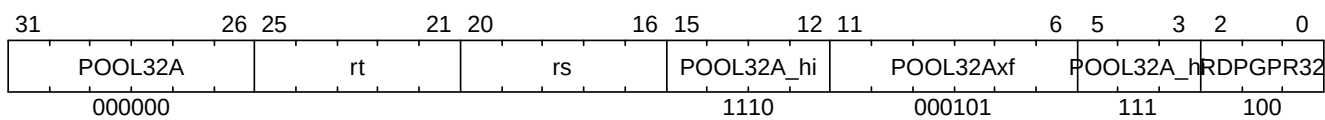


Figure 1276. Encoding (32 bits)

Decode

```
return RdpgrpOperands(dst=rt, src=rs)
```

nanoMIPS

rdpgpr.32

Assembly

RDPGPR {dst:gpr}, {src:gpr}

Availability

true

Decode priority

0

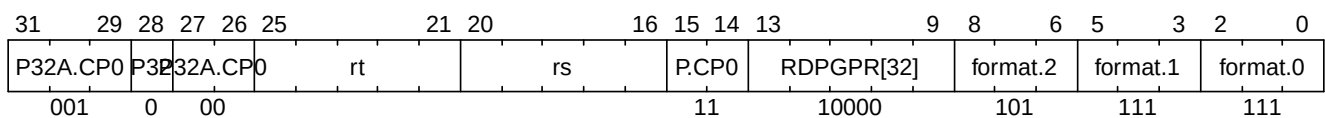


Figure 1277. Encoding (32 bits)

Decode

```
return RdpgrpOperands(dst=rt, src=rs)
```

8.460.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
ctx.GPR[operands.dst] = read_previous_gpr(ctx, operands.src)
```

8.460.4. Description

The source is read from SRSCtl.PSS and written to the destination in the current set. On a processor without shadow register sets, the operation is an ordinary GPR copy.

8.460.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.461. RECIPI.D

8.461.1. Purpose

Compute a D-format reciprocal.

8.461.2. Formats

MIPS

recip.d

Assembly

RECIPI.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

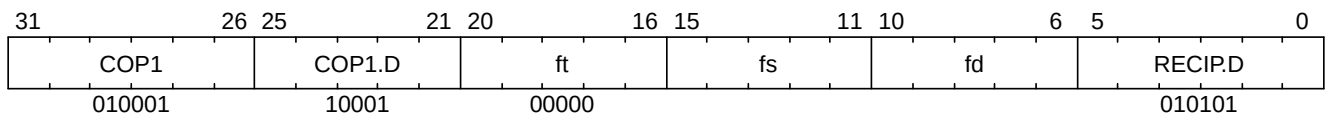


Figure 1278. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

recip.d

Assembly

RECIPI.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

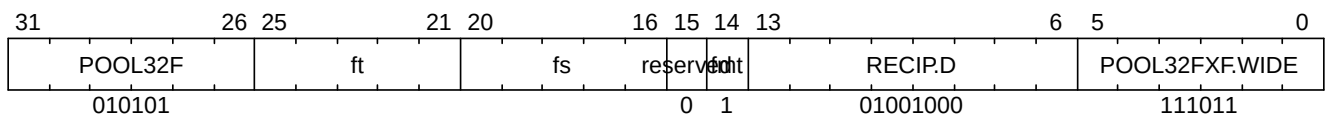


Figure 1279. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

recip.d

Assembly

RECIP.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

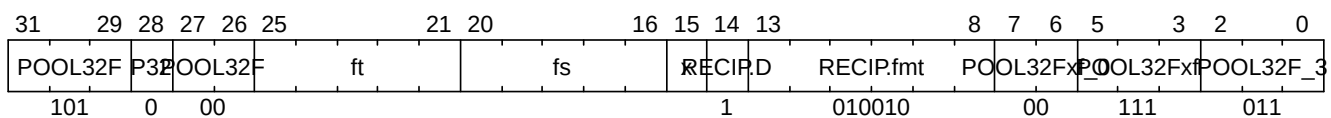


Figure 1280. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.461.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_recip(
    "D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.461.4. Description

RECIP.D validates CP1 and D-format support before updating the destination FPR.

8.461.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.462. RECIP.S

8.462.1. Purpose

Compute an S-format reciprocal.

8.462.2. Formats

MIPS

recip.s

Assembly

RECIP.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

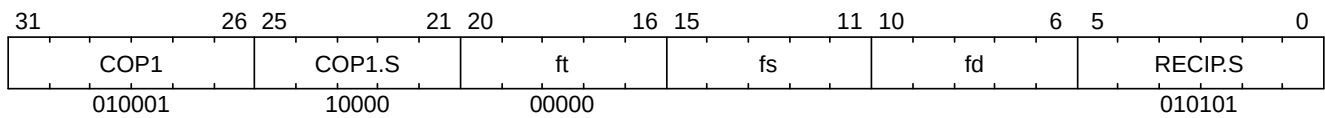


Figure 1281. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

recip.s

Assembly

RECIP.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

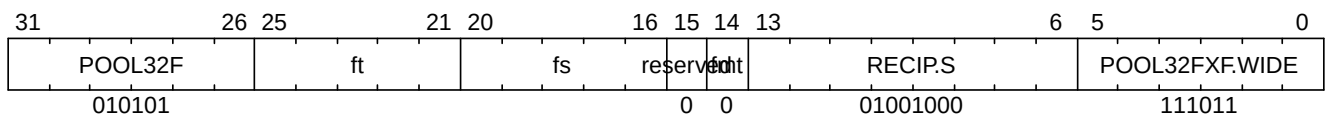


Figure 1282. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

recip.s

Assembly

RECIP.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

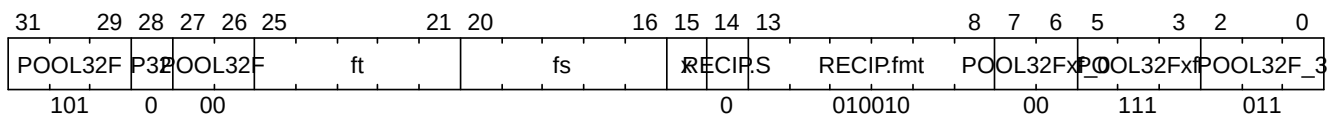


Figure 1283. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.462.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_recip(
    "S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.462.4. Description

RECIP.S validates CP1 and S-format support before updating the destination FPR.

8.462.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.463. REPL.PH

8.463.1. Purpose

Replicate a signed ten-bit immediate into two halfwords.

8.463.2. Formats

MIPS

repl.ph

Assembly

REPL.PH {rd:gpr}, {immediate:s10}

Availability

DSP && AR >= R2_3_5

Decode priority

0

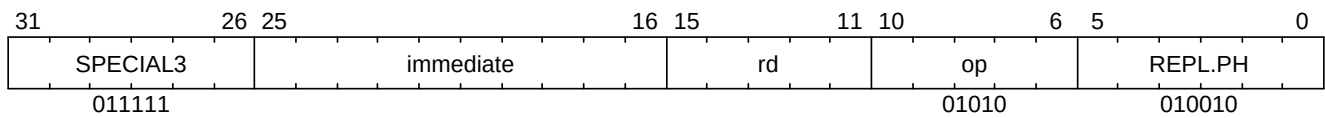


Figure 1284. Encoding (32 bits)

Decode

```
return DspReplicateOperands(  
    rd=rd,  
    immediate=sign_extend(immediate, from_nbits=10),  
)
```

microMIPS

repl.ph.32

Assembly

REPL.PH {rd:gpr}, {immediate:s10}

Availability

DSP

Decode priority

0

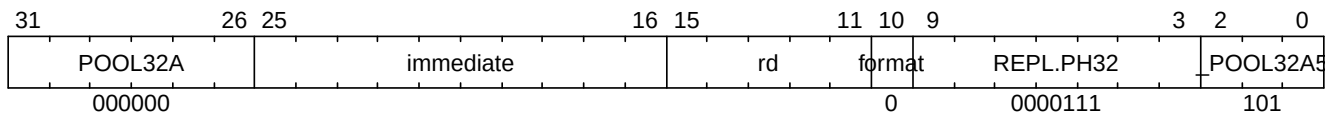


Figure 1285. Encoding (32 bits)

Decode

```
return DspReplicateOperands(  
    rd=rd,  
    immediate=sign_extend(immediate, from_nbits=10),  
)
```

nanoMIPS

repl.ph.32

Assembly

REPL.PH {rd:gpr}, {immediate:s10}

Availability

DSP

Decode priority

0

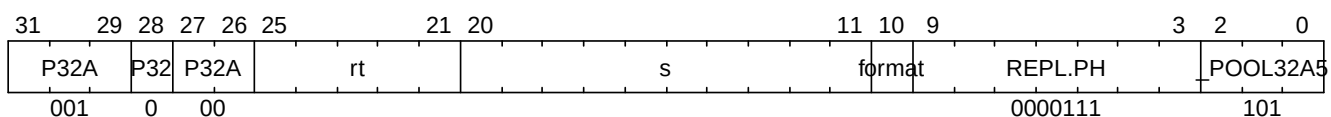


Figure 1286. Encoding (32 bits)

Decode

```
return DspReplicateOperands(  
    rd=rt,  
    immediate=sign_extend(s, from_nbits=10),  
)
```

8.463.3. Operation

```
validate_access_to_dsp_resources(ctx)  
lane = operands.immediate & 0xFFFF  
ctx.GPR[operands.rd] = sign_extend((lane << 16) | lane, from_nbits=32)
```

8.463.4. Description

The immediate is sign-extended to a halfword before replication and the packed word is sign-extended to GPR width.

8.463.5. Exceptions

None.

8.464. REPL.QB

8.464.1. Purpose

Replicate an unsigned byte immediate into all four bytes.

8.464.2. Formats

MIPS

repl.qb

Assembly

REPL.QB {rd:gpr}, {immediate:u8}

Availability

DSP && AR >= R2_3_5

Decode priority

0

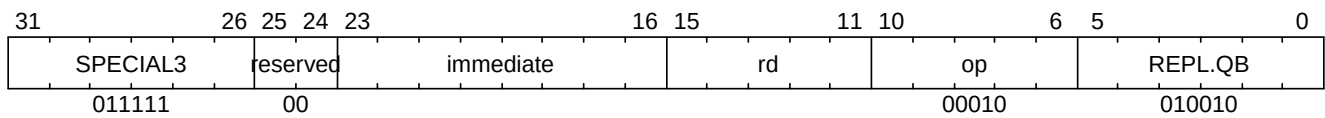


Figure 1287. Encoding (32 bits)

Decode

```
return DspReplicateOperands(rd=rd, immediate=immediate)
```

microMIPS

repl.qb.32

Assembly

REPL.QB {rd:gpr}, {immediate:u8}

Availability

DSP

Decode priority

0

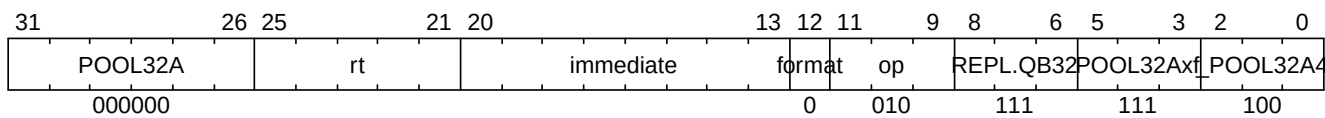


Figure 1288. Encoding (32 bits)

Decode

```
return DspReplicateOperands(rd=rt, immediate=immediate)
```

nanoMIPS

repl.qb.32

Assembly

REPL.QB {rd:gpr}, {immediate:u8}

Availability

DSP

Decode priority

0

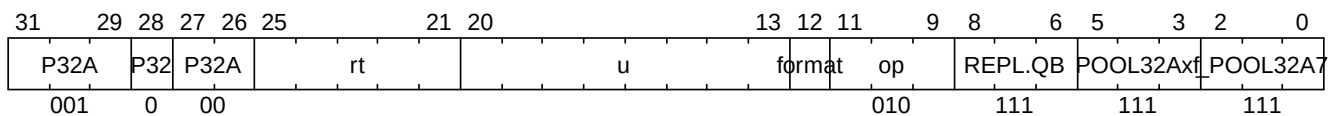


Figure 1289. Encoding (32 bits)

Decode

```
return DspReplicateOperands(rd=rt, immediate=u)
```

8.464.3. Operation

```
validate_access_to_dsp_resources(ctx)
lane = operands.immediate & 0xFF
ctx.GPR[operands.rd] = sign_extend(lane * 0x0101_0101, from_nbits=32)
```

8.464.4. Description

The replicated word is sign-extended to the implemented GPR width.

8.464.5. Exceptions

None.

8.465. REPLV.PH

8.465.1. Purpose

Replicate the low halfword into both packed halfwords.

8.465.2. Formats

MIPS

replv.ph

Assembly

REPLV.PH {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

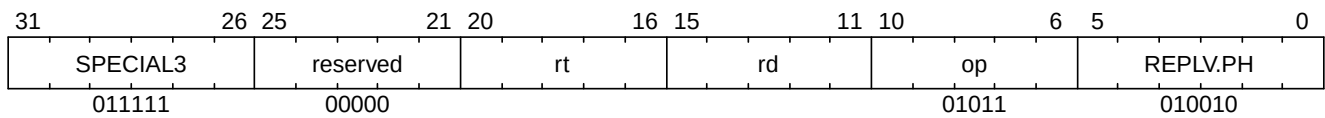


Figure 1290. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

replv.ph.32

Assembly

REPLV.PH {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

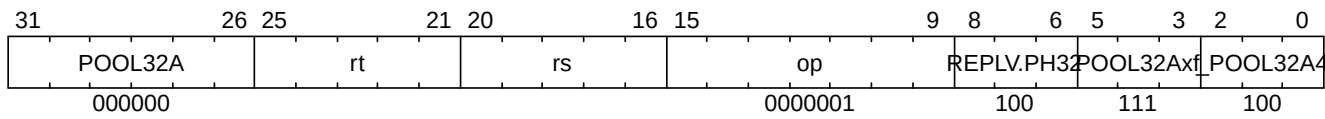


Figure 1291. Encoding (32 bits)

Decode

```
return DspPackUnary0Operands(rd=rt, rt=rs)
```

nanoMIPS

replv.ph.32

Assembly

REPLV.PH {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

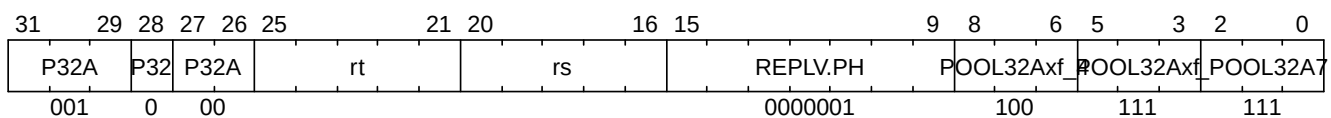


Figure 1292. Encoding (32 bits)

Decode

```
return DspPackUnary0Operands(rd=rt, rt=rs)
```

8.465.3. Operation

```
validate_access_to_dsp_resources(ctx)
lane = ctx.GPR[operands.rt] & 0xFFFF
ctx.GPR[operands.rd] = sign_extend((lane << 16) | lane, from_nbits=32)
```

8.465.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.465.5. Exceptions

None.

8.466. REPLV.QB

8.466.1. Purpose

Replicate the low byte into all four packed bytes.

8.466.2. Formats

MIPS

replv.qb

Assembly

REPLV.QB {rd:gpr}, {rt:gpr}

Availability

DSP && AR >= R2_3_5

Decode priority

0

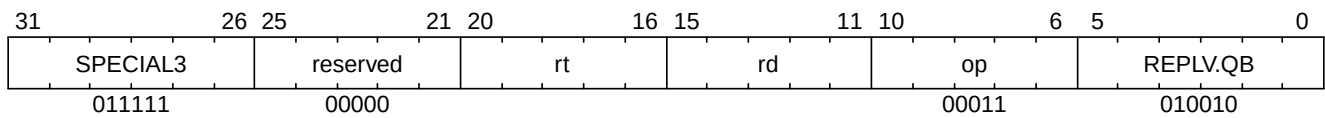


Figure 1293. Encoding (32 bits)

Decode

```
return DspPackUnary0perands(rd=rd, rt=rt)
```

microMIPS

replv.qb.32

Assembly

REPLV.QB {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

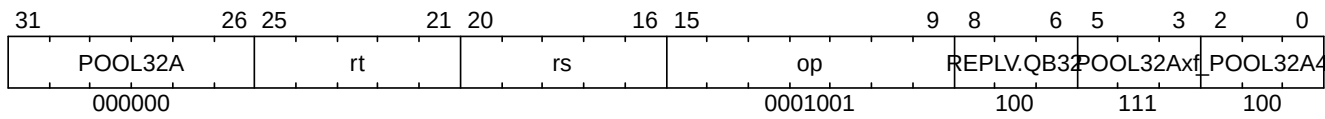


Figure 1294. Encoding (32 bits)

Decode

```
return DspPackUnary0Operands(rd=rt, rt=rs)
```

nanoMIPS

replv.qb.32

Assembly

REPLV.QB {rd:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

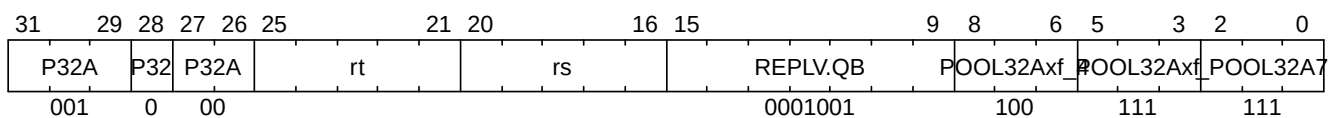


Figure 1295. Encoding (32 bits)

Decode

```
return DspPackUnary0Operands(rd=rt, rt=rs)
```

8.466.3. Operation

```
validate_access_to_dsp_resources(ctx)
lane = ctx.GPR[operands.rt] & 0xFF
ctx.GPR[operands.rd] = sign_extend(lane * 0x0101_0101, from_nbits=32)
```

8.466.4. Description

The operation consumes the low 32 bits and sign-extends its word result to the implemented GPR width.

8.466.5. Exceptions

None.

8.467. RESTORE

8.467.1. Purpose

Restore a consecutive register list and increment the stack pointer.

8.467.2. Formats

nanoMIPS

restore.32

Assembly

RESTORE {rt:gpr}, ...]

Availability

true

Decode priority

0

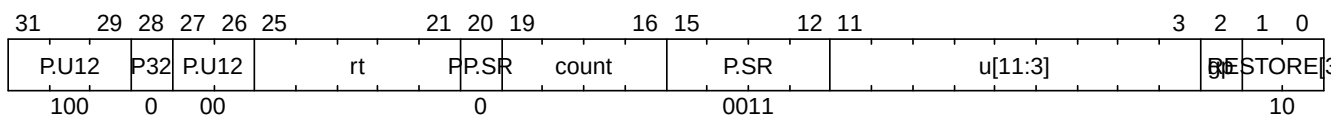


Figure 1296. Encoding (32 bits)

Decode

```
del ctx
return SaveRestoreOperands(
    rt=rt,
    u=zero_extend(u, from_nbits=12),
    count=count,
    gp=bool(gp),
)
```

8.467.3. Operation

```
base = sign_extend(ctx.GPR[29], from_nbits=ctx.GPR.width)
for index in range(operands.count):
    register = save_restore_register(
        operands.rt, index, operands.gp, operands.count
    )
    offset = int(operands.u) - ((index + 1) << 2)
    va = effective_address(ctx, base, offset, "Load")
    if va & 3:
        raise AddressErrorLoad(va, "unaligned RESTORE address")
    data = read_memory_at_va(ctx, va, nbytes=4)
    ctx.GPR[register] = sign_extend(data, from_nbits=32)
```

```
if register == 29:
    raise ValueError("RESTORE including $29 is UNPREDICTABLE")
ctx.GPR[29] = effective_address(ctx, base, int(operands.u))
```

8.467.4. Description

RESTORE loads 32-bit register words from the stack frame.

8.467.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — The gp form is unavailable on NMS.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.468. RESTORE.JRC

8.468.1. Purpose

Restore registers, increment the stack pointer, and return compactly.

8.468.2. Formats

nanoMIPS

restore.jrc.16

Assembly

RESTORE.JRC {rt:gpr}, ...]

Availability

true

Decode priority

0

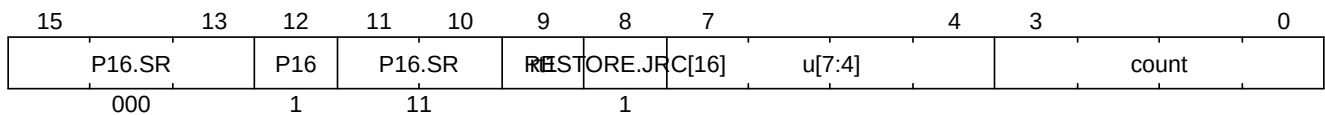


Figure 1297. Encoding (16 bits)

Decode

```
del ctx
exec.isCTI = True
rt = 30 if rt1 == 0 else 31
return SaveRestoreOperands(
    rt=rt,
    u=zero_extend(u, from_nbits=12),
    count=count,
    gp=False,
)
```

restore.jrc.32

Assembly

RESTORE.JRC {rt:gpr}, ...]

Availability

true

Decode priority

0

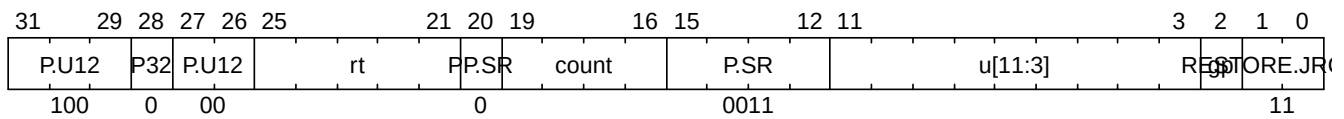


Figure 1298. Encoding (32 bits)

Decode

```

del ctx
exec.isCTI = True
return SaveRestoreOperands(
    rt=rt,
    u=zero_extend(u, from_nbits=12),
    count=count,
    gp=bool(gp),
)

```

8.468.3. Operation

```

base = sign_extend(ctx.GPR[29], from_nbits=ctx.GPR.width)
for index in range(operands.count):
    register = save_restore_register(
        operands.rt, index, operands.gp, operands.count
    )
    offset = int(operands.u) - ((index + 1) << 2)
    va = effective_address(ctx, base, offset, "Load")
    if va & 3:
        raise AddressErrorLoad(va, "unaligned RESTORE.JRC address")
    data = read_memory_at_va(ctx, va, nbytes=4)
    ctx.GPR[register] = sign_extend(data, from_nbits=32)
    if register == 29:
        raise ValueError("RESTORE.JRC including $29 is UNPREDICTABLE")
ctx.GPR[29] = effective_address(ctx, base, int(operands.u))
compact_branch(ctx, ctx.GPR[31])

```

8.468.4. Description

RESTORE.JRC performs RESTORE followed by a compact jump through \$31.

8.468.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — The gp form is unavailable on NMS.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.469. RINT.D

8.469.1. Purpose

Round a D-format value to an integral floating value.

8.469.2. Formats

MIPS

`rint.d`

Assembly

`RINT.D {fd:fpr}, {fs:fpr}`

Availability

AR >= R6

Decode priority

0

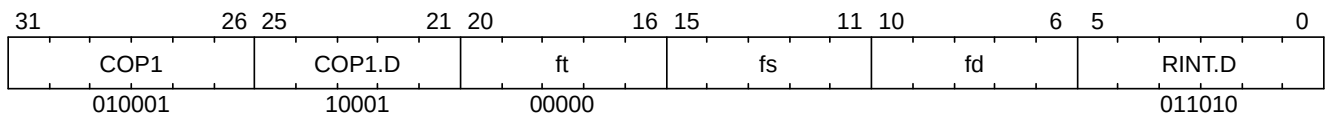


Figure 1299. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

`rint.d`

Assembly

`RINT.D {fd:fpr}, {fs:fpr}`

Availability

MMAR >= R6

Decode priority

0

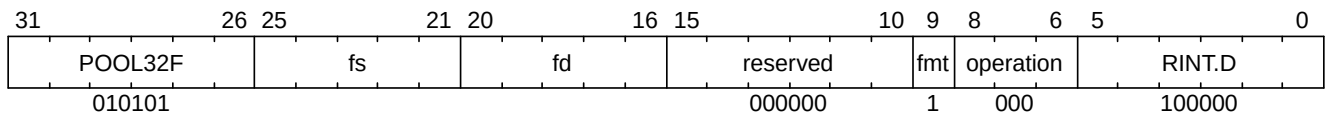


Figure 1300. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

nanoMIPS

rint.d

Assembly

RINT.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

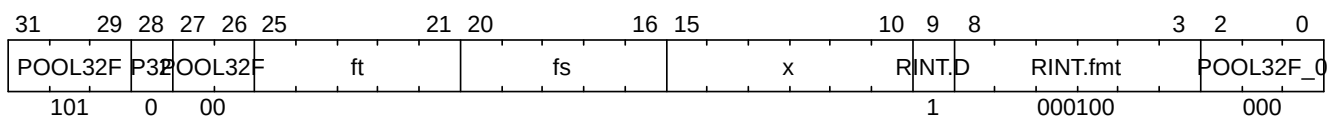


Figure 1301. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.469.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_rint(
    "D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.469.4. Description

RINT.D validates CP1 and D-format support before updating the destination FPR.

8.469.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.470. RINT.S

8.470.1. Purpose

Round an S-format value to an integral floating value.

8.470.2. Formats

MIPS

`rint.s`

Assembly

`RINT.S {fd:fpr}, {fs:fpr}`

Availability

AR >= R6

Decode priority

0

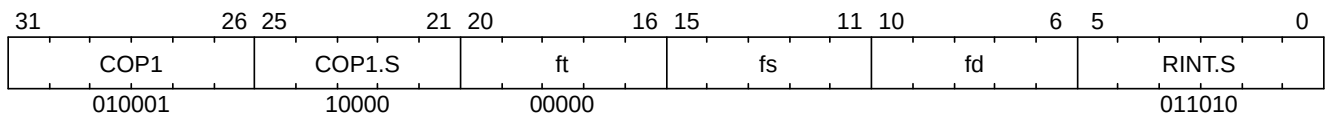


Figure 1302. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

`rint.s`

Assembly

`RINT.S {fd:fpr}, {fs:fpr}`

Availability

MMAR >= R6

Decode priority

0

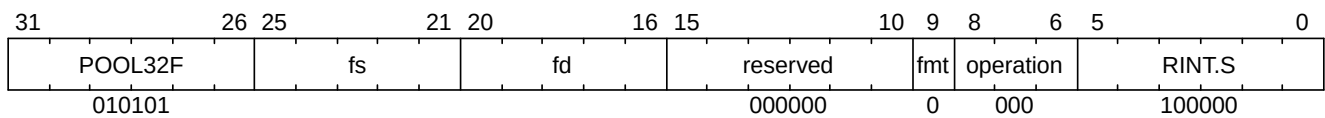


Figure 1303. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

nanoMIPS

rint.s

Assembly

RINT.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

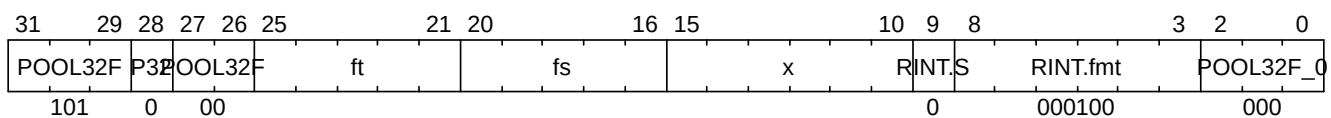


Figure 1304. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.470.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_rint(
    "S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.470.4. Description

RINT.S validates CP1 and S-format support before updating the destination FPR.

8.470.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.471. ROTR

8.471.1. Purpose

Rotate the low 32-bit word in a source GPR right by a constant amount and place the result in a destination GPR.

8.471.2. Formats

MIPS

`rotr`

Assembly

`ROTR {rt:gpr}, {rs:gpr}, {shift:u5}`

Availability

AR >= R2_3_5

Decode priority

0

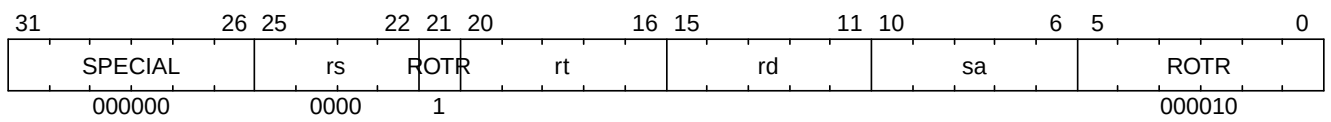


Figure 1305. Encoding (32 bits)

Decode

```
return RotrOperands(rt=rd, rs=rt, shift=sa)
```

microMIPS

`rotr.32`

Assembly

`ROTR {rt:gpr}, {rs:gpr}, {shift:u5}`

Availability

true

Decode priority

0

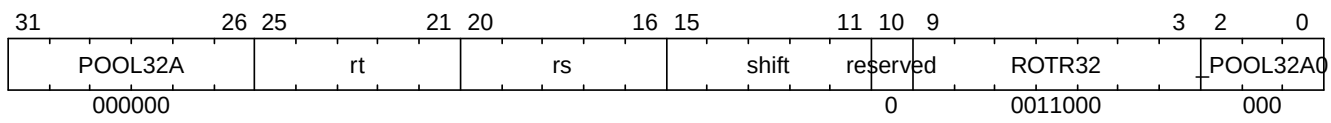


Figure 1306. Encoding (32 bits)

Decode

```
return RotrOperands(
    rt=rt,
    rs=rs,
    shift=zero_extend(shift, from_nbits=5),
)
```

nanoMIPS

rotr.32

Assembly

ROTR {rt:gpr}, {rs:gpr}, {shift:u5}

Availability

true

Decode priority

0

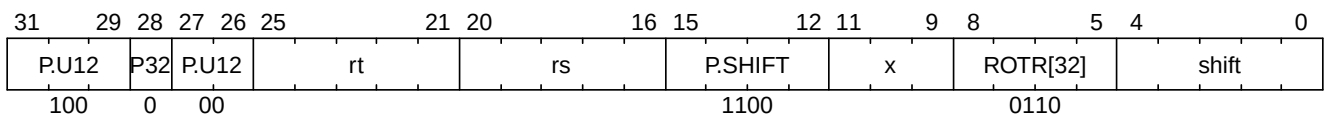


Figure 1307. Encoding (32 bits)

Decode

```
return RotrOperands(
    rt=rt,
    rs=rs,
    shift=zero_extend(shift, from_nbits=5),
)
```

8.471.3. Operation

```
source = int(zero_extend(ctx.GPR[operands.rs], from_nbits=32))
shift = int(operands.shift) & 0x1F
result = ((source >> shift) | (source << ((32 - shift) & 0x1F))) & 0xFFFF_FFFF
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.471.4. Description

ROTR[32] rotates **rs**[31:0] right by the five-bit **shift** amount. The resulting 32-bit word is sign-extended to the implemented GPR width.

8.471.5. Exceptions

None.

8.472. ROTRV

8.472.1. Purpose

Rotate the low 32-bit word in `rs` right by the low five bits of `rt` and place the result in `rd`.

8.472.2. Formats

MIPS

`rotrv`

Assembly

`ROTRV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

AR >= R2_3_5

Decode priority

0

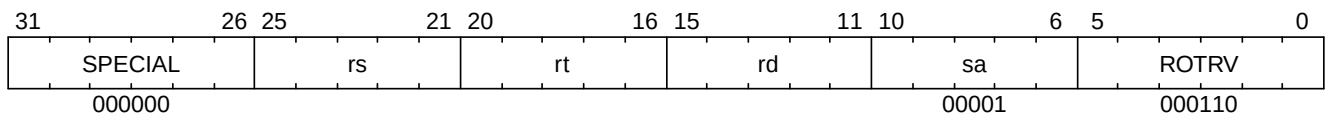


Figure 1308. Encoding (32 bits)

Decode

```
return Rotrv0perands(rd=rd, rs=rt, rt=rs)
```

microMIPS

`rotrv.32`

Assembly

`ROTRV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

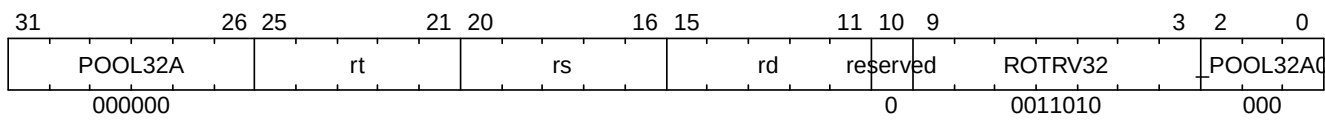


Figure 1309. Encoding (32 bits)

Decode

```
return RotrvOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

rotrv.32

Assembly

ROTRV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

!(NMS)

Decode priority

0

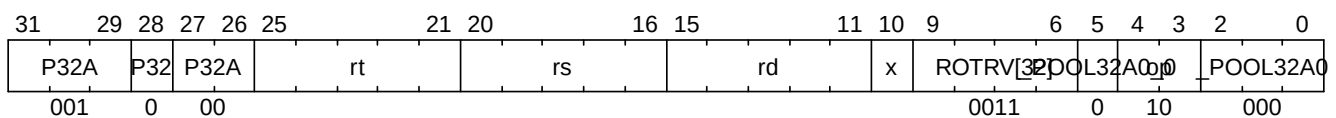


Figure 1310. Encoding (32 bits)

Decode

```
return RotrvOperands(rd=rd, rs=rs, rt=rt)
```

8.472.3. Operation

```
source = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
shift_source = zero_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
shift = int(shift_source) & 0x1F
tmp = (int(source) << 32) | int(source)
result = tmp >> shift
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.472.4. Description

ROTRV[32] uses *rs* as the 32-bit source word and the low five bits of the full-width GPR *rt* as the rotate amount. The resulting word is sign-extended to the implemented GPR width. The nanoMIPS form uses the explicit *rd*, *rs*, *rt* operand order.

8.472.5. Exceptions

- **RI** (Reserved Instruction) — ROTRV[32] is reserved when CP0 Config5.NMS is set.

8.473. ROTX

8.473.1. Purpose

Rotate and exchange bits in a 32-bit GPR word.

8.473.2. Formats

nanoMIPS

rotx.32

Assembly

ROTX {rt:gpr}, {rs:gpr}, {shift:u5}, {shiftx:u5}, {stripe:u1}

Availability

!(NMS)

Decode priority

0

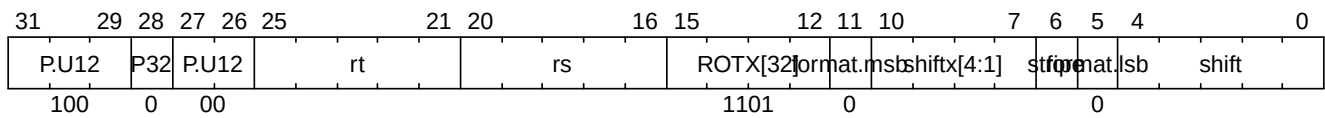


Figure 1311. Encoding (32 bits)

Decode

```
del ctx, exec
return RotxOperands(rt=rt, rs=rs, shift=shift, shiftx=shiftx, stripe=stripe)
```

8.473.3. Operation

```
source = int(zero_extend(ctx.GPR[operands.rs], from_nbits=32))
tmp0 = (source << 32) | source
shift = int(operands.shift)
shiftx = int(operands.shiftx)
stripe = int(operands.stripe)

tmp1 = 0
for i in range(47):
    selected = shift if (i & 0b01000) else shiftx
    if stripe and not (i & 0b00100):
        selected = ~selected
    bit = (tmp0 >> (i + 16 if (selected & 0x10) else i)) & 1
    tmp1 |= bit << i

tmp2 = 0
for i in range(39):
```

```

selected = shift if (i & 0b00100) else shiftx
bit = (tmp1 >> (i + 8 if (selected & 0x8) else i)) & 1
tmp2 |= bit << i

tmp3 = 0
for i in range(35):
    selected = shift if (i & 0b00010) else shiftx
    bit = (tmp2 >> (i + 4 if (selected & 0x4) else i)) & 1
    tmp3 |= bit << i

tmp4 = 0
for i in range(33):
    selected = shift if (i & 0b00001) else shiftx
    bit = (tmp3 >> (i + 2 if (selected & 0x2) else i)) & 1
    tmp4 |= bit << i

result = 0
for i in range(32):
    bit = (tmp4 >> (i + 1 if (shift & 1) else i)) & 1
    result |= bit << i
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)

```

8.473.4. Description

ROTX applies the five-stage rotate/exchange mux specified for the nanoMIPS word operation. It can implement bit and byte reversal aliases and is unavailable in NMS.

8.473.5. Exceptions

- **RI** (Reserved Instruction) — ROTX[32] is reserved when CP0 Config5.NMS is set.

8.474. ROUND.L.D

8.474.1. Purpose

Convert D format to L format using round rounding.

8.474.2. Formats

MIPS

round.l.d

Assembly

ROUND.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

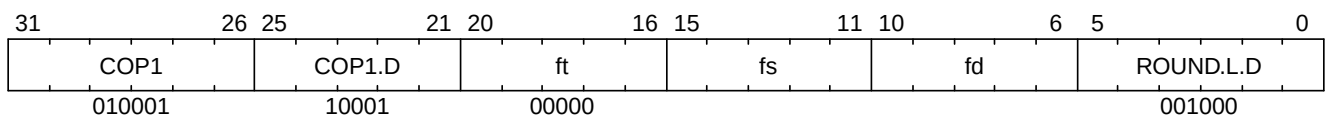


Figure 1312. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

round.l.d

Assembly

ROUND.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

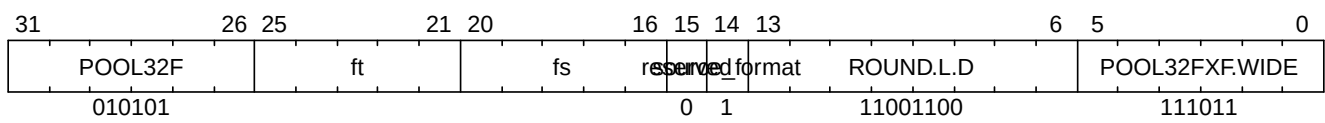


Figure 1313. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

round.l.d

Assembly

ROUND.L.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

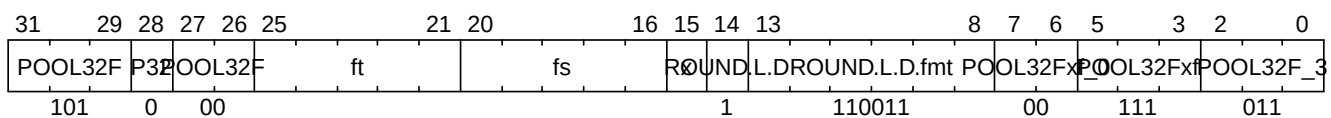


Figure 1314. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.474.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "D")
result, flags = ieee_float_to_int(
    "D", "L", fpu_rounding_mode(ctx, 0), get_fpr(ctx, operands.fs, "D"), rm=0
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.474.4. Description

ROUND.L.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.474.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.475. ROUND.L.S

8.475.1. Purpose

Convert S format to L format using round rounding.

8.475.2. Formats

MIPS

round.l.s

Assembly

ROUND.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

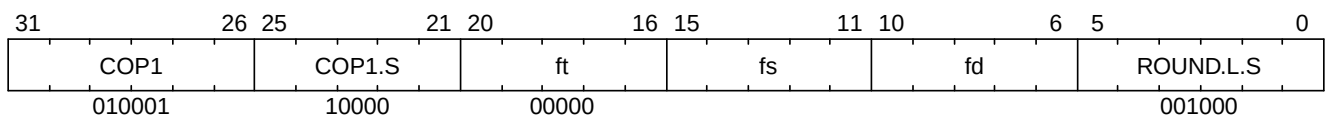


Figure 1315. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

round.l.s

Assembly

ROUND.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

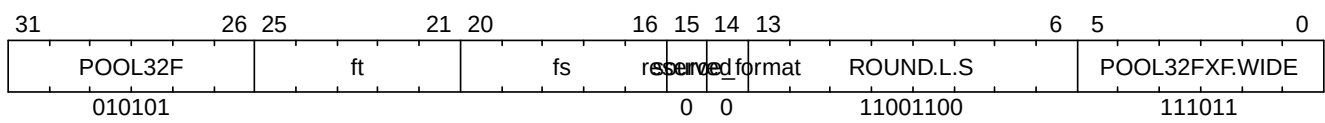


Figure 1316. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

round.l.s

Assembly

ROUND.L.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

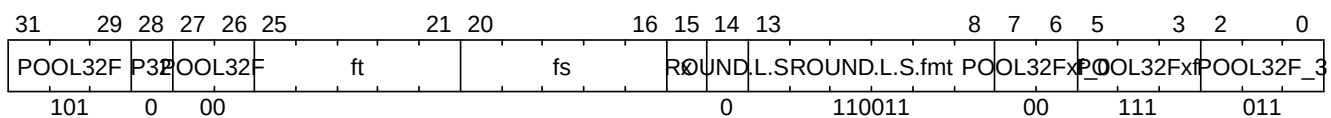


Figure 1317. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.475.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "S")
result, flags = ieee_float_to_int(
    "S", "L", fpu_rounding_mode(ctx, 0), get_fpr(ctx, operands.fs, "S"), rm=0
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.475.4. Description

ROUND.L.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.475.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.476. ROUND.W.D

8.476.1. Purpose

Convert D format to W format using round rounding.

8.476.2. Formats

MIPS

round.w.d

Assembly

ROUND.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

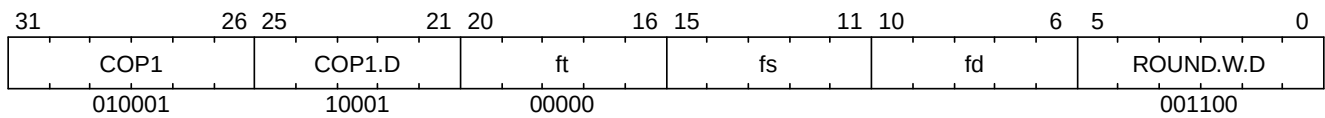


Figure 1318. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

round.w.d

Assembly

ROUND.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

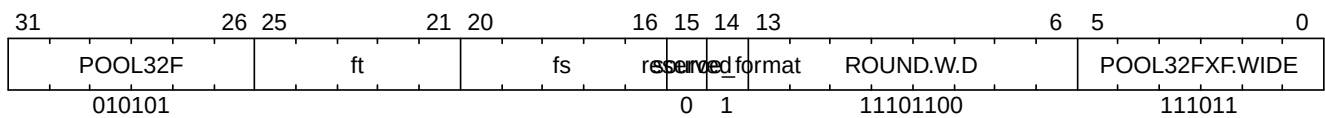


Figure 1319. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

round.w.d

Assembly

ROUND.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

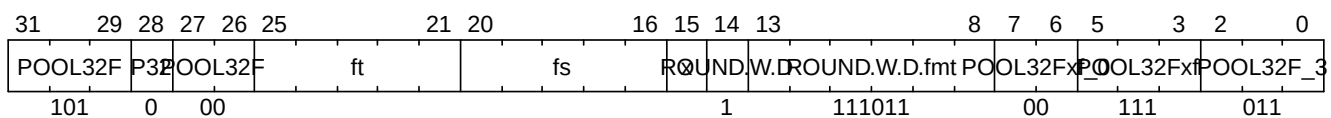


Figure 1320. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.476.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "D")
result, flags = ieee_float_to_int(
    "D", "W", fpu_rounding_mode(ctx, 0), get_fpr(ctx, operands.fs, "D"), rm=0
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.476.4. Description

ROUND.W.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.476.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.477. ROUND.W.S

8.477.1. Purpose

Convert S format to W format using round rounding.

8.477.2. Formats

MIPS

round.w.s

Assembly

ROUND.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

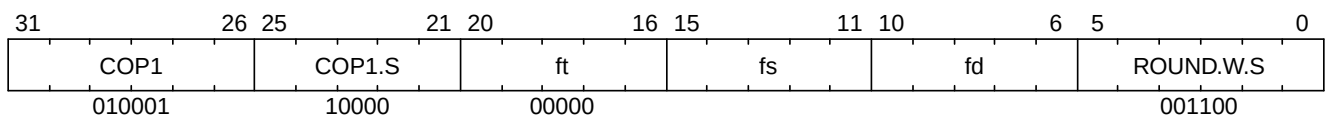


Figure 1321. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

round.w.s

Assembly

ROUND.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

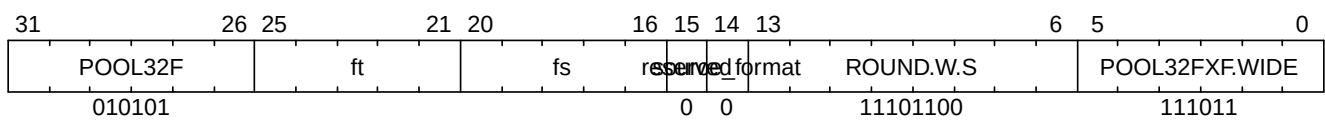


Figure 1322. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

round.w.s

Assembly

ROUND.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

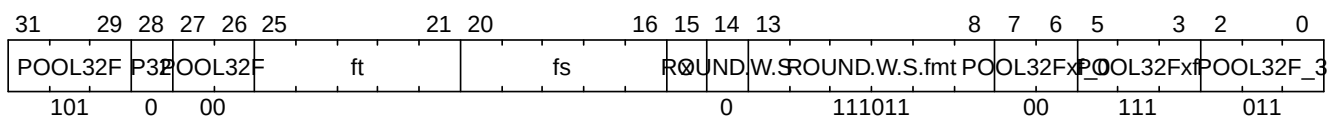


Figure 1323. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.477.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "S")
result, flags = ieee_float_to_int(
    "S", "W", fpu_rounding_mode(ctx, 0), get_fpr(ctx, operands.fs, "S"), rm=0
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.477.4. Description

ROUND.W.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.477.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.478. RSQRT.D

8.478.1. Purpose

Compute a D-format reciprocal square root.

8.478.2. Formats

MIPS

rsqrt.d

Assembly

RSQRT.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

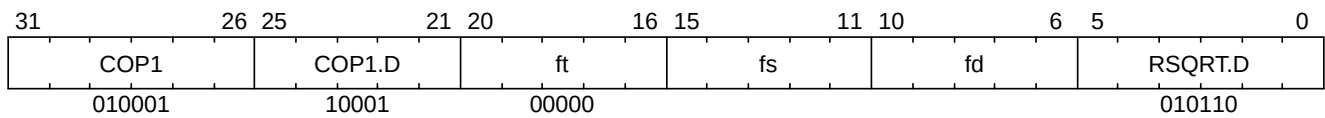


Figure 1324. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

rsqrt.d

Assembly

RSQRT.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

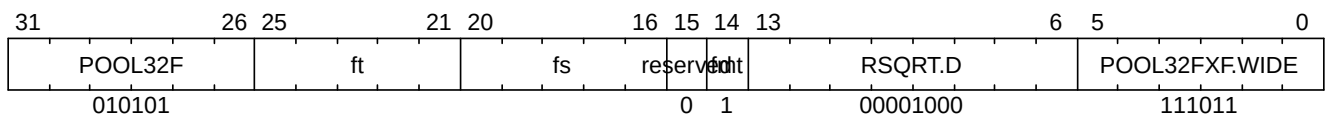


Figure 1325. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

rsqrt.d

Assembly

RSQRT.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

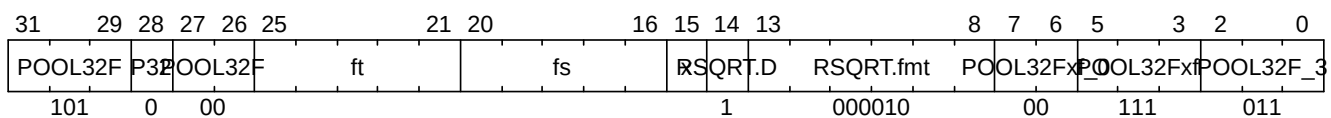


Figure 1326. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.478.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_rsqrt(
    "D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.478.4. Description

RSQRT.D validates CP1 and D-format support before updating the destination FPR.

8.478.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.479. RSQRT.S

8.479.1. Purpose

Compute an S-format reciprocal square root.

8.479.2. Formats

MIPS

rsqrt.s

Assembly

RSQRT.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

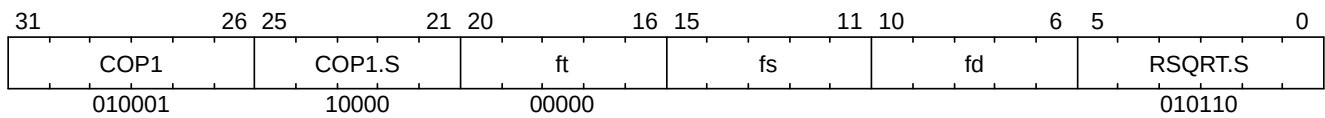


Figure 1327. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

rsqrt.s

Assembly

RSQRT.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

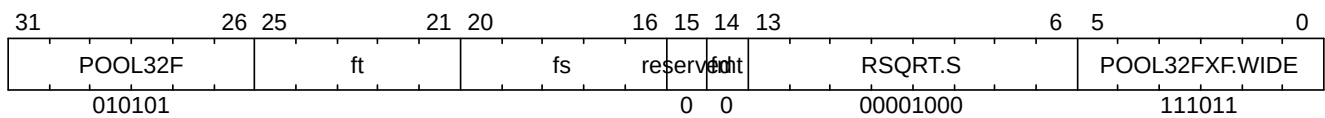


Figure 1328. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

rsqrt.s

Assembly

RSQRT.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

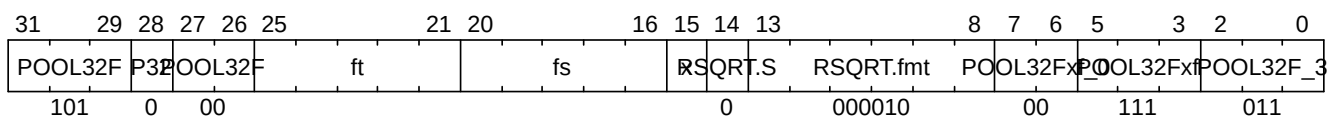


Figure 1329. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.479.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_rsqrt(
    "S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.479.4. Description

RSQRT.S validates CP1 and S-format support before updating the destination FPR.

8.479.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.480. SAVE

8.480.1. Purpose

Save a consecutive register list and decrement the stack pointer.

8.480.2. Formats

nanoMIPS

save.16

Assembly

SAVE {rt:gpr}, ...]

Availability

true

Decode priority

0

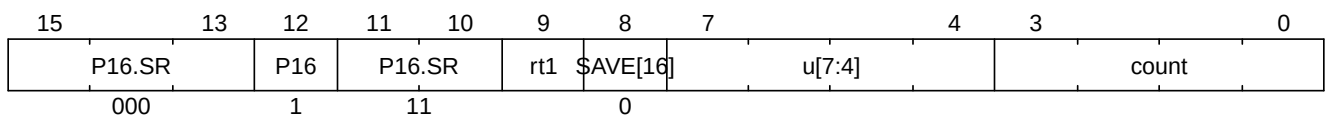


Figure 1330. Encoding (16 bits)

Decode

```
del ctx
rt = 30 if rt1 == 0 else 31
return SaveRestoreOperands(
    rt=rt,
    u=zero_extend(u, from_nbits=12),
    count=count,
    gp=False,
)
```

save.32

Assembly

SAVE {rt:gpr}, ...]

Availability

true

Decode priority

0

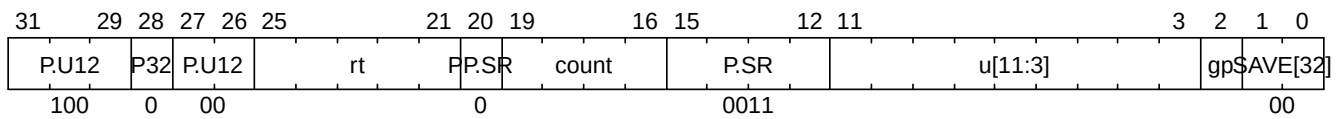


Figure 1331. Encoding (32 bits)

Decode

```
del ctx
return SaveRestoreOperands(
    rt=rt,
    u=zero_extend(u, from_nbits=12),
    count=count,
    gp=bool(gp),
)
```

8.480.3. Operation

```
base = sign_extend(ctx.GPR[29], from_nbits=ctx.GPR.width)
for index in range(operands.count):
    register = save_restore_register(
        operands.rt, index, operands.gp, operands.count
    )
    offset = -((index + 1) << 2)
    va = effective_address(ctx, base, offset, "Store")
    if va & 3:
        raise AddressErrorStore(va, "unaligned SAVE address")
    data = zero_extend(ctx.GPR[register], from_nbits=32)
    write_memory_at_va(ctx, data, va, nbytes=4)
ctx.GPR[29] = effective_address(ctx, base, -int(operands.u))
```

8.480.4. Description

SAVE stores 32-bit register words below the stack pointer.

8.480.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — The gp form is unavailable on NMS.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.481. SB

8.481.1. Purpose

Store the low byte of a GPR to memory.

8.481.2. Formats

MIPS

sb

Assembly

SB {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

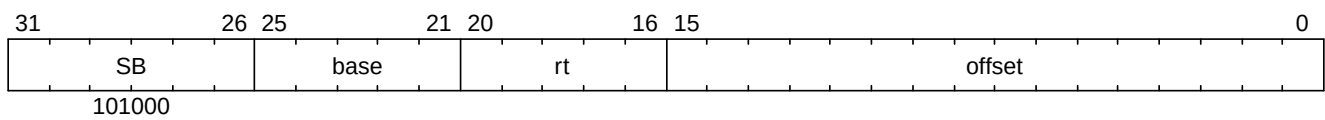


Figure 1332. Encoding (32 bits)

Decode

```
return Sb0perands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

microMIPS

sb.32

Assembly

SB {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

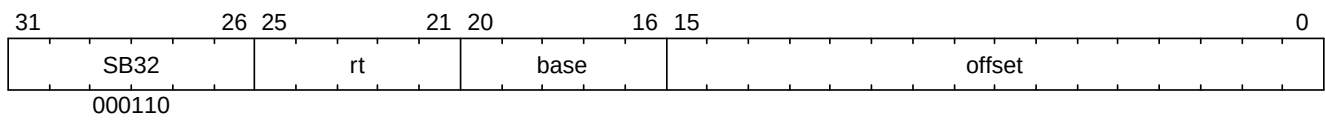


Figure 1333. Encoding (32 bits)

Decode

```
return SbOperands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

sb.16

Assembly

SB {rt:gpr3}, {offset:u4}({rs:gpr3})

Availability

true

Decode priority

0

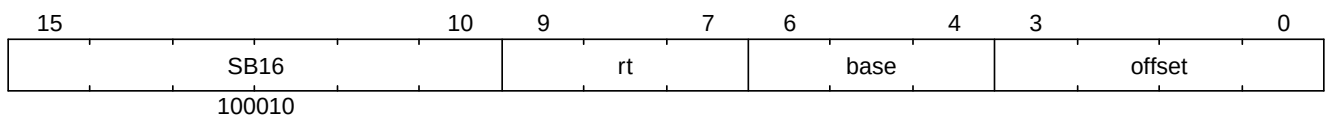


Figure 1334. Encoding (16 bits)

Decode

```
return SbOperands(  
    rt=MICROMIPS_GPR3_STORE.decode(rt),  
    rs=MICROMIPS_GPR3.decode(base),  
    offset=zero_extend(offset, from_nbits=4),  
)
```

nanoMIPS

sb.16

Assembly

SB {rt:gpr3}, {offset:u2}({rs:gpr3})

Availability

true

Decode priority

0

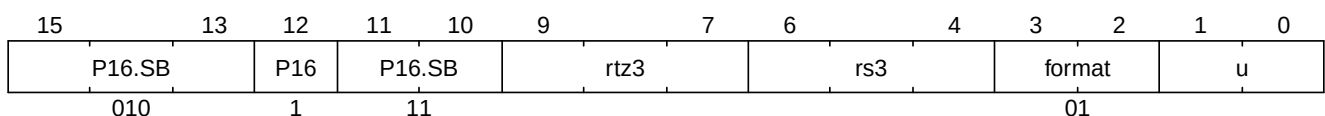


Figure 1335. Encoding (16 bits)

Decode

```
rt = GPR3_STORE.decode(rtz3)  
rs = GPR3.decode(rs3)
```

```
offset = zero_extend(u, from_nbits=2)
return SbOperands(rt=rt, rs=rs, offset=offset)
```

sb.gp

Assembly

SB {rt:gpr}, {offset:u18}(gp)

Availability

true

Decode priority

0

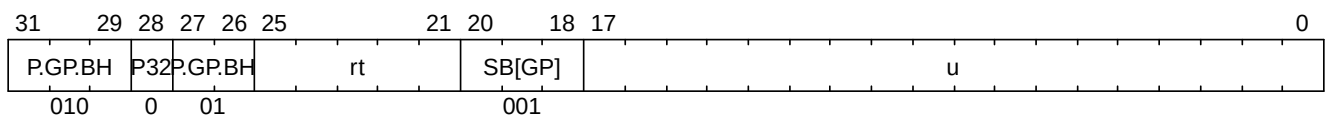


Figure 1336. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=18)
return SbOperands(rt=rt, rs=28, offset=offset)
```

sb.s9

Assembly

SB {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

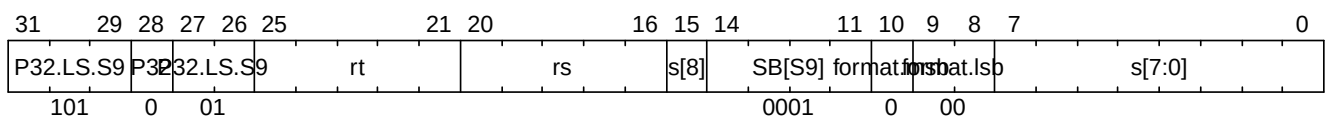


Figure 1337. Encoding (32 bits)

Decode

```
offset = sign_extend(s, from_nbits=9)
return SbOperands(rt=rt, rs=rs, offset=offset)
```

sb.u12

Assembly

SB {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

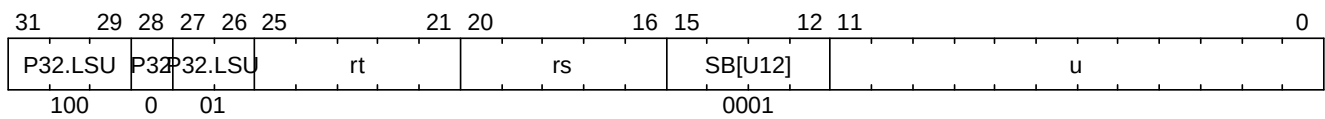


Figure 1338. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return SbOperands(rt=rt, rs=rs, offset=offset)
```

8.481.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
data = zero_extend(ctx.GPR[operands.rt], from_nbits=8)
write_memory_at_va(ctx, data, va, nbytes=1)
```

8.481.4. Description

SB adds the encoded byte offset to the base GPR, then stores the low eight bits of `rt` through the active virtual-memory backend. nanoMIPS provides compact, GP-relative, signed-nine-bit, and unsigned-twelve-bit offset forms.

8.481.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.482. SBE

8.482.1. Purpose

Store a byte using EVA addressing.

8.482.2. Formats

nanoMIPS

sbe.32

Assembly

SBE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

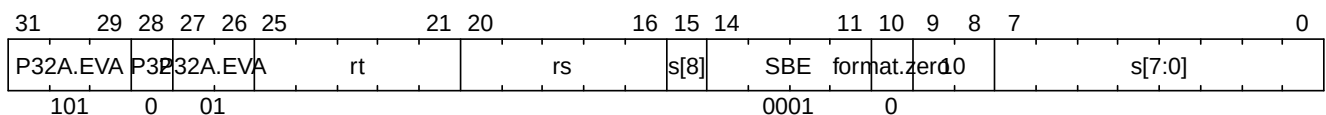


Figure 1339. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.482.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store", eva=True)
data = zero_extend(ctx.GPR[operands.rt], from_nbits=8)
write_memory_at_va(ctx, data, va, nbytes=1, eva=True)
```

8.482.4. Description

This optional privileged form uses EVA user-mode address translation.

8.482.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.483. SBX

8.483.1. Purpose

Store a byte using a register-indexed address.

8.483.2. Formats

nanoMIPS

sbx.32

Assembly

SBX {rs:gpr}({rt:gpr})

Availability

!(NMS)

Decode priority

0

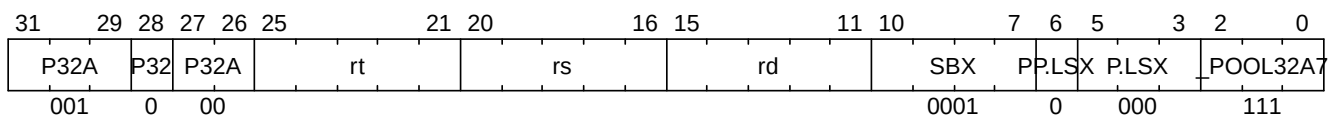


Figure 1340. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.483.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Store")
data = zero_extend(ctx.GPR[operands.rd], from_nbits=8)
write_memory_at_va(ctx, data, va, nbytes=1)
```

8.483.4. Description

The effective address is formed from two full-width GPR values.

8.483.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.484. SC

8.484.1. Purpose

Conditionally store a word from a GPR.

8.484.2. Formats

MIPS

sc

Assembly

SC {rt:gpr}, {offset:s16}({rs:gpr})

Availability

AR < R6

Decode priority

0

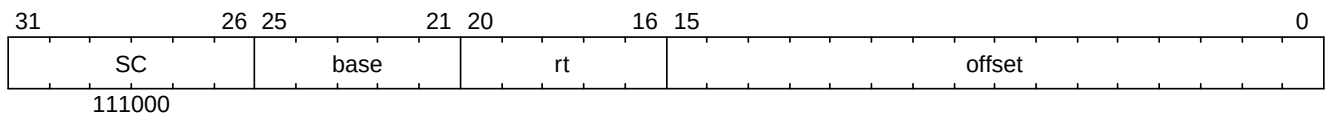


Figure 1341. Encoding (32 bits)

Decode

```
return ScOperands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=16),  
    ru=None,  
)
```

sc.r6

Assembly

SC {rt:gpr}, {offset:s9}({rs:gpr})

Availability

AR >= R6

Decode priority

0

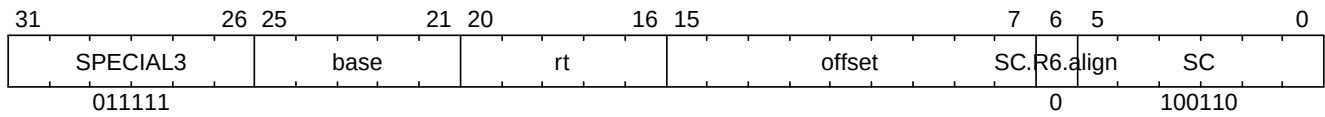


Figure 1342. Encoding (32 bits)

Decode

```
return ScOperands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=9),  
    ru=None,  
)
```

microMIPS

sc.32

Assembly

SC {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6

Decode priority

0

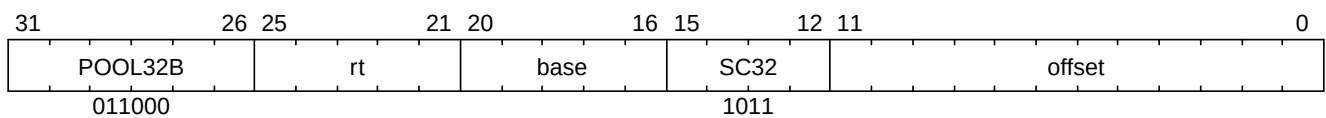


Figure 1343. Encoding (32 bits)

Decode

```
return ScOperands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(offset, from_nbits=12),  
    ru=None,  
)
```

sc.r6.32

Assembly

SC {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR >= R6

Decode priority

0

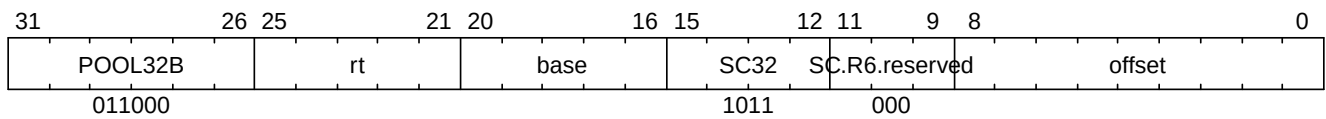


Figure 1344. Encoding (32 bits)

Decode

```
return ScOperands(
    rt=rt,
    rs=base,
    offset=sign_extend(offset, from_nbits=9),
    ru=None,
)
```

nanoMIPS

sc.32

Assembly

SC {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

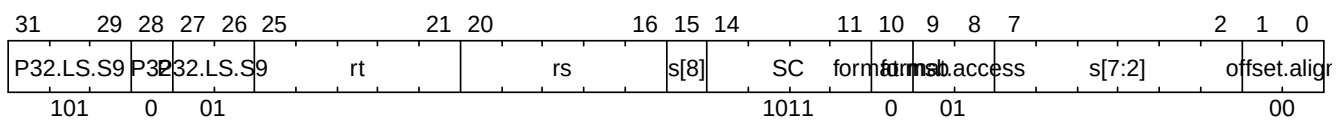


Figure 1345. Encoding (32 bits)

Decode

```
return ScOperands(
    rt=rt,
    rs=rs,
    offset=sign_extend(s, from_nbits=9),
    ru=None,
)
```

8.484.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
if va & 3:
    raise AddressErrorStore(va, "unaligned conditional store")
data = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
success = atomic_write_memory_at_va(ctx, data, va, nbytes=4)
```

```
ctx.GPR[operands.rt] = 1 if success else 0
```

8.484.4. Description

SC computes a sign-extended effective address and performs an atomic word store only when CP0 LLAddr.LLB is set and the implementation's linked address matches. The reservation is cleared and rt receives one on success or zero on failure.

8.484.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.485. SCD

8.485.1. Purpose

Conditionally store a doubleword.

8.485.2. Formats

MIPS

scd

Assembly

SCD {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

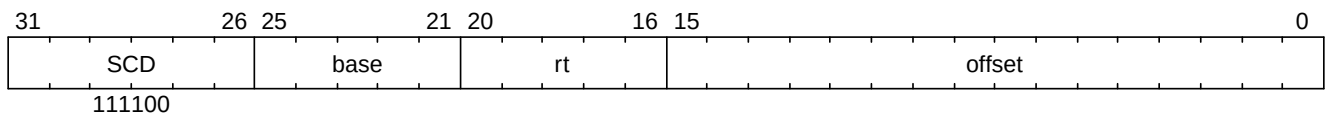


Figure 1346. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=16), None)
```

scd.r6

Assembly

SCD {offset:s9}({rs:gpr})

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

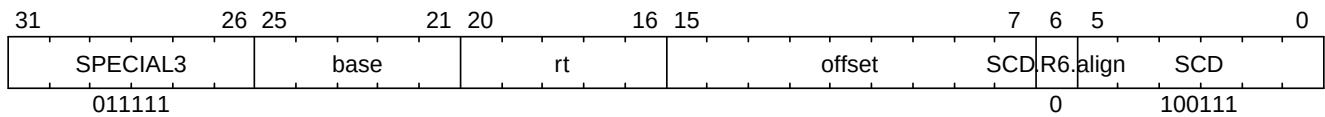


Figure 1347. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=9), None)
```

microMIPS

scd.32

Assembly

```
SCD {rt:gpr}, {offset:s12}({rs:gpr})
```

Availability

MMAR < R6 && AT >= MIPS64_32

Decode priority

0

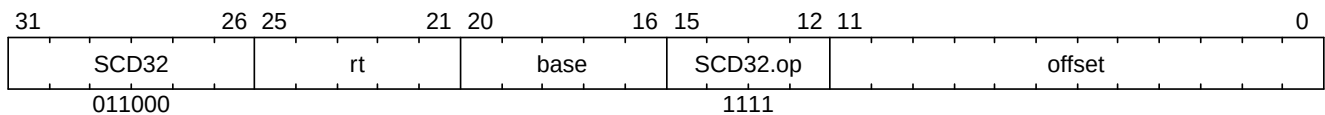


Figure 1348. Encoding (32 bits)

Decode

```
del ctx, exec
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=12), None)
```

scd.r6.32

Assembly

```
SCD {rt:gpr}, {offset:s9}({rs:gpr})
```

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

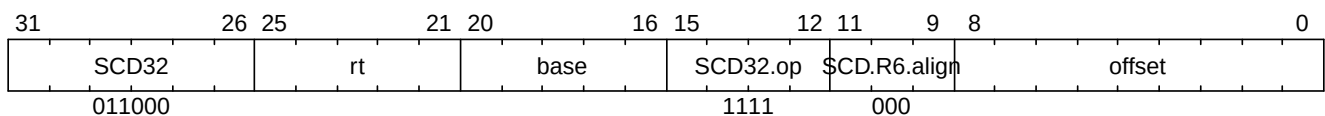


Figure 1349. Encoding (32 bits)

Decode

```
del ctx, exec
return AtomicMemoryOperands(rt, base, sign_extend(offset, from_nbits=9), None)
```

nanoMIPS

scd.32

Assembly

```
SCD {rt:gpr}, {offset:s9}({rs:gpr})
```

Availability

AT >= MIPS64_32

Decode priority

0

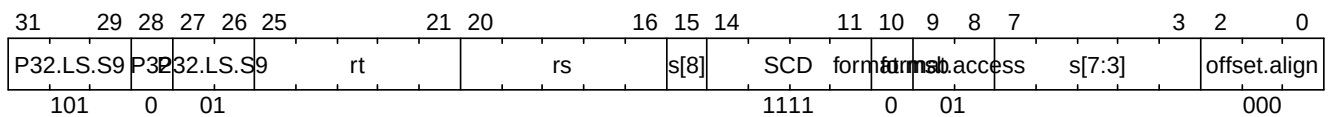


Figure 1350. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, rs, sign_extend(s, from_nbits=9), None)
```

8.485.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_atomic_memory_access(ctx, operands, store=True, pair=False)
```

8.485.4. Description

SCD atomically stores a 64-bit value when it matches an eight-byte linked reservation and writes success to the source register.

8.485.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.486. SCDP

8.486.1. Purpose

Conditionally store a pair of doublewords.

8.486.2. Formats

MIPS

`scdp`

Assembly

`SCDP {ru:gpr}, ({rs:gpr})`

Availability

AT >= MIPS64_32 && AR >= R6

Decode priority

0

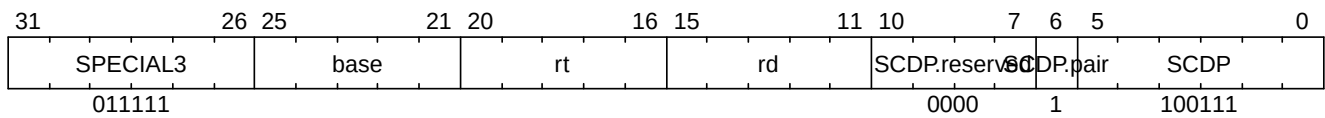


Figure 1351. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(0, from_nbits=1), rd)
```

microMIPS

`scdp.32`

Assembly

`SCDP {rt:gpr}, {ru:gpr}, ({rs:gpr})`

Availability

MMAR >= R6 && AT >= MIPS64_32

Decode priority

0

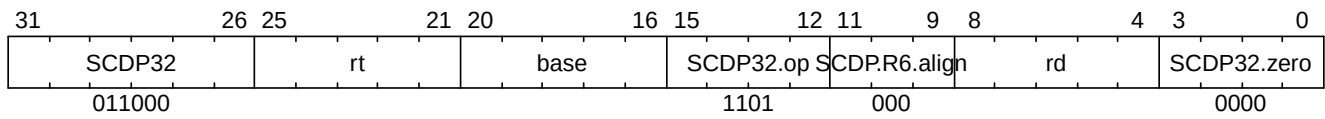


Figure 1352. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, base, sign_extend(0, from_nbits=1), rd)
```

nanoMIPS

scdp.32

Assembly

SCDP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

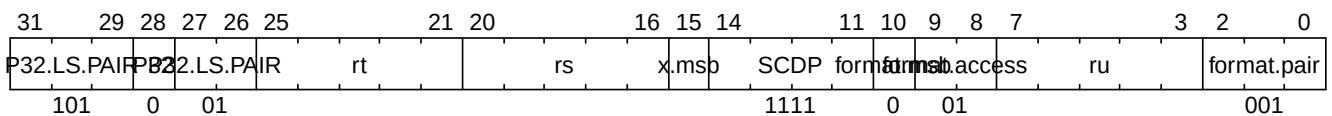


Figure 1353. Encoding (32 bits)

Decode

```
return AtomicMemoryOperands(rt, rs, sign_extend(0, from_nbits=1), ru)
```

8.486.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_atomic_memory_access(ctx, operands, store=True, pair=True)
```

8.486.4. Description

SCDP atomically stores a 128-bit pair when it matches a sixteen-byte linked reservation and writes success to the first source register.

8.486.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled; paired LL/SC operations are unavailable
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.487. SCE

8.487.1. Purpose

Store-conditional word using EVA addressing.

8.487.2. Formats

nanoMIPS

sce.32

Assembly

SCE {rt:gpr}, {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

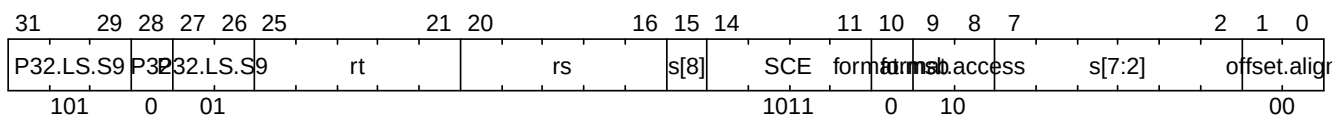


Figure 1354. Encoding (32 bits)

Decode

```
return ScOperands(  
    rt=rt,  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    ru=None,  
)
```

8.487.3. Operation

```
if not ctx.C0.Config5.EVA:  
    raise ReservedInstruction("SCE requires EVA")  
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)  
va = effective_address(ctx, base, operands.offset, "Store", eva=True)  
if va & 3:  
    raise AddressErrorStore(va, "unaligned conditional store")  
data = zero_extend(ctx.GPR[operands.rt], from_nbits=32)  
success = atomic_write_memory_at_va(ctx, data, va, nbytes=4)  
ctx.GPR[operands.rt] = 1 if success else 0
```

8.487.4. Description

SCE performs a privileged EVA conditional word store.

8.487.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — SCE requires EVA
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.488. SCWP

8.488.1. Purpose

Store-conditional word pair.

8.488.2. Formats

MIPS

scwp

Assembly

SCWP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

AR >= R6

Decode priority

0

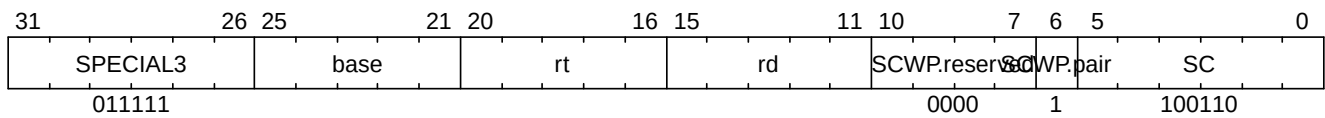


Figure 1355. Encoding (32 bits)

Decode

```
return ScOperands(  
    rt=rt,  
    rs=base,  
    offset=sign_extend(0, from_nbits=1),  
    ru=rd,  
)
```

microMIPS

scwp.32

Assembly

SCWP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

MMAR >= R6

Decode priority

0

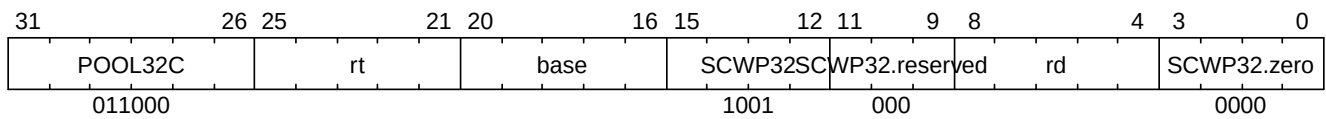


Figure 1356. Encoding (32 bits)

Decode

```
return ScOperands(
    rt=rt,
    rs=base,
    offset=sign_extend(0, from_nbits=1),
    ru=rd,
)
```

nanoMIPS

scwp.32

Assembly

SCWP {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

!(NMS)

Decode priority

0

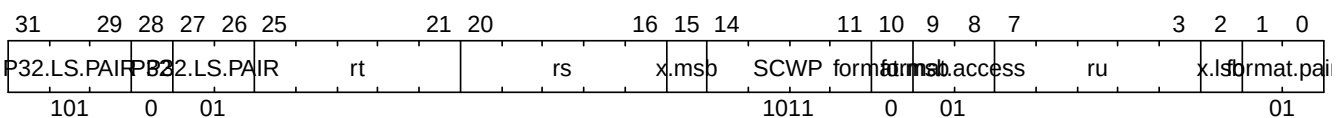


Figure 1357. Encoding (32 bits)

Decode

```
return ScOperands(
    rt=rt,
    rs=rs,
    offset=sign_extend(0, from_nbits=1),
    ru=ru,
)
```

8.488.3. Operation

```
if ctx.C0.Config5.XNP:
    raise ReservedInstruction("SCWP requires word-paired support")
if operands.ru is None:
    raise ValueError("SCWP requires a second source register")
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
```

```

if va & 7:
    raise AddressErrorStore(va, "unaligned conditional store pair")
first = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
second = int(zero_extend(ctx.GPR[operands.ru], from_nbits=32))
if memory_endianness(ctx) == "big":
    data = (first << 32) | second
else:
    data = first | (second << 32)
success = atomic_write_memory_at_va(ctx, data, va, nbytes=8)
ctx.GPR[operands.rt] = 1 if success else 0

```

8.488.4. Description

SCWP conditionally writes two adjacent words from two GPRs.

8.488.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — SCWP requires word-paired support
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.489. SCWPE

8.489.1. Purpose

Store-conditional word pair using EVA addressing.

8.489.2. Formats

nanoMIPS

scwpe.32

Assembly

SCWPE {rt:gpr}, {ru:gpr}, ({rs:gpr})

Availability

EVA

Decode priority

0

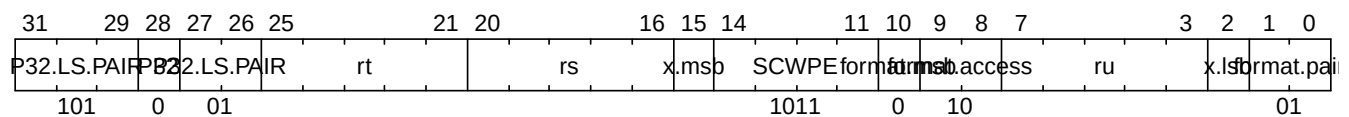


Figure 1358. Encoding (32 bits)

Decode

```
return ScOperands(
    rt=rt,
    rs=rs,
    offset=sign_extend(0, from_nbits=1),
    ru=ru,
)
```

8.489.3. Operation

```

if ctx.C0.Config5.XNP:
    raise ReservedInstruction("SCWPE requires word-paired support")
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("SCWPE requires EVA")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
if operands.ru is None:
    raise ValueError("SCWPE requires a second source register")
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store", eva=True)
if va & 7:
    raise AddressErrorStore(va, "unaligned conditional store pair")

```

```

first = int(zero_extend(ctx.GPR[operands.rt], from_nbits=32))
second = int(zero_extend(ctx.GPR[operands.ru], from_nbits=32))
if memory_endianness(ctx) == "big":
    data = (first << 32) | second
else:
    data = first | (second << 32)
success = atomic_write_memory_at_va(ctx, data, va, nbytes=8)
ctx.GPR[operands.rt] = 1 if success else 0

```

8.489.4. Description

SCWPE performs a privileged EVA paired conditional store.

8.489.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — SCWPE requires word-paired support; SCWPE requires EVA
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.490. SD

8.490.1. Purpose

Store a 64-bit doubleword to memory.

8.490.2. Formats

MIPS

sd

Assembly

SD {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

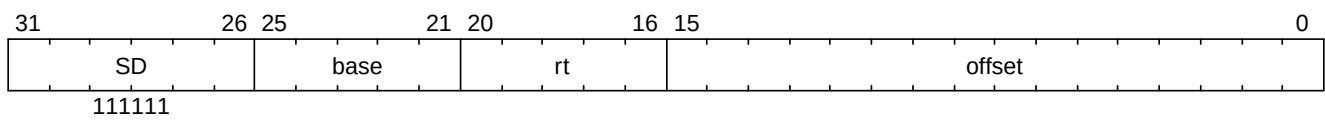


Figure 1359. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, base, sign_extend(offset, from_nbits=16))
```

microMIPS

sd

Assembly

SD {rt:gpr}, {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

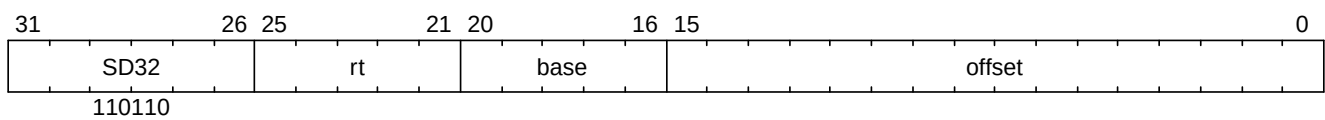


Figure 1360. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, base, sign_extend(offset, from_nbits=16))
```

nanoMIPS

sd.gp

Assembly

SD {rt:gpr}, {offset:u21}(gp)

Availability

AT >= MIPS64_32

Decode priority

0



Figure 1361. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, 28, zero_extend(u, from_nbits=21))
```

sd.s9

Assembly

SD {rt:gpr}, {offset:s9}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

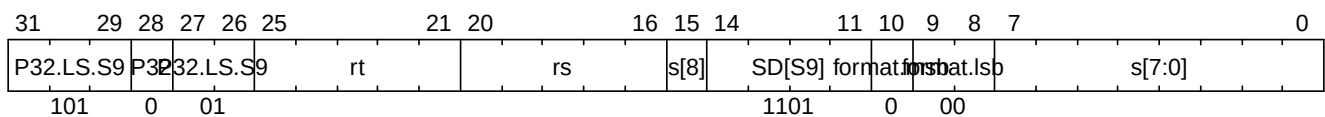


Figure 1362. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

sd.u12

Assembly

SD {rt:gpr}, {offset:u12}({rs:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

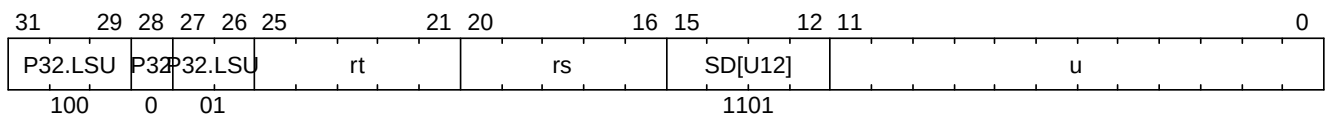


Figure 1363. Encoding (32 bits)

Decode

```
return DoubleMemoryOperands(rt, rs, zero_extend(u, from_nbits=12))
```

8.490.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_memory_access(ctx, operands, nbytes=8, signed_load=False, store=True)
```

8.490.4. Description

SD forms a full-width effective address and stores the low 64 bits of the source GPR.

8.490.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.491. SDBBP

8.491.1. Purpose

Cause a software debug breakpoint exception.

8.491.2. Formats

MIPS

sdbbp

Assembly

SDBBP {code:u20}

Availability

AR < R6

Decode priority

0

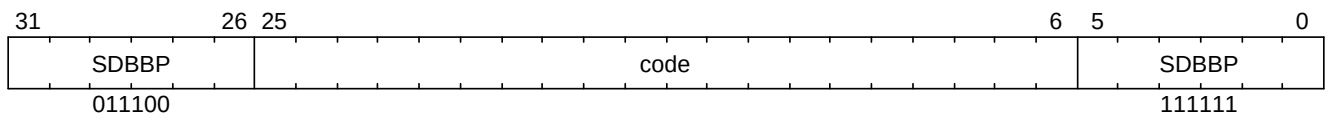


Figure 1364. Encoding (32 bits)

Decode

```
return Sdbbp0operands(code=zero_extend(code, from_nbits=20))
```

sdbbp.r6

Assembly

SDBBP {code:u20}

Availability

AR >= R6

Decode priority

0

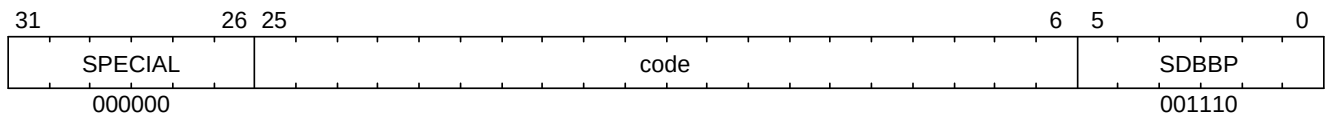


Figure 1365. Encoding (32 bits)

Decode

```
return Sdbbp0operands(code=zero_extend(code, from_nbits=20))
```

nanoMIPS

sdbbp.16

Assembly

SDBBP {code:u3}

Availability

DEBUG

Decode priority

0

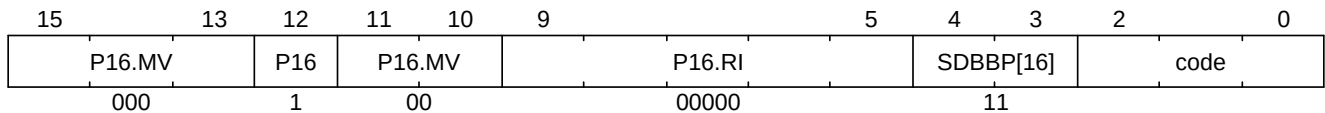


Figure 1366. Encoding (16 bits)

Decode

```
del ctx, exec
return SdbbpOperands(code=zero_extend(code, from_nbits=3))
```

sdbbp.32

Assembly

SDBBP {code:u19}

Availability

DEBUG

Decode priority

0

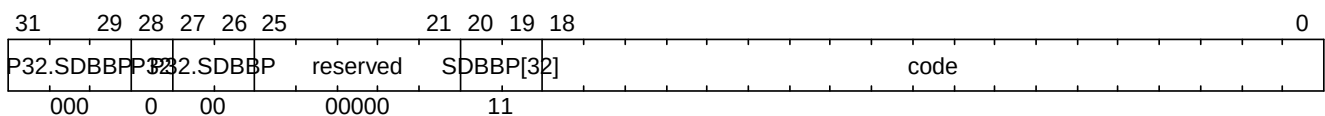


Figure 1367. Encoding (32 bits)

Decode

```
del ctx, exec
return SdbbpOperands(code=zero_extend(code, from_nbits=19))
```

8.491.3. Operation

```
del operands
if not DebugSupportPresent(ctx):
    raise ReservedInstruction("SDBBP requires CP0 debug support")
```

```
if DebugBreakpointRestricted(ctx) and EffectiveKSU(ctx) != 0:
    raise ReservedInstruction("SDBBP is restricted outside kernel mode")

set_debug_breakpoint(ctx)
raise DebugBreakpointException("SDBBP")
```

8.491.4. Description

SDBBP carries an implementation-visible code in a compact three-bit or full nineteen-bit field. It is available only when the selected profile implements debug support. If debug is unavailable, or SDBBP is restricted by Config5.SBRI outside kernel mode, it raises RI; otherwise it sets CP0.Debug.DBp and raises a software debug breakpoint exception.

8.491.5. Exceptions

- **BP** (Software Debug Breakpoint) — SDBBP
- **RI** (Reserved Instruction) — SDBBP requires CP0 debug support; SDBBP is restricted outside kernel mode

8.492. SDC1

8.492.1. Purpose

SDC1 a CP1 register to or from memory.

8.492.2. Formats

MIPS

sdc1

Assembly

SDC1 {ft:fpr}, {offset:s16}({base:gpr})

Availability

true

Decode priority

0

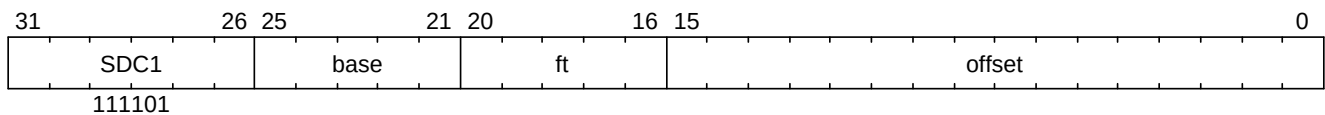


Figure 1368. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(  
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)  
)
```

microMIPS

sdc1.32

Assembly

SDC1 {ft:fpr}, {offset:s16}({base:gpr})

Availability

true

Decode priority

0

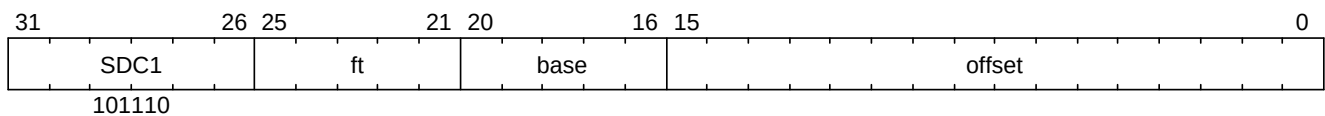


Figure 1369. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(  
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)  
)
```

nanoMIPS

sdc1.gp

Assembly

SDC1 {ft:fpr}, ...

Availability

true

Decode priority

0

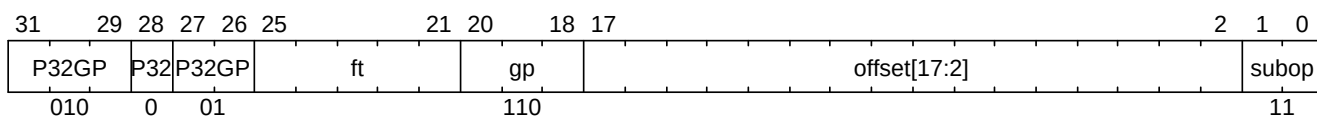


Figure 1370. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(  
    ft=ft, base=28, offset=zero_extend(offset, from_nbits=16) << 2  
)
```

sdc1.s9

Assembly

SDC1 {ft:fpr}, ...

Availability

true

Decode priority

0

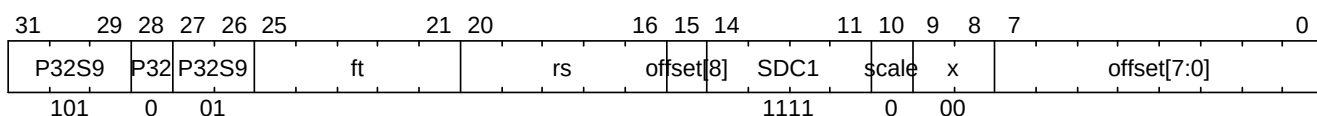


Figure 1371. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=sign_extend(offset, from_nbits=9))
```


sdc1.u12

Assembly

SDC1 {ft:fpr}, ...

Availability

true

Decode priority

0

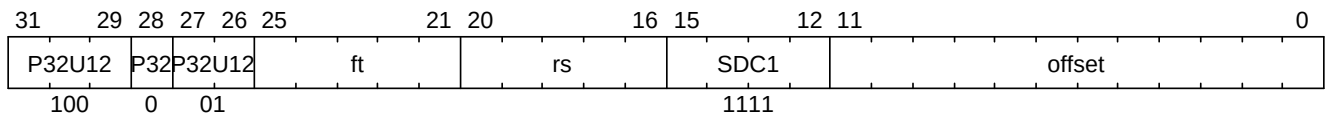


Figure 1372. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=zero_extend(offset, from_nbits=12))
```

8.492.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
write_memory_at_va(ctx, get_fpr(ctx, operands.ft, "64"), va, nbytes=8)
```

8.492.4. Description

SDC1 performs CP1 access checks before its memory access.

8.492.5. Exceptions

- **ADEL** (Address Error) — The CP1 load address is not valid.
- **ADES** (Address Error (store)) — A store address is invalid.; The CP1 store address is not valid.
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the required FPU resource is unavailable.
- **TLBL** (TLB Load) — The CP1 load address could not be translated.

- **TLBS** (TLB Store) — A store address could not be translated.; The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.493. SDC2

8.493.1. Purpose

Store a doubleword from a CP2 register to memory.

8.493.2. Formats

microMIPS

sdc2.32

Assembly

SDC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR < R6

Decode priority

0

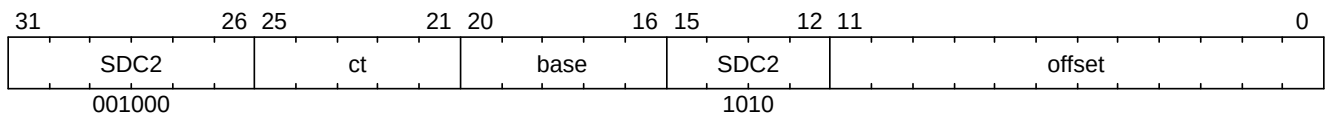


Figure 1373. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

sdc2.r6.32

Assembly

SDC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR >= R6

Decode priority

0

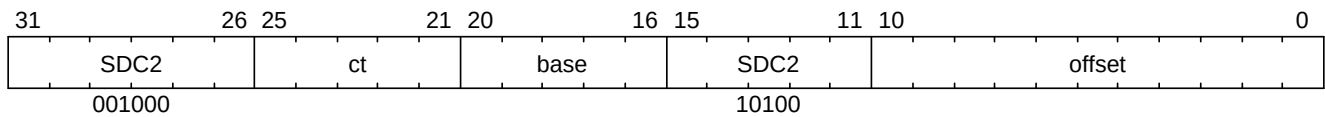


Figure 1374. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

8.493.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("SDC2 execution is not implemented")
```

8.493.4. Description

SDC2 performs the CP2 access check before any CP2 operation is attempted.

8.493.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; SDC2 execution is not implemented

8.494. SDL

8.494.1. Purpose

Perform the legacy unaligned doubleword SDL operation.

8.494.2. Formats

MIPS

sd1

Assembly

SDL {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

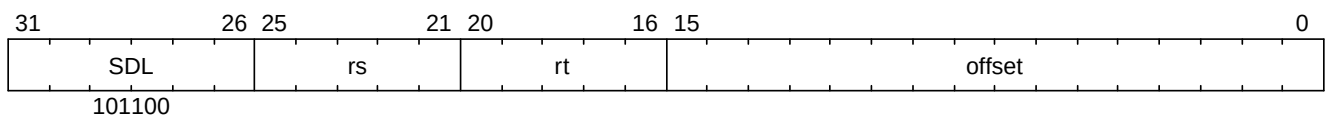


Figure 1375. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=16)
)
```

microMIPS

sd1.32

Assembly

SDL {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6 && AT >= MIPS64_32

Decode priority

0

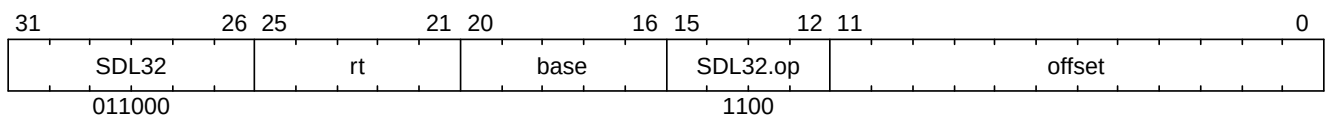


Figure 1376. Encoding (32 bits)

```
del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=base, offset=sign_extend(offset, from_nbits=12)
)
```

8.494.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_legacy_store(ctx, operands, left=True)
```

8.494.4. Description

The naturally aligned doubleword containing the effective address is merged according to the active memory endianness. This legacy MIPS64 instruction is unavailable in Release 6.

8.494.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.495. SDM

8.495.1. Purpose

Store multiple doublewords from sequential GPRs.

8.495.2. Formats

nanoMIPS

sdm.32

Assembly

SDM {rt:gpr}, {offset:s9}({rs:gpr}), {count:u3}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

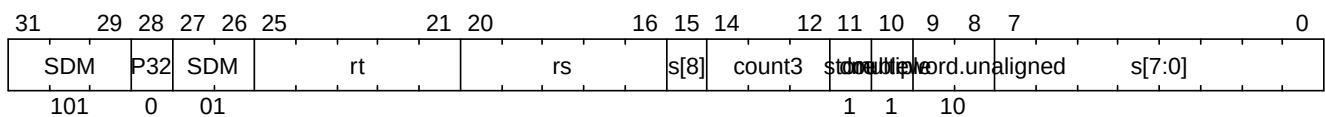


Figure 1377. Encoding (32 bits)

Decode

```
return DoubleMultipleOperands(  
    rt=rt,  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    count=8 if count3 == 0 else count3,  
)
```

8.495.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_multiple_access(ctx, operands, store=True, unaligned=False)
```

8.495.4. Description

SDM transfers 64-bit elements to consecutive aligned addresses, wrapping register numbers after r31 to r16.

8.495.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.496. SDPC

8.496.1. Purpose

Store a doubleword to a PC-relative address.

8.496.2. Formats

nanoMIPS

sdpc.48

Assembly

SDPC {rt:gpr}, {offset:s32}

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

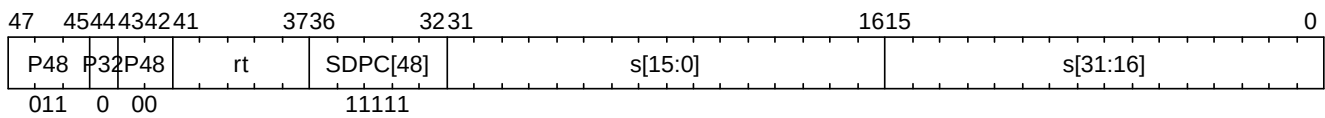


Figure 1378. Encoding (48 bits)

Decode

```
return DoublePcRelativeOperands(rt, sign_extend(s, from_nbits=32), False)
```

8.496.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_double_pc_relative_access(ctx, operands, store=True)
```

8.496.4. Description

SDPC[48] adds its signed 32-bit displacement to the following-instruction address.

8.496.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBS** (TLB Store) — A store address could not be translated.

- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.497. SDR

8.497.1. Purpose

Perform the legacy unaligned doubleword SDR operation.

8.497.2. Formats

MIPS

sdr

Assembly

SDR {offset:s16}({rs:gpr})

Availability

AT >= MIPS64_32 && AR < R6

Decode priority

0

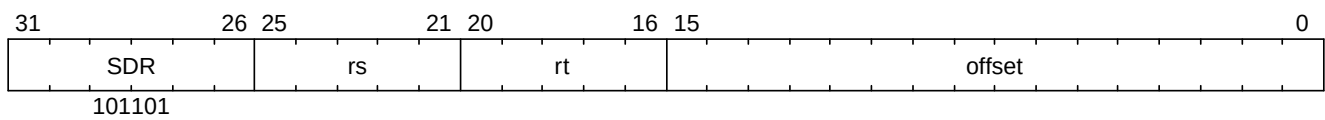


Figure 1379. Encoding (32 bits)

Decode

```
del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=16)
)
```

microMIPS

sdr.32

Assembly

SDR {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6 && AT >= MIPS64_32

Decode priority

0

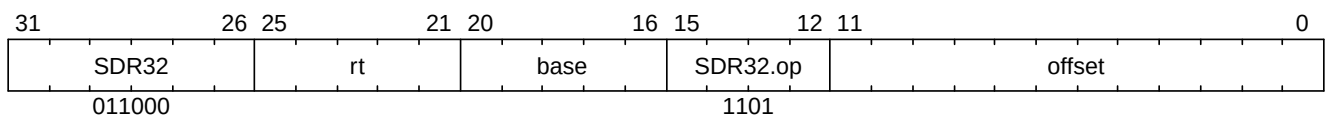


Figure 1380. Encoding (32 bits)

```

del ctx, exec
return DoubleLegacyUnalignedOperands(
    rt=rt, rs=base, offset=sign_extend(offset, from_nbits=12)
)

```

8.497.3. Operation

```

validate_access_to_64bit_operations(ctx)
execute_double_legacy_store(ctx, operands, left=False)

```

8.497.4. Description

The naturally aligned doubleword containing the effective address is merged according to the active memory endianness. This legacy MIPS64 instruction is unavailable in Release 6.

8.497.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.498. SDX

8.498.1. Purpose

Store a doubleword using a register-indexed address.

8.498.2. Formats

nanoMIPS

sdx

Assembly

SDX {rs:gpr}({rt:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

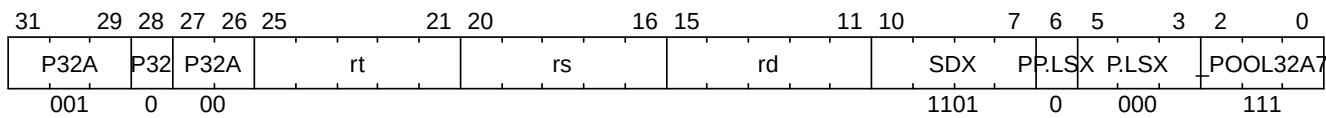


Figure 1381. Encoding (32 bits)

Decode

```
return IndexedDoubleMemoryOperands(rd, rs, rt)
```

8.498.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_indexed_double_memory_access(
    ctx, operands, nbytes=8, signed_load=False, scale=0, store=True
)
```

8.498.4. Description

SDX adds two full-width GPR values and stores the source doubleword.

8.498.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.499. SDXC1

8.499.1. Purpose

Indexed SDXC1 access through Coprocessor 1.

8.499.2. Formats

MIPS

sdxc1

Assembly

SDXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

AR >= R2_3_5 && AR < R6

Decode priority

0

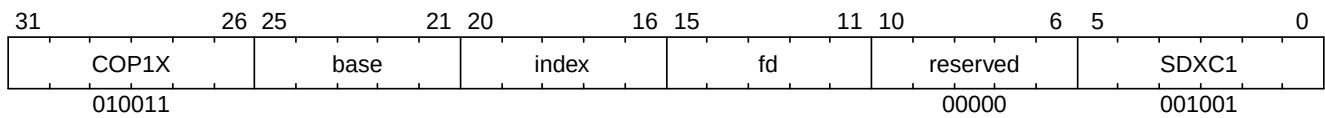


Figure 1382. Encoding (32 bits)

Decode

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

microMIPS

sdxc1.32

Assembly

SDXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

MMAR < R6

Decode priority

0

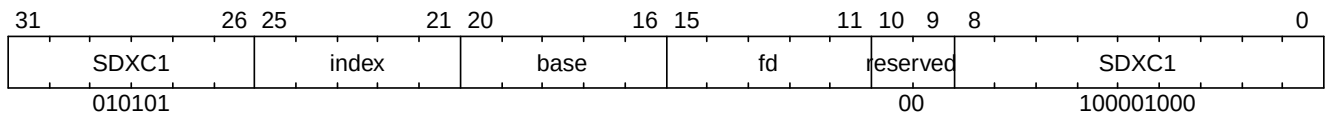


Figure 1383. Encoding (32 bits)

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

8.499.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
index = sign_extend(ctx.GPR[operands.index], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, index, "Store")
write_memory_at_va(ctx, get_fpr(ctx, operands.fd, "64"), va, nbytes=8)
```

8.499.4. Description

SDXC1 performs its CP1 access checks before transferring data.

8.499.5. Exceptions

- **ADEL** (Address Error) — The CP1 load address is not valid.
- **ADES** (Address Error (store)) — A store address is invalid.; The CP1 store address is not valid.
- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.
- **TLBL** (TLB Load) — The CP1 load address could not be translated.
- **TLBS** (TLB Store) — A store address could not be translated.; The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.500. SDXS

8.500.1. Purpose

Store a doubleword using a scaled register-indexed address.

8.500.2. Formats

nanoMIPS

sdxs

Assembly

SDXS {rs:gpr}({rt:gpr})

Availability

AT >= MIPS64_32

Decode priority

0

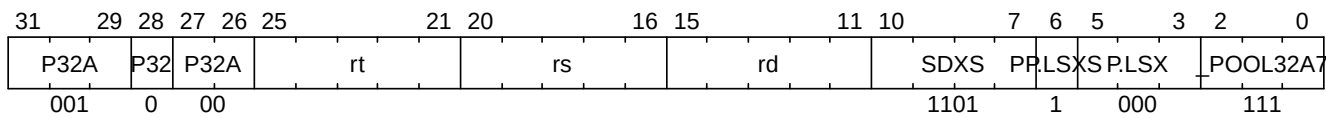


Figure 1384. Encoding (32 bits)

Decode

```
return IndexedDoubleMemoryOperands(rd, rs, rt)
```

8.500.3. Operation

```
validate_access_to_64bit_operations(ctx)
execute_indexed_double_memory_access(
    ctx, operands, nbytes=8, signed_load=False, scale=3, store=True
)
```

8.500.4. Description

SDXS adds the base GPR to eight times the index GPR.

8.500.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.501. SEB

8.501.1. Purpose

Sign-extend the low byte of a source GPR and place the result in a destination GPR.

8.501.2. Formats

MIPS

seb

Assembly

SEB {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5

Decode priority

0

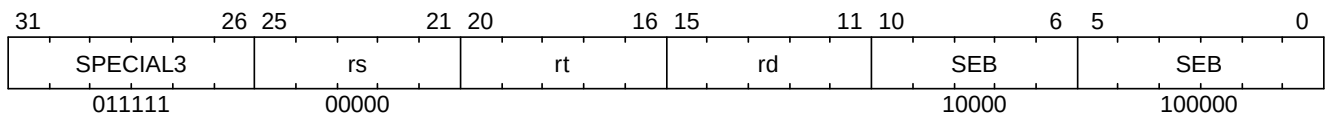


Figure 1385. Encoding (32 bits)

Decode

```
return Seb0perands(rt=rd, rs=rt)
```

microMIPS

seb.32

Assembly

SEB {rt:gpr}, {rs:gpr}

Availability

true

Decode priority

0

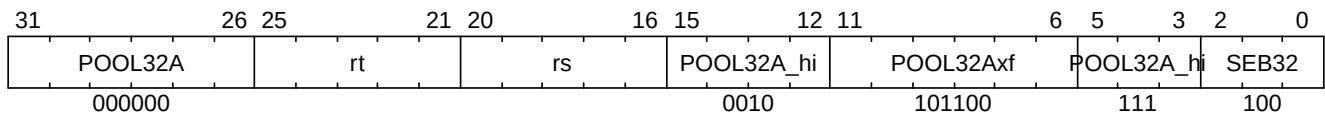


Figure 1386. Encoding (32 bits)

Decode

```
return SebOperands(rt=rt, rs=rs)
```

nanoMIPS

seb.32

Assembly

SEB {rt:gpr}, {rs:gpr}

Availability

!(NMS)

Decode priority

0

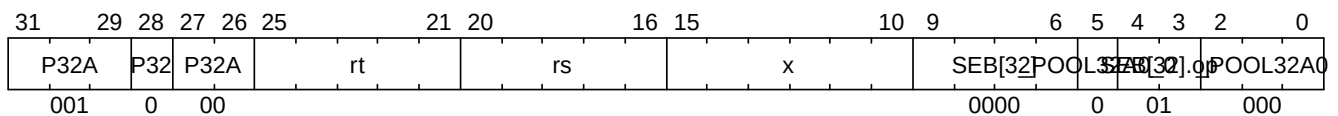


Figure 1387. Encoding (32 bits)

Decode

```
return SebOperands(rt=rt, rs=rs)
```

8.501.3. Operation

```
ctx.GPR[operands.rt] = sign_extend(ctx.GPR[operands.rs], from_nbits=8)
```

8.501.4. Description

SEB[32] sign-extends exactly eight bits of **rs** to the implemented GPR width. The form is reserved when the nanoMIPS subset (NMS) is enabled.

8.501.5. Exceptions

- **RI** (Reserved Instruction) — SEB[32] is reserved when CP0 Config5.NMS is set.; SEB[32] is reserved when C0.Config5.NMS is set.

8.502. SEH

8.502.1. Purpose

Sign-extend the low halfword of a source GPR and place the result in a destination GPR.

8.502.2. Formats

MIPS

seh

Assembly

SEH {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5

Decode priority

0

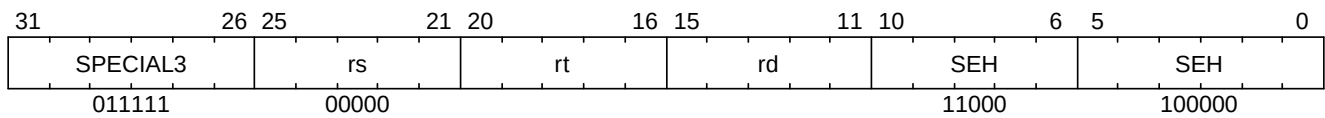


Figure 1388. Encoding (32 bits)

Decode

```
return Seh0perands(rt=rd, rs=rt)
```

microMIPS

seh.32

Assembly

SEH {rt:gpr}, {rs:gpr}

Availability

true

Decode priority

0

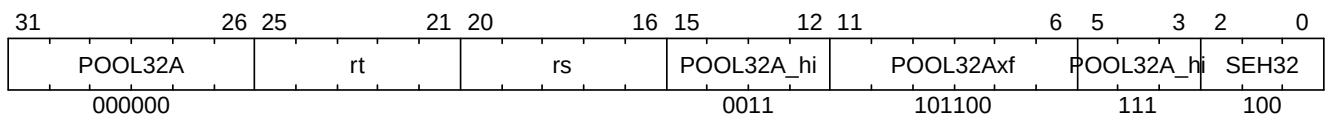


Figure 1389. Encoding (32 bits)

Decode

```
return SehOperands(rt=rt, rs=rs)
```

nanoMIPS

seh.32

Assembly

SEH {rt:gpr}, {rs:gpr}

Availability

true

Decode priority

0

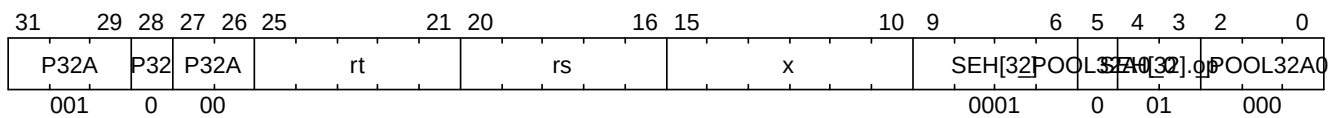


Figure 1390. Encoding (32 bits)

Decode

```
return SehOperands(rt=rt, rs=rs)
```

8.502.3. Operation

```
ctx.GPR[operands.rt] = sign_extend(ctx.GPR[operands.rs], from_nbits=16)
```

8.502.4. Description

SEH[32] sign-extends exactly sixteen bits of **rs** to the implemented GPR width.

8.502.5. Exceptions

None.

8.503. SEL.D

8.503.1. Purpose

Select a D-format source using fd bit zero.

8.503.2. Formats

MIPS

sel.d

Assembly

SEL.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

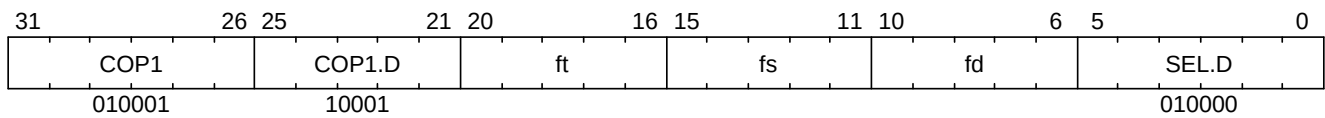


Figure 1391. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

sel.d

Assembly

SEL.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

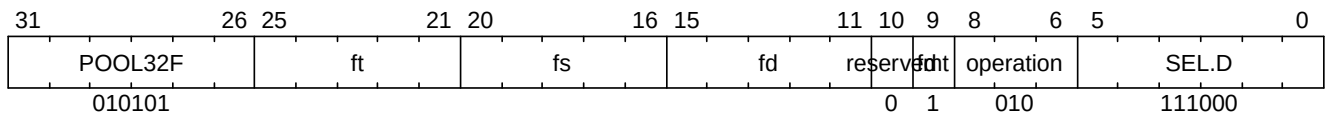


Figure 1392. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

sel.d

Assembly

SEL.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

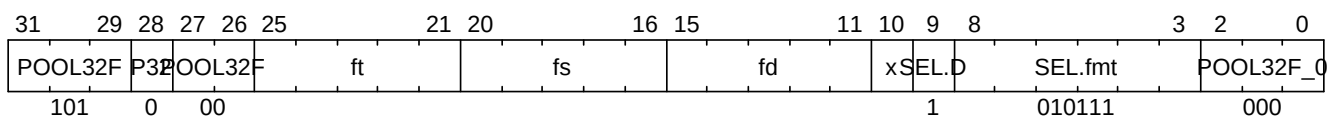


Figure 1393. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.503.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
set_fpr(
    ctx,
    operands.fd,
    get_fpr(
        ctx, operands.ft if get_fpr(ctx, operands.fd, "D") & 1 else operands.fs, "D"
    ),
    "D",
)
```

8.503.4. Description

SEL.D validates CP1 and D-format support before updating the destination FPR.

8.503.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.504. SEL.S

8.504.1. Purpose

Select an S-format source using fd bit zero.

8.504.2. Formats

MIPS

`sel.s`

Assembly

`SEL.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

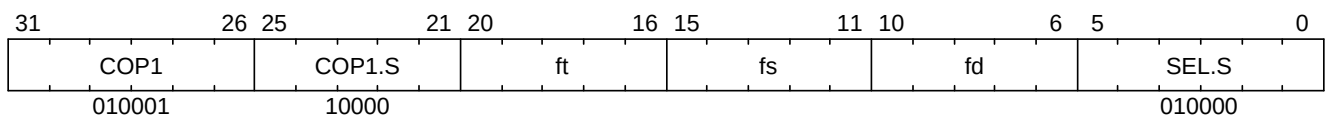


Figure 1394. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`sel.s`

Assembly

`SEL.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

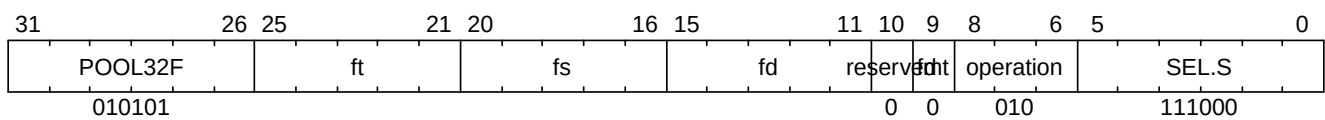


Figure 1395. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

sel.s

Assembly

SEL.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

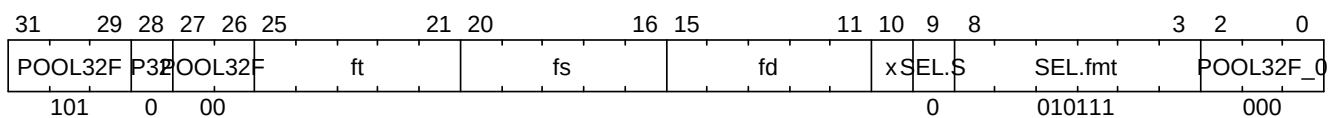


Figure 1396. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.504.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
set_fpr(
    ctx,
    operands.fd,
    get_fpr(
        ctx, operands.ft if get_fpr(ctx, operands.fd, "S") & 1 else operands.fs, "S"
    ),
    "S",
)
```

8.504.4. Description

SEL.S validates CP1 and S-format support before updating the destination FPR.

8.504.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.505. SELEQZ

8.505.1. Purpose

Conditionally select rs or zero according to rt for SELEQZ.

8.505.2. Formats

MIPS

seleqz

Assembly

SELEQZ {rs:gpr}, {rt:gpr}

Availability

AR >= R6

Decode priority

0

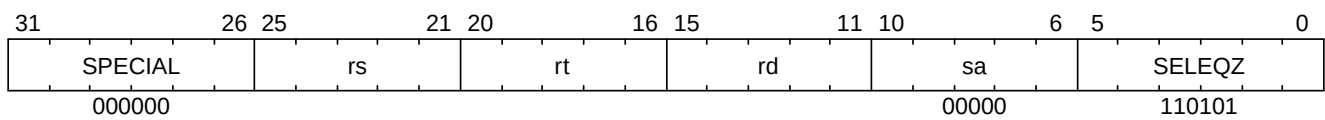


Figure 1397. Encoding (32 bits)

Decode

```
return ConditionalSelectOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

seleqz.32

Assembly

SELEQZ {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

MMAR >= R6

Decode priority

0

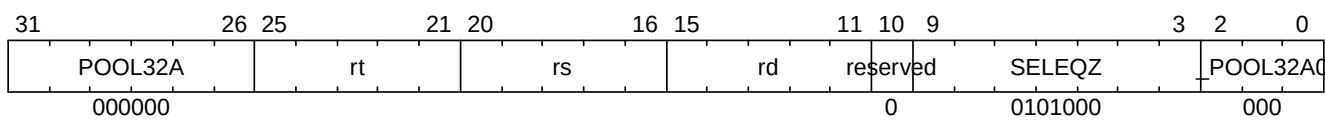


Figure 1398. Encoding (32 bits)

Decode

```
return ConditionalSelectOperands(rd=rd, rs=rs, rt=rt)
```

8.505.3. Operation

```
ctx.GPR[operands.rd] = ctx.GPR[operands.rs] if ctx.GPR[operands.rt] == 0 else 0
```

8.505.4. Description

The destination receives the full-width value of rs when the selected zero/nonzero test on rt succeeds, and zero otherwise.

8.505.5. Exceptions

None.

8.506. SELEQZ.D

8.506.1. Purpose

Select a D-format value when ft bit zero is clear.

8.506.2. Formats

MIPS

seleqz.d

Assembly

SELEQZ.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

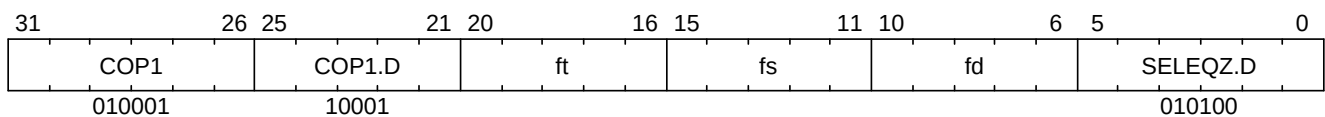


Figure 1399. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

seleqz.d

Assembly

SELEQZ.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

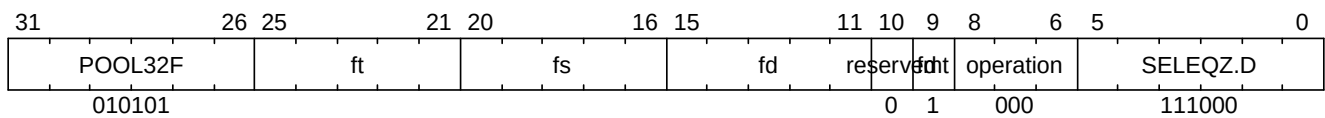


Figure 1400. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

seleqz.d

Assembly

SELEQZ.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

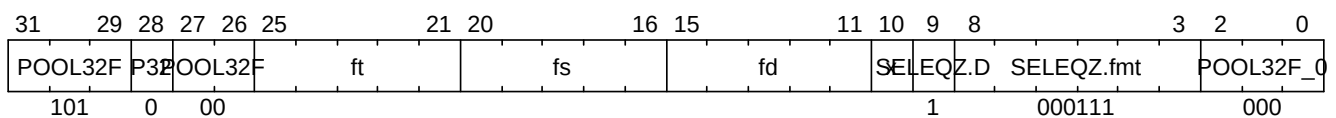


Figure 1401. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.506.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
set_fpr(
    ctx,
    operands.fd,
    get_fpr(ctx, operands.fs, "D")
    if get_fpr(ctx, operands.ft, "D") & 1 == 0
    else 0,
    "D",
)
```

8.506.4. Description

SELEQZ.D validates CP1 and D-format support before updating the destination FPR.

8.506.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.507. SELEQZ.S

8.507.1. Purpose

Select an S-format value when ft bit zero is clear.

8.507.2. Formats

MIPS

seleqz.s

Assembly

SELEQZ.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR >= R6

Decode priority

0

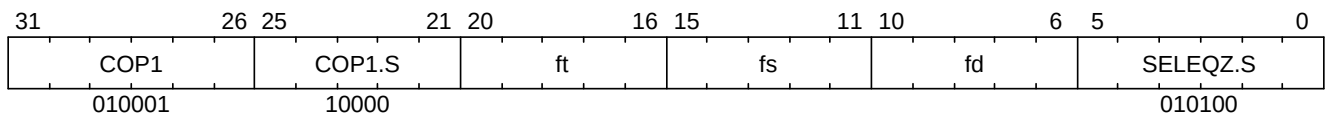


Figure 1402. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

seleqz.s

Assembly

SELEQZ.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR >= R6

Decode priority

0

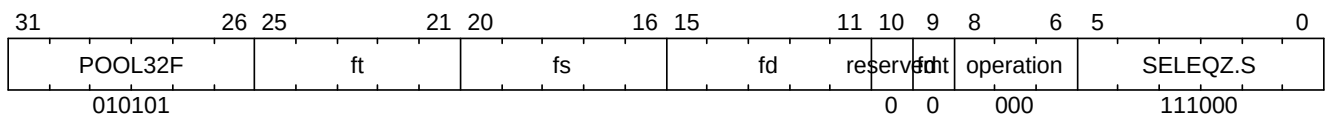


Figure 1403. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

seleqz.s

Assembly

SELEQZ.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

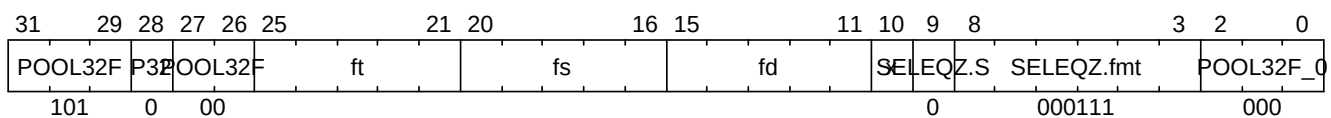


Figure 1404. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.507.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
set_fpr(
    ctx,
    operands.fd,
    get_fpr(ctx, operands.fs, "S")
    if get_fpr(ctx, operands.ft, "S") & 1 == 0
    else 0,
    "S",
)
```

8.507.4. Description

SELEQZ.S validates CP1 and S-format support before updating the destination FPR.

8.507.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported.

supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.508. SELNEZ

8.508.1. Purpose

Conditionally select rs or zero according to rt for SELNEZ.

8.508.2. Formats

MIPS

`selnez`

Assembly

`SELNEZ {rs:gpr}, {rt:gpr}`

Availability

AR >= R6

Decode priority

0

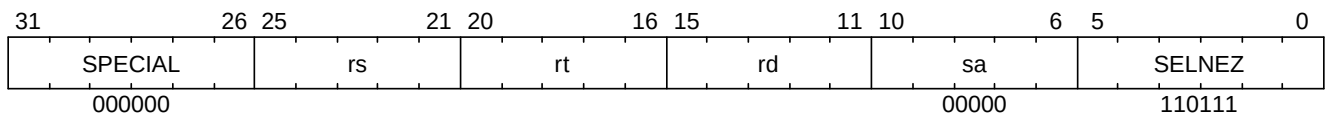


Figure 1405. Encoding (32 bits)

Decode

```
return ConditionalSelectOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`selnez.32`

Assembly

`SELNEZ {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

MMAR >= R6

Decode priority

0

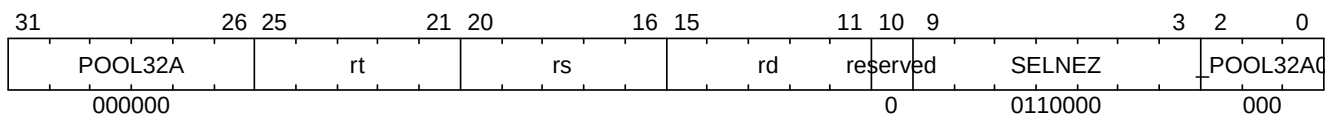


Figure 1406. Encoding (32 bits)

Decode

```
return ConditionalSelectOperands(rd=rd, rs=rs, rt=rt)
```

8.508.3. Operation

```
ctx.GPR[operands.rd] = ctx.GPR[operands.rs] if ctx.GPR[operands.rt] != 0 else 0
```

8.508.4. Description

The destination receives the full-width value of rs when the selected zero/nonzero test on rt succeeds, and zero otherwise.

8.508.5. Exceptions

None.

8.509. SELNEZ.D

8.509.1. Purpose

Select a D-format value when ft bit zero is set.

8.509.2. Formats

MIPS

`selnez.d`

Assembly

`SELNEZ.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

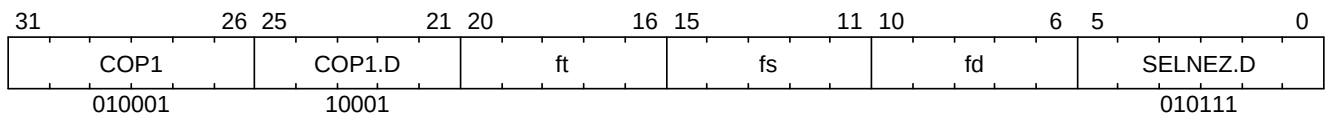


Figure 1407. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`selnez.d`

Assembly

`SELNEZ.D {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

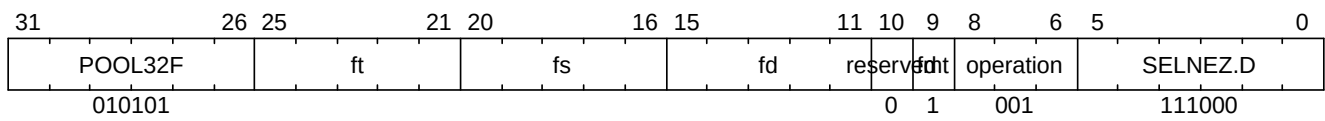


Figure 1408. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

selnez.d

Assembly

SELNEZ.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

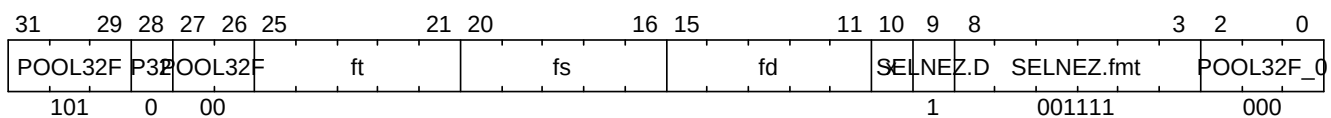


Figure 1409. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.509.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
set_fpr(
    ctx,
    operands.fd,
    get_fpr(ctx, operands.fs, "D") if get_fpr(ctx, operands.ft, "D") & 1 else 0,
    "D",
)
```

8.509.4. Description

SELNEZ.D validates CP1 and D-format support before updating the destination FPR.

8.509.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported

8.510. SELNEZ.S

8.510.1. Purpose

Select an S-format value when ft bit zero is set.

8.510.2. Formats

MIPS

`selnez.s`

Assembly

`SELNEZ.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

AR >= R6

Decode priority

0

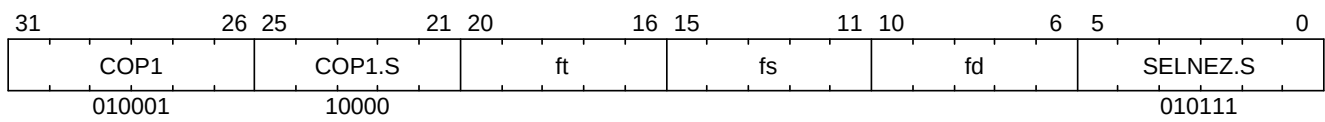


Figure 1410. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

`selnez.s`

Assembly

`SELNEZ.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

MMAR >= R6

Decode priority

0

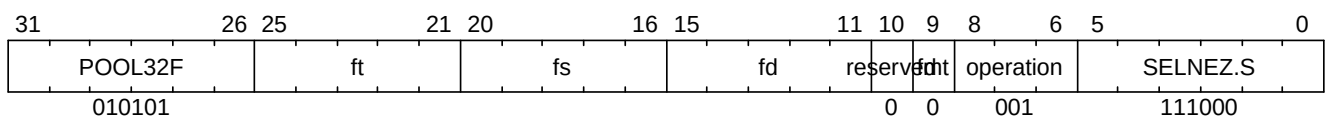


Figure 1411. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

`selnez.s`

Assembly

`SELNEZ.S {fd:fpr}, {fs:fpr}, {ft:fpr}`

Availability

true

Decode priority

0

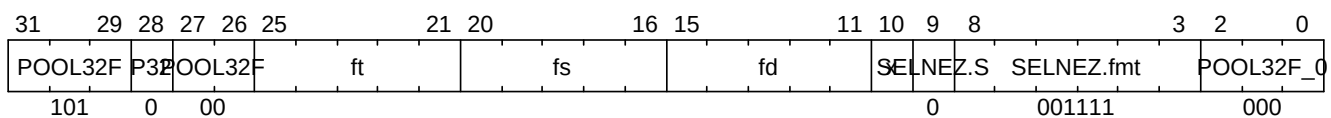


Figure 1412. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.510.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
set_fpr(
    ctx,
    operands.fd,
    get_fpr(ctx, operands.fs, "S") if get_fpr(ctx, operands.ft, "S") & 1 else 0,
    "S",
)
```

8.510.4. Description

SELNEZ.S validates CP1 and S-format support before updating the destination FPR.

8.510.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit

format is not supported; 32-bit format is not supported

8.511. SEQI

8.511.1. Purpose

Set a GPR to one when another GPR equals an immediate.

8.511.2. Formats

nanoMIPS

`seqi.32`

Assembly

`SEQI {rt:gpr}, {rs:gpr}, {imm:u12}`

Availability

true

Decode priority

0

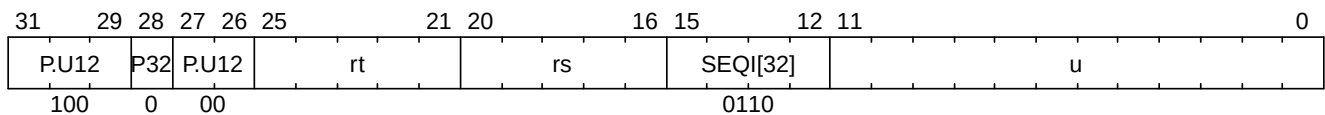


Figure 1413. Encoding (32 bits)

Decode

```
del ctx, exec
return SeqiOperands(rt=rt, rs=rs, imm=zero_extend(u, from_nbits=12))
```

8.511.3. Operation

```
ctx.GPR[operands.rt] = int(ctx.GPR[operands.rs] == int(operands.imm))
```

8.511.4. Description

SEQI[32] compares rs with its zero-extended 12-bit immediate and writes 0 or 1 to rt.

8.511.5. Exceptions

None.

8.512. SH

8.512.1. Purpose

Store the low 16 bits of a GPR to memory.

8.512.2. Formats

MIPS

sh

Assembly

SH {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

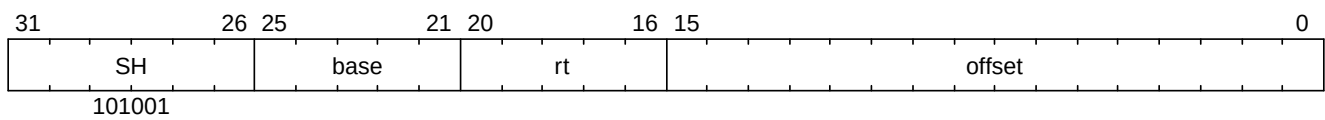


Figure 1414. Encoding (32 bits)

Decode

```
return Sh0perands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

microMIPS

sh.32

Assembly

SH {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

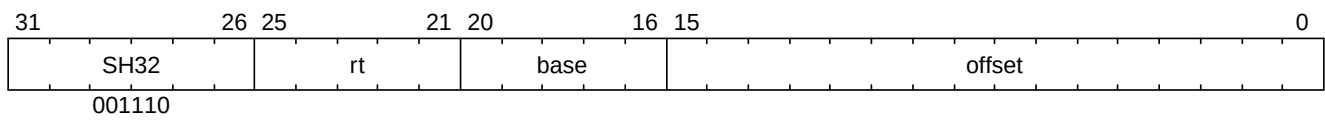


Figure 1415. Encoding (32 bits)

Decode

```
return Sh0perands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

sh.16

Assembly

SH {rt:gpr3}, {offset:u4}({rs:gpr3})

Availability

true

Decode priority

0

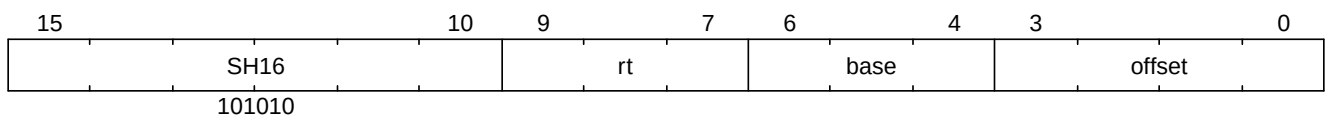


Figure 1416. Encoding (16 bits)

Decode

```
return Sh0perands(  
    rt=MICROMIPS_GPR3_STORE.decode(rt),  
    rs=MICROMIPS_GPR3.decode(base),  
    offset=zero_extend(offset, from_nbits=4) << 1,  
)
```

nanoMIPS

sh.16

Assembly

SH {rt:gpr3}, {offset:u3}({rs:gpr3})

Availability

true

Decode priority

0

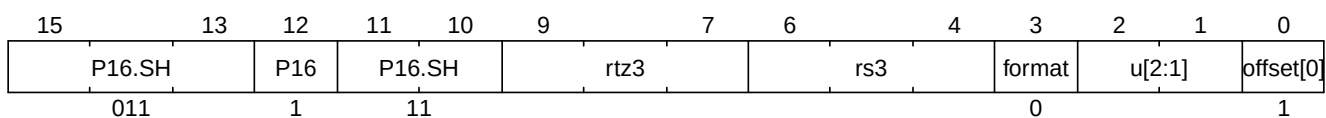


Figure 1417. Encoding (16 bits)

Decode

```
rt = GPR3_STORE.decode(rtz3)  
rs = GPR3.decode(rs3)
```

```
offset = zero_extend(u, from_nbits=3)
return ShOperands(rt=rt, rs=rs, offset=offset)
```

sh.gp

Assembly

SH {rt:gpr}, {offset:u18}(gp)

Availability

true

Decode priority

0



Figure 1418. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=18)
return ShOperands(rt=rt, rs=28, offset=offset)
```

sh.s9

Assembly

SH {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

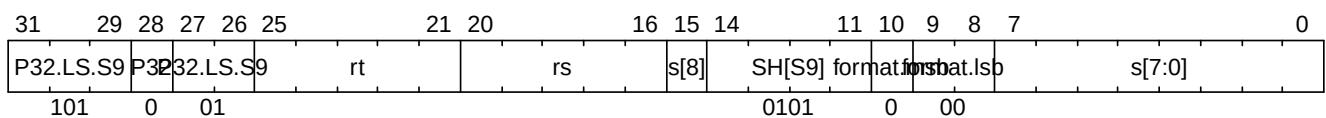


Figure 1419. Encoding (32 bits)

Decode

```
offset = sign_extend(s, from_nbits=9)
return ShOperands(rt=rt, rs=rs, offset=offset)
```

sh.u12

Assembly

SH {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

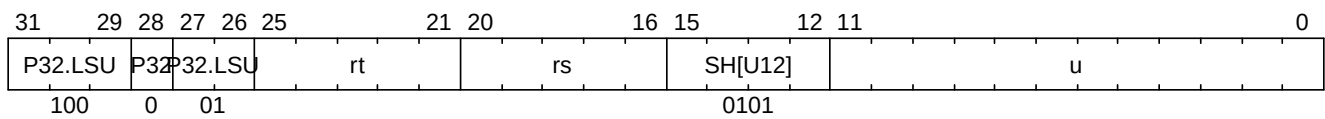


Figure 1420. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return ShOperands(rt=rt, rs=rs, offset=offset)
```

8.512.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
data = zero_extend(ctx.GPR[operands.rt], from_nbits=16)
write_memory_at_va(ctx, data, va, nbytes=2)
```

8.512.4. Description

SH adds the encoded byte offset to the base GPR, takes the low 16 bits of the source GPR, and writes two bytes through the active memory backend.

8.512.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.513. SHE

8.513.1. Purpose

Store a halfword using EVA addressing.

8.513.2. Formats

nanoMIPS

she.32

Assembly

SHE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

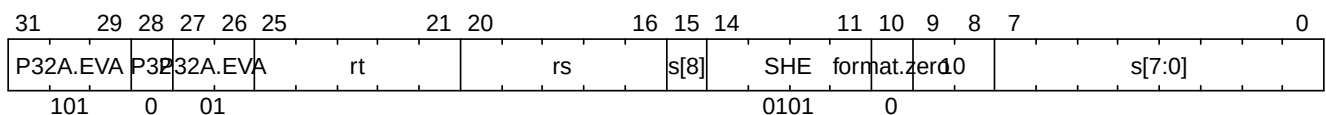


Figure 1421. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.513.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store", eva=True)
data = zero_extend(ctx.GPR[operands.rt], from_nbits=16)
write_memory_at_va(ctx, data, va, nbytes=2, eva=True)
```

8.513.4. Description

This optional privileged form uses EVA user-mode address translation.

8.513.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.514. SHILO

8.514.1. Purpose

DSP bitstream operation SHILO.

8.514.2. Formats

MIPS

shilo

Assembly

SHILO {ac:u2}, {shift:u6}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

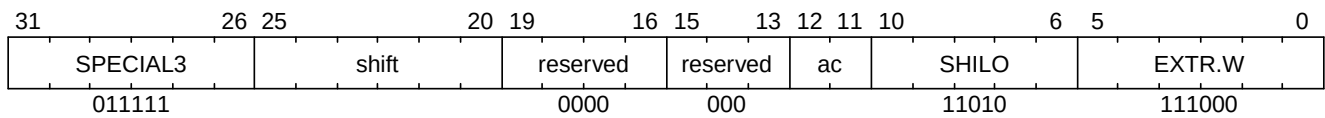


Figure 1422. Encoding (32 bits)

Decode

```
return DspShiftOperands(ac=ac, shift=shift)
```

microMIPS

shilo.32

Assembly

SHILO {ac:u2}, {shift:u6}

Availability

DSP

Decode priority

0

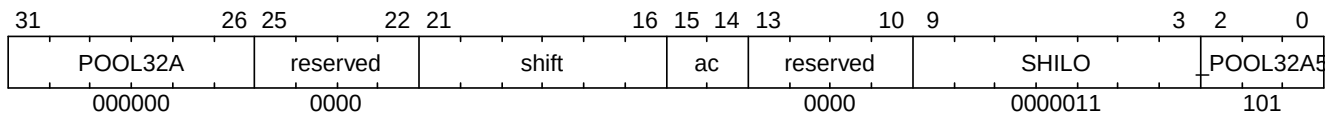


Figure 1423. Encoding (32 bits)

Decode

```
return DspShiftOperands(ac=ac, shift=shift)
```

nanoMIPS

shilo.32

Assembly

SHILO {ac:u2}, {shift:u6}

Availability

DSP

Decode priority

0

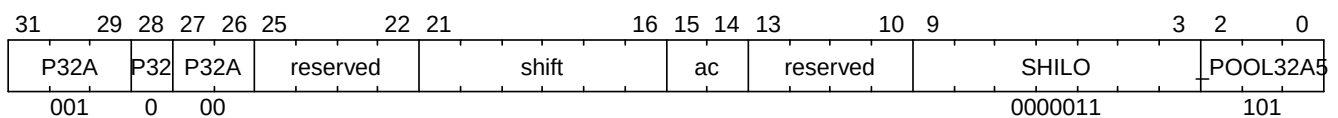


Figure 1424. Encoding (32 bits)

Decode

```
return DspShiftOperands(ac=ac, shift=shift)
```

8.514.3. Operation

```
validate_access_to_dsp_resources(ctx)
amount = operands.shift & 0x3F
signed_amount = amount - 64 if amount & 0x20 else amount
bits = acc(ctx, operands.ac)
shifted = bits >> signed_amount if signed_amount >= 0 else bits << -signed_amount
put_acc(ctx, operands.ac, shifted)
```

8.514.4. Description

SHILO is a DSP bitstream or accumulator extraction operation.

8.514.5. Exceptions

None.

8.515. SHILOV

8.515.1. Purpose

DSP bitstream operation SHILOV.

8.515.2. Formats

MIPS

shilov

Assembly

SHILOV {ac:u2}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

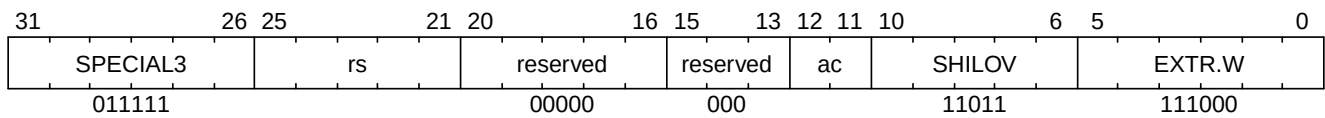


Figure 1425. Encoding (32 bits)

Decode

```
return DspShiftVariable0perands(ac=ac, rs=rs)
```

microMIPS

shilov.32

Assembly

SHILOV {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

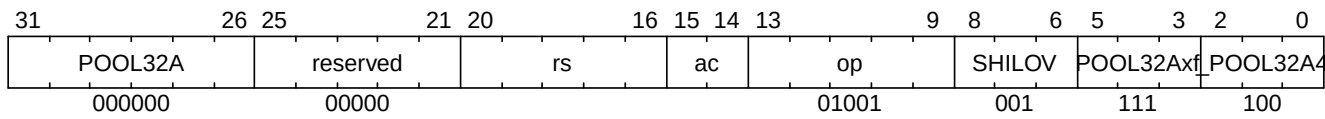


Figure 1426. Encoding (32 bits)

Decode

```
return DspShiftVariableOperands(ac=ac, rs=rs)
```

nanoMIPS

shilov.32

Assembly

SHILOV {ac:u2}, {rs:gpr}

Availability

DSP

Decode priority

0

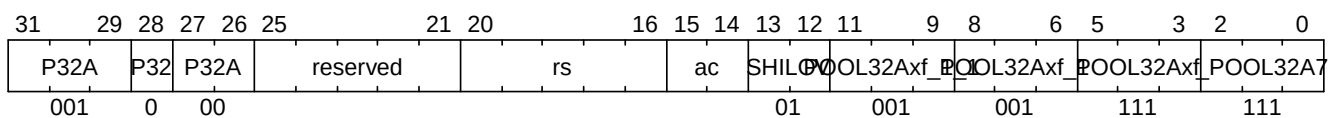


Figure 1427. Encoding (32 bits)

Decode

```
return DspShiftVariableOperands(ac=ac, rs=rs)
```

8.515.3. Operation

```
validate_access_to_dsp_resources(ctx)
amount = ctx.GPR[operands.rs] & 0x3F
signed_amount = amount - 64 if amount & 0x20 else amount
bits = acc(ctx, operands.ac)
shifted = bits >> signed_amount if signed_amount >= 0 else bits << -signed_amount
put_acc(ctx, operands.ac, shifted)
```

8.515.4. Description

SHILOV is a DSP bitstream or accumulator extraction operation.

8.515.5. Exceptions

None.

8.516. SHLL.PH

8.516.1. Purpose

DSP shift SHLL.PH.

8.516.2. Formats

MIPS

shll.ph

Assembly

SHLL.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

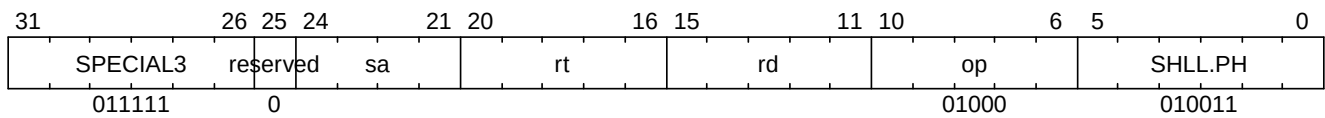


Figure 1428. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shll.ph

Assembly

SHLL.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP

Decode priority

0

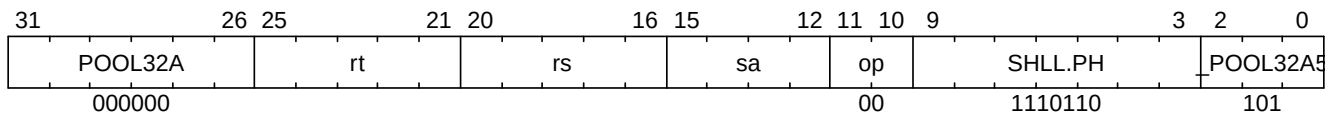


Figure 1429. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shll.ph

Assembly

SHLL.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP

Decode priority

0

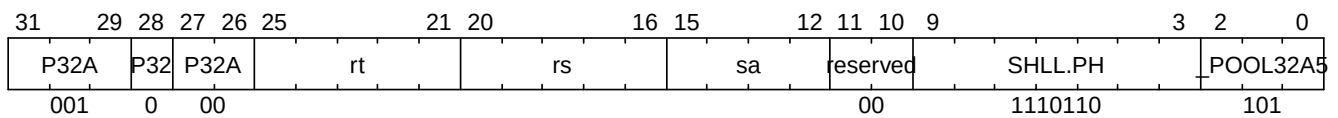


Figure 1430. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.516.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(ctx, operands.rd, ctx.GPR[operands.rt], 16, operands.sa)
```

8.516.4. Description

Execute the SHLL.PH packed shift.

8.516.5. Exceptions

None.

8.517. SHLL.QB

8.517.1. Purpose

DSP shift SHLL.QB.

8.517.2. Formats

MIPS

shll.qb

Assembly

SHLL.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

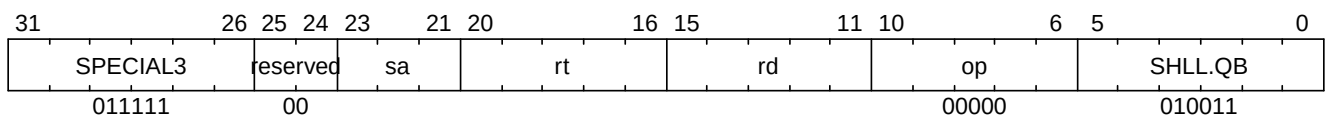


Figure 1431. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shll.qb

Assembly

SHLL.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP

Decode priority

0

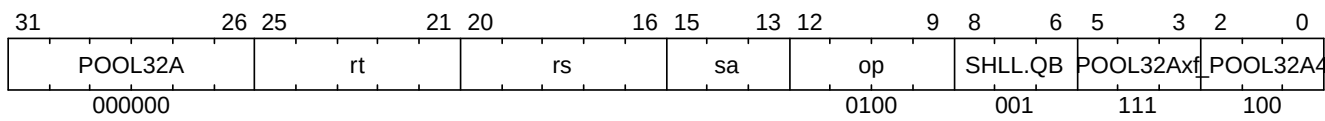


Figure 1432. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shll.qb

Assembly

SHLL.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP

Decode priority

0

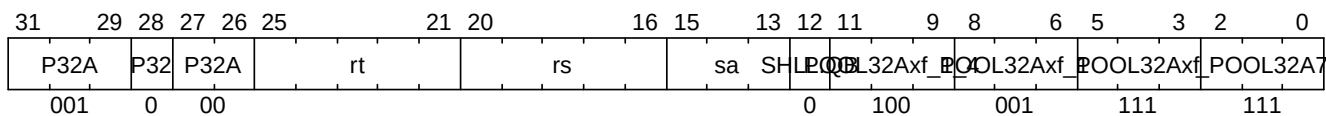


Figure 1433. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.517.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(ctx, operands.rd, ctx.GPR[operands.rt], 8, operands.sa)
```

8.517.4. Description

Execute the SHLL.QB packed shift.

8.517.5. Exceptions

None.

8.518. SHLL_S.PH

8.518.1. Purpose

DSP shift SHLL_S.PH.

8.518.2. Formats

MIPS

shll_s.ph

Assembly

SHLL_S.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

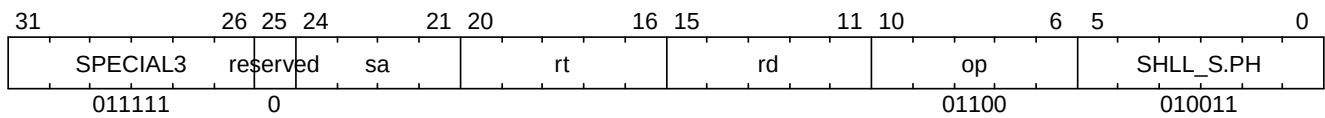


Figure 1434. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shll_s.ph

Assembly

SHLL_S.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP

Decode priority

0

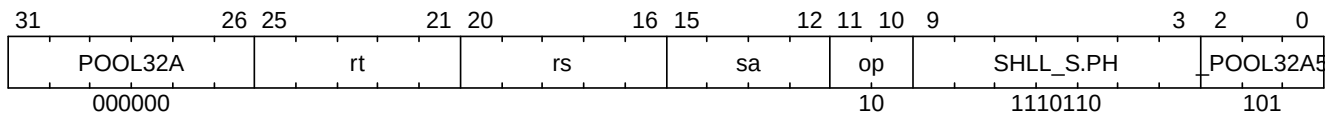


Figure 1435. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shll_s.ph

Assembly

SHLL_S.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP

Decode priority

0

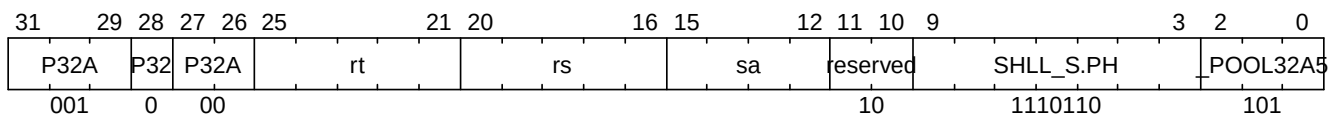


Figure 1436. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.518.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(ctx, operands.rd, ctx.GPR[operands.rt], 16, operands.sa, saturate=True)
```

8.518.4. Description

Execute the SHLL_S.PH packed shift.

8.518.5. Exceptions

None.

8.519. SHLL_S.W

8.519.1. Purpose

DSP shift SHLL_S.W.

8.519.2. Formats

MIPS

shll_s.w

Assembly

SHLL_S.W {rd:gpr}, {rt:gpr}, {sa:u5}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

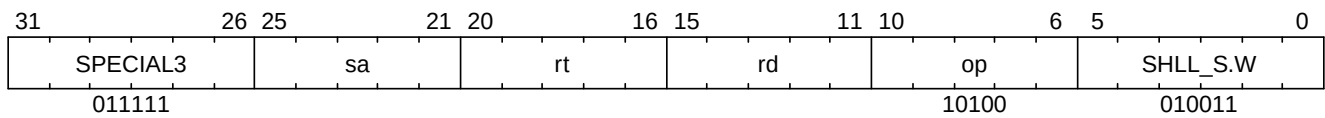


Figure 1437. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shll_s.w

Assembly

SHLL_S.W {rd:gpr}, {rt:gpr}, {sa:u5}

Availability

DSP

Decode priority

0

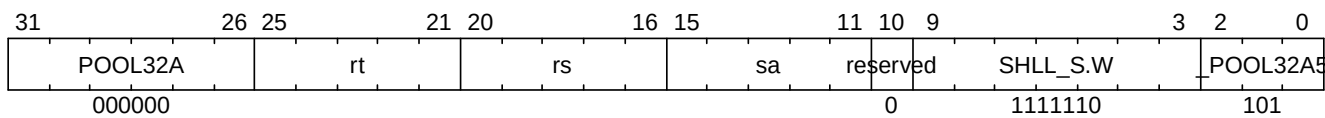


Figure 1438. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shll_s.w

Assembly

SHLL_S.W {rd:gpr}, {rt:gpr}, {sa:u5}

Availability

DSP

Decode priority

0

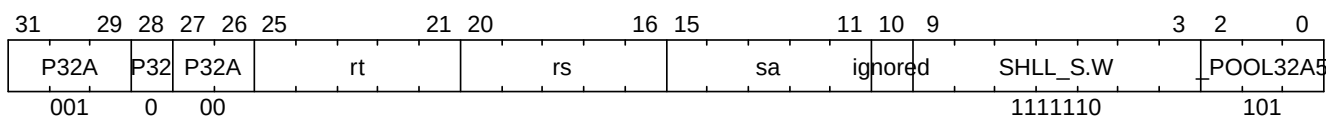


Figure 1439. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.519.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(ctx, operands.rd, ctx.GPR[operands.rt], 32, operands.sa, saturate=True)
```

8.519.4. Description

Execute the SHLL_S.W packed shift.

8.519.5. Exceptions

None.

8.520. SHLLV.PH

8.520.1. Purpose

DSP shift SHLLV.PH.

8.520.2. Formats

MIPS

shllv.ph

Assembly

SHLLV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

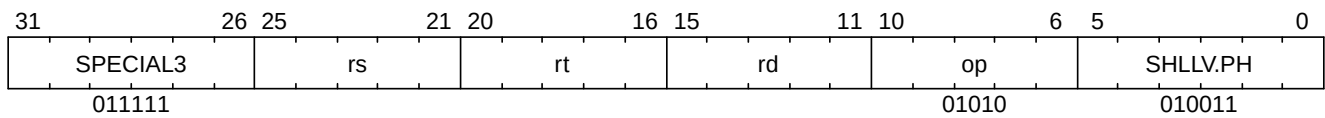


Figure 1440. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shllv.ph

Assembly

SHLLV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

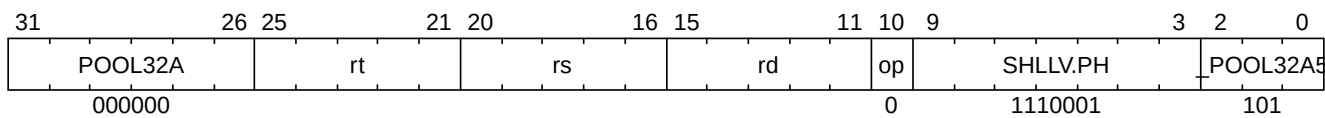


Figure 1441. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shllv.ph

Assembly

SHLLV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

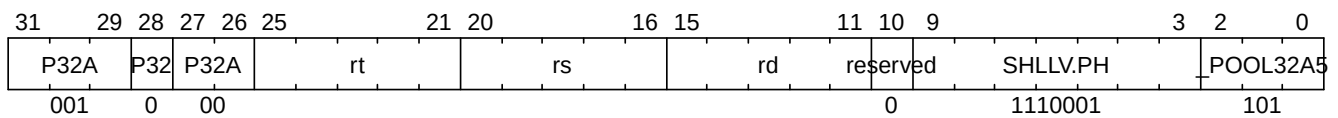


Figure 1442. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.520.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(ctx, operands.rd, ctx.GPR[operands.rt], 16, ctx.GPR[operands.rs] & 0xF)
```

8.520.4. Description

Execute the SHLLV.PH packed shift.

8.520.5. Exceptions

None.

8.521. SHLLV.QB

8.521.1. Purpose

DSP shift SHLLV.QB.

8.521.2. Formats

MIPS

shllv.qb

Assembly

SHLLV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

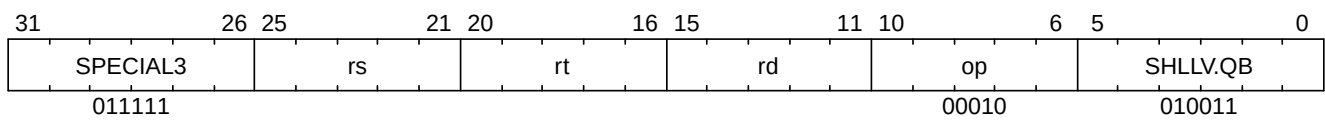


Figure 1443. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shllv.qb

Assembly

SHLLV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

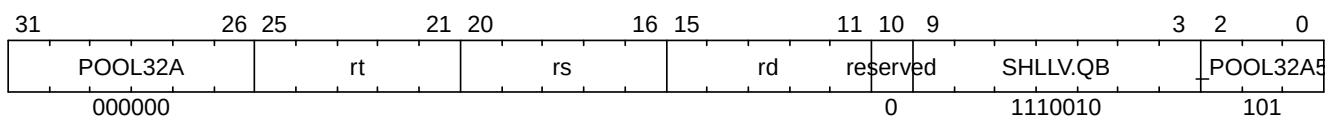


Figure 1444. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shllv.qb

Assembly

SHLLV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

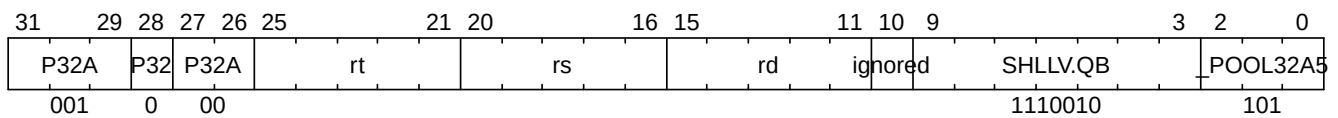


Figure 1445. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.521.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(ctx, operands.rd, ctx.GPR[operands.rt], 8, ctx.GPR[operands.rs] & 0x7)
```

8.521.4. Description

Execute the SHLLV.QB packed shift.

8.521.5. Exceptions

None.

8.522. SHLLV_S.PH

8.522.1. Purpose

DSP shift SHLLV_S.PH.

8.522.2. Formats

MIPS

shllv_s.ph

Assembly

SHLLV_S.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

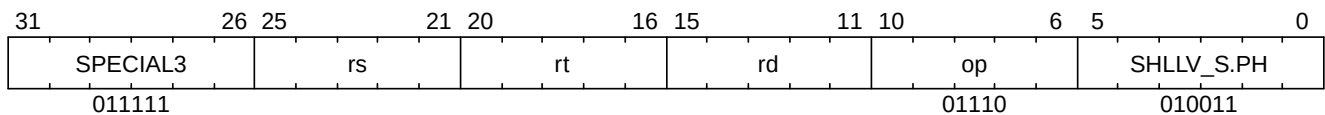


Figure 1446. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shllv_s.ph

Assembly

SHLLV_S.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

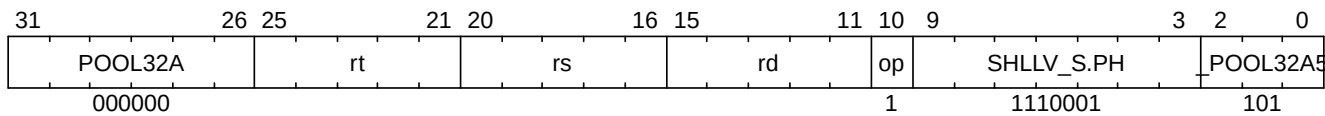


Figure 1447. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shllv_s.ph

Assembly

SHLLV_S.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

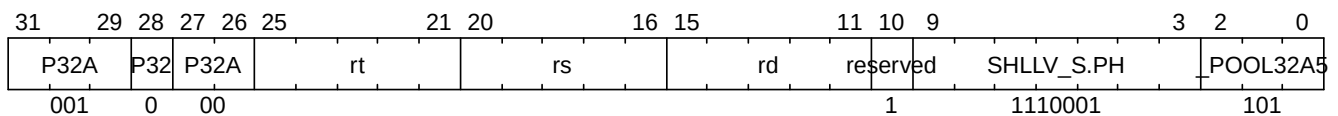


Figure 1448. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.522.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(
    ctx,
    operands.rd,
    ctx.GPR[operands.rt],
    16,
    ctx.GPR[operands.rs] & 0xF,
    saturate=True,
)
```

8.522.4. Description

Execute the SHLLV_S.PH packed shift.

8.522.5. Exceptions

None.

8.523. SHLLV_S.W

8.523.1. Purpose

DSP shift SHLLV_S.W.

8.523.2. Formats

MIPS

shllv_s.w

Assembly

SHLLV_S.W {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

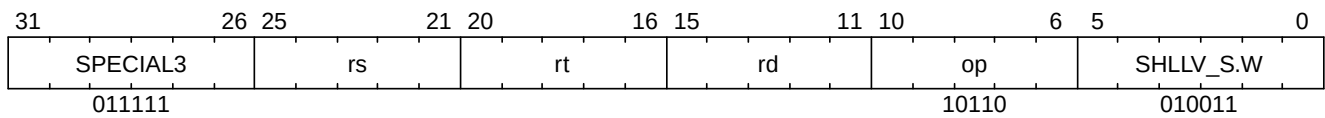


Figure 1449. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shllv_s.w

Assembly

SHLLV_S.W {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

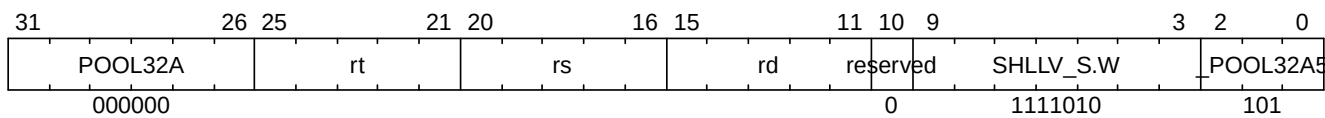


Figure 1450. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shllv_s.w

Assembly

SHLLV_S.W {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

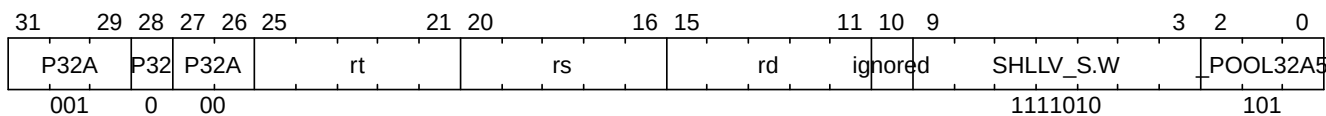


Figure 1451. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.523.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_left(
    ctx,
    operands.rd,
    ctx.GPR[operands.rt],
    32,
    ctx.GPR[operands.rs] & 0x1F,
    saturate=True,
)
```

8.523.4. Description

Execute the SHLLV_S.W packed shift.

8.523.5. Exceptions

None.

8.524. SHRA.PH

8.524.1. Purpose

DSP shift SHRA.PH.

8.524.2. Formats

MIPS

shra.ph

Assembly

SHRA.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

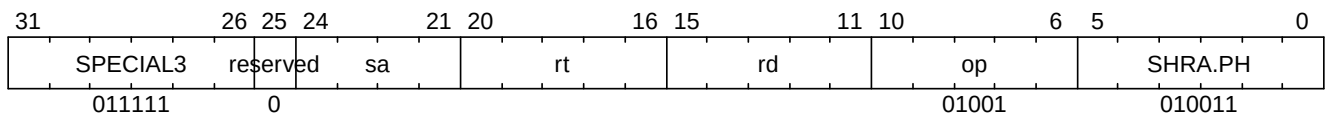


Figure 1452. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shra.ph

Assembly

SHRA.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP2

Decode priority

0

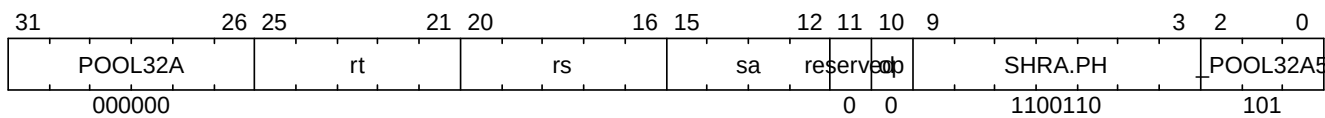


Figure 1453. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shra.ph

Assembly

SHRA.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP2

Decode priority

0

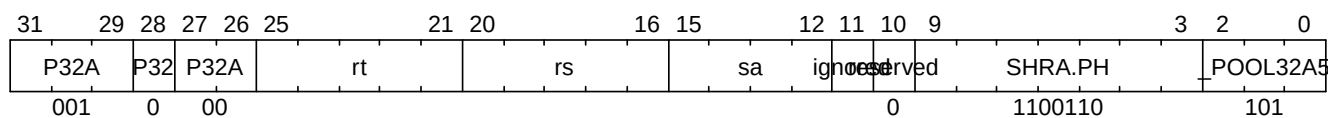


Figure 1454. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.524.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 16, operands.sa)
```

8.524.4. Description

Execute the SHRA.PH packed shift.

8.524.5. Exceptions

None.

8.525. SHRA.QB

8.525.1. Purpose

DSP shift SHRA.QB.

8.525.2. Formats

MIPS

shra.qb

Assembly

SHRA.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

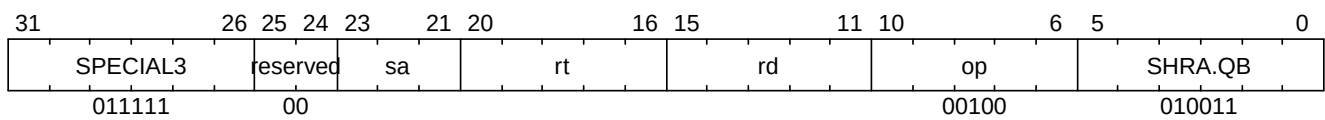


Figure 1455. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shra.qb

Assembly

SHRA.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP2

Decode priority

0

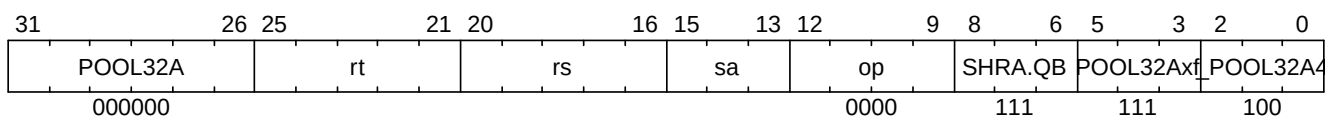


Figure 1456. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shra.qb

Assembly

SHRA.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP2

Decode priority

0

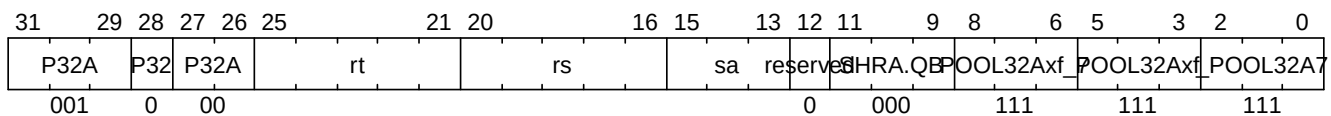


Figure 1457. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.525.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 8, operands.sa)
```

8.525.4. Description

Execute the SHRA.QB packed shift.

8.525.5. Exceptions

None.

8.526. SHRA_R.PH

8.526.1. Purpose

DSP shift SHRA_R.PH.

8.526.2. Formats

MIPS

shra_r.ph

Assembly

SHRA_R.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

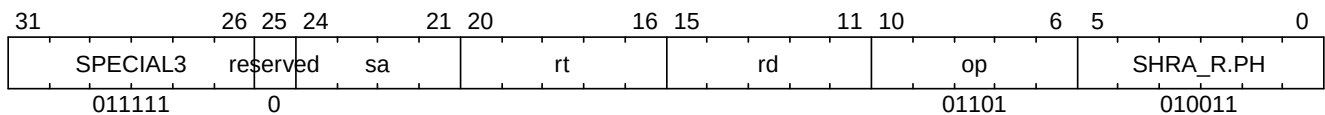


Figure 1458. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shra_r.ph

Assembly

SHRA_R.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP2

Decode priority

0

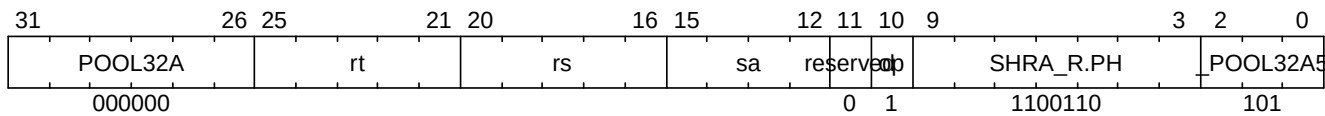


Figure 1459. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shra_r.ph

Assembly

SHRA_R.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP2

Decode priority

0

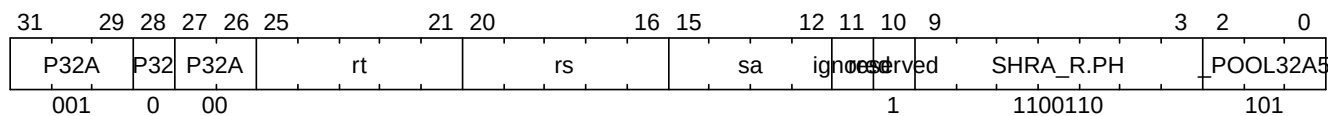


Figure 1460. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.526.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 16, operands.sa, rounded=True)
```

8.526.4. Description

Execute the SHRA_R.PH packed shift.

8.526.5. Exceptions

None.

8.527. SHRA_R.QB

8.527.1. Purpose

DSP shift SHRA_R.QB.

8.527.2. Formats

MIPS

shra_r.qb

Assembly

SHRA_R.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

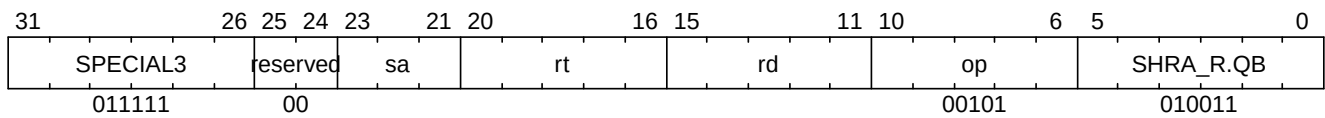


Figure 1461. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shra_r.qb

Assembly

SHRA_R.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP2

Decode priority

0

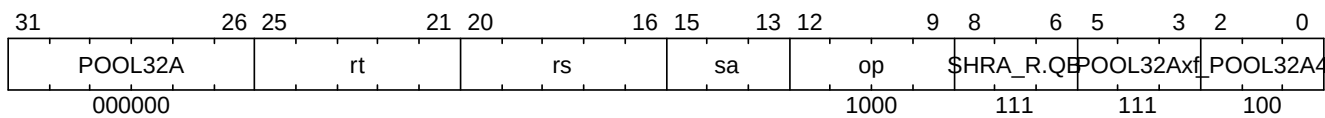


Figure 1462. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shra_r.qb

Assembly

SHRA_R.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP2

Decode priority

0

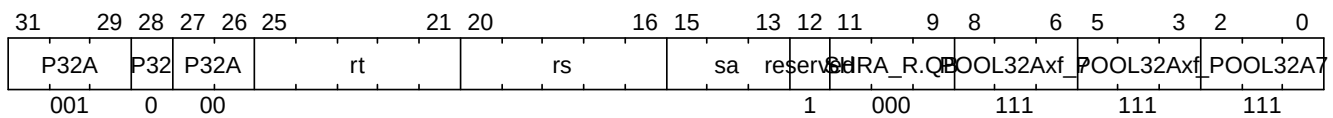


Figure 1463. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.527.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 8, operands.sa, rounded=True)
```

8.527.4. Description

Execute the SHRA_R.QB packed shift.

8.527.5. Exceptions

None.

8.528. SHRA_R.W

8.528.1. Purpose

DSP shift SHRA_R.W.

8.528.2. Formats

MIPS

shra_r.w

Assembly

SHRA_R.W {rd:gpr}, {rt:gpr}, {sa:u5}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

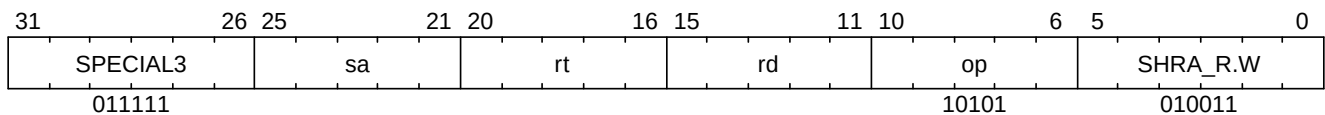


Figure 1464. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shra_r.w

Assembly

SHRA_R.W {rd:gpr}, {rt:gpr}, {sa:u5}

Availability

DSP

Decode priority

0

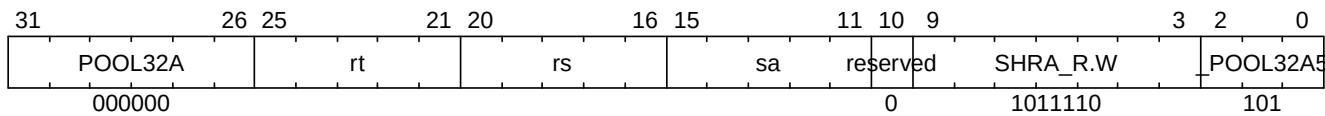


Figure 1465. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shra_r.w

Assembly

SHRA_R.W {rd:gpr}, {rt:gpr}, {sa:u5}

Availability

DSP

Decode priority

0

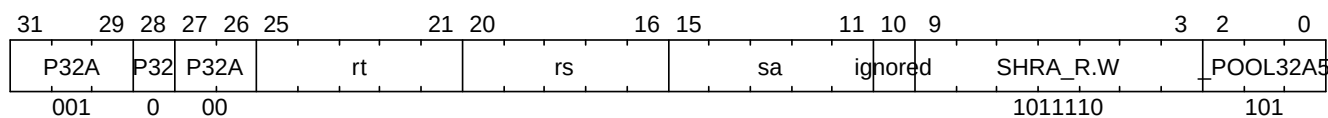


Figure 1466. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.528.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 32, operands.sa, rounded=True)
```

8.528.4. Description

Execute the SHRA_R.W packed shift.

8.528.5. Exceptions

None.

8.529. SHRAV.PH

8.529.1. Purpose

DSP shift SHRAV.PH.

8.529.2. Formats

MIPS

shrav.ph

Assembly

SHRAV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

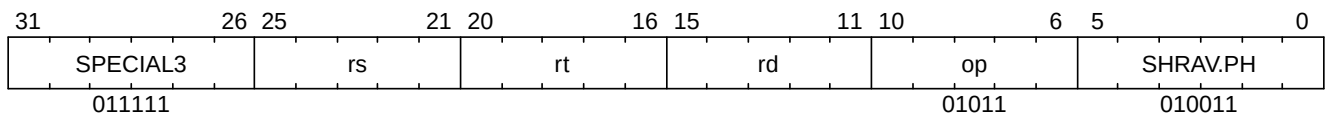


Figure 1467. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shrav.ph

Assembly

SHRAV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

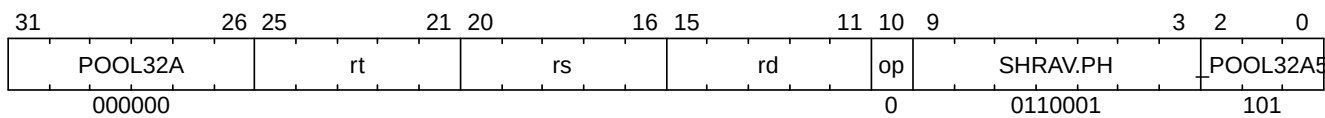


Figure 1468. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shrav.ph

Assembly

SHRAV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

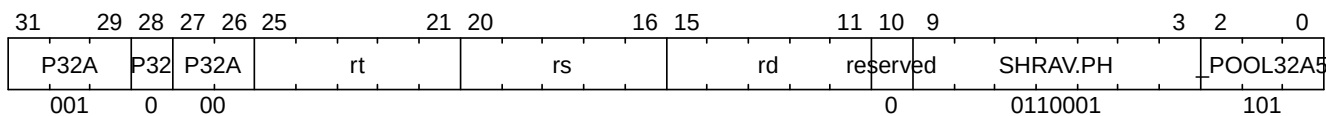


Figure 1469. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.529.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 16, ctx.GPR[operands.rs] & 0xF)
```

8.529.4. Description

Execute the SHRAV.PH packed shift.

8.529.5. Exceptions

None.

8.530. SHRAV.QB

8.530.1. Purpose

DSP shift SHRAV.QB.

8.530.2. Formats

MIPS

shrav.qb

Assembly

SHRAV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

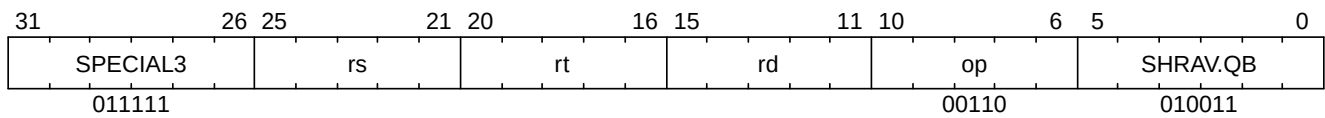


Figure 1470. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shrav.qb

Assembly

SHRAV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

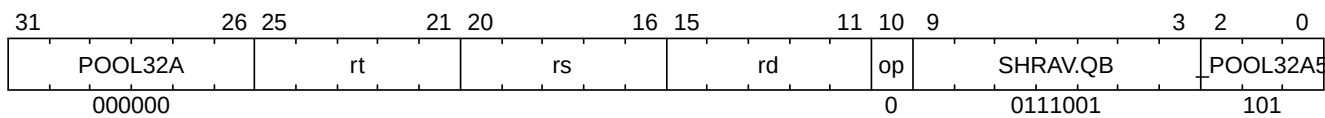


Figure 1471. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shrav.qb

Assembly

SHRAV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

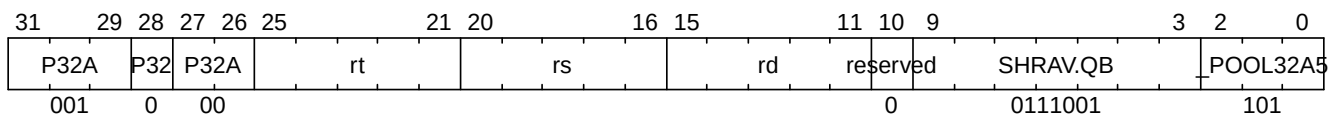


Figure 1472. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.530.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 8, ctx.GPR[operands.rs] & 0x7)
```

8.530.4. Description

Execute the SHRAV.QB packed shift.

8.530.5. Exceptions

None.

8.531. SHRAV_R.PH

8.531.1. Purpose

DSP shift SHRAV_R.PH.

8.531.2. Formats

MIPS

shrav_r.ph

Assembly

SHRAV_R.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

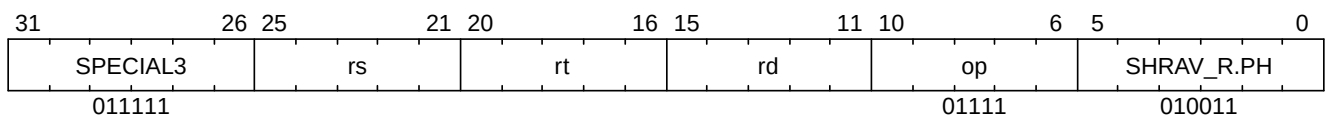


Figure 1473. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shrav_r.ph

Assembly

SHRAV_R.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

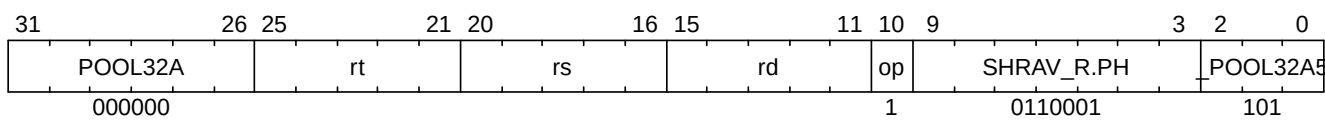


Figure 1474. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shrav_r.ph

Assembly

SHRAV_R.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

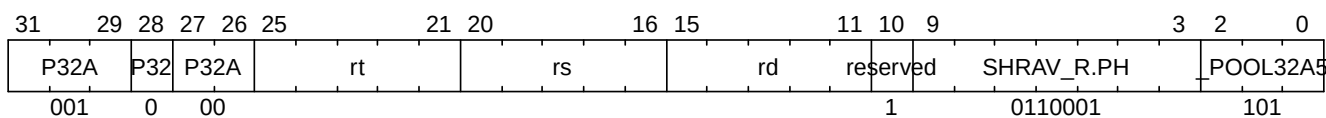


Figure 1475. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.531.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(
    ctx,
    operands.rd,
    ctx.GPR[operands.rt],
    16,
    ctx.GPR[operands.rs] & 0xF,
    rounded=True,
)
```

8.531.4. Description

Execute the SHRAV_R.PH packed shift.

8.531.5. Exceptions

None.

8.532. SHRAV_R.QB

8.532.1. Purpose

DSP shift SHRAV_R.QB.

8.532.2. Formats

MIPS

shrav_r.qb

Assembly

SHRAV_R.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

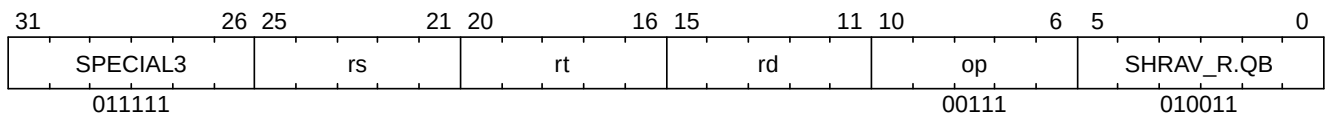


Figure 1476. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shrav_r.qb

Assembly

SHRAV_R.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

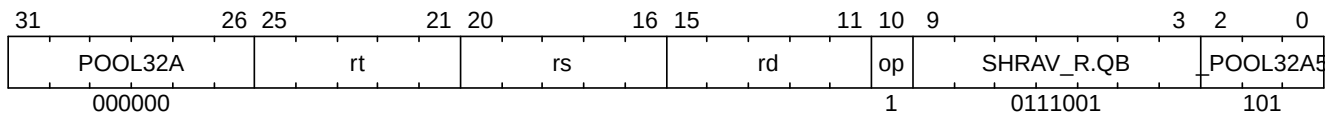


Figure 1477. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shrav_r.qb

Assembly

SHRAV_R.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

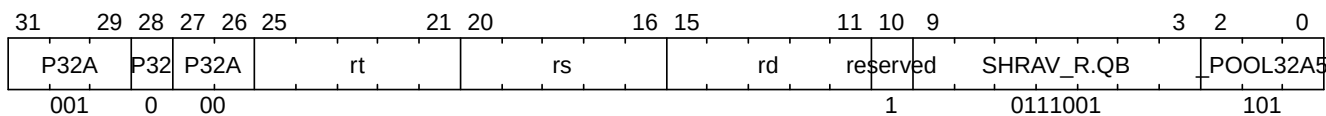


Figure 1478. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.532.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_right(
    ctx,
    operands.rd,
    ctx.GPR[operands.rt],
    8,
    ctx.GPR[operands.rs] & 0x7,
    rounded=True,
)
```

8.532.4. Description

Execute the SHRAV_R.QB packed shift.

8.532.5. Exceptions

None.

8.533. SHRAV_R.W

8.533.1. Purpose

DSP shift SHRAV_R.W.

8.533.2. Formats

MIPS

shrav_r.w

Assembly

SHRAV_R.W {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

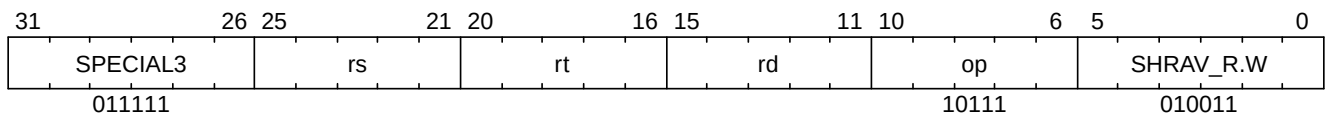


Figure 1479. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shrav_r.w

Assembly

SHRAV_R.W {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

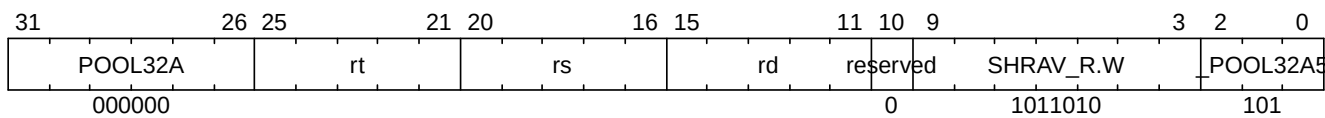


Figure 1480. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shrav_r.w

Assembly

SHRAV_R.W {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

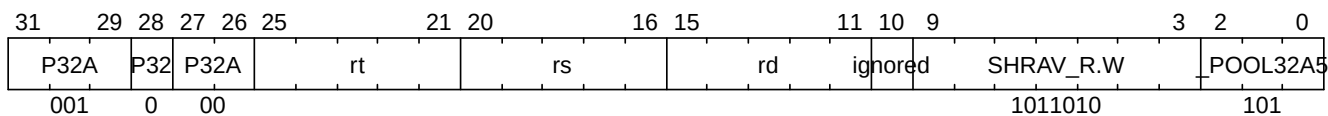


Figure 1481. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.533.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(
    ctx,
    operands.rd,
    ctx.GPR[operands.rt],
    32,
    ctx.GPR[operands.rs] & 0x1F,
    rounded=True,
)
```

8.533.4. Description

Execute the SHRAV_R.W packed shift.

8.533.5. Exceptions

None.

8.534. SHRL.PH

8.534.1. Purpose

DSP shift SHRL.PH.

8.534.2. Formats

MIPS

shrl.ph

Assembly

SHRL.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

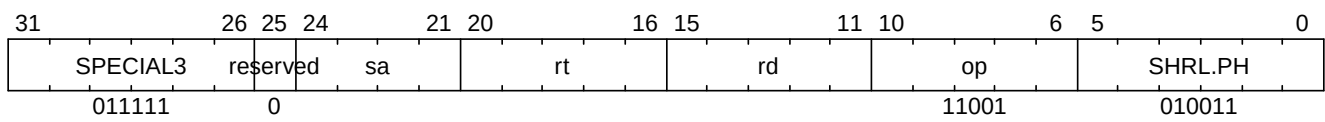


Figure 1482. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shrl.ph

Assembly

SHRL.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP2

Decode priority

0

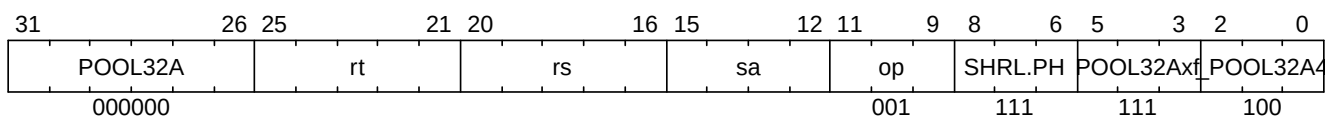


Figure 1483. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shrl.ph

Assembly

SHRL.PH {rd:gpr}, {rt:gpr}, {sa:u4}

Availability

DSP2

Decode priority

0

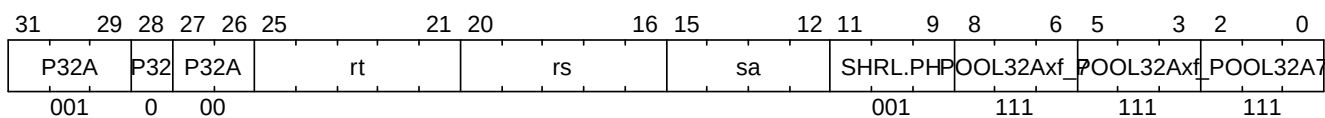


Figure 1484. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.534.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 16, operands.sa, logical=True)
```

8.534.4. Description

Execute the SHRL.PH packed shift.

8.534.5. Exceptions

None.

8.535. SHRL.QB

8.535.1. Purpose

DSP shift SHRL.QB.

8.535.2. Formats

MIPS

shrl.qb

Assembly

SHRL.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

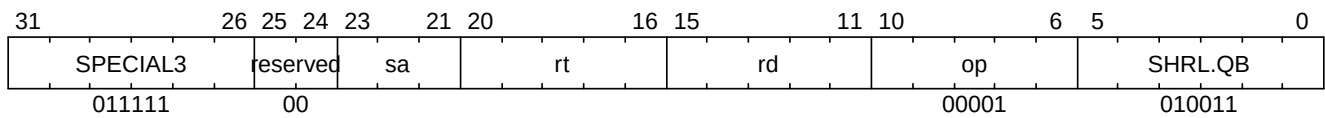


Figure 1485. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rd, rt=rt, sa=sa)
```

microMIPS

shrl.qb

Assembly

SHRL.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP

Decode priority

0

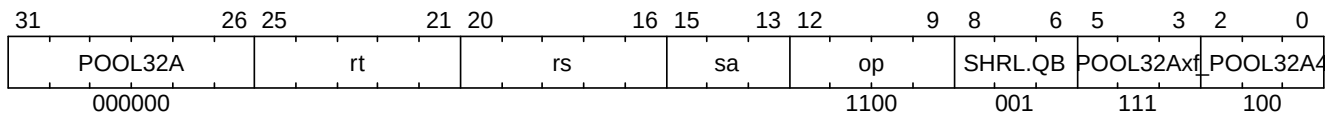


Figure 1486. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

nanoMIPS

shrl.qb

Assembly

SHRL.QB {rd:gpr}, {rt:gpr}, {sa:u3}

Availability

DSP

Decode priority

0

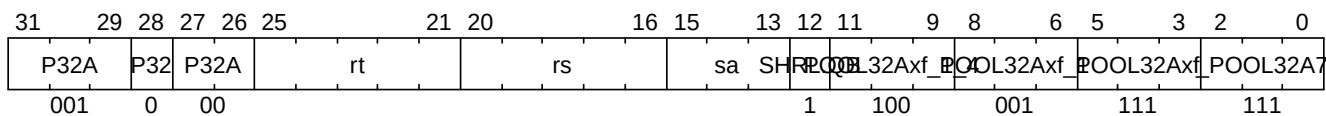


Figure 1487. Encoding (32 bits)

Decode

```
return FixedShiftOperands(rd=rt, rt=rs, sa=sa)
```

8.535.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(ctx, operands.rd, ctx.GPR[operands.rt], 8, operands.sa, logical=True)
```

8.535.4. Description

Execute the SHRL.QB packed shift.

8.535.5. Exceptions

None.

8.536. SHRLV.PH

8.536.1. Purpose

DSP shift SHRLV.PH.

8.536.2. Formats

MIPS

shrlv.ph

Assembly

SHRLV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

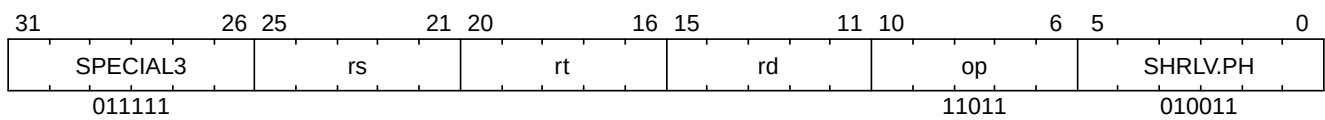


Figure 1488. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shrlv.ph

Assembly

SHRLV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

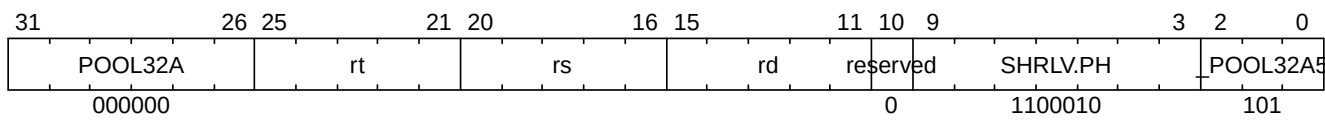


Figure 1489. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shrlv.ph

Assembly

SHRLV.PH {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP2

Decode priority

0

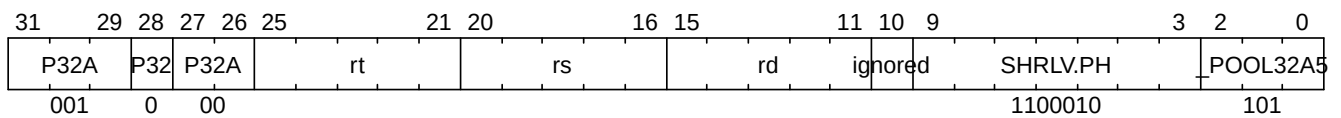


Figure 1490. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.536.3. Operation

```
validate_access_to_dsp2_resources(ctx)
write_right(
    ctx,
    operands.rd,
    ctx.GPR[operands.rt],
    16,
    ctx.GPR[operands.rs] & 0xF,
    logical=True,
)
```

8.536.4. Description

Execute the SHRLV.PH packed shift.

8.536.5. Exceptions

None.

8.537. SHRLV.QB

8.537.1. Purpose

DSP shift SHRLV.QB.

8.537.2. Formats

MIPS

shrlv.qb

Assembly

SHRLV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

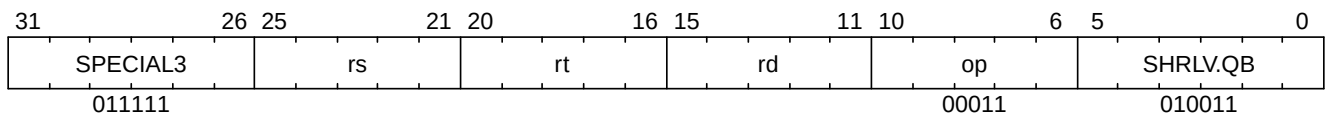


Figure 1491. Encoding (32 bits)

Decode

```
return VariableShift0perands(rd=rd, rt=rt, rs=rs)
```

microMIPS

shrlv.qb

Assembly

SHRLV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

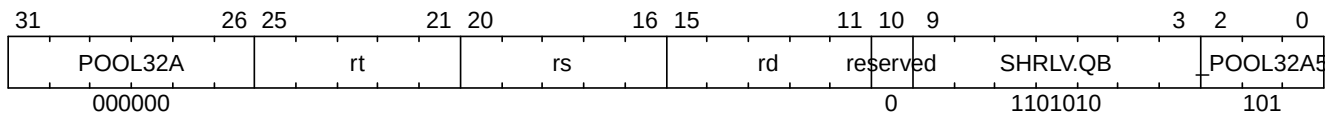


Figure 1492. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

nanoMIPS

shrlv.qb

Assembly

SHRLV.QB {rd:gpr}, {rt:gpr}, {rs:gpr}

Availability

DSP

Decode priority

0

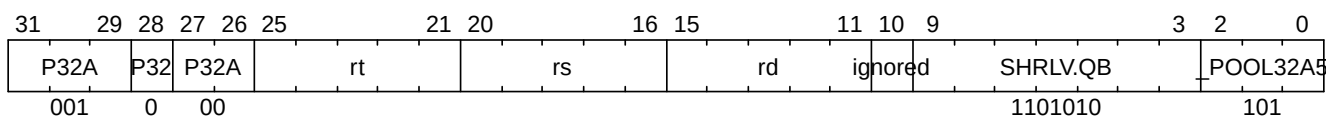


Figure 1493. Encoding (32 bits)

Decode

```
return VariableShiftOperands(rd=rd, rt=rt, rs=rs)
```

8.537.3. Operation

```
validate_access_to_dsp_resources(ctx)
write_right(
    ctx,
    operands.rd,
    ctx.GPR[operands.rt],
    8,
    ctx.GPR[operands.rs] & 0x7,
    logical=True,
)
```

8.537.4. Description

Execute the SHRLV.QB packed shift.

8.537.5. Exceptions

None.

8.538. SHX

8.538.1. Purpose

Store a halfword using a register-indexed address.

8.538.2. Formats

nanoMIPS

shx.32

Assembly

SHX {rs:gpr}({rt:gpr})

Availability

!(NMS)

Decode priority

0

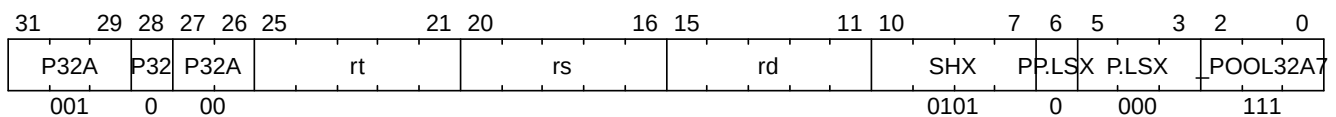


Figure 1494. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.538.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Store")
data = zero_extend(ctx.GPR[operands.rd], from_nbits=16)
write_memory_at_va(ctx, data, va, nbytes=2)
```

8.538.4. Description

The effective address is formed from two full-width GPR values.

8.538.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.539. SHXS

8.539.1. Purpose

Store a halfword using a scaled register-indexed address.

8.539.2. Formats

nanoMIPS

shxs.32

Assembly

SHXS {rs:gpr}({rt:gpr})

Availability

!(NMS)

Decode priority

0

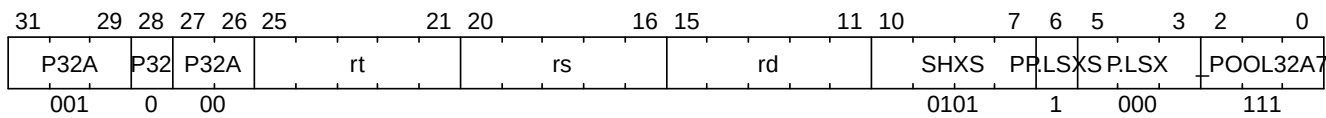


Figure 1495. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.539.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)) << 1
va = effective_address(ctx, base, index, "Store")
data = zero_extend(ctx.GPR[operands.rd], from_nbits=16)
write_memory_at_va(ctx, data, va, nbytes=2)
```

8.539.4. Description

The effective address is formed from two full-width GPR values.

8.539.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.540. SIGRIE

8.540.1. Purpose

Signal a reserved-instruction exception.

8.540.2. Formats

MIPS

sigrie

Assembly

SIGRIE {code:u16}

Availability

AR >= R6

Decode priority

0

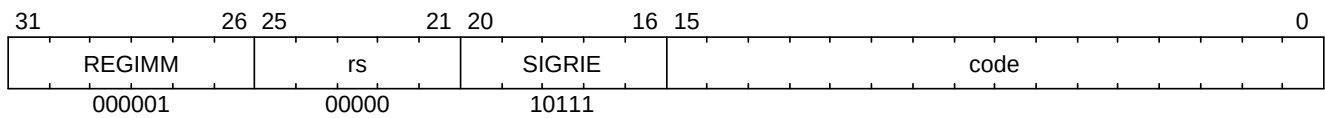


Figure 1496. Encoding (32 bits)

Decode

```
return SigrieOperands(code=zero_extend(code, from_nbits=16))
```

nanoMIPS

sigrie.32

Assembly

SIGRIE {code:u19}

Availability

true

Decode priority

0

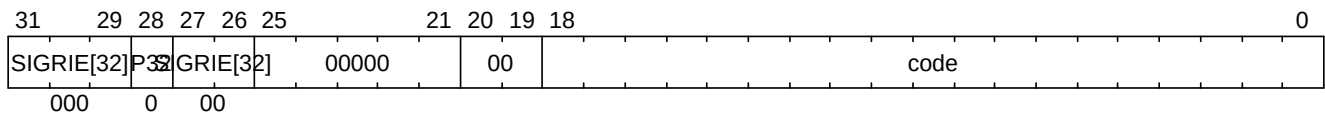


Figure 1497. Encoding (32 bits)

Decode

```
return SigrieOperands(code=zero_extend(code, from_nbits=19))
```

8.540.3. Operation

```
raise ReservedInstruction("SIGRIE")
```

8.540.4. Description

SIGRIE unconditionally raises RI.

8.540.5. Exceptions

- **RI** (Reserved Instruction) — SIGRIE

8.541. SLL

8.541.1. Purpose

Shift the 32-bit word in `rs` left by `shift` bits and place the result in `rt`.

8.541.2. Formats

MIPS

`sll`

Assembly

`SLL {rt:gpr}, {rs:gpr}, {shift:u5}`

Availability

true

Decode priority

0

Overridden by higher-priority formats

- NOP (`nop`)
- PAUSE (`pause`)

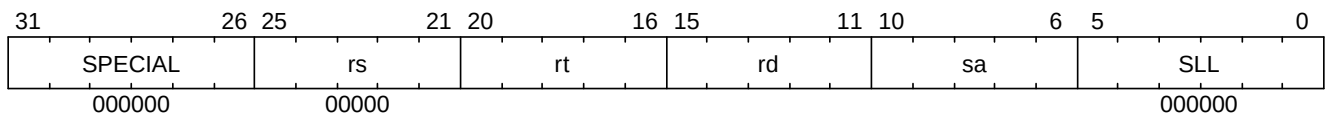


Figure 1498. Encoding (32 bits)

Decode

```
del ctx, exec
return Sll0operands(rt=rd, rs=rt, shift=zero_extend(sa, from_nbits=5))
```

microMIPS

`sll.16`

Assembly

`SLL {rt:gpr3}, {rs:gpr3}, {shift:u3}`

Availability

true

Decode priority

0

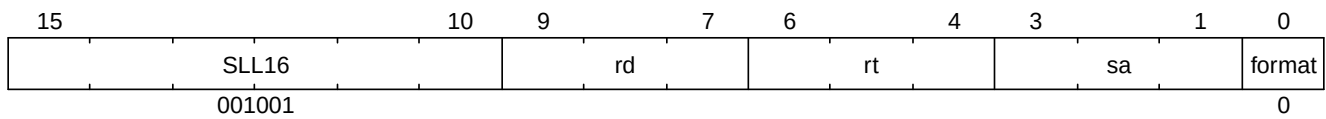


Figure 1499. Encoding (16 bits)

Decode

```
return SllOperands(
    rt=MICROMIPS_GPR3.decode(rd),
    rs=MICROMIPS_GPR3.decode(rt),
    shift=8 if sa == 0 else sa,
)
```

sll.32

Assembly

SLL {rt:gpr}, {rs:gpr}, {shift:u5}

Availability

true

Decode priority

0

Overridden by higher-priority formats

- NOP (nop.32)

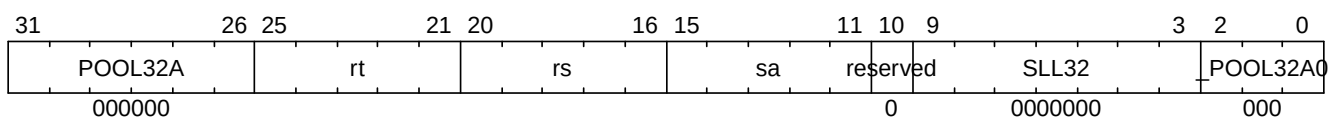


Figure 1500. Encoding (32 bits)

Decode

```
return SllOperands(rt=rt, rs=rs, shift=zero_extend(sa, from_nbits=5))
```

nanoMIPS

sll.16

Assembly

SLL {rt:gpr3}, {rs:gpr3}, {shift:u4}

Availability

true

Decode priority

0

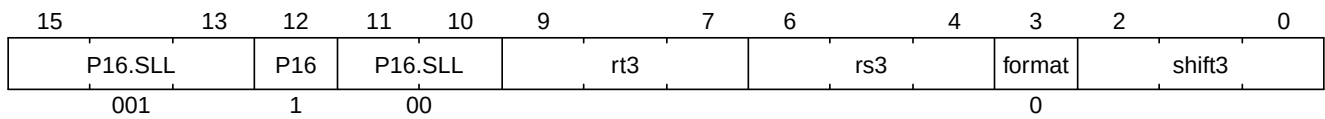


Figure 1501. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
shift = zero_extend(8 if shift3 == 0 else shift3, from_nbits=4)
return Sll0operands(rt=rt, rs=rs, shift=shift)
```

sll.32

Assembly

SLL {rt:gpr}, {rs:gpr}, {shift:u5}

Availability

true

Decode priority

0

Overridden by higher-priority formats

- EHB (ehb.32)
- NOP (nop.32)
- PAUSE (pause.32)
- SYNC (sync.32)

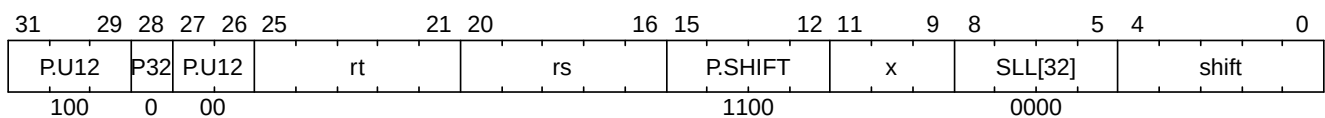


Figure 1502. Encoding (32 bits)

Decode

```
shift = zero_extend(shift, from_nbits=5)
return Sll0operands(rt=rt, rs=rs, shift=shift)
```

8.541.3. Operation

```
source = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
result = int(source) << int(operands.shift)
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.541.4. Description

SLL shifts a 32-bit source word and sign-extends the low 32-bit result to the implemented GPR width. SLL[16] uses compact GPR3 mappings and interprets an encoded shift of zero as eight. SLL[32] has intentional encoding overlap with NOP, EHB, PAUSE, and SYNC aliases; those higher-priority architectural interpretations take precedence during decoding.

8.541.5. Exceptions

None.

8.542. SLLV

8.542.1. Purpose

Shift the 32-bit word in `rs` left by the low five bits of `rt` and place the result in `rd`.

8.542.2. Formats

MIPS

`sllv`

Assembly

`SLLV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

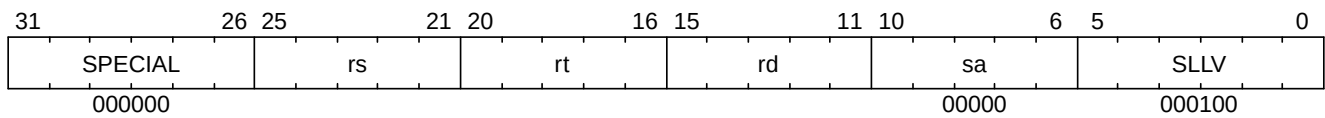


Figure 1503. Encoding (32 bits)

Decode

```
del ctx, exec
return SllvOperands(rd=rd, rs=rt, rt=rs)
```

microMIPS

`sllv.32`

Assembly

`SLLV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

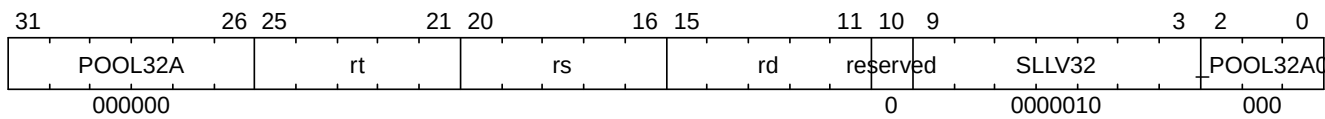


Figure 1504. Encoding (32 bits)

Decode

```
del ctx, exec
return SllvOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

sllv.32

Assembly

SLLV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

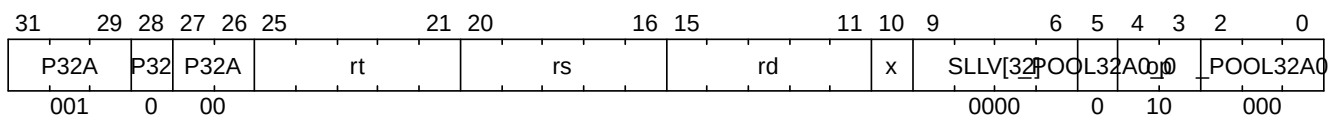


Figure 1505. Encoding (32 bits)

Decode

```
return SllvOperands(rd=rd, rs=rs, rt=rt)
```

8.542.3. Operation

```
source = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
shift_source = zero_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
shift = int(shift_source) & 0x1F
result = int(source) << shift
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.542.4. Description

SLLV uses the low five bits of the full-width shift-register value. The source is a 32-bit word, and the low 32-bit shift result is sign-extended to the implemented GPR width.

8.542.5. Exceptions

None.

8.543. SLT

8.543.1. Purpose

Set a GPR to one if the signed value in `rs` is less than the signed value in `rt`, and set it to zero otherwise.

8.543.2. Formats

MIPS

`slt`

Assembly

`SLT {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

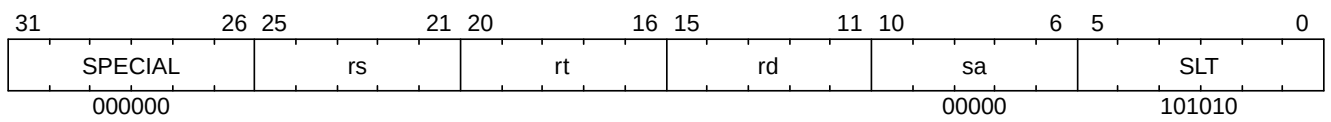


Figure 1506. Encoding (32 bits)

Decode

```
del ctx, exec
return Slt0operands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`slt.32`

Assembly

`SLT {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

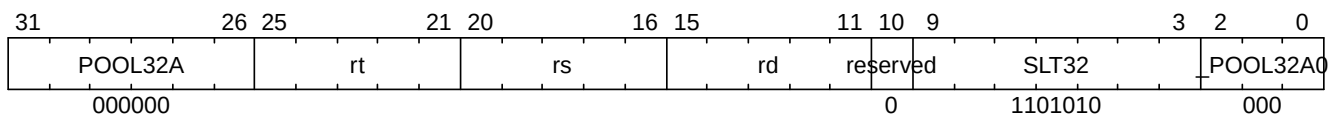


Figure 1507. Encoding (32 bits)

Decode

```
del ctx, exec
return Slt0Operands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

slt.32

Assembly

SLT {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

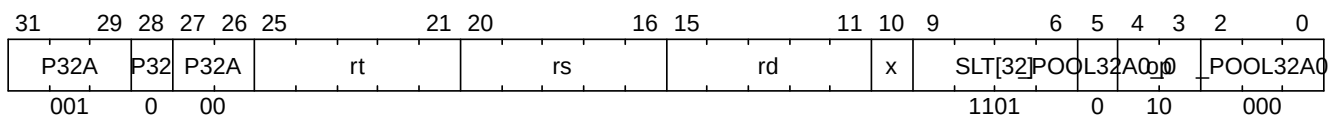


Figure 1508. Encoding (32 bits)

Decode

```
return Slt0Operands(rd=rd, rs=rs, rt=rt)
```

8.543.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
ctx.GPR[operands.rd] = int(lhs < rhs)
```

8.543.4. Description

SLT compares the full implemented GPR width as signed values. The result is the Boolean value zero or one. nanoMIPS encodes SLT as the 32-bit register-register form.

8.543.5. Exceptions

None.

8.544. SLTI

8.544.1. Purpose

Set a GPR to one if a signed source GPR is less than an unsigned immediate, and set it to zero otherwise.

8.544.2. Formats

MIPS

`slti`

Assembly

`SLTI {rt:gpr}, {rs:gpr}, {imm:s16}`

Availability

true

Decode priority

0

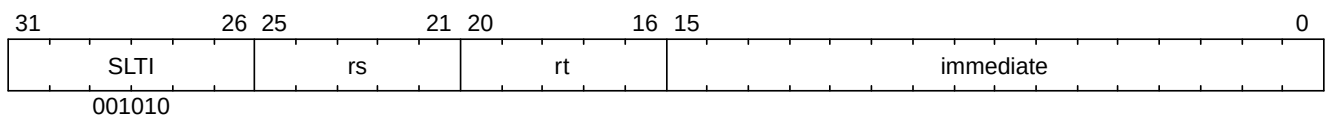


Figure 1509. Encoding (32 bits)

Decode

```
del ctx, exec
return Slti0operands(rt=rt, rs=rs, imm=sign_extend(immediate, from_nbits=16))
```

microMIPS

`slti.32`

Assembly

`SLTI {rt:gpr}, {rs:gpr}, {imm:s16}`

Availability

true

Decode priority

0

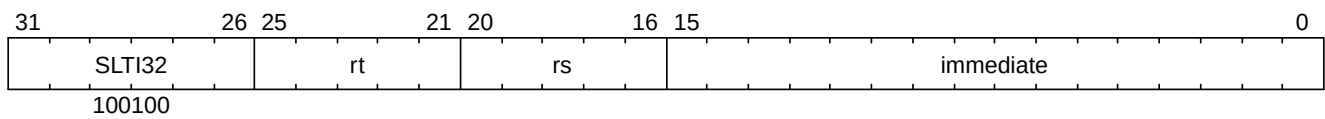


Figure 1510. Encoding (32 bits)

Decode

```
del ctx, exec
return SltiOperands(rt=rt, rs=rs, imm=sign_extend(immediate, from_nbits=16))
```

nanoMIPS

slti.32

Assembly

SLTI {rt:gpr}, {rs:gpr}, {imm:u12}

Availability

true

Decode priority

0

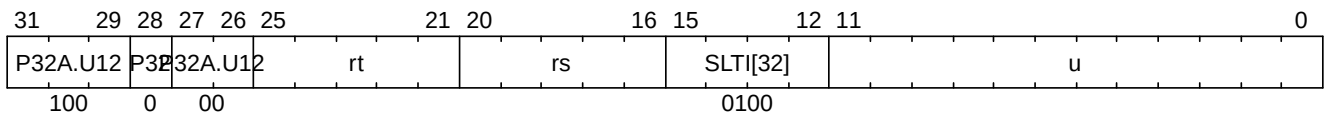


Figure 1511. Encoding (32 bits)

Decode

```
imm = zero_extend(u, from_nbits=12)
return SltiOperands(rt=rt, rs=rs, imm=imm)
```

8.544.3. Operation

```
source = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
ctx.GPR[operands.rt] = int(source < int(operands.imm))
```

8.544.4. Description

SLTI[32] zero-extends its 12-bit u field. The comparison interprets the full implemented GPR value as signed, so the same definition works for 32-bit and 64-bit GPR files.

8.544.5. Exceptions

None.

8.545. SLTIU

8.545.1. Purpose

Set a GPR to one if an unsigned source GPR is less than an unsigned immediate, and set it to zero otherwise.

8.545.2. Formats

MIPS

sltiu

Assembly

SLTIU {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

true

Decode priority

0

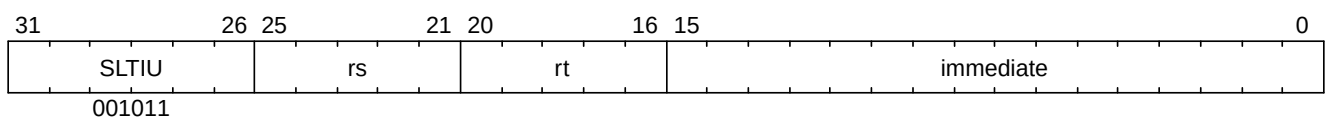


Figure 1512. Encoding (32 bits)

Decode

```
del ctx, exec
# MIPS sign-extends the immediate to the implemented GPR width before
# the unsigned comparison. Keep the signed value attached to the
# operand; the operation applies the final pipeline-width interpretation.
return SltiuOperands(
    rt=rt,
    rs=rs,
    imm=sign_extend(immediate, from_nbits=16),
)
```

microMIPS

sltiu.32

Assembly

SLTIU {rt:gpr}, {rs:gpr}, {imm:s16}

Availability

true

Decode priority

0

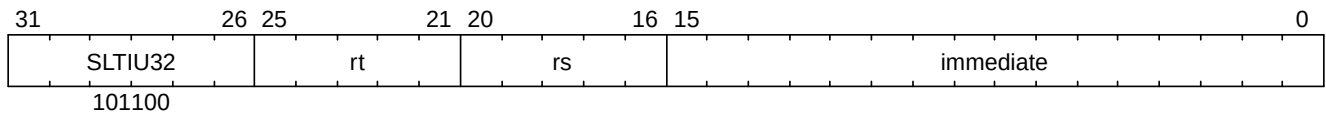


Figure 1513. Encoding (32 bits)

Decode

```

del ctx, exec
# MIPS sign-extends the immediate to the implemented GPR width before
# the unsigned comparison. Keep the signed value attached to the
# operand; the operation applies the final pipeline-width interpretation.
return SltiuOperands(
    rt=rt,
    rs=rs,
    imm=sign_extend(immediate, from_nbits=16),
)

```

nanoMIPS

sltiu.32

Assembly

SLTIU {rt:gpr}, {rs:gpr}, {imm:u12}

Availability

true

Decode priority

0

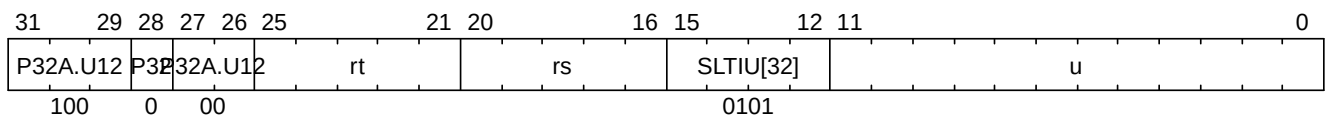


Figure 1514. Encoding (32 bits)

Decode

```

imm = zero_extend(u, from_nbits=12)
return SltiuOperands(rt=rt, rs=rs, imm=imm)

```

8.545.3. Operation

```

lhs = zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = zero_extend(ctx.GPR[operands.imm], from_nbits=ctx.GPR.width)
ctx.GPR[operands.rt] = int(lhs < rhs)

```

8.545.4. Description

The classic MIPS immediate is sign-extended to the implemented GPR width before the unsigned comparison. The nanoMIPS[32] immediate is zero-extended. The normal architectural result is written to the destination GPR.

8.545.5. Exceptions

None.

8.546. SLTU

8.546.1. Purpose

Set a GPR to one if the unsigned value in `rs` is less than the unsigned value in `rt`, and set it to zero otherwise.

8.546.2. Formats

MIPS

`sltu`

Assembly

`SLTU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

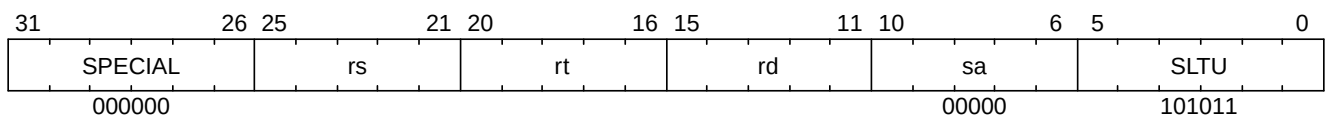


Figure 1515. Encoding (32 bits)

Decode

```
del ctx, exec
return SltuOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`sltu.32`

Assembly

`SLTU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

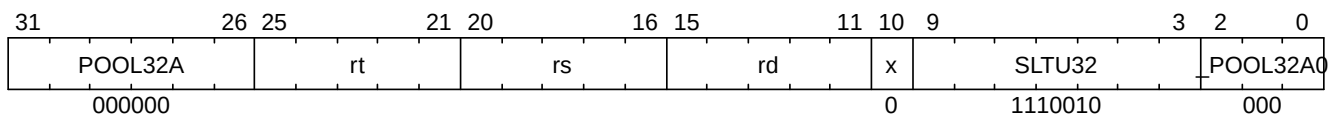


Figure 1516. Encoding (32 bits)

Encoding constraints

- `rd != 0`

Decode

```
del ctx, exec
return SltuOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

sltu.32

Assembly

SLTU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

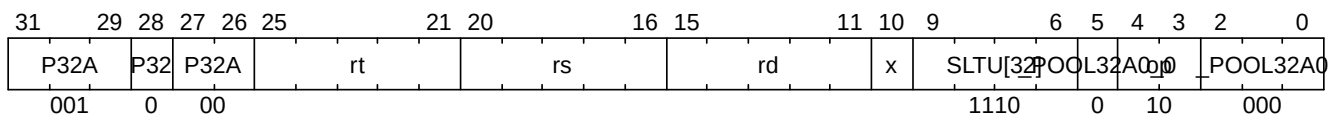


Figure 1517. Encoding (32 bits)

Encoding constraints

- `rd != 0`

Decode

```
return SltuOperands(rd=rd, rs=rs, rt=rt)
```

8.546.3. Operation

```
lhs = zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = zero_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
ctx.GPR[operands.rd] = int(lhs < rhs)
```

8.546.4. Description

SLTU compares the full implemented GPR width as unsigned values. The result is the Boolean value zero or one. The rd=0 bit pattern belongs to DVP/EVP rather than SLTU, so this form constrains rd to a nonzero GPR.

8.546.5. Exceptions

None.

8.547. SOV

8.547.1. Purpose

Set a GPR when a signed 32-bit addition overflows.

8.547.2. Formats

nanoMIPS

sov.32

Assembly

SOV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

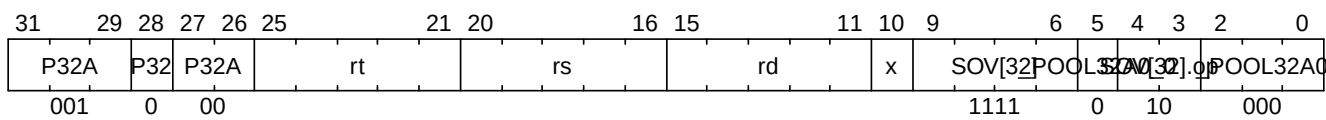


Figure 1518. Encoding (32 bits)

Decode

```
del ctx, exec
return SovOperands(rd=rd, rs=rs, rt=rt)
```

8.547.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
result = int(lhs) + int(rhs)
ctx.GPR[operands.rd] = int(int(sign_extend(result, from_nbits=32)) != result)
```

8.547.4. Description

SOV[32] writes one if the signed low-word sum of rs and rt overflows 32 bits, otherwise zero.

8.547.5. Exceptions

None.

8.548. Sqrt.D

8.548.1. Purpose

Compute a D-format square root.

8.548.2. Formats

MIPS

`sqrt.d`

Assembly

`Sqrt.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

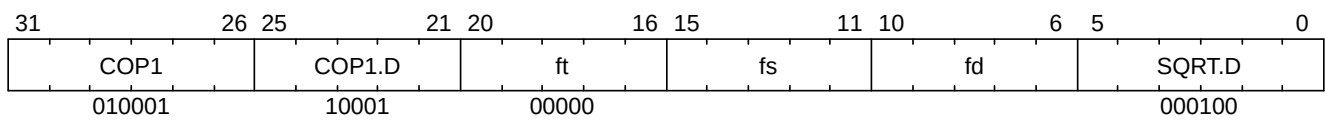


Figure 1519. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

`sqrt.d`

Assembly

`Sqrt.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

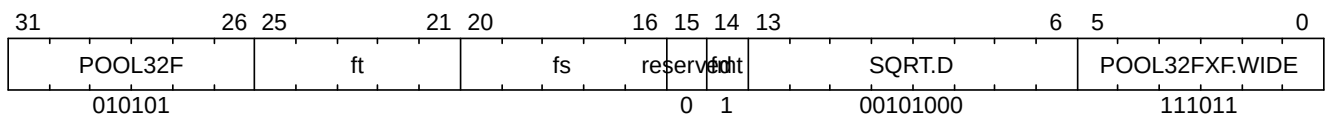


Figure 1520. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

sqrt.d

Assembly

SQRT.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

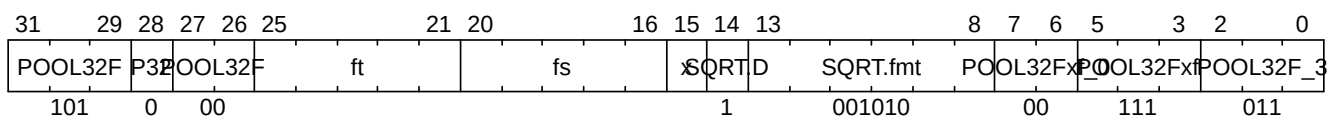


Figure 1521. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.548.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_sqrt(
    "D", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "D")
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.548.4. Description

SQRT.D validates CP1 and D-format support before updating the destination FPR.

8.548.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.549. Sqrt.S

8.549.1. Purpose

Compute an S-format square root.

8.549.2. Formats

MIPS

`sqrt.s`

Assembly

`SQRT.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

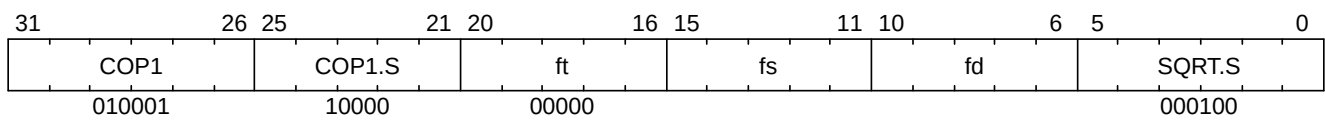


Figure 1522. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=fd, fs=fs)
```

microMIPS

`sqrt.s`

Assembly

`SQRT.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

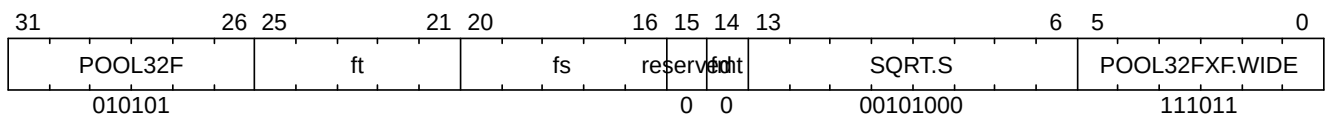


Figure 1523. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

nanoMIPS

sqrt.s

Assembly

SQRT.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

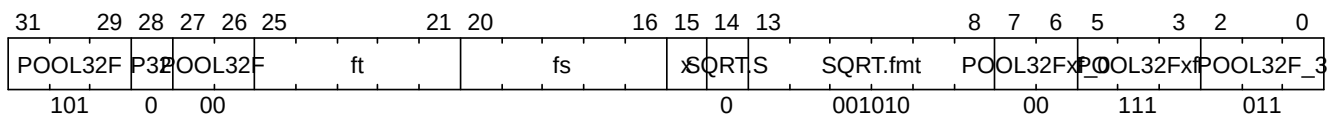


Figure 1524. Encoding (32 bits)

Decode

```
return FpuUnaryOperands(fd=ft, fs=fs)
```

8.549.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_sqrt(
    "S", fpu_rounding_mode(ctx), get_fpr(ctx, operands.fs, "S")
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.549.4. Description

SQRT.S validates CP1 and S-format support before updating the destination FPR.

8.549.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.550. SRA

8.550.1. Purpose

Arithmetic-shift the signed 32-bit word in `rs` right by `shift` bits and place the result in `rt`.

8.550.2. Formats

MIPS

`sra`

Assembly

`SRA {rt:gpr}, {rs:gpr}, {shift:u5}`

Availability

true

Decode priority

0

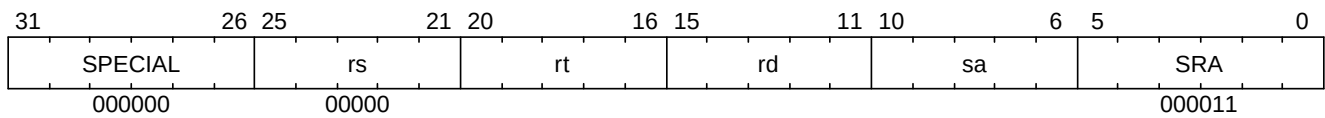


Figure 1525. Encoding (32 bits)

Decode

```
del ctx, exec
return Sra0perands(rt=rd, rs=rt, shift=zero_extend(sa, from_nbits=5))
```

microMIPS

`sra.32`

Assembly

`SRA {rt:gpr}, {rs:gpr}, {shift:u5}`

Availability

true

Decode priority

0

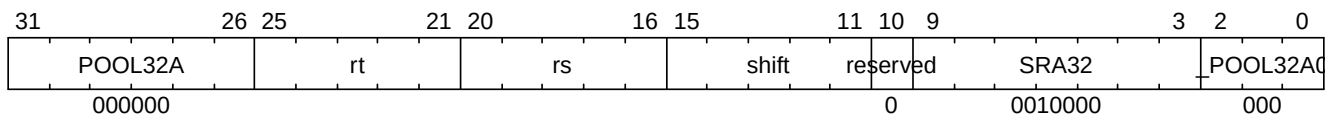


Figure 1526. Encoding (32 bits)

Decode

```
shift = zero_extend(shift, from_nbits=5)
return SraOperands(rt=rt, rs=rs, shift=shift)
```

nanoMIPS

sra.32

Assembly

SRA {rt:gpr}, {rs:gpr}, {shift:u5}

Availability

true

Decode priority

0

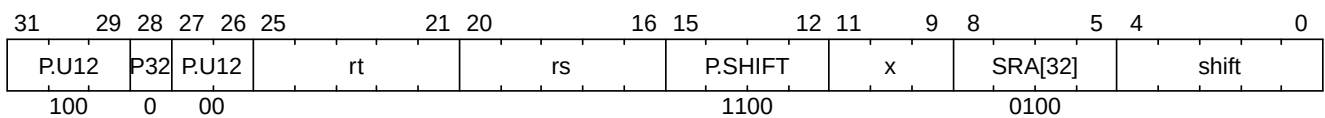


Figure 1527. Encoding (32 bits)

Decode

```
shift = zero_extend(shift, from_nbits=5)
return SraOperands(rt=rt, rs=rs, shift=shift)
```

8.550.3. Operation

```
source = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
result = int(source) >> int(operands.shift)
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.550.4. Description

SRA uses the sign bit of the low 32-bit source word to fill the shifted-in bits. The resulting 32-bit word is sign-extended to the implemented GPR width. nanoMIPS provides this operation in the 32-bit encoding only.

8.550.5. Exceptions

None.

8.551. SRAV

8.551.1. Purpose

Arithmetic-shift the signed 32-bit word in `rs` right by the low five bits of `rt` and place the result in `rd`.

8.551.2. Formats

MIPS

`sra`

Assembly

`SRAV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

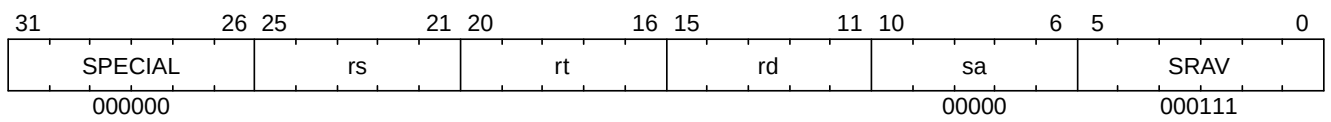


Figure 1528. Encoding (32 bits)

Decode

```
del ctx, exec
return SraVOperands(rd=rd, rs=rt, rt=rs)
```

microMIPS

`sra.32`

Assembly

`SRAV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

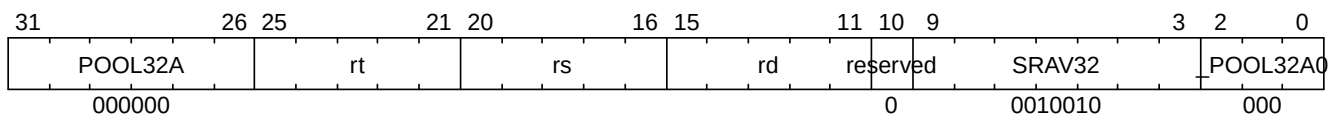


Figure 1529. Encoding (32 bits)

Decode

```
del ctx, exec
return SravOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

srav.32

Assembly

SRAV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

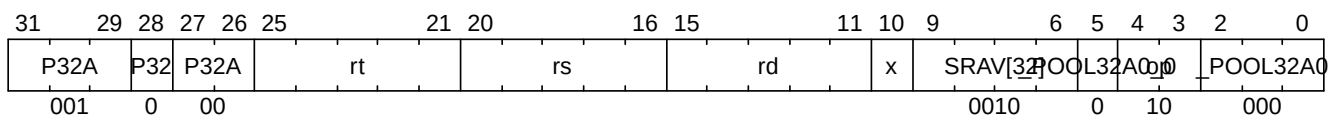


Figure 1530. Encoding (32 bits)

Decode

```
return SravOperands(rd=rd, rs=rs, rt=rt)
```

8.551.3. Operation

```
shift = int(zero_extend(ctx.GPR[operands.rt], from_nbits=5))
source = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
result = int(source) >> shift
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.551.4. Description

SRAV[32] uses the low five bits of the full-width shift-register value. It arithmetic-shifts the low 32-bit source word, duplicating its sign bit, and sign-extends the 32-bit result to the implemented GPR width.

8.551.5. Exceptions

None.

8.552. SRL

8.552.1. Purpose

Shift a 32-bit word right logically by a constant amount and place the result in a destination GPR.

8.552.2. Formats

MIPS

`srl`

Assembly

`SRL {rt:gpr}, {rs:gpr}, {shift:u5}`

Availability

true

Decode priority

0

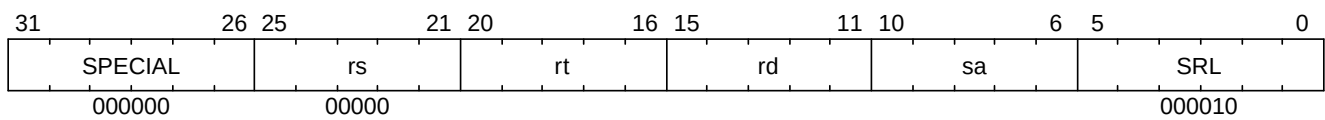


Figure 1531. Encoding (32 bits)

Decode

```
del ctx, exec
return SrlOperands(rt=rd, rs=rt, shift=zero_extend(sa, from_nbits=5))
```

microMIPS

`srl.32`

Assembly

`SRL {rt:gpr}, {rs:gpr}, {shift:u5}`

Availability

true

Decode priority

0

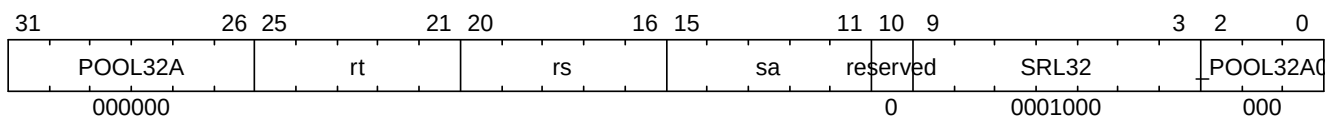


Figure 1532. Encoding (32 bits)

Decode

```
return Srl0perands(rt=rt, rs=rs, shift=zero_extend(sa, from_nbits=5))
```

srl.16

Assembly

SRL {rt:gpr3}, {rs:gpr3}, {shift:u4}

Availability

true

Decode priority

0

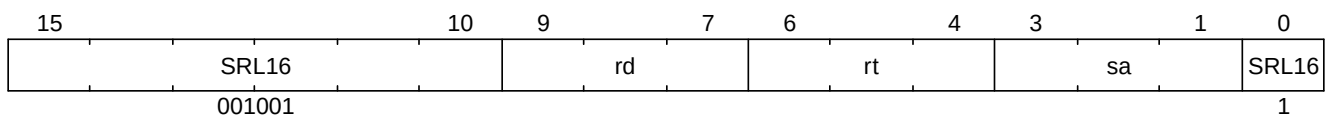


Figure 1533. Encoding (16 bits)

Decode

```
del ctx, exec
shift = zero_extend(8 if sa == 0 else sa, from_nbits=4)
return Srl0perands(
    rt=MICROMIPS_GPR3.decode(rd),
    rs=MICROMIPS_GPR3.decode(rt),
    shift=shift,
)
```

nanoMIPS

srl.16

Assembly

SRL {rt:gpr3}, {rs:gpr3}, {shift:u4}

Availability

true

Decode priority

0

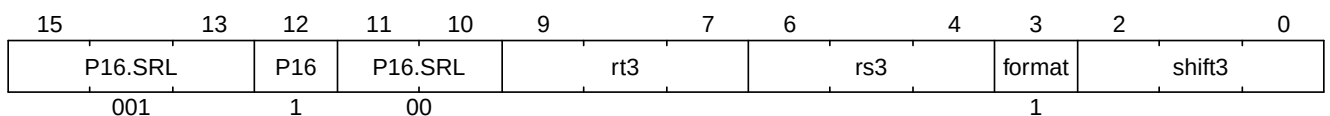


Figure 1534. Encoding (16 bits)

Decode

```
rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
shift = zero_extend(8 if shift3 == 0 else shift3, from_nbits=4)
return SrlOperands(rt=rt, rs=rs, shift=shift)
```

srl.32

Assembly

SRL {rt:gpr}, {rs:gpr}, {shift:u5}

Availability

true

Decode priority

0

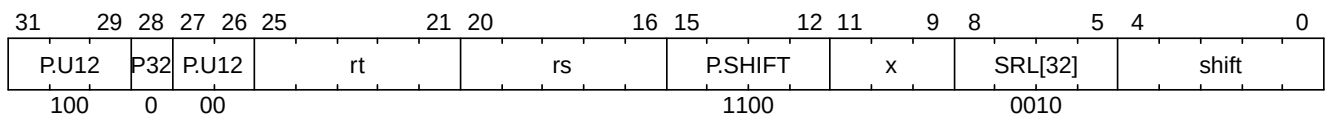


Figure 1535. Encoding (32 bits)

Decode

```
shift = zero_extend(shift, from_nbits=5)
return SrlOperands(rt=rt, rs=rs, shift=shift)
```

8.552.3. Operation

```
source = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
result = int(source) >> int(operands.shift)
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.552.4. Description

SRL shifts the low 32 bits of **rs**, filling vacated bits with zero, then sign-extends the 32-bit result to the implemented GPR width. SRL[16] uses compact GPR3 mappings and interprets an encoded shift of zero as eight; SRL[32] encodes a five-bit shift amount.

8.552.5. Exceptions

None.

8.553. SRLV

8.553.1. Purpose

Shift the 32-bit word in `rs` right logically by the amount in `rt` and place the result in `rd`.

8.553.2. Formats

MIPS

`srlv`

Assembly

`SRLV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

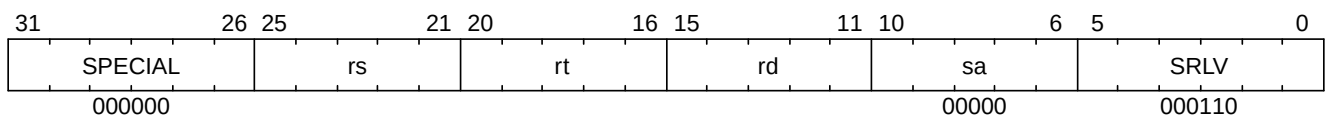


Figure 1536. Encoding (32 bits)

Decode

```
del ctx, exec
return SrlvOperands(rd=rd, rs=rt, rt=rs)
```

microMIPS

`srlv.32`

Assembly

`SRLV {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

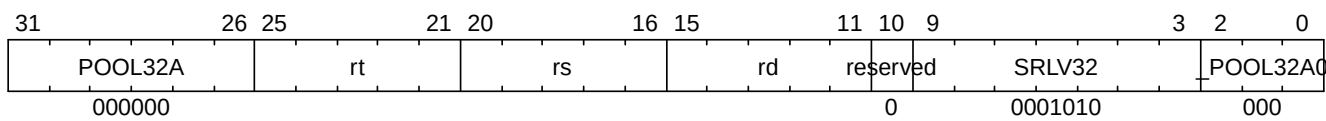


Figure 1537. Encoding (32 bits)

Decode

```
del ctx, exec
return SrlvOperands(rd=rd, rs=rt, rt=rs)
```

nanoMIPS

srlv.32

Assembly

SRLV {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

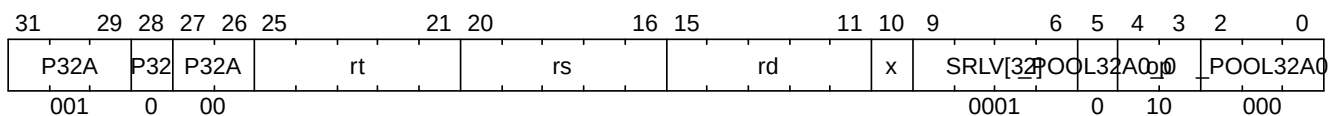


Figure 1538. Encoding (32 bits)

Decode

```
return SrlvOperands(rd=rd, rs=rs, rt=rt)
```

8.553.3. Operation

```
shift = zero_extend(ctx.GPR[operands.rt], from_nbits=5)
source = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
result = int(source) >> int(shift)
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.553.4. Description

SRLV uses the low five bits of the full-width **rt** GPR as the shift amount. It shifts the low 32 bits of **rs** with zero fill, then sign-extends the 32-bit result to the implemented GPR width.

8.553.5. Exceptions

None.

8.554. SUB

8.554.1. Purpose

Subtract two signed 32-bit GPR values and trap on overflow.

8.554.2. Formats

MIPS

sub

Assembly

SUB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

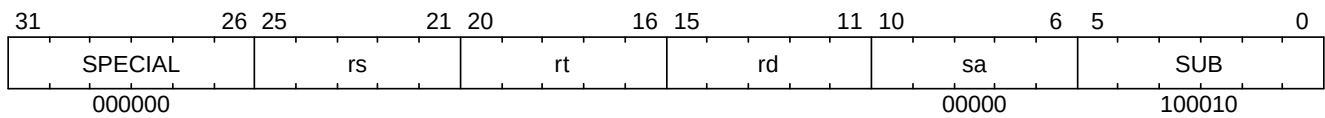


Figure 1539. Encoding (32 bits)

Decode

```
del ctx, exec
return SubOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

sub.32

Assembly

SUB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

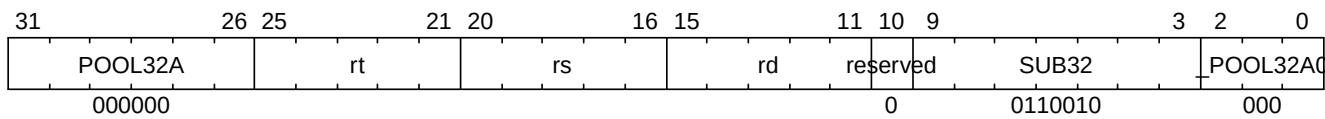


Figure 1540. Encoding (32 bits)

Decode

```
del ctx, exec
return SubOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

sub.32

Assembly

SUB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

!(NMS)

Decode priority

0

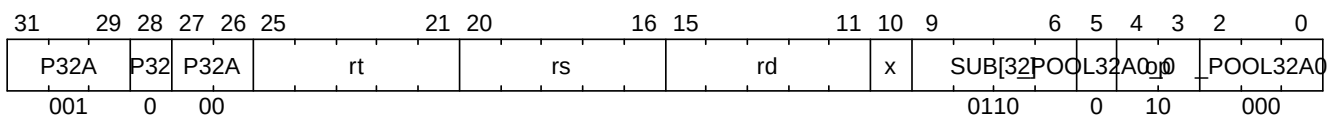


Figure 1541. Encoding (32 bits)

Decode

```
del ctx, exec
return SubOperands(rd=rd, rs=rs, rt=rt)
```

8.554.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
result = int(lhs) - int(rhs)
if int(sign_extend(result, from_nbits=32)) != result:
    raise IntegerOverflowException("SUB signed 32-bit overflow")
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.554.4. Description

SUB sign-extends both source GPRs from 32 bits, traps when the mathematical difference cannot be represented as a signed 32-bit value, and sign-extends the 32-bit result to the active GPR width. SUB is not available in the nanoMIPS subset.

8.554.5. Exceptions

- **OV** (Integer Overflow) — SUB signed 32-bit overflow
- **RI** (Reserved Instruction) — SUB is unavailable in NMS mode.; SUB is reserved when CP0

Config5.NMS is set.

8.555. SUB.D

8.555.1. Purpose

Subtract two D-format floating-point values.

8.555.2. Formats

MIPS

sub.d

Assembly

SUB.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

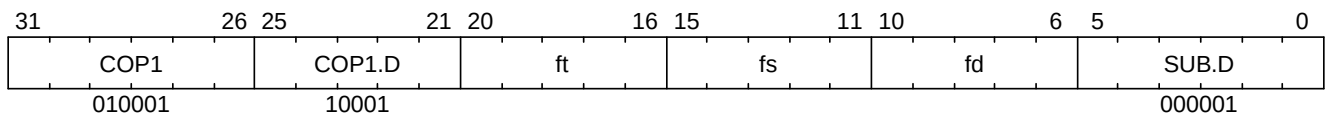


Figure 1542. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

sub.d

Assembly

SUB.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

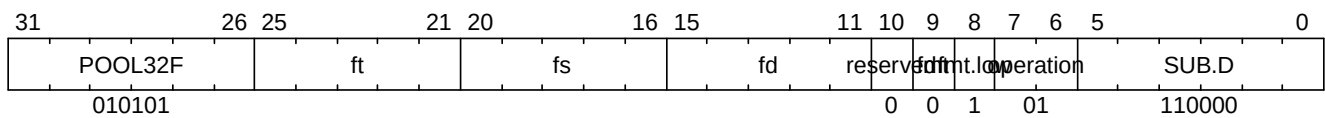


Figure 1543. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

sub.d

Assembly

SUB.D {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

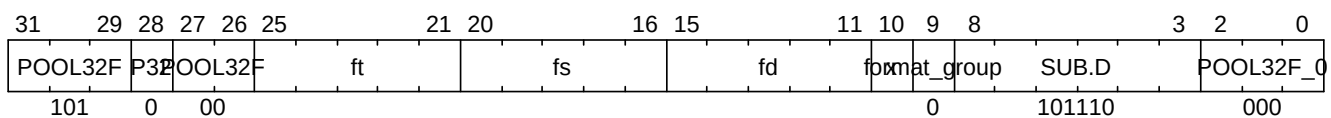


Figure 1544. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.555.3. Operation

```
validate_access_to_cp1_resources(ctx, "D")
result, flags = ieee_add(
    "D",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "D"),
    get_fpr(ctx, operands.ft, "D"),
    subtract=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "D")
```

8.555.4. Description

SUB.D validates CP1 and D-format support before updating the destination FPR.

8.555.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.556. SUB.PS

8.556.1. Purpose

Perform the paired-single sub operation.

8.556.2. Formats

MIPS

sub.ps

Assembly

SUB.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

AR < R6

Decode priority

0

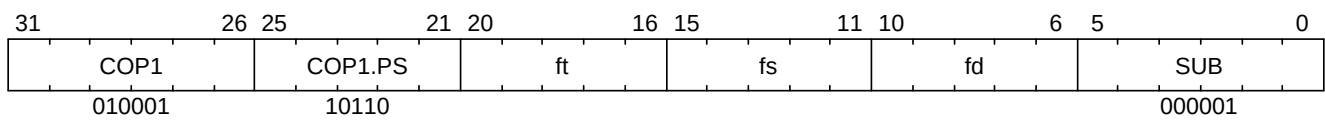


Figure 1545. Encoding (32 bits)

Decode

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

sub.ps

Assembly

SUB.PS {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

MMAR < R6

Decode priority

0

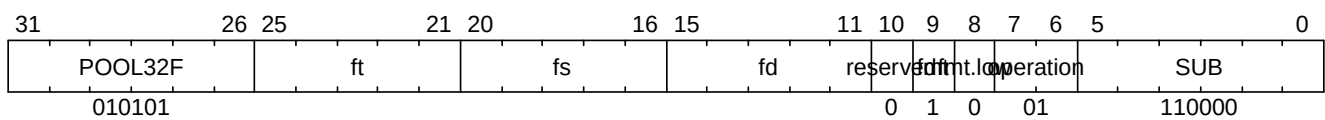


Figure 1546. Encoding (32 bits)

```
return FpuPSBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.556.3. Operation

```
validate_access_to_cp1_resources(ctx, "PS")
rounding = fpu_rounding_mode(ctx)
left = get_fpr(ctx, operands.fs, "PS")
right = get_fpr(ctx, operands.ft, "PS")
low, low_flags = ieee_add(
    "S", rounding, left & 0xFFFF_FFFF, right & 0xFFFF_FFFF, subtract=True
)
high, high_flags = ieee_add("S", rounding, left >> 32, right >> 32, subtract=True)
write_fpu_result(ctx, operands.fd, low | (high << 32), low_flags | high_flags, "PS")
```

8.556.4. Description

SUB.PS applies independently to the upper and lower S-format lanes.

8.556.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.557. SUB.S

8.557.1. Purpose

Subtract two S-format floating-point values.

8.557.2. Formats

MIPS

sub.s

Assembly

SUB.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

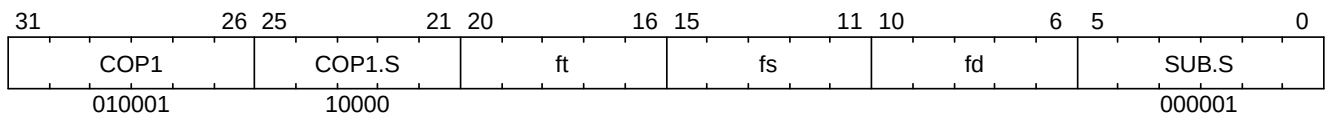


Figure 1547. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

microMIPS

sub.s

Assembly

SUB.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

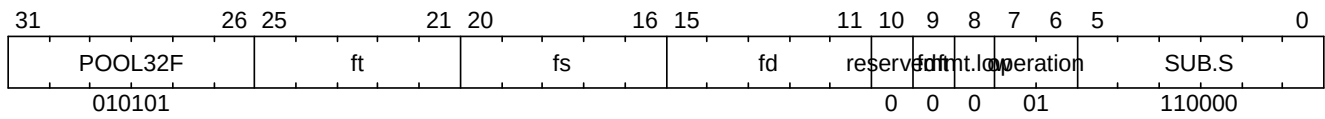


Figure 1548. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

nanoMIPS

sub.s

Assembly

SUB.S {fd:fpr}, {fs:fpr}, {ft:fpr}

Availability

true

Decode priority

0

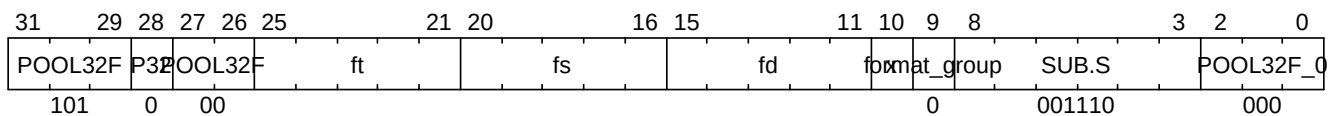


Figure 1549. Encoding (32 bits)

Decode

```
return FpuBinaryOperands(fd=fd, fs=fs, ft=ft)
```

8.557.3. Operation

```
validate_access_to_cp1_resources(ctx, "S")
result, flags = ieee_add(
    "S",
    fpu_rounding_mode(ctx),
    get_fpr(ctx, operands.fs, "S"),
    get_fpr(ctx, operands.ft, "S"),
    subtract=True,
)
write_fpu_result(ctx, operands.fd, result, flags, "S")
```

8.557.4. Description

SUB.S validates CP1 and S-format support before updating the destination FPR.

8.557.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.

- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported

8.558. SUBQ.PH

8.558.1. Purpose

DSP packed operation SUBQ.PH.

8.558.2. Formats

MIPS

subq.ph

Assembly

SUBQ.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

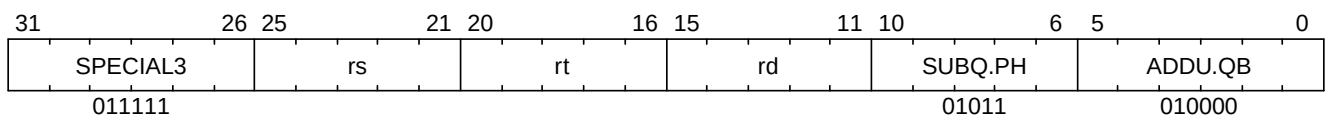


Figure 1550. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subq.ph.32

Assembly

SUBQ.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

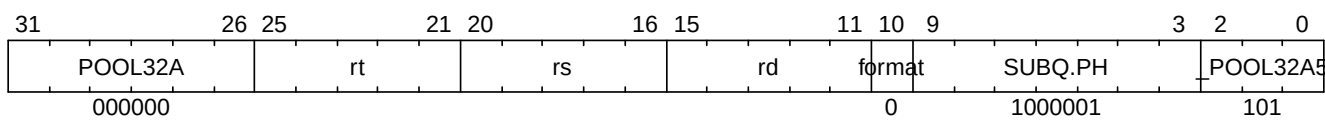


Figure 1551. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subq.ph.32

Assembly

SUBQ.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

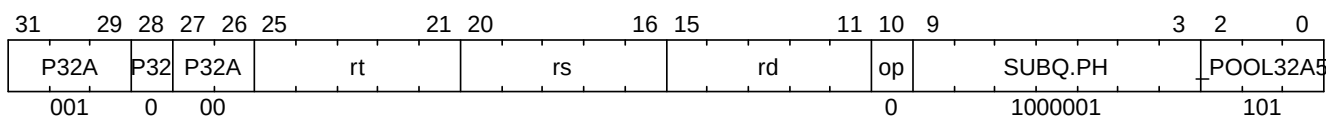


Figure 1552. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.558.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, subtract=True
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.558.4. Description

SUBQ.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.558.5. Exceptions

None.

8.559. SUBQ_S.PH

8.559.1. Purpose

DSP packed operation SUBQ_S.PH.

8.559.2. Formats

MIPS

subq_s.ph

Assembly

SUBQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

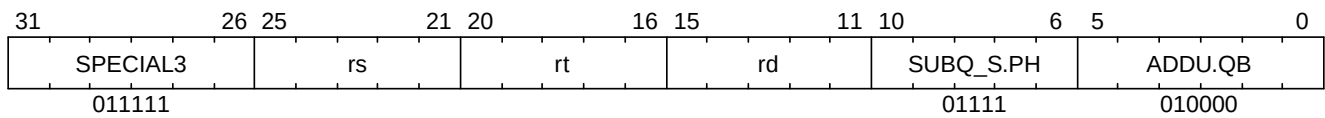


Figure 1553. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subq_s.ph.32

Assembly

SUBQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

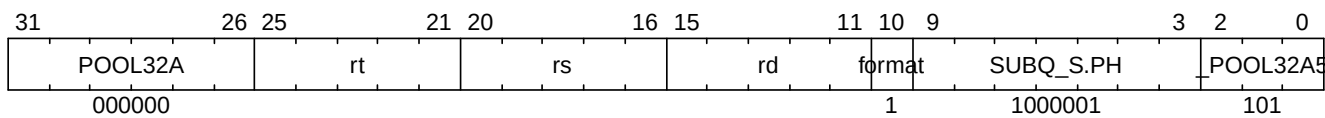


Figure 1554. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subq_s.ph.32

Assembly

SUBQ_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

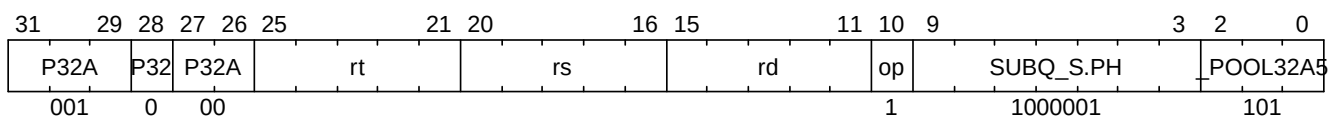


Figure 1555. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.559.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, subtract=True, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.559.4. Description

SUBQ_S.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.559.5. Exceptions

None.

8.560. SUBQ_S.W

8.560.1. Purpose

DSP packed operation SUBQ_S.W.

8.560.2. Formats

MIPS

subq_s.w

Assembly

SUBQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

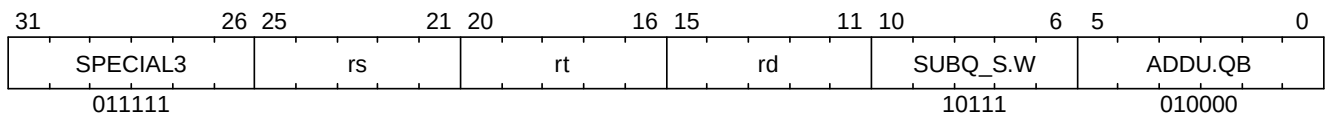


Figure 1556. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subq_s.w.32

Assembly

SUBQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

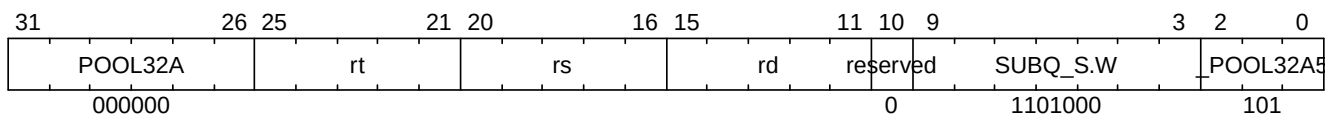


Figure 1557. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subq_s.w.32

Assembly

SUBQ_S.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

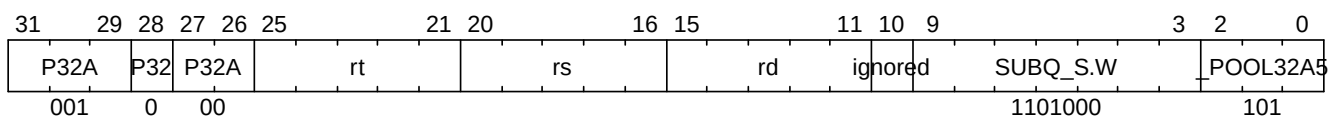


Figure 1558. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.560.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 32, 1, subtract=True, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.560.4. Description

SUBQ_S.W is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.560.5. Exceptions

None.

8.561. SUBQH.PH

8.561.1. Purpose

DSP packed operation SUBQH.PH.

8.561.2. Formats

MIPS

subqh.ph

Assembly

SUBQH.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

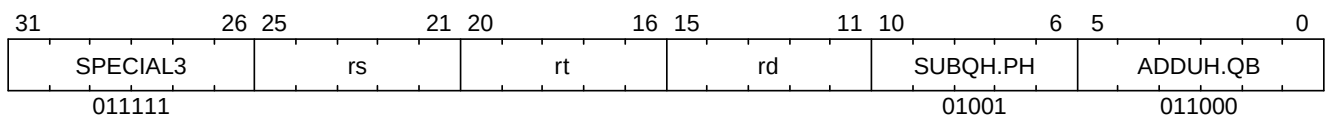


Figure 1559. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subqh.ph.32

Assembly

SUBQH.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

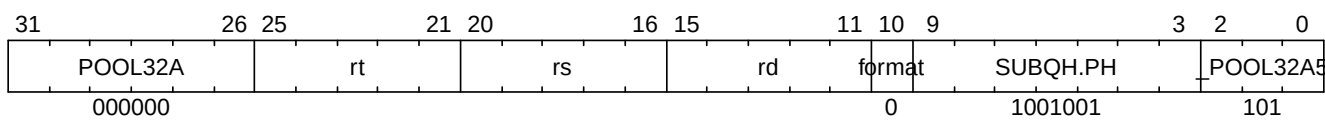


Figure 1560. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subqh.ph.32

Assembly

SUBQH.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

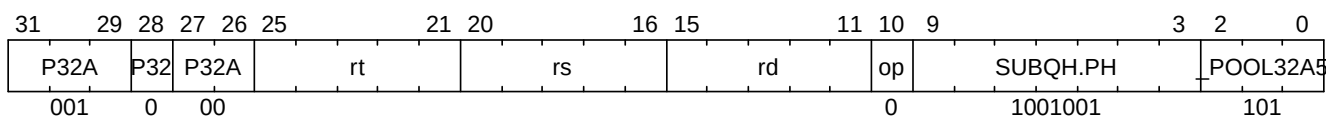


Figure 1561. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.561.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, subtract=True, half=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.561.4. Description

SUBQH.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.561.5. Exceptions

None.

8.562. SUBQH.W

8.562.1. Purpose

DSP packed operation SUBQH.W.

8.562.2. Formats

MIPS

subqh.w

Assembly

SUBQH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

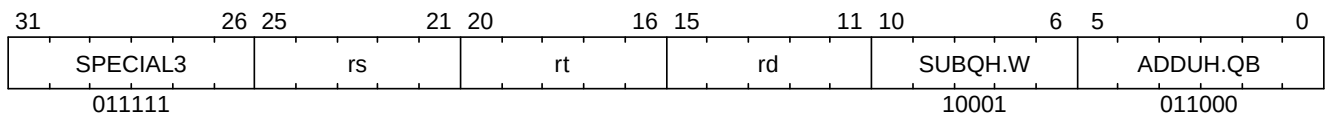


Figure 1562. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subqh.w.32

Assembly

SUBQH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

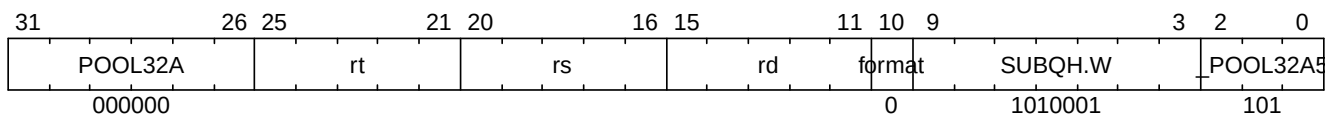


Figure 1563. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subqh.w.32

Assembly

SUBQH.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

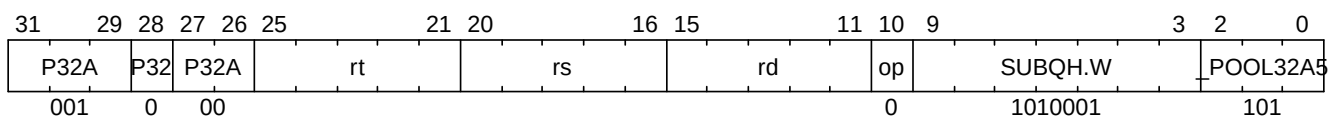


Figure 1564. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.562.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 32, 1, subtract=True, half=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.562.4. Description

SUBQH.W is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.562.5. Exceptions

None.

8.563. SUBQH_R.PH

8.563.1. Purpose

DSP packed operation SUBQH_R.PH.

8.563.2. Formats

MIPS

subqh_r.ph

Assembly

SUBQH_R.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

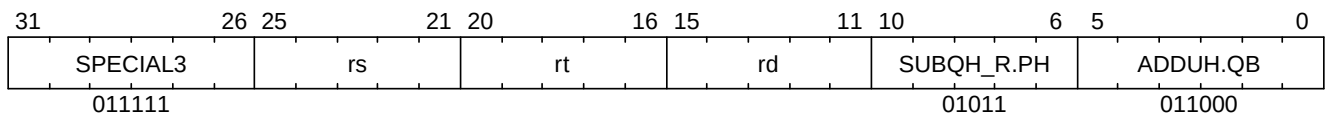


Figure 1565. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subqh_r.ph.32

Assembly

SUBQH_R.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

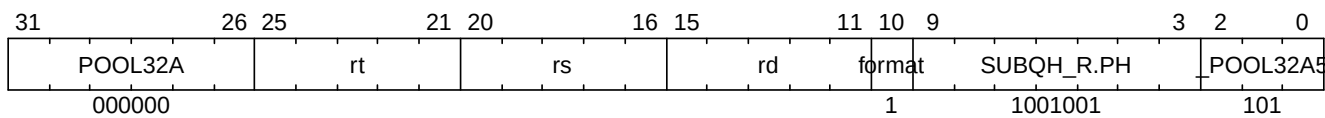


Figure 1566. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subqh_r.ph.32

Assembly

SUBQH_R.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

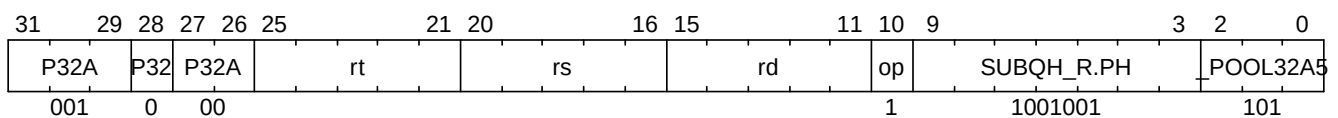


Figure 1567. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.563.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs],
    ctx.GPR[operands.rt],
    16,
    2,
    subtract=True,
    half=True,
    rounded=True,
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.563.4. Description

SUBQH_R.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.563.5. Exceptions

None.

8.564. SUBQH_R.W

8.564.1. Purpose

DSP packed operation SUBQH_R.W.

8.564.2. Formats

MIPS

subqh_r.w

Assembly

SUBQH_R.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

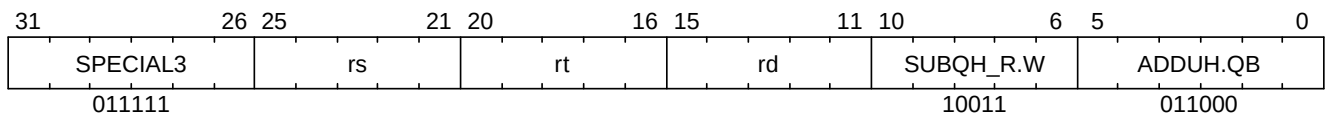


Figure 1568. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subqh_r.w.32

Assembly

SUBQH_R.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

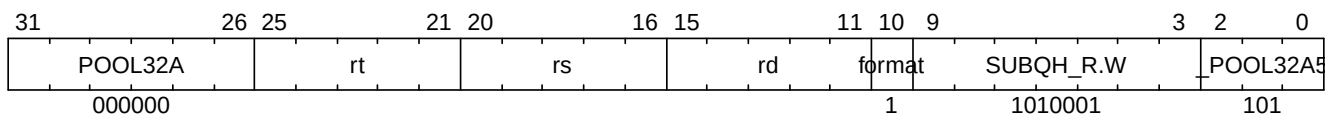


Figure 1569. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subqh_r.w.32

Assembly

SUBQH_R.W {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

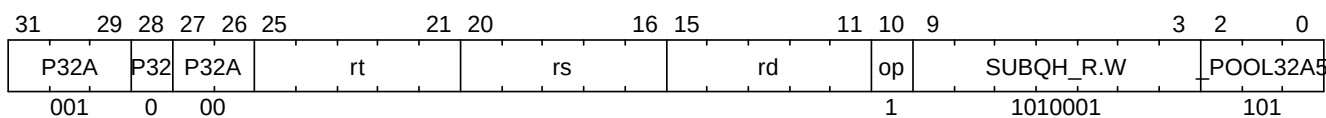


Figure 1570. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.564.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, unused = packed_signed_add(
    ctx.GPR[operands.rs],
    ctx.GPR[operands.rt],
    32,
    1,
    subtract=True,
    half=True,
    rounded=True,
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.564.4. Description

SUBQH_R.W is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.564.5. Exceptions

None.

8.565. SUBU

8.565.1. Purpose

Subtract the 32-bit integer in `rt` from the 32-bit integer in `rs`, place the result in `rd`, and do not trap on overflow.

8.565.2. Formats

MIPS

`subu`

Assembly

`SUBU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

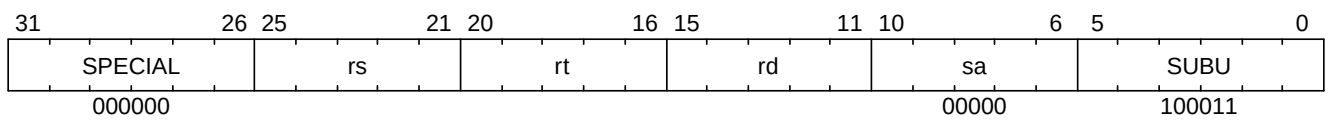


Figure 1571. Encoding (32 bits)

Decode

```
del ctx, exec
return SubuOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

`subu.32`

Assembly

`SUBU {rd:gpr}, {rs:gpr}, {rt:gpr}`

Availability

true

Decode priority

0

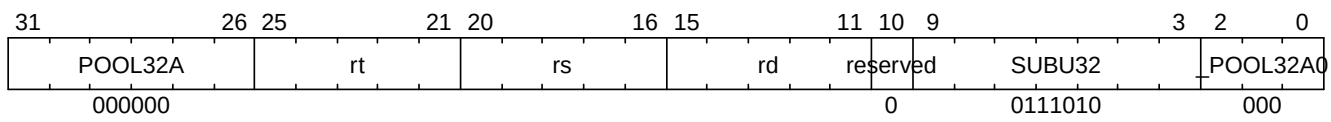


Figure 1572. Encoding (32 bits)

Decode

```
del ctx, exec
return SubuOperands(rd=rd, rs=rs, rt=rt)
```

subu.16

Assembly

```
SUBU {rd:gpr3}, {rs:gpr3}, {rt:gpr3}
```

Availability

MMAR < R6

Decode priority

0

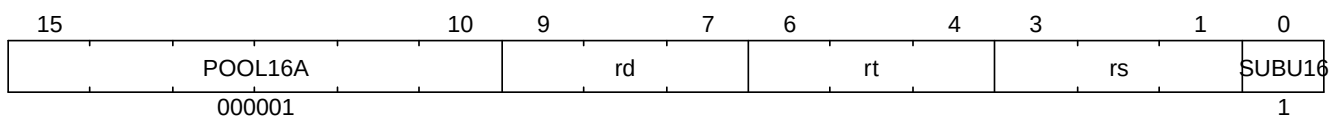


Figure 1573. Encoding (16 bits)

Decode

```
del ctx, exec
return SubuOperands(
    rd=MICROMIPS_GPR3.decode(rd),
    rs=MICROMIPS_GPR3.decode(rs),
    rt=MICROMIPS_GPR3.decode(rt),
)
```

subu.16.r6

Assembly

```
SUBU {rd:gpr3}, {rs:gpr3}, {rt:gpr3}
```

Availability

MMAR >= R6

Decode priority

0

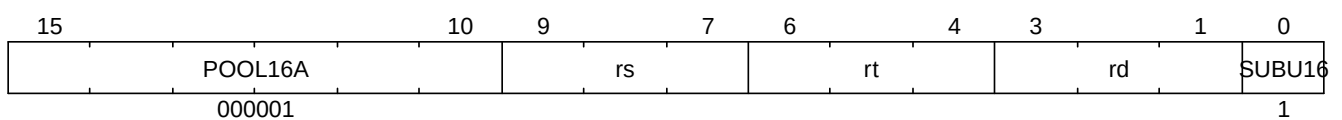


Figure 1574. Encoding (16 bits)

Decode

```
del ctx, exec
return SubuOperands(
```

```

rd=MICROMIPS_GPR3.decode(rd),
rs=MICROMIPS_GPR3.decode(rs),
rt=MICROMIPS_GPR3.decode(rt),
)

```

nanoMIPS

subu.32

Assembly

SUBU {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

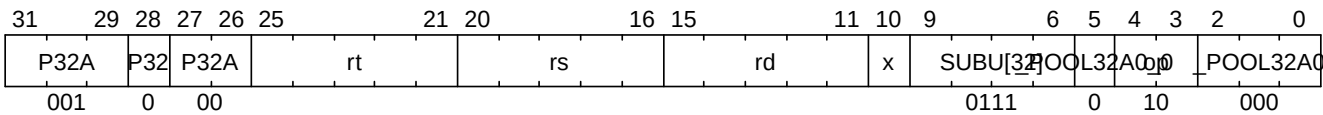


Figure 1575. Encoding (32 bits)

Decode

```

return SubuOperands(rd=rd, rs=rs, rt=rt)

```

subu.16

Assembly

SUBU {rd:gpr3}, {rs:gpr3}, {rt:gpr3}

Availability

true

Decode priority

0

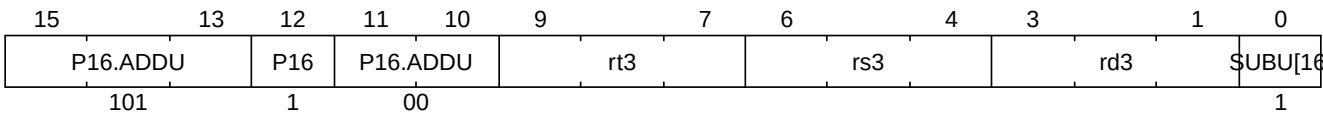


Figure 1576. Encoding (16 bits)

Decode

```

return SubuOperands(
    rd=GPR3.decode(rd3),
    rs=GPR3.decode(rs3),
    rt=GPR3.decode(rt3),
)

```

8.565.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=32)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=32)
result = lhs - rhs
ctx.GPR[operands.rd] = sign_extend(result, from_nbits=32)
```

8.565.4. Description

SUBU computes rs minus rt modulo 2^{32} and sign-extends the low 32 bits to the implemented GPR width. nanoMIPS supplies full-register and compact 16-bit forms.

8.565.5. Exceptions

None.

8.566. SUBU.PH

8.566.1. Purpose

DSP packed operation SUBU.PH.

8.566.2. Formats

MIPS

subu.ph

Assembly

SUBU.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

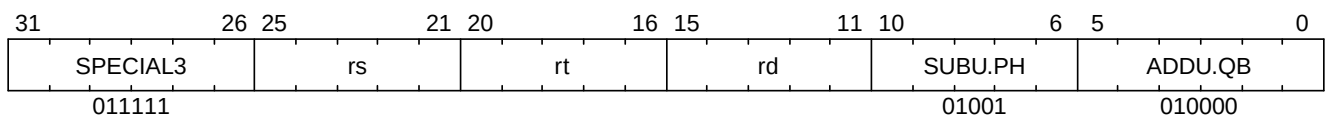


Figure 1577. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subu.ph.32

Assembly

SUBU.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

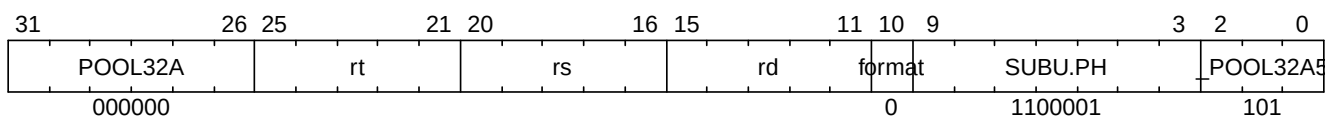


Figure 1578. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subu.ph.32

Assembly

SUBU.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

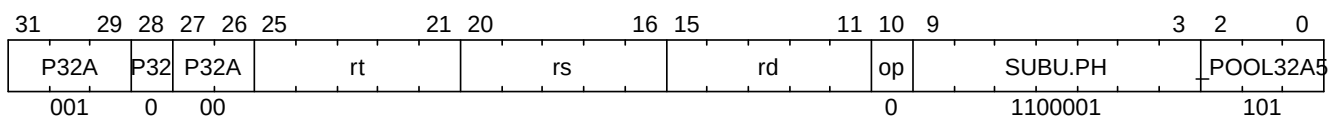


Figure 1579. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.566.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, subtract=True
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.566.4. Description

SUBU.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.566.5. Exceptions

None.

8.567. SUBU.QB

8.567.1. Purpose

DSP packed operation SUBU.QB.

8.567.2. Formats

MIPS

subu.qb

Assembly

SUBU.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

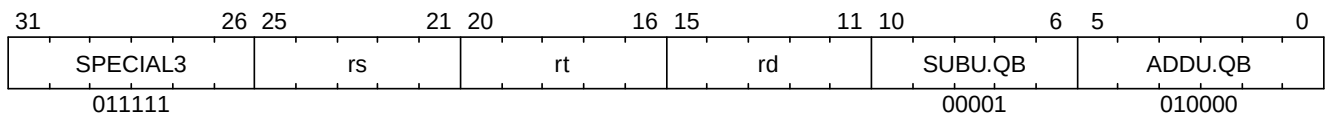


Figure 1580. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subu.qb.32

Assembly

SUBU.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

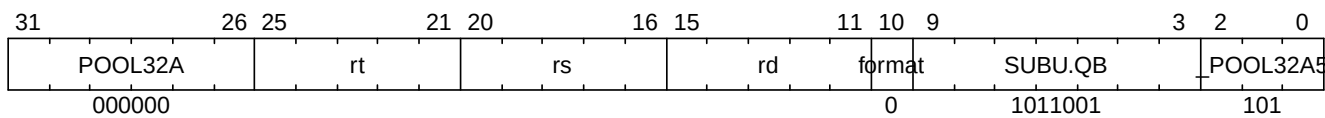


Figure 1581. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subu.qb.32

Assembly

SUBU.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

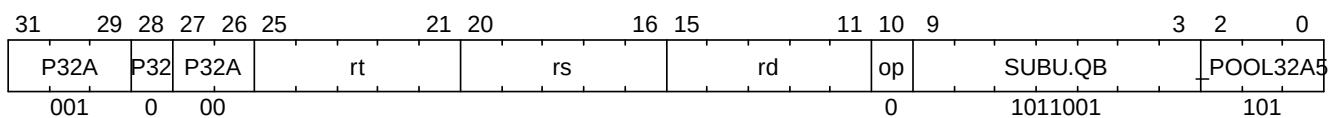


Figure 1582. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.567.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, subtract=True
)
if overflow:
    ctx.DSPControl.Ouflag |= 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.567.4. Description

SUBU.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.567.5. Exceptions

None.

8.568. SUBU_S.PH

8.568.1. Purpose

DSP packed operation SUBU_S.PH.

8.568.2. Formats

MIPS

subu_s.ph

Assembly

SUBU_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP2

Decode priority

0

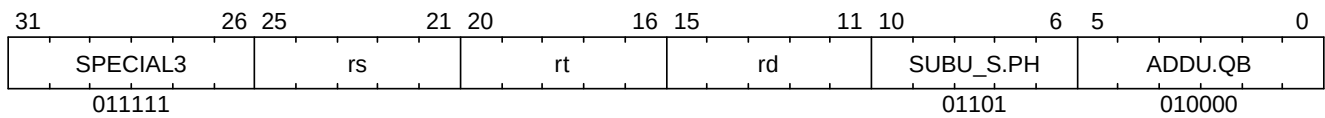


Figure 1583. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subu_s.ph.32

Assembly

SUBU_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

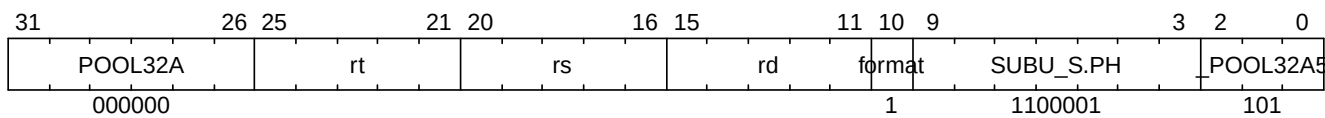


Figure 1584. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subu_s.ph.32

Assembly

SUBU_S.PH {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP2

Decode priority

0

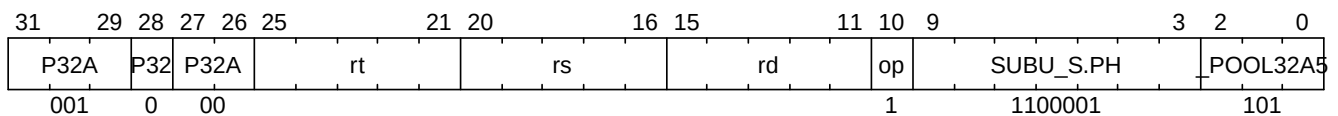


Figure 1585. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.568.3. Operation

```
validate_access_to_dsp2_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 16, 2, subtract=True, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.568.4. Description

SUBU_S.PH is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.568.5. Exceptions

None.

8.569. SUBU_S.QB

8.569.1. Purpose

DSP packed operation SUBU_S.QB.

8.569.2. Formats

MIPS

subu_s.qb

Assembly

SUBU_S.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

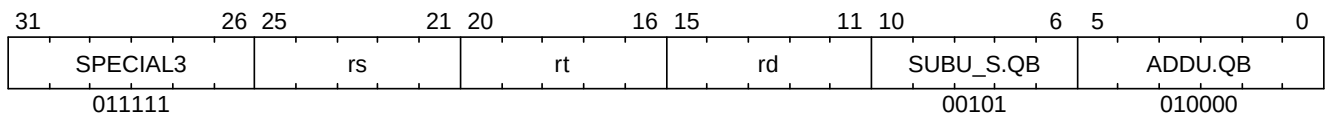


Figure 1586. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subu_s.qb.32

Assembly

SUBU_S.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

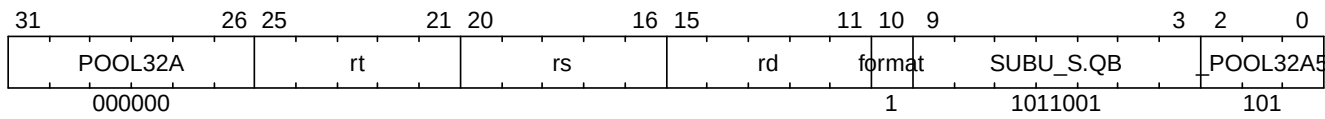


Figure 1587. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subu_s.qb.32

Assembly

SUBU_S.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

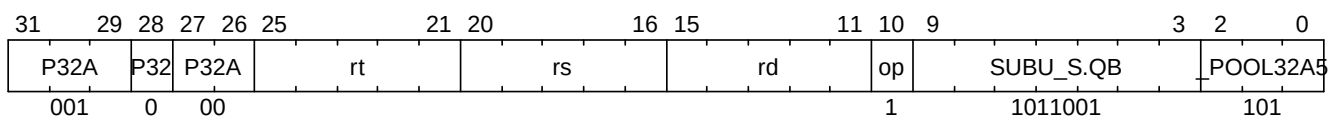


Figure 1588. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.569.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, overflow = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, subtract=True, saturate=True
)
if overflow:
    ctx.DSPControl.Ouflag = ctx.DSPControl.Ouflag | 0x10
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.569.4. Description

SUBU_S.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.569.5. Exceptions

None.

8.570. SUBUH.QB

8.570.1. Purpose

DSP packed operation SUBUH.QB.

8.570.2. Formats

MIPS

subuh.qb

Assembly

SUBUH.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

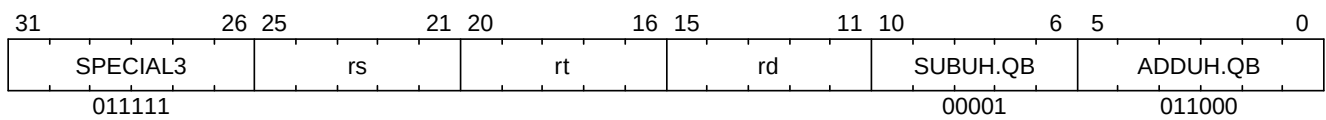


Figure 1589. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subuh.qb.32

Assembly

SUBUH.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

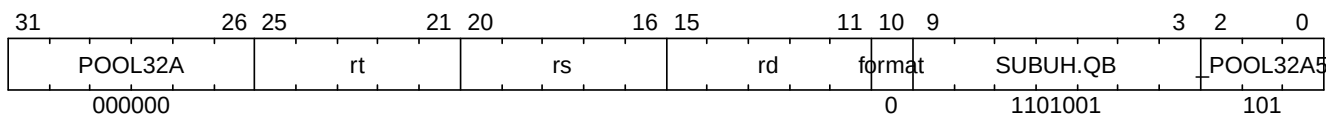


Figure 1590. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subuh.qb.32

Assembly

SUBUH.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

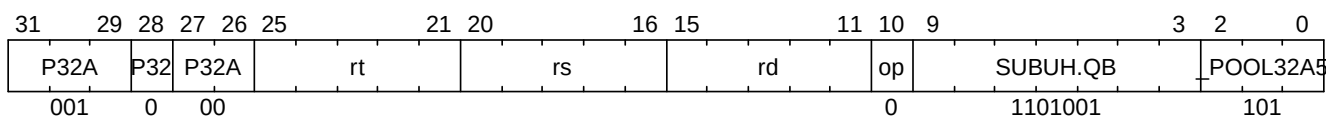


Figure 1591. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.570.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, unused = packed_unsigned_add(
    ctx.GPR[operands.rs], ctx.GPR[operands.rt], 8, 4, subtract=True, half=True
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.570.4. Description

SUBUH.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.570.5. Exceptions

None.

8.571. SUBUH_R.QB

8.571.1. Purpose

DSP packed operation SUBUH_R.QB.

8.571.2. Formats

MIPS

subuh_r.qb

Assembly

SUBUH_R.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

AR >= R2_3_5 && DSP

Decode priority

0

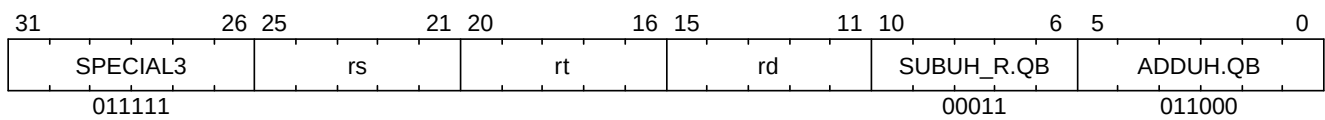


Figure 1592. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

subuh_r.qb.32

Assembly

SUBUH_R.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

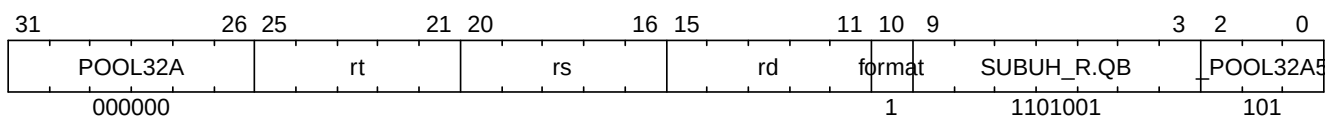


Figure 1593. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

nanoMIPS

subuh_r.qb.32

Assembly

SUBUH_R.QB {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

DSP

Decode priority

0

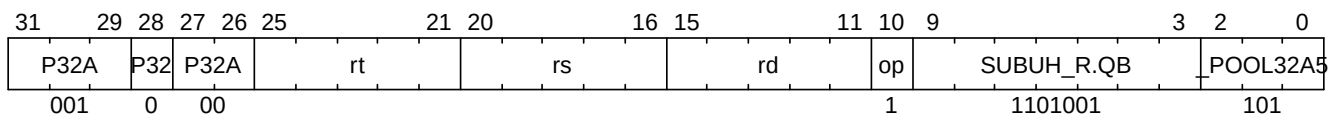


Figure 1594. Encoding (32 bits)

Decode

```
return DspTripleOperands(rd=rd, rs=rs, rt=rt)
```

8.571.3. Operation

```
validate_access_to_dsp_resources(ctx)
value, unused = packed_unsigned_add(
    ctx.GPR[operands.rs],
    ctx.GPR[operands.rt],
    8,
    4,
    subtract=True,
    half=True,
    rounded=True,
)
ctx.GPR[operands.rd] = sign_extend(value, from_nbits=32)
```

8.571.4. Description

SUBUH_R.QB is a MIPS DSP packed arithmetic, comparison, or selection operation.

8.571.5. Exceptions

None.

8.572. SUXC1

8.572.1. Purpose

Indexed SUXC1 access through Coprocessor 1.

8.572.2. Formats

MIPS

suxc1

Assembly

SUXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

AR >= R2_3_5 && AR < R6

Decode priority

0

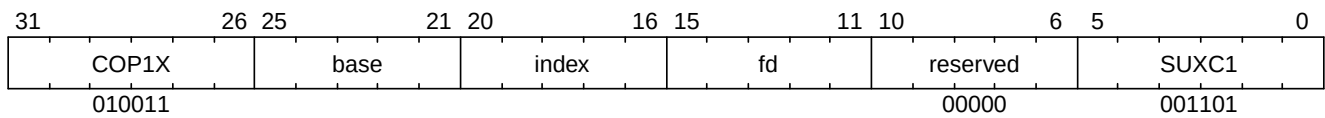


Figure 1595. Encoding (32 bits)

Decode

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

microMIPS

suxc1.32

Assembly

SUXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

MMAR < R6

Decode priority

0

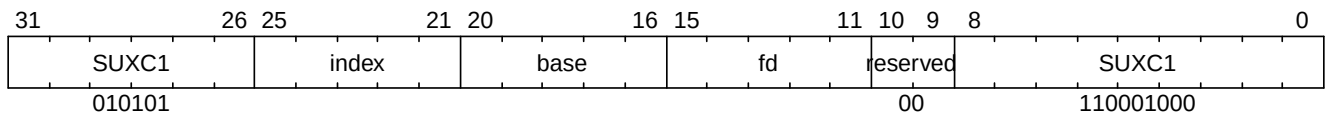


Figure 1596. Encoding (32 bits)

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

8.572.3. Operation

```
validate_access_to_cp1_resources(ctx, "64")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
index = sign_extend(ctx.GPR[operands.index], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, index, "Store")
# SUXC1 follows the same dword-alignment rule as LUXC1.
va &= -8
write_memory_at_va(
    ctx,
    get_fpr(ctx, operands.fd, "64"),
    va,
    nbytes=8,
)
```

8.572.4. Description

SUXC1 performs its CP1 access checks before transferring data.

8.572.5. Exceptions

- **ADEL** (Address Error) — The CP1 load address is not valid.
- **ADES** (Address Error (store)) — A store address is invalid.; The CP1 store address is not valid.
- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.
- **TLBL** (TLB Load) — The CP1 load address could not be translated.
- **TLBS** (TLB Store) — A store address could not be translated.; The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.573. SW

8.573.1. Purpose

Store the low 32 bits of a GPR to memory.

8.573.2. Formats

MIPS

SW

Assembly

SW {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

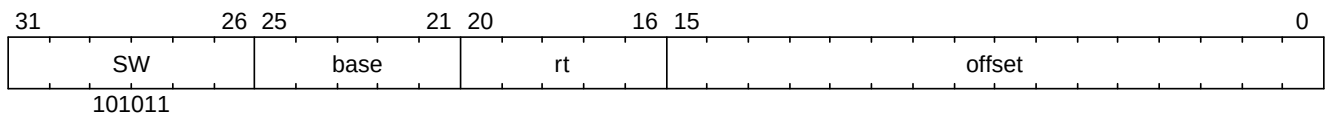


Figure 1597. Encoding (32 bits)

Decode

```
return SwOperands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

microMIPS

sw.32

Assembly

SW {rt:gpr}, {offset:s16}({rs:gpr})

Availability

true

Decode priority

0

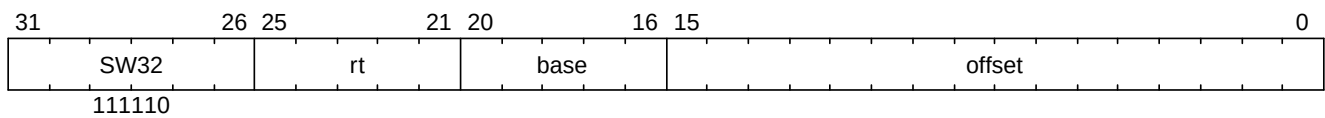


Figure 1598. Encoding (32 bits)

Decode

```
return SwOperands(rt=rt, rs=base, offset=sign_extend(offset, from_nbits=16))
```

sw.sp.16

Assembly

SW {rt:gpr}, {offset:u7}({rs:gpr})

Availability

true

Decode priority

0

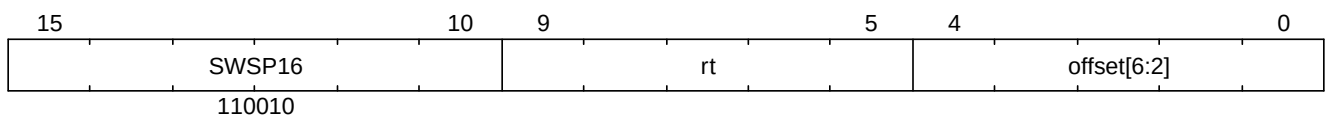


Figure 1599. Encoding (16 bits)

Decode

```
return SwOperands(  
    rt=rt,  
    rs=29,  
    offset=zero_extend(offset, from_nbits=7),  
)
```

sw.16

Assembly

SW {rt:gpr3}, {offset:u6}({rs:gpr3})

Availability

true

Decode priority

0

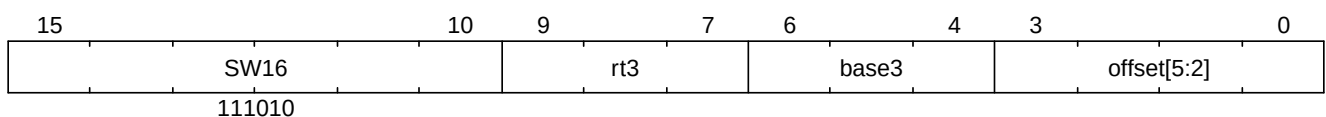


Figure 1600. Encoding (16 bits)

Decode

```
return SwOperands(  
    rt=MICROMIPS_GPR3_STORE.decode(rt3),  
    rs=MICROMIPS_GPR3.decode(base3),  
    offset=zero_extend(offset, from_nbits=6),  
)
```

)

nanoMIPS

sw.16

Assembly

SW {rt:gpr3}, {offset:u6}({rs:gpr3})

Availability

true

Decode priority

0

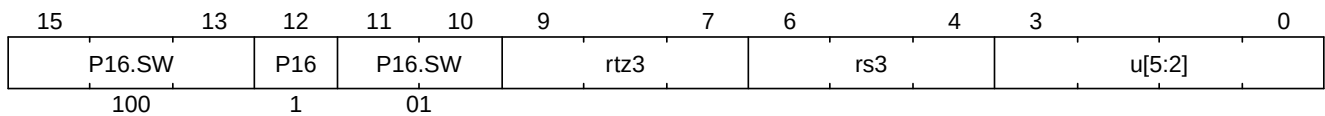


Figure 1601. Encoding (16 bits)

Decode

```
if P64_Mode(ctx, "Store"):
    raise ReservedInstruction("SW[16] is unavailable in P64 mode")
rt = GPR3_STORE.decode(rtz3)
rs = GPR3.decode(rs3)
offset = zero_extend(u, from_nbits=6)
return SwOperands(rt=rt, rs=rs, offset=offset)
```

sw.4x4

Assembly

SW {rt:gpr4}, {offset:u4}({rs:gpr4})

Availability

!(NMS)

Decode priority

0

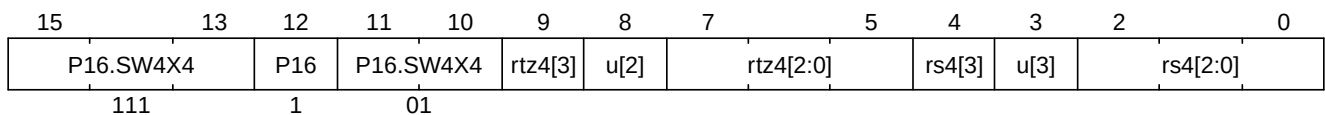


Figure 1602. Encoding (16 bits)

Decode

```
if P64_Mode(ctx, "Store"):
    raise ReservedInstruction("SW[4X4] is unavailable in P64 mode")
rt = GPR4_ZERO.decode(rtz4)
```

```

rs = GPR4.decode(rs4)
offset = zero_extend(u, from_nbits=4)
return SwOperands(rt=rt, rs=rs, offset=offset)

```

sw.gp16

Assembly

SW {rt:gpr3}, {offset:u9}(gp)

Availability

true

Decode priority

0

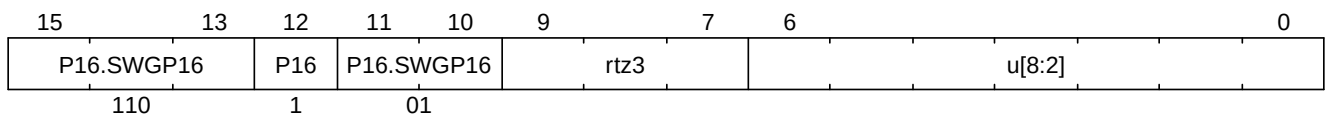


Figure 1603. Encoding (16 bits)

Decode

```

if P64_Mode(ctx, "Store"):
    raise ReservedInstruction("SW[GP16] is unavailable in P64 mode")
rt = GPR3_STORE.decode(rtz3)
offset = zero_extend(u, from_nbits=9)
return SwOperands(rt=rt, rs=28, offset=offset)

```

sw.gp

Assembly

SW {rt:gpr}, {offset:u21}(gp)

Availability

true

Decode priority

0

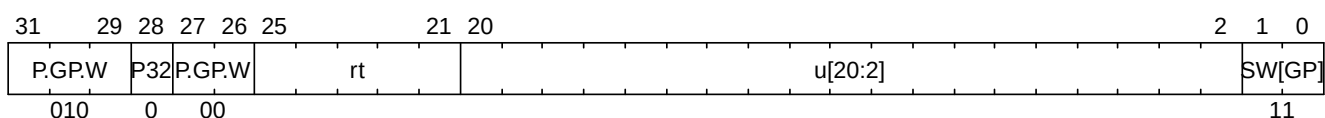


Figure 1604. Encoding (32 bits)

Decode

```

offset = zero_extend(u, from_nbits=21)
return SwOperands(rt=rt, rs=28, offset=offset)

```

sw.s9

Assembly

SW {rt:gpr}, {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

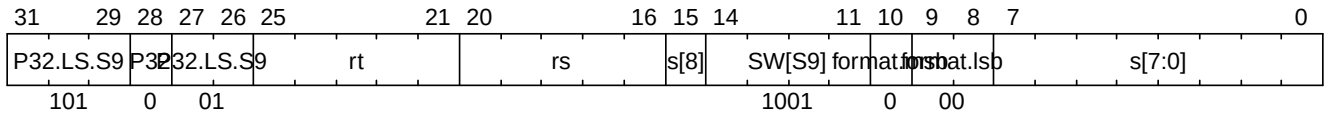


Figure 1605. Encoding (32 bits)

Decode

```
if P64_Mode(ctx, "Store"):
    raise ReservedInstruction("SW[S9] is unavailable in P64 mode")
offset = sign_extend(s, from_nbits=9)
return SwOperands(rt=rt, rs=rs, offset=offset)
```

sw.sp

Assembly

SW {rt:gpr}, {offset:u7}(sp)

Availability

true

Decode priority

0

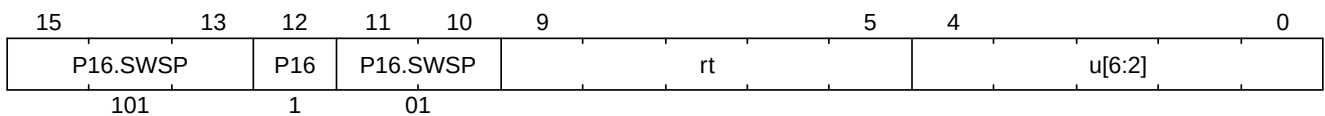


Figure 1606. Encoding (16 bits)

Decode

```
if P64_Mode(ctx, "Store"):
    raise ReservedInstruction("SW[SP] is unavailable in P64 mode")
offset = zero_extend(u, from_nbits=7)
return SwOperands(rt=rt, rs=29, offset=offset)
```

sw.u12

Assembly

SW {rt:gpr}, {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

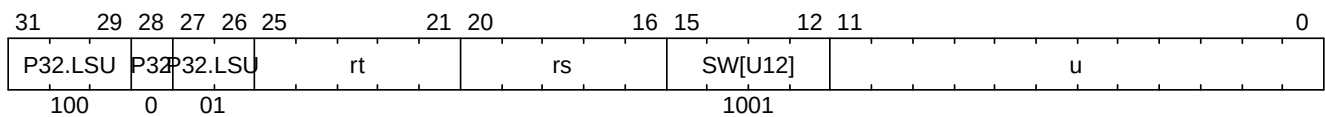


Figure 1607. Encoding (32 bits)

Decode

```
offset = zero_extend(u, from_nbits=12)
return SwOperands(rt=rt, rs=rs, offset=offset)
```

8.573.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
data = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
write_memory_at_va(ctx, data, va, nbytes=4)
```

8.573.4. Description

SW adds the encoded byte offset to the base GPR, takes the low 32 bits of the source GPR, and writes four bytes through the active memory backend.

8.573.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — SW[16] is unavailable in P64 mode; SW[4X4] is unavailable in P64 mode; SW[GP16] is unavailable in P64 mode; SW[S9] is unavailable in P64 mode; SW[SP] is unavailable in P64 mode; SW[4X4] is reserved when CP0 Config5.NMS is set.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.574. SWC1

8.574.1. Purpose

SWC1 a CP1 register to or from memory.

8.574.2. Formats

MIPS

swc1

Assembly

SWC1 {ft:fpr}, {offset:s16}({base:gpr})

Availability

true

Decode priority

0

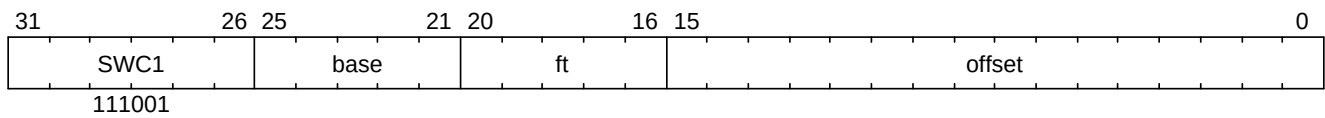


Figure 1608. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(  
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)  
)
```

microMIPS

swc1.32

Assembly

SWC1 {ft:fpr}, {offset:s16}({base:gpr})

Availability

true

Decode priority

0

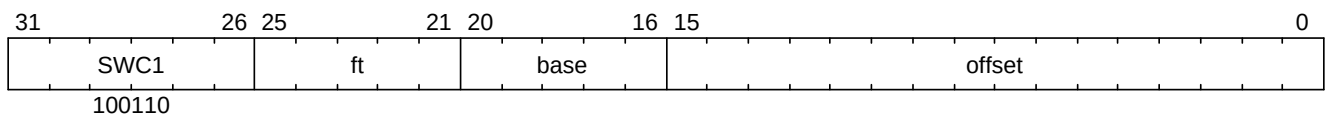


Figure 1609. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(
    ft=ft, base=base, offset=sign_extend(offset, from_nbits=16)
)
```

nanoMIPS

swc1.gp

Assembly

SWC1 {ft:fpr}, ...

Availability

true

Decode priority

0

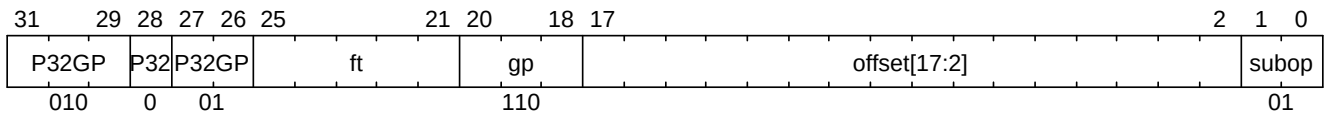


Figure 1610. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(
    ft=ft, base=28, offset=zero_extend(offset, from_nbits=16) << 2
)
```

swc1.s9

Assembly

```
SWC1 {ft:fpr}, ...
```

Availability

true

Decode priority

0

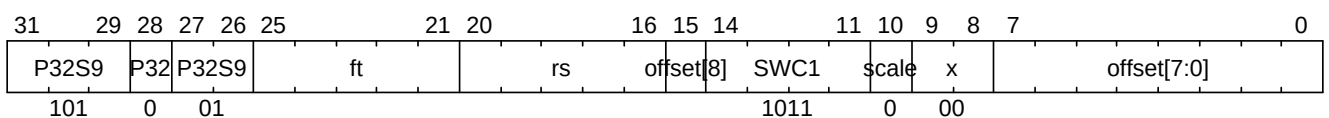


Figure 1611. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=sign_extend(offset, from_nbits=9))
```

swc1.u12

Assembly

SWC1 {ft:fpr}, ...

Availability

true

Decode priority

0

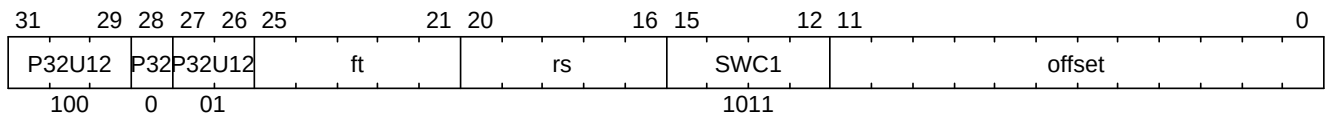


Figure 1612. Encoding (32 bits)

Decode

```
return Cop1MemoryOperands(ft=ft, base=rs, offset=zero_extend(offset, from_nbits=12))
```

8.574.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
write_memory_at_va(ctx, get_fpr(ctx, operands.ft, "32"), va, nbytes=4)
```

8.574.4. Description

SWC1 performs CP1 access checks before its memory access.

8.574.5. Exceptions

- **ADEL** (Address Error) — The CP1 load address is not valid.
- **ADES** (Address Error (store)) — A store address is invalid.; The CP1 store address is not valid.
- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the required FPU resource is unavailable.
- **TLBL** (TLB Load) — The CP1 load address could not be translated.

- **TLBS** (TLB Store) — A store address could not be translated.; The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.575. SWC2

8.575.1. Purpose

Store a word from a CP2 register to memory.

8.575.2. Formats

microMIPS

swc2.32

Assembly

SWC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR < R6

Decode priority

0

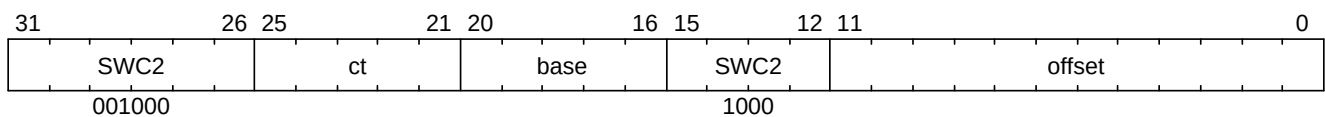


Figure 1613. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

swc2.r6.32

Assembly

SWC2 {ct:gpr}, {offset:s12}({base:gpr})

Availability

MMAR >= R6

Decode priority

0

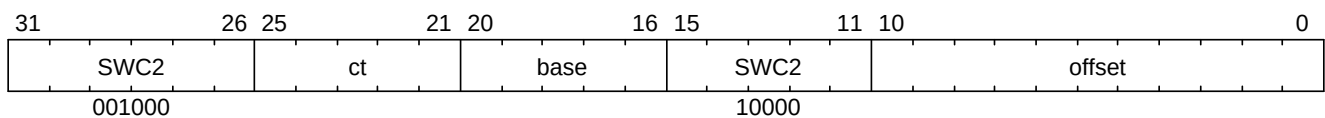


Figure 1614. Encoding (32 bits)

Decode

```
return Cop2MemoryOperands(ct=ct, base=base, offset=offset)
```

8.575.3. Operation

```
if not ctx.C0.Status.CU2:
    raise CoprocessorUnusable(2)
if not ctx.C0.Config1.C2:
    raise ReservedInstruction("Coprocessor 2 is not implemented")
raise ReservedInstruction("SWC2 execution is not implemented")
```

8.575.4. Description

SWC2 performs the CP2 access check before any CP2 operation is attempted.

8.575.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — Coprocessor 2 is not implemented; SWC2 execution is not implemented

8.576. SWE

8.576.1. Purpose

Store a word using EVA addressing.

8.576.2. Formats

nanoMIPS

swe.32

Assembly

SWE {offset:s9}({rs:gpr})

Availability

EVA

Decode priority

0

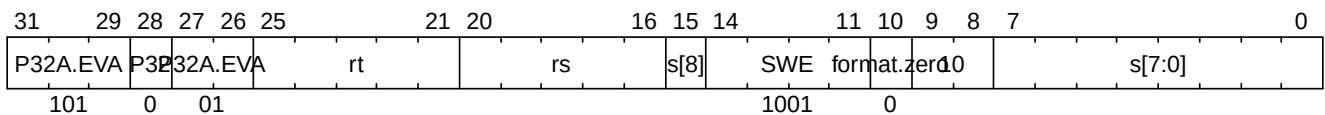


Figure 1615. Encoding (32 bits)

Decode

```
return EvaMemoryOperands(rt, rs, sign_extend(s, from_nbits=9))
```

8.576.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("EVA memory instruction requires Config5.EVA")
if not IsCoprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store", eva=True)
data = zero_extend(ctx.GPR[operands.rt], from_nbits=32)
write_memory_at_va(ctx, data, va, nbytes=4, eva=True)
```

8.576.4. Description

This optional privileged form uses EVA user-mode address translation.

8.576.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — EVA memory instruction requires Config5.EVA
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.577. SWL

8.577.1. Purpose

Store the left portion of a GPR to an unaligned word.

8.577.2. Formats

MIPS

swl

Assembly

SWL {offset:s16}({rs:gpr})

Availability

AR < R6

Decode priority

0

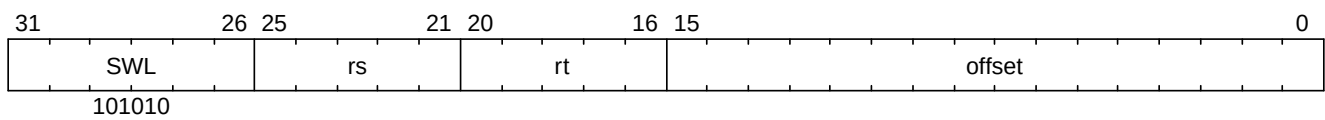


Figure 1616. Encoding (32 bits)

Decode

```
return LegacyUnalignedOperands(  
    rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=16)  
)
```

microMIPS

swl.32

Assembly

SWL {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6

Decode priority

0

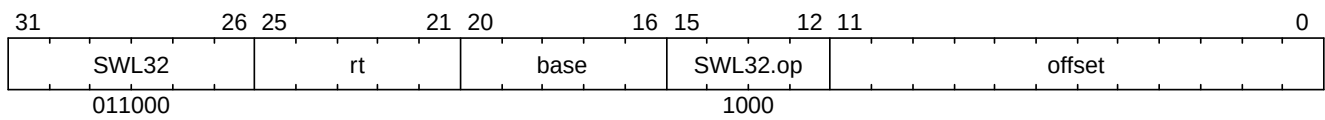


Figure 1617. Encoding (32 bits)

```

del ctx, exec
return LegacyUnalignedOperands(
    rt=rt, rs=base, offset=sign_extend(offset, from_nbits=12)
)

```

8.577.3. Operation

```

base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
word = read_memory_at_va(
    ctx,
    va,
    nbytes=4,
    unaligned_ok=True,
    partial_ok=True,
    access_type="Store",
)
byte = va & 3
order = memory_endianness(ctx)
shift = (3 - byte) * 8 if order == "little" else byte * 8
store_mask = 0xFFFF_FFFF >> shift
register = ctx.GPR[operands.rt] & 0xFFFF_FFFF
merged = (word & ~store_mask) | ((register >> shift) & store_mask)
write_memory_at_va(
    ctx,
    merged,
    va,
    nbytes=4,
    unaligned_ok=True,
    partial_ok=True,
)

```

8.577.4. Description

The naturally aligned word containing the effective byte address is merged according to the active memory endianness. This legacy instruction is unavailable in Release 6.

8.577.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.578. SWM

8.578.1. Purpose

Store multiple words from sequential GPRs.

8.578.2. Formats

microMIPS

`swm.16`

Assembly

`SWM {offset:u4}(sp)`

Availability

MMAR < R6

Decode priority

0

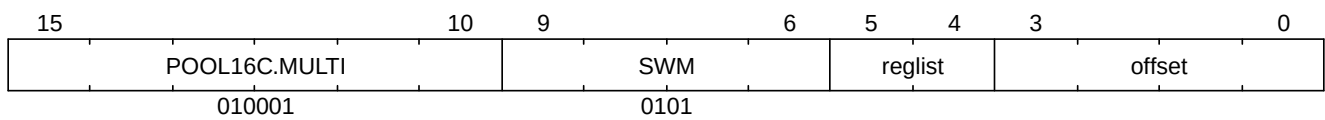


Figure 1618. Encoding (16 bits)

Decode

```
return MicroMultipleOperands(_MICRO_LWM16[reglist], 29, zero_extend(offset, 4) << 2)
```

`swm.r6.16`

Assembly

`SWM {offset:u4}(sp)`

Availability

MMAR >= R6

Decode priority

0

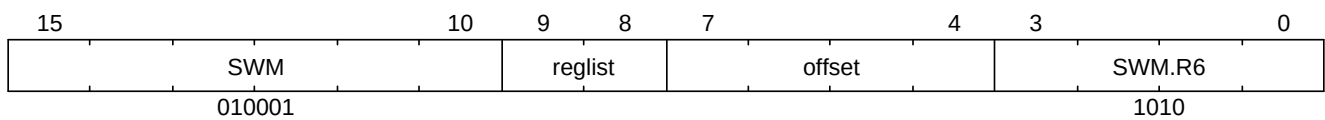


Figure 1619. Encoding (16 bits)

Decode

```
return MicroMultipleOperands(_MICRO_LWM16[reglist], 29, zero_extend(offset, 4) << 2)
```

swm.32

Assembly

SWM {offset:s12}({rs:gpr})

Availability

true

Decode priority

0

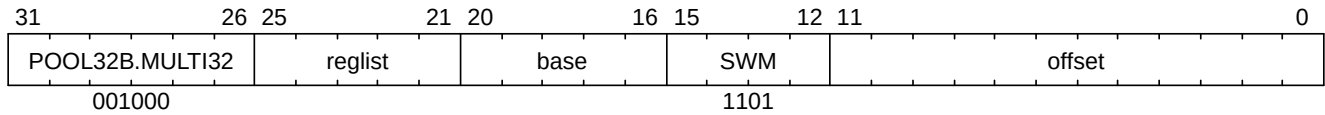


Figure 1620. Encoding (32 bits)

Decode

```
if reglist >= len(_MICRO_LWM32) or _MICRO_LWM32[reglist] is None:
    raise ReservedInstruction("reserved microMIPS multiple register list")
return MicroMultipleOperands(_MICRO_LWM32[reglist], base, sign_extend(offset, 12))
```

nanoMIPS

swm.32

Assembly

SWM {offset:s9}({rs:gpr}), {count:u3}

Availability

!(NMS)

Decode priority

0

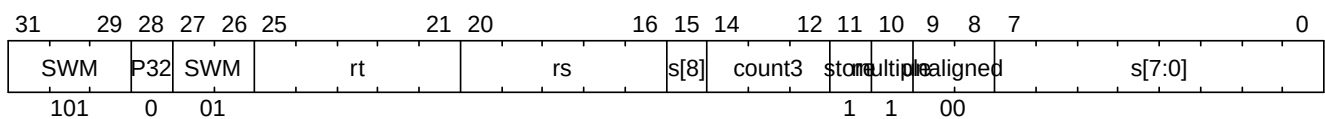


Figure 1621. Encoding (32 bits)

Decode

```
return MultipleMemoryOperands(
    rt, rs, sign_extend(s, from_nbits=9), 8 if count3 == 0 else count3
)
```

8.578.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
registers = store_multiple_registers(operands)
```

```
for index in multiple_transfer_order(operands):
    register = registers[index]
    va = effective_address(ctx, base, int(operands.offset) + index * 4, "Store")
    data = zero_extend(ctx.GPR[register], from_nbits=32)
    write_memory_at_va(ctx, data, va, nbytes=4)
```

8.578.4. Description

This full-nano form is unavailable in the nanoMIPS subset.

8.578.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — reserved microMIPS multiple register list
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.579. SWP

8.579.1. Purpose

SWP microMIPS register pair memory operation.

8.579.2. Formats

microMIPS

swp.32

Assembly

SWP {rd:gpr}, {offset:s12}({rs:gpr})

Availability

true

Decode priority

0

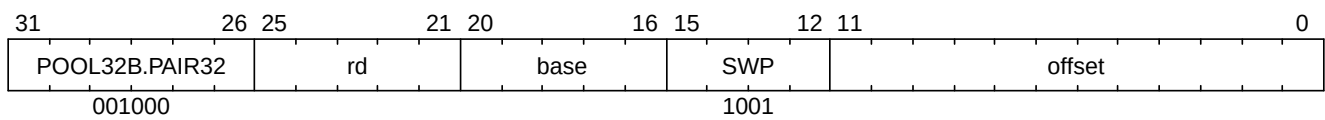


Figure 1622. Encoding (32 bits)

Decode

```
return MicroMultipleOperands((rd, rd + 1), base, sign_extend(offset, 12))
```

8.579.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
for index, register in enumerate(operands regs):
    va = effective_address(ctx, base, operands.offset + index * 4, "Store")
    write_memory_at_va(
        ctx, zero_extend(ctx.GPR[register], from_nbits=32), va, nbytes=4
    )
```

8.579.4. Description

microMIPS register-pair memory transfer.

8.579.5. Exceptions

- ADES (Address Error (store)) — A store address is invalid.
- DBE (Bus Error) — A data-memory access reported a bus error.

- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.580. SWPC

8.580.1. Purpose

Store a word to a PC-relative address.

8.580.2. Formats

nanoMIPS

swpc.48

Assembly

SWPC {rt:gpr}, {offset:s32}

Availability

!(NMS)

Decode priority

0

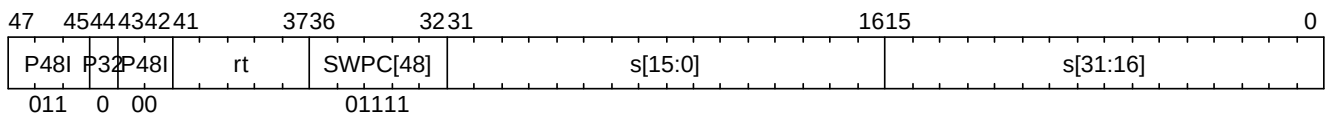


Figure 1623. Encoding (48 bits)

Decode

```
return PcRelativeMemoryOperands(rt, sign_extend(s, from_nbits=32))
```

8.580.3. Operation

```
base = (  
    ctx.next_pc  
    if P64_Mode(ctx, "Store")  
    else sign_extend(ctx.next_pc, from_nbits=32)  
)  
address = effective_address(ctx, base, operands.offset, "Store")  
data = zero_extend(ctx.GPR[operands.rt], from_nbits=32)  
write_memory_at_va(ctx, data, address, nbytes=4)
```

8.580.4. Description

The signed 32-bit displacement is relative to the following instruction.

8.580.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.

- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.581. SWR

8.581.1. Purpose

Store the right portion of a GPR to an unaligned word.

8.581.2. Formats

MIPS

SWR

Assembly

SWR {offset:s16}({rs:gpr})

Availability

AR < R6

Decode priority

0

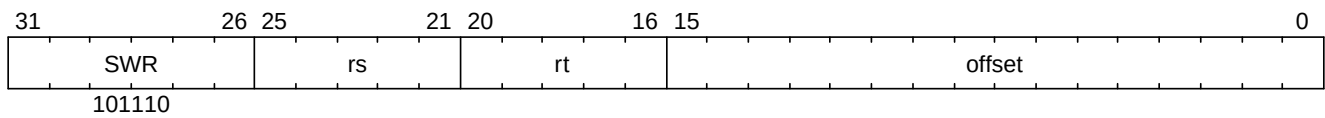


Figure 1624. Encoding (32 bits)

Decode

```
return LegacyUnalignedOperands(  
    rt=rt, rs=rs, offset=sign_extend(offset, from_nbits=16)  
)
```

microMIPS

swr.32

Assembly

SWR {rt:gpr}, {offset:s12}({rs:gpr})

Availability

MMAR < R6

Decode priority

0

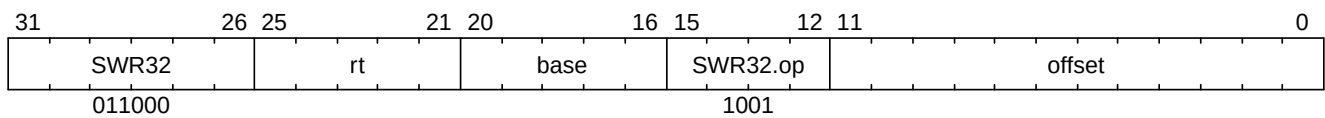


Figure 1625. Encoding (32 bits)

```

del ctx, exec
return LegacyUnalignedOperands(
    rt=rt, rs=base, offset=sign_extend(offset, from_nbits=12)
)

```

8.581.3. Operation

```

base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
word = read_memory_at_va(
    ctx,
    va,
    nbytes=4,
    unaligned_ok=True,
    partial_ok=True,
    access_type="Store",
)
byte = va & 3
order = memory_endianness(ctx)
shift = byte * 8 if order == "little" else (3 - byte) * 8
store_mask = (0xFFFF_FFFF << shift) & 0xFFFF_FFFF
register = ctx.GPR[operands.rt] & 0xFFFF_FFFF
merged = (word & ~store_mask) | ((register << shift) & store_mask)
write_memory_at_va(
    ctx,
    merged,
    va,
    nbytes=4,
    unaligned_ok=True,
    partial_ok=True,
)

```

8.581.4. Description

The naturally aligned word containing the effective byte address is merged according to the active memory endianness. This legacy instruction is unavailable in Release 6.

8.581.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.582. SWX

8.582.1. Purpose

Store a word using a register-indexed address.

8.582.2. Formats

nanoMIPS

swx.32

Assembly

SWX {rs:gpr}({rt:gpr})

Availability

!(NMS)

Decode priority

0

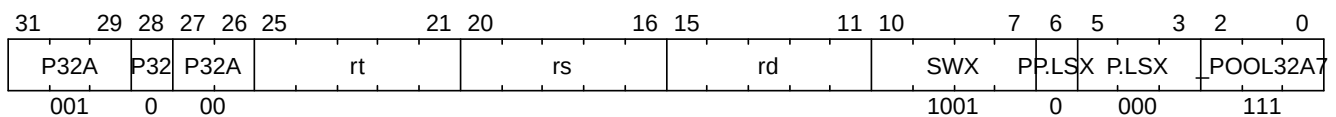


Figure 1626. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.582.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
va = effective_address(ctx, base, index, "Store")
data = zero_extend(ctx.GPR[operands.rd], from_nbits=32)
write_memory_at_va(ctx, data, va, nbytes=4)
```

8.582.4. Description

The effective address is formed from two full-width GPR values.

8.582.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.583. SWXC1

8.583.1. Purpose

Indexed SWXC1 access through Coprocessor 1.

8.583.2. Formats

MIPS

swxc1

Assembly

SWXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

AR >= R2_3_5 && AR < R6

Decode priority

0

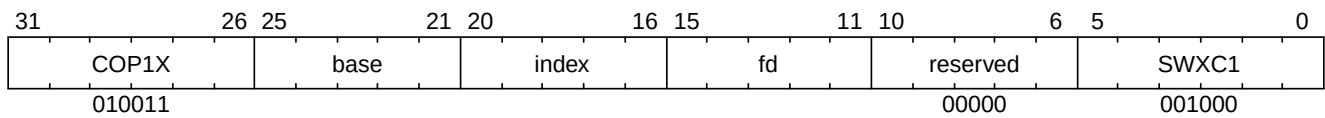


Figure 1627. Encoding (32 bits)

Decode

```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

microMIPS

swxc1.32

Assembly

SWXC1 {fd:fpr}, {index:gpr}({base:gpr})

Availability

MMAR < R6

Decode priority

0

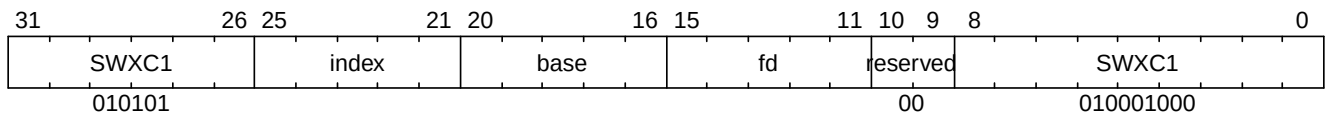


Figure 1628. Encoding (32 bits)


```
return Cp1IndexedOperands(fd=fd, base=base, index=index)
```

8.583.3. Operation

```
validate_access_to_cp1_resources(ctx, "32")
base = sign_extend(ctx.GPR[operands.base], from_nbits=ctx.GPR.width)
index = sign_extend(ctx.GPR[operands.index], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, index, "Store")
write_memory_at_va(ctx, get_fpr(ctx, operands.fd, "32"), va, nbytes=4)
```

8.583.4. Description

SWXC1 performs its CP1 access checks before transferring data.

8.583.5. Exceptions

- **ADEL** (Address Error) — The CP1 load address is not valid.
- **ADES** (Address Error (store)) — A store address is invalid.; The CP1 store address is not valid.
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.; Coprocessor 1 access is disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.; The CP1 data access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; Coprocessor 1 is not implemented or the requested FPU resource is unavailable.
- **TLBL** (TLB Load) — The CP1 load address could not be translated.
- **TLBS** (TLB Store) — A store address could not be translated.; The CP1 store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.; The CP1 data access matched a watchpoint.

8.584. SWXS

8.584.1. Purpose

Store a word using a scaled register-indexed address.

8.584.2. Formats

nanoMIPS

swxs.32

Assembly

SWXS {rs:gpr}({rt:gpr})

Availability

!(NMS)

Decode priority

0

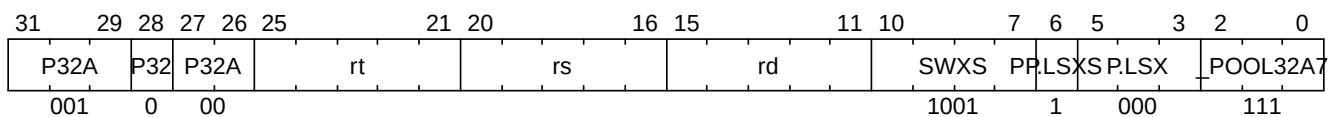


Figure 1629. Encoding (32 bits)

Decode

```
return IndexedMemoryOperands(rd, rs, rt)
```

8.584.3. Operation

```
base = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
index = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)) << 2
va = effective_address(ctx, base, index, "Store")
data = zero_extend(ctx.GPR[operands.rd], from_nbits=32)
write_memory_at_va(ctx, data, va, nbytes=4)
```

8.584.4. Description

The effective address is formed from two full-width GPR values.

8.584.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.585. SYNC

8.585.1. Purpose

Impose the memory ordering or completion constraints selected by the five-bit synchronization type.

8.585.2. Formats

MIPS

`sync`

Assembly

`SYNC {stype:u5}`

Availability

true

Decode priority

0

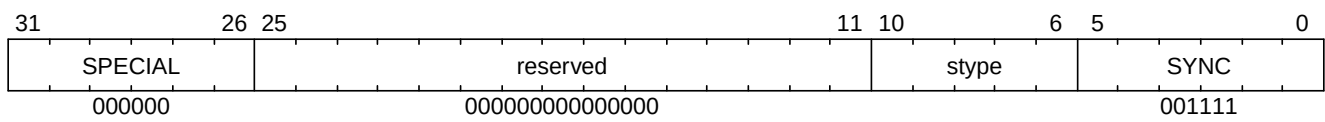


Figure 1630. Encoding (32 bits)

Decode

```
return SyncOperands(stype=zero_extend(stype, from_nbits=5))
```

microMIPS

`sync.32`

Assembly

`SYNC {stype:u5}`

Availability

true

Decode priority

0

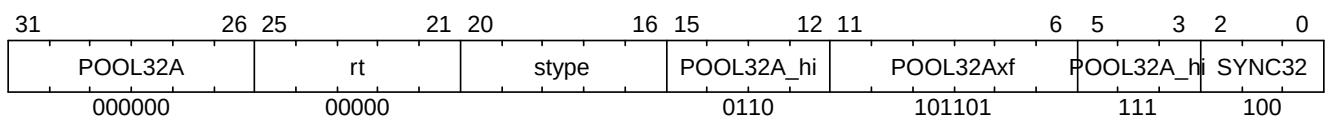


Figure 1631. Encoding (32 bits)

Decode

```
return SyncOperands(stype=zero_extend(stype, from_nbits=5))
```

nanoMIPS

sync.32

Assembly

SYNC {stype:u5}

Availability

true

Decode priority

1

Overrides lower-priority formats

- SLL (**sll.32**)

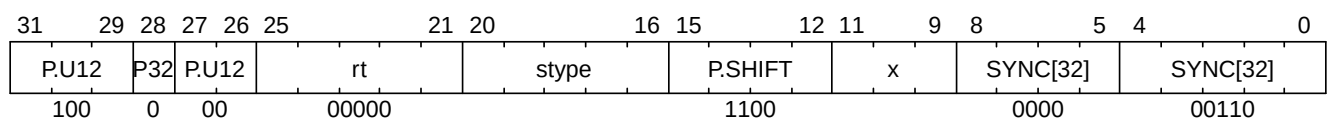


Figure 1632. Encoding (32 bits)

Decode

```
stype = zero_extend(stype, from_nbits=5)
return SyncOperands(stype=stype)
```

8.585.3. Operation

```
sync_memory_access(ctx, zero_extend(operands.stype, from_nbits=5))
```

8.585.4. Description

SYNC delegates the implementation-selected barrier action to the active memory-system backend. A strictly sequential backend needs no additional action. The **stype** value remains visible so weakly ordered and multi-core simulators can implement each barrier class.

8.585.5. Exceptions

None.

8.586. SYNCI

8.586.1. Purpose

Synchronize instruction visibility with data writes.

8.586.2. Formats

MIPS

`synci`

Assembly

`SYNCI {offset:s16}({rs:gpr})`

Availability

AR >= R2_3_5

Decode priority

0

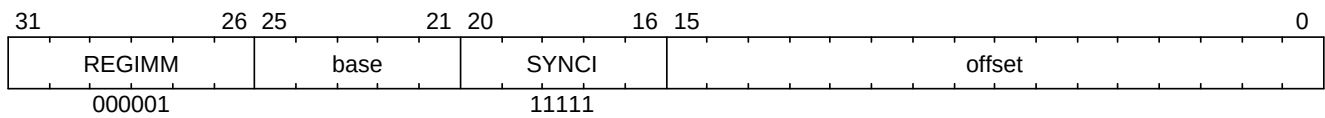


Figure 1633. Encoding (32 bits)

Decode

```
return Synci0perands(rs=base, offset=sign_extend(offset, from_nbits=16), eva=False)
```

microMIPS

`synci.32`

Assembly

`SYNCI {offset:s16}({rs:gpr})`

Availability

MMAR < R6

Decode priority

0

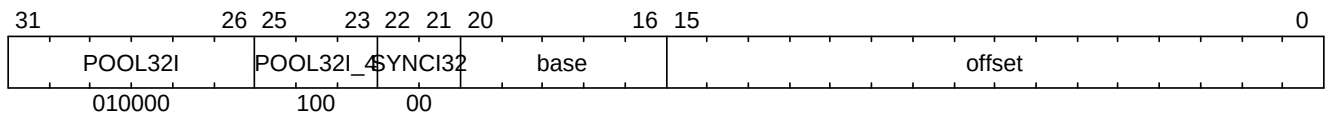


Figure 1634. Encoding (32 bits)

Decode

```
return Synci0perands(rs=base, offset=sign_extend(offset, from_nbits=16), eva=False)
```

synci.r6.32

Assembly

SYNCI {offset:s16}({rs:gpr})

Availability

MMAR >= R6

Decode priority

0

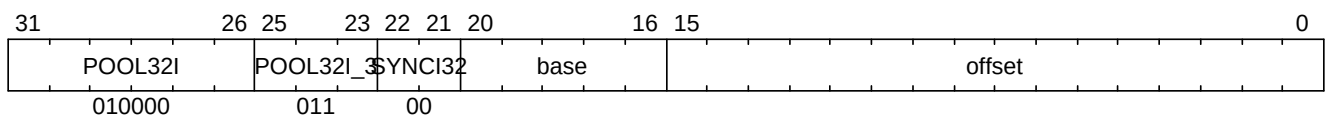


Figure 1635. Encoding (32 bits)

Decode

```
return Synci0perands(rs=base, offset=sign_extend(offset, from_nbits=16), eva=False)
```

nanoMIPS

synci.s9

Assembly

SYNCI {offset:s9}({rs:gpr})

Availability

true

Decode priority

0

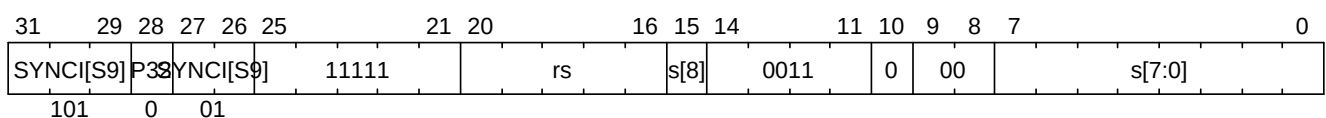


Figure 1636. Encoding (32 bits)

Decode

```
return Synci0perands(rs=rs, offset=sign_extend(s, from_nbits=9), eva=False)
```

synci.u12

Assembly

SYNCI {offset:u12}({rs:gpr})

Availability

true

Decode priority

0

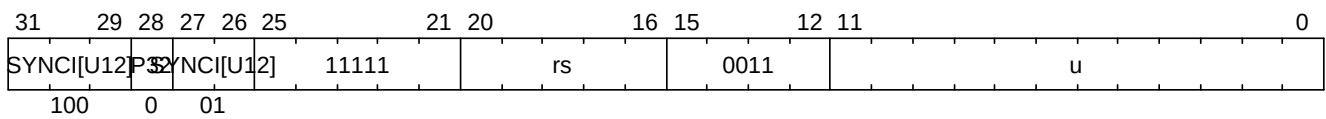


Figure 1637. Encoding (32 bits)

Decode

```
return SynciOperands(rs=rs, offset=zero_extend(u, from_nbits=12), eva=False)
```

8.586.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
address = effective_address(ctx, base, operands.offset, "Cacheop", eva=False)
synci(ctx, address, eva=False)
```

8.586.4. Description

SYNCI requests instruction-cache synchronization through the memory backend.

8.586.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.587. SYNCIE

8.587.1. Purpose

Synchronize instruction visibility using an EVA effective address.

8.587.2. Formats

nanoMIPS

`synci.eva`

Assembly

`SYNCIE {offset:s9}({rs:gpr})`

Availability

EVA

Decode priority

0

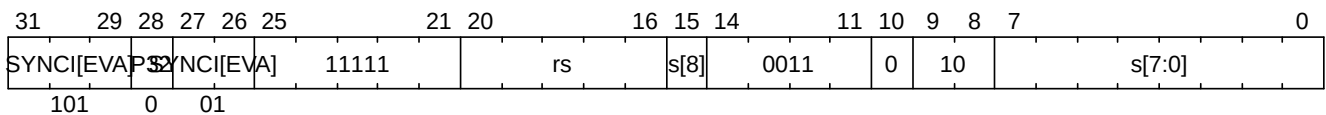


Figure 1638. Encoding (32 bits)

Decode

```
return Synci0operands(rs=rs, offset=sign_extend(s, from_nbits=9), eva=True)
```

8.587.3. Operation

```
if not ctx.C0.Config5.EVA:
    raise ReservedInstruction("SYNCIE requires EVA")
if not IsCoprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
address = effective_address(ctx, base, operands.offset, "Cacheop", eva=True)
synci(ctx, address, eva=True)
```

8.587.4. Description

SYNCIE performs instruction-cache synchronization through the EVA translation path.

8.587.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.

- **BAT** (BAT Hook) — AVP KernelBAT hook matched the virtual address
- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — SYNCIE requires EVA
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.588. SYSCALL

8.588.1. Purpose

Unconditionally raise a system-call exception.

8.588.2. Formats

MIPS

syscall

Assembly

SYSCALL {code:u20}

Availability

true

Decode priority

0

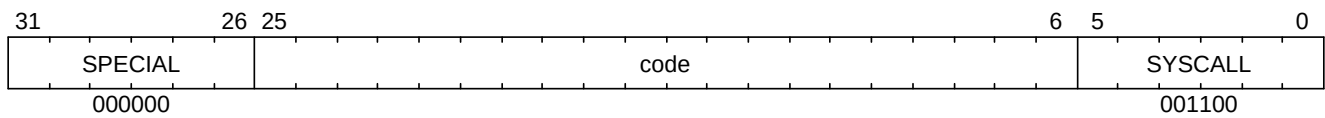


Figure 1639. Encoding (32 bits)

Decode

```
return Syscall0operands(code=zero_extend(code, from_nbits=20))
```

microMIPS

syscall.32

Assembly

SYSCALL {code:u10}

Availability

true

Decode priority

0

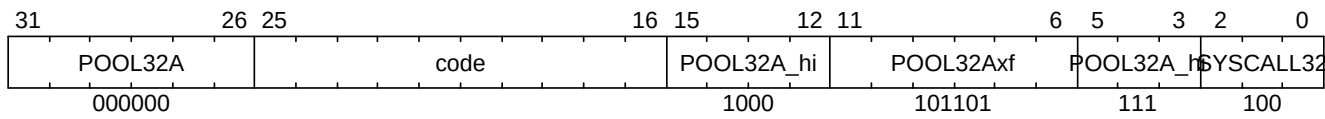


Figure 1640. Encoding (32 bits)

Decode

```
return Syscall0perands(code=zero_extend(code, from_nbits=10))
```

nanoMIPS

syscall.16

Assembly

```
SYSCALL {code:u2}
```

Availability

true

Decode priority

0

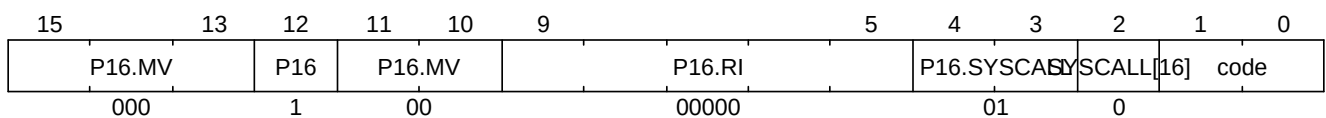


Figure 1641. Encoding (16 bits)

Decode

```
code = zero_extend(code, from_nbits=2)
return Syscall0perands(code=code)
```

syscall.32

Assembly

```
SYSCALL {code:u18}
```

Availability

true

Decode priority

0

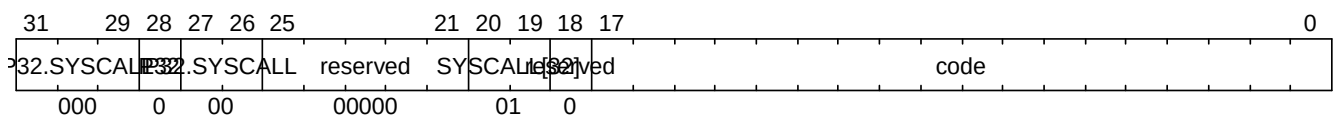


Figure 1642. Encoding (32 bits)

Decode

```
code = zero_extend(code, from_nbits=18)
return Syscall0perands(code=code)
```

8.588.3. Operation

```
raise SystemCallException("SYSCALL instruction")
```

8.588.4. Description

SYSCALL carries an implementation-visible code in either a compact two-bit or full eighteen-bit field. The architectural operation always raises the System Call exception.

8.588.5. Exceptions

- **SYSCALL** (System Call) — SYSCALL instruction

8.589. TEQ

8.589.1. Purpose

Raise a Trap exception when two GPR values are equal.

8.589.2. Formats

MIPS

teq

Assembly

TEQ {rs:gpr}, {rt:gpr}, {code:u10}

Availability

true

Decode priority

0

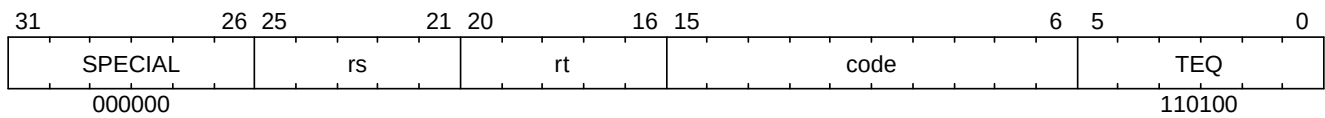


Figure 1643. Encoding (32 bits)

Decode

```
return Teq0operands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

microMIPS

teq.32

Assembly

TEQ {rs:gpr}, {rt:gpr}, {code:u4}

Availability

true

Decode priority

0

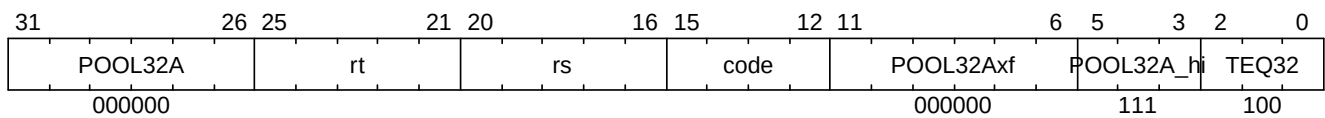


Figure 1644. Encoding (32 bits)

Decode

```
return Teq0operands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=4))
```

nanoMIPS

teq.32

Assembly

TEQ {rs:gpr}, {rt:gpr}, {code:u5}

Availability

!(NMS)

Decode priority

0

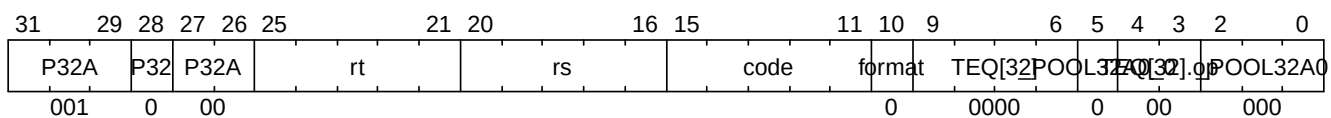


Figure 1645. Encoding (32 bits)

Decode

```
return Teq0operands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=5))
```

8.589.3. Operation

```
if ctx.GPR[operands.rs] == ctx.GPR[operands.rt]:  
    raise TrapException("TEQ condition matched")
```

8.589.4. Description

TEQ compares the complete active-width values of **rs** and **rt**. The five-bit code is carried for exception diagnostics but does not affect the comparison. TEQ is unavailable in the nanoMIPS subset.

8.589.5. Exceptions

- **RI** (Reserved Instruction) — TEQ is reserved when CP0 Config5.NMS is set.
- **TRAP** (Trap) — TEQ condition matched

8.590. TEQI

8.590.1. Purpose

Raise a Trap exception when the TEQI comparison matches.

8.590.2. Formats

MIPS

`teqi`

Assembly

`TEQI {imm:s16}`

Availability

AR < R6

Decode priority

0

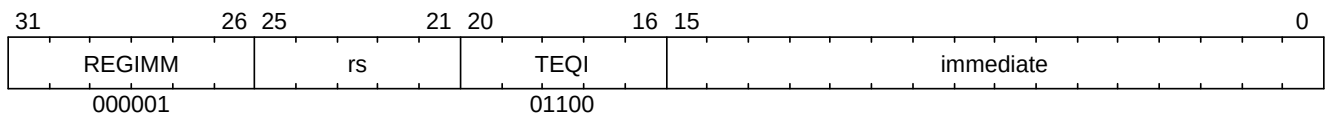


Figure 1646. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

microMIPS

`teqi.32`

Assembly

`TEQI {rs:gpr}, {imm:s16}`

Availability

MMAR < R6

Decode priority

0

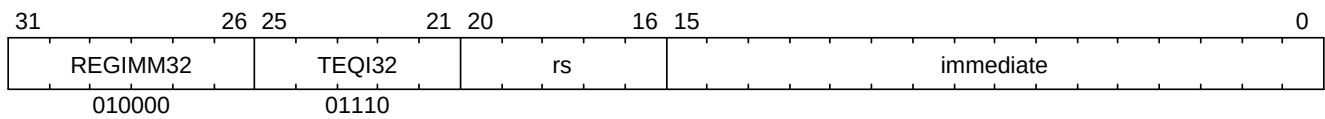


Figure 1647. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

8.590.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(operands.imm, from_nbits=ctx.GPR.width))
if lhs == rhs:
    raise TrapException("immediate trap condition matched")
```

8.590.4. Description

The 16-bit immediate is sign-extended before comparison. Unsigned forms reinterpret that sign-extended word as unsigned. Immediate trap encodings are unavailable in Release 6.

8.590.5. Exceptions

- **TRAP** (Trap) — immediate trap condition matched

8.591. TGE

8.591.1. Purpose

Raise a Trap exception when signed rs is greater than or equal to signed rt.

8.591.2. Formats

MIPS

tge

Assembly

TGE {rs:gpr}, {rt:gpr}, {code:u10}

Availability

true

Decode priority

0

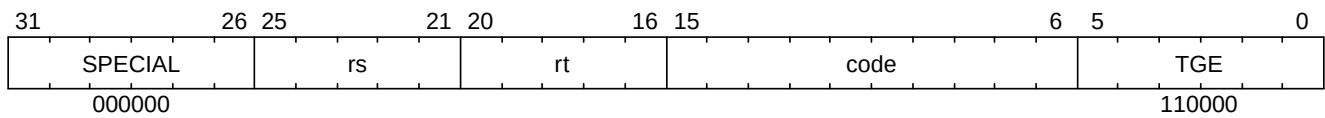


Figure 1648. Encoding (32 bits)

Decode

```
return Tge0operands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

microMIPS

tge.32

Assembly

TGE {rs:gpr}, {rt:gpr}, {code:u4}

Availability

true

Decode priority

0

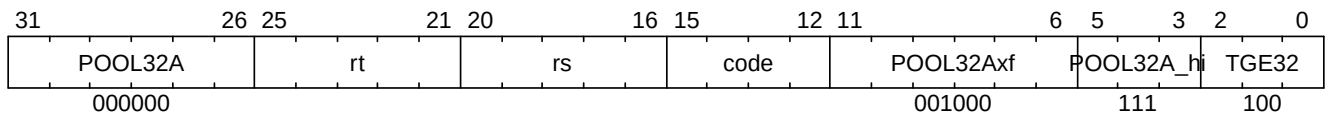


Figure 1649. Encoding (32 bits)

```
return TgeOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

8.591.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
if lhs >= rhs:
    raise TrapException("TGE condition matched")
```

8.591.4. Description

TGE compares the complete active-width values of *rs* and *rt* as signed integers. The ten-bit code is carried for exception diagnostics but does not affect the comparison.

8.591.5. Exceptions

- **TRAP** (Trap) — TGE condition matched

8.592. TGEI

8.592.1. Purpose

Raise a Trap exception when the TGEI comparison matches.

8.592.2. Formats

MIPS

`tgei`

Assembly

`TGEI {imm:s16}`

Availability

AR < R6

Decode priority

0

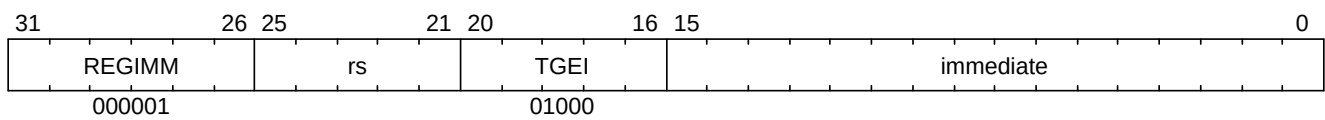


Figure 1650. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

microMIPS

`tgei.32`

Assembly

`TGEI {rs:gpr}, {imm:s16}`

Availability

MMAR < R6

Decode priority

0

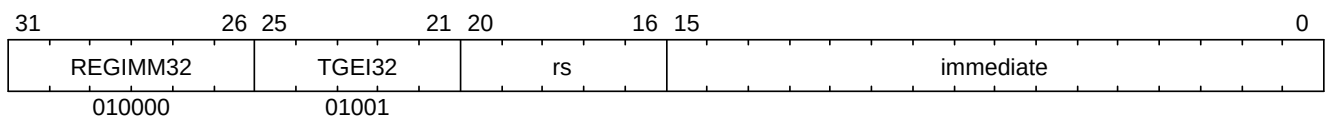


Figure 1651. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

8.592.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(operands.imm, from_nbits=ctx.GPR.width))
if lhs >= rhs:
    raise TrapException("immediate trap condition matched")
```

8.592.4. Description

The 16-bit immediate is sign-extended before comparison. Unsigned forms reinterpret that sign-extended word as unsigned. Immediate trap encodings are unavailable in Release 6.

8.592.5. Exceptions

- **TRAP** (Trap) — immediate trap condition matched

8.593. TGEIU

8.593.1. Purpose

Raise a Trap exception when the TGEIU comparison matches.

8.593.2. Formats

MIPS

`tgeiu`

Assembly

`TGEIU {imm:s16}`

Availability

AR < R6

Decode priority

0

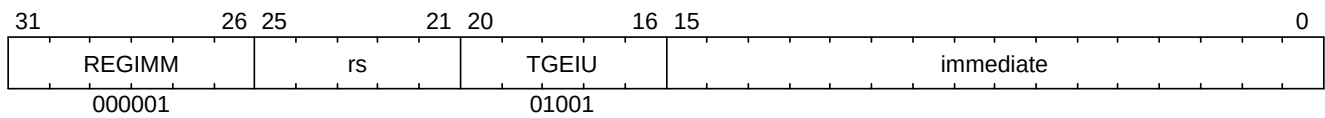


Figure 1652. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

microMIPS

`tgeiu.32`

Assembly

`TGEIU {rs:gpr}, {imm:s16}`

Availability

MMAR < R6

Decode priority

0

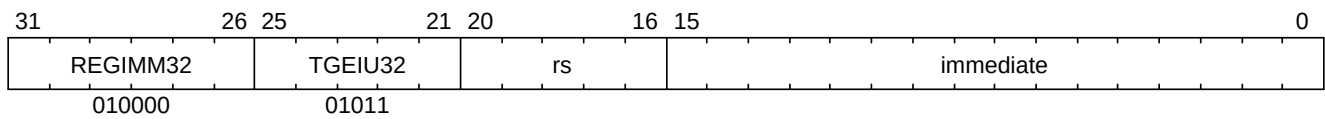


Figure 1653. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

8.593.3. Operation

```
lhs = int(zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(zero_extend(operands.imm, from_nbits=ctx.GPR.width))
if lhs >= rhs:
    raise TrapException("immediate trap condition matched")
```

8.593.4. Description

The 16-bit immediate is sign-extended before comparison. Unsigned forms reinterpret that sign-extended word as unsigned. Immediate trap encodings are unavailable in Release 6.

8.593.5. Exceptions

- **TRAP** (Trap) — immediate trap condition matched

8.594. TGEU

8.594.1. Purpose

Raise a Trap exception when unsigned rs is greater than or equal to unsigned rt.

8.594.2. Formats

MIPS

tgeu

Assembly

TGEU {rs:gpr}, {rt:gpr}, {code:u10}

Availability

true

Decode priority

0

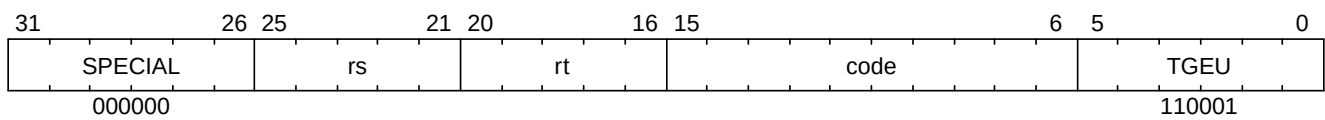


Figure 1654. Encoding (32 bits)

Decode

```
return TgeuOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

microMIPS

tgeu.32

Assembly

TGEU {rs:gpr}, {rt:gpr}, {code:u4}

Availability

true

Decode priority

0

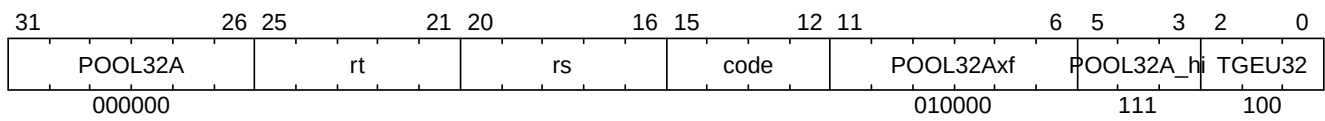


Figure 1655. Encoding (32 bits)


```
return TgeuOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

8.594.3. Operation

```
lhs = zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = zero_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
if lhs >= rhs:
    raise TrapException("TGEU condition matched")
```

8.594.4. Description

TGEU compares the complete active-width values of **rs** and **rt** as unsigned integers. The ten-bit code is carried for exception diagnostics but does not affect the comparison.

8.594.5. Exceptions

- **TRAP** (Trap) — TGEU condition matched

8.595. TLBINV

8.595.1. Purpose

Invalidate TLB entries matching ASID.

8.595.2. Formats

MIPS

tlbinv

Assembly

TLBINV

Availability

AR >= R2_3_5 && TLBINV

Decode priority

0

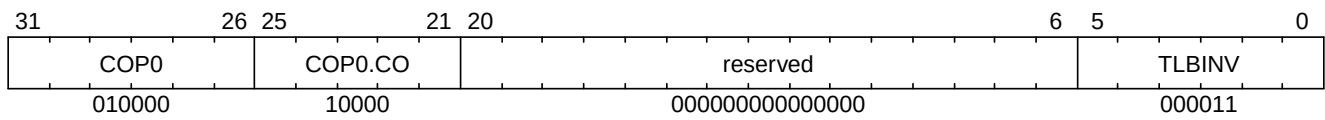


Figure 1656. Encoding (32 bits)

Decode

```
return Tlbinv0perands()
```

microMIPS

tlbinv.32

Assembly

TLBINV

Availability

TLBINV

Decode priority

0

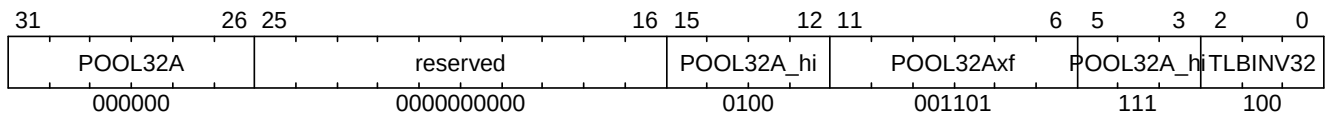


Figure 1657. Encoding (32 bits)

Decode

```
return TlbinvOperands()
```

nanoMIPS

tlbinv.32

Assembly

TLBINV

Availability

true

Decode priority

0

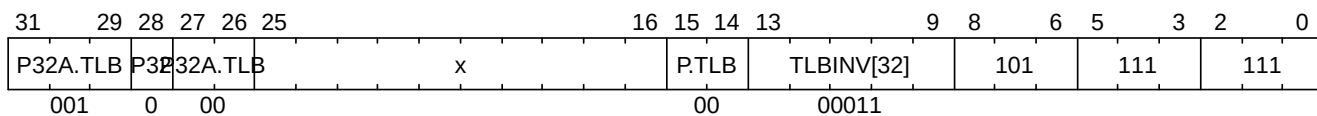


Figure 1658. Encoding (32 bits)

Decode

```
return TlbinvOperands()
```

8.595.3. Operation

```
tlbinv(ctx)
```

8.595.4. Description

TLBINV invalidates non-global entries matching EntryHi.ASID.

8.595.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — TLB invalidate is not implemented; the processor does not implement a TLB

8.596. TLBINVF

8.596.1. Purpose

Flush all TLB entries.

8.596.2. Formats

MIPS

tlbinvf

Assembly

TLBINVF

Availability

AR >= R2_3_5 && TLBINV

Decode priority

0

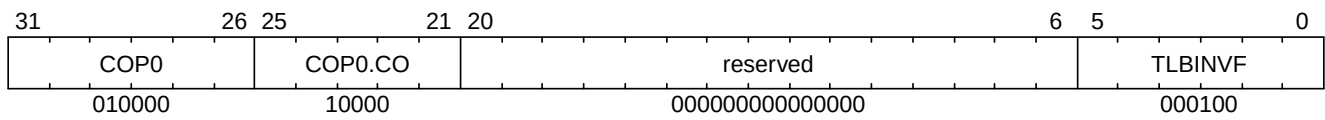


Figure 1659. Encoding (32 bits)

Decode

```
return Tlbinvf0operands()
```

microMIPS

tlbinvf.32

Assembly

TLBINVF

Availability

TLBINV

Decode priority

0

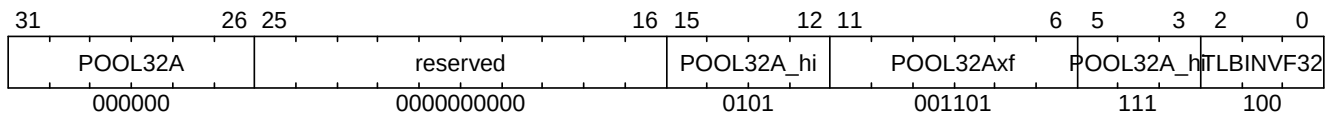


Figure 1660. Encoding (32 bits)

Decode

```
return Tlbinvf0perands()
```

nanoMIPS

`tlbinvf.32`

Assembly

TLBINVF

Availability

true

Decode priority

0

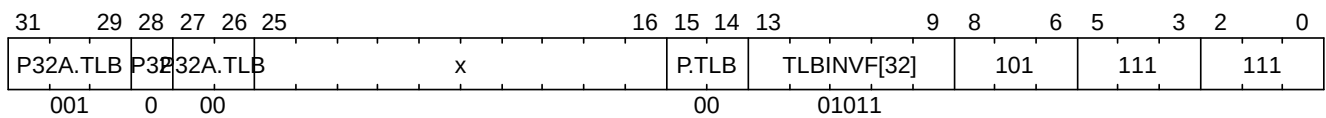


Figure 1661. Encoding (32 bits)

Decode

```
return Tlbinvf0perands()
```

8.596.3. Operation

```
tlbinv(ctx, flush=True)
```

8.596.4. Description

TLBINVF invalidates all matching non-guest TLB entries regardless of ASID.

8.596.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — TLB invalidate is not implemented; the processor does not implement a TLB

8.597. TLBP

8.597.1. Purpose

Probe the TLB for an entry matching CP0 EntryHi.

8.597.2. Formats

MIPS

tlbp

Assembly

TLBP

Availability

true

Decode priority

0

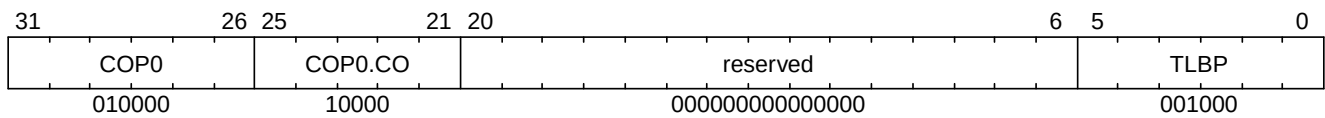


Figure 1662. Encoding (32 bits)

Decode

```
return Tlbp0operands()
```

microMIPS

tlbp.32

Assembly

TLBP

Availability

true

Decode priority

0

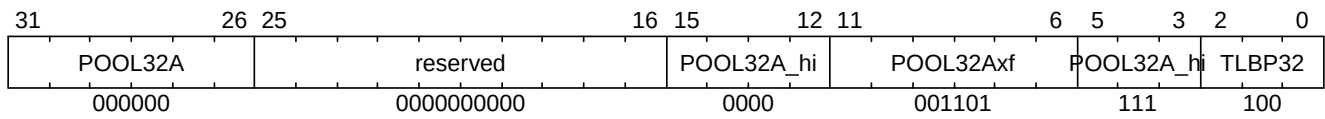


Figure 1663. Encoding (32 bits)

Decode

```
return TlbpOperands()
```

nanoMIPS

tlbp.32

Assembly

TLBP

Availability

true

Decode priority

0

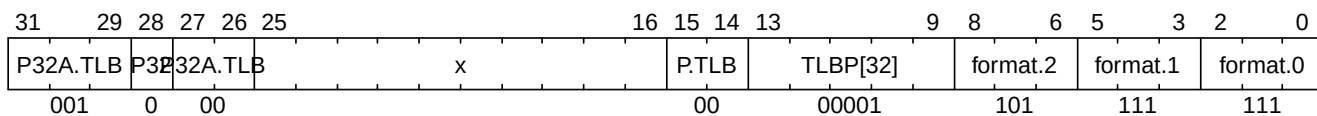


Figure 1664. Encoding (32 bits)

Decode

```
return TlbpOperands()
```

8.597.3. Operation

```
tlbp(ctx)
```

8.597.4. Description

A matching entry writes its index and clears Index.P. If no entry matches, the instruction sets Index.P.

8.597.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — the processor does not implement a TLB

8.598. TLBR

8.598.1. Purpose

Read indexed TLB entry.

8.598.2. Formats

MIPS

tlbr

Assembly

TLBR

Availability

true

Decode priority

0

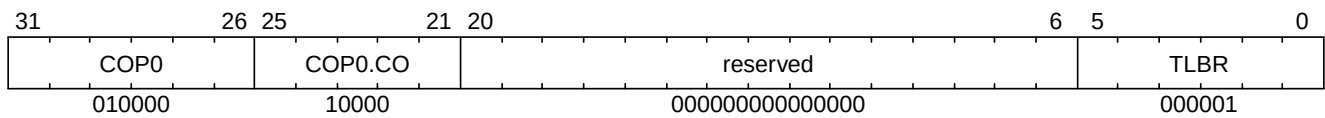


Figure 1665. Encoding (32 bits)

Decode

```
return TlbrOperands()
```

microMIPS

tlbr.32

Assembly

TLBR

Availability

true

Decode priority

0

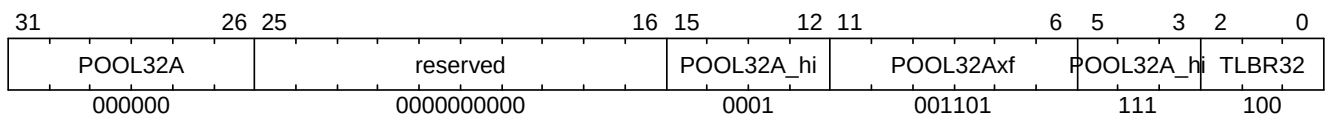


Figure 1666. Encoding (32 bits)

Decode

```
return TlbrOperands()
```

nanoMIPS

`tlbr.32`

Assembly

TLBR

Availability

true

Decode priority

0

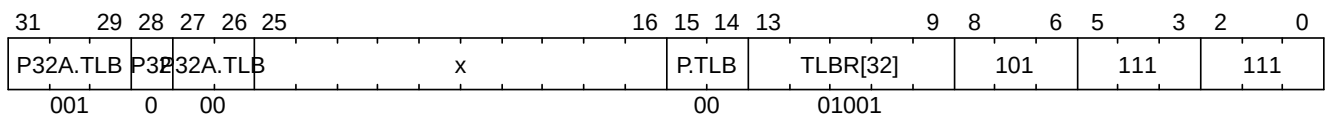


Figure 1667. Encoding (32 bits)

Decode

```
return TlbrOperands()
```

8.598.3. Operation

```
tlbr(ctx)
```

8.598.4. Description

TLBR loads the indexed translation entry into CP0 interface registers.

8.598.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — the processor does not implement a TLB

8.599. TLBWI

8.599.1. Purpose

Write a TLB entry selected by CP0 Index.

8.599.2. Formats

MIPS

tlbwi

Assembly

TLBWI

Availability

true

Decode priority

0

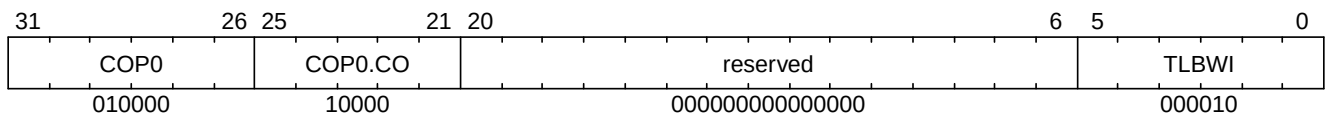


Figure 1668. Encoding (32 bits)

Decode

```
return TlbwiOperands()
```

microMIPS

tlbwi.32

Assembly

TLBWI

Availability

true

Decode priority

0

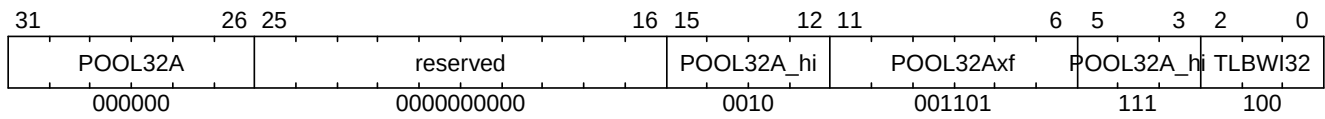


Figure 1669. Encoding (32 bits)

Decode

```
return TlbwiOperands()
```

nanoMIPS

tlbwi.32

Assembly

TLBWI

Availability

true

Decode priority

0

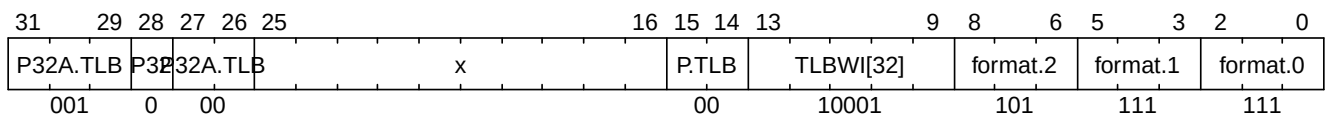


Figure 1670. Encoding (32 bits)

Decode

```
return TlbwiOperands()
```

8.599.3. Operation

```
tlbwi(ctx)
```

8.599.4. Description

The instruction copies EntryHi, EntryLo0, EntryLo1, and PageMask into the indexed TLB entry.

8.599.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — the processor does not implement a TLB

8.600. TLBWR

8.600.1. Purpose

Write a random TLB entry.

8.600.2. Formats

MIPS

`tlbwr`

Assembly

`TLBWR`

Availability

true

Decode priority

0

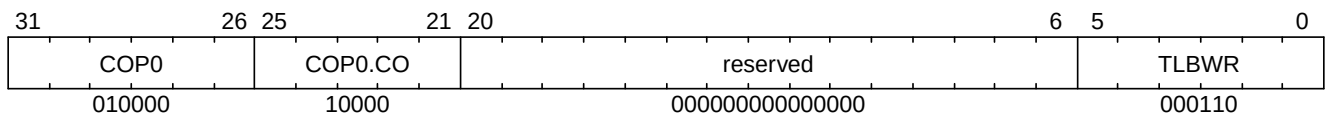


Figure 1671. Encoding (32 bits)

Decode

```
return TlbwrOperands()
```

microMIPS

`tlbwr.32`

Assembly

`TLBWR`

Availability

true

Decode priority

0

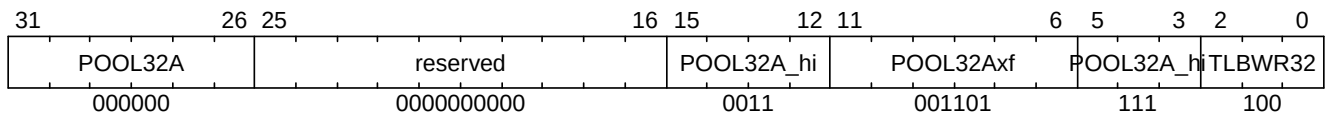


Figure 1672. Encoding (32 bits)

Decode

```
return TlbwrOperands()
```

nanoMIPS

`tlbwr.32`

Assembly

TLBWR

Availability

true

Decode priority

0

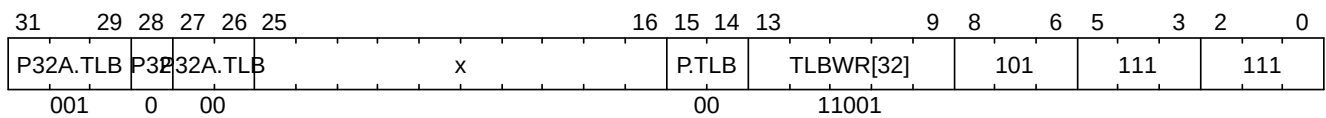


Figure 1673. Encoding (32 bits)

Decode

```
return TlbwrOperands()
```

8.600.3. Operation

```
tlbwr(ctx)
```

8.600.4. Description

TLBWR writes the current translation state to a selected TLB entry.

8.600.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — the processor does not implement a TLB

8.601. TLT

8.601.1. Purpose

Raise a Trap exception when signed rs is less than signed rt.

8.601.2. Formats

MIPS

`slt`

Assembly

`TLT {rs:gpr}, {rt:gpr}, {code:u10}`

Availability

true

Decode priority

0

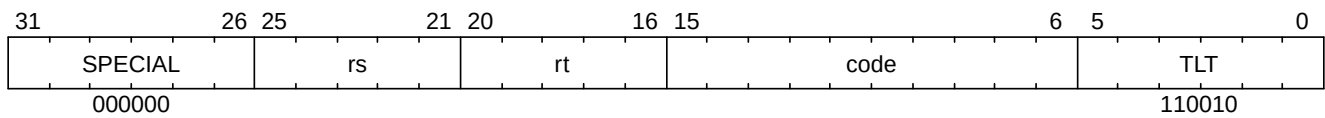


Figure 1674. Encoding (32 bits)

Decode

```
return Tlt0operands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

microMIPS

`slt.32`

Assembly

`TLT {rs:gpr}, {rt:gpr}, {code:u4}`

Availability

true

Decode priority

0

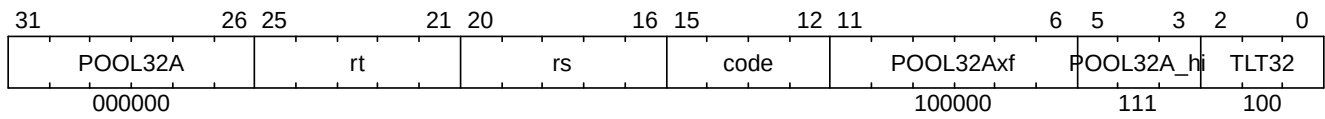


Figure 1675. Encoding (32 bits)

```
return TltOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

8.601.3. Operation

```
lhs = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = sign_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
if lhs < rhs:
    raise TrapException("TLT condition matched")
```

8.601.4. Description

TLT compares the complete active-width values of **rs** and **rt** as signed integers. The ten-bit code is carried for exception diagnostics but does not affect the comparison.

8.601.5. Exceptions

- **TRAP** (Trap) — TLT condition matched

8.602. TLTI

8.602.1. Purpose

Raise a Trap exception when the TLTI comparison matches.

8.602.2. Formats

MIPS

tlti

Assembly

TLTI {imm:s16}

Availability

AR < R6

Decode priority

0

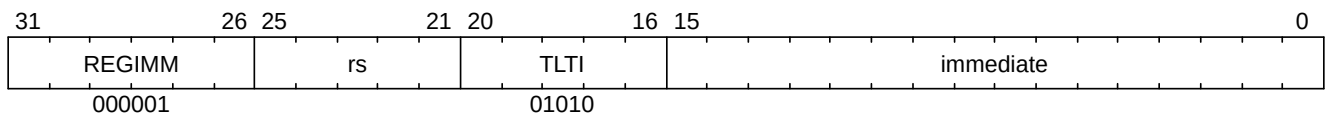


Figure 1676. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

microMIPS

tlti.32

Assembly

TLTI {rs:gpr}, {imm:s16}

Availability

MMAR < R6

Decode priority

0

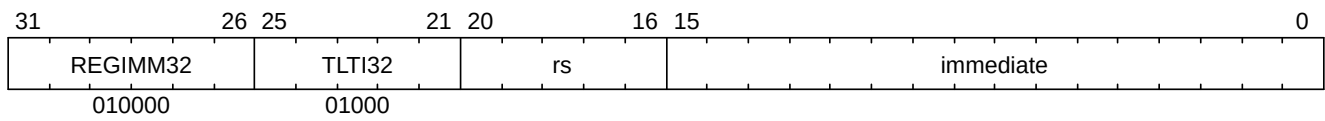


Figure 1677. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

8.602.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(operands.imm, from_nbits=ctx.GPR.width))
if lhs < rhs:
    raise TrapException("immediate trap condition matched")
```

8.602.4. Description

The 16-bit immediate is sign-extended before comparison. Unsigned forms reinterpret that sign-extended word as unsigned. Immediate trap encodings are unavailable in Release 6.

8.602.5. Exceptions

- **TRAP** (Trap) — immediate trap condition matched

8.603. TLTIU

8.603.1. Purpose

Raise a Trap exception when the TLTIU comparison matches.

8.603.2. Formats

MIPS

`tltiu`

Assembly

`TLTIU {imm:s16}`

Availability

AR < R6

Decode priority

0

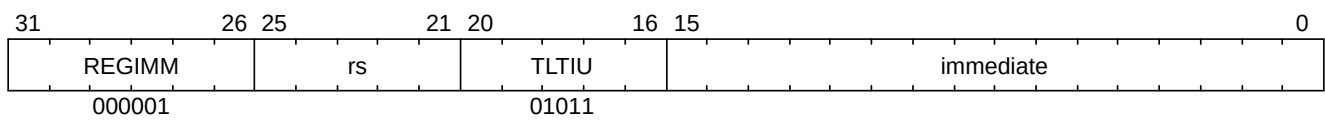


Figure 1678. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

microMIPS

`tltiu.32`

Assembly

`TLTIU {rs:gpr}, {imm:s16}`

Availability

MMAR < R6

Decode priority

0

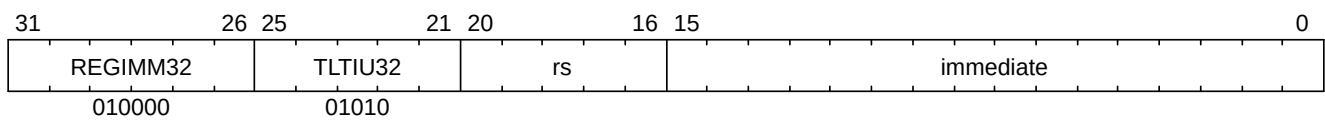


Figure 1679. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

8.603.3. Operation

```
lhs = int(zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(zero_extend(operands.imm, from_nbits=ctx.GPR.width))
if lhs < rhs:
    raise TrapException("immediate trap condition matched")
```

8.603.4. Description

The 16-bit immediate is sign-extended before comparison. Unsigned forms reinterpret that sign-extended word as unsigned. Immediate trap encodings are unavailable in Release 6.

8.603.5. Exceptions

- **TRAP** (Trap) — immediate trap condition matched

8.604. TLTU

8.604.1. Purpose

Raise a Trap exception when unsigned rs is less than unsigned rt.

8.604.2. Formats

MIPS

`tltu`

Assembly

`TLTU {rs:gpr}, {rt:gpr}, {code:u10}`

Availability

true

Decode priority

0

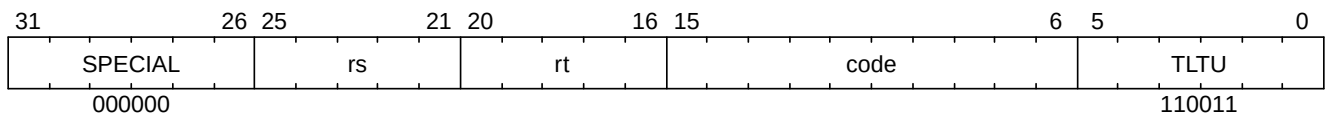


Figure 1680. Encoding (32 bits)

Decode

```
return TltuOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

microMIPS

`tltu.32`

Assembly

`TLTU {rs:gpr}, {rt:gpr}, {code:u4}`

Availability

true

Decode priority

0

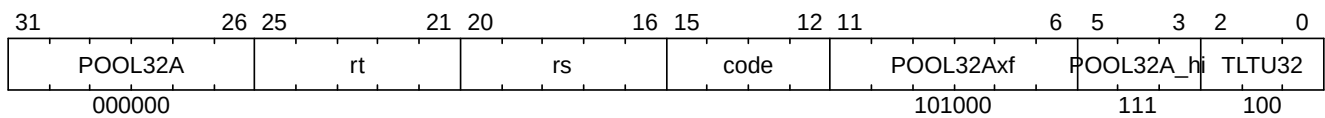


Figure 1681. Encoding (32 bits)

```
return TltuOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

8.604.3. Operation

```
lhs = zero_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
rhs = zero_extend(ctx.GPR[operands.rt], from_nbits=ctx.GPR.width)
if lhs < rhs:
    raise TrapException("TLTU condition matched")
```

8.604.4. Description

TLTU compares the complete active-width values of **rs** and **rt** as unsigned integers. The ten-bit code is carried for exception diagnostics but does not affect the comparison.

8.604.5. Exceptions

- **TRAP** (Trap) — TLTU condition matched

8.605. TNE

8.605.1. Purpose

Raise a Trap exception when two GPR values are different.

8.605.2. Formats

MIPS

`tne`

Assembly

`TNE {rs:gpr}, {rt:gpr}, {code:u10}`

Availability

true

Decode priority

0

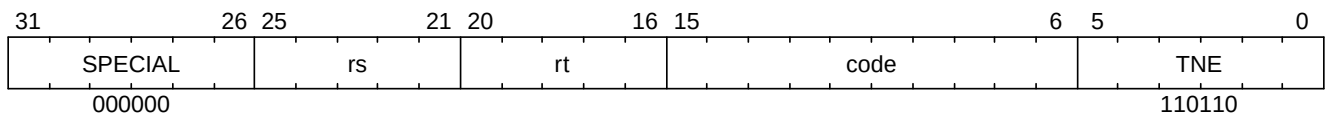


Figure 1682. Encoding (32 bits)

Decode

```
return Tne0operands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

microMIPS

`tne.32`

Assembly

`TNE {rs:gpr}, {rt:gpr}, {code:u4}`

Availability

true

Decode priority

0

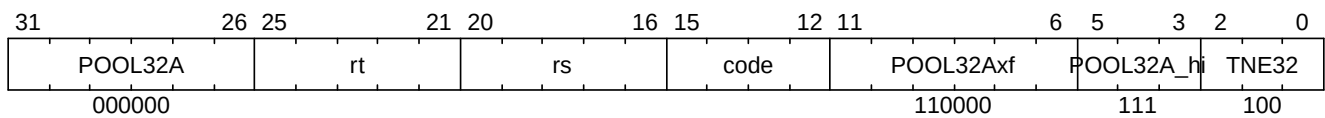


Figure 1683. Encoding (32 bits)

Decode

```
return TneOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=10))
```

nanoMIPS

tne.32

Assembly

TNE {rs:gpr}, {rt:gpr}, {code:u5}

Availability

!(NMS)

Decode priority

0

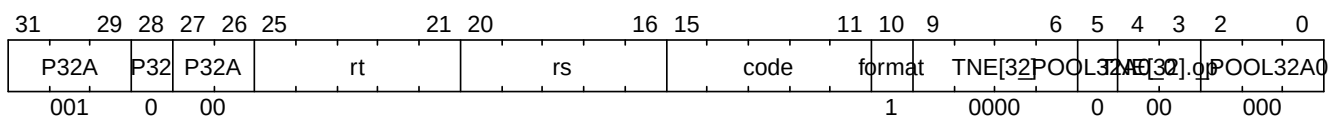


Figure 1684. Encoding (32 bits)

Decode

```
return TneOperands(rs=rs, rt=rt, code=zero_extend(code, from_nbits=5))
```

8.605.3. Operation

```
if ctx.GPR[operands.rs] != ctx.GPR[operands.rt]:  
    raise TrapException("TNE condition matched")
```

8.605.4. Description

TNE compares the complete active-width values of `rs` and `rt`. The five-bit code is carried for exception diagnostics but does not affect the comparison. TNE is unavailable in the nanoMIPS subset.

8.605.5. Exceptions

- **RI** (Reserved Instruction) — TNE is reserved when CP0 Config5.NMS is set.
- **TRAP** (Trap) — TNE condition matched

8.606. TNEI

8.606.1. Purpose

Raise a Trap exception when the TNEI comparison matches.

8.606.2. Formats

MIPS

`tnei`

Assembly

`TNEI {imm:s16}`

Availability

AR < R6

Decode priority

0

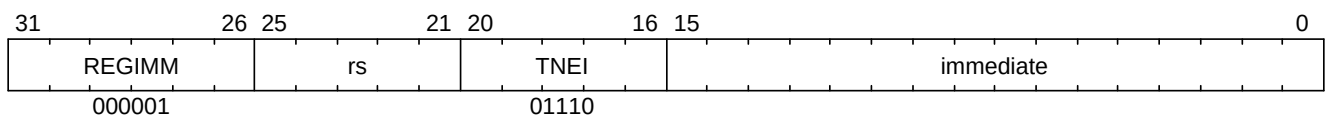


Figure 1685. Encoding (32 bits)

Decode

```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

microMIPS

`tnei.32`

Assembly

`TNEI {rs:gpr}, {imm:s16}`

Availability

MMAR < R6

Decode priority

0

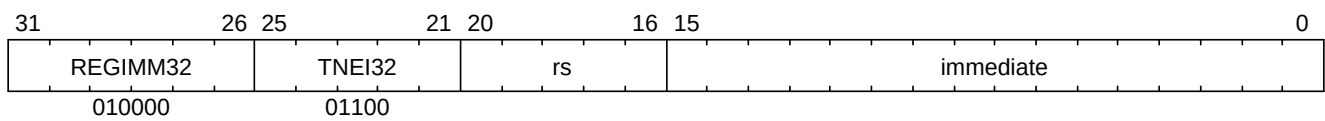


Figure 1686. Encoding (32 bits)


```
imm = sign_extend(immediate, from_nbits=16)
return TrapImmediateOperands(rs=rs, imm=imm)
```

8.606.3. Operation

```
lhs = int(sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width))
rhs = int(sign_extend(operands.imm, from_nbits=ctx.GPR.width))
if lhs != rhs:
    raise TrapException("immediate trap condition matched")
```

8.606.4. Description

The 16-bit immediate is sign-extended before comparison. Unsigned forms reinterpret that sign-extended word as unsigned. Immediate trap encodings are unavailable in Release 6.

8.606.5. Exceptions

- **TRAP** (Trap) — immediate trap condition matched

8.607. TRUNC.L.D

8.607.1. Purpose

Convert D format to L format using trunc rounding.

8.607.2. Formats

MIPS

`trunc.l.d`

Assembly

`TRUNC.L.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

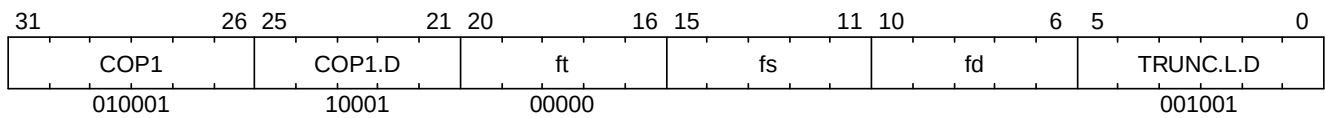


Figure 1687. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`trunc.l.d`

Assembly

`TRUNC.L.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

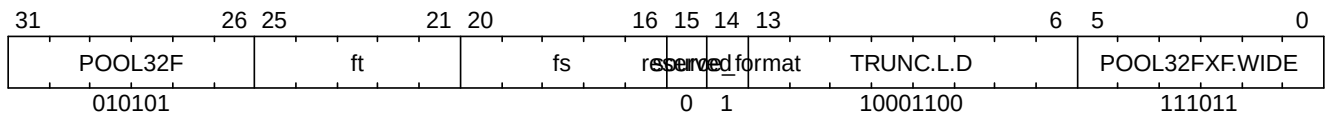


Figure 1688. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

`trunc.l.d`

Assembly

`TRUNC.L.D {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

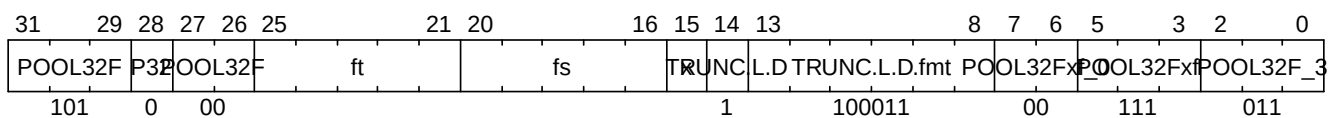


Figure 1689. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.607.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "D")
result, flags = ieee_float_to_int(
    "D", "L", fpu_rounding_mode(ctx, 1), get_fpr(ctx, operands.fs, "D"), rm=1
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.607.4. Description

TRUNC.L.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.607.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.608. TRUNC.L.S

8.608.1. Purpose

Convert S format to L format using trunc rounding.

8.608.2. Formats

MIPS

`trunc.l.s`

Assembly

`TRUNC.L.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

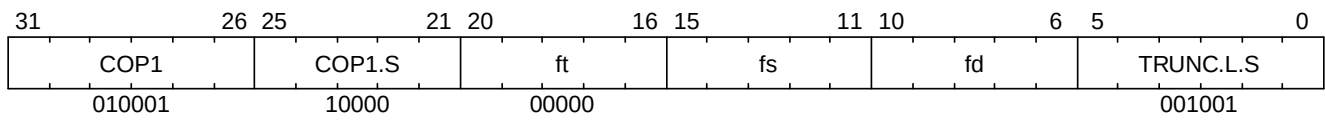


Figure 1690. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`trunc.l.s`

Assembly

`TRUNC.L.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

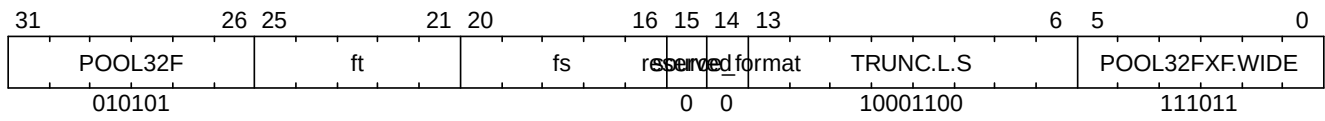


Figure 1691. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

`trunc.l.s`

Assembly

`TRUNC.L.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

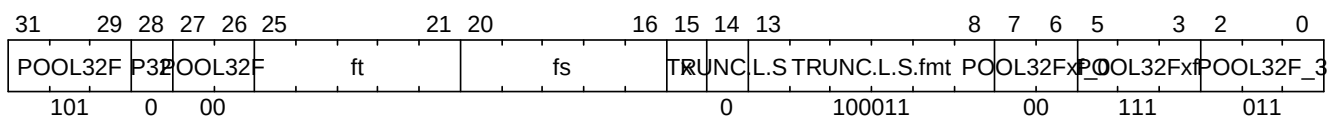


Figure 1692. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.608.3. Operation

```
validate_access_to_cp1_resources(ctx, "L", "S")
result, flags = ieee_float_to_int(
    "S", "L", fpu_rounding_mode(ctx, 1), get_fpr(ctx, operands.fs, "S"), rm=1
)
write_fpu_result(ctx, operands.fd, result, flags, "L")
```

8.608.4. Description

TRUNC.L.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.608.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.609. TRUNC.W.D

8.609.1. Purpose

Convert D format to W format using trunc rounding.

8.609.2. Formats

MIPS

trunc.w.d

Assembly

TRUNC.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

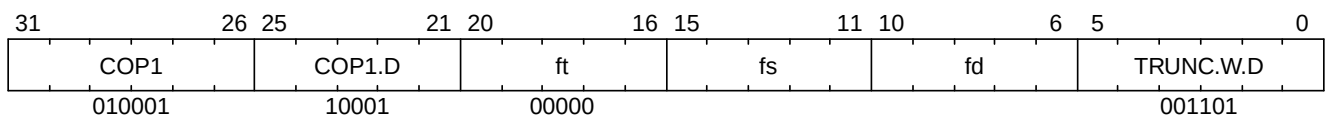


Figure 1693. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

trunc.w.d

Assembly

TRUNC.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

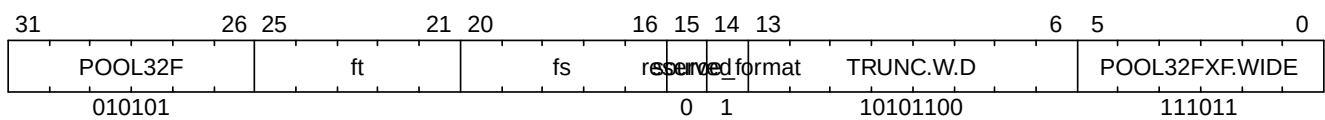


Figure 1694. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

trunc.w.d

Assembly

TRUNC.W.D {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

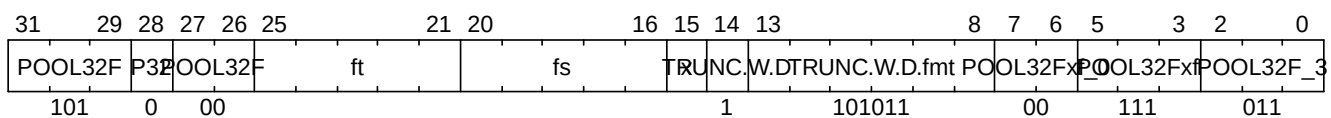


Figure 1695. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.609.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "D")
result, flags = ieee_float_to_int(
    "D", "W", fpu_rounding_mode(ctx, 1), get_fpr(ctx, operands.fs, "D"), rm=1
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.609.4. Description

TRUNC.W.D overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.609.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.610. TRUNC.W.S

8.610.1. Purpose

Convert S format to W format using trunc rounding.

8.610.2. Formats

MIPS

`trunc.w.s`

Assembly

`TRUNC.W.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

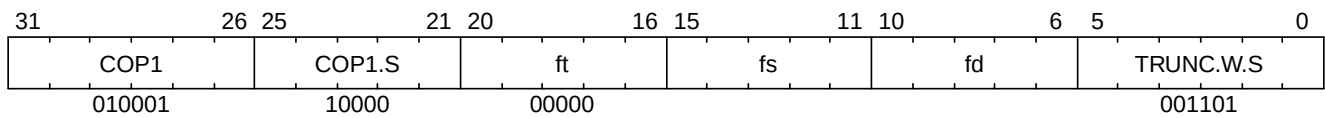


Figure 1696. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=fd, fs=fs)
```

microMIPS

`trunc.w.s`

Assembly

`TRUNC.W.S {fd:fpr}, {fs:fpr}`

Availability

true

Decode priority

0

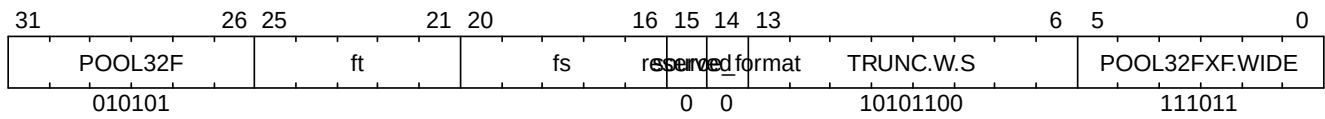


Figure 1697. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

nanoMIPS

trunc.w.s

Assembly

TRUNC.W.S {fd:fpr}, {fs:fpr}

Availability

true

Decode priority

0

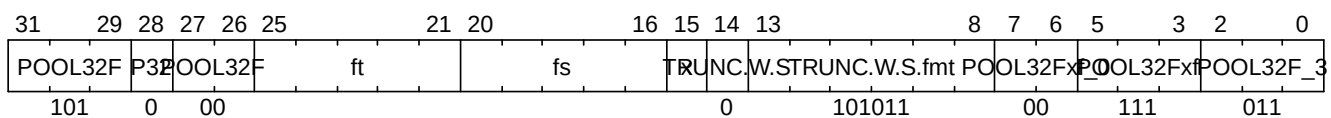


Figure 1698. Encoding (32 bits)

Decode

```
return FpuConvertOperands(fd=ft, fs=fs)
```

8.610.3. Operation

```
validate_access_to_cp1_resources(ctx, "W", "S")
result, flags = ieee_float_to_int(
    "S", "W", fpu_rounding_mode(ctx, 1), get_fpr(ctx, operands.fs, "S"), rm=1
)
write_fpu_result(ctx, operands.fd, result, flags, "W")
```

8.610.4. Description

TRUNC.W.S overrides FCSR.RM for this conversion and reports the resulting IEEE exceptions.

8.610.5. Exceptions

- **CPU** (Coprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **FPE** (Floating Point Exception) — A CP1 operation raised an enabled IEEE condition.
- **RI** (Reserved Instruction) — FPU is not implemented; D format is not supported; S format is not supported; W format is not supported; L format is not supported; L format requires 64-bit operation mode; PS format is not supported; PS format requires 64-bit operation mode; 64-bit format is not supported; 32-bit format is not supported; 64-bit operations are not enabled

8.611. UALDM

8.611.1. Purpose

Load multiple possibly unaligned doublewords.

8.611.2. Formats

nanoMIPS

`ualdm.32`

Assembly

`UALDM {rt:gpr}, {offset:s9}({rs:gpr}), {count:u3}`

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

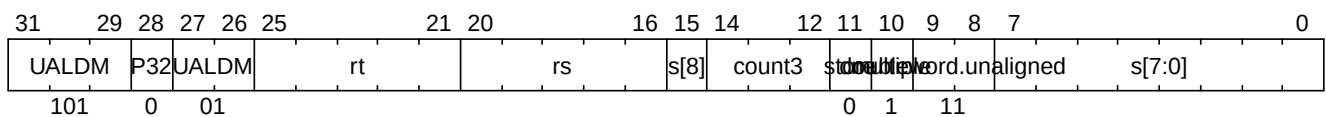


Figure 1699. Encoding (32 bits)

Decode

```
return DoubleMultipleOperands(  
    rt=rt,  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    count=8 if count3 == 0 else count3,  
)
```

8.611.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_multiple_access(ctx, operands, store=False, unaligned=True)
```

8.611.4. Description

UALDM transfers signed 64-bit elements without requiring doubleword alignment.

8.611.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.

- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.612. UALH

8.612.1. Purpose

Load a signed halfword from a possibly unaligned address.

8.612.2. Formats

nanoMIPS

`ualh.32`

Assembly

`UALH {offset:s9}({rs:gpr})`

Availability

`!(NMS)`

Decode priority

`0`

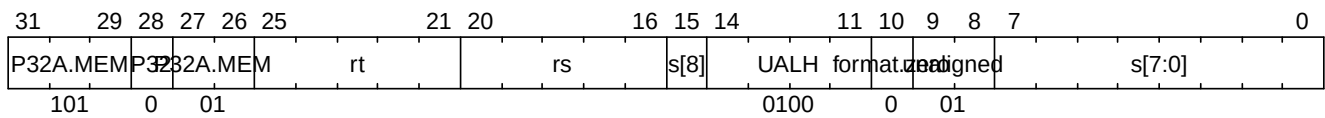


Figure 1700. Encoding (32 bits)

Decode

```
return MultipleMemoryOperands(rt, rs, sign_extend(s, from_nbits=9), 1)
```

8.612.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Load")
data = read_memory_at_va(ctx, va, nbytes=2, unaligned_ok=True)
ctx.GPR[operands.rt] = sign_extend(data, from_nbits=16)
```

8.612.4. Description

This full-nano form is unavailable in the nanoMIPS subset.

8.612.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.

- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.613. UALWM

8.613.1. Purpose

Load multiple words from a possibly unaligned address.

8.613.2. Formats

nanoMIPS

`ualwm.32`

Assembly

`UALWM {offset:s9}({rs:gpr}), {count:u3}`

Availability

!(NMS)

Decode priority

0

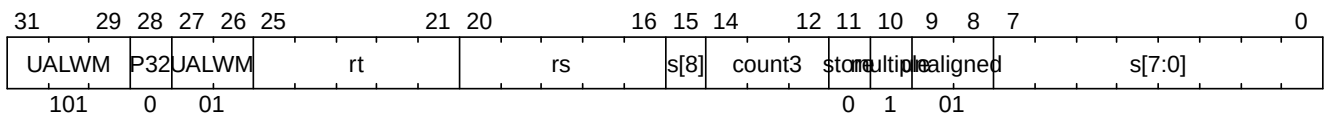


Figure 1701. Encoding (32 bits)

Decode

```
return MultipleMemoryOperands(  
    rt, rs, sign_extend(s, from_nbits=9), 8 if count3 == 0 else count3  
)
```

8.613.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)  
for index in range(operands.count):  
    register = (  
        operands.rt + index  
        if operands.rt + index < 32  
        else operands.rt + index - 16  
    )  
    va = effective_address(ctx, base, int(operands.offset) + index * 4, "Load")  
    data = read_memory_at_va(ctx, va, nbytes=4, unaligned_ok=True)  
    ctx.GPR[register] = sign_extend(data, from_nbits=32)  
    if register == operands.rs and index != operands.count - 1:  
        raise ValueError("multiple load updating its base early is UNPREDICTABLE")
```

8.613.4. Description

This full-nano form is unavailable in the nanoMIPS subset.

8.613.5. Exceptions

- **ADEL** (Address Error (load or fetch)) — A load or instruction-fetch address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **TLBL** (TLB Load/Fetch) — A load or instruction-fetch address could not be translated.
- **TLBRI** (TLB Read Inhibit) — A load selected a read-inhibited translation.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.614. UASDM

8.614.1. Purpose

Store multiple possibly unaligned doublewords.

8.614.2. Formats

nanoMIPS

`uasdm.32`

Assembly

`UASDM {rt:gpr}, {offset:s9}({rs:gpr}), {count:u3}`

Availability

AT >= MIPS64_32 && !(NMS)

Decode priority

0

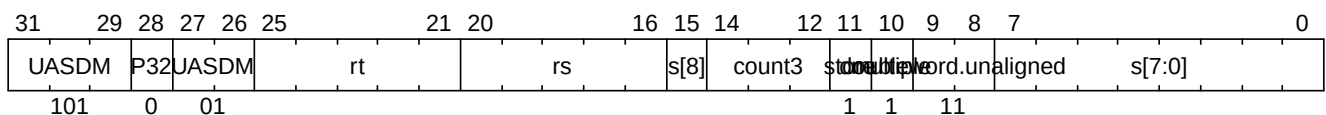


Figure 1702. Encoding (32 bits)

Decode

```
return DoubleMultipleOperands(  
    rt=rt,  
    rs=rs,  
    offset=sign_extend(s, from_nbits=9),  
    count=8 if count3 == 0 else count3,  
)
```

8.614.3. Operation

```
validate_access_to_64bit_operations(ctx)  
execute_double_multiple_access(ctx, operands, store=True, unaligned=True)
```

8.614.4. Description

UASDM transfers 64-bit elements without requiring doubleword alignment.

8.614.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.

- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **RI** (Reserved Instruction) — 64-bit operations are disabled
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.615. UASH

8.615.1. Purpose

Store a halfword to a possibly unaligned address.

8.615.2. Formats

nanoMIPS

uash.32

Assembly

UASH {offset:s9}({rs:gpr})

Availability

!(NMS)

Decode priority

0

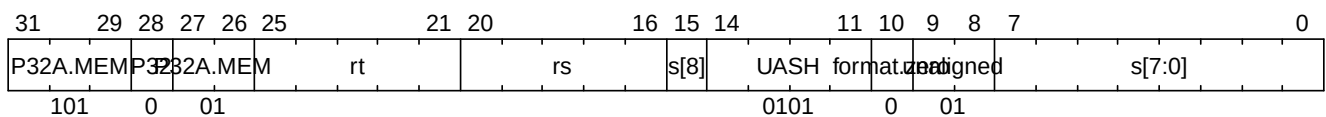


Figure 1703. Encoding (32 bits)

Decode

```
return MultipleMemoryOperands(rt, rs, sign_extend(s, from_nbits=9), 1)
```

8.615.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)
va = effective_address(ctx, base, operands.offset, "Store")
data = zero_extend(ctx.GPR[operands.rt], from_nbits=16)
write_memory_at_va(ctx, data, va, nbytes=2, unaligned_ok=True)
```

8.615.4. Description

This full-nano form is unavailable in the nanoMIPS subset.

8.615.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.

- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.616. UASWM

8.616.1. Purpose

Store multiple words to a possibly unaligned address.

8.616.2. Formats

nanoMIPS

uaswm.32

Assembly

UASWM {offset:s9}({rs:gpr}), {count:u3}

Availability

!(NMS)

Decode priority

0

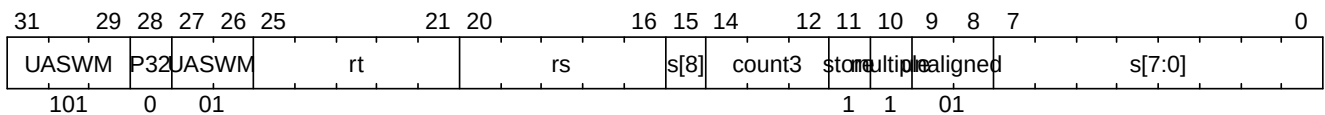


Figure 1704. Encoding (32 bits)

Decode

```
return MultipleMemoryOperands(  
    rt, rs, sign_extend(s, from_nbits=9), 8 if count3 == 0 else count3  
)
```

8.616.3. Operation

```
base = sign_extend(ctx.GPR[operands.rs], from_nbits=ctx.GPR.width)  
for index in range(operands.count):  
    register = (  
        0  
        if operands.rt == 0  
        else operands.rt + index  
        if operands.rt + index < 32  
        else operands.rt + index - 16  
    )  
    va = effective_address(ctx, base, int(operands.offset) + index * 4, "Store")  
    data = zero_extend(ctx.GPR[register], from_nbits=32)  
    write_memory_at_va(ctx, data, va, nbytes=4, unaligned_ok=True)
```

8.616.4. Description

This full-nano form is unavailable in the nanoMIPS subset.

8.616.5. Exceptions

- **ADES** (Address Error (store)) — A store address is invalid.
- **DBE** (Bus Error) — A data-memory access reported a bus error.
- **MOD** (TLB Modified) — A store selected a valid but non-writable translation.
- **TLBS** (TLB Store) — A store address could not be translated.
- **WATCH** (Watch) — A data-memory access matched an enabled watchpoint.

8.617. WAIT

8.617.1. Purpose

Enter wait state.

8.617.2. Formats

MIPS

`wait`

Assembly

`WAIT {code:u19}`

Availability

true

Decode priority

0

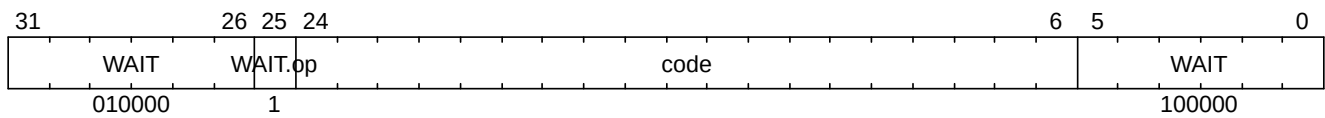


Figure 1705. Encoding (32 bits)

Decode

```
return Wait0perands(code=zero_extend(code, from_nbits=19))
```

microMIPS

`wait.mm.32`

Assembly

`WAIT {code:u10}`

Availability

MMAR >= R6

Decode priority

0

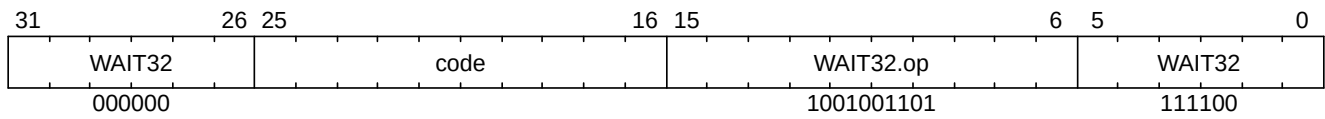


Figure 1706. Encoding (32 bits)

Decode

```
return WaitOperands(code=zero_extend(code, from_nbits=10))
```

nanoMIPS

wait.32

Assembly

WAIT {code:u10}

Availability

true

Decode priority

0

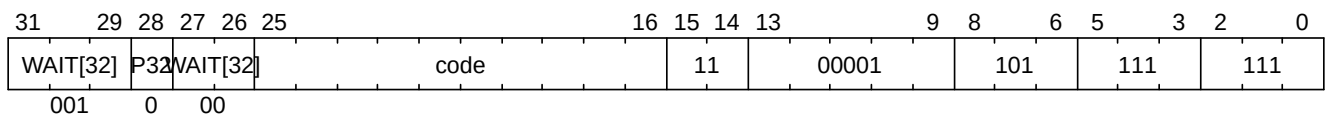


Figure 1707. Encoding (32 bits)

Decode

```
return WaitOperands(code=zero_extend(code, from_nbits=10))
```

8.617.3. Operation

```
if JumpDelaySlot(ctx) or ForbiddenSlot(ctx):
    raise ReservedInstruction("WAIT is not permitted in a control-transfer slot")
if not IsCoproprocessor0Enabled(ctx):
    raise CoprocessorUnusable(0)
ctx.in_wait_state = True
```

8.617.4. Description

WAIT sleeps until an interrupt or implementation-defined wake event.

8.617.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.
- **RI** (Reserved Instruction) — WAIT is not permitted in a control-transfer slot

8.618. WRDSP

8.618.1. Purpose

Write selected DSPControl fields from a GPR.

8.618.2. Formats

MIPS

`wrdsp`

Assembly

`WRDSP {rs:gpr}, {mask:u10}`

Availability

DSP && AR >= R2_3_5

Decode priority

0

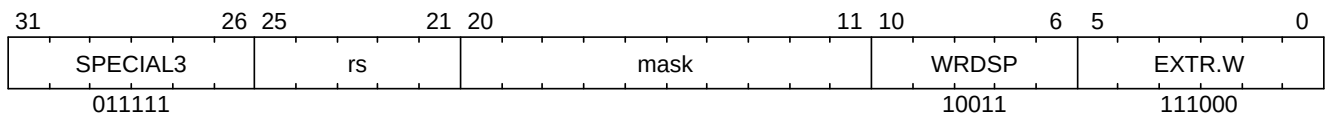


Figure 1708. Encoding (32 bits)

Decode

```
return DspControlWriteOperands(rs=rs, mask=mask)
```

microMIPS

`wrdsp.32`

Assembly

`WRDSP {rs:gpr}, {mask:u7}`

Availability

DSP

Decode priority

0

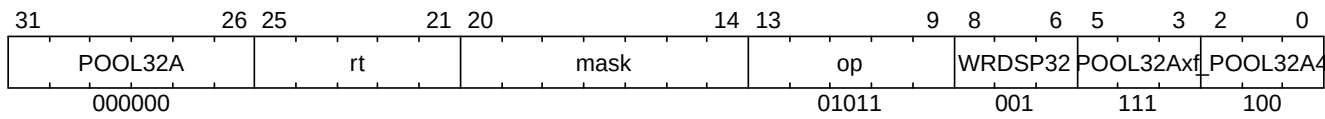


Figure 1709. Encoding (32 bits)

Decode

```
return DspControlWriteOperands(rs=rt, mask=mask)
```

nanoMIPS

wrdsp.32

Assembly

WRDSP {rs:gpr}, {mask:u7}

Availability

DSP

Decode priority

0

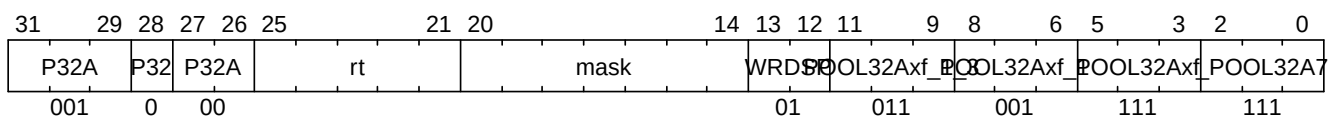


Figure 1710. Encoding (32 bits)

Decode

```
return DspControlWriteOperands(rs=rt, mask=mask)
```

8.618.3. Operation

```
validate_access_to_dsp_resources(ctx)
value = zero_extend(ctx.GPR[operands.rs], from_nbits=32)
if operands.mask & 0x01:
    ctx.DSPControl.Pos = value & 0x3F
if operands.mask & 0x02:
    ctx.DSPControl.Scound = (value >> 7) & 0x3F
if operands.mask & 0x04:
    ctx.DSPControl.C = (value >> 13) & 1
if operands.mask & 0x08:
    ctx.DSPControl.Ouflag = (value >> 16) & 0xFF
if operands.mask & 0x10:
    ctx.DSPControl.Ccond = (value >> 24) & 0xF
if operands.mask & 0x20:
    ctx.DSPControl.EFI = (value >> 14) & 1
```

8.618.4. Description

The mask independently selects Pos, Scound, C, Ouflag, Ccond, and EFI; unselected fields retain their old values.

8.618.5. Exceptions

None.

8.619. WRPGPR

8.619.1. Purpose

Write a GPR in the previous shadow register set.

8.619.2. Formats

MIPS

`wrpgpr`

Assembly

`WRPGPR {dst:gpr}, {src:gpr}`

Availability

AR >= R2_3_5

Decode priority

0

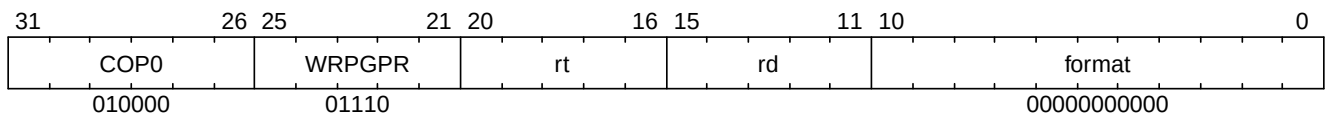


Figure 1711. Encoding (32 bits)

Decode

```
return WrpgprOperands(dst=rd, src=rt)
```

microMIPS

`wrpgpr.32`

Assembly

`WRPGPR {dst:gpr}, {src:gpr}`

Availability

true

Decode priority

0

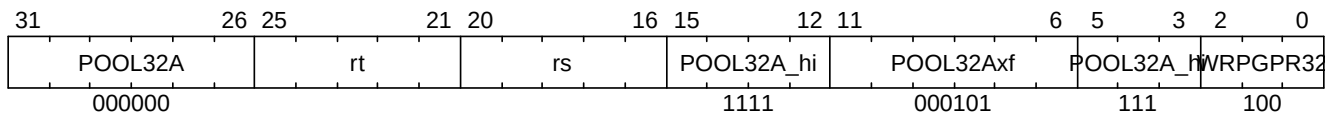


Figure 1712. Encoding (32 bits)

Decode

```
return WrpgprOperands(dst=rt, src=rs)
```

nanoMIPS

`wrpgpr.32`

Assembly

`WRPGPR {dst:gpr}, {src:gpr}`

Availability

true

Decode priority

0

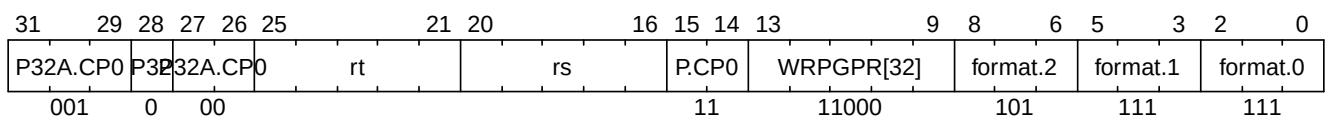


Figure 1713. Encoding (32 bits)

Decode

```
return WrpgprOperands(dst=rt, src=rs)
```

8.619.3. Operation

```
if not IsCoproprocessor0Enabled(ctx):  
    raise CoprocessorUnusable(0)  
write_previous_gpr(ctx, operands.dst, ctx.GPR[operands.src])
```

8.619.4. Description

The source is read from the current shadow set and the destination belongs to SRSCtl.PSS. On a processor without shadow register sets, the operation is an ordinary GPR copy.

8.619.5. Exceptions

- **CPU** (Coproprocessor Unusable) — Coprocessor access was attempted while that coprocessor was disabled.

8.620. WSBH

8.620.1. Purpose

Swap the bytes within each 16-bit halfword of the low 32-bit word in `rs` and place the result in `rt`.

8.620.2. Formats

MIPS

`wsbh`

Assembly

`WSBH {rt:gpr}, {rs:gpr}`

Availability

AR >= R2_3_5

Decode priority

0

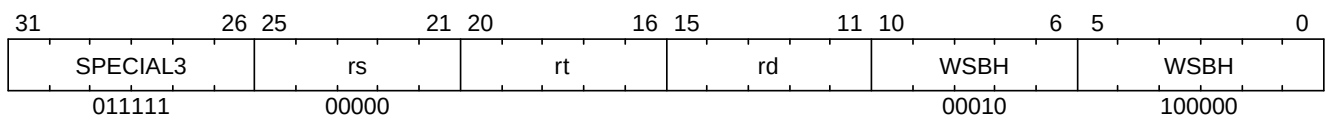


Figure 1714. Encoding (32 bits)

Decode

```
return WsbhOperands(rt=rd, rs=rt)
```

microMIPS

`wsbh.32`

Assembly

`WSBH {rt:gpr}, {rs:gpr}`

Availability

true

Decode priority

0

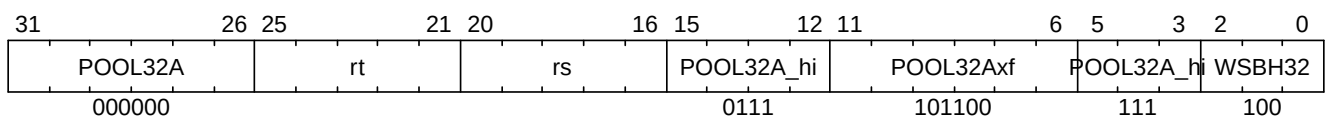


Figure 1715. Encoding (32 bits)

Decode

```
return WsbhOperands(rt=rt, rs=rs)
```

nanoMIPS

wsbh.32

Assembly

WSBH {rt:gpr}, {rs:gpr}

Availability

!(NMS)

Decode priority

0

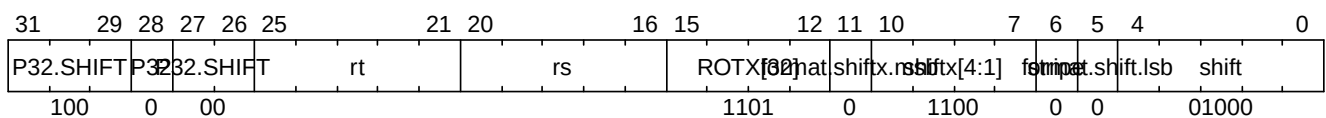


Figure 1716. Encoding (32 bits)

Decode

```
return WsbhOperands(rt=rt, rs=rs)
```

8.620.3. Operation

```
source = int(zero_extend(ctx.GPR[operands.rs], from_nbits=32))
result = (
    ((source & 0x0000_00FF) << 8)
    | ((source & 0x0000_FF00) >> 8)
    | ((source & 0x00FF_0000) << 8)
    | ((source & 0xFF00_0000) >> 8)
)
ctx.GPR[operands.rt] = sign_extend(result, from_nbits=32)
```

8.620.4. Description

The nanoMIPS WSBH spelling is an assembly alias for ROTX[32] with shift=8, shiftx=24, and stripe=0. It reverses the two bytes in each halfword, then sign-extends the resulting word to the implemented GPR width. The alias is unavailable when the NMS subset is enabled.

8.620.5. Exceptions

- **RI** (Reserved Instruction) — WSBH is reserved when CP0 Config5.NMS is set.

8.621. XOR

8.621.1. Purpose

Compute the logical XOR of two GPRs and place it in the destination GPR.

8.621.2. Formats

MIPS

xor

Assembly

XOR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

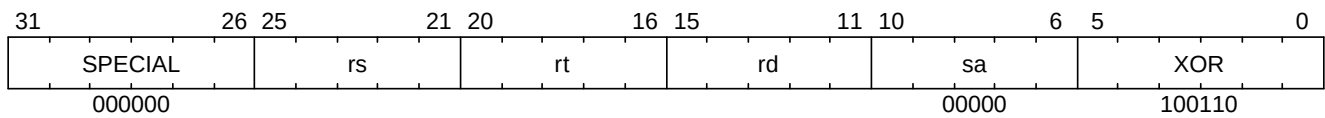


Figure 1717. Encoding (32 bits)

Decode

```
del ctx, exec
return XorOperands(rd=rd, rs=rs, rt=rt)
```

microMIPS

xor.32

Assembly

XOR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

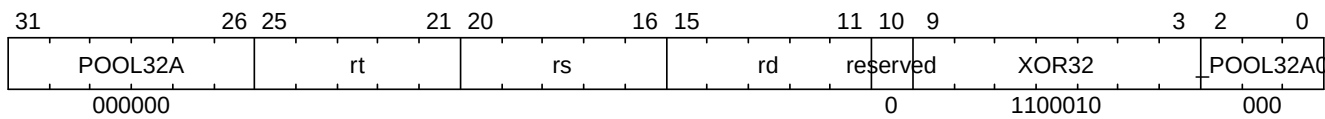


Figure 1718. Encoding (32 bits)

Decode

```
del ctx, exec
return XorOperands(rd=rd, rs=rs, rt=rt)
```

xor.16

Assembly

XOR {rd:gpr3}, {rs:gpr3}

Availability

MMAR < R6

Decode priority

0

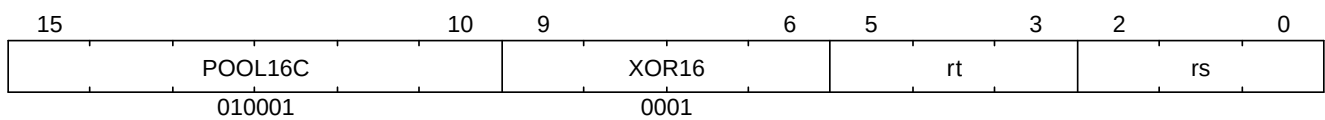


Figure 1719. Encoding (16 bits)

Decode

```
del ctx, exec
target = MICROMIPS_GPR3.decode(rt)
return XorOperands(
    rd=target,
    rs=MICROMIPS_GPR3.decode(rs),
    rt=target,
)
```

xor.16.r6

Assembly

XOR {rd:gpr3}, {rs:gpr3}

Availability

MMAR >= R6

Decode priority

0

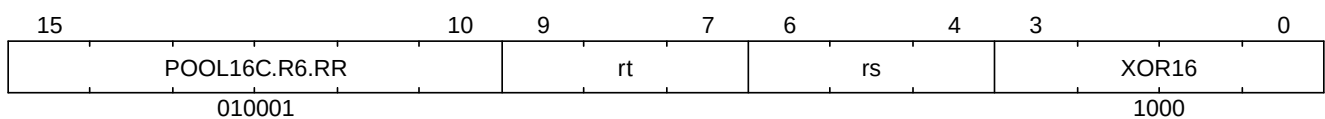


Figure 1720. Encoding (16 bits)

Decode

```
del ctx, exec
```



```

target = MICROMIPS_GPR3.decode(rt)
return XorOperands(
    rd=target,
    rs=MICROMIPS_GPR3.decode(rs),
    rt=target,
)

```

nanoMIPS

xor.32

Assembly

XOR {rd:gpr}, {rs:gpr}, {rt:gpr}

Availability

true

Decode priority

0

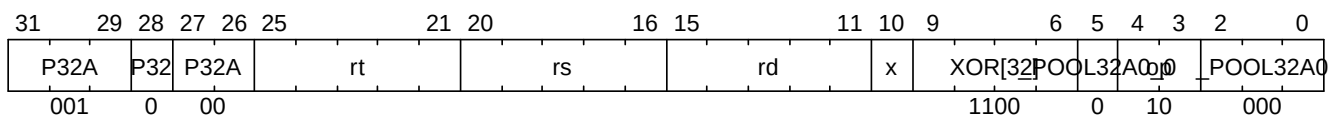


Figure 1721. Encoding (32 bits)

Decode

```

return XorOperands(rd=rd, rs=rs, rt=rt)

```

xor.16

Assembly

XOR {rd:gpr3}, {rs:gpr3}

Availability

true

Decode priority

0

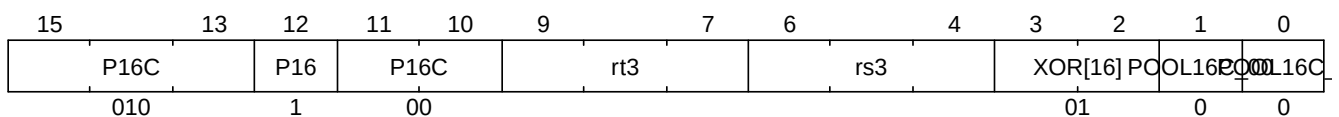


Figure 1722. Encoding (16 bits)

Decode

```

rt = GPR3.decode(rt3)
rs = GPR3.decode(rs3)
return XorOperands(rd=rt, rs=rs, rt=rt)

```

8.621.3. Operation

```
ctx.GPR[operands.rd] = ctx.GPR[operands.rs] ^ ctx.GPR[operands.rt]
```

8.621.4. Description

The nanoMIPS[32] form names an independent destination. The compact nanoMIPS[16] form aliases its encoded rt3 register as both source and destination. The operation is performed across the full implemented GPR width.

8.621.5. Exceptions

None.

8.622. XORI

8.622.1. Purpose

Compute the logical XOR of a GPR and an unsigned immediate.

8.622.2. Formats

MIPS

`xori`

Assembly

`XORI {rt:gpr}, {rs:gpr}, {imm:u16}`

Availability

true

Decode priority

0

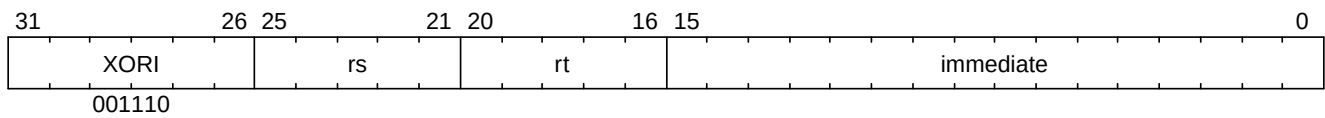


Figure 1723. Encoding (32 bits)

Decode

```
del ctx, exec
return XoriOperands(rt=rt, rs=rs, imm=zero_extend(immediate, from_nbits=16))
```

microMIPS

`xori.32`

Assembly

`XORI {rt:gpr}, {rs:gpr}, {imm:u16}`

Availability

true

Decode priority

0

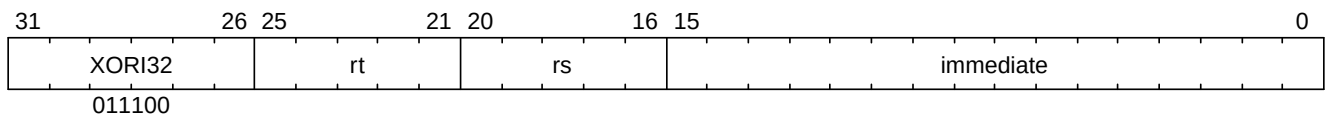


Figure 1724. Encoding (32 bits)

Decode

```
del ctx, exec
return XoriOperands(rt=rt, rs=rs, imm=zero_extend(immediate, from_nbits=16))
```

nanoMIPS

xori.32

Assembly

XORI {rt:gpr}, {rs:gpr}, {imm:u12}

Availability

true

Decode priority

0

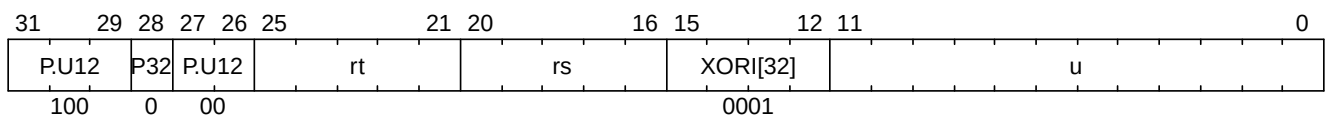


Figure 1725. Encoding (32 bits)

Decode

```
return XoriOperands(rt=rt, rs=rs, imm=zero_extend(u, from_nbits=12))
```

8.622.3. Operation

```
ctx.GPR[operands.rt] = ctx.GPR[operands.rs] ^ int(operands.imm)
```

8.622.4. Description

The immediate is zero-extended and XORed with the full-width source GPR.

8.622.5. Exceptions

None.

Appendix A: MIPS Encoding Summary

This chapter gives a hierarchical view of the instruction encoding space across all declared revisions, features, and processor states. Named entries descend into another opcode pool; multiple names in one cell are overlapping possibilities, and an em dash marks an unallocated encoding.

A.1. MIPS pool

Pool path

MIPS

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[31:26] → selector[5:0]



Figure 1726. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	BREAK / SPECIAL pool	REGIMM pool	J	JAL	BEQ	BNE	4 options	4 options
001???	5 options	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	AUI / LUI
010???	DERET / WAIT / COP0 pool	COP1 pool	—	COP1X pool	BEQL	BNEL	4 options	4 options
011???	4 options	DADDIU	LDL	LDR	SDBBP / SPECIAL2 pool	DAUI / JALX	—	SPECIAL3 pool
100???	LB	LH	LWL	LW	LBU	LHU	LWR	LWU
101???	SB	SH	SWL	SW	SDL	SDR	SWR	CACHE
110???	LL	LWC1	BC	PREF	LLD	LDC1	JIC / POP66 / BEQZC	LD

High / low	???000	???001	???010	???011	???100	???101	???110	???111
111???	SC	SWC1	BALC	4 options	SCD	SDC1	JIALC / POP76 / BNEZC	SD

A.1.1. Selector 000110

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BGEUC
- BGEZALC
- BLEZ
- BLEZALC

A.1.2. Selector 000111

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BGTZ
- BGTZALC
- BLTUC
- BLTZALC

A.1.3. Selector 001000

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- ADDI
- BEQZALC
- BOVC
- POP10
- BEQC

A.1.4. Selector 010110

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BGEC
- BGEZC

- BLEZC
- BLEZL

A.1.5. Selector 010111

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BGTZC
- BGTZL
- BLTC
- BLTZC

A.1.6. Selector 011000

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BNEC
- BNEZALC
- BNVC
- DADDI

A.1.7. Selector 111011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDIUPC
- LDPC
- PCREL pool
- PCREL.LS pool

A.2. SPECIAL pool

Pool path

MIPS / SPECIAL

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

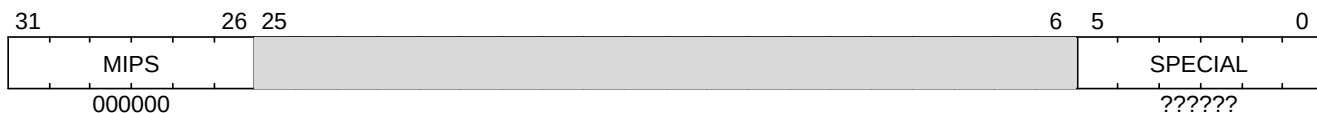


Figure 1727. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	SLL / NOP / PAUSE	MOVF / MOVT	ROTR / SRL	SRA	SLLV	LSA	ROTRV / SRLV	SRAV
001???	JR / JR.HB	JALR / JALR.HB	MOVZ	MOVN	SYSCALL	—	SDBBP	SYNC
010???	CLZ.R6 / MFHI / CLZ	CLO.R6 / MTHI / CLO	DCLZ.R6 / MFLO / DCLZ	DCLO.R6 / MTLO / DCLO	DSLLV	DLSA	DROTRV / DSRLV	DSRAV
011???	4 options	MUHU / MULTU / MULU	DIV / DIV.R6 / MOD	DIVU / DIVU.R6 / MODU	5 options	5 options	4 options	4 options
100???	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
101???	—	—	SLT	SLTU	DADD	DADDU	DSUB	DSUBU
110???	TGE	TGEU	TLT	TLTU	TEQ	SELEQZ	TNE	SELNEZ
111???	DSLL	—	DROTR / DSRL	DSRA	DSLL32	—	DROTR32 / DSRL32	DSRA32

A.2.1. Selector 011000

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MUH
- MUL.R6
- MULT
- MUL

A.2.2. Selector 011100

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- DMUH.R6
- DMUL.R6
- DMULT

- DMUH
- DMUL

A.2.3. Selector 011101

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- DMUHU.R6
- DMULTU
- DMULU.R6
- DMUHU
- DMULU

A.2.4. Selector 011110

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- DDIV
- DDIV.R6
- DMOD.R6
- DMOD

A.2.5. Selector 011111

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- DDIVU
- DDIVU.R6
- DMODU.R6
- DMODU

A.3. REGIMM pool

Pool path

MIPS / REGIMM

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:16] → selector[4:0]

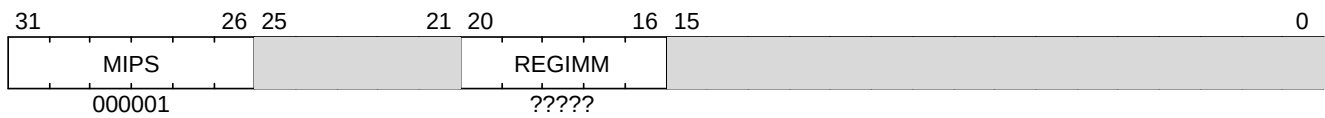


Figure 1728. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	BLTZ	BGEZ	BLTZL	BGEZL	—	—	DAHI	—
01???	TGEI	TGEIU	TLTI	TLTIU	TEQI	—	TNEI	—
10???	BLTZAL / NAL	BAL / BGEZAL	BLTZALL	BGEZALL	—	—	—	SIGRIE
11???	BPOSGE32C	—	—	—	BPOSGE32	—	DATI	SYNCI

A.4. COP0 pool

Pool path

MIPS / COP0

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[25:21] → selector[4:0]

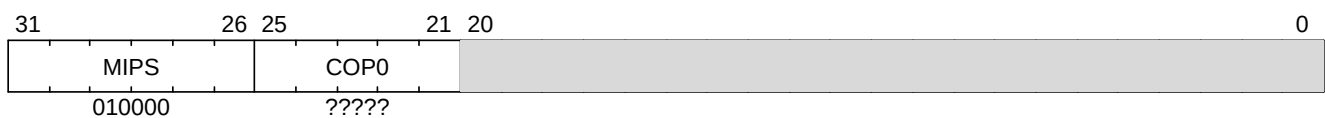


Figure 1729. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	MFC0	DMFC0	MFHC0	—	MTC0	DMTC0	MTHC0	—
01???	—	—	RDPGPR	4 options	—	—	WRPGPR	—
10???	COP0.CO pool	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

A.4.1. Selector 01011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- DI
- DVP
- EI
- EVP

A.5. COP0.CO pool

Pool path

MIPS / COP0 / COP0.CO

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

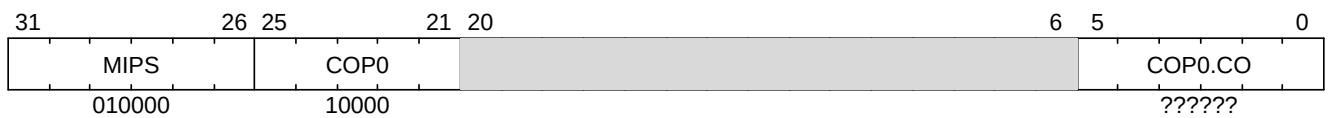


Figure 1730. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	TLBR	TLBWI	TLBINV	TLBINVF	—	TLBWR	—
001???	TLBP	—	—	—	—	—	—	—
010???	—	—	—	—	—	—	—	—
011???	ERET / ERETNC	—	—	—	—	—	—	—
100???	—	—	—	—	—	—	—	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

A.6. COP1 pool

Pool path

MIPS / COP1

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[25:21] → selector[4:0]

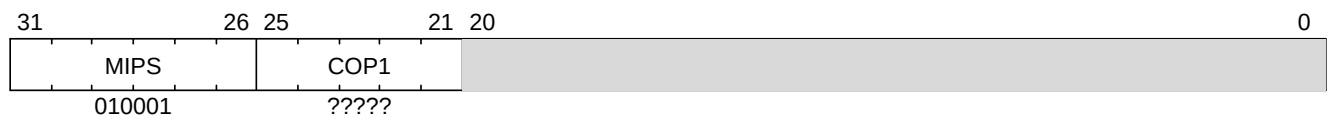


Figure 1731. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	MFC1	DMFC1	CFC1	MFHC1	MTC1	DMTC1	CTC1	MTHC1
01???	5 options	BC1EQZ	—	—	—	BC1NEZ	—	—
10???	COP1.CO ND.S pool / COP1.S pool	COP1.CO ND.D pool / COP1.D pool	—	—	COP1.CM P.S pool / COP1.FM T20 pool	COP1.CM P.D pool / COP1.FM T21 pool	COP1.CO ND.PS pool / COP1.PS pool	—
11???	—	—	—	—	—	—	—	—

A.6.1. Selector 01000

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- BC1
- BC1F
- BC1FL
- BC1T
- BC1TL

A.7. COP1.COND.S pool

Pool path

MIPS / COP1 / COP1.COND.S

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:4] → selector[1:0]



Figure 1732. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	—	C.COND.S / C.cond.S

A.8. COP1.S pool

Pool path

MIPS / COP1 / COP1.S

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

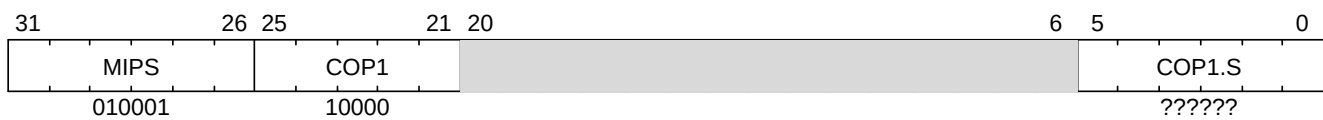


Figure 1733. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	ADD.S	SUB.S	MUL.S	DIV.S	SQRT.S	ABS.S	MOV.S	NEG.S
001???	ROUND.L.S	TRUNC.L.S	CEIL.L.S	FLOOR.L.S	ROUND.W.S	TRUNC.W.S	CEIL.W.S	FLOOR.W.S
010???	SEL.S	4 options	MOVZ / MOVZ.S	MOVN / MOVN.S	SELEQZ.S	RECIP.S	RSQRT.S	SELNEZ.S
011???	MADDF.S	MSUBF.S	RINT.S	CLASS.S	MIN.S	MINA.S	MAX.S	MAXA.S

High / low	???000	???001	???010	???011	???100	???101	???110	???111
100???	—	CVT.D.S	—	—	CVT.W.S	CVT.L.S	CVT.PS.S	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

A.8.1. Selector 010001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MOVF
- MOVT
- MOVF.S
- MOVT.S

A.9. COP1.COND.D pool

Pool path

MIPS / COP1 / COP1.COND.D

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:4] → selector[1:0]



Figure 1734. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	—	C.COND.D / C.cond.D

A.10. COP1.D pool

Pool path

MIPS / COP1 / COP1.D

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

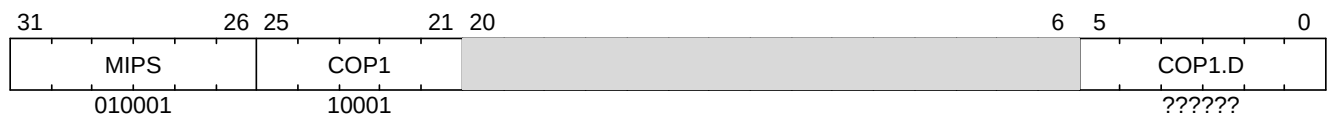


Figure 1735. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	ADD.D	SUB.D	MUL.D	DIV.D	SQRT.D	ABS.D	MOV.D	NEG.D
001???	ROUND.L.D	TRUNC.L.D	CEIL.L.D	FLOOR.L.D	ROUND.W.D	TRUNC.W.D	CEIL.W.D	FLOOR.W.D
010???	SEL.D	4 options	MOVZ / MOVZ.D	MOVN / MOVN.D	SELEQZ.D	RECIP.D	RSQRT.D	SELNEZ.D
011???	MADDF.D	MSUBF.D	RINT.D	CLASS.D	MIN.D	MINA.D	MAX.D	MAXA.D
100???	CVT.S.D	—	—	—	CVT.W.D	CVT.L.D	—	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

A.10.1. Selector 010001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MOVF
- MOVT
- MOVF.D
- MOVT.D

A.11. COP1.CMP.S pool

Pool path

MIPS / COP1 / COP1.CMP.S

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:5] → selector[0:0]

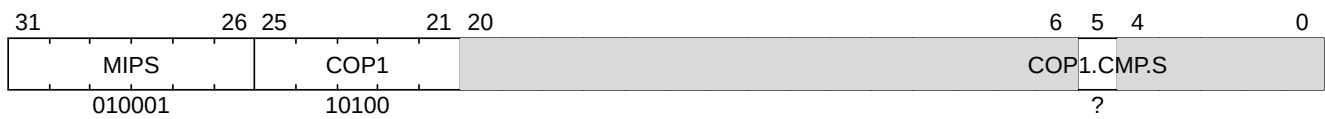


Figure 1736. 32-bit opcode fields

High / low	0	1
?	CMP.S	—

A.12. COP1.FMT20 pool

Pool path

MIPS / COP1 / COP1.FMT20

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

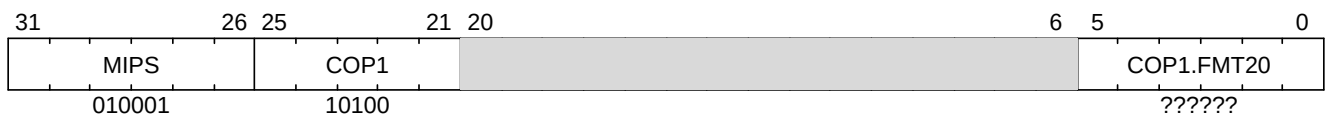


Figure 1737. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	—	—	—	—	—	—	—
001???	—	—	—	—	—	—	—	—
010???	—	—	—	—	—	—	—	—
011???	—	—	—	—	—	—	—	—
100???	CVT.S.W	CVT.D.W	—	—	—	—	—	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

A.13. COP1.CMP.D pool

Pool path

MIPS / COP1 / COP1.CMP.D

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:5] → selector[0:0]

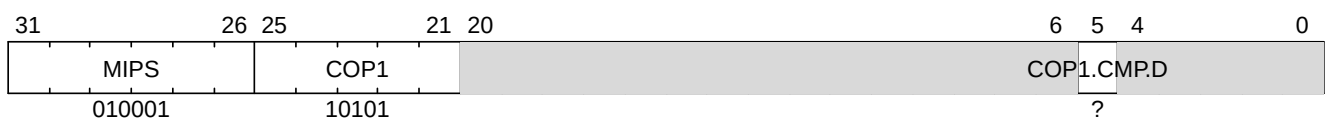


Figure 1738. 32-bit opcode fields

High / low	0	1
?	CMP.D	—

A.14. COP1.FMT21 pool

Pool path

MIPS / COP1 / COP1.FMT21

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

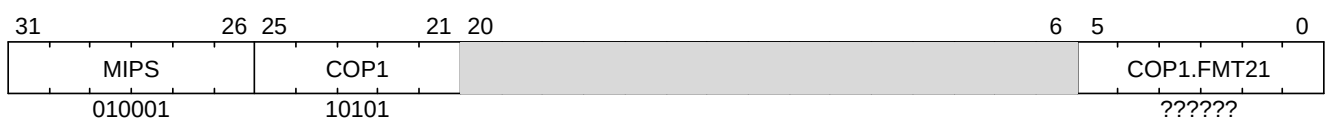


Figure 1739. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	—	—	—	—	—	—	—
001???	—	—	—	—	—	—	—	—
010???	—	—	—	—	—	—	—	—

High / low	???000	???001	???010	???011	???100	???101	???110	???111
011???	—	—	—	—	—	—	—	—
100???	CVT.S.L	CVT.D.L	—	—	—	—	—	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

A.15. COP1.COND.PS pool

Pool path

MIPS / COP1 / COP1.COND.PS

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:4] → selector[1:0]

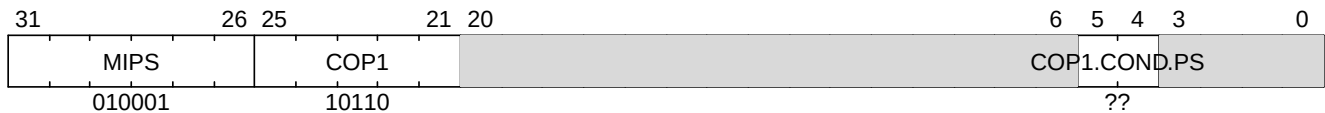


Figure 1740. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	—	C.COND.PS / C.cond.PS

A.16. COP1.PS pool

Pool path

MIPS / COP1 / COP1.PS

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

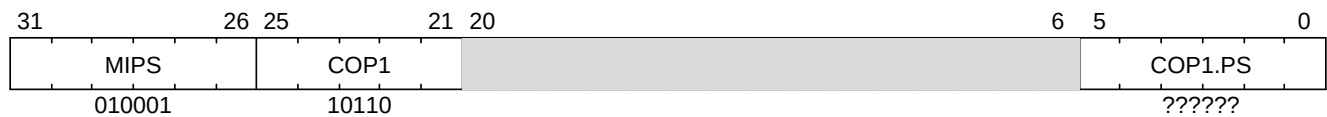


Figure 1741. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	ADD / ADD.PS	SUB / SUB.PS	MUL / MUL.PS	—	—	ABS / ABS.PS	MOV / MOV.PS	NEG / NEG.PS
001???	—	—	—	—	—	—	—	—
010???	—	4 options	MOVZ / MOVZ.PS	MOVN / MOVN.PS	—	—	—	—
011???	—	—	—	—	—	—	—	—
100???	CVT.S.PU	—	—	—	—	—	—	—
101???	CVT.S.PL	—	—	—	PLL / PLL.PS	PLU / PLU.PS	PUL / PUL.PS	PUU / PUU.PS
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

A.16.1. Selector 010001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MOVF
- MOVT
- MOVF.PS
- MOVT.PS

A.17. COP1X pool

Pool path

MIPS / COP1X

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

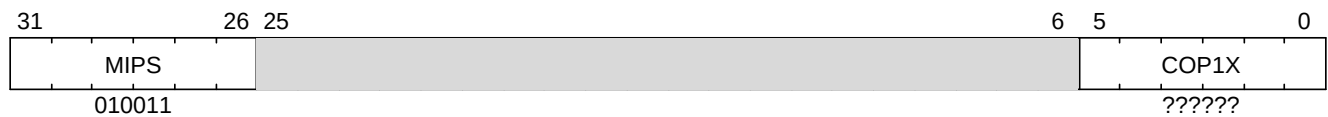


Figure 1742. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	LWXC1	LDXC1	—	—	—	LUXC1	—	—
001???	SWXC1	SDXC1	—	—	—	SUXC1	—	PREFX
010???	—	—	—	—	—	—	—	—
011???	—	—	—	—	—	—	ALNV.PS	—
100???	MADD / MADD.S	MADD / MADD.D	—	—	—	—	MADD / MADD.PS	—
101???	MSUB / MSUB.S	MSUB / MSUB.D	—	—	—	—	MSUB / MSUB.PS	—
110???	NMADD / NMADD.S	NMADD / NMADD.D	—	—	—	—	NMADD / NMADD.PS	—
111???	NMSUB / NMSUB.S	NMSUB / NMSUB.D	—	—	—	—	NMSUB / NMSUB.PS	—

A.18. SPECIAL2 pool

Pool path

MIPS / SPECIAL2

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

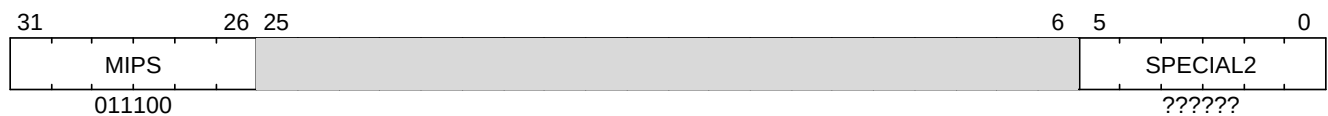


Figure 1743. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	MADD	MADDU	MUL	—	MSUB	MSUBU	—	—
001???	—	—	—	—	—	—	—	—
010???	—	—	—	—	—	—	—	—

High / low	???000	???001	???010	???011	???100	???101	???110	???111
011???	—	—	—	—	—	—	—	—
100???	CLZ	CLO	—	—	DCLZ	DCLO	—	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

A.19. SPECIAL3 pool

Pool path

MIPS / SPECIAL3

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

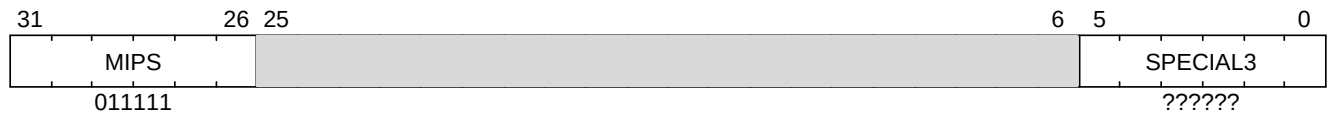


Figure 1744. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	EXT	DEXTM	DEXTU	DEXT	INS	DINSM	DINSU	DINS
001???	—	—	1 options	—	INSV	—	—	SPECIAL3 .CRC pool
010???	ADDU.QB pool	9 options	16 options	22 options	—	—	—	—
011???	ADDUH.Q B pool	—	—	—	—	—	—	—
100???	5 options	—	—	—	4 options	CACHE.R6 / CACHE	SC / SCWP	SCD / SCDP
101???	—	—	—	—	—	—	—	—
110???	DPA.W.P H pool	APPEND pool	—	—	—	PREF	LL / LLWP	LLD / LLDP

High / low	???000	???001	???010	???011	???100	???101	???110	???111
111???	EXTR.W pool	—	—	RDHWR	—	—	—	—

A.19.1. Selector 001010

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- SPECIAL3.INDEXED pool

A.19.2. Selector 010001

This selector has 9 possible interpretations across the declared revisions, features, and processor states:

- PACKRL.PH
- PRECR.QB.PH
- PRECR_SRA.PH.W
- PRECR_SRA_R.PH.W
- PRECRQ.PH.W
- PRECRQ.QB.PH
- PRECRQ_RS.PH.W
- PRECRQU_S.QB.PH
- CMPU.EQ.QB pool

A.19.3. Selector 010010

This selector has 16 possible interpretations across the declared revisions, features, and processor states:

- BITREV
- PRECEQ.W.PHL
- PRECEQ.W.PHR
- PRECEQU.PH.QBL
- PRECEQU.PH.QBLA
- PRECEQU.PH.QBR
- PRECEQU.PH.QBRA
- PRECEU.PH.QBL
- PRECEU.PH.QBLA

- PRECEU.PH.QBR
- PRECEU.PH.QBRA
- REPL.PH
- REPL.QB
- REPLV.PH
- REPLV.QB
- ABSQ_S.PH pool

A.19.4. Selector 010011

This selector has 22 possible interpretations across the declared revisions, features, and processor states:

- SHLL.PH
- SHLL.QB
- SHLL_S.PH
- SHLL_S.W
- SHLLV.PH
- SHLLV.QB
- SHLLV_S.PH
- SHLLV_S.W
- SHRA.PH
- SHRA.QB
- SHRA_R.PH
- SHRA_R.QB
- SHRA_R.W
- SHRAV.PH
- SHRAV.QB
- SHRAV_R.PH
- SHRAV_R.QB
- SHRAV_R.W
- SHRL.PH
- SHRL.QB
- SHRLV.PH
- SHRLV.QB

A.19.5. Selector 100000

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- ALIGN
- BITSWAP
- SEB
- SEH
- WSBH

A.19.6. Selector 100100

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- DALIGN
- DBITSWAP
- DSBH
- DSHD

A.20. SPECIAL3.INDEXED pool

Pool path

MIPS / SPECIAL3 / SPECIAL3.INDEXED

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

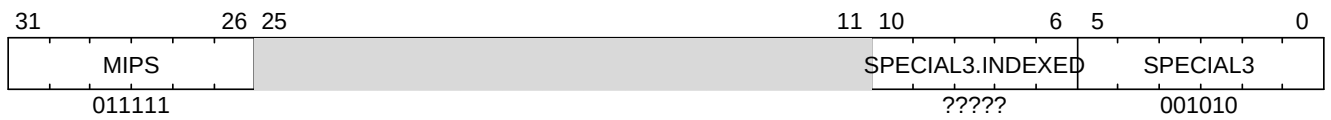


Figure 1745. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	LWX	—	—	—	LHX	—	LBUX	—
01???	—	—	—	—	—	—	—	—

High / low	??000	??001	??010	??011	??100	??101	??110	??111
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

A.21. SPECIAL3.CRC pool

Pool path

MIPS / SPECIAL3 / SPECIAL3.CRC

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:8] → selector[2:0]

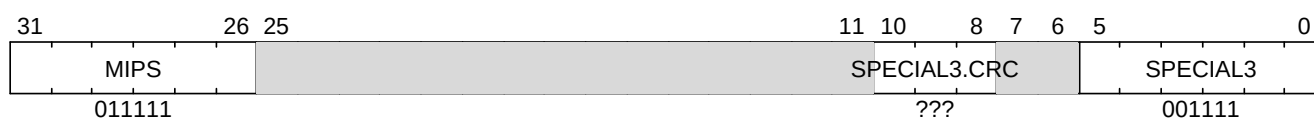


Figure 1746. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	CRC32D	CRC32CD	—	—	—	—	—	—

A.22. ADDU.QB pool

Pool path

MIPS / SPECIAL3 / ADDU.QB

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

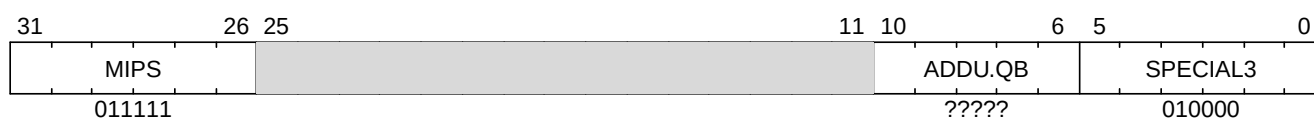


Figure 1747. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	ADDU.QB[D SP] / ADDU.QB	SUBU.QB	—	—	ADDU_S.QB	SUBU_S.QB	1 options	1 options
01???	ADDU.PH	SUBU.PH	ADDQ.PH	SUBQ.PH	ADDU_S.PH	SUBU_S.PH	ADDQ_S.PH	SUBQ_S.PH
10???	ADDSC	ADDWC	MODSUB	—	RADDU.W.QB	—	ADDQ_S.W	SUBQ_S.W
11???	—	—	—	—	1 options	1 options	MULQ_S.PH	MULQ_RS.PH

A.22.1. Selector 00110

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEU_S.PH.QBL

A.22.2. Selector 00111

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEU_S.PH.QBR

A.22.3. Selector 11100

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEQ_S.W.PHL

A.22.4. Selector 11101

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEQ_S.W.PHR

A.23. CMPU.EQ.QB pool

Pool path

MIPS / SPECIAL3 / CMPU.EQ.QB

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

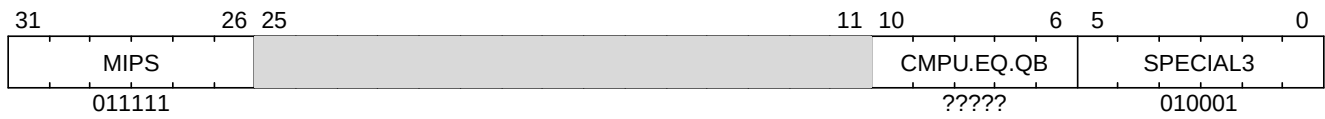


Figure 1748. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	2 options	CMPU.LT.QB	CMPU.LE.QB	PICK.QB	CMPGU.EQ.QB	CMPGU.LT.QB	CMPGU.LE.QB	—
01???	CMP.EQ.PH	CMP.LT.PH	CMP.LE.PH	PICK.PH	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	CMPGDU.EQ.QB	CMPGDU.LT.QB	CMPGDU.LE.QB	—	—	—	—	—

A.23.1. Selector 00000

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- CMPU.EQ.QB[DSP]
- CMPU.EQ.QB

A.24. ABSQ_S.PH pool

Pool path

MIPS / SPECIAL3 / ABSQ_S.PH

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

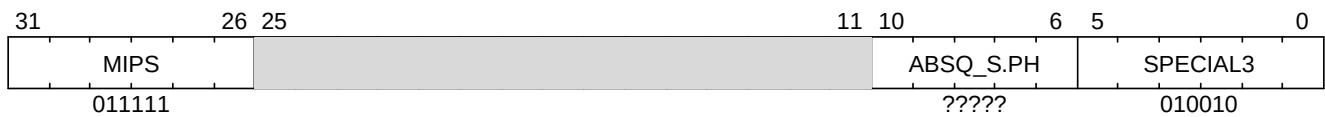


Figure 1749. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	ABSQ_S.QB	—	—	—	—	—	—
01???	—	2 options	—	—	—	—	—	—
10???	—	ABSQ_S.W	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

A.24.1. Selector 01001

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- ABSQ_S.PH[DSP]
- ABSQ_S.PH

A.25. ADDUH.QB pool

Pool path

MIPS / SPECIAL3 / ADDUH.QB

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

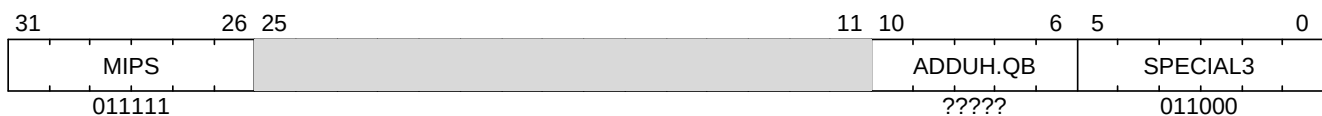


Figure 1750. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	2 options	SUBUH.QB	ADDUH_R.QB	SUBUH_R.QB	—	—	—	—
01???	ADDQH.PH	SUBQH.PH	ADDQH_R.PH	SUBQH_R.PH	MUL.PH	—	MUL_S.PH	—
10???	ADDQH.W	SUBQH.W	ADDQH_R.W	SUBQH_R.W	—	—	MULQ_S.W	MULQ_RS.W
11???	—	—	—	—	—	—	—	—

A.25.1. Selector 00000

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- ADDUH.QB[DSP]
- ADDUH.QB

A.26. DPA.W.PH pool

Pool path

MIPS / SPECIAL3 / DPA.W.PH

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

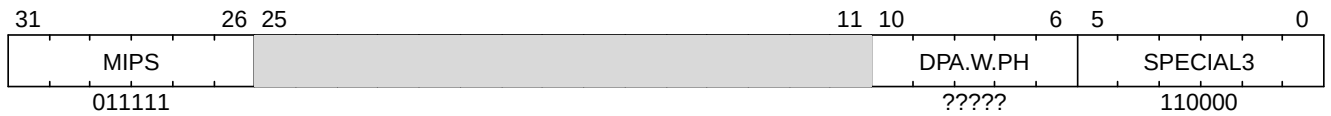


Figure 1751. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	2 options	2 options	2 options	2 options	2 options	2 options	2 options	2 options
01???	2 options	2 options	—	2 options	2 options	2 options	—	2 options
10???	2 options	—	2 options	—	2 options	—	2 options	—
11???	2 options	2 options	2 options	2 options	—	—	—	—

A.26.1. Selector 00000

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPA.W.PH

A.26.2. Selector 00001

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPS.W.PH

A.26.3. Selector 00010

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- MULSA.W.PH

A.26.4. Selector 00011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPAU.H.QBL

A.26.5. Selector 00100

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPAQ_S.W.PH

A.26.6. Selector 00101

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPSQ_S.W.PH

A.26.7. Selector 00110

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- MULSAQ_S.W.PH

A.26.8. Selector 00111

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPAU.H.QBR

A.26.9. Selector 01000

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPAX.W.PH

A.26.10. Selector 01001

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPSX.W.PH

A.26.11. Selector 01011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPSU.H.QBL

A.26.12. Selector 01100

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPAQ_SA.L.W

A.26.13. Selector 01101

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPSQ_SA.L.W

A.26.14. Selector 01111

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPSU.H.QBR

A.26.15. Selector 10000

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- MAQ_SA.W.PHL

A.26.16. Selector 10010

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- MAQ_SA.W.PHR

A.26.17. Selector 10100

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- MAQ_S.W.PHL

A.26.18. Selector 10110

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- MAQ_S.W.PHR

A.26.19. Selector 11000

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPAQX_S.W.PH

A.26.20. Selector 11001

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPSQX_S.W.PH

A.26.21. Selector 11010

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPAQX_SA.W.PH

A.26.22. Selector 11011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- DPA.W.PH[DSP]
- DPSQX_SA.W.PH

A.27. APPEND pool

Pool path

MIPS / SPECIAL3 / APPEND

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

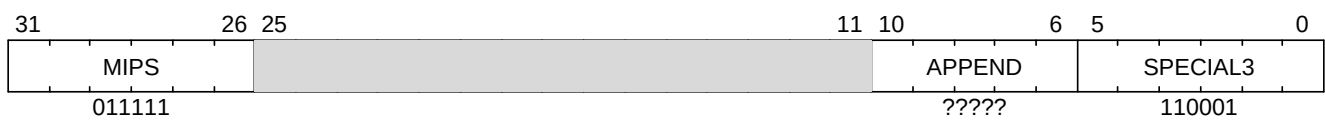


Figure 1752. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	APPEND[DS P] / APPEND	PREPEND	—	—	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	BALIGN	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

A.28. EXTR.W pool

Pool path

MIPS / SPECIAL3 / EXTR.W

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:6] → selector[4:0]

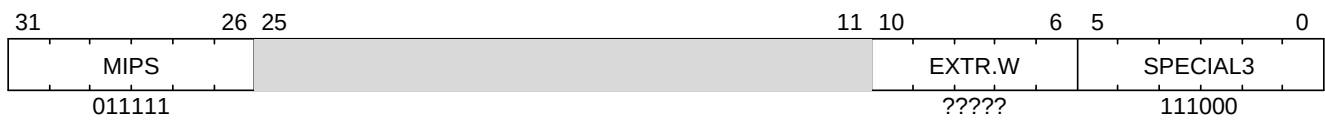


Figure 1753. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	EXTR.W[DS P] / EXTR.W	EXTRV.W	EXTP	EXTPV	EXTR_R.W	EXTRV_R.W	EXTR_RS.W	EXTRV_RS.W
01???	—	—	EXTPDP	EXTPDPV	—	—	EXTR_S.H	EXTRV_S.H
10???	—	—	RDDSP	WRDSP	—	—	—	—
11???	—	—	SHILO	SHILOV	—	—	—	MTHLIP

A.29. PCREL pool

Pool path

MIPS / PCREL

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:16] → selector[4:0]

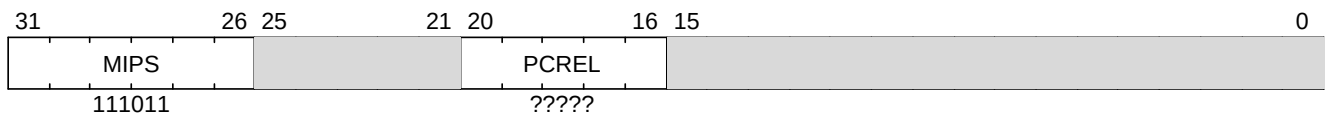


Figure 1754. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	AUIPC	ALUIPC

A.30. PCREL.LS pool

Pool path

MIPS / PCREL.LS

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:19] → selector[1:0]

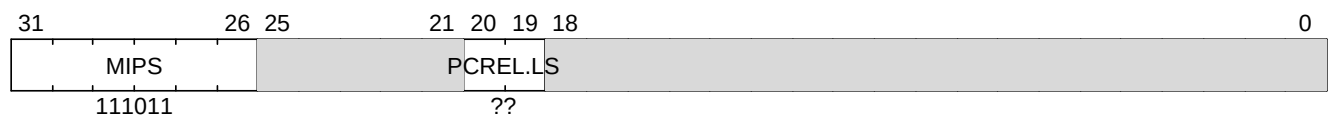


Figure 1755. 32-bit opcode fields

High / low	00	01	10	11
??	—	LWPC	LWUPC	—

Appendix B: MIPS16E Encoding Summary

This chapter gives a hierarchical view of the instruction encoding space across all declared revisions, features, and processor states. Named entries descend into another opcode pool; multiple names in one cell are overlapping possibilities, and an em dash marks an unallocated encoding.

B.1. MIPS16E pool

Pool path

MIPS16E

Selector width

5 bits

Encoding width

16, 32 bits

Selector layout

16-bit: instruction[15:11] → selector[4:0]; 32-bit: instruction[31:27] → selector[4:0]

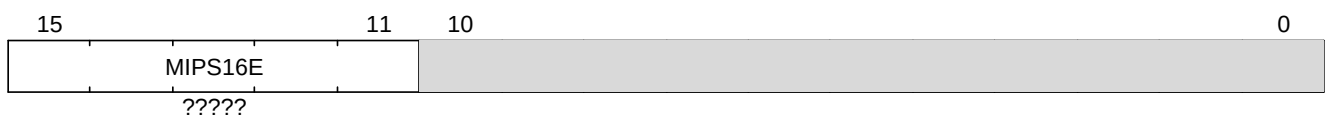


Figure 1756. 16-bit opcode fields



Figure 1757. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	ADDIUSP / ADDIU	ADDIUPC / ADDIU	—	—	—	—	—	—
01???	RRI-A / ADDIU	ADDIU8 / ADDIU	—	—	I8 pool	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	EXTEND pool	—

B.2. I8 pool

Pool path

MIPS16E / I8

Selector width

3 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[10:8] → selector[2:0]

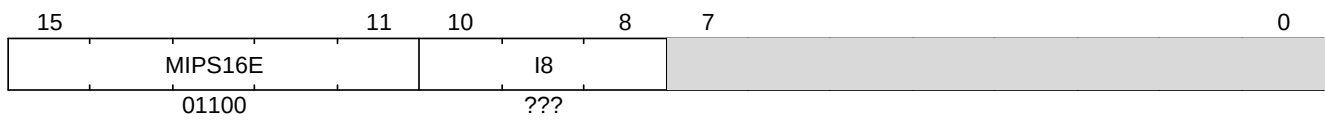


Figure 1758. 16-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	—	—	ADJSP / ADDIU	—	MOV32R / NOP	—	—

B.3. EXTEND pool

Pool path

MIPS16E / EXTEND

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:11] → selector[4:0]

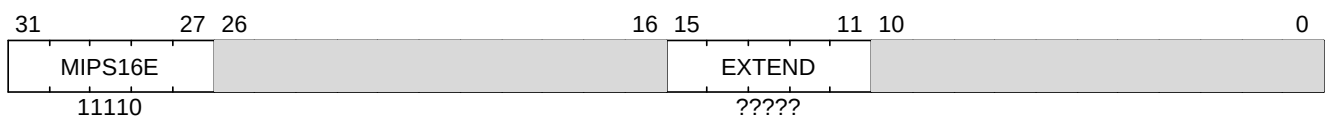


Figure 1759. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	ADDIUSP / ADDIU	ADDIUPC / ADDIU	—	—	—	—	—	—
01???	RRI-A / ADDIU	ADDIU8 / ADDIU	—	—	EXTEND.I 8 pool	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

B.4. EXTEND.I8 pool

Pool path

MIPS16E / EXTEND / EXTEND.I8

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:8] → selector[2:0]

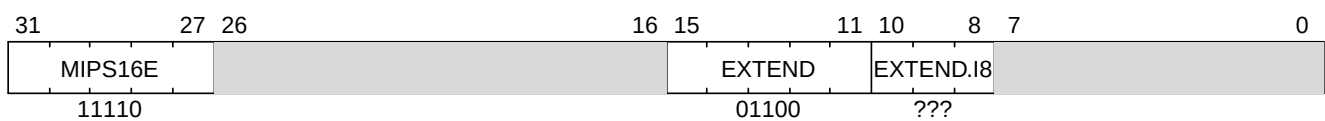


Figure 1760. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	—	—	ADJSP / ADDIU	—	—	—	—

Appendix C: microMIPS Encoding Summary

This chapter gives a hierarchical view of the instruction encoding space across all declared revisions, features, and processor states. Named entries descend into another opcode pool; multiple names in one cell are overlapping possibilities, and an em dash marks an unallocated encoding.

C.1. microMIPS pool

Pool path

microMIPS

Selector width

6 bits

Encoding width

16, 32 bits

Selector layout

16-bit: instruction[15:10] → selector[5:0]; 32-bit: instruction[31:26] → selector[5:0]

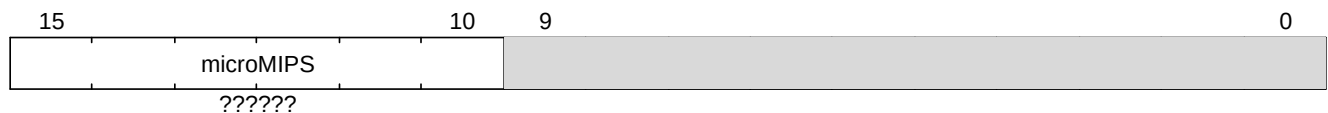


Figure 1761. 16-bit opcode fields

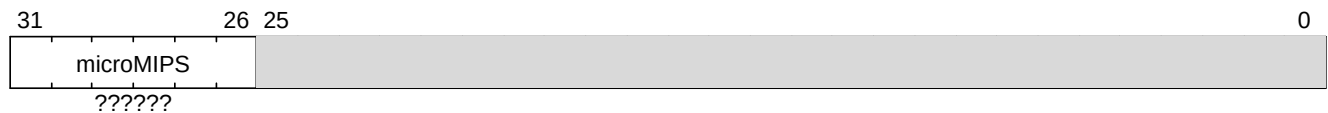


Figure 1762. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	11 options	5 options	LBU16 / LBU	MOVE16 / MOVE	4 options	LBU32 / LBU	SB32 / SB	LB32 / LB
001???	8 options	4 options	LHU16 / LHU	ANDI16 / ANDI	ADDIU32 / ADDIU	LHU32 / LHU	SH32 / SH	LH32 / LH
010???	10 options	12 options	LWSP16 / LW	ADDIURS5 / ADDIUSP / ADDIU	ORI32 / ORI	7 options	DALIGN / POOL32S pool	DADDIU32 / DADDIU
011???	29 options	LWGP / LW	LW16 / LW	3 options	XORI32 / XORI	6 options	5 options	4 options
100???	BEQZC32 / JIC / BEQZC	MOVEP	SB16 / SB	4 options	SLTI32 / SLTI	4 options	SWC1	LWC1

High / low	???000	???001	???010	???011	???100	???101	???110	???111
101???	BNEZC32 / JIALC / BNEZC	—	SH16 / SH	4 options	SLTIU32 / SLTIU	BALC / BNE32 / BNE	SDC1	LDC1
110???	BGEUC / BGEZALC / BLEZALC	—	SWSP16 / SW	4 options	ANDI32 / ANDI	5 options	SD32 / SD	LD32 / LD
111???	BGTZALC / BLTUC / BLTZALC	—	SW16 / SW	LI16 / LI	DAUI / JALX32 / JALX	6 options	SW32 / SW	LW32 / LW

C.1.1. Selector 000000

This selector has 11 possible interpretations across the declared revisions, features, and processor states:

- CFC2
- COP2.32
- CTC2
- MFC2
- MFHC2
- MTC2
- MTHC2
- WAIT32
- POOL32A pool
- COP2
- WAIT

C.1.2. Selector 000001

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- ADDU.16
- ADDU.16.R6
- POOL16A
- ADDU
- SUBU

C.1.3. Selector 000100

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDI32
- AUI32
- ADDI
- AUI

C.1.4. Selector 001000

This selector has 8 possible interpretations across the declared revisions, features, and processor states:

- CACHE.32
- LDC2
- LWC2
- SDC2
- SWC2
- POOL32B.MULTI32 pool
- POOL32B.PAIR32 pool
- CACHE

C.1.5. Selector 001001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- SLL16
- SRL16
- SLL
- SRL

C.1.6. Selector 010000

This selector has 10 possible interpretations across the declared revisions, features, and processor states:

- BC2EQZC
- BC2F
- BC2NEZC
- BC2T

- DAHI/DATI
- POOL32B.BRANCH pool
- POOL32I pool
- REGIMM32 pool
- DAHI
- DATI

C.1.7. Selector 010001

This selector has 12 possible interpretations across the declared revisions, features, and processor states:

- LWM
- MOVEP.R6
- SWM
- POOL16C pool
- POOL16C.LOGICAL pool
- POOL16C.LOGICAL_R6 pool
- POOL16C.MULTI pool
- POOL16C.R6.BREAK pool
- POOL16C.R6.JR pool
- POOL16C.R6.JRSP pool
- POOL16C.R6.RR pool
- MOVEP

C.1.8. Selector 010101

This selector has 7 possible interpretations across the declared revisions, features, and processor states:

- LDXC1
- LUXC1
- LWXC1
- SDXC1
- SUXC1
- SWXC1
- POOL32F pool

C.1.9. Selector 011000

This selector has 29 possible interpretations across the declared revisions, features, and processor states:

- LDL32
- LDR32
- LLD32
- LLDP32
- LWL32
- LWR32
- PREF32
- PREFE32
- SCD32
- SCDP32
- SDL32
- SDR32
- SWL32
- SWR32
- POOL32B pool
- POOL32C pool
- LDL
- LDR
- LLD
- LLDP
- LWL
- LWR
- PREF
- SCD
- SCDP
- SDL
- SDR
- SWL
- SWR

C.1.10. Selector 011011

This selector has 3 possible interpretations across the declared revisions, features, and processor states:

- ADDIU.R1SP.16
- ADDIU.R2.16
- ADDIU

C.1.11. Selector 011101

This selector has 6 possible interpretations across the declared revisions, features, and processor states:

- BEQZALC
- BOVC
- JALS32
- POP35
- BEQC
- JALS

C.1.12. Selector 011110

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- ADDIUPCR3
- PCREL
- R6PCREL pool
- ADDIUPC
- AUIPC

C.1.13. Selector 011111

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BNEC32
- BNEZALC
- BNVC
- BNEC

C.1.14. Selector 100011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BEQZ16
- BEQZC16
- BEQ
- BEQZC

C.1.15. Selector 100101

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BC32
- BEQ32
- BC
- BEQ

C.1.16. Selector 101011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- BNEZ16
- BNEZC16
- BNE
- BNEZC

C.1.17. Selector 110011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- B16
- BC16
- BC
- BEQ

C.1.18. Selector 110101

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- BGTZC
- BLTC
- BLTZC
- J32
- J

C.1.19. Selector 111101

This selector has 6 possible interpretations across the declared revisions, features, and processor states:

- BGEC
- BGEZC32
- BLEZC
- JAL32
- BGEZC
- JAL

C.2. POOL32A pool

Pool path

microMIPS / POOL32A

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[2:0] → selector[2:0]

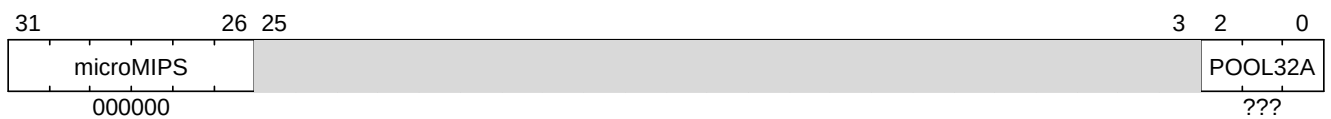


Figure 1763. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	_POOL32 A0 pool	—	—	—	42 options	_POOL32 A5 pool	—	_POOL32 A7 pool

C.2.1. Selector 100

This selector has 42 possible interpretations across the declared revisions, features, and processor states:

- BITSWAP
- RDHWR32
- RDPGPR32
- SEB32
- SEH32
- SYNC32
- SYSCALL32
- TEQ32
- TGE32
- TGEU32
- TLBINV32
- TLBINVF32
- TLBP32
- TLBR32
- TLBWI32
- TLBWR32
- TLT32
- TLTU32
- TNE32
- WRPGPR32
- WSBH32
- [_POOL32A4 pool](#)
- RDHWR
- RDPGPR
- SEB
- SEH
- SYNC
- SYSCALL
- TEQ
- TGE
- TGEU

- TLBINV
- TLBINVF
- TLBP
- TLBR
- TLBWI
- TLBWR
- TLT
- TLTU
- TNE
- WRPGPR
- WSBH

C.3. _POOL32A0 pool

Pool path

microMIPS / POOL32A / _POOL32A0

Selector width

7 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:3] → selector[6:0]

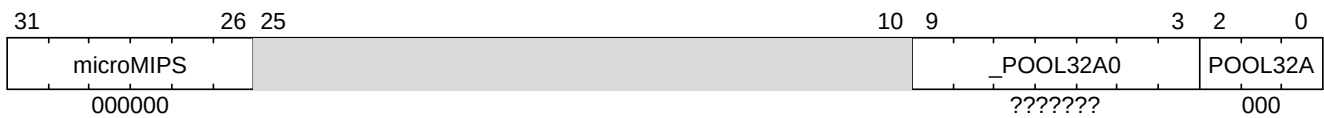


Figure 1764. 32-bit opcode fields

High / low	0000000	0000001	0000010	0000011	0000100	0000101	0000110	0000111
0000???	SLL32 / NOP / SLL	—	SLLV32 / SLLV	4 options	—	—	—	—
0001???	SRL32 / SRL	—	SRLV32 / SRLV	MOVZ32 / MUH / MOVZ	—	—	—	—
0010???	SRA32 / SRA	—	SRAV32 / SRAV	MULU	—	—	—	—
0011???	ROTR32 / ROTR	—	ROTRV32 / ROTRV	MUHU	—	—	—	—

High / low	0000000	0000001	0000010	0000011	0000100	0000101	0000110	0000111
0100???	—	—	ADD32 / ADD	DIV.R6 / LWXS / DIV	—	—	—	—
0101???	SELEQZ	—	ADDU32 / ADDU	MOD	—	—	—	—
0110???	SELNEZ	—	SUB32 / SUB	DIVU.R6 / DIVU	—	—	—	—
0111???	RDHWR32 / RDHWR	—	SUBU32 / SUBU	MODU	—	—	—	—
1000???	—	—	MUL32 / MUL	—	—	—	—	—
1001???	—	—	AND32 / AND	—	—	—	—	—
1010???	—	—	OR32 / OR	—	—	—	—	—
1011???	—	—	NOR32 / NOR	—	—	—	—	—
1100???	—	—	XOR32 / XOR	—	—	—	—	—
1101???	—	—	SLT32 / SLT	—	—	—	—	—
1110???	—	—	SLTU32 / SLTU	—	—	—	—	—
1111???	—	—	—	—	—	—	—	—

C.3.1. Selector 0000011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MOVN32
- MUL.R6
- MOVN
- MUL

C.4. _POOL32A4 pool

Pool path

microMIPS / POOL32A / _POOL32A4

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:3] → selector[2:0]

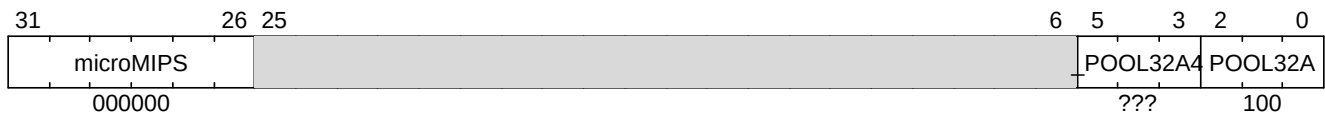


Figure 1765. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	INS32 / INS	—	—	—	EXT32 / EXT	POOL32A yf pool	2 options

C.4.1. Selector 111

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- POOL32Axf pool
- POOL32Axf.DSP pool

C.5. POOL32Ayf pool

Pool path

microMIPS / POOL32A / _POOL32A4 / POOL32Ayf

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[8:6] → selector[2:0]

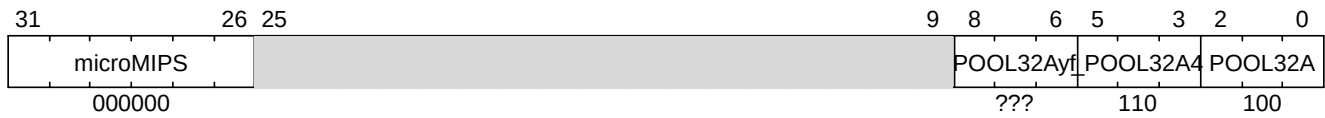


Figure 1766. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	—	—	POOL32A yf_3 pool	—	—	—	—

C.6. P00L32Ayf_3 pool

Pool path

microMIPS / POOL32A / _POOL32A4 / POOL32Ayf / POOL32Ayf_3

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:9] $\rightarrow$ selector[1:0]

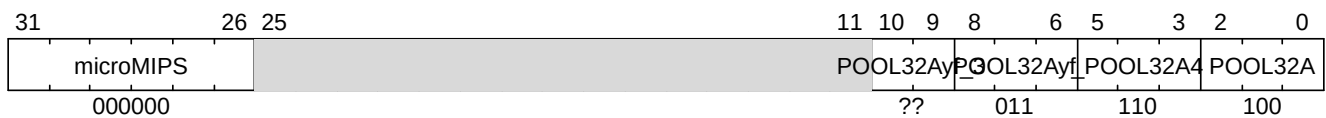


Figure 1767. 32-bit opcode fields

High / low	00	01	10	11
??	MFHC0	MTHC0	—	—

C.7. POOL32Axf pool

Pool path

microMIPS / P00L32A / _P00L32A4 / P00L32Axf

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[8:6] $\rightarrow$ selector[2:0]

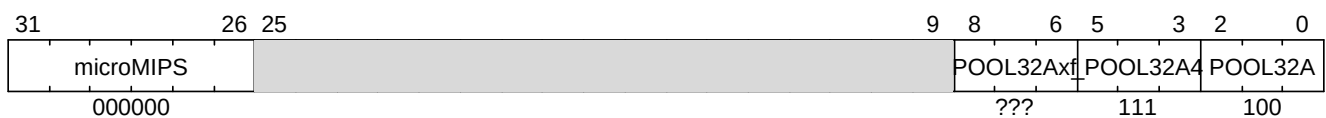


Figure 1768. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	19 options	25 options	POOL32A xf_3 pool	60 options	19 options	—	5 options

C.7.1. Selector 001

This selector has 19 possible interpretations across the declared revisions, features, and processor states:

- EXTP
- EXTPDP
- EXTR.W
- EXTR_R.W
- EXTR_RS.W
- EXTR_S.H
- MAQ_S.W.PHL
- MAQ_S.W.PHR
- MAQ_SA.W.PHL
- MAQ_SA.W.PHR
- MTHLIP32
- RDDSP32
- SHILOV
- SHLL.QB
- SHRL.QB
- WRDSP32
- MTHLIP
- RDDSP
- WRDSP

C.7.2. Selector 010

This selector has 25 possible interpretations across the declared revisions, features, and processor states:

- BALIGN
- DPA.W.PH
- DPAQ_S.W.PH
- DPAQ_SA.L.W
- DPAQX_S.W.PH
- DPAQX_SA.W.PH
- DPAU.H.QBL
- DPAU.H.QBR

- DPAX.W.PH
- DPS.W.PH
- DPSQ_S.W.PH
- DPSQ_SA.L.W
- DPSQX_S.W.PH
- DPSQX_SA.W.PH
- DPSU.H.QBL
- DPSU.H.QBR
- DPSX.W.PH
- EXTPDPV
- EXTPV
- EXTRV.W
- EXTRV_R.W
- EXTRV_RS.W
- EXTRV_S.H
- MULSA.W.PH
- MULSAQ_S.W.PH

C.7.3. Selector 100

This selector has 60 possible interpretations across the declared revisions, features, and processor states:

- ABSQ_S.PH
- ABSQ_S.QB
- ABSQ_S.W
- BITREV32
- CL032
- CLZ32
- DIV32
- DIVU32
- INSV
- JALR.HB32
- JALR32
- JALRC.HB32
- JALRC32
- JALRS.HB32

- JALRS32
- MADD32
- MADDU32
- MSUB32
- MSUBU32
- MULT32
- MULTU32
- PRECEQ.W.PHL32
- PRECEQ.W.PHR32
- PRECEQU.PH.QBL32
- PRECEQU.PH.QBLA32
- PRECEQU.PH.QBR32
- PRECEQU.PH.QBRA32
- PRECEU.PH.QBL32
- PRECEU.PH.QBLA32
- PRECEU.PH.QBR32
- PRECEU.PH.QBRA32
- RADDU.W.QB
- REPLV.PH32
- REPLV.QB32
- BITREV
- CLO
- CLZ
- DIV
- DIVU
- JALR
- JALRC
- JALRS
- MADD
- MADDU
- MSUB
- MSUBU
- MULT
- MULTU
- PRECEQ.W.PHL

- PRECEQ.W.PHR
- PRECEQU.PH.QBL
- PRECEQU.PH.QBLA
- PRECEQU.PH.QBR
- PRECEQU.PH.QBRA
- PRECEU.PH.QBL
- PRECEU.PH.QBLA
- PRECEU.PH.QBR
- PRECEU.PH.QBRA
- REPLV.PH
- REPLV.QB

C.7.4. Selector 101

This selector has 19 possible interpretations across the declared revisions, features, and processor states:

- DI32
- DVP32
- EI32
- ERET32
- EVP32
- MFHI32
- MFLO32
- MTHI32
- MTLO32
- DI
- DVP
- EI
- ERET
- ERETNC
- EVP
- MFHI
- MFLO
- MTHI
- MTLO

C.7.5. Selector 111

This selector has 5 possible interpretations across the declared revisions, features, and processor states:

- REPL.QB32
- SHRA.QB
- SHRA_R.QB
- SHRL.PH
- REPL.QB

C.8. POOL32Axf_3 pool

Pool path

microMIPS / POOL32A / _POOL32A4 / POOL32Axf / POOL32Axf_3

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:9] → selector[1:0]

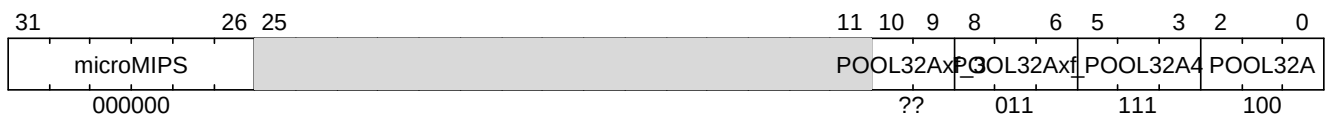


Figure 1769. 32-bit opcode fields

High / low	00	01	10	11
??	MFC0	MTC0	—	—

C.9. POOL32Axf.DSP pool

Pool path

microMIPS / POOL32A / _POOL32A4 / POOL32Axf.DSP

Selector width

8 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:6] → selector[7:0]

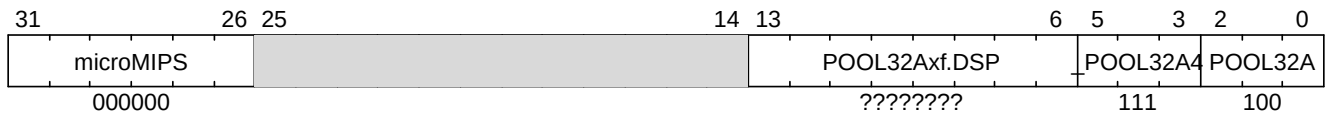


Figure 1770. 32-bit opcode fields

High / low	?????000	?????001	?????010	?????011	?????100	?????101	?????110	?????111
00000???	—	MFHI.DSP3 2 / MFHI	—	—	—	—	—	—
00001???	—	—	—	—	—	—	—	—
00010???	—	—	—	—	—	—	—	—
00011???	—	—	—	—	—	—	—	—
00100???	—	—	—	—	—	—	—	—
00101???	—	—	MADD	—	—	—	—	—
00110???	—	—	MULT.DSP3 2 / MULT	—	—	—	—	—
00111???	—	—	—	—	—	—	—	—
01000???	—	MFLO.DSP3 2 / MFLO	—	—	—	—	—	—
01001???	—	—	—	—	—	—	—	—
01010???	—	—	—	—	—	—	—	—
01011???	—	—	—	—	—	—	—	—
01100???	—	—	—	—	—	—	—	—
01101???	—	—	MADDU	—	—	—	—	—
01110???	—	—	MULTU.DSP 32 / MULTU	—	—	—	—	—
01111???	—	—	—	—	—	—	—	—
10000???	—	MTHI.DSP3 2 / MTHI	—	—	—	—	—	—
10001???	—	—	—	—	—	—	—	—
10010???	—	—	—	—	—	—	—	—
10011???	—	—	—	—	—	—	—	—
10100???	—	—	—	—	—	—	—	—
10101???	—	—	MSUB	—	—	—	—	—
10110???	—	—	—	—	—	—	—	—
10111???	—	—	—	—	—	—	—	—

High / low	????000	????001	????010	????011	????100	????101	????110	????111
11000???	—	MTLO.DSP3 2 / MTLO	—	—	—	—	—	—
11001???	—	—	—	—	—	—	—	—
11010???	—	—	—	—	—	—	—	—
11011???	—	—	—	—	—	—	—	—
11100???	—	—	—	—	—	—	—	—
11101???	—	—	MSUBU	—	—	—	—	—
11110???	—	—	—	—	—	—	—	—
11111???	—	—	—	—	—	—	—	—

C.10. _POOL32A5 pool

Pool path

microMIPS / POOL32A / _POOL32A5

Selector width

7 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:3] → selector[6:0]

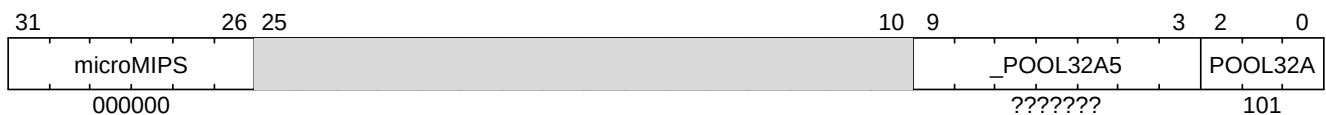


Figure 1771. 32-bit opcode fields

High / low	????000	????001	????010	????011	????100	????101	????110	????111
0000???	CMP.EQ.PH	ADDQ.PH / ADDQ_S.PH	—	SHILO	1 options	MUL.PH / MUL_S.PH	—	REPL.PH32 / REPL.PH
0001???	CMP.LT.PH	ADDQH.PH / ADDQH_R.P H	—	—	1 options	2 options	—	—
0010???	CMP.LE.PH	ADDQH.W / ADDQH_R.W	1 options	—	—	2 options	—	—
0011???	CMPGU.EQ. QB	ADDU.QB / ADDU_S.QB	1 options	—	—	2 options	—	—

High / low	????000	????001	????010	????011	????100	????101	????110	????111
0100???	CMPGU.LT.QB	ADDU.PH / ADDU_S.PH	MULQ_RS.PH	—	—	2 options	—	—
0101???	CMPGU.LE.QB	ADDUH.QB / ADDUH_R.QB	MULQ_S.PH	—	LHX	2 options	—	—
0110???	CMPGDU.EQ.QB	SHRAV.PH / SHRAV_R.PH	MULQ_RS.W	—	LWX	PACKRL.PH 32 / PACKRL.PH	—	—
0111???	CMPGDU.LT.QB	SHRAV.QB / SHRAV_R.QB	MULQ_S.W	—	—	PICK.QB	—	—
1000???	CMPGDU.LE.QB	SUBQ.PH / SUBQ_S.PH	APPEND	—	LBUX	PICK.PH	—	—
1001???	CMPU.EQ.QB	SUBQH.PH / SUBQH_R.PH	PREPEND	—	—	—	—	—
1010???	CMPU.LT.QB	SUBQH.W / SUBQH_R.W	MODSUB	—	—	—	—	—
1011???	CMPU.LE.QB	SUBU.QB / SUBU_S.QB	SHRAV_R.W	—	—	—	SHRA_R.W	—
1100???	ADDQ_S.W	SUBU.PH / SUBU_S.PH	SHRLV.PH	—	—	—	SHRA.PH / SHRA_R.PH	—
1101???	SUBQ_S.W	SUBUH.QB / SUBUH_R.QB	SHRLV.QB	—	—	—	—	—
1110???	ADDSC	SHLLV.PH / SHLLV_S.PH	SHLLV.QB	—	—	—	SHLL.PH / SHLL_S.PH	—
1111???	ADDWC	4 options	SHLLV_S.W	—	—	—	SHLL_S.W	—

C.10.1. Selector 0000100

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEQ_S.W.PHL

C.10.2. Selector 0001100

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEQ_S.W.PHR

C.10.3. Selector 0001101

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- PRECR.QB.PH32
- PRECR.QB.PH

C.10.4. Selector 0010010

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEU_S.PH.QBL

C.10.5. Selector 0010101

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- PRECRQ.QB.PH32
- PRECRQ.QB.PH

C.10.6. Selector 0011010

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEU_S.PH.QBR

C.10.7. Selector 0011101

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- PRECRQ.PH.W32
- PRECRQ.PH.W

C.10.8. Selector 0100101

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- PRECRQ_RS.PH.W32
- PRECRQ_RS.PH.W

C.10.9. Selector 0101101

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- PRECRQU_S.QB.PH32
- PRECRQU_S.QB.PH

C.10.10. Selector 1111001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- PRECR_SRA.PH.W32
- PRECR_SRA_R.PH.W32
- PRECR_SRA.PH.W
- PRECR_SRA_R.PH.W

C.11. _POOL32A7 pool

Pool path

microMIPS / POOL32A / _POOL32A7

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:3] → selector[2:0]

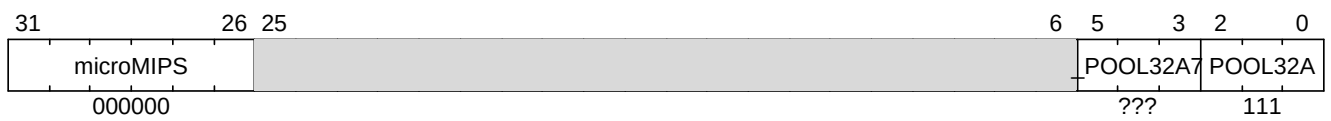


Figure 1772. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	BREAK32 / BREAK	LSA	—	ALIGN	—	—	—	—

C.12. POOL32B.MULTI32 pool

Pool path

microMIPS / POOL32B.MULTI32

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

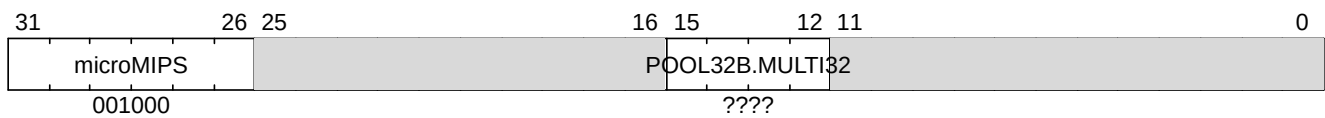


Figure 1773. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	—	LWM	—	—
1???	—	—	—	—	—	SWM	—	—

C.13. POOL32B.PAIR32 pool

Pool path

microMIPS / POOL32B.PAIR32

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

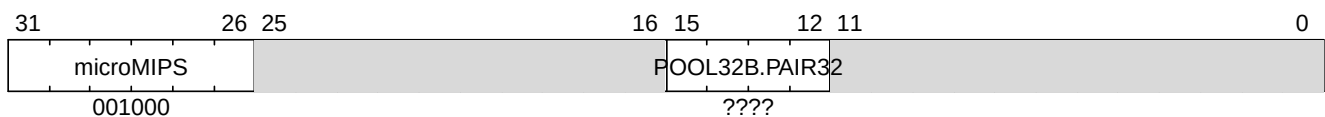


Figure 1774. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	LWP	—	—	—	—	—	—
1???	—	SWP	—	—	—	—	—	—

C.14. POOL32B.BRANCH pool

Pool path

microMIPS / POOL32B.BRANCH

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[25:21] → selector[4:0]

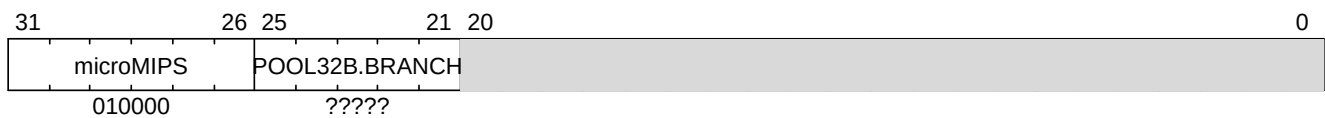


Figure 1775. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	—	BNEZC32 / BNEZC	—	BEQZC32 / BEQZC
01???	—	—	—	—	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

C.15. POOL32I pool

Pool path

microMIPS / POOL32I

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[25:23] → selector[2:0]

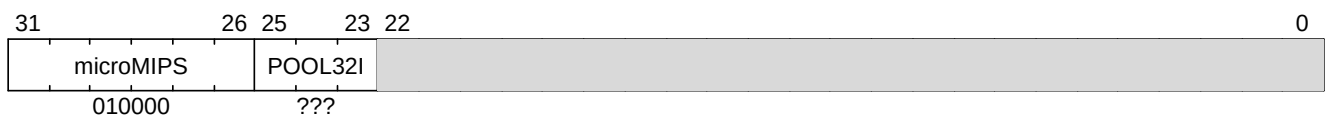


Figure 1776. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	POOL32I_0 pool	POOL32I_1 pool	POOL32I_2 pool	LUI32 / POOL32I_3 pool / LUI	POOL32I_4 pool	—	POOL32I_6 pool	1 options

C.15.1. Selector 111

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- POOL32I_7.LEGACY pool

C.16. POOL32I_0 pool

Pool path

microMIPS / POOL32I / POOL32I_0

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[22:21] → selector[1:0]

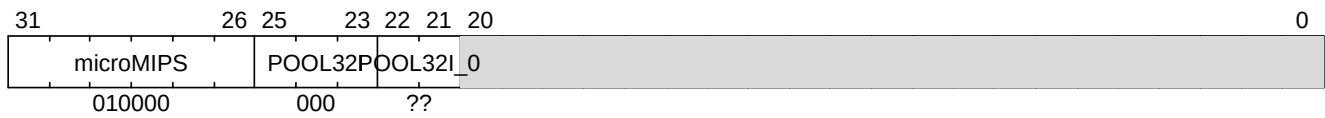


Figure 1777. 32-bit opcode fields

High / low	00	01	10	11
??	BLTZ32 / BLTZ	BLTZAL32 / BLTZAL	BGEZ32 / BGEZ	BGEZAL32 / BGEZAL

C.17. POOL32I_1 pool

Pool path

microMIPS / POOL32I / POOL32I_1

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[22:21] → selector[1:0]

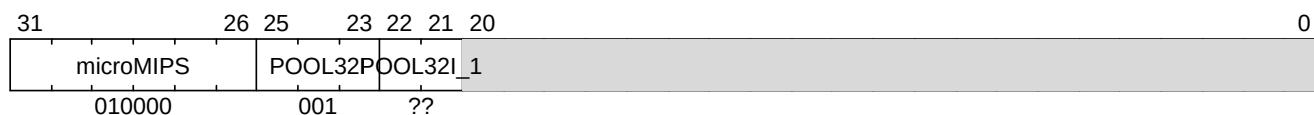


Figure 1778. 32-bit opcode fields

High / low	00	01	10	11
??	BLEZ32 / BLEZ	—	BGTZ32 / BGTZ	—

C.18. POOL32I_2 pool

Pool path

microMIPS / POOL32I / POOL32I_2

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[22:21] → selector[1:0]

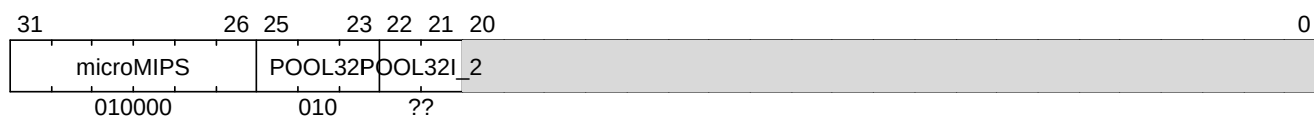


Figure 1779. 32-bit opcode fields

High / low	00	01	10	11
??	BC1EQZC	BC1NEZC	—	—

C.19. POOL32I_3 pool

Pool path

microMIPS / POOL32I / POOL32I_3

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[22:21] → selector[1:0]



Figure 1780. 32-bit opcode fields

High / low	00	01	10	11
??	SYNCI32 / SYNCI	—	—	—

C.20. POOL32I_4 pool

Pool path

microMIPS / POOL32I / POOL32I_4

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[22:21] → selector[1:0]

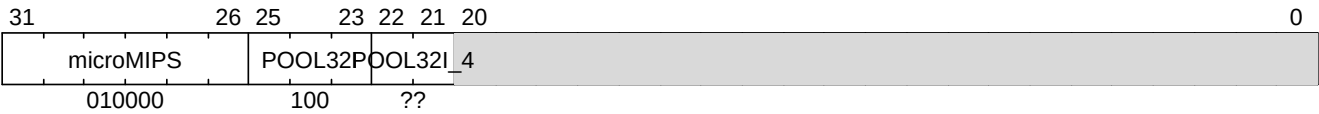


Figure 1781. 32-bit opcode fields

High / low	00	01	10	11
??	SYNCI32 / SYNCI	BLTZALS32 / BLTZALS	—	BGEZALS32 / BGEZALS

C.21. POOL32I_6 pool

Pool path

microMIPS / POOL32I / POOL32I_6

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[22:21] → selector[1:0]



Figure 1782. 32-bit opcode fields

High / low	00	01	10	11
??	—	BPOSGE32C32 / BPOSGE32C	—	BPOSGE3232 / BPOSGE32

C.22. POOL32I_7.LEGACY pool

Pool path

microMIPS / POOL32I / POOL32I_7.LEGACY

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[22:21] → selector[1:0]

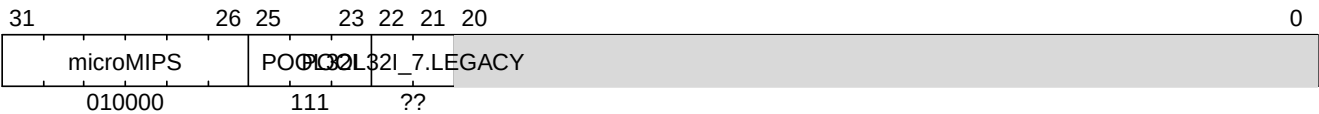


Figure 1783. 32-bit opcode fields

High / low	00	01	10	11
??	BC1F	BC1T	—	—

C.23. REGIMM32 pool

Pool path

microMIPS / REGIMM32

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[25:21] → selector[4:0]



Figure 1784. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	—	—	—	—
01???	TLTI32 / TLTI	TGEI32 / TGEI	TLTIU32 / TLTIU	TGEIU32 / TGEIU	TNEI32 / TNEI	—	TEQI32 / TEQI	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

C.24. POOL16C pool

Pool path

microMIPS / POOL16C

Selector width

4 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[9:6] → selector[3:0]

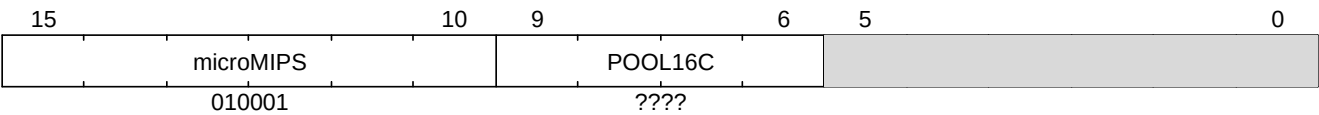


Figure 1785. 16-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	NOT16 / NOT	XOR16 / XOR	—	OR16 / OR	—	—	4 options	4 options
1???	MFHI16 / MFHI	MFLO16 / MFLO	BREAK16 / BREAK	—	JRADDIUSP	—	—	—

C.24.1. Selector 0110

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- JR16
- JRC16

- JR
- JRC

C.24.2. Selector 0111

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- JALR16
- JALRS16
- JALR
- JALRS

C.25. POOL16C.LOGICAL pool

Pool path

microMIPS / POOL16C.LOGICAL

Selector width

4 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[9:6] → selector[3:0]

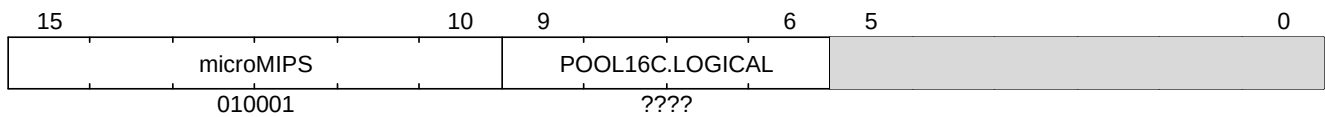


Figure 1786. 16-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	AND16 / AND	—	—	—	—	—
1???	—	—	—	—	—	—	—	—

C.26. POOL16C.LOGICAL_R6 pool

Pool path

microMIPS / POOL16C.LOGICAL_R6

Selector width

4 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[3:0] → selector[3:0]

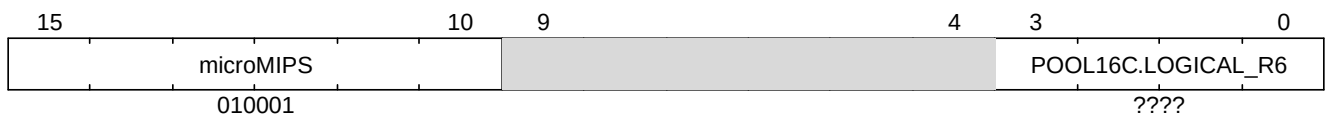


Figure 1787. 16-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	AND16 / AND	—	—	—	—	—	—
1???	—	—	—	—	—	—	—	—

C.27. POOL16C.MULTI pool

Pool path

microMIPS / POOL16C.MULTI

Selector width

4 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[9:6] → selector[3:0]

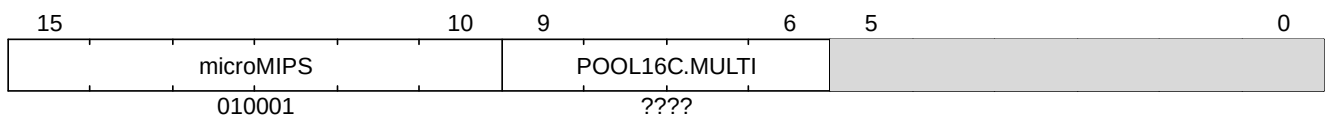


Figure 1788. 16-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	LWM	SWM	—	—
1???	—	—	—	—	—	—	—	—

C.28. POOL16C.R6.BREAK pool

Pool path

microMIPS / POOL16C.R6.BREAK

Selector width

6 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[5:0] → selector[5:0]

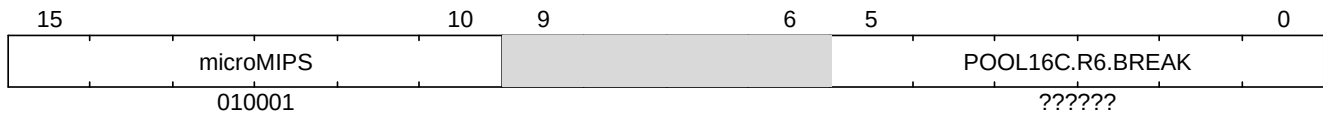


Figure 1789. 16-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	—	—	—	—	—	—	—
001???	—	—	—	—	—	—	—	—
010???	—	—	—	—	—	—	—	—
011???	—	—	—	BREAK16 / BREAK	—	—	—	—
100???	—	—	—	—	—	—	—	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

C.29. POOL16C.R6.JR pool

Pool path

microMIPS / POOL16C.R6.JR

Selector width

5 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[4:0] → selector[4:0]

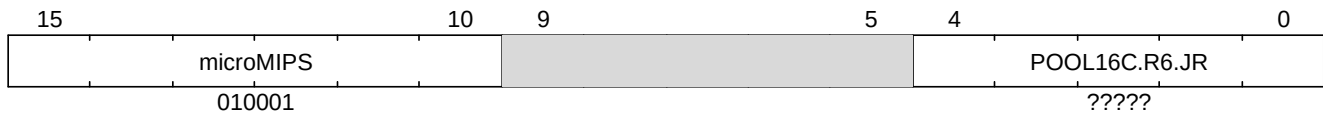


Figure 1790. 16-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	JRC16 / JRC	—	—	—	—
01???	—	—	—	JALRC16 / JALRC	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

C.30. POOL16C.R6.JRSP pool

Pool path

microMIPS / POOL16C.R6.JRSP

Selector width

5 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[4:0] → selector[4:0]

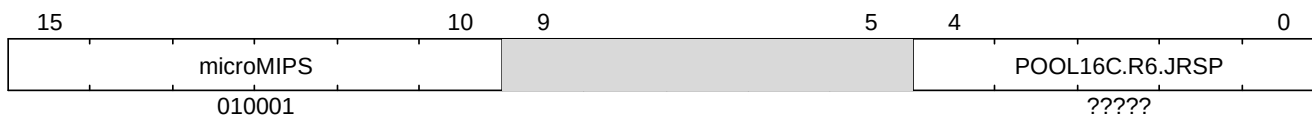


Figure 1791. 16-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	—	—	—	JRCADDIUS P	—	—	—	—
11???	—	—	—	—	—	—	—	—

C.31. POOL16C.R6.RR pool

Pool path

microMIPS / POOL16C.R6.RR

Selector width

4 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[3:0] → selector[3:0]

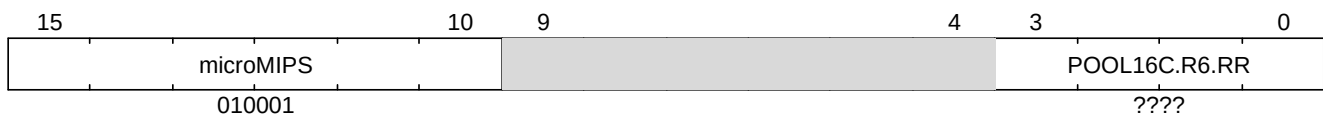


Figure 1792. 16-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	NOT16.R6 / NOT	—	—	—	—	—	—	—
1???	XOR16 / XOR	OR16 / OR	—	—	—	—	—	—

C.32. POOL32F pool

Pool path

microMIPS / POOL32F

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

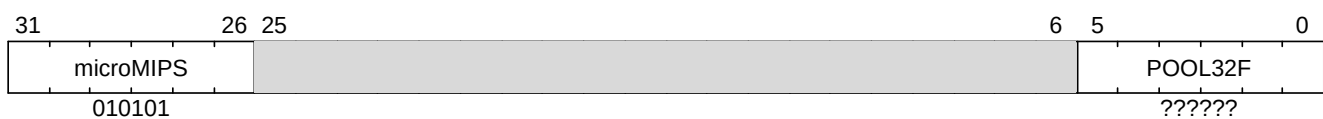


Figure 1793. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	9 options	MADD / MADD.S	NMADD / NMADD.S	MIN.D / MIN.S	—	CMP.S	—	—
001???	—	MADD / MADD.D	NMADD / NMADD.D	MAX.D / MAX.S	—	—	—	—
010???	—	MADD / MADD.PS	NMADD / NMADD.PS	—	—	CMP.D	—	—
011???	—	ALNV.PS	—	—	—	—	—	—

High / low	???000	???001	???010	???011	???100	???101	???110	???111
100???	13 options	MSUB / MSUB.S	NMSUB / NMSUB.S	MINA.D / MINA.S	—	—	—	—
101???	—	MSUB / MSUB.D	NMSUB / NMSUB.D	MAXA.D / MAXA.S	—	—	—	—
110???	14 options	MSUB / MSUB.PS	NMSUB / NMSUB.PS	—	—	—	—	—
111???	18 options	—	—	11 options	C.cond.D / C.cond.PS / C.cond.S	—	—	—

C.32.1. Selector 000000

This selector has 9 possible interpretations across the declared revisions, features, and processor states:

- CVT.PS.S
- PLL
- PLU
- PUL
- PUU
- PLL.PS
- PLU.PS
- PUL.PS
- PUU.PS

C.32.2. Selector 100000

This selector has 13 possible interpretations across the declared revisions, features, and processor states:

- CLASS.D
- CLASS.S
- MOVF
- MOVT
- PREFX
- RINT.D
- RINT.S
- MOVF.D
- MOVF.PS

- MOVF.S
- MOVT.D
- MOVT.PS
- MOVT.S

C.32.3. Selector 110000

This selector has 14 possible interpretations across the declared revisions, features, and processor states:

- ADD
- ADD.D
- ADD.S
- DIV.D
- DIV.S
- MUL
- MUL.D
- MUL.S
- SUB
- SUB.D
- SUB.S
- ADD.PS
- MUL.PS
- SUB.PS

C.32.4. Selector 111000

This selector has 18 possible interpretations across the declared revisions, features, and processor states:

- MADDF.D
- MADDF.S
- MOVN
- MOVZ
- MSUBF.D
- MSUBF.S
- SEL.D
- SEL.S
- SELEQZ.D

- SELEQZ.S
- SELNEZ.D
- SELNEZ.S
- MOVN.D
- MOVN.PS
- MOVN.S
- MOVZ.D
- MOVZ.PS
- MOVZ.S

C.32.5. Selector 111011

This selector has 11 possible interpretations across the declared revisions, features, and processor states:

- CFC1
- CTC1
- DMFC1
- DMTC1
- MFHC1
- MOVF
- MOVT
- MTC1
- MTHC1
- POOL32FXF pool
- POOL32FXF.WIDE pool

C.33. POOL32FXF pool

Pool path

microMIPS / POOL32F / POOL32FXF

Selector width

7 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[12:6] → selector[6:0]

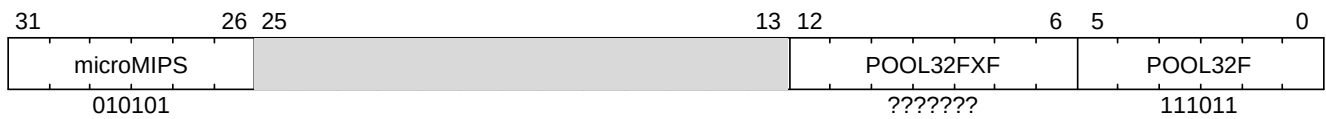


Figure 1794. 32-bit opcode fields

High / low	????000	????001	????010	????011	????100	????101	????110	????111
0000???	MFC1	4 options	—	—	CVT.S.PL	—	—	—
0001???	—	—	—	—	—	4 options	—	—
0010???	—	—	—	—	—	—	—	—
0011???	—	—	—	—	—	—	—	—
0100???	—	—	—	—	CVT.S.PU	—	—	—
0101???	—	—	—	—	—	4 options	—	—
0110???	—	—	—	—	—	—	—	—
0111???	—	—	—	—	—	—	—	—
1000???	—	—	—	—	—	—	—	—
1001???	—	—	—	—	—	CVT.D.L / CVT.D.S / CVT.D.W	—	—
1010???	—	—	—	—	—	—	—	—
1011???	—	—	—	—	—	—	—	—
1100???	—	—	—	—	—	—	—	—
1101???	—	—	—	—	—	CVT.S.D / CVT.S.L / CVT.S.W	—	—
1110???	—	—	—	—	—	—	—	—
1111???	—	—	—	—	—	—	—	—

C.33.1. Selector 0000001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MOV
- MOV.D
- MOV.S
- MOV.PS

C.33.2. Selector 0001101

This selector has 4 possible interpretations across the declared revisions, features, and processor

states:

- ABS
- ABS.D
- ABS.S
- ABS.PS

C.33.3. Selector 0101101

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- NEG
- NEG.D
- NEG.S
- NEG.PS

C.34. POOL32FXF.WIDE pool

Pool path

microMIPS / POOL32F / POOL32FXF.WIDE

Selector width

8 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:6] → selector[7:0]

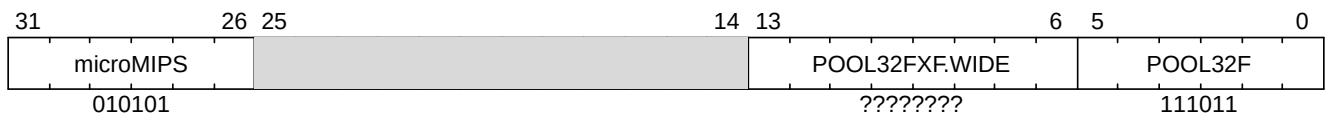


Figure 1795. 32-bit opcode fields

High / low	00000000	00000001	00000010	00000011	00000100	00000101	00000110	00000111
00000000	—	—	—	—	CVT.L.D / CVT.L.S	—	—	—
00000001	RSQRT.D / RSQRT.S	—	—	—	FLOOR.L.D / FLOOR.L.S	—	—	—
00000010	—	—	—	—	—	—	—	—
00000011	—	—	—	—	—	—	—	—

High / low	?????000	?????001	?????010	?????011	?????100	?????101	?????110	?????111
00100???	—	—	—	—	CVT.W.D / CVT.W.S	—	—	—
00101???	SQRT.D / SQRT.S	—	—	—	FLOOR.W.D / FLOOR.W.S	—	—	—
00110???	—	—	—	—	—	—	—	—
00111???	—	—	—	—	—	—	—	—
01000???	—	—	—	—	—	—	—	—
01001???	RECIP.D / RECIP.S	—	—	—	CEIL.L.D / CEIL.L.S	—	—	—
01010???	—	—	—	—	—	—	—	—
01011???	—	—	—	—	—	—	—	—
01100???	—	—	—	—	—	—	—	—
01101???	—	—	—	—	CEIL.W.D / CEIL.W.S	—	—	—
01110???	—	—	—	—	—	—	—	—
01111???	—	—	—	—	—	—	—	—
10000???	—	—	—	—	—	—	—	—
10001???	—	—	—	—	TRUNC.L.D / TRUNC.L.S	—	—	—
10010???	—	—	—	—	—	—	—	—
10011???	—	—	—	—	—	—	—	—
10100???	—	—	—	—	—	—	—	—
10101???	—	—	—	—	TRUNC.W.D / TRUNC.W.S	—	—	—
10110???	—	—	—	—	—	—	—	—
10111???	—	—	—	—	—	—	—	—
11000???	—	—	—	—	—	—	—	—
11001???	—	—	—	—	ROUND.L.D / ROUND.L.S	—	—	—
11010???	—	—	—	—	—	—	—	—
11011???	—	—	—	—	—	—	—	—
11100???	—	—	—	—	—	—	—	—

High / low	????000	????001	????010	????011	????100	????101	????110	????111
11101???	—	—	—	—	ROUND.W.D / ROUND.W.S	—	—	—
11110???	—	—	—	—	—	—	—	—
11111???	—	—	—	—	—	—	—	—

C.35. POOL32S pool

Pool path

microMIPS / POOL32S

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:0] → selector[5:0]

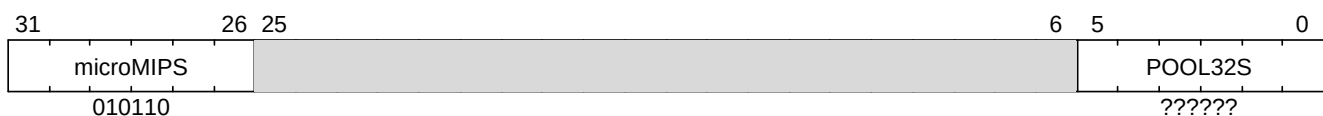


Figure 1796. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	8 options	—	—	—	DINSM	—	—	—
001???	9 options	—	—	—	DINS	—	—	—
010???	16 options	—	—	—	DEXTU	—	—	—
011???	16 options	—	—	—	DADDI32 / DADDI	—	—	—
100???	—	—	—	—	DEXTM	—	—	—
101???	—	—	—	—	DEXT	—	—	—
110???	—	—	—	—	DINSU	—	—	—
111???	—	—	—	—	POOL32S xf_3 pool / POOL32S xf_4 pool	—	—	—

C.35.1. Selector 000000

This selector has 8 possible interpretations across the declared revisions, features, and processor states:

- DR0TR32
- DSLL32
- DSRA32
- DSRL32
- DR0TR
- DSLL
- DSRA
- DSRL

C.35.2. Selector 001000

This selector has 9 possible interpretations across the declared revisions, features, and processor states:

- DLSA
- DR0TR3232
- DSLL3232
- DSRA3232
- DSRL3232
- DR0TR32
- DSLL32
- DSRA32
- DSRL32

C.35.3. Selector 010000

This selector has 16 possible interpretations across the declared revisions, features, and processor states:

- DADD32
- DADDU32
- DR0TRV32
- DSLLV32
- DSRAV32
- DSRLV32
- DSUB32

- DSUBU32
- DADD
- DADDU
- DROTRV
- DSLLV
- DSRAV
- DSRLV
- DSUB
- DSUBU

C.35.4. Selector 011000

This selector has 16 possible interpretations across the declared revisions, features, and processor states:

- DDIV.R6
- DDIVU.R6
- DMOD.R6
- DMODU.R6
- DMUH.R6
- DMUHU.R6
- DMUL.R6
- DMULU.R6
- DDIV
- DDIVU
- DMOD
- DMODU
- DMUH
- DMUHU
- DMUL
- DMULU

C.36. POOL32Sxf_3 pool

Pool path

microMIPS / POOL32S / POOL32Sxf_3

Selector width

5 bits

Encoding width
32 bits

Selector layout
32-bit: instruction[10:6] → selector[4:0]

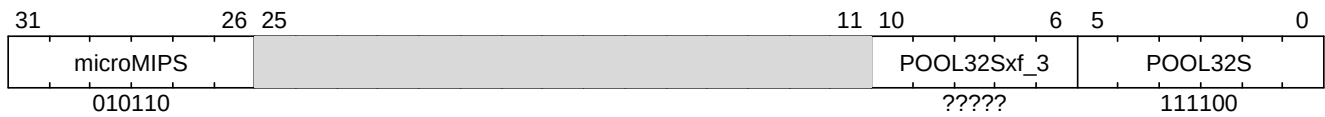


Figure 1797. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	DMFC032 / DMFC0	—	—	—	—
01???	—	—	—	DMTC032 / DMTC0	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

C.37. POOL32Sxf_4 pool

Pool path
microMIPS / POOL32S / POOL32Sxf_4

Selector width
10 bits

Encoding width
32 bits

Selector layout
32-bit: instruction[15:6] → selector[9:0]

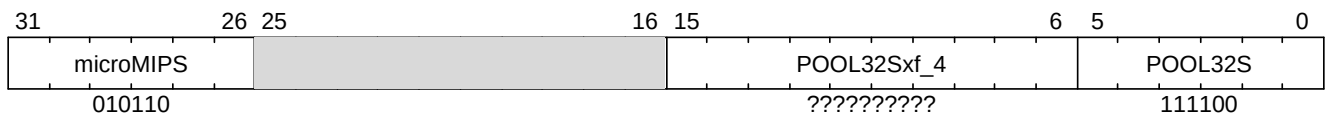


Figure 1798. 32-bit opcode fields

High / low	???????00 0	???????00 1	???????01 0	???????01 1	???????10 0	???????10 1	???????11 0	???????11 1
0000000?? ?	—	—	—	—	—	—	—	—
0000001?? ?	—	—	—	—	—	—	—	—

High / low	???????0 0	???????0 1	???????0 0	???????0 1	???????1 0	???????1 1	???????1 0	???????1 1
000010?? ?	—	—	—	—	—	—	—	—
000011?? ?	—	—	—	—	—	—	—	—
0000100?? ?	—	—	—	—	—	—	—	—
0000101?? ?	—	—	—	—	DBITSWAP	—	—	—
0000110?? ?	—	—	—	—	—	—	—	—
0000111?? ?	—	—	—	—	—	—	—	—
0001000?? ?	—	—	—	—	—	—	—	—
0001001?? ?	—	—	—	—	—	—	—	—
0001010?? ?	—	—	—	—	—	—	—	—
0001011?? ?	—	—	—	—	—	—	—	—
0001100?? ?	—	—	—	—	—	—	—	—
0001101?? ?	—	—	—	—	—	—	—	—
0001110?? ?	—	—	—	—	—	—	—	—
0001111?? ?	—	—	—	—	—	—	—	—
0010000?? ?	—	—	—	—	—	—	—	—
0010001?? ?	—	—	—	—	—	—	—	—
0010010?? ?	—	—	—	—	—	—	—	—
0010011?? ?	—	—	—	—	—	—	—	—
0010100?? ?	—	—	—	—	—	—	—	—
0010101?? ?	—	—	—	—	—	—	—	—
0010110?? ?	—	—	—	—	—	—	—	—
0010111?? ?	—	—	—	—	—	—	—	—

High / low	???????0 0	???????0 1	???????0 0	???????0 1	???????1 0	???????1 1	???????1 0	???????1 1
001100?? ?	—	—	—	—	—	—	—	—
0011001?? ?	—	—	—	—	—	—	—	—
0011010?? ?	—	—	—	—	—	—	—	—
0011011?? ?	—	—	—	—	—	—	—	—
0011100?? ?	—	—	—	—	—	—	—	—
0011101?? ?	—	—	—	—	—	—	—	—
0011110?? ?	—	—	—	—	—	—	—	—
0011111?? ?	—	—	—	—	—	—	—	—
0100000?? ?	—	—	—	—	—	—	—	—
0100001?? ?	—	—	—	—	—	—	—	—
0100010?? ?	—	—	—	—	—	—	—	—
0100011?? ?	—	—	—	—	—	—	—	—
0100100?? ?	—	—	—	—	—	—	—	—
0100101?? ?	—	—	—	—	DCL032 / DCLO	—	—	—
0100110?? ?	—	—	—	—	—	—	—	—
0100111?? ?	—	—	—	—	—	—	—	—
0101000?? ?	—	—	—	—	—	—	—	—
0101001?? ?	—	—	—	—	—	—	—	—
0101010?? ?	—	—	—	—	—	—	—	—
0101011?? ?	—	—	—	—	—	—	—	—
0101100?? ?	—	—	—	—	—	—	—	—
0101101?? ?	—	—	—	—	DCLZ32 / DCLZ	—	—	—

High / low	???????0 0	???????0 1	???????0 0	???????0 1	???????1 0	???????1 1	???????1 0	???????1 1
010110?? ?	—	—	—	—	—	—	—	—
010111?? ?	—	—	—	—	—	—	—	—
011000?? ?	—	—	—	—	—	—	—	—
0110001?? ?	—	—	—	—	—	—	—	—
0110010?? ?	—	—	—	—	—	—	—	—
0110011?? ?	—	—	—	—	—	—	—	—
0110100?? ?	—	—	—	—	—	—	—	—
0110101?? ?	—	—	—	—	—	—	—	—
0110110?? ?	—	—	—	—	—	—	—	—
0110111?? ?	—	—	—	—	—	—	—	—
0111000?? ?	—	—	—	—	—	—	—	—
0111001?? ?	—	—	—	—	—	—	—	—
0111010?? ?	—	—	—	—	—	—	—	—
0111011?? ?	—	—	—	—	—	—	—	—
0111100?? ?	—	—	—	—	—	—	—	—
0111101?? ?	—	—	—	—	DSBH32 / DSBH	—	—	—
0111110?? ?	—	—	—	—	—	—	—	—
0111111?? ?	—	—	—	—	—	—	—	—
1000000?? ?	—	—	—	—	—	—	—	—
1000001?? ?	—	—	—	—	—	—	—	—
1000010?? ?	—	—	—	—	—	—	—	—
1000011?? ?	—	—	—	—	—	—	—	—

High / low	???????0 0	???????0 1	???????0 0	???????0 1	???????1 0	???????1 1	???????1 0	???????1 1
1000100?? ?	—	—	—	—	—	—	—	—
1000101?? ?	—	—	—	—	DMULT32 / DMULT	—	—	—
1000110?? ?	—	—	—	—	—	—	—	—
1000111?? ?	—	—	—	—	—	—	—	—
1001000?? ?	—	—	—	—	—	—	—	—
1001001?? ?	—	—	—	—	—	—	—	—
1001010?? ?	—	—	—	—	—	—	—	—
1001011?? ?	—	—	—	—	—	—	—	—
1001100?? ?	—	—	—	—	—	—	—	—
1001101?? ?	—	—	—	—	DMULTU32 / DMULTU	—	—	—
1001110?? ?	—	—	—	—	—	—	—	—
1001111?? ?	—	—	—	—	—	—	—	—
1010000?? ?	—	—	—	—	—	—	—	—
1010001?? ?	—	—	—	—	—	—	—	—
1010010?? ?	—	—	—	—	—	—	—	—
1010011?? ?	—	—	—	—	—	—	—	—
1010100?? ?	—	—	—	—	—	—	—	—
1010101?? ?	—	—	—	—	DDIV32 / DDIV	—	—	—
1010110?? ?	—	—	—	—	—	—	—	—
1010111?? ?	—	—	—	—	—	—	—	—
1011000?? ?	—	—	—	—	—	—	—	—
1011001?? ?	—	—	—	—	—	—	—	—

High / low	???????0 0	???????0 1	???????0 0	???????0 1	???????1 0	???????1 1	???????1 0	???????1 1
1011010?? ?	—	—	—	—	—	—	—	—
1011011?? ?	—	—	—	—	—	—	—	—
1011100?? ?	—	—	—	—	—	—	—	—
1011101?? ?	—	—	—	—	DDIVU32 / DDIVU	—	—	—
1011110?? ?	—	—	—	—	—	—	—	—
1011111?? ?	—	—	—	—	—	—	—	—
1100000?? ?	—	—	—	—	—	—	—	—
1100001?? ?	—	—	—	—	—	—	—	—
1100010?? ?	—	—	—	—	—	—	—	—
1100011?? ?	—	—	—	—	—	—	—	—
1100100?? ?	—	—	—	—	—	—	—	—
1100101?? ?	—	—	—	—	—	—	—	—
1100110?? ?	—	—	—	—	—	—	—	—
1100111?? ?	—	—	—	—	—	—	—	—
1101000?? ?	—	—	—	—	—	—	—	—
1101001?? ?	—	—	—	—	—	—	—	—
1101010?? ?	—	—	—	—	—	—	—	—
1101011?? ?	—	—	—	—	—	—	—	—
1101100?? ?	—	—	—	—	—	—	—	—
1101101?? ?	—	—	—	—	—	—	—	—
1101110?? ?	—	—	—	—	—	—	—	—
1101111?? ?	—	—	—	—	—	—	—	—

High / low	???????0 0	???????0 1	???????0 0	???????0 1	???????1 0	???????1 1	???????1 0	???????1 1
111000?? ?	—	—	—	—	—	—	—	—
111001?? ?	—	—	—	—	—	—	—	—
111010?? ?	—	—	—	—	—	—	—	—
111011?? ?	—	—	—	—	—	—	—	—
111010?? ?	—	—	—	—	—	—	—	—
1110101?? ?	—	—	—	—	—	—	—	—
1110110?? ?	—	—	—	—	—	—	—	—
1110111?? ?	—	—	—	—	—	—	—	—
1111000?? ?	—	—	—	—	—	—	—	—
1111001?? ?	—	—	—	—	—	—	—	—
1111010?? ?	—	—	—	—	—	—	—	—
1111011?? ?	—	—	—	—	—	—	—	—
1111100?? ?	—	—	—	—	—	—	—	—
1111101?? ?	—	—	—	—	DSHD32 / DSHD	—	—	—
1111110?? ?	—	—	—	—	—	—	—	—
1111111?? ?	—	—	—	—	—	—	—	—

C.38. POOL32B pool

Pool path

microMIPS / POOL32B

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

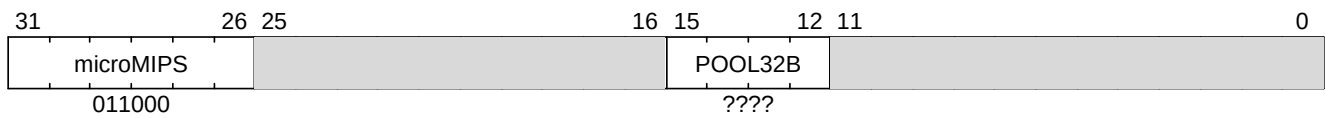


Figure 1799. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	—	—	—	—
1???	—	—	—	SC32 / SC	—	—	LWJ32 / LWJ	—

C.39. POOL32C pool

Pool path

microMIPS / POOL32C

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

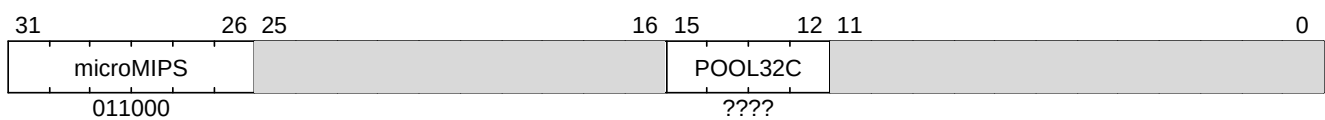


Figure 1800. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	LLWP32 / LLWP	—	LL32 / LL	—	—	—	—
1???	—	SCWP32 / SCWP	—	—	—	—	—	—

C.40. R6PCREL pool

Pool path

microMIPS / R6PCREL

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:19] → selector[1:0]

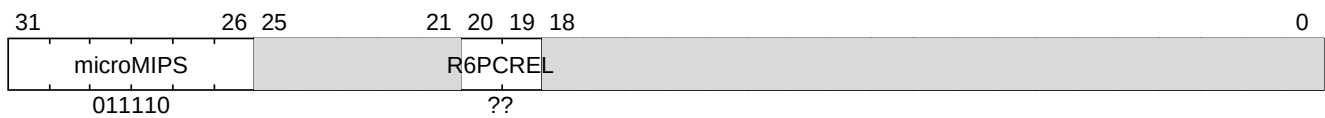


Figure 1801. 32-bit opcode fields

High / low	00	01	10	11
??	2 options	LWPC	LWUPC	R6PCREL_11 pool

C.40.1. Selector 00

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- ADDIUPC.32.MICROMIPS
- ADDIUPC

C.41. R6PCREL_11 pool

Pool path

microMIPS / R6PCREL / R6PCREL_11

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[18:18] → selector[0:0]

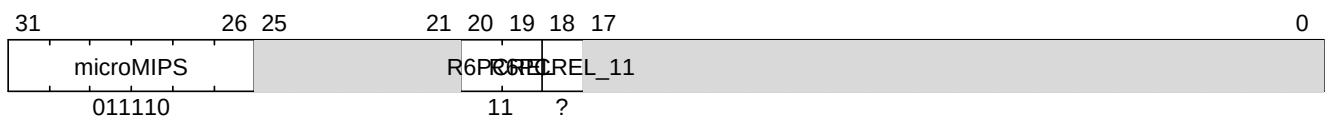


Figure 1802. 32-bit opcode fields

High / low	0	1
?	2 options	R6PCREL_11_1 pool

C.41.1. Selector 0

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- LDPC.32.MICROMIPS
- LDPC

C.42. R6PCREL_11_1 pool

Pool path

microMIPS / R6PCREL / R6PCREL_11 / R6PCREL_11_1

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[17:16] → selector[1:0]

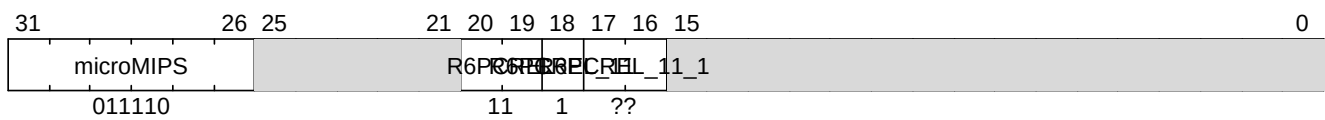


Figure 1803. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	—	2 options

C.42.1. Selector 11

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- ALUIPC.32.MICROMIPS
- ALUIPC

Appendix D: nanoMIPS Encoding Summary

This chapter gives a hierarchical view of the instruction encoding space across all declared revisions, features, and processor states. Named entries descend into another opcode pool; multiple names in one cell are overlapping possibilities, and an em dash marks an unallocated encoding.

D.1. NM pool

Pool path

NM

Selector width

1 bits

Encoding width

16, 32, 48 bits

Selector layout

16-bit: instruction[12:12] → selector[0:0]; 32-bit: instruction[28:28] → selector[0:0]; 48-bit: instruction[44:44] → selector[0:0]

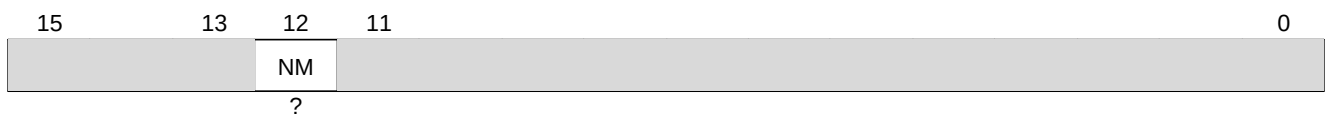


Figure 1804. 16-bit opcode fields

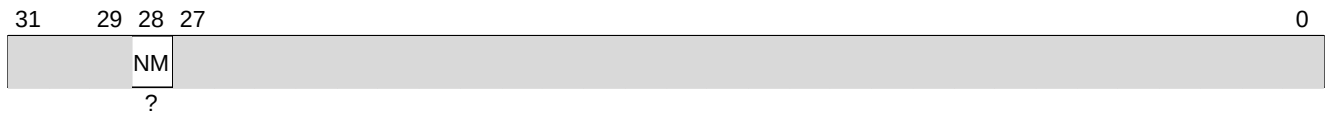


Figure 1805. 32-bit opcode fields

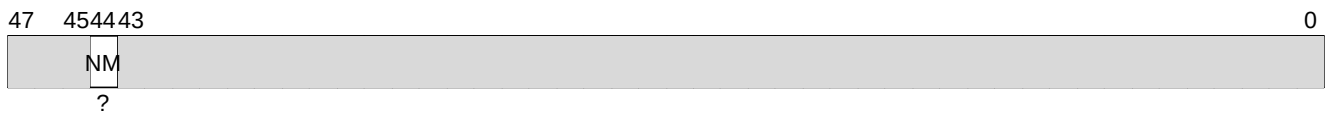


Figure 1806. 48-bit opcode fields

High / low	0	1
?	P32 pool	P16 pool

D.2. P32 pool

Pool path

NM / P32

Selector width

5 bits

Encoding width

32, 48 bits

Selector layout

32-bit: instruction[31:29] → selector[4:2]; instruction[27:26] → selector[1:0]; 48-bit: instruction[47:45] → selector[4:2]; instruction[43:42] → selector[1:0]

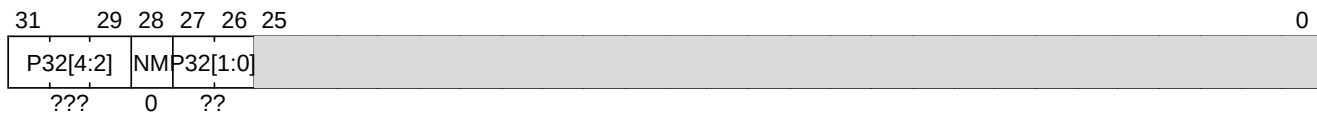


Figure 1807. 32-bit opcode fields

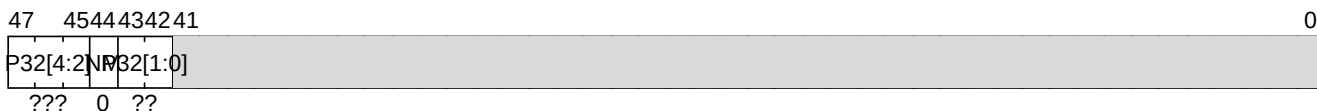


Figure 1808. 48-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	7 options	4 options	P.MOVE.BALC / MOVE.BALC	—	10 options	—	BC.32 / P.BAL pool / BC	—
01???	P.GP.D pool / P.GP.W pool	6 options	P32.JRC / P.J pool / JALRC	—	P48 pool / P48I pool	—	—	—
10???	4 options	9 options	P.BR1 pool / P32B pool	—	POOL32F pool	25 options	P.BR2 pool	—
11???	EXTD / EXTD32 / POOL32S pool	—	P.BRI pool	—	4 options	—	—	—

D.2.1. Selector 00000

This selector has 7 possible interpretations across the declared revisions, features, and processor states:

- ADDIU[32]
- SIGRIE[32]
- P32.BREAK pool
- P32.SDBBP pool
- P32.SYSCALL pool
- ADDIU

- SIGRIE

D.2.2. Selector 00001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDIUPC[32]
- DADDIUPC[32]
- ADDIUPC
- DADDIUPC

D.2.3. Selector 00100

This selector has 10 possible interpretations across the declared revisions, features, and processor states:

- DERET.32
- RDHWR[32]
- WAIT[32]
- P32A pool
- P32A.CP0 pool
- P32A.ERET pool
- P32A.TLB pool
- DERET
- RDHWR
- WAIT

D.2.4. Selector 01001

This selector has 6 possible interpretations across the declared revisions, features, and processor states:

- P32GP
- P.GP.BH pool
- LDC1
- LWC1
- SDC1
- SWC1

D.2.5. Selector 10000

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- P.ROTX pool
- P.U12 pool
- P32.SHIFT pool
- P32A.U12 pool

D.2.6. Selector 10001

This selector has 9 possible interpretations across the declared revisions, features, and processor states:

- P32U12
- SYNCI[U12]
- P.LS.U12 pool
- P32.LSU pool
- LDC1
- LWC1
- SDC1
- SWC1
- SYNCI

D.2.7. Selector 10101

This selector has 25 possible interpretations across the declared revisions, features, and processor states:

- CACHE.EVA
- CACHE.S9
- LDM
- LWM
- P32S9
- SDM
- SWM
- SYNCI[EVA]
- SYNCI[S9]
- UALDM
- UALWM

- UASDM
- UASWM
- P.LS.S9 pool
- P32.LS.PAIR pool
- P32.LS.S9 pool
- P32A.EVA pool
- P32A.MEM pool
- CACHE
- LDC1
- LWC1
- SDC1
- SWC1
- SYNCI
- SYNCIE

D.2.8. Selector 11100

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ALUIPC.32
- P.LUI
- ALUIPC
- LUI

D.3. P32.BREAK pool

Pool path

NM / P32 / P32.BREAK

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:19] → selector[1:0]



Figure 1809. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	BREAK[32] / BREAK	—

D.4. P32.SDBBP pool

Pool path

NM / P32 / P32.SDBBP

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:19] → selector[1:0]

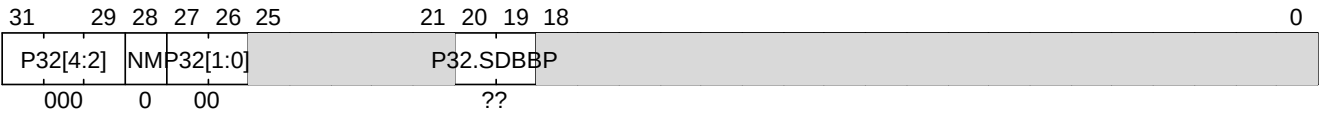


Figure 1810. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	—	SDBBP[32] / SDBBP

D.5. P32.SYSCALL pool

Pool path

NM / P32 / P32.SYSCALL

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:19] → selector[1:0]

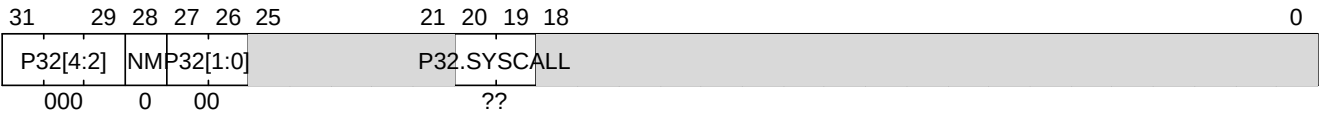


Figure 1811. 32-bit opcode fields

High / low	00	01	10	11
??	—	SYSCALL[32] / SYSCALL	—	—

D.6. P32A pool

Pool path

NM / P32 / P32A

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[2:0] → selector[2:0]

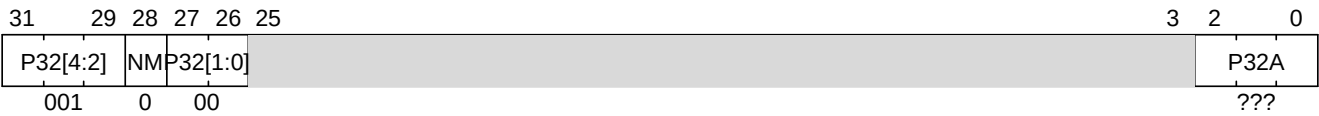


Figure 1812. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	_POOL32A0 pool / CRC32 pool	—	—	—	—	_POOL32A5 pool	—	_POOL32A7 pool

D.7. _POOL32A0 pool

Pool path

NM / P32 / P32A / _POOL32A0

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:5] → selector[0:0]

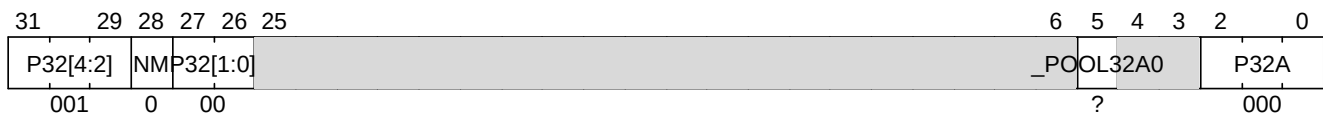


Figure 1813. 32-bit opcode fields

High / low	0	1
?	DVP / EVP / _POOL32A0_0 pool	_POOL32A0_1 pool

D.8. _POOL32A0_0 pool

Pool path

NM / P32 / P32A / _POOL32A0 / _POOL32A0_0

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:6] → selector[3:0]

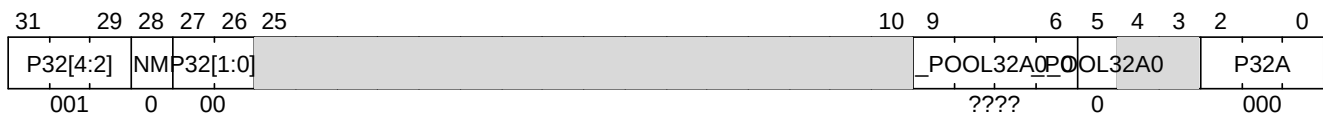


Figure 1814. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	10 options	6 options	4 options	4 options	4 options	4 options	4 options	4 options
1???	MOVN/MOVZ [32] / MOVN / MOVZ	AND[32] / AND	OR[32] / OR	NOR[32] / NOR	XOR[32] / XOR	SLT[32] / SLT	SLTU[32] / SLTU	SOV[32] / SOV

D.8.1. Selector 0000

This selector has 10 possible interpretations across the declared revisions, features, and processor states:

- MUL[32]
- SEB[32]

- SLLV[32]
- TEQ[32]
- TNE[32]
- MUL
- SEB
- SLLV
- TEQ
- TNE

D.8.2. Selector 0001

This selector has 6 possible interpretations across the declared revisions, features, and processor states:

- MUH[32]
- SEH[32]
- SRLV[32]
- MUH
- SEH
- SRLV

D.8.3. Selector 0010

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MULU[32]
- SRAV[32]
- MULU
- SRAV

D.8.4. Selector 0011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MUHU[32]
- ROTRV[32]
- MUHU
- ROTRV

D.8.5. Selector 0100

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADD[32]
- DIV[32]
- ADD
- DIV

D.8.6. Selector 0101

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDU[32]
- MOD[32]
- ADDU
- MOD

D.8.7. Selector 0110

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- DIVU[32]
- SUB[32]
- DIVU
- SUB

D.8.8. Selector 0111

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- MODU[32]
- SUBU[32]
- MODU
- SUBU

D.9. _POOL32A0_1 pool

Pool path

NM / P32 / P32A / _POOL32A0 / _POOL32A0_1

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:6] → selector[5:2]; instruction[4:3] → selector[1:0]

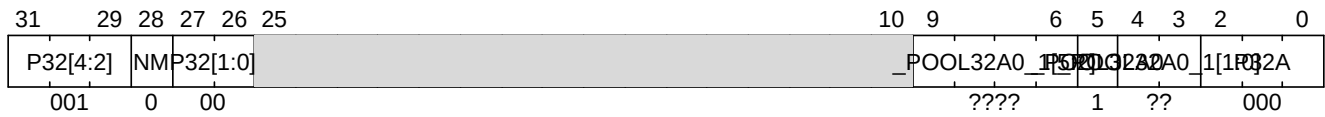


Figure 1815. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	—	MFC0[32] / MFC0	MFHC0[32] / MFHC0	—	—	MTC0[32] / MTC0	MTHC0[32] / MTHC0
001???	—	—	—	—	—	—	—	—
010???	—	—	DMFC0	—	—	—	DMTC0	—
011???	—	—	—	—	—	—	—	—
100???	—	—	—	—	—	—	—	—
101???	—	—	—	—	—	—	—	—
110???	—	—	—	—	—	—	—	—
111???	—	—	—	—	—	—	—	—

D.10. CRC32 pool

Pool path

NM / P32 / P32A / CRC32

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[12:10] → selector[2:0]



Figure 1816. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	CRC32B	CRC32H	CRC32W	CRC32D	CRC32CB	CRC32CH	CRC32CW	CRC32CD

D.11. _POOL32A5 pool

Pool path

NM / P32 / P32A / _POOL32A5

Selector width

7 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:3] → selector[6:0]

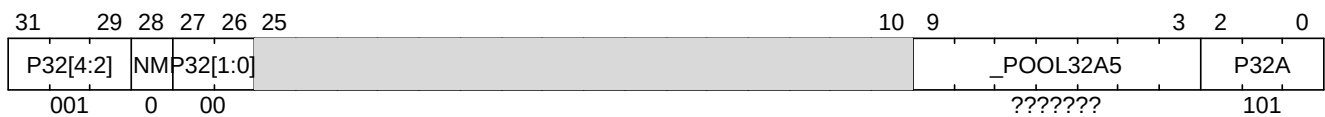


Figure 1817. 32-bit opcode fields

High / low	????000	????001	????010	????011	????100	????101	????110	????111
0000???	CMP.EQ.PH	ADDQ.PH / ADDQ_S.PH	—	SHILO	1 options	MUL.PH / MUL_S.PH	—	REPL.PH
0001???	CMP.LT.PH	ADDQH.PH / ADDQH_R.P H	—	—	1 options	PRECR.QB. PH	—	—
0010???	CMP.LE.PH	ADDQH.W / ADDQH_R.W	1 options	—	—	PRECRQ.QB .PH	—	—
0011???	CMPGU.EQ. QB	ADDU.QB / ADDU_S.QB	1 options	—	—	PRECRQ.PH .W	—	—
0100???	CMPGU.LT. QB	ADDU.PH / ADDU_S.PH	MULQ_RS.P H	—	—	1 options	—	—
0101???	CMPGU.LE. QB	ADDUH.QB / ADDUH_R.Q B	MULQ_S.PH	—	—	1 options	—	—
0110???	CMPGDU.EQ .QB	SHRAV.PH / SHRAV_R.P H	MULQ_RS.W	—	—	PACKRL.PH	—	—
0111???	CMPGDU.LT .QB	SHRAV.QB / SHRAV_R.Q B	MULQ_S.W	—	—	PICK.QB	—	—

High / low	????000	????001	????010	????011	????100	????101	????110	????111
1000???	CMPGDU.LE.QB	SUBQ.PH / SUBQ_S.PH	APPEND	—	—	PICK.PH	—	—
1001???	CMPU.EQ.QB	SUBQH.PH / SUBQH_R.PH	—	—	—	—	—	—
1010???	CMPU.LT.QB	SUBQH.W / SUBQH_R.W	MODSUB	—	—	—	—	—
1011???	CMPU.LE.QB	SUBU.QB / SUBU_S.QB	SHRAV_R.W	—	—	—	SHRA_R.W	—
1100???	ADDQ_S.W	SUBU.PH / SUBU_S.PH	SHRLV.PH	—	—	—	SHRA.PH / SHRA_R.PH	—
1101???	SUBQ_S.W	SUBUH.QB / SUBUH_R.QB	SHRLV.QB	—	—	—	—	—
1110???	ADDSC	SHLLV.PH / SHLLV_S.PH	SHLLV.QB	—	—	—	SHLL.PH / SHLL_S.PH	—
1111???	ADDWC	2 options	SHLLV_S.W	—	—	—	SHLL_S.W	—

D.11.1. Selector 0000100

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEQ_S.W.PHL

D.11.2. Selector 0001100

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEQ_S.W.PHR

D.11.3. Selector 0010010

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEU_S.PH.QBL

D.11.4. Selector 0011010

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULEU_S.PH.QBR

D.11.5. Selector 0100101

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECRQ_RS.PH.W

D.11.6. Selector 0101101

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECRQU_S.QB.PH

D.11.7. Selector 1111001

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- PRECR_SRA.PH.W
- PRECR_SRA_R.PH.W

D.12. _POOL32A7 pool

Pool path

NM / P32 / P32A / _POOL32A7

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:3] → selector[2:0]

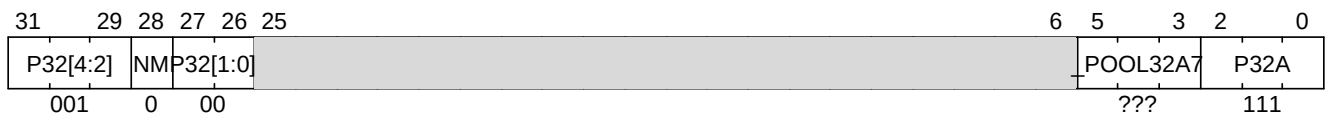


Figure 1818. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	PLSX pool	LSA[32] / LSA	—	EXTW[32] / EXTW	—	—	—	POOL32A xf pool

D.13. P.LSX pool

Pool path

NM / P32 / P32A / _POOL32A7 / P.LSX

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[6:6] → selector[0:0]

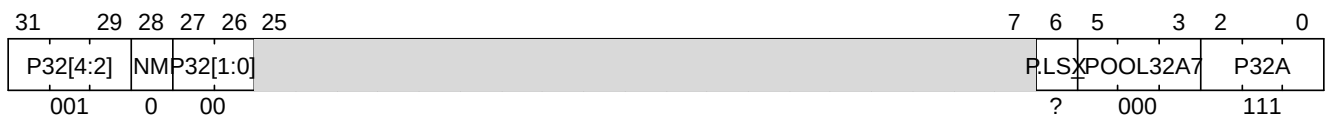


Figure 1819. 32-bit opcode fields

High / low	0	1
?	PP.LSX pool	PP.LSXS pool

D.14. PP.LSX pool

Pool path

NM / P32 / P32A / _POOL32A7 / P.LSX / PP.LSX

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:7] → selector[3:0]

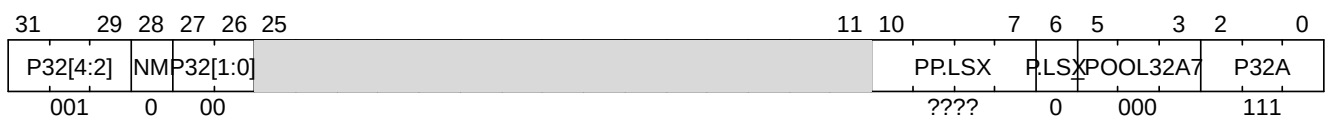


Figure 1820. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	LBX	SBX	LBUX	—	LHX	SHX	LHUX	LWUX
1???	LWX	SWX	—	—	LDX	SDX	—	—

D.15. PP.LSXS pool

Pool path

NM / P32 / P32A / _POOL32A7 / P.LSX / PP.LSXS

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:7] → selector[3:0]

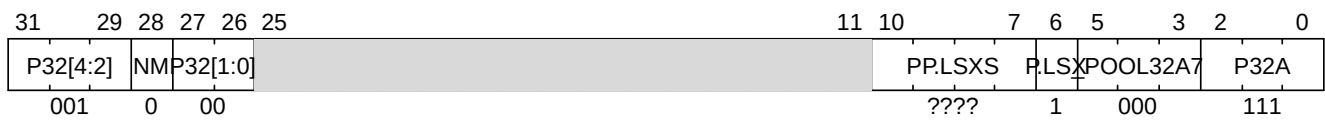


Figure 1821. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	LHXS	SHXS	LHUXS	LWUXS
1???	LWXS	SWXS	—	—	LDXS	SDXS	—	—

D.16. POOL32Axf pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[8:6] → selector[2:0]

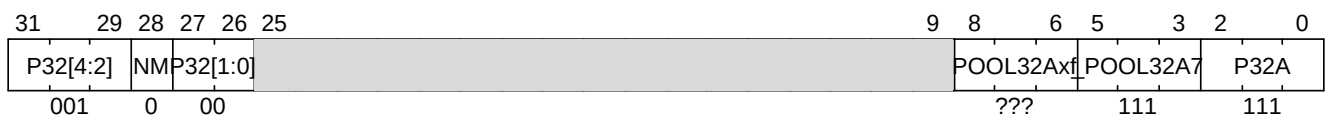


Figure 1822. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	POOL32Axf_1 pool	POOL32Axf_2 pool	—	POOL32Axf_4 pool	P.DI pool / POOL32Axf_5 pool	—	REPL.QB / POOL32Axf_7 pool

D.17. POOL32Axf_1 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_1

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[11:9] → selector[2:0]

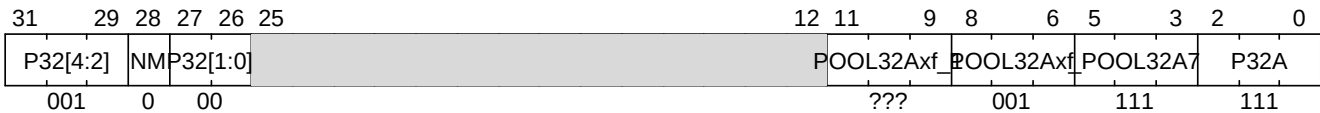


Figure 1823. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	1 options	1 options	—	1 options	1 options	1 options	—	1 options

D.17.1. Selector 000

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [POOL32Axf_1_0 pool](#)

D.17.2. Selector 001

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [POOL32Axf_1_1 pool](#)

D.17.3. Selector 011

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [POOL32Axf_1_3 pool](#)

D.17.4. Selector 100

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [POOL32Axf_1_4 pool](#)

D.17.5. Selector 101

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [POOL32Axf_1_5 pool](#)

D.17.6. Selector 111

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [POOL32Axf_1_7 pool](#)

D.18. POOL32Axf_1_0 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_1 / POOL32Axf_1_0

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:12] → selector[1:0]

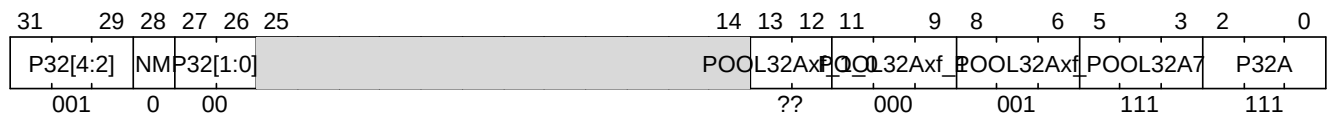


Figure 1824. 32-bit opcode fields

High / low	00	01	10	11
??	MFHI.DSP / MFHI	MFLO.DSP / MFLO	MTHI.DSP / MTHI	MTLO.DSP / MTLO

D.19. POOL32Axf_1_1 pool

Pool path

NM / P32 / P32A / _P00L32A7 / P00L32Axf / P00L32Axf_1 / P00L32Axf_1_1

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:12] $\rightarrow$ selector[1:0]

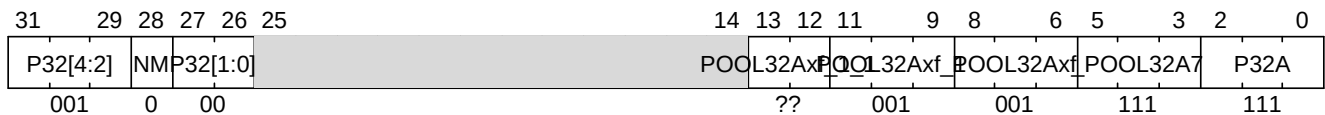


Figure 1825. 32-bit opcode fields

High / low	00	01	10	11
??	MTHLIP	SHILOV	—	—

D.20. POOL32Axf_1_3 pool

Pool path

NM / P32 / P32A / P00L32A7 / P00L32Axf / P00L32Axf 1 / P00L32Axf 1 3

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:12] $\rightarrow$ selector[1:0]

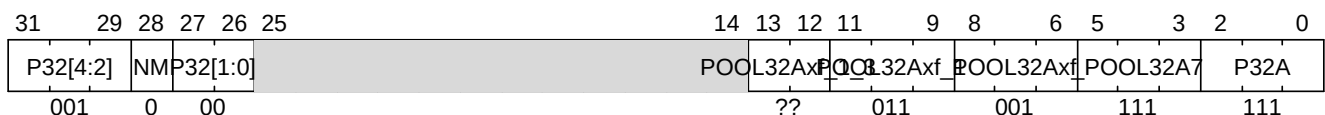


Figure 1826. 32-bit opcode fields

High / low	00	01	10	11
??	RDDSP	WRDSP	EXTP	EXTPDP

D.21. POOL32Axf_1_4 pool

Pool path

NM / P32 / P32A / POOL32A7 / POOL32Axf / POOL32Axf 1 / POOL32Axf 1 4

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[12:12] → selector[0:0]

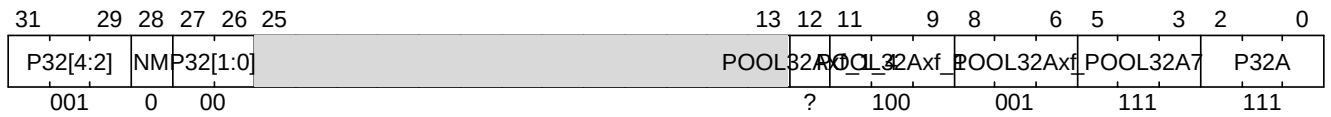


Figure 1827. 32-bit opcode fields

High / low	0	1
?	SHLL.QB	SHRL.QB

D.22. POOL32Axf_1_5 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_1 / POOL32Axf_1_5

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[12:12] → selector[0:0]

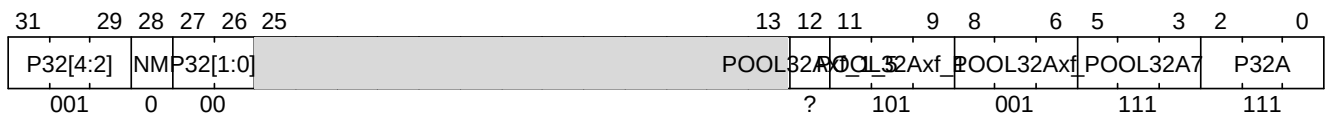


Figure 1828. 32-bit opcode fields

High / low	0	1
?	1 options	1 options

D.22.1. Selector 0

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [MAQ_S\[A\].W.PHR pool](#)

D.22.2. Selector 1

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- [MAQ_S\[A\].W.PHL pool](#)

D.23. MAQ_S[A].W.PHR pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_1 / POOL32Axf_1_5 / MAQ_S[A].W.PHR

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:13] → selector[0:0]

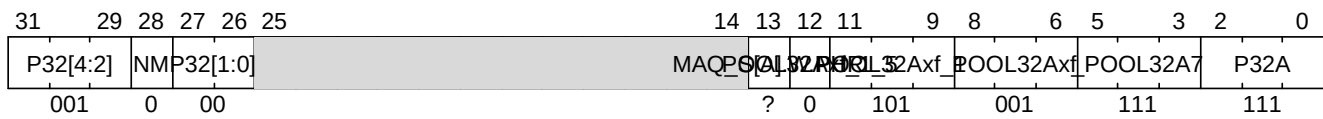


Figure 1829. 32-bit opcode fields

High / low	0	1
?	MAQ_S.W.PHR	MAQ_SA.W.PHR

D.24. MAQ_S[A].W.PHL pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_1 / POOL32Axf_1_5 / MAQ_S[A].W.PHL

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:13] → selector[0:0]

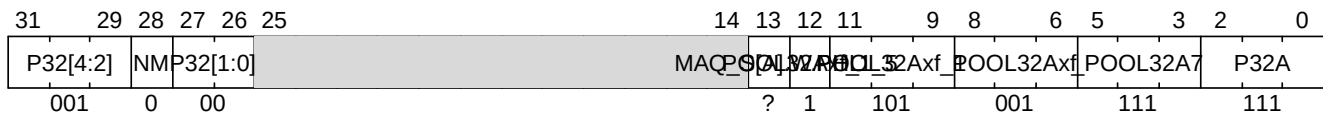


Figure 1830. 32-bit opcode fields

High / low	00	01	10	11
??	POOL32Axf_2(DSP)_0_7 pool	POOL32Axf_2(DSP)_8_15 pool	POOL32Axf_2(DSP)_16_23 pool	POOL32Axf_2(DSP)_24_31 pool

D.27. POOL32Axf_2(DSP)_0_7 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_2 / POOL32Axf_2(DSP)_0_7

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:11] → selector[2:0]



Figure 1833. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	DPA.W.PH	DPAQ_S.W. PH	DPS.W.PH	DPSQ_S.W. PH	—	MADD	MULT	EXTRV.W

D.28. POOL32Axf_2(DSP)_8_15 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_2 / POOL32Axf_2(DSP)_8_15

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:11] → selector[2:0]



Figure 1834. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	DPAX.W.PH	DPAQ_SA.L .W	DPSX.W.PH	DPSQ_SA.L .W	—	MADDU	MULTU	EXTRV_R.W

D.29. POOL32Axf_2(DSP)_16_23 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_2 / POOL32Axf_2(DSP)_16_23

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:11] → selector[2:0]

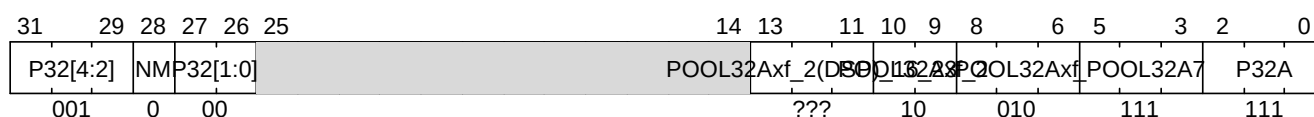


Figure 1835. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	DPAU.H.QB L	DPAQX_S.W .PH	DPSU.H.QB L	DPSQX_S.W .PH	EXTPV	MSUB	MULSA.W.P H	EXTRV_RS. W

D.30. POOL32Axf_2(DSP)_24_31 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_2 / POOL32Axf_2(DSP)_24_31

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:11] → selector[2:0]

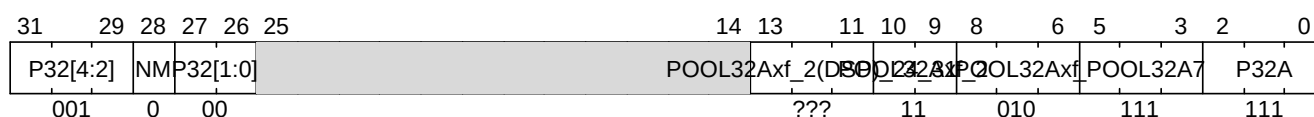


Figure 1836. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	DPAU.H.QBR	1 options	DPSU.H.QBR	1 options	EXTPDPV	MSUBU	1 options	EXTRV_S.H

D.30.1. Selector 001

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- DPAQX_SA.W.PH

D.30.2. Selector 011

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- DPSQX_SA.W.PH

D.30.3. Selector 110

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- MULSAQ_S.W.PH

D.31. POOL32Axf_4 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_4

Selector width

7 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:9] → selector[6:0]

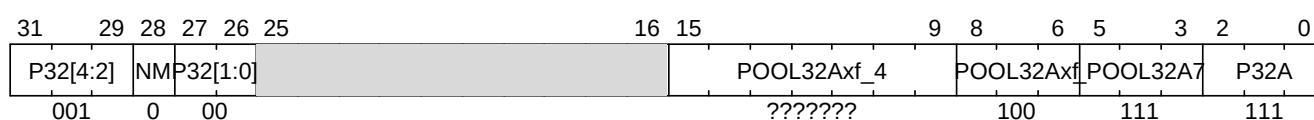


Figure 1837. 32-bit opcode fields

High / low	????000	????001	????010	????011	????100	????101	????110	????111
0000???	ABSQ_S.QB	REPLV.PH	—	—	—	—	—	—

High / low	????000	????001	????010	????011	????100	????101	????110	????111
0001???	ABSQ_S.PH	REPLV.QB	—	—	—	—	—	—
0010???	ABSQ_S.W	—	—	—	—	—	—	—
0011???	BITREV	—	—	—	—	—	—	—
0100???	INSV	—	—	—	—	CLO[32] / CLO	—	—
0101???	PRECEQ.W. PHL	—	—	—	—	CLZ[32] / CLZ	—	—
0110???	PRECEQ.W. PHR	—	—	—	—	—	—	—
0111???	1 options	1 options	—	—	—	—	—	—
1000???	—	—	—	—	—	—	—	—
1001???	1 options	1 options	—	—	—	—	—	—
1010???	—	—	—	—	—	—	—	—
1011???	1 options	1 options	—	—	—	—	—	—
1100???	—	—	—	—	—	—	—	—
1101???	1 options	1 options	—	—	—	—	—	—
1110???	—	—	—	—	—	—	—	—
1111???	RADDU.W.Q B	—	—	—	—	—	—	—

D.31.1. Selector 0111000

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEQU.PH.QBL

D.31.2. Selector 0111001

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEQU.PH.QBLA

D.31.3. Selector 1001000

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEQU.PH.QBR

D.31.4. Selector 1001001

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEQU.PH.QBRA

D.31.5. Selector 1011000

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEU.PH.QBL

D.31.6. Selector 1011001

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEU.PH.QBLA

D.31.7. Selector 1101000

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEU.PH.QBR

D.31.8. Selector 1101001

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- PRECEU.PH.QBRA

D.32. P.DI pool

Pool path

NM / P32 / P32A / _P00L32A7 / P00L32Axf / P.DI

Selector width

7 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:14] → selector[6:5]; instruction[13:9] → selector[4:0]

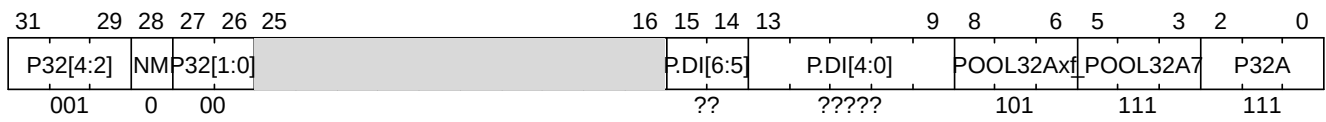


Figure 1838. 32-bit opcode fields

High / low	????000	????001	????010	????011	????100	????101	????110	????111
0000???	—	—	—	—	—	—	—	—
0001???	—	—	—	—	—	—	—	—
0010???	—	—	—	—	—	—	—	—
0011???	—	—	—	—	—	—	—	—
0100???	—	—	—	DI.32 / DI	—	—	—	—
0101???	—	—	—	EI.32 / EI	—	—	—	—
0110???	—	—	—	—	—	—	—	—
0111???	—	—	—	—	—	—	—	—
1000???	—	—	—	—	—	—	—	—
1001???	—	—	—	—	—	—	—	—
1010???	—	—	—	—	—	—	—	—
1011???	—	—	—	—	—	—	—	—
1100???	—	—	—	—	—	—	—	—
1101???	—	—	—	—	—	—	—	—
1110???	—	—	—	—	—	—	—	—
1111???	—	—	—	—	—	—	—	—

D.33. POOL32Axf_5 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_5

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:14] → selector[1:0]

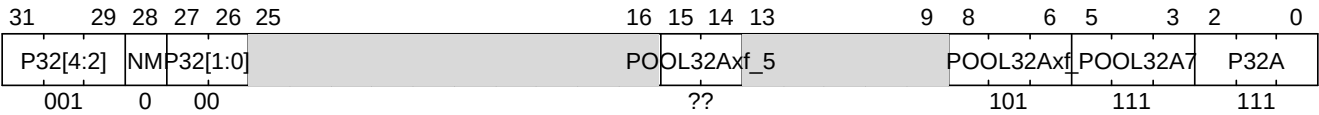


Figure 1839. 32-bit opcode fields

High / low	00	01	10	11
??	1 options	—	—	—

D.33.1. Selector 00

This selector has 1 possible interpretation across the declared revisions, features, and processor states:

- POOL32Axf_5_group0 pool

D.34. POOL32Axf_5_group0 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_5 / POOL32Axf_5_group0

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:9] → selector[4:0]

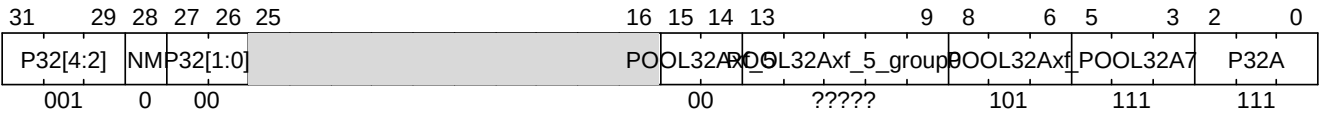


Figure 1840. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	—	—	—	GINVT
01???	—	—	—	—	—	—	—	GINVI
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

D.35. POOL32Axf_7 pool

Pool path

NM / P32 / P32A / _POOL32A7 / POOL32Axf / POOL32Axf_7

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[11:9] → selector[2:0]

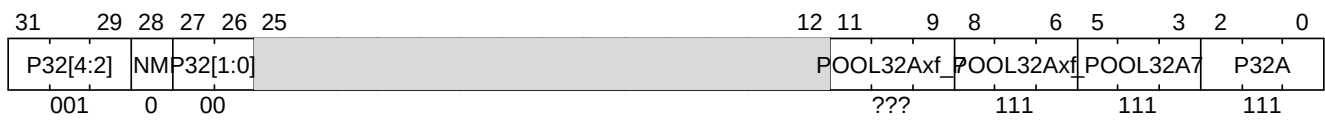


Figure 1841. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	SHRA.QB / SHRA_R.QB	SHRL.PH	—	—	—	—	—	—

D.36. P32A.CP0 pool

Pool path

NM / P32 / P32A.CP0

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:14] → selector[1:0]

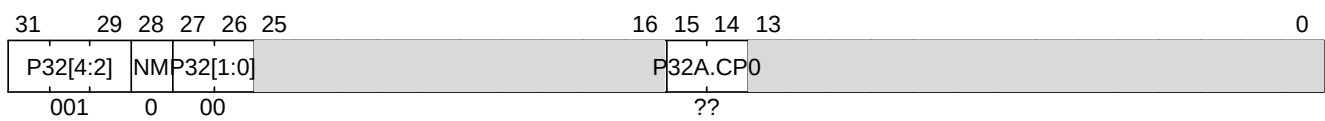


Figure 1842. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	—	P.CP0 pool

D.37. P.CP0 pool

Pool path

NM / P32 / P32A.CP0 / P.CP0

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:9] → selector[4:0]

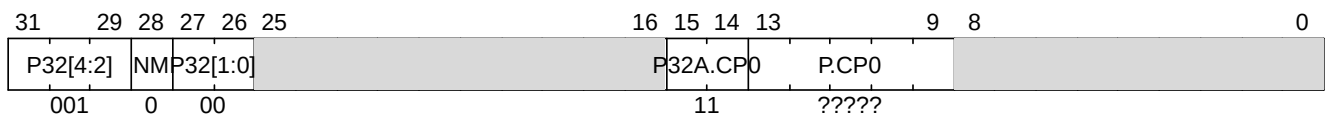


Figure 1843. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	RDPGPR[32] / RDPGPR	—	—	—	—	—	—	—
11???	WRPGPR[32] / WRPGPR	—	—	—	—	—	—	—

D.38. P32A.ERET pool

Pool path

NM / P32 / P32A.ERET

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:14] → selector[1:0]

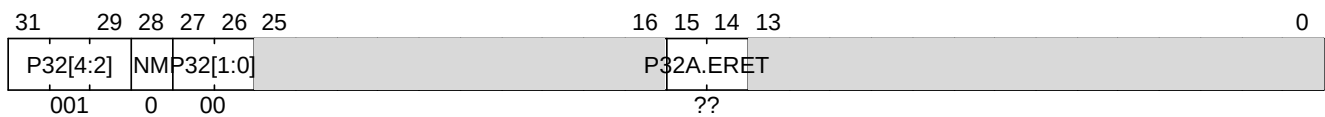


Figure 1844. 32-bit opcode fields

High / low	00	01	10	11
??	—	—	—	P.ERET pool

D.39. P.ERET pool

Pool path

NM / P32 / P32A.ERET / P.ERET

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:9] → selector[4:0]

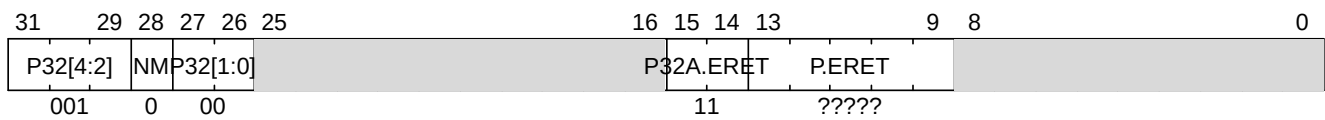


Figure 1845. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	4 options	—	—	—	—	—	—

D.39.1. Selector 11001

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ERET[32]
- ERETNC[32]
- ERET
- ERETNC

D.40. P32A.TLB pool

Pool path

NM / P32 / P32A.TLB

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:14] → selector[1:0]

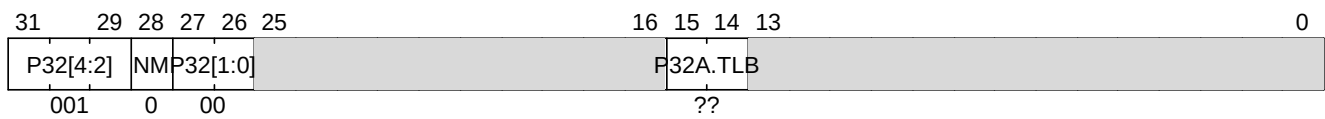


Figure 1846. 32-bit opcode fields

High / low	00	01	10	11
??	P.TLB pool	—	—	—

D.41. P.TLB pool

Pool path

NM / P32 / P32A.TLB / P.TLB

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:9] → selector[4:0]

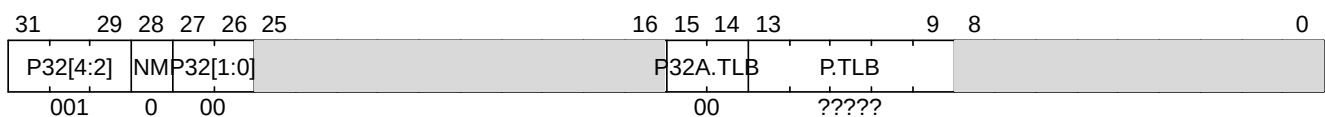


Figure 1847. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	TLBP[32] / TLBP	—	TLBINV[32] / TLBINV	—	—	—	—
01???	—	TLBR[32] / TLBR	—	TLBINVF[32] / TLBINVF	—	—	—	—
10???	—	TLBWI[32] / TLBWI	—	—	—	—	—	—
11???	—	TLBWR[32] / TLBWR	—	—	—	—	—	—

D.42. P.BAL pool

Pool path

NM / P32 / P.BAL

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[25:25] → selector[0:0]

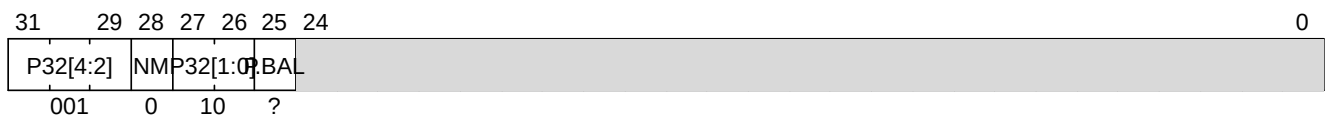


Figure 1848. 32-bit opcode fields

High / low	0	1
?	—	BALC[32] / BALC

D.43. P.GP.D pool

Pool path

NM / P32 / P.GP.D

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[1:0] → selector[1:0]

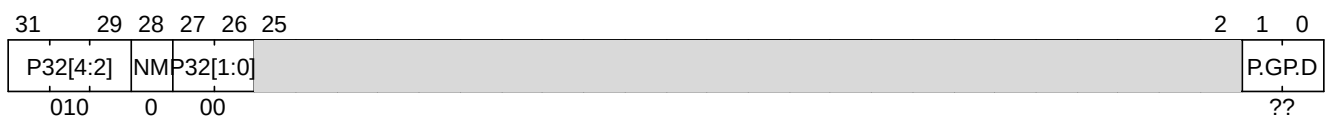


Figure 1849. 32-bit opcode fields

High / low	00	01	10	11
??	—	4 options	—	—

D.43.1. Selector 01

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- LD[GP]
- SD[GP]
- LD
- SD

D.44. P.GP.W pool

Pool path

NM / P32 / P.GP.W

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[1:0] → selector[1:0]

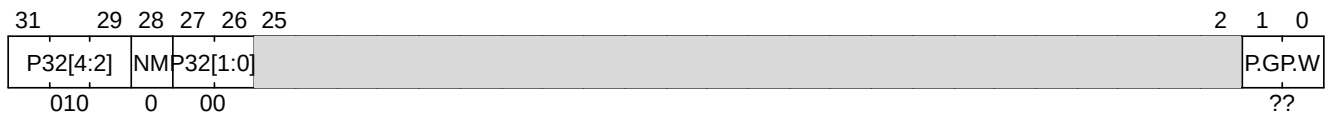


Figure 1850. 32-bit opcode fields

High / low	00	01	10	11
??	4 options	—	LW[GP] / LW	SW[GP] / SW

D.44.1. Selector 00

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDIU[GP.W]
- DADDIU[GP.W]
- ADDIU
- DADDIU

D.45. P.GP.BH pool

Pool path

NM / P32 / P.GP.BH

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:18] → selector[2:0]

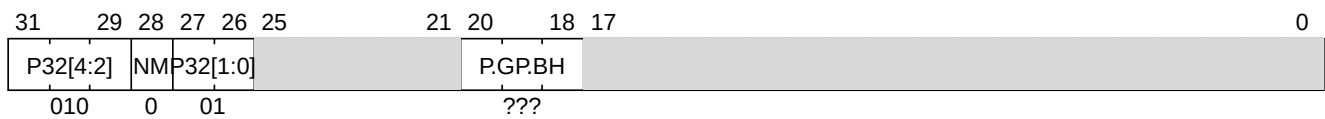


Figure 1851. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	LB[GP] / LB	SB[GP] / SB	LBU[GP] / LBU	4 options	4 options	SH[GP] / SH	—	P.GP.M64 pool

D.45.1. Selector 011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDIU[GP.B]
- DADDIU[GP.B]
- ADDIU
- DADDIU

D.45.2. Selector 100

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- LH[GP]
- LHU[GP]
- LH
- LHU

D.46. P.GP.M64 pool

Pool path

NM / P32 / P.GP.BH / P.GP.M64

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[1:0] → selector[1:0]

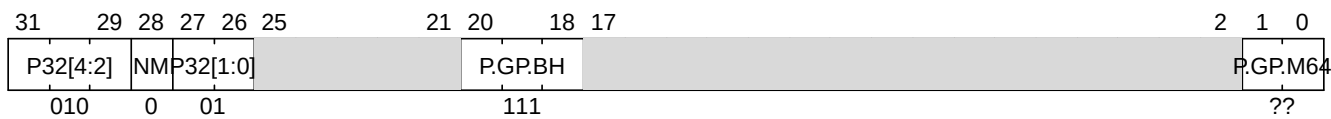


Figure 1852. 32-bit opcode fields

High / low	00	01	10	11
??	LWU[GP] / LWU	—	—	—

D.47. P.J pool

Pool path

NM / P32 / P.J

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

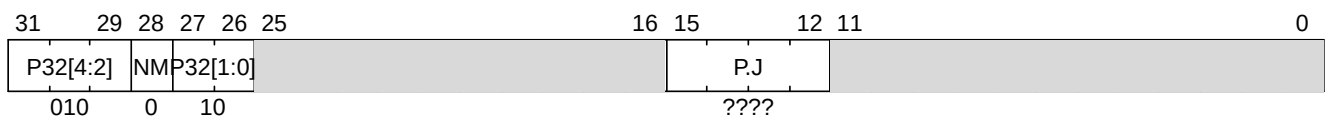


Figure 1853. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	—	—	—	—
1???	BALRSC / BRSC	—	—	—	—	—	—	—

D.48. P48 pool

Pool path

NM / P32 / P48

Selector width

5 bits

Encoding width

48 bits

Selector layout

48-bit: instruction[36:32] → selector[4:0]

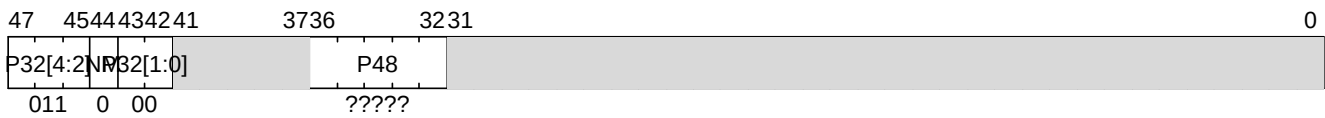


Figure 1854. 48-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	4 options	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	LDPC[48] / LDPC	—	—	—	SDPC[48] / SDPC

D.48.1. Selector 00011

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDIUPC[48]
- DADDIUPC[48]
- ADDIUPC
- DADDIUPC

D.49. P48I pool

Pool path

NM / P32 / P48I

Selector width

5 bits

Encoding width
48 bits

Selector layout
48-bit: instruction[36:32] → selector[4:0]

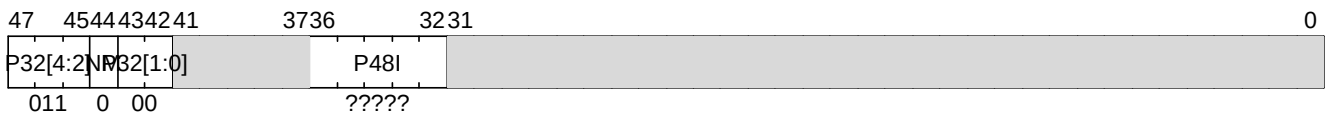


Figure 1855. 48-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	LI[48] / LI	ADDIU[48] / ADDIU	4 options	—	—	—	—	—
01???	—	—	—	LWPC[48] / LWPC	—	—	—	SWPC[48] / SWPC
10???	—	DADDIU[48] / DADDIU	—	—	DLUI[48] / DLUI	—	—	—
11???	—	—	—	—	—	—	—	—

D.49.1. Selector 00010

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDIU[GP48]
- DADDIU[GP48]
- ADDIU
- DADDIU

D.50. P.ROTX pool

Pool path
NM / P32 / P.ROTX

Selector width
3 bits

Encoding width
32 bits

Selector layout
32-bit: instruction[15:13] → selector[2:0]

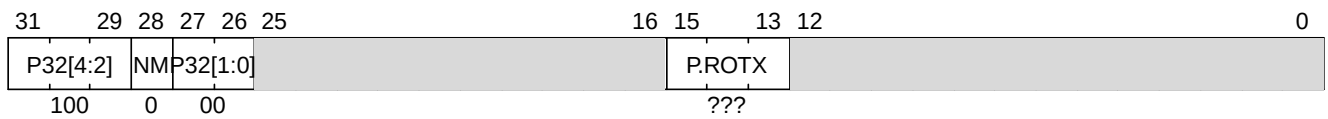


Figure 1856. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	—	—	—	—	DROTX	—	—

D.51. P.U12 pool

Pool path

NM / P32 / P.U12

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

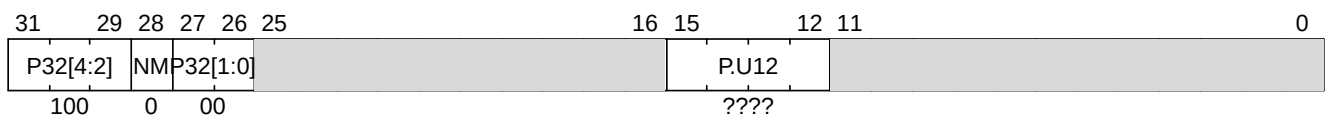


Figure 1857. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	ORI[32] / ORI	XORI[32] / XORI	ANDI[32] / ANDI	P.SR pool	—	—	SEQI[32] / SEQI	DADDIU[U12] / DADDIU
1???	ADDIU[NEG] / ADDIU	DADDIU[NEG] / DADDIU	—	—	P.SHIFT pool	ROTX[32] / ROTX	6 options	EXT[32] / P.EXT pool / EXT

D.51.1. Selector 1110

This selector has 6 possible interpretations across the declared revisions, features, and processor states:

- INS[32]
- P.INS
- DINS
- DINSM

- DINSU
- INS

D.52. P.SR pool

Pool path

NM / P32 / P.U12 / P.SR

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:20] → selector[0:0]

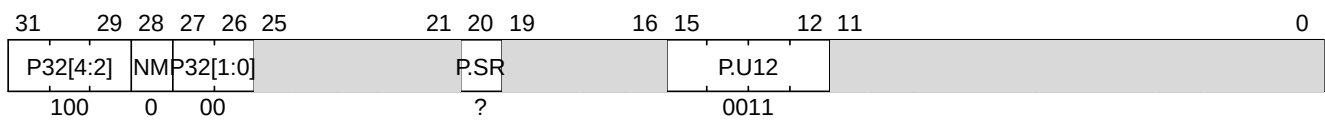


Figure 1858. 32-bit opcode fields

High / low	0	1
?	PP.SR pool	—

D.53. PP.SR pool

Pool path

NM / P32 / P.U12 / P.SR / PP.SR

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[1:0] → selector[1:0]

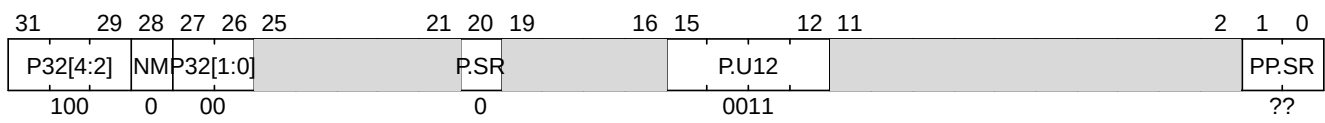


Figure 1859. 32-bit opcode fields

High / low	00	01	10	11
??	SAVE[32] / SAVE	—	RESTORE[32] / RESTORE	2 options

D.53.1. Selector 11

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- `RESTORE.JRC[32]`
- `RESTORE.JRC`

D.54. P.SHIFT pool

Pool path

`NM / P32 / P.U12 / P.SHIFT`

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[8:5] → selector[3:0]

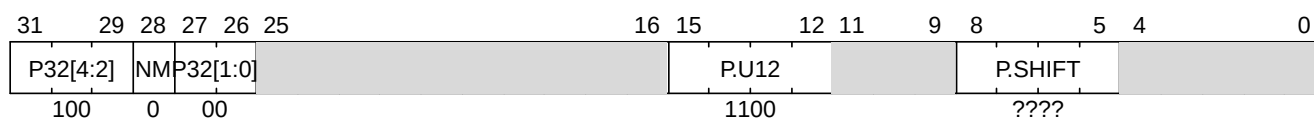


Figure 1860. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	9 options	—	SRL[32] / SRL	—	SRA[32] / SRA	—	ROTR[32] / ROTR	—
1???	P.SLL / DSLL	P.SLL / DSLL32	P.SLL / DSRL	P.SLL / DSRL32	P.SLL / DSRA	P.SLL / DSRA32	P.SLL / DROTR	P.SLL / DROTR32

D.54.1. Selector 0000

This selector has 9 possible interpretations across the declared revisions, features, and processor states:

- `EHB`
- `NOP[32]`
- `PAUSE[32]`

- SLL[32]
- SYNC[32]
- NOP
- PAUSE
- SLL
- SYNC

D.55. P.EXT pool

Pool path

NM / P32 / P.U12 / P.EXT

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[11:11] → selector[1:1]; instruction[5:5] → selector[0:0]

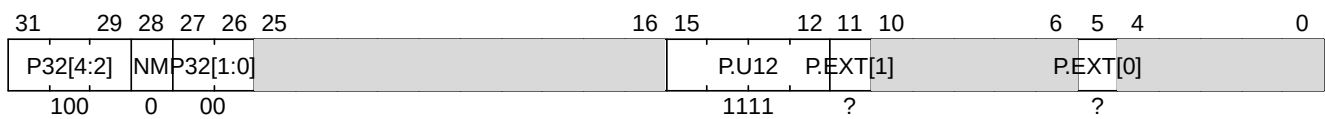


Figure 1861. 32-bit opcode fields

High / low	00	01	10	11
??	—	DEXTU	DEXTM	DEXT

D.56. P32.SHIFT pool

Pool path

NM / P32 / P32.SHIFT

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

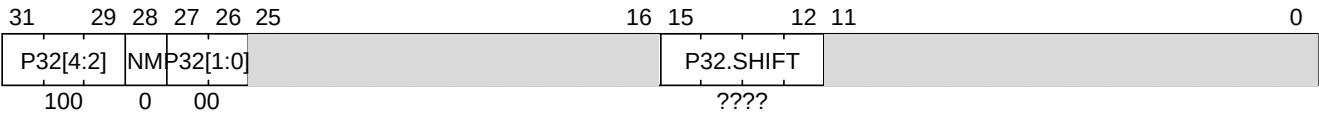


Figure 1862. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	—	—	—	—
1???	—	—	—	—	—	ROTX[32] / WSBH	—	—

D.57. P32A.U12 pool

Pool path

NM / P32 / P32A.U12

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

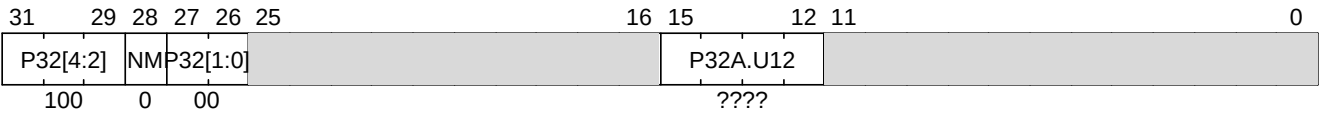


Figure 1863. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	SLTI[32] / SLTI	SLTIU[32] / SLTIU	—	—
1???	—	—	—	—	—	—	—	—

D.58. P.LS.U12 pool

Pool path

NM / P32 / P.LS.U12

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

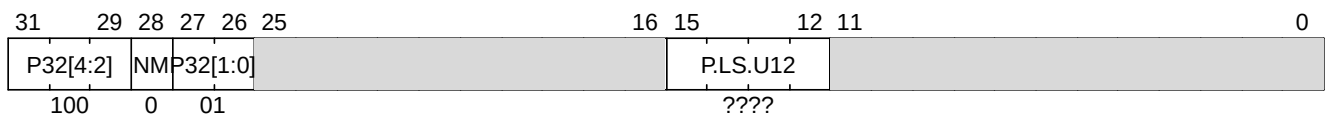


Figure 1864. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	PREF[U12] / PREF	—	—	—	—
1???	—	—	—	—	—	—	—	—

D.59. P32.LSU pool

Pool path

NM / P32 / P32.LSU

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:12] → selector[3:0]

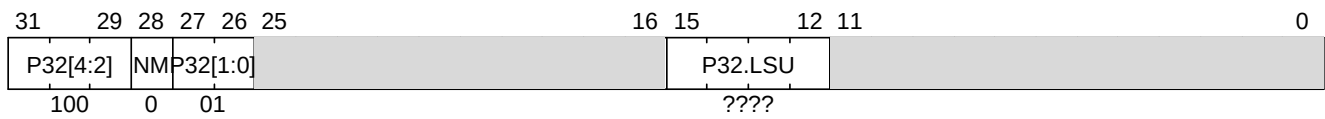


Figure 1865. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	LB[U12] / LB	SB[U12] / SB	LBU[U12] / LBU	—	LH[U12] / LH	SH[U12] / SH	LHU[U12] / LHU	LWU[U12] / LWU
1???	LW[U12] / LW	SW[U12] / SW	—	—	LD[U12] / LD	SD[U12] / SD	—	—

D.60. P.BR1 pool

Pool path

NM / P32 / P.BR1

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:14] → selector[1:0]

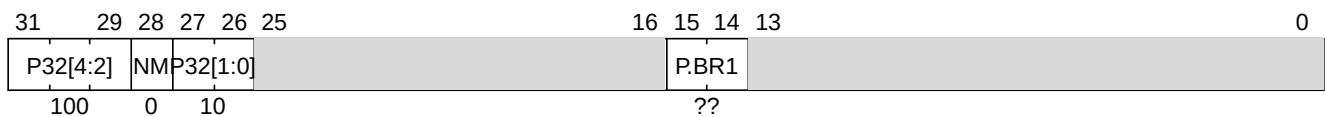


Figure 1866. 32-bit opcode fields

High / low	00	01	10	11
??	BEQC[32] / BEQC	P.BR3A pool	BGEC / BGEZC[32] / BGEZC	BGEUC

D.61. P.BR3A pool

Pool path

NM / P32 / P.BR1 / P.BR3A

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:16] → selector[4:0]

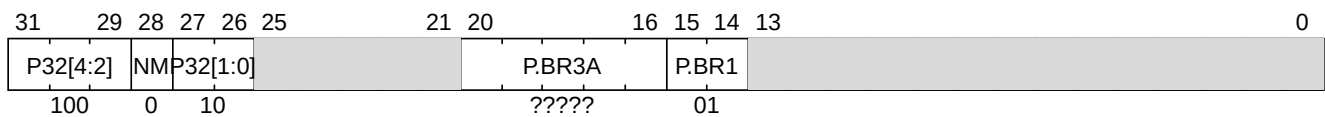


Figure 1867. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	—	—	—	—	BPOSGE32C	—	—	—
01???	—	—	—	—	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

D.62. P32B pool

Pool path

NM / P32 / P32B

Selector width

5 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:16] → selector[4:0]

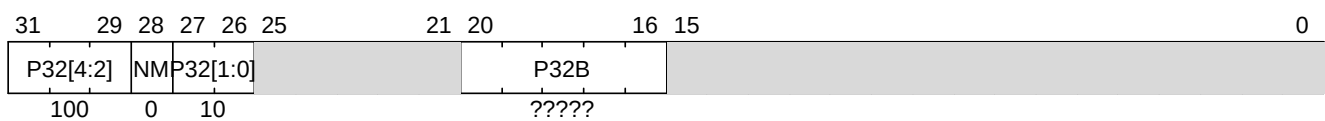


Figure 1868. 32-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	BC1EQZC	BC1NEZC	—	—	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

D.63. POOL32F pool

Pool path

NM / P32 / POOL32F

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[2:0] → selector[2:0]

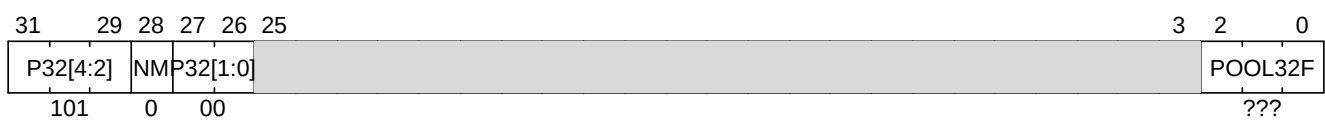


Figure 1869. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	POOL32F_0 pool	—	—	POOL32F_3 pool	—	CMP.fmt pool	—	—

D.64. POOL32F_0 pool

Pool path

NM / P32 / POOL32F / POOL32F_0

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[8:3] → selector[5:0]

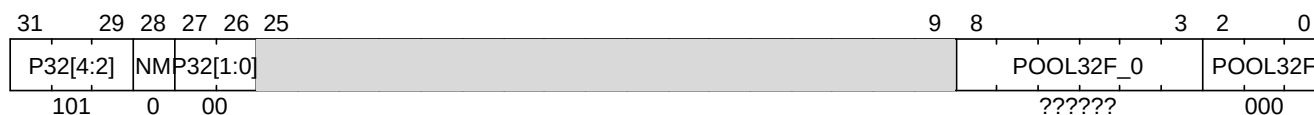


Figure 1870. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	—	—	—	RINT.fmt pool	—	ADD.S	SELEQZ.fmt pool
001???	—	—	—	—	CLASS.fmt pool	—	SUB.S	SELNEZ.fmt pool
010???	—	—	—	—	—	—	MUL.S	SEL.fmt pool
011???	—	—	—	—	—	—	DIV.S	—
100???	—	—	—	—	—	—	ADD.D	—
101???	—	—	—	—	—	—	SUB.D	—
110???	—	—	—	—	—	—	MUL.D	MADDF.fmt pool
111???	—	—	—	—	—	—	DIV.D	MSUBF.fmt pool

D.65. RINT.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_0 / RINT.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]



Figure 1871. 32-bit opcode fields

High / low	0	1
?	RINT.S	RINT.D

D.66. SELEQZ.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_0 / SELEQZ.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

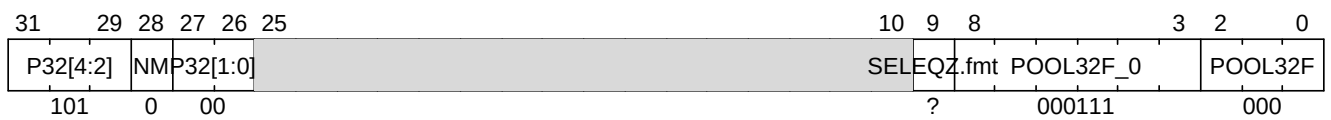


Figure 1872. 32-bit opcode fields

High / low	0	1
?	SELEQZ.S	SELEQZ.D

D.67. CLASS.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_0 / CLASS.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

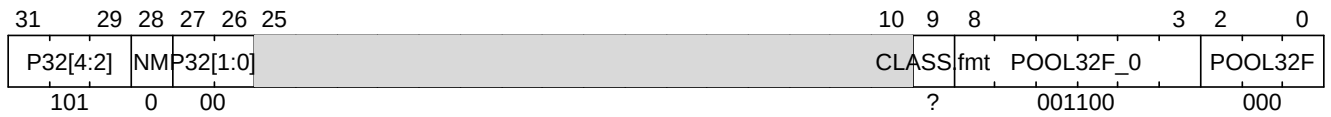


Figure 1873. 32-bit opcode fields

High / low	0	1
?	CLASS.S	CLASS.D

D.68. SELNEZ.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_0 / SELNEZ.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

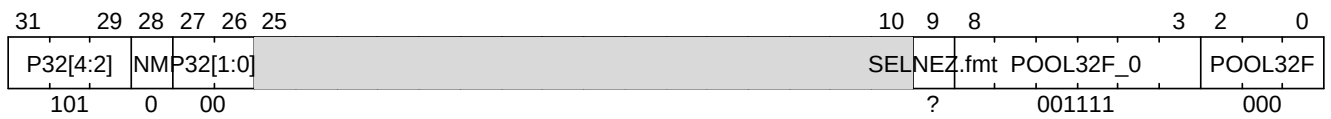


Figure 1874. 32-bit opcode fields

High / low	0	1
?	SELNEZ.S	SELNEZ.D

D.69. SEL.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_0 / SEL.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

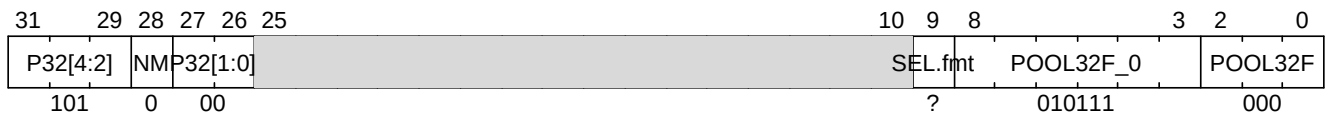


Figure 1875. 32-bit opcode fields

High / low	0	1
?	SEL.S	SEL.D

D.70. MADDF.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_0 / MADDF.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

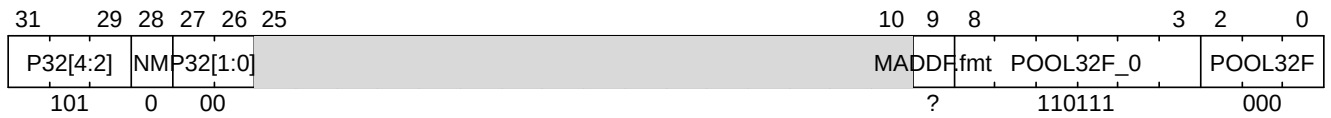


Figure 1876. 32-bit opcode fields

High / low	0	1
?	MADDF.S	MADDF.D

D.71. MSUBF.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_0 / MSUBF.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

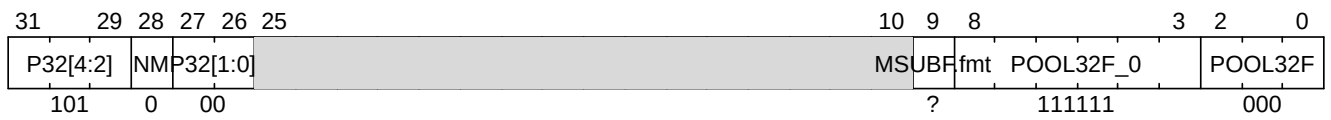


Figure 1877. 32-bit opcode fields

High / low	0	1
?	MSUBF.S	MSUBF.D

D.72. POOL32F_3 pool

Pool path

NM / P32 / POOL32F / POOL32F_3

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:3] → selector[2:0]



Figure 1878. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	MIN.fmt pool	MAX.fmt pool	—	—	MINA.fmt pool	MAXA.fmt pool	—	POOL32F xf pool

D.73. MIN.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / MIN.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

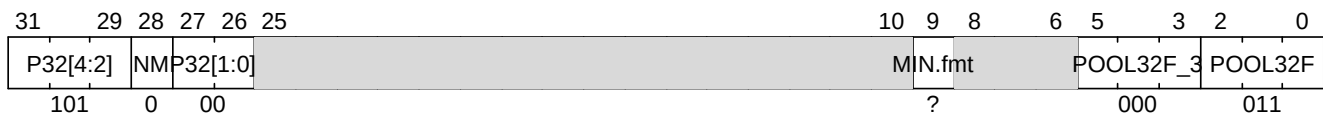


Figure 1879. 32-bit opcode fields

High / low	0	1
?	MIN.S	MIN.D

D.74. MAX.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / MAX.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]



Figure 1880. 32-bit opcode fields

High / low	0	1
?	MAX.S	MAX.D

D.75. MINA.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / MINA.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

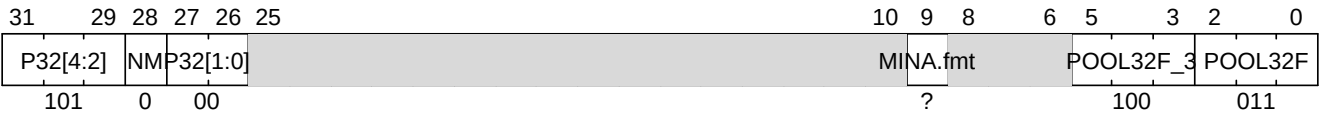


Figure 1881. 32-bit opcode fields

High / low	0	1
?	MINA.S	MINA.D

D.76. MAXA.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / MAXA.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[9:9] → selector[0:0]

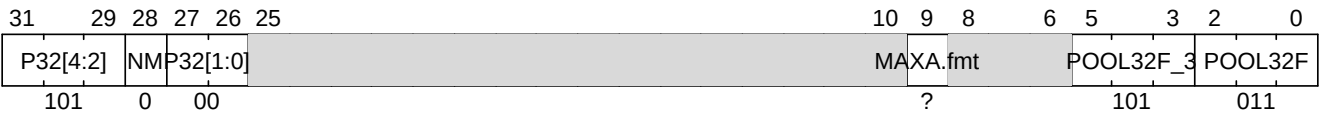


Figure 1882. 32-bit opcode fields

High / low	0	1
?	MAXA.S	MAXA.D

D.77. POOL32Fxf pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[7:6] → selector[1:0]

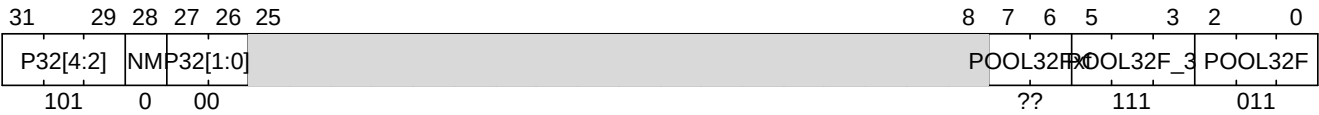


Figure 1883. 32-bit opcode fields

High / low	00	01	10	11
??	POOL32Fxf_0 pool	POOL32Fxf_1 pool	—	—

D.78. POOL32Fxf_0 pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:8] → selector[5:0]

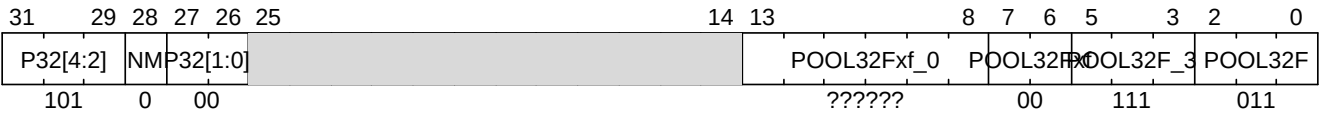


Figure 1884. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	CVT.L.D.fmt pool / CVT.L.S.fmt pool	RSQRT.fmt pool	2 options	—	—	—	—
001???	—	CVT.W.D.fmt pool / CVT.W.S.fmt pool	SQRT.fmt pool	2 options	—	—	—	—
010???	CFC1	—	RECIP.fmt pool	CEIL.L.D.fmt pool / CEIL.L.S.fmt pool	—	—	—	—
011???	CTC1	—	—	CEIL.W.D.fmt pool / CEIL.W.S.fmt pool	—	—	—	—

High / low	???000	???001	???010	???011	???100	???101	???110	???111
100???	MFC1	—	—	2 options	DMFC1	—	—	—
101???	MTC1	—	—	2 options	DMTC1	—	—	—
110???	MFHC1	—	—	2 options	—	—	—	—
111???	MTHC1	—	—	2 options	—	—	—	—

D.78.1. Selector 000011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- [FLOOR.L.D.fmt pool](#)
- [FLOOR.L.S.fmt pool](#)

D.78.2. Selector 001011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- [FLOOR.W.D.fmt pool](#)
- [FLOOR.W.S.fmt pool](#)

D.78.3. Selector 100011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- [TRUNC.L.D.fmt pool](#)
- [TRUNC.L.S.fmt pool](#)

D.78.4. Selector 101011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- [TRUNC.W.D.fmt pool](#)
- [TRUNC.W.S.fmt pool](#)

D.78.5. Selector 110011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- [ROUND.L.D.fmt pool](#)
- [ROUND.L.S.fmt pool](#)

D.78.6. Selector 111011

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- [ROUND.W.D.fmt pool](#)
- [ROUND.W.S.fmt pool](#)

D.79. CVT.L.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CVT.L.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

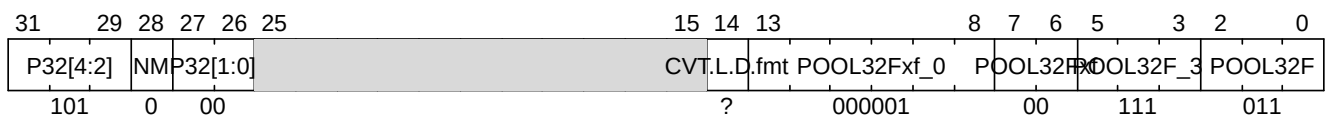


Figure 1885. 32-bit opcode fields

High / low	0	1
?	—	CVT.L.D

D.80. CVT.L.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CVT.L.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

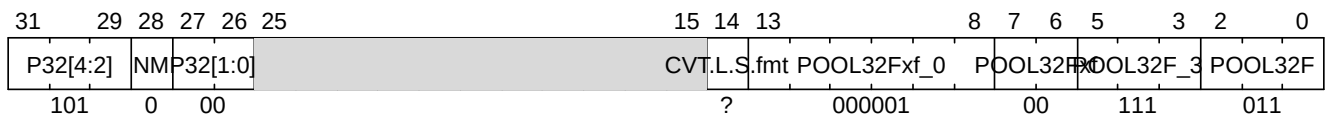


Figure 1886. 32-bit opcode fields

High / low	0	1
?	CVT.L.S	—

D.81. RSQRT.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / RSQRT.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

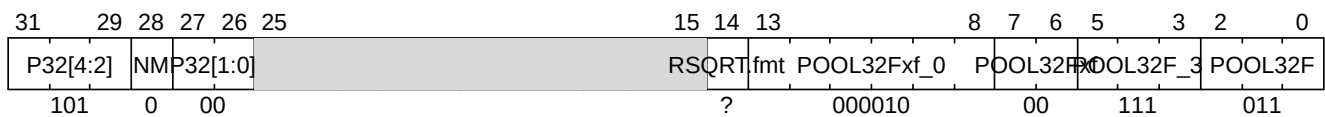


Figure 1887. 32-bit opcode fields

High / low	0	1
?	RSQRT.S	RSQRT.D

D.82. FLOOR.L.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / FLOOR.L.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

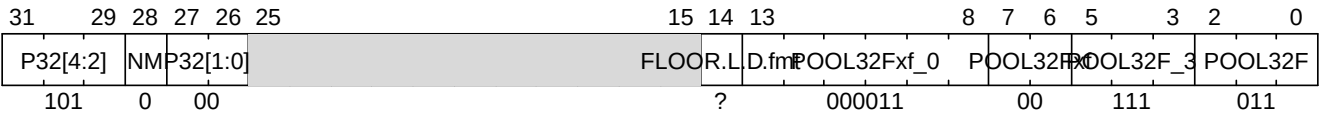


Figure 1888. 32-bit opcode fields

High / low	0	1
?	—	FLOOR.L.D

D.83. FLOOR.L.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / FLOOR.L.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

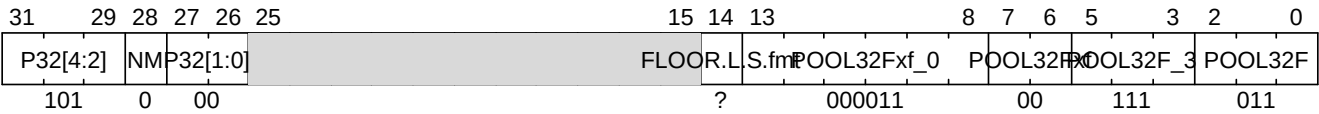


Figure 1889. 32-bit opcode fields

High / low	0	1
?	FLOOR.L.S	—

D.84. CVT.W.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CVT.W.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

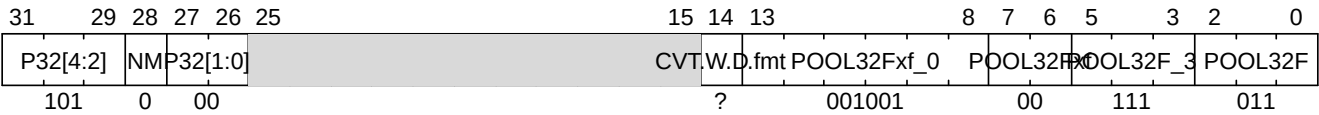


Figure 1890. 32-bit opcode fields

High / low	0	1
?	—	CVT.W.D

D.85. CVT.W.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CVT.W.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

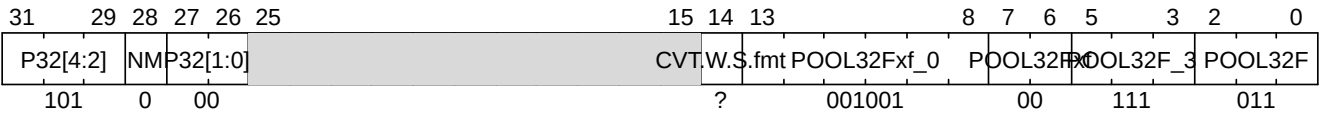


Figure 1891. 32-bit opcode fields

High / low	0	1
?	CVT.W.S	—

D.86. SQRT.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / SQRT.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

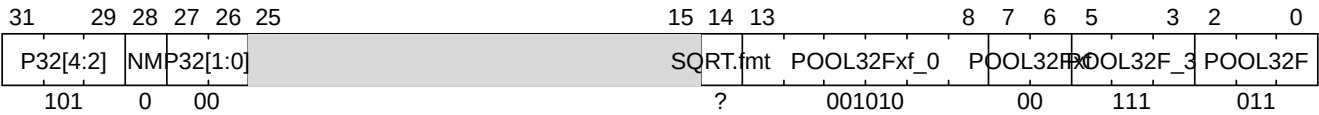


Figure 1892. 32-bit opcode fields

High / low	0	1
?	SQRD.S	SQRD.D

D.87. FLOOR.W.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / FLOOR.W.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

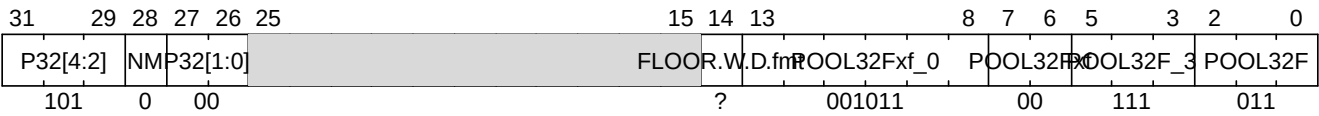


Figure 1893. 32-bit opcode fields

High / low	0	1
?	—	FLOOR.W.D

D.88. FLOOR.W.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / FLOOR.W.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

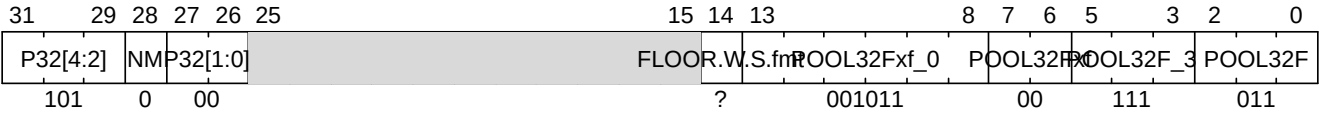


Figure 1894. 32-bit opcode fields

High / low	0	1
?	FLOOR.W.S	—

D.89. RECIP.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / RECIP.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

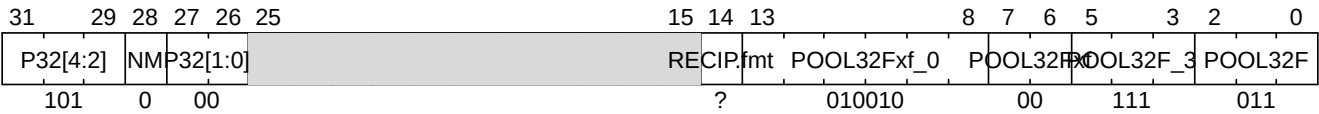


Figure 1895. 32-bit opcode fields

High / low	0	1
?	RECIP.S	RECIP.D

D.90. CEIL.L.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CEIL.L.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

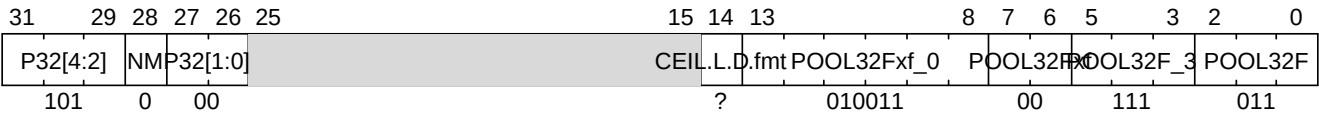


Figure 1896. 32-bit opcode fields

High / low	0	1
?	—	CEIL.L.D

D.91. CEIL.L.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CEIL.L.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

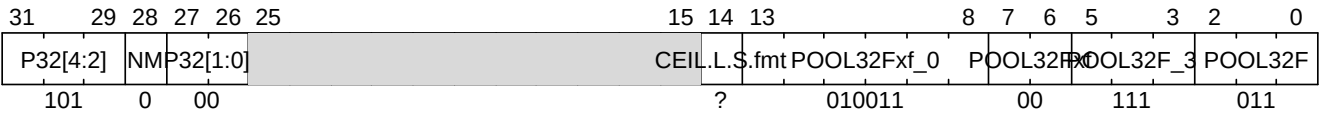


Figure 1897. 32-bit opcode fields

High / low	0	1
?	CEIL.L.S	—

D.92. CEIL.W.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CEIL.W.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

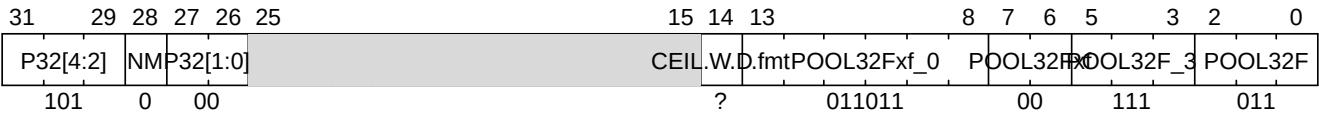


Figure 1898. 32-bit opcode fields

High / low	0	1
?	—	CEIL.W.D

D.93. CEIL.W.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / CEIL.W.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

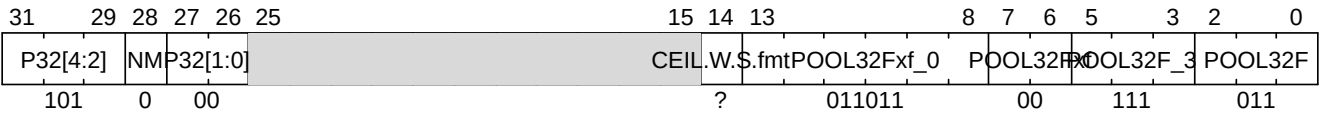


Figure 1899. 32-bit opcode fields

High / low	0	1
?	CEIL.W.S	—

D.94. TRUNC.L.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / TRUNC.L.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

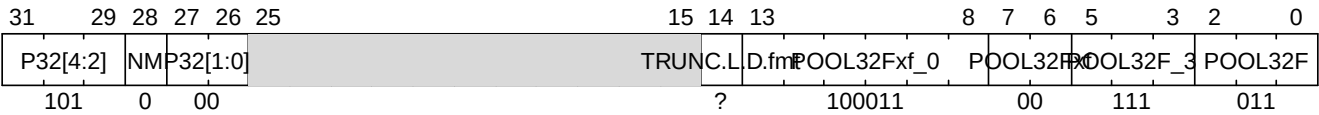


Figure 1900. 32-bit opcode fields

High / low	0	1
?	—	TRUNC.L.D

D.95. TRUNC.L.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / TRUNC.L.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

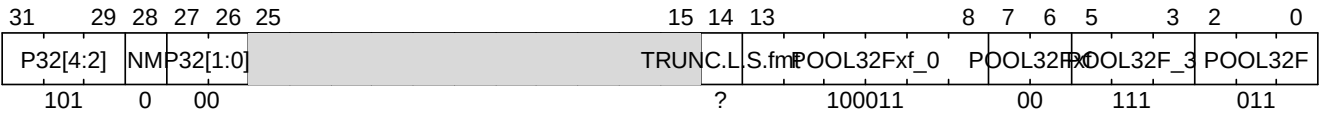


Figure 1901. 32-bit opcode fields

High / low	0	1
?	TRUNC.L.S	—

D.96. TRUNC.W.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / TRUNC.W.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

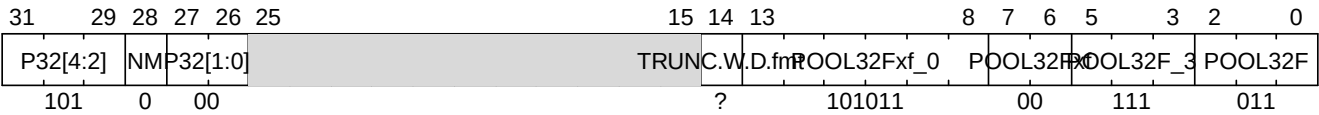


Figure 1902. 32-bit opcode fields

High / low	0	1
?	—	TRUNC.W.D

D.97. TRUNC.W.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / TRUNC.W.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

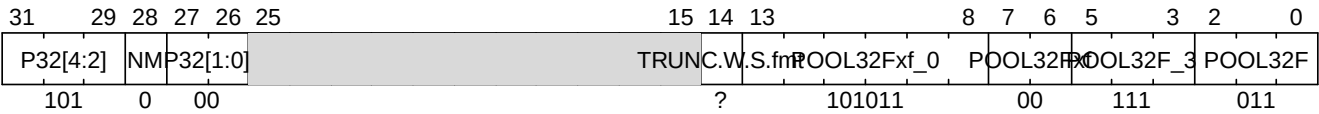


Figure 1903. 32-bit opcode fields

High / low	0	1
?	TRUNC.W.S	—

D.98. ROUND.L.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / ROUND.L.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

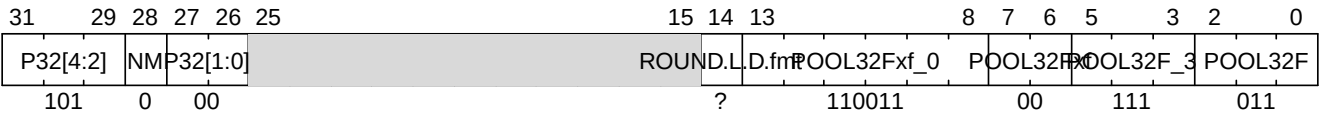


Figure 1904. 32-bit opcode fields

High / low	0	1
?	—	ROUND.L.D

D.99. ROUND.L.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / ROUND.L.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

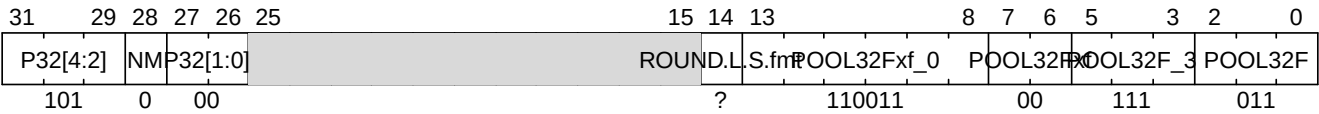


Figure 1905. 32-bit opcode fields

High / low	0	1
?	ROUND.L.S	—

D.100. ROUND.W.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / ROUND.W.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

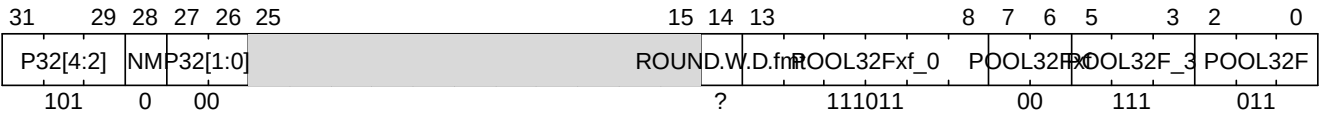


Figure 1906. 32-bit opcode fields

High / low	0	1
?	—	ROUND.W.D

D.101. ROUND.W.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_0 / ROUND.W.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

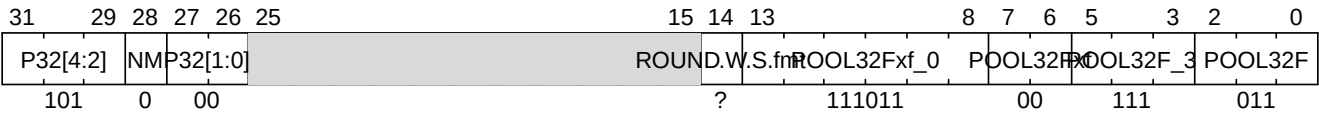


Figure 1907. 32-bit opcode fields

High / low	0	1
?	ROUND.W.S	—

D.102. POOL32Fxf_1 pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_1

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[13:8] → selector[5:0]

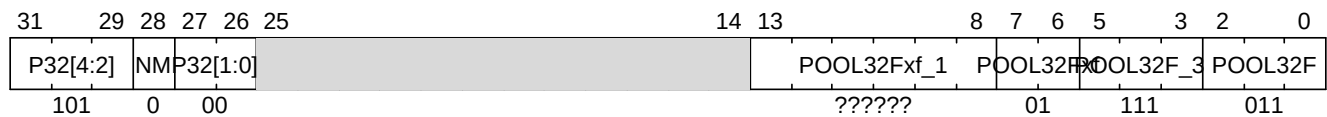


Figure 1908. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	MOV.S	—	—	ABS.S	—	—	—	—
001???	—	—	—	NEG.S	—	—	—	—
010???	—	—	—	CVT.D.L.f mt pool / CVT.D.S.f mt pool	—	—	—	—
011???	—	—	—	CVT.S.D.f mt pool / CVT.S.L.f mt pool	—	—	—	—
100???	MOV.D	—	—	ABS.D	—	—	—	—
101???	—	—	—	NEG.D	—	—	—	—
110???	—	—	—	CVT.D.W.f mt pool	—	—	—	—
111???	—	—	—	CVT.S.W.f mt pool	—	—	—	—

D.103. CVT.D.L.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_1 / CVT.D.L.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

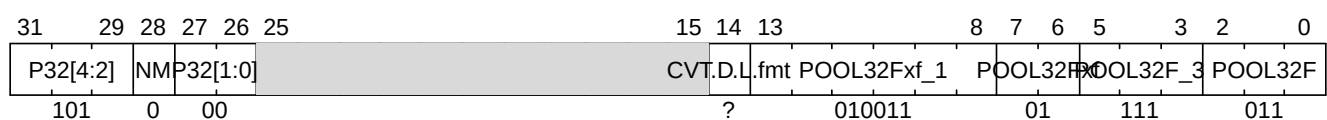


Figure 1909. 32-bit opcode fields

High / low	0	1
?	—	CVT.D.L

D.104. CVT.D.S.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_1 / CVT.D.S.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

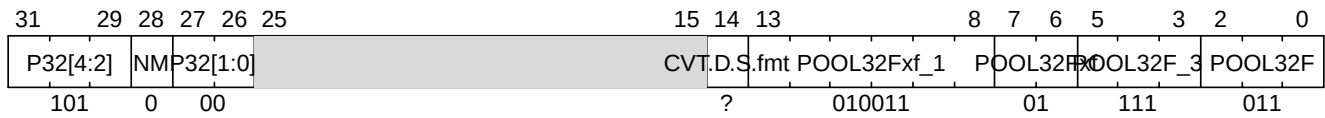


Figure 1910. 32-bit opcode fields

High / low	0	1
?	CVT.D.S	—

D.105. CVT.S.D.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_1 / CVT.S.D.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

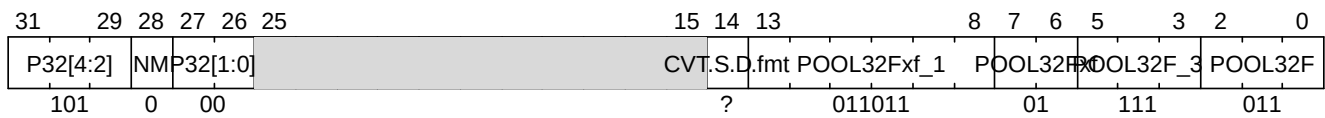


Figure 1911. 32-bit opcode fields

High / low	0	1
?	CVT.S.D	—

D.106. CVT.S.L.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_1 / CVT.S.L.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

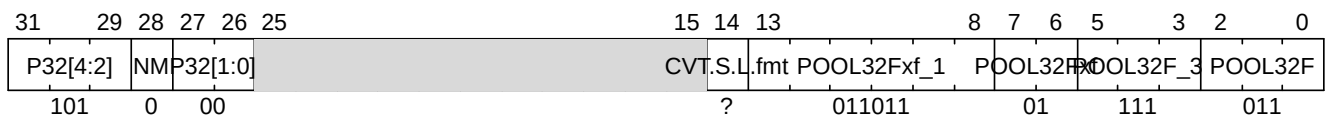


Figure 1912. 32-bit opcode fields

High / low	0	1
?	—	CVT.S.L

D.107. CVT.D.W.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_1 / CVT.D.W.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

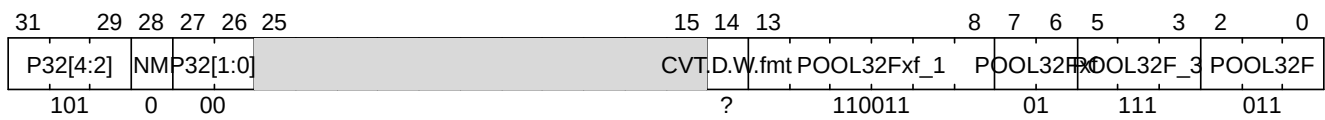


Figure 1913. 32-bit opcode fields

High / low	0	1
?	CVT.D.W	—

D.108. CVT.S.W.fmt pool

Pool path

NM / P32 / POOL32F / POOL32F_3 / POOL32Fxf / POOL32Fxf_1 / CVT.S.W.fmt

Selector width

1 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:14] → selector[0:0]

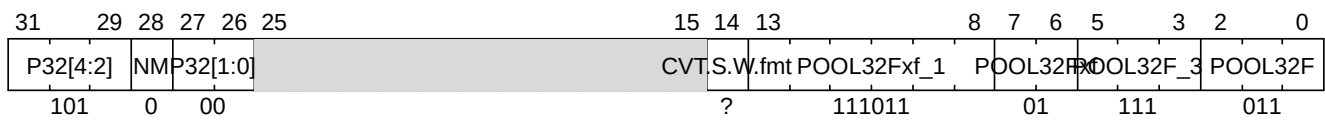


Figure 1914. 32-bit opcode fields

High / low	0	1
?	CVT.S.W	—

D.109. CMP.fmt pool

Pool path

NM / P32 / POOL32F / CMP.fmt

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:3] → selector[2:0]

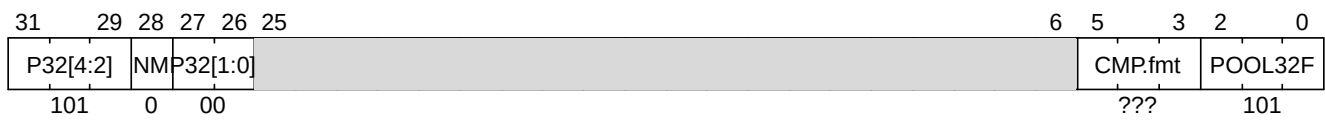


Figure 1915. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	CMP.S	—	CMP.D	—	—	—	—	—

D.110. P.LS.S9 pool

Pool path

NM / P32 / P.LS.S9

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[10:8] → selector[2:0]

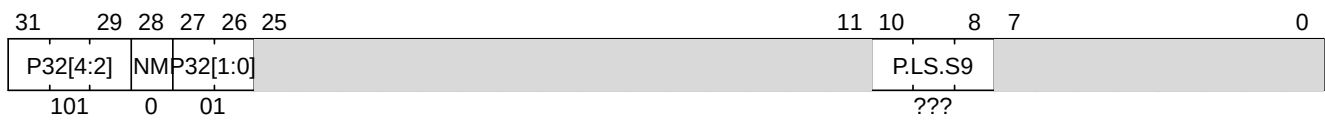


Figure 1916. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	P.LS.S0 pool	—	P.LS.E0 pool	—	—	—	—	—

D.111. P.LS.S0 pool

Pool path

NM / P32 / P.LS.S9 / P.LS.S0

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:11] → selector[3:0]

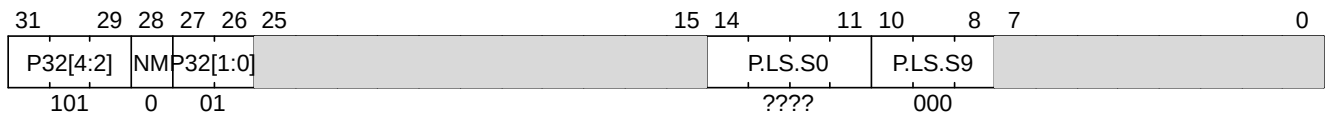


Figure 1917. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	PREF[S9] / PREF	—	—	—	—
1???	—	—	—	—	—	—	—	—

D.112. P.LS.E0 pool

Pool path

NM / P32 / P.LS.S9 / P.LS.E0

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:11] → selector[3:0]

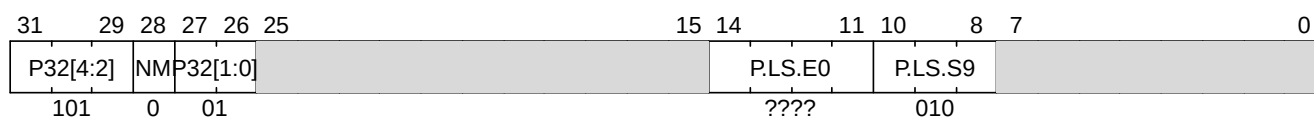


Figure 1918. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	PREFE[32] / PREF	—	—	—	—
1???	—	—	—	—	—	—	—	—

D.113. P32.LS.PAIR pool

Pool path

NM / P32 / P32.LS.PAIR

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:11] → selector[3:0]

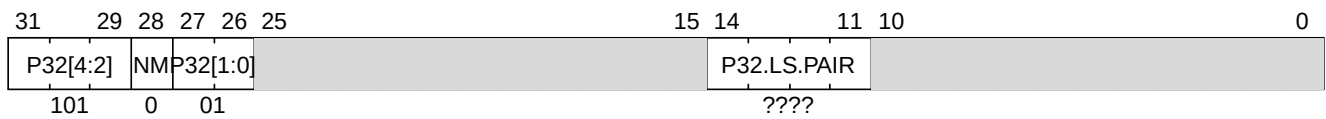


Figure 1919. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	—	—	—	—
1???	—	—	LLWP / LLWPE	SCWP / SCWPE	—	—	LLDP	SCDP

D.114. P32.LS.S9 pool

Pool path

NM / P32 / P32.LS.S9

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:11] → selector[3:0]

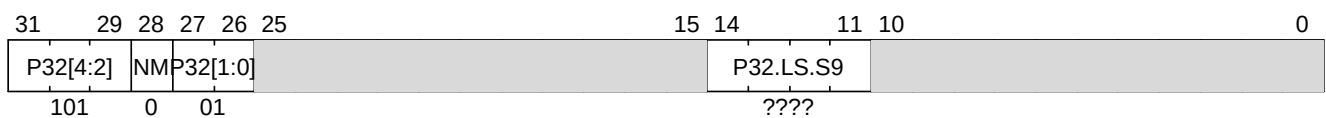


Figure 1920. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	LB[S9] / LB	SB[S9] / SB	LBU[S9] / LBU	—	LH[S9] / LH	SH[S9] / SH	LHU[S9] / LHU	LWU[S9] / LWU
1???	LW[S9] / LW	SW[S9] / SW	LL / LLE	SC / SCE	LD[S9] / LD	SD[S9] / SD	LLD	SCD

D.115. P32A.EVA pool

Pool path

NM / P32 / P32A.EVA

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:11] → selector[3:0]

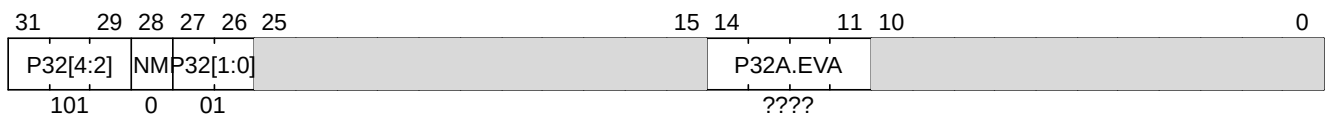


Figure 1921. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	LBE	SBE	LBUE	—	LHE	SHE	LHUE	—
1???	LWE	SWE	—	—	—	—	—	—

D.116. P32A.MEM pool

Pool path

NM / P32 / P32A.MEM

Selector width

4 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[14:11] → selector[3:0]

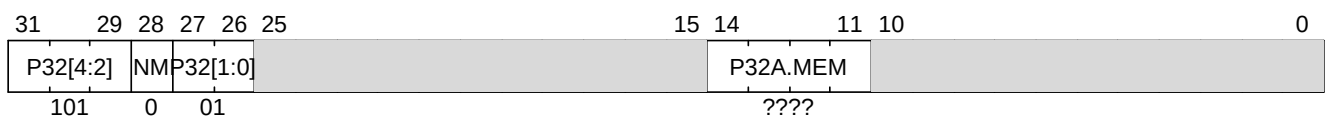


Figure 1922. 32-bit opcode fields

High / low	?000	?001	?010	?011	?100	?101	?110	?111
0???	—	—	—	—	UALH	UASH	—	—
1???	—	—	—	—	—	—	—	—

D.117. P.BR2 pool

Pool path

NM / P32 / P.BR2

Selector width

2 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[15:14] → selector[1:0]

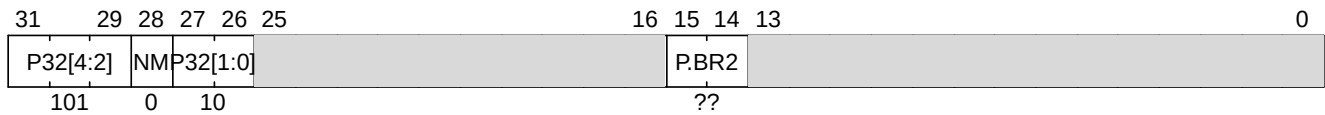


Figure 1923. 32-bit opcode fields

High / low	00	01	10	11
??	BNEC[32] / BNEC	—	BLTC[32] / BLTC	BLTUC

D.118. POOL32S pool

Pool path

NM / P32 / POOL32S

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[2:0] → selector[2:0]

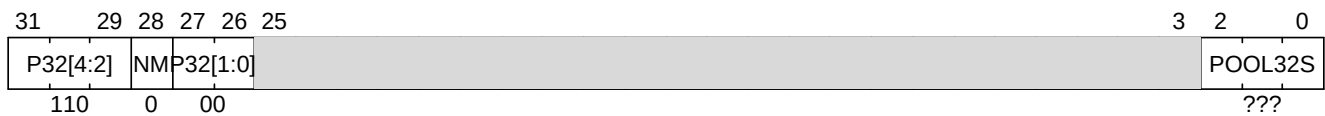


Figure 1924. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	POOL32S_0 pool	—	—	—	POOL32S_4 pool	—	—	—

D.119. POOL32S_0 pool

Pool path

NM / P32 / POOL32S / POOL32S_0

Selector width

6 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[8:3] → selector[5:0]

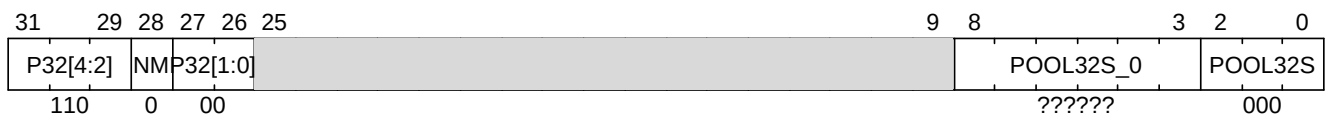


Figure 1925. 32-bit opcode fields

High / low	???000	???001	???010	???011	???100	???101	???110	???111
000???	—	DLSA	DSLLV	DMUL[32] / DMUL	—	—	—	—
001???	—	—	DSRLV	DMUH[32] / DMUH	—	—	—	—
010???	—	—	DSRAV	DMULU[32] / DMULU	—	—	—	—
011???	—	—	DROTRV	DMUHU[32] / DMUHU	—	—	—	—
100???	—	—	DADD	DDIV[32] / DDIV	—	—	—	—
101???	—	—	DADDU	DMOD[32] / DMOD	—	—	—	—
110???	—	—	DSUB	DDIVU[32] / DDIVU	—	—	—	—
111???	—	—	DSUBU	DMODU[32] / DMODU	—	—	—	—

D.120. POOL32S_4 pool

Pool path

NM / P32 / POOL32S / POOL32S_4

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[5:3] → selector[2:0]

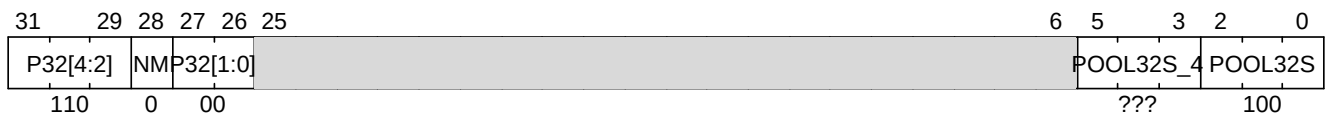


Figure 1926. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	—	—	—	—	—	—	—	DCLO / DCLZ

D.121. P.BRI pool

Pool path

NM / P32 / P.BRI

Selector width

3 bits

Encoding width

32 bits

Selector layout

32-bit: instruction[20:18] → selector[2:0]

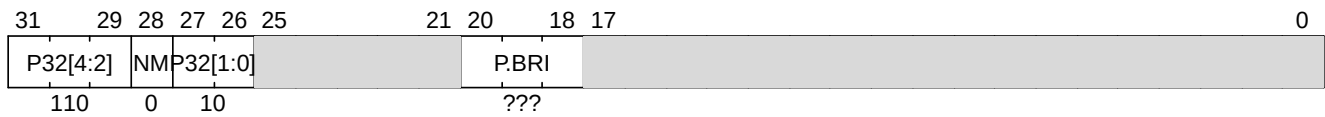


Figure 1927. 32-bit opcode fields

High / low	000	001	010	011	100	101	110	111
???	BEQIC	BBEQZC	BGEIC	BGEIUC	BNEIC	BBNEZC	BLTIC	BLTIUC

D.122. P16 pool

Pool path

NM / P16

Selector width

5 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[15:13] → selector[4:2]; instruction[11:10] → selector[1:0]

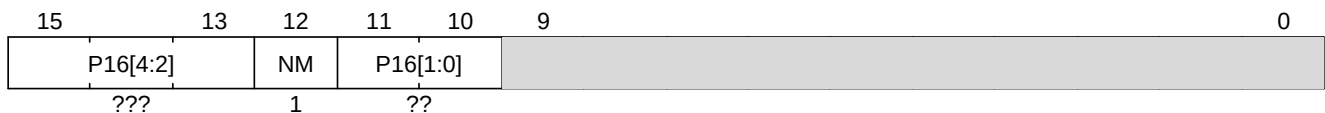


Figure 1928. 16-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	P16.MOVE / P16.MV pool / MOVE	P16.LW / LW	BC[16] / BC	P16.SR pool	4 options	P16.LWSP / LW	BALC[16] / BALC	4 options
01???	LWXS[16] / P16C pool / LWXS	P16.LWGP1 6 / LW	—	6 options	P16.A1 pool	P16.LW4X4 / LW	—	6 options
10???	P16.NOP / P16.A2 pool / NOP	P16.SW / SW	BEQZC.16 / BEQZC	—	P16.ADD U pool	P16.SWSP / SW	BNEZC.16 / BNEZC	P16.MOVEP / MOVEP
11???	P16.LI / LI	P16.SWGP1 6 / SW	7 options	—	ANDI[16] / ANDI	P16.SW4X4 / SW	—	2 options

D.122.1. Selector 00100

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- P16.SLL
- P16.SRL
- SLL
- SRL

D.122.2. Selector 00111

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDU.4X4
- P16.4X4
- ADDU
- MUL

D.122.3. Selector 01011

This selector has 6 possible interpretations across the declared revisions, features, and processor states:

- P16.LB
- P16.LBU
- P16.SB
- LB
- LBU
- SB

D.122.4. Selector 01111

This selector has 6 possible interpretations across the declared revisions, features, and processor states:

- P16.LH
- P16.LHU
- P16.SH
- LH
- LHU
- SH

D.122.5. Selector 11010

This selector has 7 possible interpretations across the declared revisions, features, and processor states:

- BEQC[16]
- BNEC.16
- P16.JRC
- BEQC
- BNEC
- JALRC
- JRC

D.122.6. Selector 11111

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- P16.MOVEP.REV

- **MOVEP**

D.123. P16.MV pool

Pool path

NM / P16 / P16.MV

Selector width

5 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[9:5] → selector[4:0]

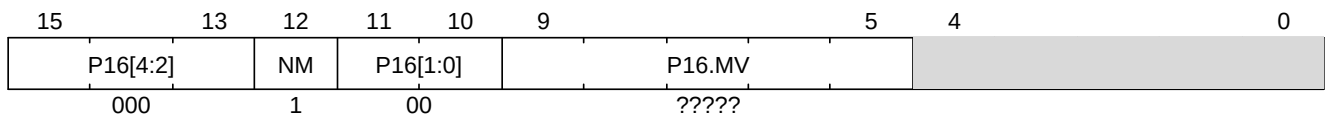


Figure 1929. 16-bit opcode fields

High / low	??000	??001	??010	??011	??100	??101	??110	??111
00???	P16.RI pool	—	—	—	—	—	—	—
01???	—	—	—	—	—	—	—	—
10???	—	—	—	—	—	—	—	—
11???	—	—	—	—	—	—	—	—

D.124. P16.RI pool

Pool path

NM / P16 / P16.MV / P16.RI

Selector width

2 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[4:3] → selector[1:0]

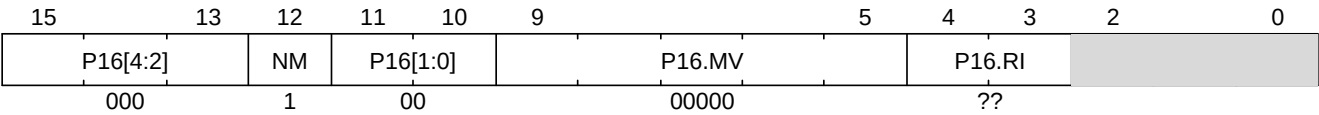


Figure 1930. 16-bit opcode fields

High / low	00	01	10	11
??	—	P16.SYSCALL pool	BREAK[16] / BREAK	SDBBP[16] / SDBBP

D.125. P16.SYSCALL pool

Pool path

NM / P16 / P16.MV / P16.RI / P16.SYSCALL

Selector width

1 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[2:2] → selector[0:0]

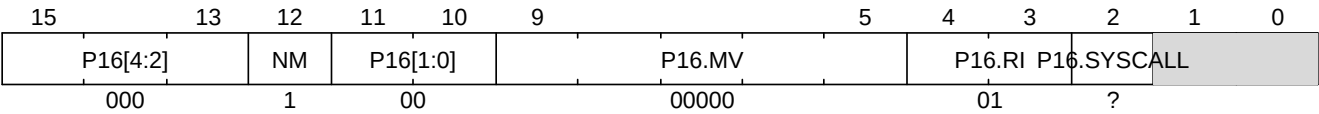


Figure 1931. 16-bit opcode fields

High / low	0	1
?	SYSCALL[16] / SYSCALL	—

D.126. P16.SR pool

Pool path

NM / P16 / P16.SR

Selector width

1 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[8:8] → selector[0:0]

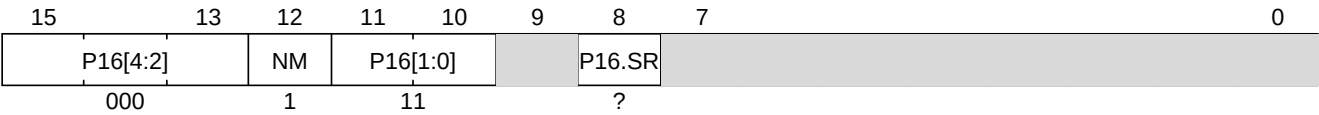


Figure 1932. 16-bit opcode fields

High / low	0	1
?	SAVE[16] / SAVE	2 options

D.126.1. Selector 1

This selector has 2 possible interpretations across the declared revisions, features, and processor states:

- `RESTORE.JRC[16]`
- `RESTORE.JRC`

D.127. P16C pool

Pool path

`NM / P16 / P16C`

Selector width

1 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[0:0] → selector[0:0]

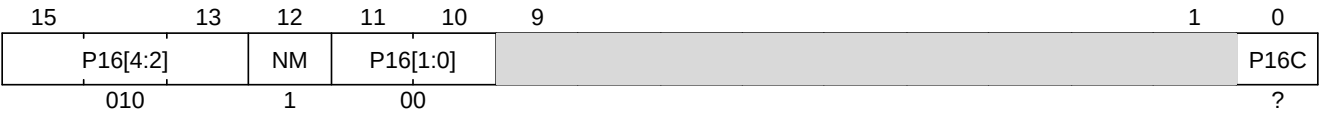


Figure 1933. 16-bit opcode fields

High / low	0	1
?	POOL16C_0 pool	—

D.128. POOL16C_0 pool

Pool path

`NM / P16 / P16C / POOL16C_0`

Selector width

1 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[1:1] → selector[0:0]

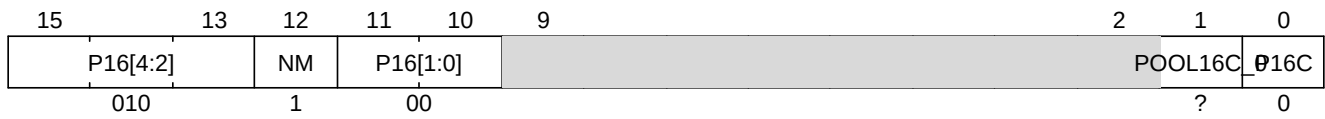


Figure 1934. 16-bit opcode fields

High / low	0	1
?	POOL16C_00 pool	—

D.129. POOL16C_00 pool

Pool path

NM / P16 / P16C / POOL16C_0 / POOL16C_00

Selector width

2 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[3:2] → selector[1:0]

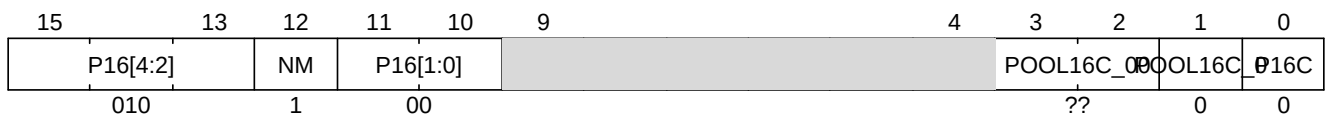


Figure 1935. 16-bit opcode fields

High / low	00	01	10	11
??	NOT[16] / NOT	XOR[16] / XOR	AND[16] / AND	OR[16] / OR

D.130. P16.A1 pool

Pool path

NM / P16 / P16.A1

Selector width

1 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[6:6] → selector[0:0]

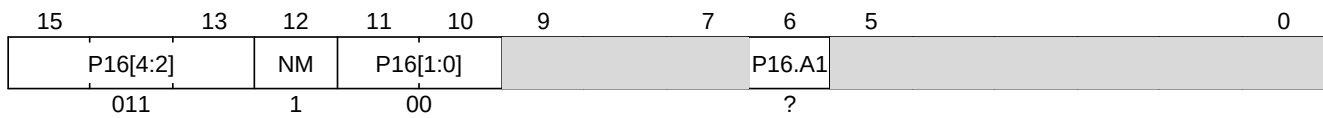


Figure 1936. 16-bit opcode fields

High / low	0	1
?	—	4 options

D.130.1. Selector 1

This selector has 4 possible interpretations across the declared revisions, features, and processor states:

- ADDIU[R1.SP]
- DADDIU[R1.SP]
- ADDIU
- DADDIU

D.131. P16.A2 pool

Pool path

NM / P16 / P16.A2

Selector width

1 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[3:3] → selector[0:0]



Figure 1937. 16-bit opcode fields

High / low	0	1
?	ADDIU[R2] / ADDIU	ADDIU[RS5] / ADDIU

D.132. P16.ADDU pool

Pool path

NM / P16 / P16.ADDU

Selector width

1 bits

Encoding width

16 bits

Selector layout

16-bit: instruction[0:0] → selector[0:0]

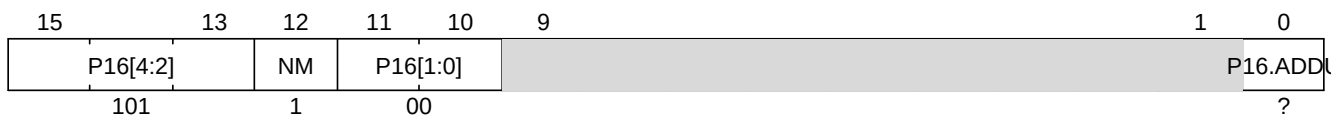


Figure 1938. 16-bit opcode fields

High / low	0	1
?	ADDU[16] / ADDU	SUBU[16] / SUBU