

DRAFT

Qualcomm uC extensions

Version 0.1,

Table of Contents

Licensing and Acknowledgements	1
Copyright and license information	2
Acknowledgements	3
Versions	4
1. Extension description	5
2. CSR summary	6
3. Csrs (in alphabetical order)	7
3.1. mclcie0	8
3.1.1. Attributes	8
3.1.2. Format	8
3.1.3. Field Summary	8
3.1.4. Fields	9
IRQ0	9
IRQ1	10
IRQ2	10
IRQ3	10
IRQ4	11
IRQ5	11
IRQ6	11
IRQ7	12
IRQ8	12
IRQ9	12
IRQ10	13
IRQ11	13
IRQ12	13
IRQ13	14
IRQ14	14
IRQ15	14
IRQ16	15
IRQ17	15
IRQ18	15
IRQ19	16
IRQ20	16
IRQ21	16
IRQ22	17
IRQ23	17
IRQ24	17
IRQ25	18

IRQ26	18
IRQ27	18
IRQ28	19
IRQ29	19
IRQ30	19
IRQ31	20
3.2. mcliciel	21
3.2.1. Attributes	21
3.2.2. Format	21
3.2.3. Field Summary	21
3.2.4. Fields	22
IRQ32	22
IRQ33	23
IRQ34	23
IRQ35	23
IRQ36	24
IRQ37	24
IRQ38	24
IRQ39	25
IRQ40	25
IRQ41	25
IRQ42	26
IRQ43	26
IRQ44	26
IRQ45	27
IRQ46	27
IRQ47	27
IRQ48	28
IRQ49	28
IRQ50	28
IRQ51	29
IRQ52	29
IRQ53	29
IRQ54	30
IRQ55	30
IRQ56	30
IRQ57	31
IRQ58	31
IRQ59	31

IRQ60	32
IRQ61	32
IRQ62	32
IRQ63	33
3.3. mclicie2	34
3.3.1. Attributes	34
3.3.2. Format	34
3.3.3. Field Summary	34
3.3.4. Fields	35
IRQ64	35
IRQ65	36
IRQ66	36
IRQ67	36
IRQ68	37
IRQ69	37
IRQ70	37
IRQ71	38
IRQ72	38
IRQ73	38
IRQ74	39
IRQ75	39
IRQ76	39
IRQ77	40
IRQ78	40
IRQ79	40
IRQ80	41
IRQ81	41
IRQ82	41
IRQ83	42
IRQ84	42
IRQ85	42
IRQ86	43
IRQ87	43
IRQ88	43
IRQ89	44
IRQ90	44
IRQ91	44
IRQ92	45
IRQ93	45

IRQ94	45
IRQ95	46
3.4. mclicie3	47
3.4.1. Attributes	47
3.4.2. Format	47
3.4.3. Field Summary	47
3.4.4. Fields	48
IRQ96	48
IRQ97	49
IRQ98	49
IRQ99	49
IRQ100	50
IRQ101	50
IRQ102	50
IRQ103	51
IRQ104	51
IRQ105	51
IRQ106	52
IRQ107	52
IRQ108	52
IRQ109	53
IRQ110	53
IRQ111	53
IRQ112	54
IRQ113	54
IRQ114	54
IRQ115	55
IRQ116	55
IRQ117	55
IRQ118	56
IRQ119	56
IRQ120	56
IRQ121	57
IRQ122	57
IRQ123	57
IRQ124	58
IRQ125	58
IRQ126	58
IRQ127	59

3.5. mclcie4.....	60
3.5.1. Attributes	60
3.5.2. Format.....	60
3.5.3. Field Summary.....	60
3.5.4. Fields.....	61
IRQ128	61
IRQ129.....	62
IRQ130.....	62
IRQ131.....	62
IRQ132.....	63
IRQ133.....	63
IRQ134.....	63
IRQ135.....	64
IRQ136.....	64
IRQ137.....	64
IRQ138.....	65
IRQ139.....	65
IRQ140.....	65
IRQ141.....	66
IRQ142.....	66
IRQ143.....	66
IRQ144.....	67
IRQ145.....	67
IRQ146.....	67
IRQ147.....	68
IRQ148.....	68
IRQ149.....	68
IRQ150.....	69
IRQ151.....	69
IRQ152.....	69
IRQ153.....	70
IRQ154.....	70
IRQ155.....	70
IRQ156	71
IRQ157	71
IRQ158	71
IRQ159.....	72
3.6. mclcie5.....	73
3.6.1. Attributes.....	73

3.6.2. Format	73
3.6.3. Field Summary	73
3.6.4. Fields	74
IRQ160	74
IRQ161	75
IRQ162	75
IRQ163	75
IRQ164	76
IRQ165	76
IRQ166	76
IRQ167	77
IRQ168	77
IRQ169	77
IRQ170	78
IRQ171	78
IRQ172	78
IRQ173	79
IRQ174	79
IRQ175	79
IRQ176	80
IRQ177	80
IRQ178	80
IRQ179	81
IRQ180	81
IRQ181	81
IRQ182	82
IRQ183	82
IRQ184	82
IRQ185	83
IRQ186	83
IRQ187	83
IRQ188	84
IRQ189	84
IRQ190	84
IRQ191	85
3.7. mclicie6	86
3.7.1. Attributes	86
3.7.2. Format	86
3.7.3. Field Summary	86

3.7.4. Fields	87
IRQ192	87
IRQ193	88
IRQ194	88
IRQ195	88
IRQ196	89
IRQ197	89
IRQ198	89
IRQ199	90
IRQ200	90
IRQ201	90
IRQ202	91
IRQ203	91
IRQ204	91
IRQ205	92
IRQ206	92
IRQ207	92
IRQ208	93
IRQ209	93
IRQ210	93
IRQ211	94
IRQ212	94
IRQ213	94
IRQ214	95
IRQ215	95
IRQ216	95
IRQ217	96
IRQ218	96
IRQ219	96
IRQ220	97
IRQ221	97
IRQ222	97
IRQ223	98
3.8. mclicie7	99
3.8.1. Attributes	99
3.8.2. Format	99
3.8.3. Field Summary	99
3.8.4. Fields	100
IRQ224	100

IRQ225	101
IRQ226	101
IRQ227	101
IRQ228	102
IRQ229	102
IRQ230	102
IRQ231	103
IRQ232	103
IRQ233	103
IRQ234	104
IRQ235	104
IRQ236	104
IRQ237	105
IRQ238	105
IRQ239	105
IRQ240	106
IRQ241	106
IRQ242	106
IRQ243	107
IRQ244	107
IRQ245	107
IRQ246	108
IRQ247	108
IRQ248	108
IRQ249	109
IRQ250	109
IRQ251	109
IRQ252	110
IRQ253	110
IRQ254	110
IRQ255	111
3.9. mclicip0	112
3.9.1. Attributes	112
3.9.2. Format	112
3.9.3. Field Summary	112
3.9.4. Fields	113
IRQ0	113
IRQ1	114
IRQ2	114

IRQ3	114
IRQ4	115
IRQ5	115
IRQ6	115
IRQ7	116
IRQ8	116
IRQ9	116
IRQ10	117
IRQ11	117
IRQ12	117
IRQ13	118
IRQ14	118
IRQ15	118
IRQ16	119
IRQ17	119
IRQ18	119
IRQ19	120
IRQ20	120
IRQ21	120
IRQ22	121
IRQ23	121
IRQ24	121
IRQ25	122
IRQ26	122
IRQ27	122
IRQ28	123
IRQ29	123
IRQ30	123
IRQ31	124
3.10. mclicip1	125
3.10.1. Attributes	125
3.10.2. Format	125
3.10.3. Field Summary	125
3.10.4. Fields	126
IRQ32	126
IRQ33	127
IRQ34	127
IRQ35	127
IRQ36	128

IRQ37	128
IRQ38	128
IRQ39	129
IRQ40	129
IRQ41	129
IRQ42	130
IRQ43	130
IRQ44	130
IRQ45	131
IRQ46	131
IRQ47	131
IRQ48	132
IRQ49	132
IRQ50	132
IRQ51	133
IRQ52	133
IRQ53	133
IRQ54	134
IRQ55	134
IRQ56	134
IRQ57	135
IRQ58	135
IRQ59	135
IRQ60	136
IRQ61	136
IRQ62	136
IRQ63	137
3.11. mclicip2	138
3.11.1. Attributes	138
3.11.2. Format	138
3.11.3. Field Summary	138
3.11.4. Fields	139
IRQ64	139
IRQ65	140
IRQ66	140
IRQ67	140
IRQ68	141
IRQ69	141
IRQ70	141

IRQ71	142
IRQ72	142
IRQ73	142
IRQ74	143
IRQ75	143
IRQ76	143
IRQ77	144
IRQ78	144
IRQ79	144
IRQ80	145
IRQ81	145
IRQ82	145
IRQ83	146
IRQ84	146
IRQ85	146
IRQ86	147
IRQ87	147
IRQ88	147
IRQ89	148
IRQ90	148
IRQ91	148
IRQ92	149
IRQ93	149
IRQ94	149
IRQ95	150
3.12. mclicip3	151
3.12.1. Attributes	151
3.12.2. Format	151
3.12.3. Field Summary	151
3.12.4. Fields	152
IRQ96	152
IRQ97	153
IRQ98	153
IRQ99	153
IRQ100	154
IRQ101	154
IRQ102	154
IRQ103	155
IRQ104	155

IRQ105	155
IRQ106	156
IRQ107	156
IRQ108	156
IRQ109	157
IRQ110	157
IRQ111	157
IRQ112	158
IRQ113	158
IRQ114	158
IRQ115	159
IRQ116	159
IRQ117	159
IRQ118	160
IRQ119	160
IRQ120	160
IRQ121	161
IRQ122	161
IRQ123	161
IRQ124	162
IRQ125	162
IRQ126	162
IRQ127	163
3.13. mclicip4	164
3.13.1. Attributes	164
3.13.2. Format	164
3.13.3. Field Summary	164
3.13.4. Fields	165
IRQ128	165
IRQ129	166
IRQ130	166
IRQ131	166
IRQ132	167
IRQ133	167
IRQ134	167
IRQ135	168
IRQ136	168
IRQ137	168
IRQ138	169

IRQ139	169
IRQ140	169
IRQ141	170
IRQ142	170
IRQ143	170
IRQ144	171
IRQ145	171
IRQ146	171
IRQ147	172
IRQ148	172
IRQ149	172
IRQ150	173
IRQ151	173
IRQ152	173
IRQ153	174
IRQ154	174
IRQ155	174
IRQ156	175
IRQ157	175
IRQ158	175
IRQ159	176
3.14. mclicip5	177
3.14.1. Attributes	177
3.14.2. Format	177
3.14.3. Field Summary	177
3.14.4. Fields	178
IRQ160	178
IRQ161	179
IRQ162	179
IRQ163	179
IRQ164	180
IRQ165	180
IRQ166	180
IRQ167	181
IRQ168	181
IRQ169	181
IRQ170	182
IRQ171	182
IRQ172	182

IRQ173	183
IRQ174	183
IRQ175	183
IRQ176	184
IRQ177	184
IRQ178	184
IRQ179	185
IRQ180	185
IRQ181	185
IRQ182	186
IRQ183	186
IRQ184	186
IRQ185	187
IRQ186	187
IRQ187	187
IRQ188	188
IRQ189	188
IRQ190	188
IRQ191	189
3.15. mclicip6	190
3.15.1. Attributes	190
3.15.2. Format	190
3.15.3. Field Summary	190
3.15.4. Fields	191
IRQ192	191
IRQ193	192
IRQ194	192
IRQ195	192
IRQ196	193
IRQ197	193
IRQ198	193
IRQ199	194
IRQ200	194
IRQ201	194
IRQ202	195
IRQ203	195
IRQ204	195
IRQ205	196
IRQ206	196

IRQ207	196
IRQ208	197
IRQ209	197
IRQ210	197
IRQ211	198
IRQ212	198
IRQ213	198
IRQ214	199
IRQ215	199
IRQ216	199
IRQ217	200
IRQ218	200
IRQ219	200
IRQ220	201
IRQ221	201
IRQ222	201
IRQ223	202
3.16. mclicip7	203
3.16.1. Attributes	203
3.16.2. Format	203
3.16.3. Field Summary	203
3.16.4. Fields	204
IRQ224	204
IRQ225	205
IRQ226	205
IRQ227	205
IRQ228	206
IRQ229	206
IRQ230	206
IRQ231	207
IRQ232	207
IRQ233	207
IRQ234	208
IRQ235	208
IRQ236	208
IRQ237	209
IRQ238	209
IRQ239	209
IRQ240	210

IRQ241	210
IRQ242	210
IRQ243	211
IRQ244	211
IRQ245	211
IRQ246	212
IRQ247	212
IRQ248	212
IRQ249	213
IRQ250	213
IRQ251	213
IRQ252	214
IRQ253	214
IRQ254	214
IRQ255	215
4. Instruction summary	216
5. Instructions (in alphabetical order)	223
5.1. qc16.bexti	223
5.2. qc16.bseti	224
5.3. qc16.clrint	225
5.4. qc16.delay	226
5.5. qc16.di	227
5.6. qc16.dir	228
5.7. qc16.ei	229
5.8. qc16.eir	230
5.9. qc16.extu	231
5.10. qc16.mienter.nest	232
5.11. qc16.mienter	234
5.12. qc16.mileaveret	236
5.13. qc16.mnret	238
5.14. qc16.mpyaddi	239
5.15. qc16.mret	240
5.16. qc16.mveqz	241
5.17. qc16.setint	242
5.18. qc32.addsat	243
5.19. qc32.addusat	244
5.20. qc32.beqi	245
5.21. qc32.bgei	246
5.22. qc32.bgeui	247
5.23. qc32.bltn	248

5.24. qc32.bltni	249
5.25. qc32.bnei	250
5.26. qc32.brev32	251
5.27. qc32.clo	252
5.28. qc32.clrint	253
5.29. qc32.clrinti	254
5.30. qc32.compress2	255
5.31. qc32.compress3	256
5.32. qc32.csrrwr	257
5.33. qc32.csrrwri	258
5.34. qc32.cto	259
5.35. qc32.delay	260
5.36. qc32.expand2	261
5.37. qc32.expand3	262
5.38. qc32.extds	263
5.39. qc32.extdspr	264
5.40. qc32.extdsprh	265
5.41. qc32.extdsr	266
5.42. qc32.extdu	267
5.43. qc32.extdupr	268
5.44. qc32.extduprh	269
5.45. qc32.extdur	270
5.46. qc32.exts	271
5.47. qc32.extu	272
5.48. qc32.insb	273
5.49. qc32.insbh	274
5.50. qc32.insbhr	275
5.51. qc32.insbi	276
5.52. qc32.insbpr	277
5.53. qc32.insbprh	278
5.54. qc32.insbr	279
5.55. qc32.insbri	280
5.56. qc32.li	281
5.57. qc32.lieq	282
5.58. qc32.lieqi	283
5.59. qc32.lige	284
5.60. qc32.ligei	285
5.61. qc32.ligeu	286
5.62. qc32.ligeui	287
5.63. qc32.lilt	288
5.64. qc32.lilti	289

5.65. qc32.liltu	290
5.66. qc32.liltui	291
5.67. qc32.line	292
5.68. qc32.linei	293
5.69. qc32.lrb	294
5.70. qc32.lrbu	295
5.71. qc32.lrh	296
5.72. qc32.lrhu	297
5.73. qc32.lrw	298
5.74. qc32.lwm	299
5.75. qc32.lwmi	300
5.76. qc32.mienter.nest	301
5.77. qc32.mienter	303
5.78. qc32.mileaveret	305
5.79. qc32.mnret	307
5.80. qc32.mpyaddi	308
5.81. qc32.mveq	309
5.82. qc32.mveqi	310
5.83. qc32.mvge	311
5.84. qc32.mvgei	312
5.85. qc32.mvgeu	313
5.86. qc32.mvgeui	314
5.87. qc32.mvlt	315
5.88. qc32.mvlti	316
5.89. qc32.mvltu	317
5.90. qc32.mvltui	318
5.91. qc32.mvne	319
5.92. qc32.mvnei	320
5.93. qc32.norm	321
5.94. qc32.normeu	322
5.95. qc32.normu	323
5.96. qc32.selecteqi	324
5.97. qc32.selectieq	325
5.98. qc32.selectieqi	326
5.99. qc32.selectiieq	327
5.100. qc32.selectiine	328
5.101. qc32.selectine	329
5.102. qc32.selectinei	330
5.103. qc32.selectnei	331
5.104. qc32.setint	332
5.105. qc32.setinti	333

5.106. qc32.setwm	334
5.107. qc32.setwmi	335
5.108. qc32.sh4add	336
5.109. qc32.sh5add	337
5.110. qc32.sh6add	338
5.111. qc32.sh7add	339
5.112. qc32.sh8add	340
5.113. qc32.slasat	341
5.114. qc32.sllsat	342
5.115. qc32.srb	343
5.116. qc32.srh	344
5.117. qc32.srw	345
5.118. qc32.subsat	346
5.119. qc32.subusat	347
5.120. qc32.swm	348
5.121. qc32.swmi	349
5.122. qc32.wrap	350
5.123. qc32.wrapi	351
5.124. qc48.addai	352
5.125. qc48.addi	353
5.126. qc48.andai	354
5.127. qc48.andi	355
5.128. qc48.beqi	356
5.129. qc48.bgei	357
5.130. qc48.bgeui	358
5.131. qc48.blti	359
5.132. qc48.bltoi	360
5.133. qc48.bnei	361
5.134. qc48.j	362
5.135. qc48.jal	363
5.136. qc48.lb	364
5.137. qc48.lbu	365
5.138. qc48.lh	366
5.139. qc48.lhu	367
5.140. qc48.li	368
5.141. qc48.lw	369
5.142. qc48.orai	370
5.143. qc48.ori	371
5.144. qc48.sb	372
5.145. qc48.sh	373
5.146. qc48.sw	374

5.147. qc48.xorai	375
5.148. qc48.xori	376
6. IDL Functions	377
6.1. abort_current_instruction (builtin)	377
6.2. access_check	377
6.3. assert (builtin)	377
6.4. atomic_check_then_write_32 (builtin)	377
6.5. atomic_check_then_write_64 (builtin)	378
6.6. current_translation_mode	378
6.7. delay (builtin)	379
6.8. effective_ldst_mode	379
6.9. exception_handling_mode	379
6.10. highest_set_bit	380
6.11. ialign	380
6.12. implemented? (builtin)	381
6.13. is_naturally_aligned	381
6.14. jump_halfword	381
6.15. lowest_set_bit	381
6.16. mode	382
6.17. mpv	382
6.18. mtval_for	382
6.19. mtval_readonly?	383
6.20. notify_mode_change (builtin)	383
6.21. page_walk	384
6.22. pma_applies? (builtin)	386
6.23. pmp_check	386
6.24. pmp_match	387
6.25. pmp_match_32	387
6.26. pmp_match_64	388
6.27. raise	389
6.28. read_memory	390
6.29. read_memory_aligned	390
6.30. read_physical_memory (builtin)	391
6.31. set_mode	391
6.32. sext	391
6.33. stval_for	392
6.34. stval_readonly?	392
6.35. translate	393
6.36. unpredictable (builtin)	393
6.37. virtual_mode?	394
6.38. vstval_for	394

6.39. vstval_readonly?	395
6.40. write_memory	395
6.41. write_memory_aligned	395
6.42. write_physical_memory (builtin)	396
6.43. xlen	396

DRAFT

Licensing and Acknowledgements



This document is in the [Development state](#).

Change should be expected

DRAFT

Copyright and license information

This document is released under the [Creative Commons Attribution 4.0 International License](#).

Copyright 2024 by Qualcomm Technologies, Inc..

DRAFT

Acknowledgements

Contributors to version 0.1 of the specification (in alphabetical order) include:

- Derek Hower <dhower@qti.qualcomm.com> (Qualcomm Technologies, Inc.)
- Albert Yosher <ayosher@qti.qualcomm.com> (Qualcomm Technologies, Inc.)

We express our gratitude to everyone that contributed to, reviewed or improved this specification through their comments and questions.

DRAFT

Versions

The following versions have been defined:

Version

0.1

State

development

DRAFT

Chapter 1. Extension description

The Xqciu extension includes a set of instructions that improve RISC-V code density and performance in microcontrollers. It fills several gaps:

Long immediates

This extension provides wide-immediate versions of loads, stores, branches, jumps, and several arithmetic operations.

Addressing modes

New indexed addressing modes are added for load and store instructions.

Load/store multiple

Xqciu adds instructions that load multiple sequential values from memory into multiple registers. Analogous instructions for stores are also included.

Branch immediate instructions

The base RISC-V ISA provides conditional branches that compare two register values. Xqciu adds conditional branch instructions that compare an immediate value to a register value, reducing a common code sequence into a single instruction.

Conditional instructions

Xqciu includes a variety of conditional select instructions that pick one of two operand values based on the result of a condition and conditional move instructions that either move a value or retain a value in a destination register based on the result of a condition. Conditional select and conditional move instructions help reduce code size and avoid branches in common code sequences.

Extra bit manipulation instructions

Xqciu expands upon the standard B extension by adding instructions for bit insertion/extraction, bit set/clear, sign and zero extension, and bit counting instructions.

Fast interrupt instructions

Xqciu adds instructions to accelerate interrupt handling, including instructions to quickly enable/disable interrupts and instructions to automatically save an interrupt frame.

Saturating arithmetic

Xqciu adds saturating arithmetic instructions to improve performance of fixed-point calculations.

Xqciu adds 16, 32, and 48-bit instruction encodings. The 16 and 32-bit encodings are allocated in the custom opcode space so as not to conflict with standard RISC-V instructions. The 48-bit instructions follow the guidance for 48-bit instructions in the RISC-V standard, but are not allocated in reserved custom space since no such space has been defined by RISC-V International.

Chapter 2. CSR summary

The following 16 are added by this extension.

RV32	RV64	CSR	Name	v0.1
✓	✓	mclicie0	IRQ Enable 0	✓
✓	✓	mclicie1	IRQ Enable 1	✓
✓	✓	mclicie2	IRQ Enable 2	✓
✓	✓	mclicie3	IRQ Enable 3	✓
✓	✓	mclicie4	IRQ Enable 4	✓
✓	✓	mclicie5	IRQ Enable 5	✓
✓	✓	mclicie6	IRQ Enable 6	✓
✓	✓	mclicie7	IRQ Enable 7	✓
✓	✓	mclicip0	IRQ Pending 0	✓
✓	✓	mclicip1	IRQ Pending 1	✓
✓	✓	mclicip2	IRQ Pending 2	✓
✓	✓	mclicip3	IRQ Pending 3	✓
✓	✓	mclicip4	IRQ Pending 4	✓
✓	✓	mclicip5	IRQ Pending 5	✓
✓	✓	mclicip6	IRQ Pending 6	✓
✓	✓	mclicip7	IRQ Pending 7	✓

Chapter 3. Csrs (in alphabetical order)

DRAFT

3.1. mclcie0

IRQ Enable 0

Enable bits for IRQs 0-31

3.1.1. Attributes

CSR Address	0x7f0
Length	32-bit-bit
Privilege Mode	M

3.1.2. Format

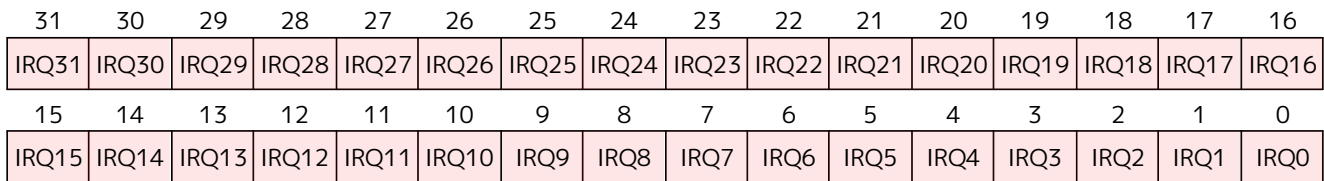


Figure 1. mclcie0 format

3.1.3. Field Summary

Name	Location	Type	Reset Value
IRQ0	0	RW	0
IRQ1	1	RW	0
IRQ2	2	RW	0
IRQ3	3	RW	0
IRQ4	4	RW	0
IRQ5	5	RW	0
IRQ6	6	RW	0
IRQ7	7	RW	0
IRQ8	8	RW	0
IRQ9	9	RW	0
IRQ10	10	RW	0
IRQ11	11	RW	0
IRQ12	12	RW	0
IRQ13	13	RW	0

Name	Location	Type	Reset Value
IRQ14	14	RW	0
IRQ15	15	RW	0
IRQ16	16	RW	0
IRQ17	17	RW	0
IRQ18	18	RW	0
IRQ19	19	RW	0
IRQ20	20	RW	0
IRQ21	21	RW	0
IRQ22	22	RW	0
IRQ23	23	RW	0
IRQ24	24	RW	0
IRQ25	25	RW	0
IRQ26	26	RW	0
IRQ27	27	RW	0
IRQ28	28	RW	0
IRQ29	29	RW	0
IRQ30	30	RW	0
IRQ31	31	RW	0

3.1.4. Fields

IRQ0

Location 0
Description IRQ1 enabled
Type RW
Reset value 0

IRQ1

Location*1***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ2

Location*2***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ3

Location*3***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ4

Location

4

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ5

Location

5

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ6

Location

6

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ7

Location

7

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ8

Location

8

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ9

Location

9

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ10

Location*10***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ11

Location*11***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ12

Location*12***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ13

Location*13***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ14

Location*14***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ15

Location*15***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ16

Location*16***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ17

Location*17***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ18

Location*18***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ19

Location*19***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ20

Location*20***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ21

Location*21***Description***IRQ1 enabled***Type***RW***Reset value***0*

IRQ22

Location

22

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ23

Location

23

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ24

Location

24

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ25

Location

25

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ26

Location

26

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ27

Location

27

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ28

Location

28

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ29

Location

29

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ30

Location

30

Description*IRQ1 enabled***Type***RW***Reset value**

0

IRQ31

Location*31***Description***IRQ1 enabled***Type***RW***Reset value***0*

DRAFT

3.2. mclcie1

IRQ Enable 1

Enable bits for IRQs 32-63

3.2.1. Attributes

CSR Address	0x7f1
Length	32-bit-bit
Privilege Mode	M

3.2.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ63	IRQ62	IRQ61	IRQ60	IRQ59	IRQ58	IRQ57	IRQ56	IRQ55	IRQ54	IRQ53	IRQ52	IRQ51	IRQ50	IRQ49	IRQ48
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ47	IRQ46	IRQ45	IRQ44	IRQ43	IRQ42	IRQ41	IRQ40	IRQ39	IRQ38	IRQ37	IRQ36	IRQ35	IRQ34	IRQ33	IRQ32

Figure 2. mclcie1 format

3.2.3. Field Summary

Name	Location	Type	Reset Value
IRQ32	0	RW	0
IRQ33	1	RW	0
IRQ34	2	RW	0
IRQ35	3	RW	0
IRQ36	4	RW	0
IRQ37	5	RW	0
IRQ38	6	RW	0
IRQ39	7	RW	0
IRQ40	8	RW	0
IRQ41	9	RW	0
IRQ42	10	RW	0
IRQ43	11	RW	0
IRQ44	12	RW	0
IRQ45	13	RW	0

Name	Location	Type	Reset Value
IRQ46	14	RW	0
IRQ47	15	RW	0
IRQ48	16	RW	0
IRQ49	17	RW	0
IRQ50	18	RW	0
IRQ51	19	RW	0
IRQ52	20	RW	0
IRQ53	21	RW	0
IRQ54	22	RW	0
IRQ55	23	RW	0
IRQ56	24	RW	0
IRQ57	25	RW	0
IRQ58	26	RW	0
IRQ59	27	RW	0
IRQ60	28	RW	0
IRQ61	29	RW	0
IRQ62	30	RW	0
IRQ63	31	RW	0

3.2.4. Fields

IRQ32

Location 0
Description IRQ33 enabled
Type RW
Reset value 0

IRQ33

Location	1
Description	IRQ33 enabled
Type	RW
Reset value	0

IRQ34

Location	2
Description	IRQ33 enabled
Type	RW
Reset value	0

IRQ35

Location	3
Description	IRQ33 enabled
Type	RW
Reset value	0

IRQ36

Location

4

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ37

Location

5

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ38

Location

6

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ39

Location	7
Description	<i>IRQ33 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ40

Location	8
Description	<i>IRQ33 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ41

Location	9
Description	<i>IRQ33 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ42

Location*10***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ43

Location*11***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ44

Location*12***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ45

Location*13***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ46

Location*14***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ47

Location*15***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ48

Location*16***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ49

Location*17***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ50

Location*18***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ51

Location*19***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ52

Location*20***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ53

Location*21***Description***IRQ33 enabled***Type***RW***Reset value***0*

IRQ54

Location

22

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ55

Location

23

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ56

Location

24

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ57

Location

25

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ58

Location

26

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ59

Location

27

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ60

Location

28

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ61

Location

29

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ62

Location

30

Description*IRQ33 enabled***Type***RW***Reset value**

0

IRQ63

Location	31
Description	IRQ33 enabled
Type	RW
Reset value	0

DRAFT

3.3. mclcie2

IRQ Enable 2

Enable bits for IRQs 64-95

3.3.1. Attributes

CSR Address	0x7f2
Length	32-bit-bit
Privilege Mode	M

3.3.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ95	IRQ94	IRQ93	IRQ92	IRQ91	IRQ90	IRQ89	IRQ88	IRQ87	IRQ86	IRQ85	IRQ84	IRQ83	IRQ82	IRQ81	IRQ80
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ79	IRQ78	IRQ77	IRQ76	IRQ75	IRQ74	IRQ73	IRQ72	IRQ71	IRQ70	IRQ69	IRQ68	IRQ67	IRQ66	IRQ65	IRQ64

Figure 3. mclcie2 format

3.3.3. Field Summary

Name	Location	Type	Reset Value
IRQ64	0	RW	0
IRQ65	1	RW	0
IRQ66	2	RW	0
IRQ67	3	RW	0
IRQ68	4	RW	0
IRQ69	5	RW	0
IRQ70	6	RW	0
IRQ71	7	RW	0
IRQ72	8	RW	0
IRQ73	9	RW	0
IRQ74	10	RW	0
IRQ75	11	RW	0
IRQ76	12	RW	0
IRQ77	13	RW	0

Name	Location	Type	Reset Value
IRQ78	14	RW	0
IRQ79	15	RW	0
IRQ80	16	RW	0
IRQ81	17	RW	0
IRQ82	18	RW	0
IRQ83	19	RW	0
IRQ84	20	RW	0
IRQ85	21	RW	0
IRQ86	22	RW	0
IRQ87	23	RW	0
IRQ88	24	RW	0
IRQ89	25	RW	0
IRQ90	26	RW	0
IRQ91	27	RW	0
IRQ92	28	RW	0
IRQ93	29	RW	0
IRQ94	30	RW	0
IRQ95	31	RW	0

3.3.4. Fields

IRQ64

Location

0

Description

IRQ65 enabled

Type

RW

Reset value

0

IRQ65

Location

1

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ66

Location

2

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ67

Location

3

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ68

Location

4

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ69

Location

5

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ70

Location

6

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ71

Location

7

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ72

Location

8

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ73

Location

9

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ74

Location*10***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ75

Location*11***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ76

Location*12***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ77

Location*13***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ78

Location*14***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ79

Location*15***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ80

Location	16
Description	<i>IRQ65 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ81

Location	17
Description	<i>IRQ65 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ82

Location	18
Description	<i>IRQ65 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ83

Location*19***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ84

Location*20***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ85

Location*21***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ86

Location

22

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ87

Location

23

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ88

Location

24

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ89

Location

25

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ90

Location

26

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ91

Location

27

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ92

Location

28

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ93

Location

29

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ94

Location

30

Description*IRQ65 enabled***Type***RW***Reset value**

0

IRQ95

Location*31***Description***IRQ65 enabled***Type***RW***Reset value***0*

DRAFT

3.4. mclcie3

IRQ Enable 3

Enable bits for IRQs 96-127

3.4.1. Attributes

CSR Address	0x7f3
Length	32-bit-bit
Privilege Mode	M

3.4.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ127	IRQ126	IRQ125	IRQ124	IRQ123	IRQ122	IRQ121	IRQ120	IRQ119	IRQ118	IRQ117	IRQ116	IRQ115	IRQ114	IRQ113	IRQ112
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ111	IRQ110	IRQ109	IRQ108	IRQ107	IRQ106	IRQ105	IRQ104	IRQ103	IRQ102	IRQ101	IRQ100	IRQ99	IRQ98	IRQ97	IRQ96

Figure 4. mclcie3 format

3.4.3. Field Summary

Name	Location	Type	Reset Value
IRQ96	0	RW	0
IRQ97	1	RW	0
IRQ98	2	RW	0
IRQ99	3	RW	0
IRQ100	4	RW	0
IRQ101	5	RW	0
IRQ102	6	RW	0
IRQ103	7	RW	0
IRQ104	8	RW	0
IRQ105	9	RW	0
IRQ106	10	RW	0
IRQ107	11	RW	0
IRQ108	12	RW	0
IRQ109	13	RW	0

Name	Location	Type	Reset Value
IRQ110	14	RW	0
IRQ111	15	RW	0
IRQ112	16	RW	0
IRQ113	17	RW	0
IRQ114	18	RW	0
IRQ115	19	RW	0
IRQ116	20	RW	0
IRQ117	21	RW	0
IRQ118	22	RW	0
IRQ119	23	RW	0
IRQ120	24	RW	0
IRQ121	25	RW	0
IRQ122	26	RW	0
IRQ123	27	RW	0
IRQ124	28	RW	0
IRQ125	29	RW	0
IRQ126	30	RW	0
IRQ127	31	RW	0

3.4.4. Fields

IRQ96

Location 0	
Description IRQ97 enabled	
Type RW	
Reset value 0	

IRQ97

Location

1

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ98

Location

2

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ99

Location

3

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ100

Location

4

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ101

Location

5

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ102

Location

6

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ103

Location

7

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ104

Location

8

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ105

Location

9

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ106

Location*10***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ107

Location*11***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ108

Location*12***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ109

Location*13***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ110

Location*14***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ111

Location*15***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ112

Location*16***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ113

Location*17***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ114

Location*18***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ115

Location*19***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ116

Location*20***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ117

Location*21***Description***IRQ97 enabled***Type***RW***Reset value***0*

IRQ118

Location

22

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ119

Location

23

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ120

Location

24

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ121

Location

25

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ122

Location

26

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ123

Location

27

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ124

Location

28

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ125

Location

29

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ126

Location

30

Description*IRQ97 enabled***Type***RW***Reset value**

0

IRQ127

Location*31***Description***IRQ97 enabled***Type***RW***Reset value***0*

DRAFT

3.5. mclicie4

IRQ Enable 4

Enable bits for IRQs 128-159

3.5.1. Attributes

CSR Address	0x7f4
Length	32-bit-bit
Privilege Mode	M

3.5.2. Format



Figure 5. mclicie4 format

3.5.3. Field Summary

Name	Location	Type	Reset Value
IRQ128	0	RW	0
IRQ129	1	RW	0
IRQ130	2	RW	0
IRQ131	3	RW	0
IRQ132	4	RW	0
IRQ133	5	RW	0
IRQ134	6	RW	0
IRQ135	7	RW	0
IRQ136	8	RW	0
IRQ137	9	RW	0
IRQ138	10	RW	0
IRQ139	11	RW	0
IRQ140	12	RW	0
IRQ141	13	RW	0

Name	Location	Type	Reset Value
IRQ142	14	RW	0
IRQ143	15	RW	0
IRQ144	16	RW	0
IRQ145	17	RW	0
IRQ146	18	RW	0
IRQ147	19	RW	0
IRQ148	20	RW	0
IRQ149	21	RW	0
IRQ150	22	RW	0
IRQ151	23	RW	0
IRQ152	24	RW	0
IRQ153	25	RW	0
IRQ154	26	RW	0
IRQ155	27	RW	0
IRQ156	28	RW	0
IRQ157	29	RW	0
IRQ158	30	RW	0
IRQ159	31	RW	0

3.5.4. Fields

IRQ128

Location 0
Description IRQ129 enabled
Type RW
Reset value 0

IRQ129

Location

1

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ130

Location

2

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ131

Location

3

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ132

Location

4

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ133

Location

5

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ134

Location

6

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ135

Location

7

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ136

Location

8

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ137

Location

9

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ138

Location*10***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ139

Location*11***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ140

Location*12***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ141

Location*13***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ142

Location*14***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ143

Location*15***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ144

Location*16***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ145

Location*17***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ146

Location*18***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ147

Location*19***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ148

Location*20***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ149

Location*21***Description***IRQ129 enabled***Type***RW***Reset value***0*

IRQ150

Location

22

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ151

Location

23

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ152

Location

24

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ153

Location

25

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ154

Location

26

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ155

Location

27

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ156

Location

28

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ157

Location

29

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ158

Location

30

Description*IRQ129 enabled***Type***RW***Reset value**

0

IRQ159

Location*31***Description***IRQ129 enabled***Type***RW***Reset value***0*

DRAFT

3.6. mclcie5

IRQ Enable 5

Enable bits for IRQs 160-191

3.6.1. Attributes

CSR Address	0x7f5
Length	32-bit-bit
Privilege Mode	M

3.6.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ191	IRQ190	IRQ189	IRQ188	IRQ187	IRQ186	IRQ185	IRQ184	IRQ183	IRQ182	IRQ181	IRQ180	IRQ179	IRQ178	IRQ177	IRQ176
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ175	IRQ174	IRQ173	IRQ172	IRQ171	IRQ170	IRQ169	IRQ168	IRQ167	IRQ166	IRQ165	IRQ164	IRQ163	IRQ162	IRQ161	IRQ160

Figure 6. mclcie5 format

3.6.3. Field Summary

Name	Location	Type	Reset Value
IRQ160	0	RW	0
IRQ161	1	RW	0
IRQ162	2	RW	0
IRQ163	3	RW	0
IRQ164	4	RW	0
IRQ165	5	RW	0
IRQ166	6	RW	0
IRQ167	7	RW	0
IRQ168	8	RW	0
IRQ169	9	RW	0
IRQ170	10	RW	0
IRQ171	11	RW	0
IRQ172	12	RW	0
IRQ173	13	RW	0

Name	Location	Type	Reset Value
IRQ174	14	RW	0
IRQ175	15	RW	0
IRQ176	16	RW	0
IRQ177	17	RW	0
IRQ178	18	RW	0
IRQ179	19	RW	0
IRQ180	20	RW	0
IRQ181	21	RW	0
IRQ182	22	RW	0
IRQ183	23	RW	0
IRQ184	24	RW	0
IRQ185	25	RW	0
IRQ186	26	RW	0
IRQ187	27	RW	0
IRQ188	28	RW	0
IRQ189	29	RW	0
IRQ190	30	RW	0
IRQ191	31	RW	0

3.6.4. Fields

IRQ160

Location 0
Description IRQ161 enabled
Type RW
Reset value 0

IRQ161

Location*1***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ162

Location*2***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ163

Location*3***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ164

Location

4

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ165

Location

5

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ166

Location

6

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ167

Location

7

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ168

Location

8

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ169

Location

9

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ170

Location*10***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ171

Location*11***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ172

Location*12***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ173

Location*13***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ174

Location*14***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ175

Location*15***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ176

Location*16***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ177

Location*17***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ178

Location*18***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ179

Location*19***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ180

Location*20***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ181

Location*21***Description***IRQ161 enabled***Type***RW***Reset value***0*

IRQ182

Location

22

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ183

Location

23

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ184

Location

24

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ185

Location

25

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ186

Location

26

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ187

Location

27

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ188

Location

28

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ189

Location

29

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ190

Location

30

Description*IRQ161 enabled***Type***RW***Reset value**

0

IRQ191

Location*31***Description***IRQ161 enabled***Type***RW***Reset value***0*

DRAFT

3.7. mclcie6

IRQ Enable 6

Enable bits for IRQs 192-223

3.7.1. Attributes

CSR Address	0x7f6
Length	32-bit-bit
Privilege Mode	M

3.7.2. Format



Figure 7. mclcie6 format

3.7.3. Field Summary

Name	Location	Type	Reset Value
IRQ192	0	RW	0
IRQ193	1	RW	0
IRQ194	2	RW	0
IRQ195	3	RW	0
IRQ196	4	RW	0
IRQ197	5	RW	0
IRQ198	6	RW	0
IRQ199	7	RW	0
IRQ200	8	RW	0
IRQ201	9	RW	0
IRQ202	10	RW	0
IRQ203	11	RW	0
IRQ204	12	RW	0
IRQ205	13	RW	0

Name	Location	Type	Reset Value
IRQ206	14	RW	0
IRQ207	15	RW	0
IRQ208	16	RW	0
IRQ209	17	RW	0
IRQ210	18	RW	0
IRQ211	19	RW	0
IRQ212	20	RW	0
IRQ213	21	RW	0
IRQ214	22	RW	0
IRQ215	23	RW	0
IRQ216	24	RW	0
IRQ217	25	RW	0
IRQ218	26	RW	0
IRQ219	27	RW	0
IRQ220	28	RW	0
IRQ221	29	RW	0
IRQ222	30	RW	0
IRQ223	31	RW	0

3.7.4. Fields

IRQ192

Location

0

Description

IRQ193 enabled

Type

RW

Reset value

0

IRQ193

Location

1

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ194

Location

2

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ195

Location

3

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ196

Location

4

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ197

Location

5

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ198

Location

6

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ199

Location

7

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ200

Location

8

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ201

Location

9

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ202

Location*10***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ203

Location*11***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ204

Location*12***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ205

Location*13***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ206

Location*14***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ207

Location*15***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ208

Location*16***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ209

Location*17***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ210

Location*18***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ211

Location*19***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ212

Location*20***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ213

Location*21***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ214

Location

22

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ215

Location

23

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ216

Location

24

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ217

Location

25

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ218

Location

26

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ219

Location

27

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ220

Location

28

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ221

Location

29

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ222

Location

30

Description*IRQ193 enabled***Type***RW***Reset value**

0

IRQ223

Location*31***Description***IRQ193 enabled***Type***RW***Reset value***0*

DRAFT

3.8. mclcie7

IRQ Enable 7

Enable bits for IRQs 224-255

3.8.1. Attributes

CSR Address	0x7f7
Length	32-bit-bit
Privilege Mode	M

3.8.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ255	IRQ254	IRQ253	IRQ252	IRQ251	IRQ250	IRQ249	IRQ248	IRQ247	IRQ246	IRQ245	IRQ244	IRQ243	IRQ242	IRQ241	IRQ240
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ239	IRQ238	IRQ237	IRQ236	IRQ235	IRQ234	IRQ233	IRQ232	IRQ231	IRQ230	IRQ229	IRQ228	IRQ227	IRQ226	IRQ225	IRQ224

Figure 8. mclcie7 format

3.8.3. Field Summary

Name	Location	Type	Reset Value
IRQ224	0	RW	0
IRQ225	1	RW	0
IRQ226	2	RW	0
IRQ227	3	RW	0
IRQ228	4	RW	0
IRQ229	5	RW	0
IRQ230	6	RW	0
IRQ231	7	RW	0
IRQ232	8	RW	0
IRQ233	9	RW	0
IRQ234	10	RW	0
IRQ235	11	RW	0
IRQ236	12	RW	0
IRQ237	13	RW	0

Name	Location	Type	Reset Value
IRQ238	14	RW	0
IRQ239	15	RW	0
IRQ240	16	RW	0
IRQ241	17	RW	0
IRQ242	18	RW	0
IRQ243	19	RW	0
IRQ244	20	RW	0
IRQ245	21	RW	0
IRQ246	22	RW	0
IRQ247	23	RW	0
IRQ248	24	RW	0
IRQ249	25	RW	0
IRQ250	26	RW	0
IRQ251	27	RW	0
IRQ252	28	RW	0
IRQ253	29	RW	0
IRQ254	30	RW	0
IRQ255	31	RW	0

3.8.4. Fields

IRQ224

Location 0
Description IRQ225 enabled
Type RW
Reset value 0

IRQ225

Location

1

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ226

Location

2

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ227

Location

3

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ228

Location

4

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ229

Location

5

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ230

Location

6

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ231

Location	7
Description	<i>IRQ225 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ232

Location	8
Description	<i>IRQ225 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ233

Location	9
Description	<i>IRQ225 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ234

Location*10***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ235

Location*11***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ236

Location*12***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ237

Location	13
Description	IRQ225 enabled
Type	RW
Reset value	0

IRQ238

Location	14
Description	IRQ225 enabled
Type	RW
Reset value	0

IRQ239

Location	15
Description	IRQ225 enabled
Type	RW
Reset value	0

IRQ240

Location*16***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ241

Location*17***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ242

Location*18***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ243

Location*19***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ244

Location*20***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ245

Location*21***Description***IRQ225 enabled***Type***RW***Reset value***0*

IRQ246

Location

22

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ247

Location

23

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ248

Location

24

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ249

Location	25
Description	<i>IRQ225 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ250

Location	26
Description	<i>IRQ225 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ251

Location	27
Description	<i>IRQ225 enabled</i>
Type	<i>RW</i>
Reset value	0

IRQ252

Location

28

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ253

Location

29

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ254

Location

30

Description*IRQ225 enabled***Type***RW***Reset value**

0

IRQ255

Location	31
Description	IRQ225 enabled
Type	RW
Reset value	0

DRAFT

3.9. mclicip0

IRQ Pending 0

Pending bits for IRQs 0-31

3.9.1. Attributes

CSR Address	0x7f0
Length	32-bit-bit
Privilege Mode	M

3.9.2. Format

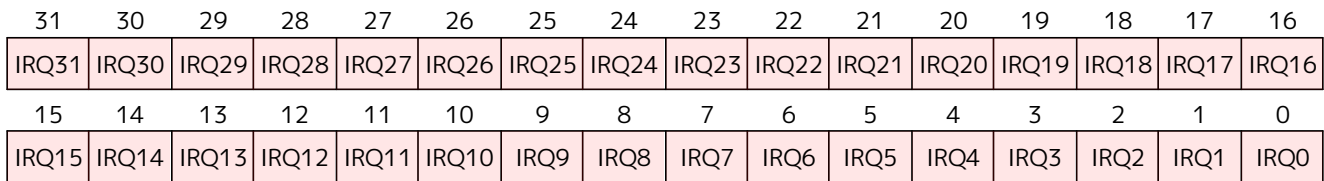


Figure 9. mclicip0 format

3.9.3. Field Summary

Name	Location	Type	Reset Value
IRQ0	0	RW	0
IRQ1	1	RW	0
IRQ2	2	RW	0
IRQ3	3	RW	0
IRQ4	4	RW	0
IRQ5	5	RW	0
IRQ6	6	RW	0
IRQ7	7	RW	0
IRQ8	8	RW	0
IRQ9	9	RW	0
IRQ10	10	RW	0
IRQ11	11	RW	0
IRQ12	12	RW	0
IRQ13	13	RW	0

Name	Location	Type	Reset Value
IRQ14	14	RW	0
IRQ15	15	RW	0
IRQ16	16	RW	0
IRQ17	17	RW	0
IRQ18	18	RW	0
IRQ19	19	RW	0
IRQ20	20	RW	0
IRQ21	21	RW	0
IRQ22	22	RW	0
IRQ23	23	RW	0
IRQ24	24	RW	0
IRQ25	25	RW	0
IRQ26	26	RW	0
IRQ27	27	RW	0
IRQ28	28	RW	0
IRQ29	29	RW	0
IRQ30	30	RW	0
IRQ31	31	RW	0

3.9.4. Fields

IRQ0

Location

0

Description
IRQ1 pending
Type
RW
Reset value

0

IRQ1

Location

1

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ2

Location

2

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ3

Location

3

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ4

Location

4

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ5

Location

5

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ6

Location

6

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ7

Location

7

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ8

Location

8

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ9

Location

9

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ10

Location*10***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ11

Location*11***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ12

Location*12***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ13

Location*13***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ14

Location*14***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ15

Location*15***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ16

Location*16***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ17

Location*17***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ18

Location*18***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ19

Location*19***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ20

Location*20***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ21

Location*21***Description***IRQ1 pending***Type***RW***Reset value***0*

IRQ22

Location

22

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ23

Location

23

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ24

Location

24

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ25

Location

25

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ26

Location

26

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ27

Location

27

Description*IRQ1 pending***Type***RW***Reset value**

0

IRQ28

Location
28
Description
<i>IRQ1 pending</i>
Type
<i>RW</i>
Reset value
0

IRQ29

Location
29
Description
<i>IRQ1 pending</i>
Type
<i>RW</i>
Reset value
0

IRQ30

Location
30
Description
<i>IRQ1 pending</i>
Type
<i>RW</i>
Reset value
0

IRQ31

Location	
31	
Description	
IRQ1 pending	
Type	
RW	
Reset value	
0	

DRAFT

3.10. mclicip1

IRQ Pending 1

Pending bits for IRQs 32-63

3.10.1. Attributes

CSR Address	0x7f1
Length	32-bit-bit
Privilege Mode	M

3.10.2. Format

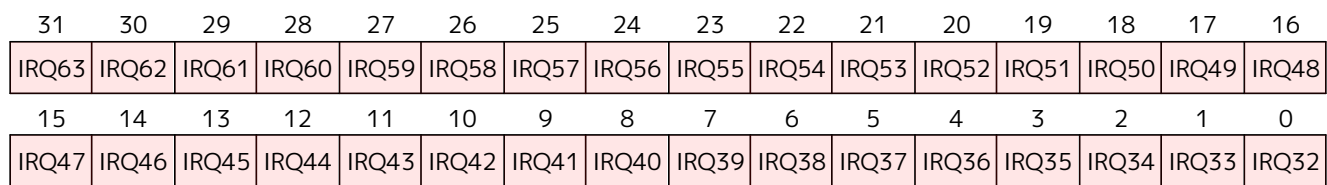


Figure 10. mclicip1 format

3.10.3. Field Summary

Name	Location	Type	Reset Value
IRQ32	0	RW	0
IRQ33	1	RW	0
IRQ34	2	RW	0
IRQ35	3	RW	0
IRQ36	4	RW	0
IRQ37	5	RW	0
IRQ38	6	RW	0
IRQ39	7	RW	0
IRQ40	8	RW	0
IRQ41	9	RW	0
IRQ42	10	RW	0
IRQ43	11	RW	0
IRQ44	12	RW	0
IRQ45	13	RW	0

Name	Location	Type	Reset Value
IRQ46	14	RW	0
IRQ47	15	RW	0
IRQ48	16	RW	0
IRQ49	17	RW	0
IRQ50	18	RW	0
IRQ51	19	RW	0
IRQ52	20	RW	0
IRQ53	21	RW	0
IRQ54	22	RW	0
IRQ55	23	RW	0
IRQ56	24	RW	0
IRQ57	25	RW	0
IRQ58	26	RW	0
IRQ59	27	RW	0
IRQ60	28	RW	0
IRQ61	29	RW	0
IRQ62	30	RW	0
IRQ63	31	RW	0

3.10.4. Fields

IRQ32

Location 0 Description IRQ33 pending Type RW Reset value 0
--

IRQ33

Location

1

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ34

Location

2

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ35

Location

3

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ36

Location

4

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ37

Location

5

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ38

Location

6

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ39

Location

7

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ40

Location

8

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ41

Location

9

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ42

Location*10***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ43

Location*11***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ44

Location*12***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ45

Location*13***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ46

Location*14***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ47

Location*15***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ48

Location*16***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ49

Location*17***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ50

Location*18***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ51

Location*19***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ52

Location*20***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ53

Location*21***Description***IRQ33 pending***Type***RW***Reset value***0*

IRQ54

Location

22

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ55

Location

23

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ56

Location

24

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ57

Location

25

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ58

Location

26

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ59

Location

27

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ60

Location

28

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ61

Location

29

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ62

Location

30

Description*IRQ33 pending***Type***RW***Reset value**

0

IRQ63

Location	31
Description	IRQ33 pending
Type	RW
Reset value	0

DRAFT

3.11. mclicip2

IRQ Pending 2

Pending bits for IRQs 64-95

3.11.1. Attributes

CSR Address	0x7f2
Length	32-bit-bit
Privilege Mode	M

3.11.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ95	IRQ94	IRQ93	IRQ92	IRQ91	IRQ90	IRQ89	IRQ88	IRQ87	IRQ86	IRQ85	IRQ84	IRQ83	IRQ82	IRQ81	IRQ80
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ79	IRQ78	IRQ77	IRQ76	IRQ75	IRQ74	IRQ73	IRQ72	IRQ71	IRQ70	IRQ69	IRQ68	IRQ67	IRQ66	IRQ65	IRQ64

Figure 11. mclicip2 format

3.11.3. Field Summary

Name	Location	Type	Reset Value
IRQ64	0	RW	0
IRQ65	1	RW	0
IRQ66	2	RW	0
IRQ67	3	RW	0
IRQ68	4	RW	0
IRQ69	5	RW	0
IRQ70	6	RW	0
IRQ71	7	RW	0
IRQ72	8	RW	0
IRQ73	9	RW	0
IRQ74	10	RW	0
IRQ75	11	RW	0
IRQ76	12	RW	0
IRQ77	13	RW	0

Name	Location	Type	Reset Value
IRQ78	14	RW	0
IRQ79	15	RW	0
IRQ80	16	RW	0
IRQ81	17	RW	0
IRQ82	18	RW	0
IRQ83	19	RW	0
IRQ84	20	RW	0
IRQ85	21	RW	0
IRQ86	22	RW	0
IRQ87	23	RW	0
IRQ88	24	RW	0
IRQ89	25	RW	0
IRQ90	26	RW	0
IRQ91	27	RW	0
IRQ92	28	RW	0
IRQ93	29	RW	0
IRQ94	30	RW	0
IRQ95	31	RW	0

3.11.4. Fields

IRQ64

Location 0
Description IRQ65 pending
Type RW
Reset value 0

IRQ65

Location

1

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ66

Location

2

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ67

Location

3

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ68

Location

4

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ69

Location

5

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ70

Location

6

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ71

Location

7

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ72

Location

8

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ73

Location

9

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ74

Location*10***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ75

Location*11***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ76

Location*12***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ77

Location*13***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ78

Location*14***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ79

Location*15***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ80

Location*16***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ81

Location*17***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ82

Location*18***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ83

Location*19***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ84

Location*20***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ85

Location*21***Description***IRQ65 pending***Type***RW***Reset value***0*

IRQ86

Location

22

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ87

Location

23

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ88

Location

24

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ89

Location

25

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ90

Location

26

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ91

Location

27

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ92

Location

28

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ93

Location

29

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ94

Location

30

Description*IRQ65 pending***Type***RW***Reset value**

0

IRQ95

Location*31***Description***IRQ65 pending***Type***RW***Reset value***0*

DRAFT

3.12. mclicip3

IRQ Pending 3

Pending bits for IRQs 96-127

3.12.1. Attributes

CSR Address	0x7f3
Length	32-bit-bit
Privilege Mode	M

3.12.2. Format

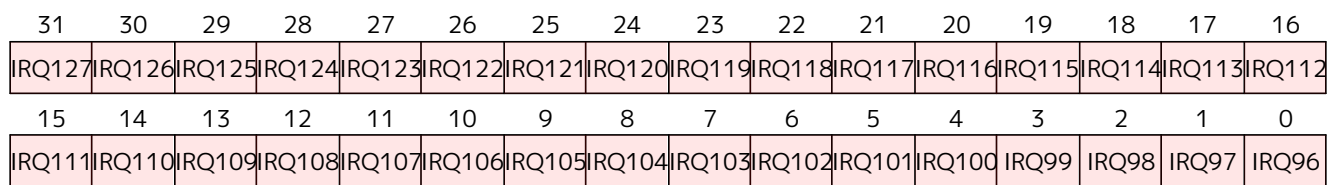


Figure 12. mclicip3 format

3.12.3. Field Summary

Name	Location	Type	Reset Value
IRQ96	0	RW	0
IRQ97	1	RW	0
IRQ98	2	RW	0
IRQ99	3	RW	0
IRQ100	4	RW	0
IRQ101	5	RW	0
IRQ102	6	RW	0
IRQ103	7	RW	0
IRQ104	8	RW	0
IRQ105	9	RW	0
IRQ106	10	RW	0
IRQ107	11	RW	0
IRQ108	12	RW	0
IRQ109	13	RW	0

Name	Location	Type	Reset Value
IRQ110	14	RW	0
IRQ111	15	RW	0
IRQ112	16	RW	0
IRQ113	17	RW	0
IRQ114	18	RW	0
IRQ115	19	RW	0
IRQ116	20	RW	0
IRQ117	21	RW	0
IRQ118	22	RW	0
IRQ119	23	RW	0
IRQ120	24	RW	0
IRQ121	25	RW	0
IRQ122	26	RW	0
IRQ123	27	RW	0
IRQ124	28	RW	0
IRQ125	29	RW	0
IRQ126	30	RW	0
IRQ127	31	RW	0

3.12.4. Fields

IRQ96

Location

0

Description

IRQ97 pending

Type

RW

Reset value

0

IRQ97

Location

1

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ98

Location

2

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ99

Location

3

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ100

Location

4

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ101

Location

5

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ102

Location

6

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ103

Location

7

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ104

Location

8

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ105

Location

9

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ106

Location*10***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ107

Location*11***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ108

Location*12***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ109

Location*13***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ110

Location*14***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ111

Location*15***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ112

Location*16***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ113

Location*17***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ114

Location*18***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ115

Location*19***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ116

Location*20***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ117

Location*21***Description***IRQ97 pending***Type***RW***Reset value***0*

IRQ118

Location

22

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ119

Location

23

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ120

Location

24

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ121

Location

25

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ122

Location

26

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ123

Location

27

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ124

Location

28

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ125

Location

29

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ126

Location

30

Description*IRQ97 pending***Type***RW***Reset value**

0

IRQ127

Location*31***Description***IRQ97 pending***Type***RW***Reset value***0*

DRAFT

3.13. mclicip4

IRQ Pending 4

Pending bits for IRQs 128-159

3.13.1. Attributes

CSR Address	0x7f4
Length	32-bit-bit
Privilege Mode	M

3.13.2. Format



Figure 13. mclicip4 format

3.13.3. Field Summary

Name	Location	Type	Reset Value
IRQ128	0	RW	0
IRQ129	1	RW	0
IRQ130	2	RW	0
IRQ131	3	RW	0
IRQ132	4	RW	0
IRQ133	5	RW	0
IRQ134	6	RW	0
IRQ135	7	RW	0
IRQ136	8	RW	0
IRQ137	9	RW	0
IRQ138	10	RW	0
IRQ139	11	RW	0
IRQ140	12	RW	0
IRQ141	13	RW	0

Name	Location	Type	Reset Value
IRQ142	14	RW	0
IRQ143	15	RW	0
IRQ144	16	RW	0
IRQ145	17	RW	0
IRQ146	18	RW	0
IRQ147	19	RW	0
IRQ148	20	RW	0
IRQ149	21	RW	0
IRQ150	22	RW	0
IRQ151	23	RW	0
IRQ152	24	RW	0
IRQ153	25	RW	0
IRQ154	26	RW	0
IRQ155	27	RW	0
IRQ156	28	RW	0
IRQ157	29	RW	0
IRQ158	30	RW	0
IRQ159	31	RW	0

3.13.4. Fields

IRQ128

Location

0

Description

IRQ129 pending

Type

RW

Reset value

0

IRQ129

Location

1

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ130

Location

2

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ131

Location

3

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ132

Location

4

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ133

Location

5

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ134

Location

6

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ135

Location

7

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ136

Location

8

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ137

Location

9

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ138

Location*10***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ139

Location*11***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ140

Location*12***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ141

Location*13***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ142

Location*14***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ143

Location*15***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ144

Location*16***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ145

Location*17***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ146

Location*18***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ147

Location*19***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ148

Location*20***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ149

Location*21***Description***IRQ129 pending***Type***RW***Reset value***0*

IRQ150

Location

22

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ151

Location

23

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ152

Location

24

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ153

Location

25

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ154

Location

26

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ155

Location

27

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ156

Location

28

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ157

Location

29

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ158

Location

30

Description*IRQ129 pending***Type***RW***Reset value**

0

IRQ159

Location*31***Description***IRQ129 pending***Type***RW***Reset value***0*

DRAFT

3.14. mclicip5

IRQ Pending 5

Pending bits for IRQs 160-191

3.14.1. Attributes

CSR Address	0x7f5
Length	32-bit-bit
Privilege Mode	M

3.14.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ191	IRQ190	IRQ189	IRQ188	IRQ187	IRQ186	IRQ185	IRQ184	IRQ183	IRQ182	IRQ181	IRQ180	IRQ179	IRQ178	IRQ177	IRQ176
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ175	IRQ174	IRQ173	IRQ172	IRQ171	IRQ170	IRQ169	IRQ168	IRQ167	IRQ166	IRQ165	IRQ164	IRQ163	IRQ162	IRQ161	IRQ160

Figure 14. mclicip5 format

3.14.3. Field Summary

Name	Location	Type	Reset Value
IRQ160	0	RW	0
IRQ161	1	RW	0
IRQ162	2	RW	0
IRQ163	3	RW	0
IRQ164	4	RW	0
IRQ165	5	RW	0
IRQ166	6	RW	0
IRQ167	7	RW	0
IRQ168	8	RW	0
IRQ169	9	RW	0
IRQ170	10	RW	0
IRQ171	11	RW	0
IRQ172	12	RW	0
IRQ173	13	RW	0

Name	Location	Type	Reset Value
IRQ174	14	RW	0
IRQ175	15	RW	0
IRQ176	16	RW	0
IRQ177	17	RW	0
IRQ178	18	RW	0
IRQ179	19	RW	0
IRQ180	20	RW	0
IRQ181	21	RW	0
IRQ182	22	RW	0
IRQ183	23	RW	0
IRQ184	24	RW	0
IRQ185	25	RW	0
IRQ186	26	RW	0
IRQ187	27	RW	0
IRQ188	28	RW	0
IRQ189	29	RW	0
IRQ190	30	RW	0
IRQ191	31	RW	0

3.14.4. Fields

IRQ160

Location 0
Description IRQ161 pending
Type RW
Reset value 0

IRQ161

Location

1

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ162

Location

2

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ163

Location

3

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ164

Location

4

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ165

Location

5

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ166

Location

6

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ167

Location

7

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ168

Location

8

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ169

Location

9

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ170

Location*10***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ171

Location*11***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ172

Location*12***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ173

Location*13***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ174

Location*14***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ175

Location*15***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ176

Location*16***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ177

Location*17***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ178

Location*18***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ179

Location*19***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ180

Location*20***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ181

Location*21***Description***IRQ161 pending***Type***RW***Reset value***0*

IRQ182

Location

22

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ183

Location

23

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ184

Location

24

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ185

Location

25

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ186

Location

26

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ187

Location

27

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ188

Location

28

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ189

Location

29

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ190

Location

30

Description*IRQ161 pending***Type***RW***Reset value**

0

IRQ191

Location*31***Description***IRQ161 pending***Type***RW***Reset value***0*

DRAFT

3.15. mclicip6

IRQ Pending 6

Pending bits for IRQs 192-223

3.15.1. Attributes

CSR Address	0x7f6
Length	32-bit-bit
Privilege Mode	M

3.15.2. Format



Figure 15. mclicip6 format

3.15.3. Field Summary

Name	Location	Type	Reset Value
IRQ192	0	RW	0
IRQ193	1	RW	0
IRQ194	2	RW	0
IRQ195	3	RW	0
IRQ196	4	RW	0
IRQ197	5	RW	0
IRQ198	6	RW	0
IRQ199	7	RW	0
IRQ200	8	RW	0
IRQ201	9	RW	0
IRQ202	10	RW	0
IRQ203	11	RW	0
IRQ204	12	RW	0
IRQ205	13	RW	0

Name	Location	Type	Reset Value
IRQ206	14	RW	0
IRQ207	15	RW	0
IRQ208	16	RW	0
IRQ209	17	RW	0
IRQ210	18	RW	0
IRQ211	19	RW	0
IRQ212	20	RW	0
IRQ213	21	RW	0
IRQ214	22	RW	0
IRQ215	23	RW	0
IRQ216	24	RW	0
IRQ217	25	RW	0
IRQ218	26	RW	0
IRQ219	27	RW	0
IRQ220	28	RW	0
IRQ221	29	RW	0
IRQ222	30	RW	0
IRQ223	31	RW	0

3.15.4. Fields

IRQ192

Location

0

Description

IRQ193 pending

Type

RW

Reset value

0

IRQ193

Location

1

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ194

Location

2

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ195

Location

3

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ196

Location

4

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ197

Location

5

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ198

Location

6

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ199

Location

7

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ200

Location

8

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ201

Location

9

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ202

Location*10***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ203

Location*11***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ204

Location*12***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ205

Location*13***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ206

Location*14***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ207

Location*15***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ208

Location*16***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ209

Location*17***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ210

Location*18***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ211

Location*19***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ212

Location*20***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ213

Location*21***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ214

Location

22

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ215

Location

23

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ216

Location

24

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ217

Location

25

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ218

Location

26

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ219

Location

27

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ220

Location

28

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ221

Location

29

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ222

Location

30

Description*IRQ193 pending***Type***RW***Reset value**

0

IRQ223

Location*31***Description***IRQ193 pending***Type***RW***Reset value***0*

DRAFT

3.16. mclicip7

IRQ Pending 7

Pending bits for IRQs 224-255

3.16.1. Attributes

CSR Address	0x7f7
Length	32-bit-bit
Privilege Mode	M

3.16.2. Format



Figure 16. mclicip7 format

3.16.3. Field Summary

Name	Location	Type	Reset Value
IRQ224	0	RW	0
IRQ225	1	RW	0
IRQ226	2	RW	0
IRQ227	3	RW	0
IRQ228	4	RW	0
IRQ229	5	RW	0
IRQ230	6	RW	0
IRQ231	7	RW	0
IRQ232	8	RW	0
IRQ233	9	RW	0
IRQ234	10	RW	0
IRQ235	11	RW	0
IRQ236	12	RW	0
IRQ237	13	RW	0

Name	Location	Type	Reset Value
IRQ238	14	RW	0
IRQ239	15	RW	0
IRQ240	16	RW	0
IRQ241	17	RW	0
IRQ242	18	RW	0
IRQ243	19	RW	0
IRQ244	20	RW	0
IRQ245	21	RW	0
IRQ246	22	RW	0
IRQ247	23	RW	0
IRQ248	24	RW	0
IRQ249	25	RW	0
IRQ250	26	RW	0
IRQ251	27	RW	0
IRQ252	28	RW	0
IRQ253	29	RW	0
IRQ254	30	RW	0
IRQ255	31	RW	0

3.16.4. Fields

IRQ224

Location 0 Description IRQ225 pending Type RW Reset value 0

IRQ225

Location

1

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ226

Location

2

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ227

Location

3

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ228

Location

4

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ229

Location

5

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ230

Location

6

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ231

Location

7

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ232

Location

8

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ233

Location

9

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ234

Location*10***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ235

Location*11***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ236

Location*12***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ237

Location*13***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ238

Location*14***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ239

Location*15***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ240

Location*16***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ241

Location*17***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ242

Location*18***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ243

Location*19***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ244

Location*20***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ245

Location*21***Description***IRQ225 pending***Type***RW***Reset value***0*

IRQ246

Location

22

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ247

Location

23

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ248

Location

24

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ249

Location

25

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ250

Location

26

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ251

Location

27

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ252

Location

28

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ253

Location

29

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ254

Location

30

Description*IRQ225 pending***Type***RW***Reset value**

0

IRQ255

Location*31***Description***IRQ225 pending***Type***RW***Reset value***0*

DRAFT

Chapter 4. Instruction summary

The following 148 instructions are added by this extension:

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1
✓		qc16.bexti rd, shamt	Single-Bit extract (Immediate)	✓
✓		qc16.bseti rd, shamt	Single-Bit set (Immediate)	✓
✓		qc16.clrint rs1	Clear interrupt (Register)	✓
✓		qc16.delay imm	Delay execution	✓
✓		qc16.di rd	Disable interrupts	✓
✓		qc16.dir rd	Disable interrupts (Register)	✓
✓		qc16.ei rs1	Enable interrupts	✓
✓		qc16.eir rs1	Restore interrupts (Register)	✓
✓		qc16.extu rd, width	Extract bits unsigned	✓
✓		`qc16.mienter.nest`	Machine mode interrupt enter	✓
✓		`qc16.mienter`	Machine mode interrupt enter	✓
✓		`qc16.mileaveret`	Machine mode interrupt exit	✓
✓		`qc16.mnret`	Machine NMI Return	✓
✓		qc16.mpyaddi rd, rs1, uimm	Multiply and accumulate (Immediate)	✓
✓		`qc16.mret`	Machine Exception Return	✓
✓		qc16.mveqz rd, rs1	Conditional Move if equal to zero	✓
✓		qc16.setint rs1	Set interrupt (Register)	✓
✓		qc32.addsat rd, rs1, rs2	Saturating signed addition	✓
✓		qc32.addusat rd, rs1, rs2	Saturating unsigned addition	✓
✓		qc32.beqi rs1, imm, offset	Branch on equal (Immediate)	✓
✓		qc32.bgei rs1, imm, offset	Branch on greater than or equal (immediate)	✓
✓		qc32.bgeui rs1, imm, offset	Branch on greater than or equal unsigned (immediate)	✓
✓		qc32.blتي rs1, imm, offset	Branch on less than (immediate)	✓
✓		qc32.bltui rs1, imm, offset	Branch on less than unsigned (immediate)	✓
✓		qc32.bnei rs1, imm, offset	Branch on not equal (immediate)	✓
✓		qc32.brev32 rd, rs1	Reverse bit order	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1
✓		qc32.clo rd, rs1	Count leading ones	✓
✓		qc32.clrint rs3	Clear interrupt (Register)	✓
✓		qc32.clrinti imm	Clear interrupt (Immediate)	✓
✓		qc32.compress2 rd, rs1	Bit compression (every 2nd bit)	✓
✓		qc32.compress3 rd, rs1	Bit compression (every 3rd bit)	✓
✓	✓	qc32.csrrwr rd, rs1, rs2	Atomic Read/Write CSR (Register)	✓
✓	✓	qc32.csrrwri rd, imm, rs2	Atomic Read/Write CSR (Register) Immediate	✓
✓		qc32.cto rd, rs1	Count trailing ones	✓
✓		qc32.delay imm	Delay execution	✓
✓		qc32.expand2 rd, rs1	Bit expansion (every 2nd bit)	✓
✓		qc32.expand3 rd, rs1	Bit expansion (every 3rd bit)	✓
✓		qc32.extds rd, rs1, width, shamt	Extract bits from pair signed (Immediate)	✓
✓		qc32.extdspr rd, rs1, rs2	Extract bits from pair signed, packed descriptor (Register)	✓
✓		qc32.extdsprh rd, rs1, rs2	Extract bits from pair signed, packed descriptor high part (Register)	✓
✓		qc32.extdsr rd, rs1, rs2	Extract bits from pair signed (Register)	✓
✓		qc32.extdu rd, rs1, width, shamt	Extract bits from pair unsigned (Immediate)	✓
✓		qc32.extdupr rd, rs1, rs2	Extract bits from pair unsigned, packed descriptor (Register)	✓
✓		qc32.extduprh rd, rs1, rs2	Extract bits from pair unsigned, packed descriptor high part (Register)	✓
✓		qc32.extdur rd, rs1, rs2	Extract bits from pair unsigned (Register)	✓
✓		qc32.exts rd, rs1, width, shamt	Extract bits signed	✓
✓		qc32.extu rd, rs1, width, shamt	Extract bits unsigned	✓
✓		qc32.insb rd, rs1, width, shamt	Insert bits (Register)	✓
✓		qc32.insbh rd, rs1, width, shamt	Insert bits in 64-bit higher part (Register)	✓
✓		qc32.insbhr rd, rs1, rs2	Insert bits in 64-bit higher part (Register)	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1
✓		qc32.insbi rd, imm, width, shamt	Insert bits (Immediate)	✓
✓		qc32.insbpr rd, rs1, rs2	Insert bits, packed descriptor (Register)	✓
✓		qc32.insbprh rd, rs1, rs2	Insert bits, packed descriptor high part (Register)	✓
✓		qc32.insbr rd, rs1, rs2	Insert bits (Register)	✓
✓		qc32.insbri rd, rs1, imm	Insert bits (Immediate)	✓
✓		qc32.li rd, imm	Load immediate large	✓
✓		qc32.lieq rd, rs1, rs2, simm	Conditional load immediate if equal (Register)	✓
✓		qc32.lieqi rd, rs1, imm, simm	Conditional load immediate if equal (Immediate)	✓
✓		qc32.lige rd, rs1, rs2, simm	Conditional load immediate if great or equal than (Register)	✓
✓		qc32.ligei rd, rs1, imm, simm	Conditional load immediate if great or equal than (Immediate)	✓
✓		qc32.ligeu rd, rs1, rs2, simm	Conditional load immediate if great or equal than unsigned (Register)	✓
✓		qc32.ligeui rd, rs1, imm, simm	Conditional load immediate if great or equal than unsigned (Immediate)	✓
✓		qc32.lilt rd, rs1, rs2, simm	Conditional load immediate if less than (Register)	✓
✓		qc32.lilti rd, rs1, imm, simm	Conditional load immediate if less than (Immediate)	✓
✓		qc32.liltu rd, rs1, rs2, simm	Conditional load immediate if less than unsigned (Register)	✓
✓		qc32.liltui rd, rs1, imm, simm	Conditional load immediate if less than unsigned (Immediate)	✓
✓		qc32.line rd, rs1, rs2, simm	Conditional load immediate if not equal (Register)	✓
✓		qc32.linei rd, rs1, imm, simm	Conditional load immediate if not equal (Immediate)	✓
✓		qc32.lrb rd, rs1, rs2, shamt	Load indexed byte	✓
✓		qc32.lrbu rd, rs1, rs2, shamt	Load indexed unsigned byte	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1
✓		qc32.lrh rd, rs1, rs2, shamt	Load indexed halfword	✓
✓		qc32.lrhu rd, rs1, rs2, shamt	Load indexed unsigned halfword	✓
✓		qc32.lrw rd, rs1, rs2, shamt	Load indexed word	✓
✓		qc32.lwm rd, rs2, imm(rs1)	Load word multiple	✓
✓		qc32.lwmi rd, length, imm(rs1)	Load word multiple (Immediate)	✓
✓		`qc32.mienter.nest`	Machine mode interrupt enter	✓
✓		`qc32.mienter`	Machine mode interrupt enter	✓
✓		`qc32.mileaveret`	Machine mode interrupt exit	✓
✓		`qc32.mnret`	Machine NMI Return	✓
✓		qc32.mpyaddi rd, rs1, imm	Multiply and accumulate (Immediate)	✓
✓		qc32.mveq rd, rs1, rs2, rs3	Conditional move if equal (Register)	✓
✓		qc32.mveqi rd, rs1, imm, rs3	Conditional move if equal (Immediate)	✓
✓		qc32.mvge rd, rs1, rs2, rs3	Conditional move if great or equal than (Register)	✓
✓		qc32.mvgei rd, rs1, imm, rs3	Conditional move if great or equal than (Immediate)	✓
✓		qc32.mvgeu rd, rs1, rs2, rs3	Conditional move if great or equal than unsigned (Register)	✓
✓		qc32.mvgeui rd, rs1, imm, rs3	Conditional move if great or equal than unsigned (Immediate)	✓
✓		qc32.mvlt rd, rs1, rs2, rs3	Conditional move if less than (Register)	✓
✓		qc32.mvlti rd, rs1, imm, rs3	Conditional move if less than (Immediate)	✓
✓		qc32.mvltu rd, rs1, rs2, rs3	Conditional move if less than unsigned (Register)	✓
✓		qc32.mvltui rd, rs1, imm, rs3	Conditional move if less than unsigned (Immediate)	✓
✓		qc32.mvne rd, rs1, rs2, rs3	Conditional move if not equal (Register)	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1
✓		qc32.mvnei rd, rs1, imm, rs3	Conditional move if not equal (Immediate)	✓
✓		qc32.norm rd, rs1	Signed Normalization	✓
✓		qc32.normeu rd, rs1	Unsigned Even Normalization	✓
✓		qc32.normu rd, rs1	Unsigned Normalization	✓
✓		qc32.selecteqi rd, imm, rs2, rs3	Select load immediate or register if equal (Immediate)	✓
✓		qc32.selectieq rd, rs1, rs2, simm2	Select load immediate or register if equal (Register)	✓
✓		qc32.selectieqi rd, imm, rs2, simm2	Select load immediate or register if equal (Immediate)	✓
✓		qc32.selectiieq rd, rs1, simm1, simm2	Select load immediate if equal (Register)	✓
✓		qc32.selectiine rd, rs1, simm1, simm2	Select load immediate if not equal (Register)	✓
✓		qc32.selectine rd, rs1, rs2, simm2	Select load immediate or register if not equal (Register)	✓
✓		qc32.selectinei rd, imm, rs2, simm2	Select load immediate or register if not equal (Immediate)	✓
✓		qc32.selectnei rd, imm, rs2, rs3	Select load immediate or register if not equal (Immediate)	✓
✓		qc32.setint rs3	Set interrupt (Register)	✓
✓		qc32.setinti imm	Set interrupt (Immediate)	✓
✓		qc32.setwm rs3, rs2, imm(rs1)	Set word multiple (Register)	✓
✓		qc32.setwmi rs3, rs2, imm(rs1)	Set word multiple (Immediate)	✓
✓		qc32.sh4add rd, rs1, rs2	Shift left by 4 and add	✓
✓		qc32.sh5add rd, rs1, rs2	Shift left by 5 and add	✓
✓		qc32.sh6add rd, rs1, rs2	Shift left by 6 and add	✓
✓		qc32.sh7add rd, rs1, rs2	Shift left by 7 and add	✓
✓		qc32.sh8add rd, rs1, rs2	Shift left by 8 and add	✓
✓		qc32.slasat rd, rs1, rs2	Saturating arithmetic left shift	✓
✓		qc32.sllsat rd, rs1, rs2	Saturating logical left shift	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1
✓		qc32.srb rs3, rs1, rs2, shamt	Store indexed byte	✓
✓		qc32.srh rs3, rs1, rs2, shamt	Store indexed halfword	✓
✓		qc32.srw rs3, rs1, rs2, shamt	Store indexed word	✓
✓		qc32.subsat rd, rs1, rs2	Saturating signed subtraction	✓
✓		qc32.subusat rd, rs1, rs2	Saturating unsigned subtraction	✓
✓		qc32.swm rs3, rs2, imm(rs1)	Store word multiple	✓
✓		qc32.swmi rs3, length, imm(rs1)	Store word multiple (immediate)	✓
✓		qc32.wrap rd, rs1, rs2	Wraparound (Register)	✓
✓		qc32.wrapi rd, rs1, imm	Wraparound (Immediate)	✓
✓		qc48.addai rd, imm	Add immediate	✓
✓		qc48.addi rd, rs1, imm	Add immediate	✓
✓		qc48.andai rd, imm	And immediate	✓
✓		qc48.andi rd, rs1, imm	Add immediate	✓
✓		qc48.beqi rs1, imm, offset	Branch on equal (immediate)	✓
✓		qc48.bgei rs1, imm, offset	Branch on greater than or equal (immediate)	✓
✓		qc48.bgeui rs1, imm, offset	Branch on greater than or equal unsigned (immediate)	✓
✓		qc48.blتي rs1, imm, offset	Branch on less than (immediate)	✓
✓		qc48.blтui rs1, imm, offset	Branch on less than unsigned (immediate)	✓
✓		qc48.bnei rs1, imm, offset	Branch on not equal (immediate)	✓
✓		qc48.j imm	Jump	✓
✓		qc48.jal imm	Jump and link	✓
✓		qc48.lb rd, imm(rs1)	Load byte	✓
✓		qc48.lbu rd, imm(rs1)	Load byte unsigned	✓
✓		qc48.lh rd, imm(rs1)	Load halfword	✓
✓		qc48.lhu rd, imm(rs1)	Load halfword unsigned	✓
✓		qc48.li rd, imm	Load immediate large	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1
✓		qc48.lw rd, imm(rs1)	Load word	✓
✓		qc48.orai rd, imm	Or immediate	✓
✓		qc48.ori rd, rs1, imm	Or immediate	✓
✓		qc48.sb rs2, imm(rs1)	Store byte	✓
✓		qc48.sh rs2, imm(rs1)	Store halfword	✓
✓		qc48.sw rs2, imm(rs1)	Store word	✓
✓		qc48.xorai rd, imm	Exclusive Or immediate	✓
✓		qc48.xori rd, rs1, imm	Exclusive Or immediate	✓

DRAFT

Chapter 5. Instructions (in alphabetical order)

5.1. qc16.bexti

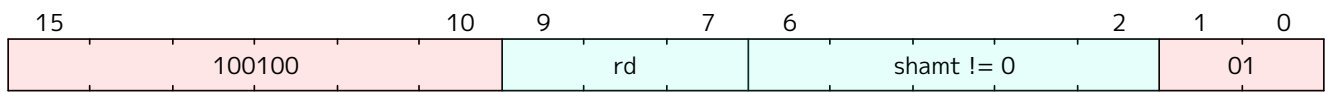
Synopsis

Single-Bit extract (Immediate)

Mnemonic

```
qc16.bexti rd, shamt
```

Encoding



Description

This instruction returns a single bit extracted from rd. The index is read from the lower log2(XLEN) bits of shamt.

Decode Variables

```
Bits<5> shamt = $encoding[6:2];
Bits<3> rd = $encoding[9:7];
```

Operation

```
XReg index = shamt & (xlen() - 1);
XReg reg = rd + 8;
X[reg] = (X[reg] >> index) & 1;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.2. qc16.bseti

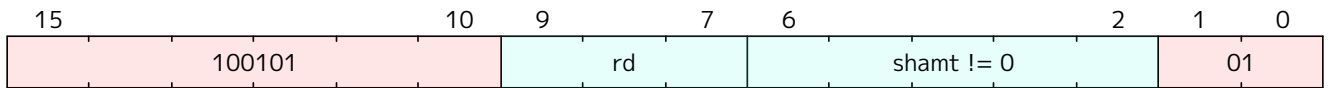
Synopsis

Single-Bit set (Immediate)

Mnemonic

```
qc16.bseti rd, shamt
```

Encoding



Description

This instruction returns `rd` with a single bit set at the index specified in `shamt`. The index is read from the lower $\log_2(\text{XLEN})$ bits of `shamt`.

Decode Variables

```
Bits<5> shamt = $encoding[6:2];
Bits<3> rd = $encoding[9:7];
```

Operation

```
XReg index = shamt & (xlen() - 1);
XReg reg = rd + 8;
X[reg] = X[reg] | (1 << index);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.3. qc16.clrint

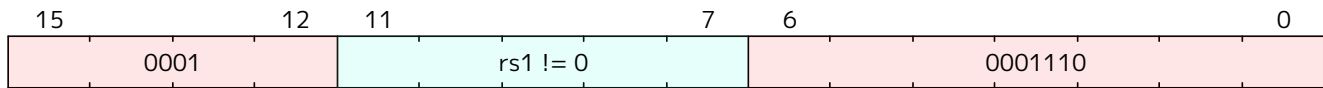
Synopsis

Clear interrupt (Register)

Mnemonic

```
qc16.clrint rs1
```

Encoding



Description

Clear interrupt, interrupt number is in rs1.

Decode Variables

```
Bits<5> rs1 = $encoding[11:7];
```

Operation

```
XReg idx = rs1 / 32;  
XReg bit = rs1 % 32;  
XReg pre_csr = CSR[mclcip0 + idx].sw_read();  
CSR[mclcip0 + idx].sw_write(pre_csr & ~(1 << bit));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.4. qc16.delay

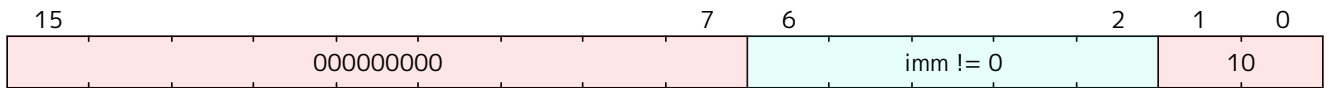
Synopsis

Delay execution

Mnemonic

```
qc16.delay imm
```

Encoding



Description

Delay execution for given imm amount of cycles.

Decode Variables

```
Bits<5> imm = $encoding[6:2];
```

Operation

```
delay(imm);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.5. qc16.di

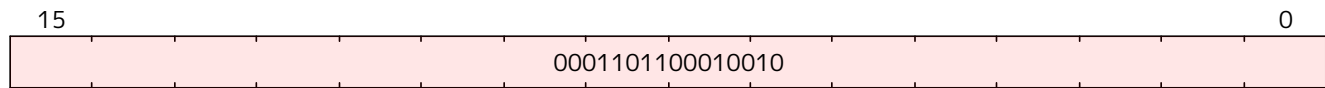
Synopsis

Disable interrupts

Mnemonic

qc16.di rd

Encoding



Description

Globally disable interrupts. Equivalent to "csrrci zero, mstatus, 8".

Decode Variables

Operation

```
XReg pre_mstatus = CSR[mstatus].sw_read();
CSR[mstatus].sw_write(pre_mstatus & ~8);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.6. qc16.dir

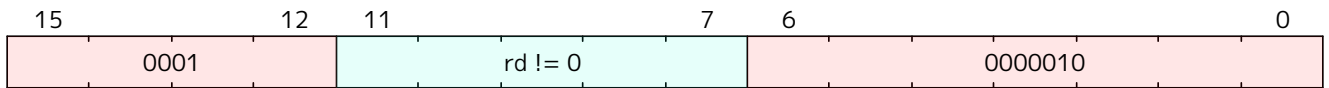
Synopsis

Disable interrupts (Register)

Mnemonic

```
qc16.dir rd
```

Encoding



Description

Globally disable interrupts, write previous value of mstatus to rd. Equivalent to "csrrci rd, mstatus, 8".

Decode Variables

```
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg pre_mstatus = CSR[mstatus].sw_read();
CSR[mstatus].sw_write(pre_mstatus & ~8);
X[rd] = pre_mstatus;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.7. qc16.ei

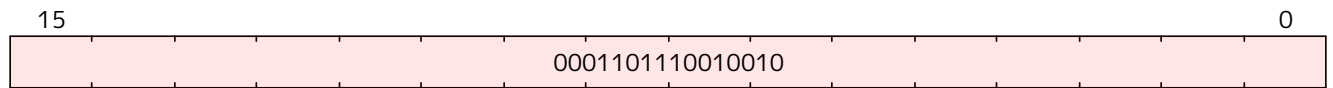
Synopsis

Enable interrupts

Mnemonic

qc16.ei rs1

Encoding



Description

Globally enable interrupts. Equivalent to "csrrsi zero, mstatus, 8".

Decode Variables

Operation

```
XReg pre_mstatus = CSR[mstatus].sw_read();
CSR[mstatus].sw_write(pre_mstatus | 8);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.8. qc16.eir

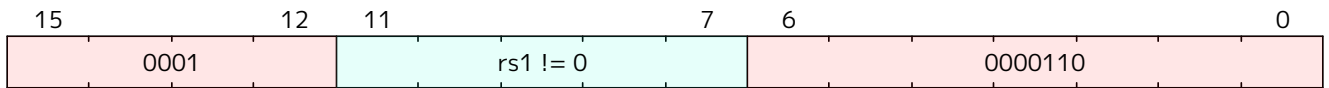
Synopsis

Restore interrupts (Register)

Mnemonic

```
qc16.eir rs1
```

Encoding



Description

Globally restore interrupts, write rs1 to mstatus. Equivalent to "csrrs zero, mstatus, `rs1`".

Decode Variables

```
Bits<5> rs1 = $encoding[11:7];
```

Operation

```
CSR[mstatus].sw_write(X[rs1]);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.9. qc16.extu

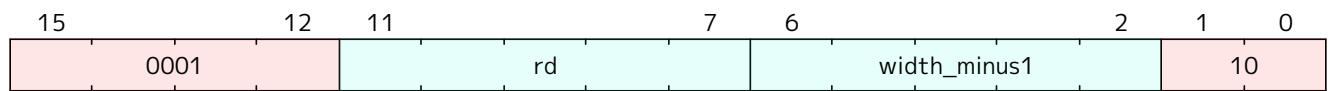
Synopsis

Extract bits unsigned

Mnemonic

```
qc16.extu rd, width
```

Encoding



Description

Extract a subset of bits from `rd` starting from LSB. The width of the subset is determined by `(width_minus1[4:0] + 1)` (1..32).

Decode Variables

```
Bits<5> width_minus1 = $encoding[6:2];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = width_minus1 + 1;
X[rd] = (X[rd] >> 0) & ((1 << width) - 1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.10. qc16.mienter.nest

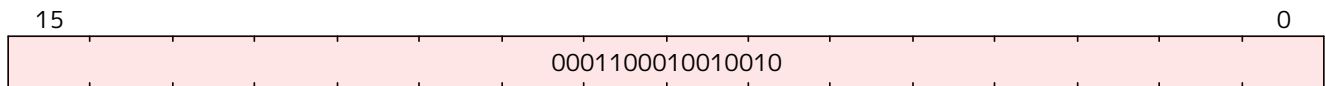
Synopsis

Machine mode interrupt enter

Mnemonic

qc16.mienter.nest

Encoding



Description

Machine mode interrupt enter, interrupt nesting is enabled. Interrupt frame is saved in the stack. Interrupts are enabled.

Decode Variables

Operation

```

XReg virtual_address = X[2];
XReg mepc_val = CSR[mepc].sw_read();
XReg mnepc_val = CSR[mnepc].sw_read();
XReg mcause_val = CSR[mcause].sw_read();
XReg flags_val = CSR[flags].sw_read();
if (CSR[mcause].NMI != 1'b1) {
    write_memory<32>(virtual_address - 4, mepc_val);
} else {
    write_memory<32>(virtual_address - 4, mnepc_val);
}
write_memory<32>(virtual_address - 8, X[8][31:0]);
write_memory<32>(virtual_address - 12, mcause_val);
write_memory<32>(virtual_address - 16, X[1][31:0]);
write_memory<32>(virtual_address - 20, flags_val);
write_memory<32>(virtual_address - 24, X[5][31:0]);
write_memory<32>(virtual_address - 28, X[6][31:0]);
write_memory<32>(virtual_address - 32, X[7][31:0]);
write_memory<32>(virtual_address - 36, X[10][31:0]);
write_memory<32>(virtual_address - 40, X[11][31:0]);
write_memory<32>(virtual_address - 44, X[12][31:0]);
write_memory<32>(virtual_address - 48, X[13][31:0]);
write_memory<32>(virtual_address - 52, X[14][31:0]);
write_memory<32>(virtual_address - 56, X[15][31:0]);
write_memory<32>(virtual_address - 60, X[16][31:0]);
write_memory<32>(virtual_address - 64, X[17][31:0]);
write_memory<32>(virtual_address - 68, X[28][31:0]);

```

```
write_memory<32>(virtual_address - 72, X[29][31:0]);
write_memory<32>(virtual_address - 76, X[30][31:0]);
write_memory<32>(virtual_address - 80, X[31][31:0]);
X[2] = X[2] - 96;
CSR[mstatus].MIE = 1'b1;
```

Included in

Extension	Minimum version
Xqciu	>= 0

DRAFT

5.11. qc16.mienter

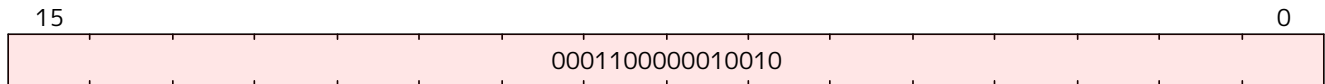
Synopsis

Machine mode interrupt enter

Mnemonic

qc16.mienter

Encoding



Description

Machine mode interrupt enter, interrupt nesting is disabled. Interrupt frame is saved in the stack. Interrupts are disabled.

Decode Variables

Operation

```
XReg virtual_address = X[2];
XReg mepc_val = CSR[mepc].sw_read();
XReg mnepc_val = CSR[mnepc].sw_read();
XReg mcause_val = CSR[mcause].sw_read();
XReg flags_val = CSR[flags].sw_read();
if (CSR[mcause].NMI != 1'b1) {
    write_memory<32>(virtual_address - 4, mepc_val);
} else {
    write_memory<32>(virtual_address - 4, mnepc_val);
}
write_memory<32>(virtual_address - 8, X[8][31:0]);
write_memory<32>(virtual_address - 12, mcause_val);
write_memory<32>(virtual_address - 16, X[1][31:0]);
write_memory<32>(virtual_address - 20, flags_val);
write_memory<32>(virtual_address - 24, X[5][31:0]);
write_memory<32>(virtual_address - 28, X[6][31:0]);
write_memory<32>(virtual_address - 32, X[7][31:0]);
write_memory<32>(virtual_address - 36, X[10][31:0]);
write_memory<32>(virtual_address - 40, X[11][31:0]);
write_memory<32>(virtual_address - 44, X[12][31:0]);
write_memory<32>(virtual_address - 48, X[13][31:0]);
write_memory<32>(virtual_address - 52, X[14][31:0]);
write_memory<32>(virtual_address - 56, X[15][31:0]);
write_memory<32>(virtual_address - 60, X[16][31:0]);
write_memory<32>(virtual_address - 64, X[17][31:0]);
write_memory<32>(virtual_address - 68, X[28][31:0]);
```

```
write_memory<32>(virtual_address - 72, X[29][31:0]);  
write_memory<32>(virtual_address - 76, X[30][31:0]);  
write_memory<32>(virtual_address - 80, X[31][31:0]);  
X[2] = X[2] - 96;
```

Included in

Extension	Minimum version
Xqciu	>= 0

DRAFT

5.12. qc16.mileaveret

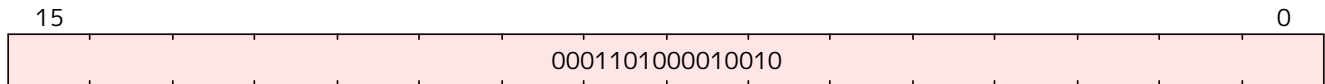
Synopsis

Machine mode interrupt exit

Mnemonic

qc16.mileaveret

Encoding



Description

Machine mode interrupt exit. Interrupt frame is restored from the stack.

Decode Variables

Operation

```

XReg virtual_address = X[2] + 96;
XReg prev_retpc = read_memory<32>(virtual_address - 4);
XReg curr_retpc = CSR[mcause].NMI ? CSR[mnepc].sw_read() : CSR[mepc].sw_read();
if (CSR[mcause].NMI != 1'b1) {
    CSR[mepc].sw_write(prev_retpc);
} else {
    CSR[mnepc].sw_write(prev_retpc);
}
X[8] = read_memory<32>(virtual_address - 8);
CSR[mcause].sw_write(read_memory<32>(virtual_address - 12));
X[1] = read_memory<32>(virtual_address - 16);
CSR[flags].sw_write(read_memory<32>(virtual_address - 20));
X[5] = read_memory<32>(virtual_address - 24);
X[6] = read_memory<32>(virtual_address - 28);
X[7] = read_memory<32>(virtual_address - 32);
X[10] = read_memory<32>(virtual_address - 36);
X[11] = read_memory<32>(virtual_address - 40);
X[12] = read_memory<32>(virtual_address - 44);
X[13] = read_memory<32>(virtual_address - 48);
X[14] = read_memory<32>(virtual_address - 52);
X[15] = read_memory<32>(virtual_address - 56);
X[16] = read_memory<32>(virtual_address - 60);
X[17] = read_memory<32>(virtual_address - 64);
X[28] = read_memory<32>(virtual_address - 68);
X[29] = read_memory<32>(virtual_address - 72);
X[30] = read_memory<32>(virtual_address - 76);
X[31] = read_memory<32>(virtual_address - 80);

```

```
X[2] = X[2] + 96;  
PC = curr_retpc;
```

Included in

Extension	Minimum version
Xqciu	>= 0

DRAFT

5.13. qc16.mnret

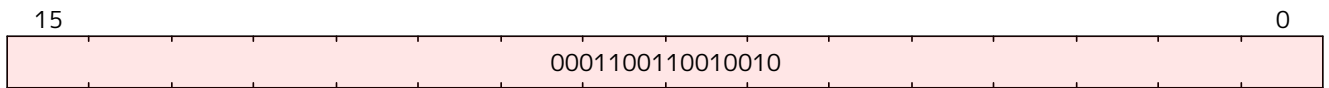
Synopsis

Machine NMI Return

Mnemonic

qc16.mnret

Encoding



Description

Returns from an NMI in M-mode.

Decode Variables

Operation

```
CSR[mncause].MIE = CSR[mncause].MPIE;
CSR[mncause].MPIE = 1;
if (CSR[mncause].MPP == 2'b00) {
    set_mode(PrivilegeMode::U);
} else if (CSR[mncause].MPP == 2'b01) {
    set_mode(PrivilegeMode::S);
} else if (CSR[mncause].MPP == 2'b11) {
    set_mode(PrivilegeMode::M);
}
CSR[mncause].MPP = implemented?(ExtensionName::U) ? 2'b00 : 2'b11;
CSR[mcause].NMI = 0;
PC = CSR[mnepc].sw_read();
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.14. qc16.mpyaddi

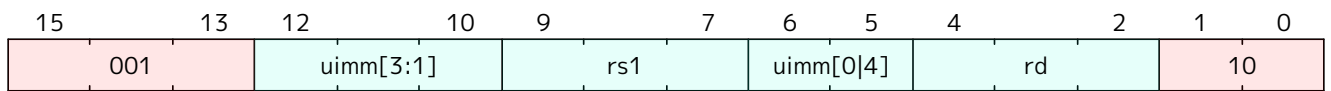
Synopsis

Multiply and accumulate (Immediate)

Mnemonic

```
qc16.mpyaddi rd, rs1, uimm
```

Encoding



Description

Increments rd by the multiplication of rs1 and an unsigned immediate

Decode Variables

```
Bits<5> uimm = { $encoding[5], $encoding[12:10], $encoding[6] };
Bits<3> rs1 = $encoding[9:7];
Bits<3> rd = $encoding[4:2];
```

Operation

```
XReg orig_val = X[rd + 8];
X[rd + 8] = orig_val + (X[rs1 + 8] * uimm);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.15. qc16.mret

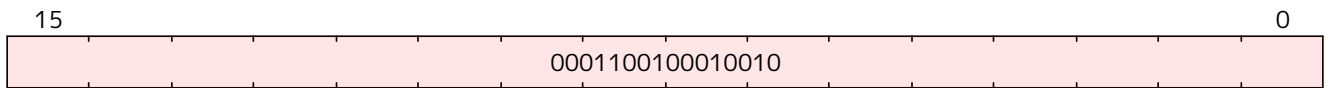
Synopsis

Machine Exception Return

Mnemonic

qc16.mret

Encoding



Description

Returns from an exception in M-mode.

Decode Variables

Operation

```
if (implemented?(ExtensionName::S) && CSR[mstatus].MPP != 2'b11) {
    CSR[mstatus].MPRV = 0;
}
CSR[mstatus].MIE = CSR[mstatus].MPIE;
CSR[mstatus].MPIE = 1;
if (CSR[mstatus].MPP == 2'b00) {
    set_mode(PrivilegeMode::U);
} else if (CSR[mstatus].MPP == 2'b01) {
    set_mode(PrivilegeMode::S);
} else if (CSR[mstatus].MPP == 2'b11) {
    set_mode(PrivilegeMode::M);
}
CSR[mstatus].MPP = implemented?(ExtensionName::U) ? 2'b00 : 2'b11;
PC = CSR[mepc].sw_read();
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.16. qc16.mveqz

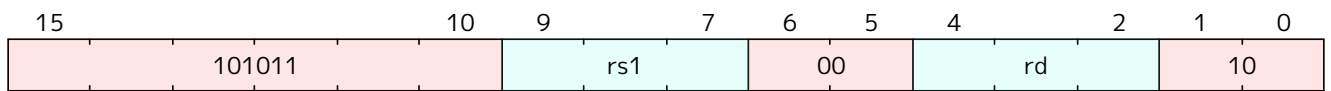
Synopsis

Conditional Move if equal to zero

Mnemonic

```
qc16.mveqz rd, rs1
```

Encoding



Description

Move rs1 to rd if rd == 0, keep rd value otherwise

Decode Variables

```
Bits<3> rs1 = $encoding[9:7];
Bits<3> rd = $encoding[4:2];
```

Operation

```
XReg orig_val = X[rd + 8];
X[rd + 8] = (orig_val == 0) ? X[rs1 + 8] : orig_val;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.17. qc16.setint

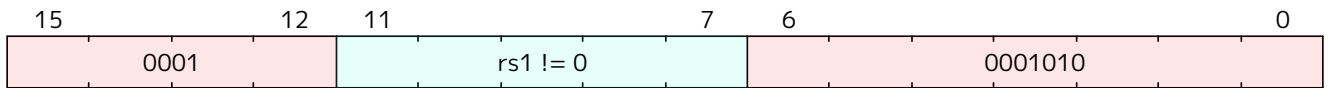
Synopsis

Set interrupt (Register)

Mnemonic

```
qc16.setint rs1
```

Encoding



Description

Set interrupt, interrupt number is in rs1.

Decode Variables

```
Bits<5> rs1 = $encoding[11:7];
```

Operation

```
XReg idx = rs1 / 32;  
XReg bit = rs1 % 32;  
XReg pre_csr = CSR[mclcip0 + idx].sw_read();  
CSR[mclcip0 + idx].sw_write(pre_csr | (1 << bit));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.18. qc32.addsat

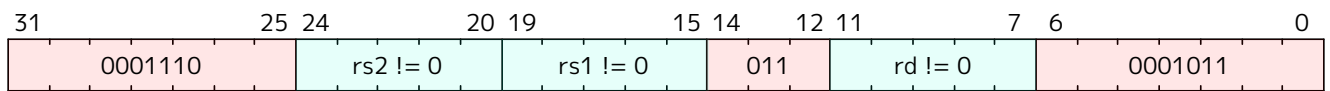
Synopsis

Saturating signed addition

Mnemonic

```
qc32.addsat rd, rs1, rs2
```

Encoding



Description

Add signed values rs1 and rs2, saturate the signed result, and write to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg sum = X[rs1] + X[rs2];
XReg most_negative_number = 1 << (xlen() - 1);
XReg most_positive_number = (1 << (xlen() - 1)) - 1;
if (X[rs1][xlen() - 1] == X[rs2][xlen() - 1]) {
    if (sum[xlen() - 1] != X[rs1][xlen() - 1]) {
        if ($signed(X[rs1]) < 0) {
            X[rd] = most_negative_number;
        } else {
            X[rd] = most_positive_number;
        }
    }
}
X[rd] = sum;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.19. qc32.addusat

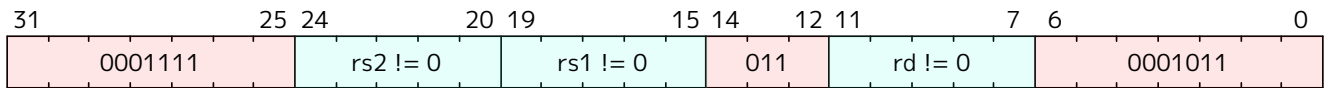
Synopsis

Saturating unsigned addition

Mnemonic

```
qc32.addusat rd, rs1, rs2
```

Encoding



Description

Add unsigned values rs1 and rs2, saturate the unsigned result, and write to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg sum = X[rs1] + X[rs2];
if ((X[rs1][xlen() - 1] == 1) || (X[rs2][xlen() - 1] == 1)) {
    if (sum[xlen() - 1] == 0) {
        X[rd] = ~{XLEN{1'b0}};
    }
}
X[rd] = sum;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.20. qc32.beqi

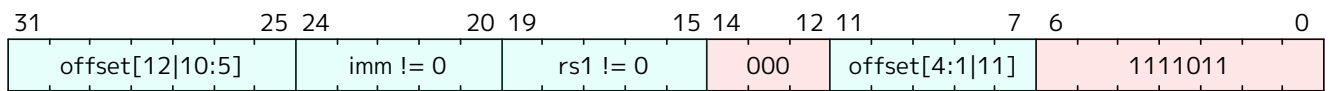
Synopsis

Branch on equal (Immediate)

Mnemonic

```
qc32.beqi rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is equal to the signed immediate

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<5> imm = $encoding[24:20];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if ($signed(X[rs1]) == $signed(imm)) {  
  jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.21. qc32.bgei

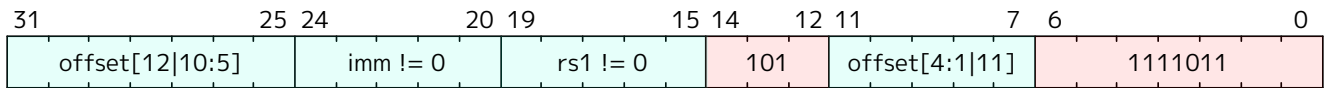
Synopsis

Branch on greater than or equal (immediate)

Mnemonic

```
qc32.bgei rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is greater than or equal to the immediate.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<5> imm = $encoding[24:20];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if ($signed(X[rs1]) >= $signed(imm)) {  
  jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.22. qc32.bgeui

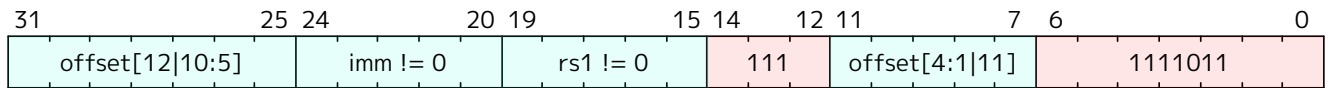
Synopsis

Branch on greater than or equal unsigned (immediate)

Mnemonic

```
qc32.bgeui rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the unsigned value in rs1 is greater than or equal to the unsigned immediate.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<5> imm = $encoding[24:20];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if (X[rs1] >= imm) {  
    jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.23. qc32.blti

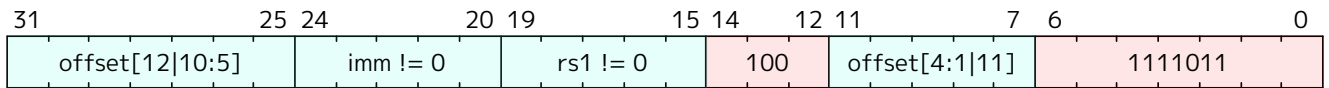
Synopsis

Branch on less than (immediate)

Mnemonic

```
qc32.blti rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is less than the immediate.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<5> imm = $encoding[24:20];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if ($signed(X[rs1]) < $signed(imm)) {  
  jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.24. qc32.bltoi

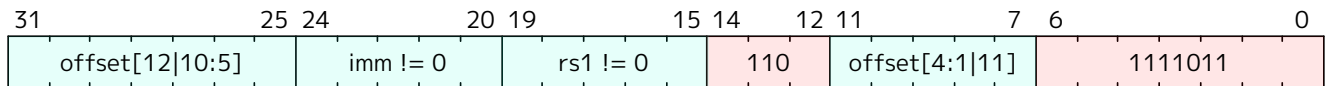
Synopsis

Branch on less than unsigned (immediate)

Mnemonic

```
qc32.bltoi rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the unsigned value in rs1 is less than the unsigned immediate.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],
$encoding[11:8], 1'd0});
Bits<5> imm = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if (X[rs1] < imm) {
    jump_halfword(PC + $signed(offset));
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.25. qc32.bnei

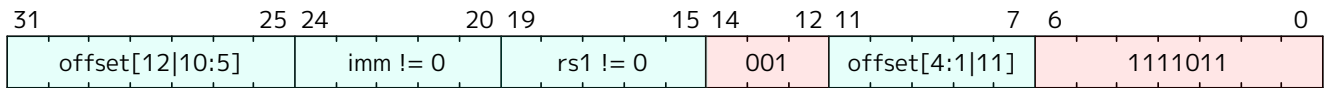
Synopsis

Branch on not equal (immediate)

Mnemonic

```
qc32.bnei rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is not equal to the signed immediate

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<5> imm = $encoding[24:20];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if ($signed(X[rs1]) != $signed(imm)) {  
  jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.26. qc32.brev32

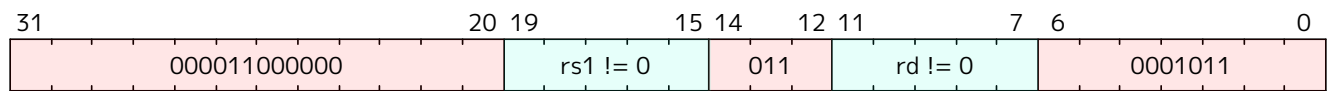
Synopsis

Reverse bit order

Mnemonic

```
qc32.brev32 rd, rs1
```

Encoding



Description

Reverses the bit order of rs1 and writes the result to rd

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg orig_val = X[rs1];
XReg tmp;
for (U32 i = 0; i < xlen(); i++) {
    tmp[xlen() - 1 - i] = orig_val[i];
}
X[rd] = tmp;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.27. qc32.clo

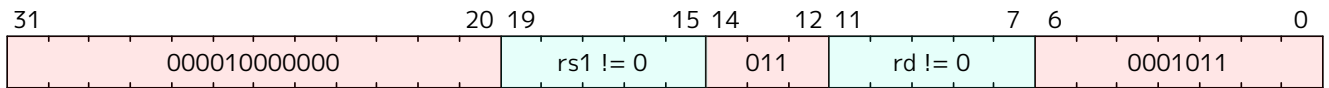
Synopsis

Count leading ones

Mnemonic

```
qc32.clo rd, rs1
```

Encoding



Description

Count the number of ones before the first zero in rs1, starting from the MSB and progressing to the LSB. Accordingly, if the input is ~0, the output is XLEN, and if the most-significant bit of the input is a 0, the output is 0. output written to the rd

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = (xlen() - 1) - $signed(highest_set_bit(~X[rs1]));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.28. qc32.clrint

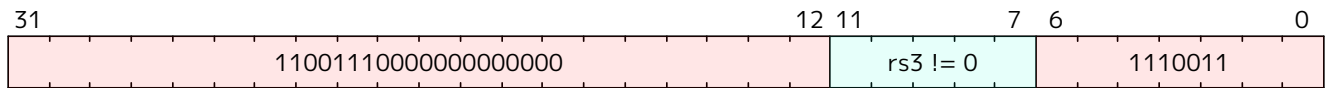
Synopsis

Clear interrupt (Register)

Mnemonic

```
qc32.clrint rs3
```

Encoding



Description

Clear interrupt, interrupt number is in rs3.

Decode Variables

```
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
XReg idx = rs3 / 32;  
XReg bit = rs3 % 32;  
XReg pre_csr = CSR[mclcip0 + idx].sw_read();  
CSR[mclcip0 + idx].sw_write(pre_csr & ~(1 << bit));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.29. qc32.clrinti

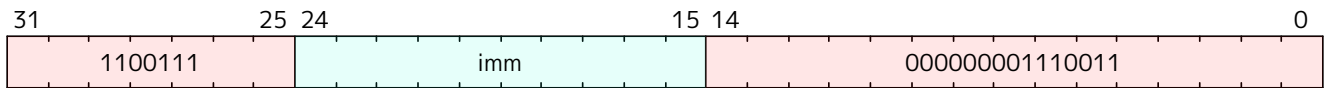
Synopsis

Clear interrupt (Immediate)

Mnemonic

```
qc32.clrinti imm
```

Encoding



Description

Clear interrupt, interrupt number is in `imm` (0 - 1023).

Decode Variables

```
Bits<10> imm = $encoding[24:15];
```

Operation

```
XReg idx = imm / 32;  
XReg bit = imm % 32;  
XReg pre_csr = CSR[mclicip0 + idx].sw_read();  
CSR[mclicip0 + idx].sw_write(pre_csr & ~(1 << bit));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.30. qc32.compress2

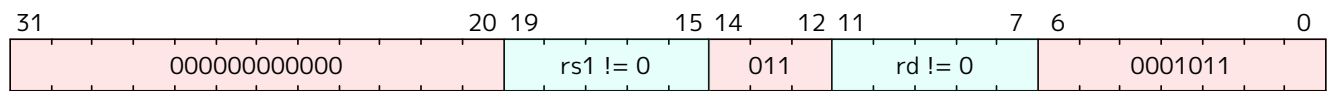
Synopsis

Bit compression (every 2nd bit)

Mnemonic

```
qc32.compress2 rd, rs1
```

Encoding



Description

Bit compression (every 2nd bit) of rs1, zero-pad bits [31:16] of the result. Write result to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg b0 = {X[rs1][14], X[rs1][12], X[rs1][10], X[rs1][8], X[rs1][6], X[rs1][4], X[rs1][2], X[rs1][0]};
XReg b1 = {X[rs1][30], X[rs1][28], X[rs1][26], X[rs1][24], X[rs1][22], X[rs1][20], X[rs1][18], X[rs1][16]};
X[rd] = {16'b0, b1[7:0], b0[7:0]};
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.31. qc32.compress3

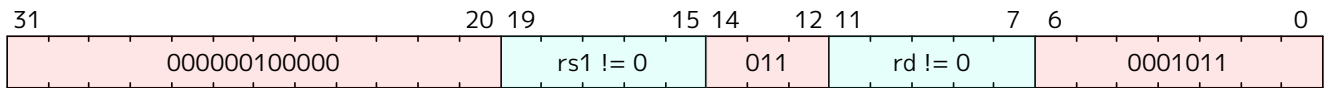
Synopsis

Bit compression (every 3rd bit)

Mnemonic

```
qc32.compress3 rd, rs1
```

Encoding



Description

Bit compression (every 3rd bit) of rs1, zero-pad bits [31:11] of the result. Write result to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg b0 = {X[rs1][21], X[rs1][18], X[rs1][15], X[rs1][12], X[rs1][9], X[rs1][6], X[rs1][3], X[rs1][0]};
XReg b1 = {5'b0, X[rs1][30], X[rs1][27], X[rs1][24]};
X[rd] = {21'b0, b1[2:0], b0[7:0]};
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.32. qc32.csrrwr

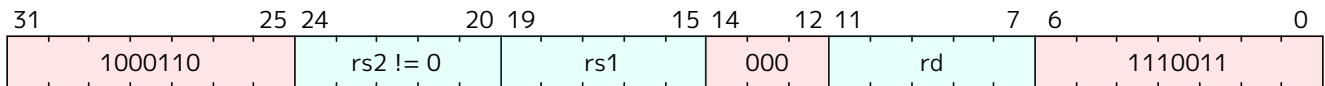
Synopsis

Atomic Read/Write CSR (Register)

Mnemonic

```
qc32.csrrwr rd, rs1, rs2
```

Encoding



Description

Atomically swap values in the CSRs and integer registers. Read the old value of the CSR, zero-extends the value to XLEN bits, and then write it to integer register `rd`. The CSR number is in `rs2` register. The initial value in `rs1` is written to the CSR. If `rd=x0`, then the instruction shall not read the CSR and shall not cause any of the side effects that might occur on a CSR read.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg csr = X[rs2];
if (rd != 0) {
    X[rd] = CSR[csr].sw_read();
}
CSR[csr].sw_write(X[rs1]);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.33. qc32.csrrwri

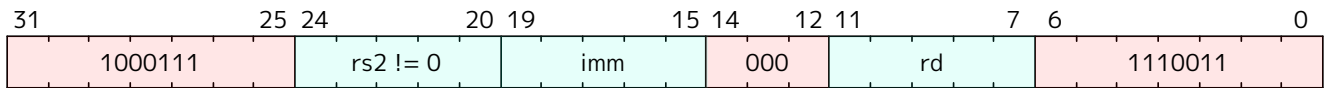
Synopsis

Atomic Read/Write CSR (Register) Immediate

Mnemonic

```
qc32.csrrwri rd, imm, rs2
```

Encoding



Description

Atomically write CSR using a 5-bit immediate `imm`, and load the previous value into `rd`. Read the old value of the CSR, zero-extends the value to `XLEN` bits, and then write it to integer register `rd`. The CSR number is in `rs2` register. The 5-bit `uimm` field is zero-extended and written to the CSR. If `rd=x0`, then the instruction shall not read the CSR and shall not cause any of the side effects that might occur on a CSR read.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> imm = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg csr = X[rs2];
if (rd != 0) {
    X[rd] = CSR[csr].sw_read();
}
CSR[csr].sw_write({{XLEN - 5{1'b0}}, imm});
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.34. qc32.cto

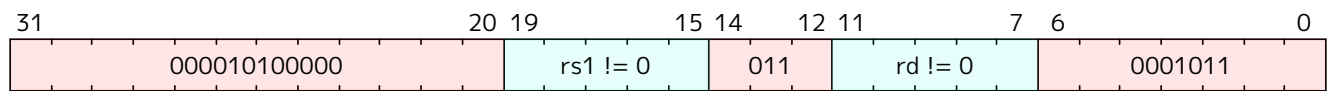
Synopsis

Count trailing ones

Mnemonic

```
qc32.cto rd, rs1
```

Encoding



Description

Count the number of ones before the first zero in rs1, starting from the LSB and progressing to the MSB. Accordingly, if the input is -0, the output is XLEN, and if the least-significant bit of the input is a 0, the output is 0. Output written to rd

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = (xlen() - 1) - $signed(lowest_set_bit(~X[rs1]));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.35. qc32.delay

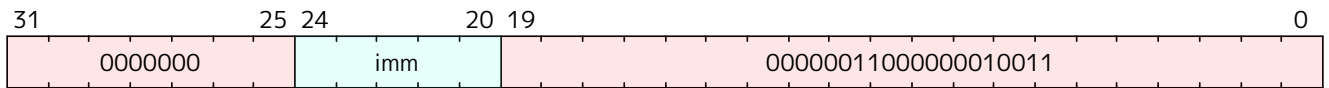
Synopsis

Delay execution

Mnemonic

```
qc32.delay imm
```

Encoding



Description

Delay execution for given imm amount of cycles.

Decode Variables

```
Bits<5> imm = $encoding[24:20];
```

Operation

```
delay(imm);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.36. qc32.expand2

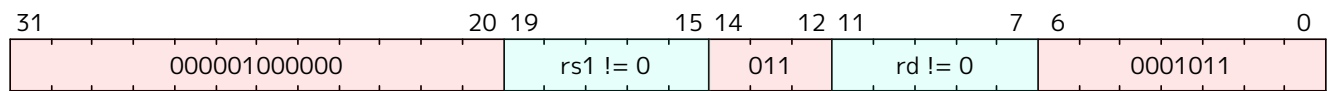
Synopsis

Bit expansion (every 2nd bit)

Mnemonic

```
qc32.expand2 rd, rs1
```

Encoding



Description

Bit expansion (every 2nd bit) of rs1, bits [31:16] of rs1 are ignored. Write result to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg b0 = {X[rs1][3], X[rs1][3], X[rs1][2], X[rs1][2], X[rs1][1], X[rs1][1], X[rs1][0], X[rs1][0]};
XReg b1 = {X[rs1][7], X[rs1][7], X[rs1][6], X[rs1][6], X[rs1][5], X[rs1][5], X[rs1][4], X[rs1][4]};
XReg b2 = {X[rs1][11], X[rs1][11], X[rs1][10], X[rs1][10], X[rs1][9], X[rs1][9], X[rs1][8], X[rs1][8]};
XReg b3 = {X[rs1][15], X[rs1][15], X[rs1][14], X[rs1][14], X[rs1][13], X[rs1][13], X[rs1][12], X[rs1][12]};
X[rd] = {b3[7:0], b2[7:0], b1[7:0], b0[7:0]};
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.37. qc32.expand3

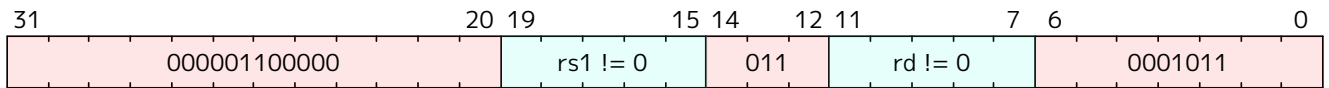
Synopsis

Bit expansion (every 3rd bit)

Mnemonic

```
qc32.expand3 rd, rs1
```

Encoding



Description

Bit expansion (every 3rd bit) of rs1, bits [31:11] of rs1 are ignored. Write result to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg b0 = {X[rs1][2], X[rs1][2], X[rs1][1], X[rs1][1], X[rs1][1], X[rs1][0], X[rs1][0], X[rs1][0]};
XReg b1 = {X[rs1][5], X[rs1][4], X[rs1][4], X[rs1][4], X[rs1][3], X[rs1][3], X[rs1][3], X[rs1][2]};
XReg b2 = {X[rs1][7], X[rs1][7], X[rs1][7], X[rs1][6], X[rs1][6], X[rs1][6], X[rs1][5], X[rs1][5]};
XReg b3 = {X[rs1][10], X[rs1][10], X[rs1][9], X[rs1][9], X[rs1][9], X[rs1][8], X[rs1][8], X[rs1][8]};
X[rd] = {b3[7:0], b2[7:0], b1[7:0], b0[7:0]};
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.38. qc32.extds

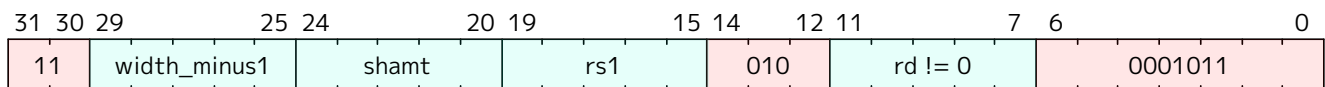
Synopsis

Extract bits from pair signed (Immediate)

Mnemonic

```
qc32.extds rd, rs1, width, shamt
```

Encoding



Description

Extract a subset of bits from the register pair $[rs1, rs1+1]$ into rd , and sign extend the result. The width of the subset is determined by $(width_minus1 + 1)$ (1..32), and the offset (into the pair) of the subset is determined by $shamt$.

Decode Variables

```
Bits<5> width_minus1 = $encoding[29:25];
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = width_minus1 + 1;
X[rd] = sext((pair >> shamt) & ((1 << width) - 1), width_minus1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.39. qc32.extdspr

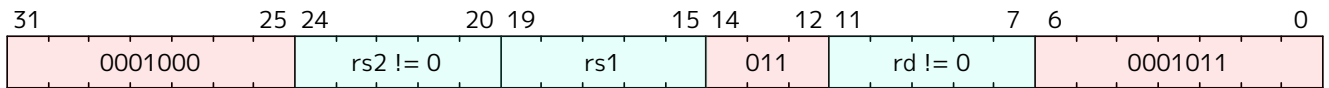
Synopsis

Extract bits from pair signed, packed descriptor (Register)

Mnemonic

```
qc32.extdspr rd, rs1, rs2
```

Encoding



Description

Extract a subset of bits from the register pair [rs1, rs1+1] into rd, and sign extend the result. The width of the subset is determined by rs2 bits [15:8] + 1 (1..32), and the offset of the subset is determined by rs2 bits [7:0].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = X[rs2][15:8] + 1;
XReg shamt = X[rs2][7:0];
X[rd] = sext((pair >> shamt) & 1 << width) - 1, (width - 1;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.40. qc32.extdsprh

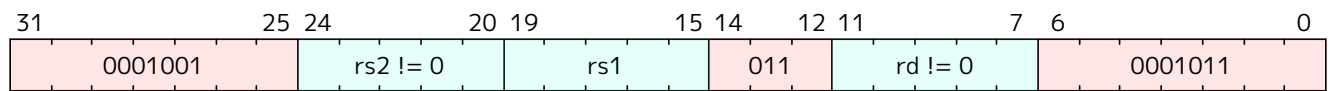
Synopsis

Extract bits from pair signed, packed descriptor high part (Register)

Mnemonic

```
qc32.extdsprh rd, rs1, rs2
```

Encoding



Description

Extract a subset of bits from the register pair [rs1, rs1+1] into rd, and sign extend the result. The width of the subset is determined by rs2 bits [31:24] + 1 (1..32), and the offset of the subset is determined by rs2 bits [23:16].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = X[rs2][31:24] + 1;
XReg shamt = X[rs2][23:16];
X[rd] = sext((pair >> shamt) & 1 << width) - 1, (width - 1;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.41. qc32.extdsr

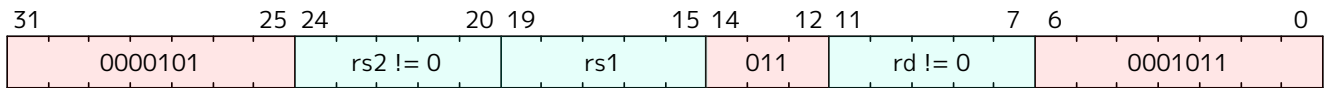
Synopsis

Extract bits from pair signed (Register)

Mnemonic

```
qc32.extdsr rd, rs1, rs2
```

Encoding



Description

Extract a subset of bits from the register pair [rs1, rs1+1] into rd, and sign extend the result. The width of the subset is determined by rs2 bits [31:16] + 1 (1..32), and the offset of the subset is determined by rs2 bits [15:0].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = X[rs2][31:16] + 1;
XReg shamt = X[rs2][15:0];
X[rd] = sext((pair >> shamt) & 1 << width) - 1, (width - 1;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.42. qc32.extdu

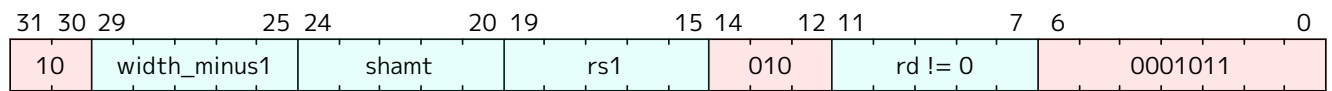
Synopsis

Extract bits from pair unsigned (Immediate)

Mnemonic

```
qc32.extdu rd, rs1, width, shamt
```

Encoding



Description

Extract a subset of bits from the register pair [rs1, rs1+1] into rd. The width of the subset is determined by (width_minus1 + 1) (1..32), and the offset (into the pair) of the subset is determined by shamt.

Decode Variables

```
Bits<5> width_minus1 = $encoding[29:25];
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = width_minus1 + 1;
X[rd] = (pair >> shamt) & ((1 << width) - 1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.43. qc32.extdupr

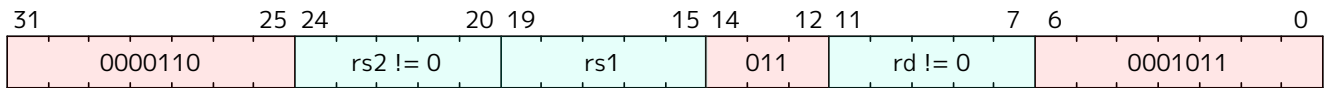
Synopsis

Extract bits from pair unsigned, packed descriptor (Register)

Mnemonic

```
qc32.extdupr rd, rs1, rs2
```

Encoding



Description

Extract a subset of bits from the register pair [rs1, rs1+1] into rd. The width of the subset is determined by rs2 bits [15:8] + 1 (1..32), and the offset of the subset is determined by rs2 bits [7:0].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = X[rs2][15:8] + 1;
XReg shamt = X[rs2][7:0];
X[rd] = (pair >> shamt) & ((1 << width) - 1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.44. qc32.extduprh

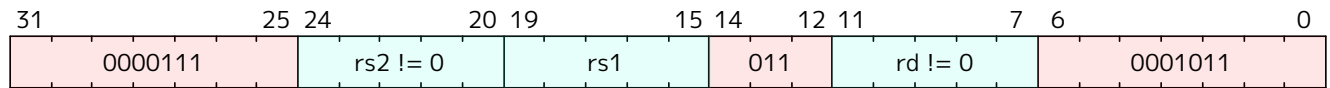
Synopsis

Extract bits from pair unsigned, packed descriptor high part (Register)

Mnemonic

```
qc32.extduprh rd, rs1, rs2
```

Encoding



Description

Extract a subset of bits from the register pair [rs1, rs1+1] into rd. The width of the subset is determined by rs2 bits [31:24] + 1 (1..32), and the offset of the subset is determined by rs2 bits [23:16].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = X[rs2][31:24] + 1;
XReg shamt = X[rs2][23:16];
X[rd] = (pair >> shamt) & ((1 << width) - 1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.45. qc32.extdur

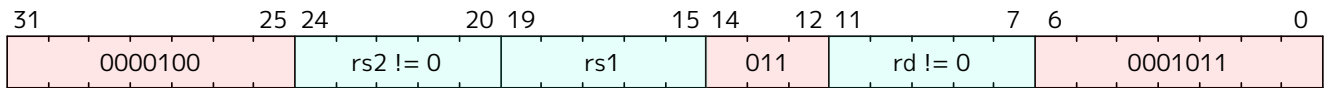
Synopsis

Extract bits from pair unsigned (Register)

Mnemonic

```
qc32.extdur rd, rs1, rs2
```

Encoding



Description

Extract a subset of bits from the register pair [rs1, rs1+1] into rd. The width of the subset is determined by rs2 bits [31:16] + 1 (1..32), and the offset of the subset is determined by rs2 bits [15:0].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> pair = {X[rs1 + 1], X[rs1]};
XReg width = X[rs2][31:16] + 1;
XReg shamt = X[rs2][15:0];
X[rd] = (pair >> shamt) & ((1 << width) - 1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.46. qc32.exts

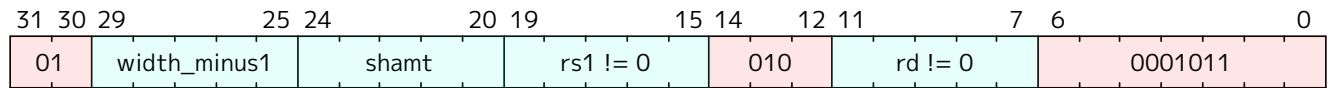
Synopsis

Extract bits signed

Mnemonic

```
qc32.exts rd, rs1, width, shamt
```

Encoding



Description

Extract a subset of bits from rs1 into rd, and sign-extend the result. The width of the subset is determined by (width_minus1 + 1) (1..32), and the offset of the subset is determined by shamt.

Decode Variables

```
Bits<5> width_minus1 = $encoding[29:25];
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = width_minus1 + 1;
XReg unsigned_extraction = (X[rs1] >> shamt) & ((1 << width) - 1);
X[rd] = sext(unsigned_extraction, width_minus1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.47. qc32.extu

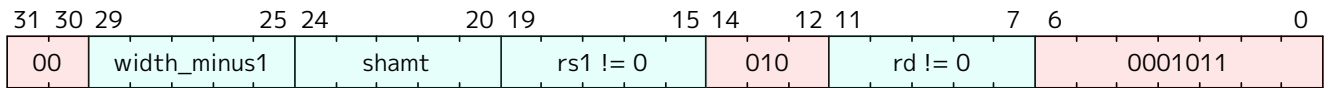
Synopsis

Extract bits unsigned

Mnemonic

```
qc32.extu rd, rs1, width, shamt
```

Encoding



Description

Extract a subset of bits from rs1 into rd. The width of the subset is determined by (width_minus1 + 1) (1..32), and the offset of the subset is determined by shamt.

Decode Variables

```
Bits<5> width_minus1 = $encoding[29:25];
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = width_minus1 + 1;
X[rd] = (X[rs1] >> shamt) & ((1 << width) - 1);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.48. qc32.insb

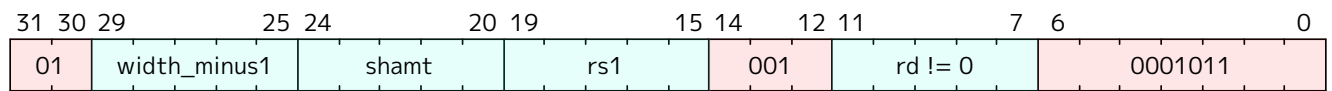
Synopsis

Insert bits (Register)

Mnemonic

```
qc32.insb rd, rs1, width, shamt
```

Encoding



Description

Insertion of a subset of bits from rs1 into rd. The width of the subset is determined by (width_minus1 + 1) (1..32), and the offset of the subset is determined by shamt.

Decode Variables

```
Bits<5> width_minus1 = $encoding[29:25];
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = width_minus1 + 1;
XReg mask = ((1 << width) - 1) << shamt;
XReg orig_val = X[rd];
X[rd] = (orig_val & ~mask) | ((X[rs1] << shamt) & mask);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.49. qc32.insbh

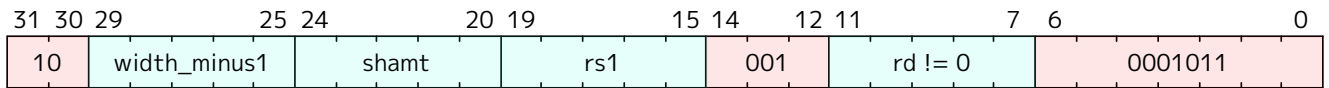
Synopsis

Insert bits in 64-bit higher part (Register)

Mnemonic

```
qc32.insbh rd, rs1, width, shamt
```

Encoding



Description

Insertion of a subset of bits from rs1 into rd. Instruction intended for insertion bits into bitfield within 64-bits, when bitfield crosses 32-bit boundary. Lower part of 64-bit destination is inserted using QC32.INSB, higher part using QC32.INSBH. The width of the subset is determined by (width_minus1 + 1) (1..32), and the offset of the subset is determined by shamt. In case when width + offset ≤ 32, the destination register is left unchanged.

Decode Variables

```
Bits<5> width_minus1 = $encoding[29:25];
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = width_minus1 + 1;
if (width + shamt > 32) {
    XReg mask = 1 << (width + shamt - 32 - 1);
    XReg orig_val = X[rd];
    X[rd] = (orig_val & ~mask) | X[rs1] & mask;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.50. qc32.insbhr

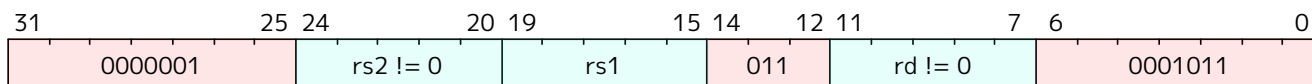
Synopsis

Insert bits in 64-bit higher part (Register)

Mnemonic

```
qc32.insbhr rd, rs1, rs2
```

Encoding



Description

Insertion of a subset of bits from rs1 into rd. Instruction intended for insertion bits into bitfield within 64-bits, when bitfield crosses 32-bit boundary. Lower part of 64-bit destination is inserted using QC32.INSB, higher part using QC32.INSBH. The width of the subset is determined by rs2 bits [31:16] + 1 (1..32), and the offset of the subset is determined by rs2 bits [15:0]. In case when width + offset $\leq$ 32, the destination register is left unchanged.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = X[rs2][31:16] + 1;
XReg shamt = X[rs2][15:0];
if (width + shamt > 32) {
    XReg mask = 1 << (width + shamt - 32 - 1);
    XReg orig_val = X[rd];
    X[rd] = (orig_val & ~mask) | X[rs1] & mask;
}
```

Included in

Extension	Minimum version
Xqciu	≥ 0

5.51. qc32.insbi

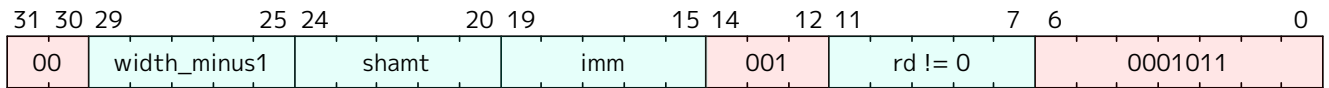
Synopsis

Insert bits (Immediate)

Mnemonic

```
qc32.insbi rd, imm, width, shamt
```

Encoding



Description

Insertion of a subset of bits of an imm into rd. The width of the subset is determined by (width_minus1 + 1) (1..32), and the offset of the subset is determined by shamt.

Decode Variables

```
Bits<5> width_minus1 = $encoding[29:25];
Bits<5> shamt = $encoding[24:20];
Bits<5> imm = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = width_minus1 + 1;
XReg mask = ((1 << width) - 1) << shamt;
XReg orig_val = X[rd];
X[rd] = (orig_val & ~mask) | ((imm << shamt) & mask);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.52. qc32.insbpr

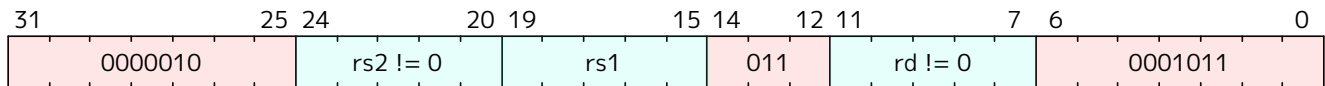
Synopsis

Insert bits, packed descriptor (Register)

Mnemonic

```
qc32.insbpr rd, rs1, rs2
```

Encoding



Description

Insertion of a subset of bits from *rs1* into *rd*. The width of the subset is determined by *rs2* bits [15:8] + 1 (1..32), and the offset of the subset is determined by *rs2* bits [7:0].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = X[rs2][15:8] + 1;
XReg shamt = X[rs2][7:0];
XReg mask = ((1 << width) - 1) << shamt;
XReg orig_val = X[rd];
X[rd] = (orig_val & ~mask) | ((X[rs1] << shamt) & mask);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.53. qc32.insbprh

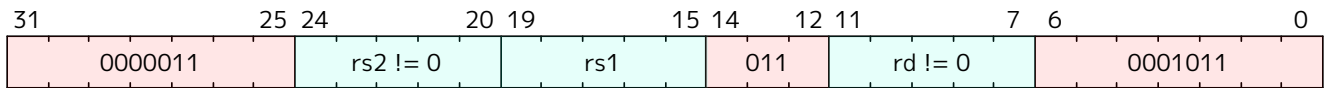
Synopsis

Insert bits, packed descriptor high part (Register)

Mnemonic

```
qc32.insbprh rd, rs1, rs2
```

Encoding



Description

Insertion of a subset of bits from *rs1* into *rd*. The width of the subset is determined by *rs2* bits [31:24] + 1 (1..32), and the offset of the subset is determined by *rs2* bits [23:15].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = X[rs2][31:24] + 1;
XReg shamt = X[rs2][23:16];
XReg mask = ((1 << width) - 1) << shamt;
XReg orig_val = X[rd];
X[rd] = (orig_val & ~mask) | ((X[rs1] << shamt) & mask);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.54. qc32.insbr

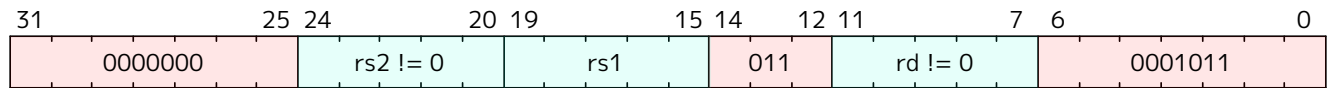
Synopsis

Insert bits (Register)

Mnemonic

```
qc32.insbr rd, rs1, rs2
```

Encoding



Description

Insertion of a subset of bits from rs1 into rd. The width of the subset is determined by rs2 bits [31:16] + 1 (1..32), and the offset of the subset is determined by rs2 bits [15:0].

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = X[rs2][31:16] + 1;
XReg shamt = X[rs2][15:0];
XReg mask = ((1 << width) - 1) << shamt;
XReg orig_val = X[rd];
X[rd] = (orig_val & ~mask) | ((X[rs1] << shamt) & mask);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.55. qc32.insbri

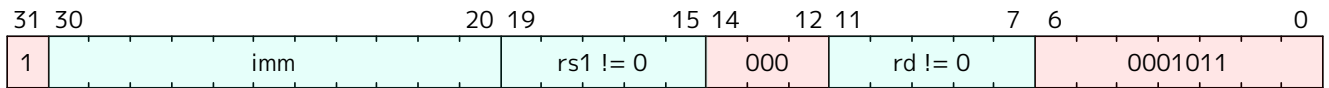
Synopsis

Insert bits (Immediate)

Mnemonic

```
qc32.insbri rd, rs1, imm
```

Encoding



Description

Insertion of a subset of bits of an imm into rd. The width of the subset is determined by rs1 bits [31:16] + 1, and the offset of the subset is determined by rs1 bits [15:0].

Decode Variables

```
Bits<11> imm = $encoding[30:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg width = X[rs1][31:16] + 1;
XReg shamt = X[rs1][15:0];
XReg mask = ((1 << width) - 1) << shamt;
XReg orig_val = X[rd];
X[rd] = (orig_val & ~mask) | ((imm << shamt) & mask);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.56. qc32.li

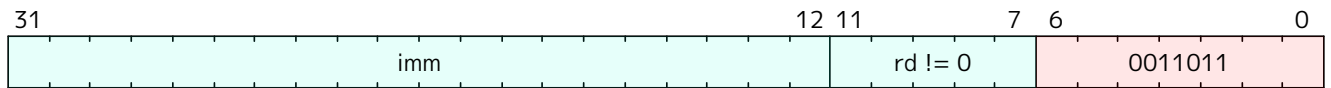
Synopsis

Load immediate large

Mnemonic

```
qc32.li rd, imm
```

Encoding



Description

Loads the 20-bit immediate `imm` into `rd`.

Decode Variables

```
Bits<20> imm = {$encoding[31], $encoding[15:12], $encoding[30:16]};
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = sext(imm, 20);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.57. qc32.lieq

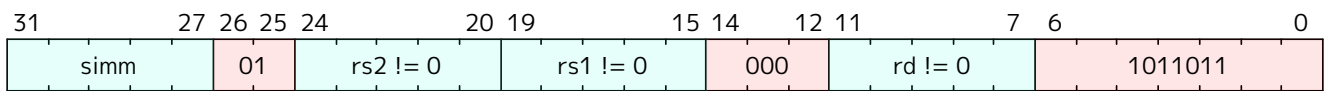
Synopsis

Conditional load immediate if equal (Register)

Mnemonic

```
qc32.lieq rd, rs1, rs2, simm
```

Encoding



Description

Move `simm` to `rd` if the value in `rs1` is equal to value `rs2`

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] == X[rs2]) {
    X[rd] = sext(simm, 20);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.58. qc32.lieqi

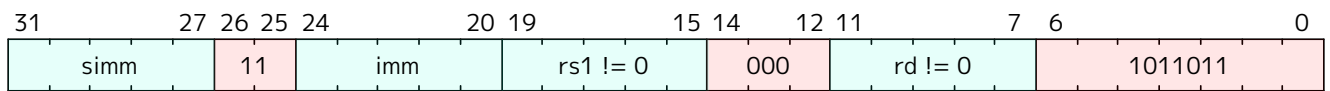
Synopsis

Conditional load immediate if equal (Immediate)

Mnemonic

```
qc32.lieqi rd, rs1, imm, simm
```

Encoding



Description

Move simm to rd if the value in rs1 is equal to value imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) == $signed(imm)) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.59. qc32.lige

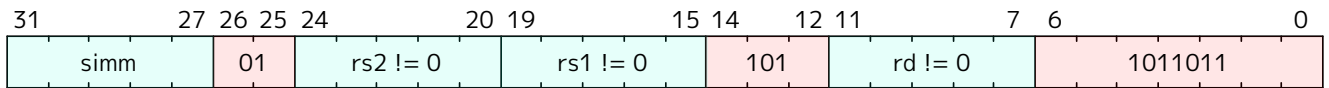
Synopsis

Conditional load immediate if great or equal than (Register)

Mnemonic

```
qc32.lige rd, rs1, rs2, simm
```

Encoding



Description

Move simm to rd if the value in rs1 is great or equal than value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) >= $signed(X[rs2])) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.60. qc32.ligei

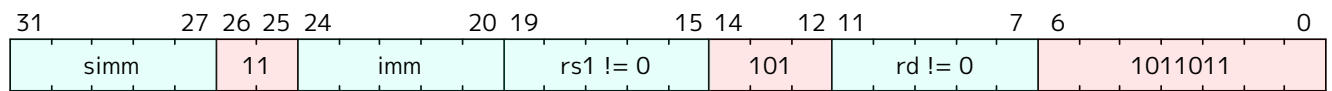
Synopsis

Conditional load immediate if great or equal than (Immediate)

Mnemonic

```
qc32.ligei rd, rs1, imm, simm
```

Encoding



Description

Move simm to rd if the value in rs1 is great or equal than value imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) >= $signed(imm)) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.61. qc32.ligeu

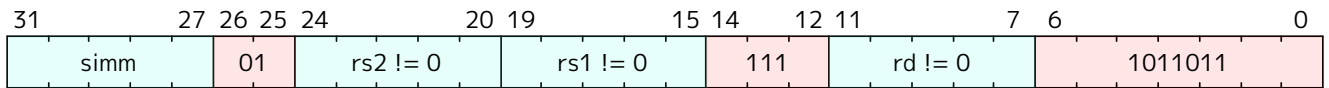
Synopsis

Conditional load immediate if great or equal than unsigned (Register)

Mnemonic

```
qc32.ligeu rd, rs1, rs2, simm
```

Encoding



Description

Move `simm` to `rd` if the unsigned value in `rs1` is great or equal than unsigned value `rs2`

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] >= X[rs2]) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.62. qc32.ligeui

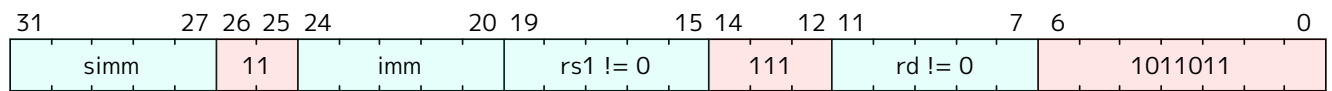
Synopsis

Conditional load immediate if great or equal than unsigned (Immediate)

Mnemonic

```
qc32.ligeui rd, rs1, imm, simm
```

Encoding



Description

Move simm to rd if the unsigned value in rs1 is great or equal than unsigned value imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] >= imm) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.63. qc32.lilt

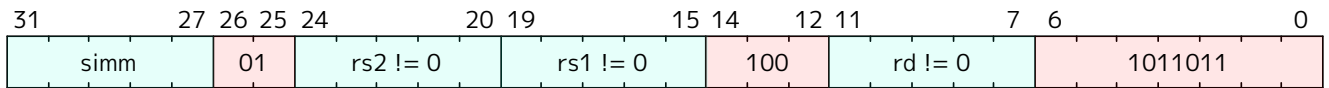
Synopsis

Conditional load immediate if less than (Register)

Mnemonic

```
qc32.lilt rd, rs1, rs2, simm
```

Encoding



Description

Move simm to rd if the value in rs1 is less than value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) < $signed(X[rs2])) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.64. qc32.liti

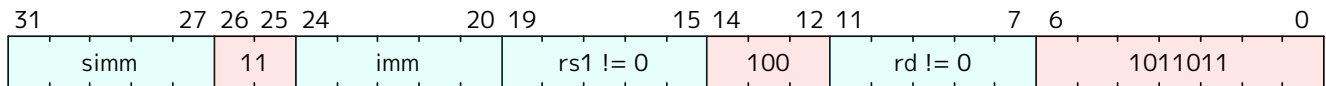
Synopsis

Conditional load immediate if less than (Immediate)

Mnemonic

```
qc32.liti rd, rs1, imm, simm
```

Encoding



Description

Move simm to rd if the value in rs1 is less than value imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) < $signed(imm)) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.65. qc32.liltu

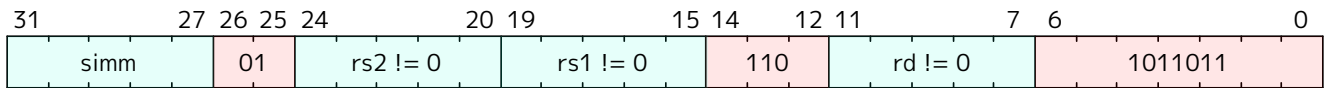
Synopsis

Conditional load immediate if less than unsigned (Register)

Mnemonic

```
qc32.liltu rd, rs1, rs2, simm
```

Encoding



Description

Move `simm` to `rd` if the unsigned value in `rs1` is less than unsigned value `rs2`

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] < X[rs2]) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.66. qc32.liltui

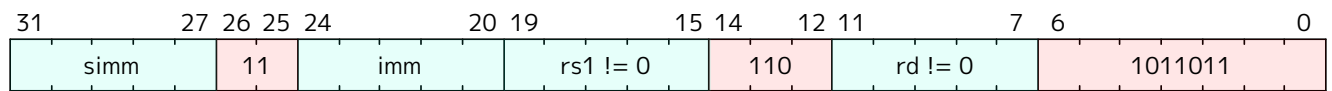
Synopsis

Conditional load immediate if less than unsigned (Immediate)

Mnemonic

```
qc32.liltui rd, rs1, imm, simm
```

Encoding



Description

Move simm to rd if the unsigned value in rs1 is less than unsigned value imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] < imm) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.67. qc32.line

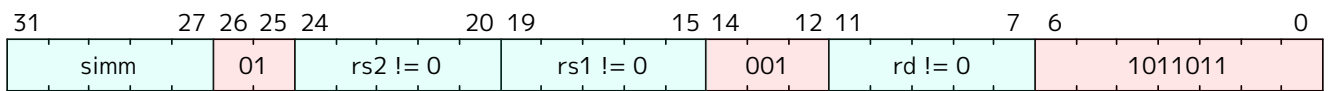
Synopsis

Conditional load immediate if not equal (Register)

Mnemonic

```
qc32.line rd, rs1, rs2, simm
```

Encoding



Description

Move `simm` to `rd` if the value in `rs1` is not equal to value `rs2`

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) != $signed(X[rs2])) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.68. qc32.linei

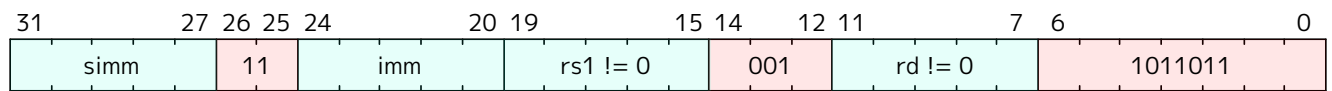
Synopsis

Conditional load immediate if not equal (Immediate)

Mnemonic

```
qc32.linei rd, rs1, imm, simm
```

Encoding



Description

Move simm to rd if the value in rs1 is not equal to value imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> simm = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) != $signed(imm)) {
    X[rd] = sext(simm, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.69. qc32.lrb

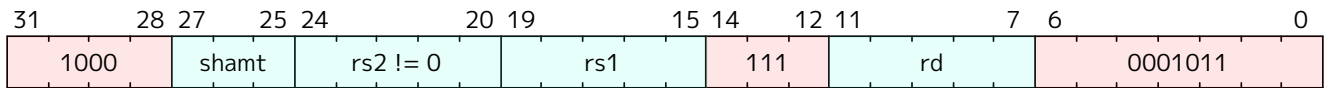
Synopsis

Load indexed byte

Mnemonic

```
qc32.lrb rd, rs1, rs2, shamt
```

Encoding



Description

Load 8 bits of data into register `rd` from an address formed by adding `rs1` to `rs2`, shifted by `shamt`. Sign extend the result.

Decode Variables

```
Bits<3> shamt = $encoding[27:25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + (X[rs2] << shamt);
X[rd] = sext(read_memory<8>(virtual_address), 8);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.70. qc32.lrbu

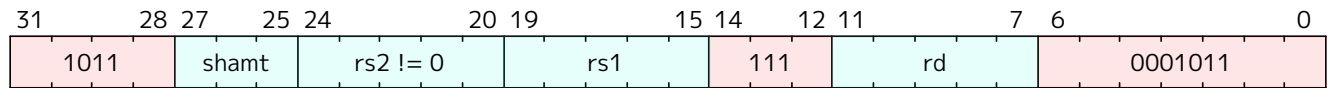
Synopsis

Load indexed unsigned byte

Mnemonic

```
qc32.lrbu rd, rs1, rs2, shamt
```

Encoding



Description

Load 8 bits of data into register rd from an address formed by adding rs1 to rs2, shifted by shamt.

Decode Variables

```
Bits<3> shamt = $encoding[27:25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + (X[rs2] << shamt);
X[rd] = read_memory<8>(virtual_address);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.71. qc32.lrh

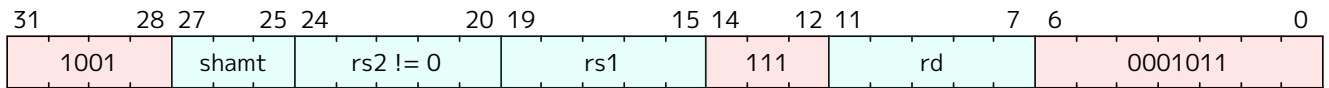
Synopsis

Load indexed halfword

Mnemonic

```
qc32.lrh rd, rs1, rs2, shamt
```

Encoding



Description

Load 16 bits of data into register rd from an address formed by adding rs1 to rs2, shifted by shamt. Sign extend the result.

Decode Variables

```
Bits<3> shamt = $encoding[27:25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + (X[rs2] << shamt);
X[rd] = sext(read_memory<16>(virtual_address), 16);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.72. qc32.lrhu

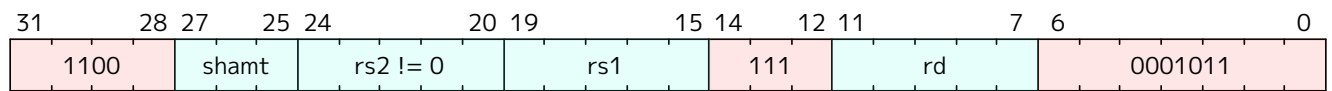
Synopsis

Load indexed unsigned halfword

Mnemonic

```
qc32.lrhu rd, rs1, rs2, shamt
```

Encoding



Description

Load 16 bits of data into register rd from an address formed by adding rs1 to rs2, shifted by shamt.

Decode Variables

```
Bits<3> shamt = $encoding[27:25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + (X[rs2] << shamt);
X[rd] = read_memory<16>(virtual_address);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.73. qc32.lrw

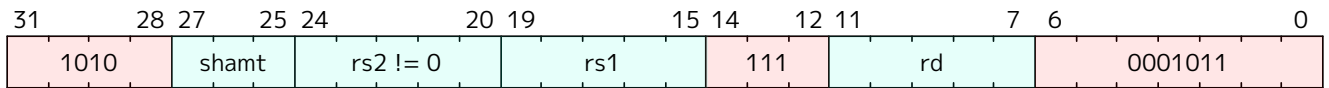
Synopsis

Load indexed word

Mnemonic

```
qc32.lrw rd, rs1, rs2, shamt
```

Encoding



Description

Load 16 bits of data into register rd from an address formed by adding rs1 to rs2, shifted by shamt.

Decode Variables

```
Bits<3> shamt = $encoding[27:25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + (X[rs2] << shamt);
X[rd] = read_memory<32>(virtual_address);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.74. qc32.lwm

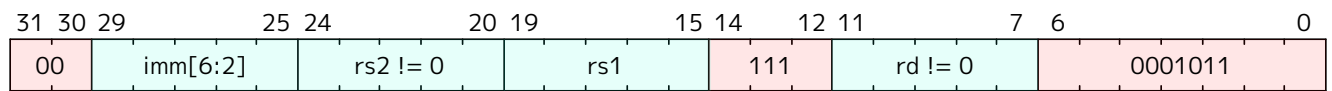
Synopsis

Load word multiple

Mnemonic

```
qc32.lwm rd, rs2, imm(rs1)
```

Encoding



Description

Loads multiple words starting from address (rs1 + imm) to registers, starting from rd. The number of words is in rs2

Decode Variables

```
Bits<7> imm = {$encoding[29:25], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg vaddr = X[rs1] + imm;
XReg num_words = X[rs2];
raise(ExceptionCode::IllegalInstruction, $encoding) if (rd + num_words) > 32;
for (U32 i = 0; i < num_words; i++) {
    X[rd + i] = read_memory<32>(vaddr);
    vaddr = vaddr + 4;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.75. qc32.lwmi

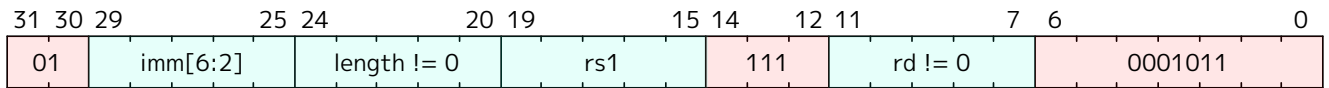
Synopsis

Load word multiple (Immediate)

Mnemonic

```
qc32.lwmi rd, length, imm(rs1)
```

Encoding



Description

Loads multiple words starting from address (rs1 + imm) to registers, starting from rd. The number of words is in the length immediate.

Decode Variables

```
Bits<7> imm = {$encoding[29:25], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> length = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg vaddr = X[rs1] + imm;
raise(ExceptionCode::IllegalInstruction, $encoding) if (rd + length) > 32;
for (U32 i = 0; i < length; i++) {
    X[rd + i] = read_memory<32>(vaddr);
    vaddr = vaddr + 4;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.76. qc32.mienter.nest

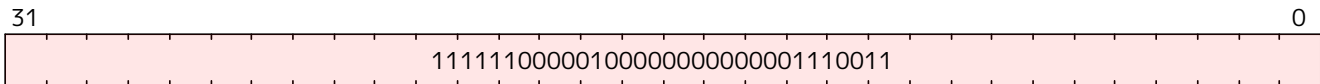
Synopsis

Machine mode interrupt enter

Mnemonic

qc32.mienter.nest

Encoding



Description

Machine mode interrupt enter, interrupt nesting is enabled. Interrupt frame is saved in the stack. Interrupts are enabled.

Decode Variables

Operation

```
XReg virtual_address = X[2];
XReg mepc_val = CSR[mepc].sw_read();
XReg mnepc_val = CSR[mnepc].sw_read();
XReg mcause_val = CSR[mcause].sw_read();
XReg flags_val = CSR[flags].sw_read();
if (CSR[mcause].NMI != 1'b1) {
    write_memory<32>(virtual_address - 4, mepc_val);
} else {
    write_memory<32>(virtual_address - 4, mnepc_val);
}
write_memory<32>(virtual_address - 8, X[8][31:0]);
write_memory<32>(virtual_address - 12, mcause_val);
write_memory<32>(virtual_address - 16, X[1][31:0]);
write_memory<32>(virtual_address - 20, flags_val);
write_memory<32>(virtual_address - 24, X[5][31:0]);
write_memory<32>(virtual_address - 28, X[6][31:0]);
write_memory<32>(virtual_address - 32, X[7][31:0]);
write_memory<32>(virtual_address - 36, X[10][31:0]);
write_memory<32>(virtual_address - 40, X[11][31:0]);
write_memory<32>(virtual_address - 44, X[12][31:0]);
write_memory<32>(virtual_address - 48, X[13][31:0]);
write_memory<32>(virtual_address - 52, X[14][31:0]);
write_memory<32>(virtual_address - 56, X[15][31:0]);
write_memory<32>(virtual_address - 60, X[16][31:0]);
write_memory<32>(virtual_address - 64, X[17][31:0]);
write_memory<32>(virtual_address - 68, X[28][31:0]);
```

```
write_memory<32>(virtual_address - 72, X[29][31:0]);
write_memory<32>(virtual_address - 76, X[30][31:0]);
write_memory<32>(virtual_address - 80, X[31][31:0]);
X[2] = X[2] - 96;
CSR[mstatus].MIE = 1'b1;
```

Included in

Extension	Minimum version
Xqciu	>= 0

DRAFT

5.77. qc32.mienter

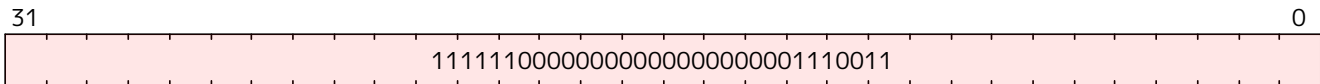
Synopsis

Machine mode interrupt enter

Mnemonic

qc32.mienter

Encoding



Description

Machine mode interrupt enter, interrupt nesting is disabled. Interrupt frame is saved in the stack. Interrupts are disabled.

Decode Variables

Operation

```

XReg virtual_address = X[2];
XReg mepc_val = CSR[mepc].sw_read();
XReg mnepc_val = CSR[mnepc].sw_read();
XReg mcause_val = CSR[mcause].sw_read();
XReg flags_val = CSR[flags].sw_read();
if (CSR[mcause].NMI != 1'b1) {
    write_memory<32>(virtual_address - 4, mepc_val);
} else {
    write_memory<32>(virtual_address - 4, mnepc_val);
}
write_memory<32>(virtual_address - 8, X[8][31:0]);
write_memory<32>(virtual_address - 12, mcause_val);
write_memory<32>(virtual_address - 16, X[1][31:0]);
write_memory<32>(virtual_address - 20, flags_val);
write_memory<32>(virtual_address - 24, X[5][31:0]);
write_memory<32>(virtual_address - 28, X[6][31:0]);
write_memory<32>(virtual_address - 32, X[7][31:0]);
write_memory<32>(virtual_address - 36, X[10][31:0]);
write_memory<32>(virtual_address - 40, X[11][31:0]);
write_memory<32>(virtual_address - 44, X[12][31:0]);
write_memory<32>(virtual_address - 48, X[13][31:0]);
write_memory<32>(virtual_address - 52, X[14][31:0]);
write_memory<32>(virtual_address - 56, X[15][31:0]);
write_memory<32>(virtual_address - 60, X[16][31:0]);
write_memory<32>(virtual_address - 64, X[17][31:0]);
write_memory<32>(virtual_address - 68, X[28][31:0]);

```

```
write_memory<32>(virtual_address - 72, X[29][31:0]);
write_memory<32>(virtual_address - 76, X[30][31:0]);
write_memory<32>(virtual_address - 80, X[31][31:0]);
X[2] = X[2] - 96;
```

Included in

Extension	Minimum version
Xqciu	>= 0

DRAFT

5.78. qc32.mileaveret

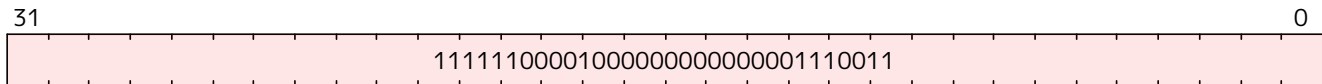
Synopsis

Machine mode interrupt exit

Mnemonic

qc32.mileaveret

Encoding



Description

Machine mode interrupt exit. Interrupt frame is restored from the stack.

Decode Variables

Operation

```
XReg virtual_address = X[2] + 96;
XReg prev_retpc = read_memory<32>(virtual_address - 4);
XReg curr_retpc = CSR[mcause].NMI ? CSR[mnepc].sw_read() : CSR[mepc].sw_read();
if (CSR[mcause].NMI != 1'b1) {
    CSR[mepc].sw_write(prev_retpc);
} else {
    CSR[mnepc].sw_write(prev_retpc);
}
X[8] = read_memory<32>(virtual_address - 8);
CSR[mcause].sw_write(read_memory<32>(virtual_address - 12));
X[1] = read_memory<32>(virtual_address - 16);
CSR[flags].sw_write(read_memory<32>(virtual_address - 20));
X[5] = read_memory<32>(virtual_address - 24);
X[6] = read_memory<32>(virtual_address - 28);
X[7] = read_memory<32>(virtual_address - 32);
X[10] = read_memory<32>(virtual_address - 36);
X[11] = read_memory<32>(virtual_address - 40);
X[12] = read_memory<32>(virtual_address - 44);
X[13] = read_memory<32>(virtual_address - 48);
X[14] = read_memory<32>(virtual_address - 52);
X[15] = read_memory<32>(virtual_address - 56);
X[16] = read_memory<32>(virtual_address - 60);
X[17] = read_memory<32>(virtual_address - 64);
X[28] = read_memory<32>(virtual_address - 68);
X[29] = read_memory<32>(virtual_address - 72);
X[30] = read_memory<32>(virtual_address - 76);
X[31] = read_memory<32>(virtual_address - 80);
```

```
X[2] = X[2] + 96;  
PC = curr_retpc;
```

Included in

Extension	Minimum version
Xqciu	>= 0

DRAFT

5.79. qc32.mnret

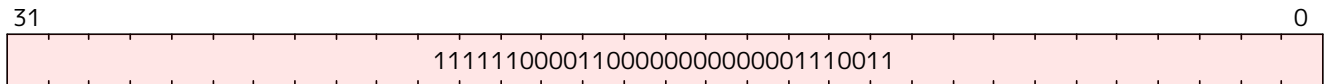
Synopsis

Machine NMI Return

Mnemonic

qc32.mnret

Encoding



Description

Returns from an NMI in M-mode.

Decode Variables

Operation

```
CSR[mncause].MIE = CSR[mncause].MPIE;
CSR[mncause].MPIE = 1;
if (CSR[mncause].MPP == 2'b00) {
    set_mode(PrivilegeMode::U);
} else if (CSR[mncause].MPP == 2'b01) {
    set_mode(PrivilegeMode::S);
} else if (CSR[mncause].MPP == 2'b11) {
    set_mode(PrivilegeMode::M);
}
CSR[mncause].MPP = implemented?(ExtensionName::U) ? 2'b00 : 2'b11;
CSR[mncause].NMI = 0;
PC = CSR[mnepc].sw_read();
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.80. qc32.mpyaddi

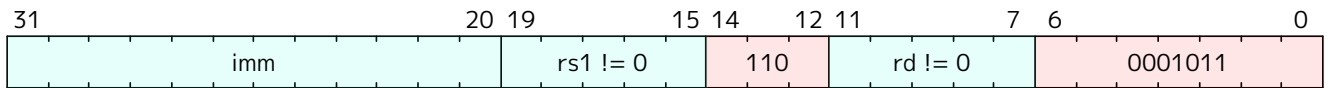
Synopsis

Multiply and accumulate (Immediate)

Mnemonic

```
qc32.mpyaddi rd, rs1, imm
```

Encoding



Description

Increments rd by the multiplication of rs1 and a signed immediate imm

Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg orig_val = X[rd];
X[rd] = orig_val + (X[rs1] * sext(imm, 12));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.81. qc32.mveq

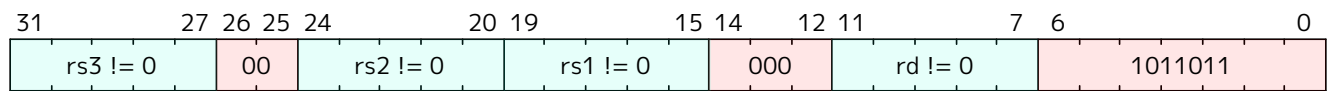
Synopsis

Conditional move if equal (Register)

Mnemonic

```
qc32.mveq rd, rs1, rs2, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is equal to value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] == X[rs2]) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.82. qc32.mveqi

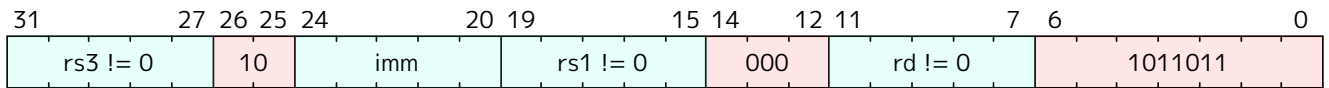
Synopsis

Conditional move if equal (Immediate)

Mnemonic

```
qc32.mveqi rd, rs1, imm, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is equal to value of imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) == $signed(imm)) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.83. qc32.mvge

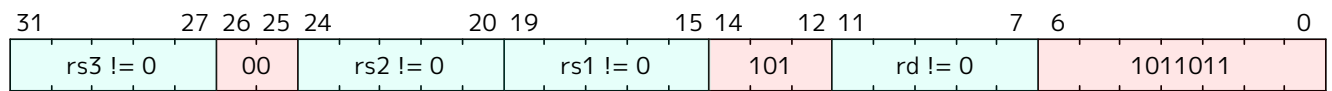
Synopsis

Conditional move if great or equal than (Register)

Mnemonic

```
qc32.mvge rd, rs1, rs2, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is great or equal than value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) >= $signed(X[rs2])) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.84. qc32.mvgei

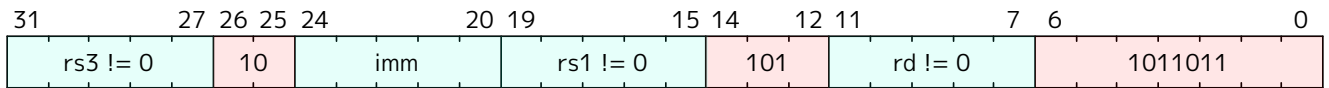
Synopsis

Conditional move if great or equal than (Immediate)

Mnemonic

```
qc32.mvgei rd, rs1, imm, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is great or equal than value of imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) >= $signed(imm)) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.85. qc32.mvgeu

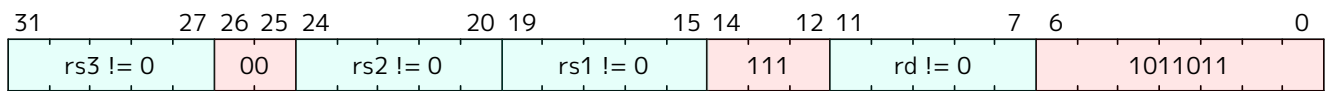
Synopsis

Conditional move if great or equal than unsigned (Register)

Mnemonic

```
qc32.mvgeu rd, rs1, rs2, rs3
```

Encoding



Description

Move rs3 to rd if the unsigned value in rs1 is great or equal than unsigned value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] >= X[rs2]) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.86. qc32.mvgeui

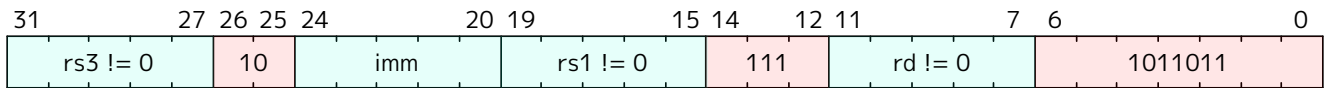
Synopsis

Conditional move if great or equal than unsigned (Immediate)

Mnemonic

```
qc32.mvgeui rd, rs1, imm, rs3
```

Encoding



Description

Move rs3 to rd if the unsigned value in rs1 is great or equal than unsigned value of imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] >= imm) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.87. qc32.mvlt

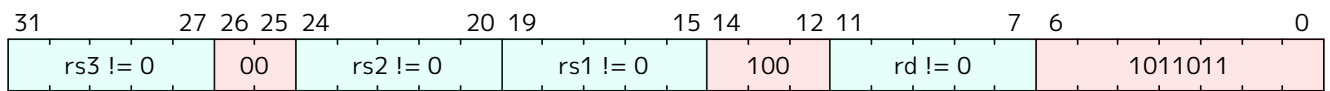
Synopsis

Conditional move if less than (Register)

Mnemonic

```
qc32.mvlt rd, rs1, rs2, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is less than value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) < $signed(X[rs2])) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.88. qc32.mvlti

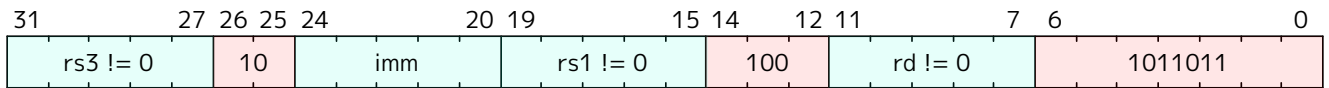
Synopsis

Conditional move if less than (Immediate)

Mnemonic

```
qc32.mvlti rd, rs1, imm, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is less than value of imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) < $signed(imm)) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.89. qc32.mvltu

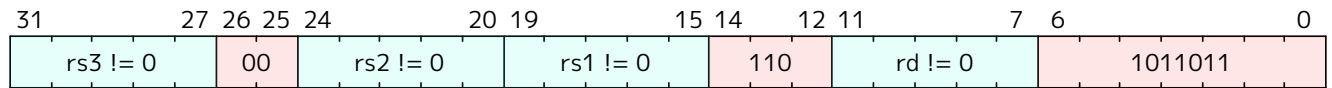
Synopsis

Conditional move if less than unsigned (Register)

Mnemonic

```
qc32.mvltu rd, rs1, rs2, rs3
```

Encoding



Description

Move rs3 to rd if the unsigned value in rs1 is less than unsigned value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] < X[rs2]) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.90. qc32.mvltui

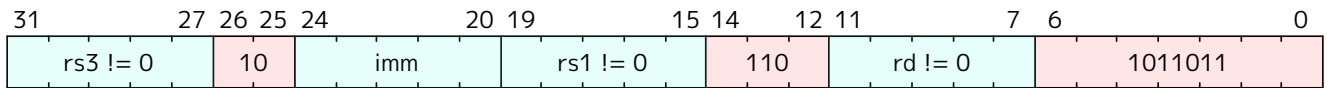
Synopsis

Conditional move if less than unsigned (Immediate)

Mnemonic

```
qc32.mvltui rd, rs1, imm, rs3
```

Encoding



Description

Move rs3 to rd if the unsigned value in rs1 is less than unsigned value of imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] < imm) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.91. qc32.mvne

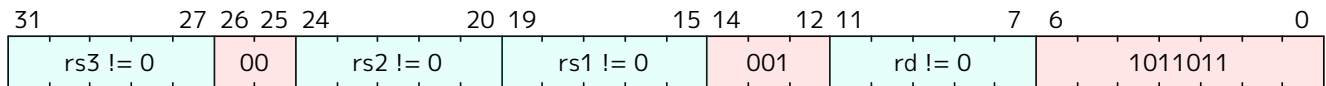
Synopsis

Conditional move if not equal (Register)

Mnemonic

```
qc32.mvne rd, rs1, rs2, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is not equal to value rs2

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rs1] != X[rs2]) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.92. qc32.mvnei

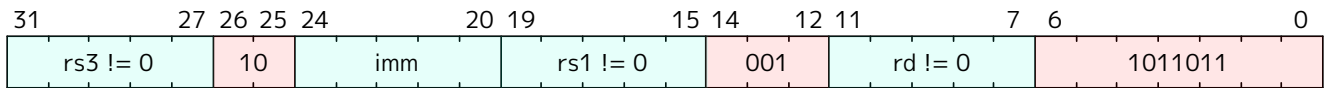
Synopsis

Conditional move if not equal (Immediate)

Mnemonic

```
qc32.mvnei rd, rs1, imm, rs3
```

Encoding



Description

Move rs3 to rd if the value in rs1 is not equal to value imm

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> imm = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rs1]) != $signed(imm)) {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.93. qc32.norm

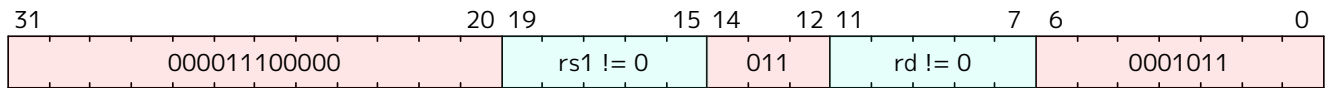
Synopsis

Signed Normalization

Mnemonic

```
qc32.norm rd, rs1
```

Encoding



Description

Signed normalization of rs1. Exponent written in bits[7:0] of the result. Mantissa written in bits[31:8] of the result. Write result to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg clz = (xlen() - 1) - $signed(highest_set_bit(X[rs1]));
XReg clo = (xlen() - 1) - $signed(highest_set_bit(~X[rs1]));
XReg exp = (X[rs1][31] == 1) ? (X[rs1] << (clo - 1)) : (X[rs1] << (clz - 1));
XReg mnt = (X[rs1][31] == 1) ? (-(clo - 1)) : (-(clz - 1));
X[rd] = {mnt[23:0], exp[7:0]};
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.94. qc32.normeu

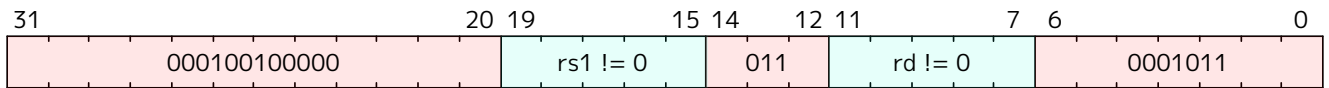
Synopsis

Unsigned Even Normalization

Mnemonic

```
qc32.normeu rd, rs1
```

Encoding



Description

Unsigned even normalization of rs1 (exponent is even). Even exponent written in bits[7:0] of the result. Mantissa (based on even exponent) written in bits[31:8] of the result. Write result to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg clz = xlen() - 1 - $signed(highest_set_bit(X[rs1]) & ~1);
XReg exp = (X[rs1] << clz);
XReg mnt = (-clz);
X[rd] = {mnt[23:0], exp[7:0]};
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.95. qc32.normu

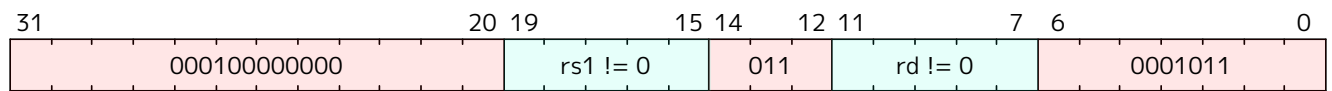
Synopsis

Unsigned Normalization

Mnemonic

```
qc32.normu rd, rs1
```

Encoding



Description

Unsigned normalization of rs1. Exponent written in bits[7:0] of the result. Mantissa written in bits[31:8] of the result. Write result to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg clz = (xlen() - 1) - $signed(highest_set_bit(X[rs1]));
XReg exp = (X[rs1] << clz);
XReg mnt = (-clz);
X[rd] = {mnt[23:0], exp[7:0]};
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.96. qc32.selecteqi

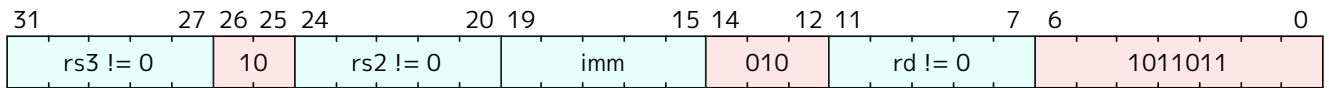
Synopsis

Select load immediate or register if equal (Immediate)

Mnemonic

```
qc32.selecteqi rd, imm, rs2, rs3
```

Encoding



Description

Move rs2 to rd if the value in rd is equal to value imm, move rs3 to rd otherwise

Decode Variables

```
Bits<5> imm = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rd]) == $signed(imm)) {
    X[rd] = X[rs2];
} else {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.97. qc32.selectieq

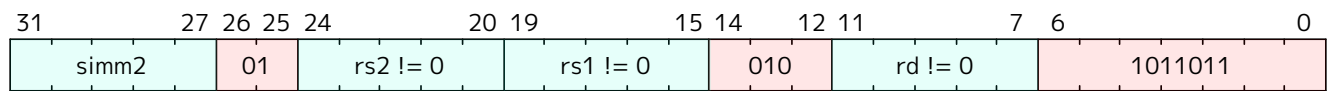
Synopsis

Select load immediate or register if equal (Register)

Mnemonic

```
qc32.selectieq rd, rs1, rs2, simm2
```

Encoding



Description

Move rs2 to rd if the value in rd is equal to value rs1, move simm2 to rd otherwise

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm2 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if (X[rd] == X[rs1]) {
    X[rd] = X[rs2];
} else {
    X[rd] = sext(simm2, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.98. qc32.selectieqi

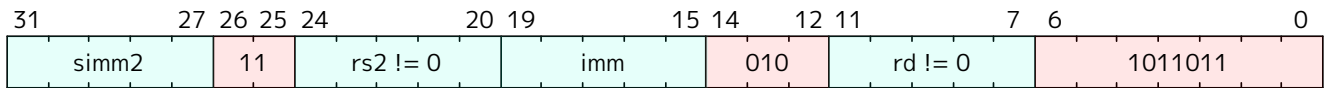
Synopsis

Select load immediate or register if equal (Immediate)

Mnemonic

```
qc32.selectieqi rd, imm, rs2, simm2
```

Encoding



Description

Move rs2 to rd if the value in rd is equal to value imm, move simm2 to rd otherwise

Decode Variables

```
Bits<5> imm = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm2 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rd]) == $signed(imm)) {
    X[rd] = X[rs2];
} else {
    X[rd] = sext(simm2, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.99. qc32.selectiieq

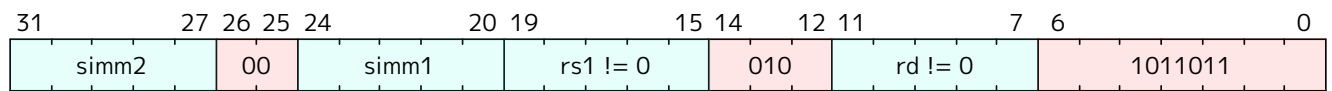
Synopsis

Select load immediate if equal (Register)

Mnemonic

```
qc32.selectiieq rd, rs1, simm1, simm2
```

Encoding



Description

Move simm1 to rd if the value in rd is equal to value rs1, move simm2 to rd otherwise

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> simm1 = $encoding[24:20];
Bits<5> simm2 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rd]) == $signed(X[rs1])) {
    X[rd] = sext(simm1, 5);
} else {
    X[rd] = sext(simm2, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.100. qc32.selectiine

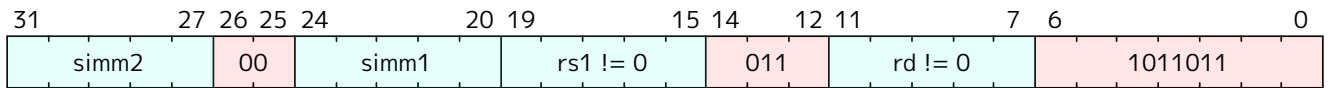
Synopsis

Select load immediate if not equal (Register)

Mnemonic

```
qc32.selectiine rd, rs1, simm1, simm2
```

Encoding



Description

Move simm1 to rd if the value in rd is not equal to value rs1, move simm2 to rd otherwise

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> simm1 = $encoding[24:20];
Bits<5> simm2 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rd]) != $signed(X[rs1])) {
    X[rd] = sext(simm1, 5);
} else {
    X[rd] = sext(simm2, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.101. qc32.selectine

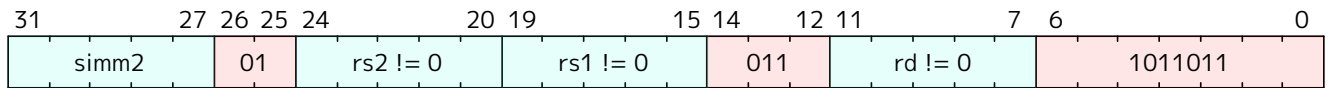
Synopsis

Select load immediate or register if not equal (Register)

Mnemonic

```
qc32.selectine rd, rs1, rs2, simm2
```

Encoding



Description

Move rs2 to rd if the value in rd is not equal to value rs1, move simm2 to rd otherwise

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm2 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rd]) != $signed(X[rs1])) {
    X[rd] = X[rs2];
} else {
    X[rd] = sext(simm2, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.102. qc32.selectinei

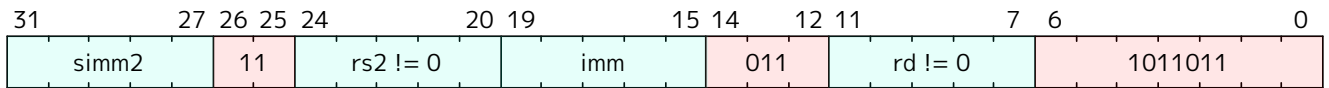
Synopsis

Select load immediate or register if not equal (Immediate)

Mnemonic

```
qc32.selectinei rd, imm, rs2, simm2
```

Encoding



Description

Move rs2 to rd if the value in rd is not equal to value imm, move simm2 to rd otherwise

Decode Variables

```
Bits<5> imm = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> simm2 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rd]) != $signed(imm)) {
    X[rd] = X[rs2];
} else {
    X[rd] = sext(simm2, 5);
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.103. qc32.selectnei

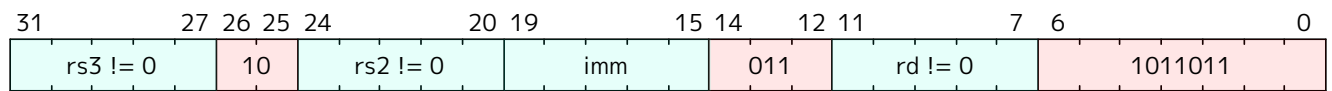
Synopsis

Select load immediate or register if not equal (Immediate)

Mnemonic

```
qc32.selectnei rd, imm, rs2, rs3
```

Encoding



Description

Move rs2 to rd if the value in rd is not equal to value imm, move rs3 to rd otherwise

Decode Variables

```
Bits<5> imm = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[31:27];
Bits<5> rd = $encoding[11:7];
```

Operation

```
if ($signed(X[rd]) != $signed(imm)) {
    X[rd] = X[rs2];
} else {
    X[rd] = X[rs3];
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.104. qc32.setint

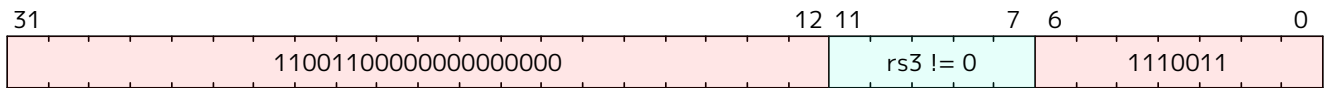
Synopsis

Set interrupt (Register)

Mnemonic

```
qc32.setint rs3
```

Encoding



Description

Set interrupt, interrupt number is in rs3.

Decode Variables

```
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
Bits<12> MCLICIP0_ADDR = CSR[mclicip0].address();
XReg idx = rs3 / 32;
XReg bit = rs3 % 32;
XReg pre_csr = CSR[MCLICIP0_ADDR + idx].sw_read();
CSR[MCLICIP0_ADDR + idx].sw_write(pre_csr | (1 << bit));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.105. qc32.setinti

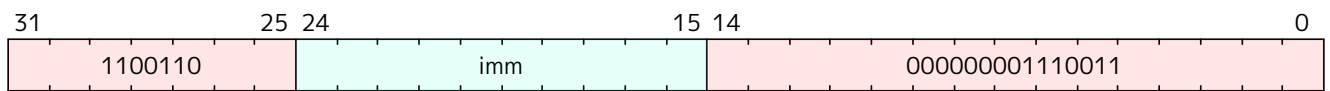
Synopsis

Set interrupt (Immediate)

Mnemonic

```
qc32.setinti imm
```

Encoding



Description

Set interrupt, interrupt number is in `imm` (0 - 1023).

Decode Variables

```
Bits<10> imm = $encoding[24:15];
```

Operation

```
Bits<12> MCLICIP0_ADDR = CSR[mclicip0].address();
XReg idx = imm / 32;
XReg bit = imm % 32;
XReg pre_csr = CSR[MCLICIP0_ADDR + idx].sw_read();
CSR[MCLICIP0_ADDR + idx].sw_write(pre_csr | (1 << bit));
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.106. qc32.setwm

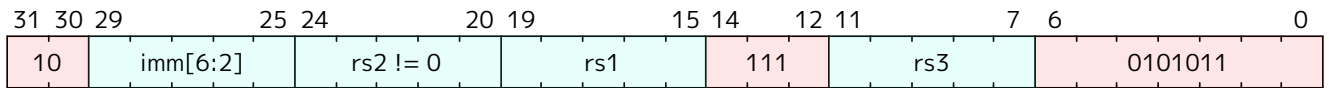
Synopsis

Set word multiple (Register)

Mnemonic

```
qc32.setwm rs3, rs2, imm(rs1)
```

Encoding



Description

Stores the value of rs3 multiple times into the address starting at (rs1 + imm). The number of writes is in rs2.

Decode Variables

```
Bits<7> imm = {$encoding[29:25], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
XReg vaddr = X[rs1] + imm;
Bits<32> write_value = X[rs3][31:0];
XReg num_words = X[rs2];
for (U32 i = 0; i < num_words; i++) {
    write_memory<32>(vaddr, write_value);
    vaddr = vaddr + 4;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.107. qc32.setwmi

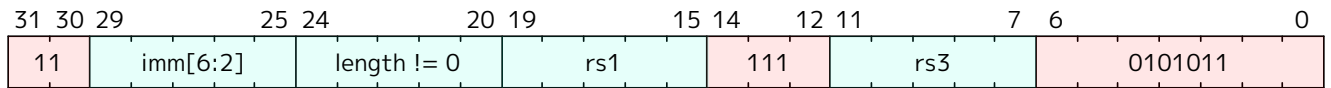
Synopsis

Set word multiple (Immediate)

Mnemonic

```
qc32.setwmirs3, rs2, imm(rs1)
```

Encoding



Description

Stores the value of rs3 multiple times into the address starting at (rs1 + imm). The number of writes is in length.

Decode Variables

```
Bits<7> imm = {$encoding[29:25], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> length = $encoding[24:20];
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
XReg vaddr = X[rs1] + imm;
Bits<32> write_value = X[rs3][31:0];
for (U32 i = 0; i < length; i++) {
    write_memory<32>(vaddr, write_value);
    vaddr = vaddr + 4;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.108. qc32.sh4add

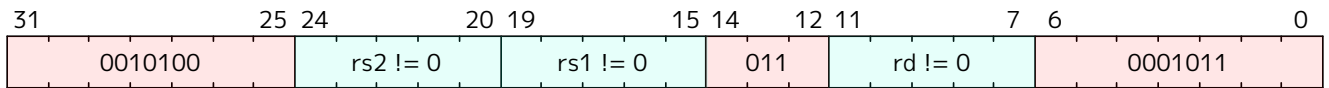
Synopsis

Shift left by 4 and add

Mnemonic

```
qc32.sh4add rd, rs1, rs2
```

Encoding



Description

This instruction shifts rs1 to the left by 4 bit and adds it to rs2.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rs2] + (X[rs1] << 4);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.109. qc32.sh5add

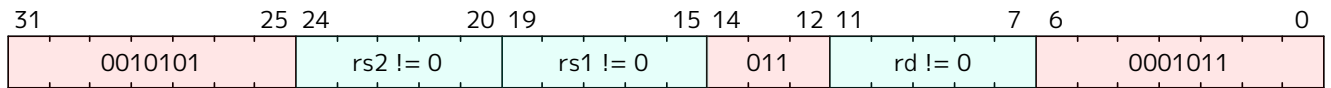
Synopsis

Shift left by 5 and add

Mnemonic

```
qc32.sh5add rd, rs1, rs2
```

Encoding



Description

This instruction shifts rs1 to the left by 5 bit and adds it to rs2.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rs2] + (X[rs1] << 5);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.110. qc32.sh6add

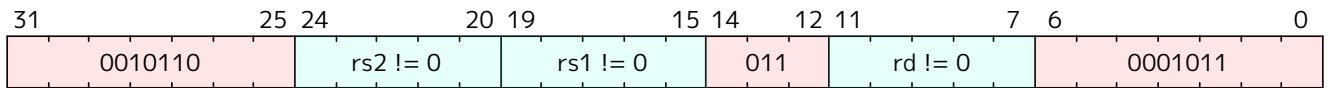
Synopsis

Shift left by 6 and add

Mnemonic

```
qc32.sh6add rd, rs1, rs2
```

Encoding



Description

This instruction shifts rs1 to the left by 6 bit and adds it to rs2.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rs2] + (X[rs1] << 6);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.111. qc32.sh7add

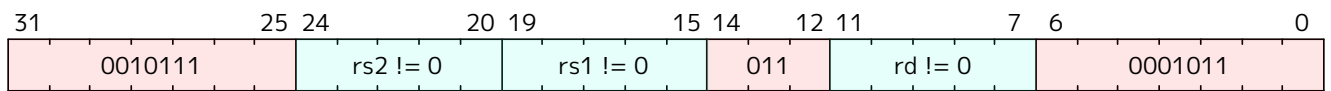
Synopsis

Shift left by 7 and add

Mnemonic

```
qc32.sh7add rd, rs1, rs2
```

Encoding



Description

This instruction shifts rs1 to the left by 7 bit and adds it to rs2.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rs2] + (X[rs1] << 7);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.112. qc32.sh8add

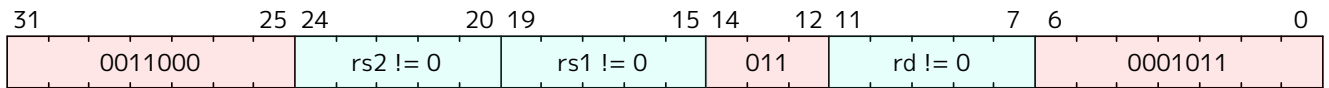
Synopsis

Shift left by 8 and add

Mnemonic

```
qc32.sh8add rd, rs1, rs2
```

Encoding



Description

This instruction shifts rs1 to the left by 8 bit and adds it to rs2.

Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rs2] + (X[rs1] << 8);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.113. qc32.slasat

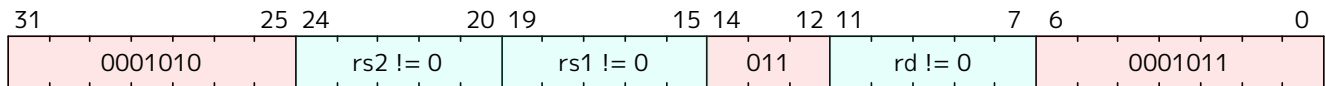
Synopsis

Saturating arithmetic left shift

Mnemonic

```
qc32.slasat rd, rs1, rs2
```

Encoding



Description

Left shift rs1 by the value of rs2, and saturate the signed result. The number of words is in length.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> shifted_value = X[rs1] << X[rs2][4:0];
XReg most_negative_number = 1 << (xlen() - 1);
XReg most_positive_number = (1 << (xlen() - 1) - 1);
if ($signed(shifted_value) < $signed(most_negative_number)) {
    X[rd] = most_negative_number;
} else if ($signed(shifted_value) > $signed(most_positive_number)) {
    X[rd] = most_positive_number;
} else {
    X[rd] = shifted_value;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.114. qc32.sllsat

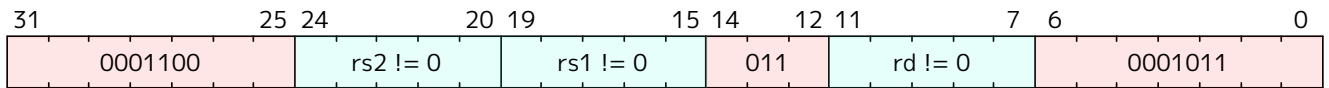
Synopsis

Saturating logical left shift

Mnemonic

```
qc32.sllsat rd, rs1, rs2
```

Encoding



Description

Left shift rs1 by the value of rs2, and saturate the unsigned result. The number of words is in length.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
Bits<{1'b0, XLEN} * 2> sext_double_width_rs1 = {{XLEN{X[rs1][xlen() - 1]}}, X[rs1]};
Bits<{1'b0, XLEN} * 2> shifted_value = sext_double_width_rs1 << X[rs2][4:0];
XReg largest_unsigned_value = {XLEN{1'b1}};
if (shifted_value > largest_unsigned_value) {
    X[rd] = largest_unsigned_value;
} else {
    X[rd] = shifted_value;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.115. qc32.srb

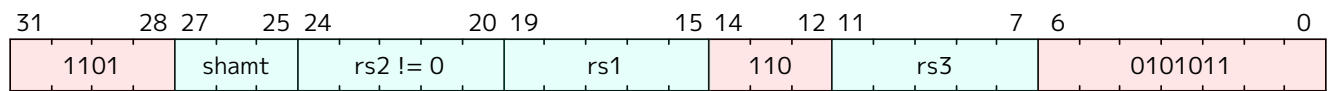
Synopsis

Store indexed byte

Mnemonic

```
qc32.srb rs3, rs1, rs2, shamt
```

Encoding



Description

Store 8 bits of data from register rs3 to an address formed by adding rs1 to rs2, shifted by shamt.

Decode Variables

```
Bits<3> shamt = $encoding[27:25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + (X[rs2] << shamt);
write_memory<8>(virtual_address, X[rs3][7:0]);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.116. qc32.srh

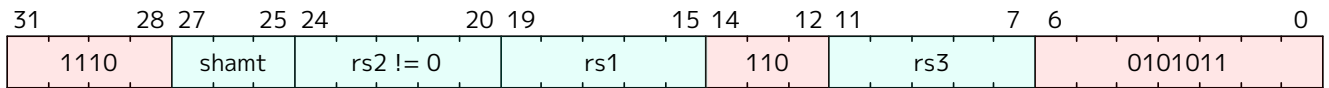
Synopsis

Store indexed halfword

Mnemonic

qc32.srh rs3, rs1, rs2, shamt

Encoding



Description

Store 16 bits of data from register rs3 to an address formed by adding rs1 to rs2, shifted by shamt.

Decode Variables

Bits<3> shamt = \$encoding[27:25];
Bits<5> rs1 = \$encoding[19:15];
Bits<5> rs2 = \$encoding[24:20];
Bits<5> rs3 = \$encoding[11:7];

Operation

XReg virtual_address = **X**[rs1] + (**X**[rs2] << shamt);
write_memory<16>(virtual_address, **X**[rs3][15:0]);

Included in

Extension	Minimum version
Xqciu	>= 0

5.117. qc32.srw

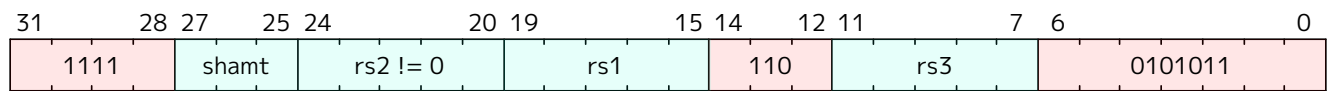
Synopsis

Store indexed word

Mnemonic

```
qc32.srw rs3, rs1, rs2, shamt
```

Encoding



Description

Store 32 bits of data from register rs3 to an address formed by adding rs1 to rs2, shifted by shamt.

Decode Variables

```
Bits<3> shamt = $encoding[27:25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + (X[rs2] << shamt);
write_memory<32>(virtual_address, X[rs3][31:0]);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.118. qc32.subsat

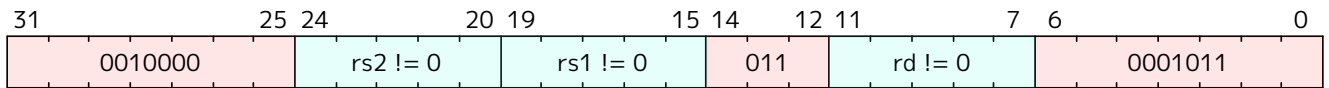
Synopsis

Saturating signed subtraction

Mnemonic

```
qc32.subsat rd, rs1, rs2
```

Encoding



Description

Subtract signed values rs1 and rs2, saturate the signed result, and write to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg result = X[rs1] - X[rs2];
XReg most_negative_number = 1 << (xlen() - 1);
XReg most_positive_number = (1 << (xlen() - 1)) - 1;
if (X[rs1][xlen() - 1] != X[rs2][xlen() - 1]) {
    if (result[xlen() - 1] != X[rs1][xlen() - 1]) {
        if ($signed(X[rs1]) < 0) {
            X[rd] = most_negative_number;
        } else {
            X[rd] = most_positive_number;
        }
    }
}
X[rd] = result;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.119. qc32.subusat

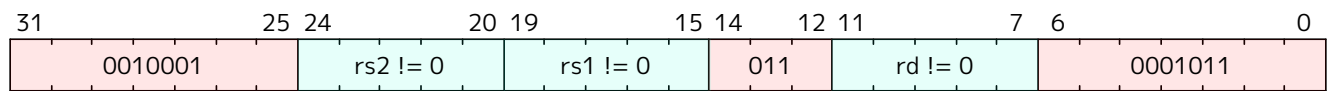
Synopsis

Saturating unsigned subtraction

Mnemonic

```
qc32.subusat rd, rs1, rs2
```

Encoding



Description

Subtract unsigned values rs1 and rs2, saturate the unsigned result, and write to rd.

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = (X[rs1] < X[rs2]) ? 0 : X[rs1] - X[rs2];
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.120. qc32.swm

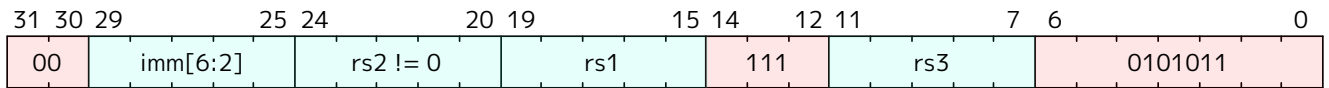
Synopsis

Store word multiple

Mnemonic

```
qc32.swm rs3, rs2, imm(rs1)
```

Encoding



Description

Stores multiple words from the registers starting at rs3 to the address starting at (rs1 + imm). The number of words is in rs2.

Decode Variables

```
Bits<7> imm = {$encoding[29:25], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
XReg vaddr = X[rs1] + imm;
XReg num_words = X[rs2];
raise(ExceptionCode::IllegalInstruction, $encoding) if (rs3 + num_words) > 32;
for (U32 i = 0; i < num_words; i++) {
    write_memory<32>(vaddr, X[rs3 + i]);
    vaddr = vaddr + 4;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.121. qc32.swmi

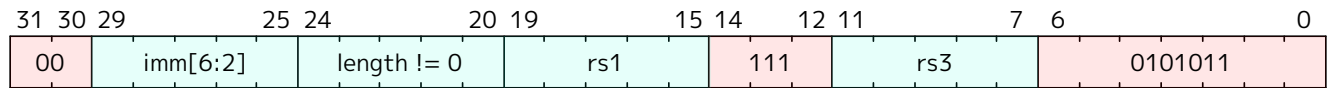
Synopsis

Store word multiple (immediate)

Mnemonic

```
qc32.swmi rs3, length, imm(rs1)
```

Encoding



Description

Stores multiple words from the registers starting at rs3 to the address starting at (rs1 + imm). The number of words is in length immediate.

Decode Variables

```
Bits<7> imm = {$encoding[29:25], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> length = $encoding[24:20];
Bits<5> rs3 = $encoding[11:7];
```

Operation

```
XReg vaddr = X[rs1] + imm;
raise(ExceptionCode::IllegalInstruction, $encoding) if (rs3 + length) > 32;
for (U32 i = 0; i < length; i++) {
    write_memory<32>(vaddr, X[rs3 + i]);
    vaddr = vaddr + 4;
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.122. qc32.wrap

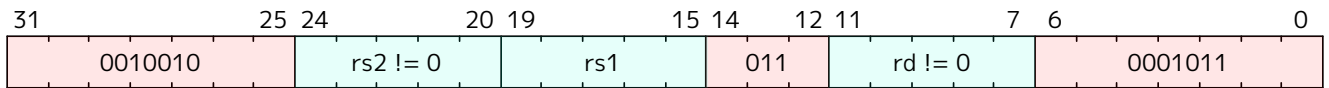
Synopsis

Wraparound (Register)

Mnemonic

```
qc32.wrap rd, rs1, rs2
```

Encoding



Description

If $rs1 \geq rs2$ perform subtraction between $rs1$ and $rs2$. If $rs1 < 0$, perform addition between $rs1$ and $rs2$, else, select $rs1$. The result is stored in rd .

Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg rs1_value = X[rs1];
X[rd] = (($signed(rs1_value) >= $signed(X[rs2])) ? rs1_value - X[rs2] : (($signed
(rs1_value) < 0) ? (rs1_value + X[rs2]) : rs1_value);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.123. qc32.wrapi

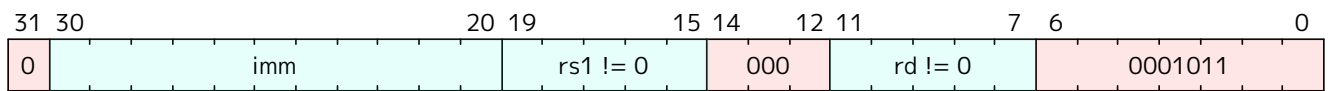
Synopsis

Wraparound (Immediate)

Mnemonic

```
qc32.wrapi rd, rs1, imm
```

Encoding



Description

If $rs1 \geq imm$ perform subtraction between $rs1$ and imm . If $rs1 < 0$, perform addition between $rs1$ and imm , else, select $rs1$. The result is stored in rd .

Decode Variables

```
Bits<11> imm = $encoding[30:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg rs1_value = X[rs1];
X[rd] = ($signed(rs1_value) >= $signed(imm)) ? rs1_value - imm : (($signed(rs1_value) < 0) ? (rs1_value + imm) : rs1_value);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.124. qc48.addai

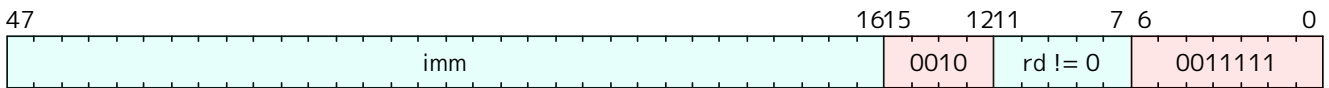
Synopsis

Add immediate

Mnemonic

```
qc48.addai rd, imm
```

Encoding



Description

Add a 32-bit immediate `imm` to the value in `rd`, and store the result back in `rd`.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] + imm;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.125. qc48.addi

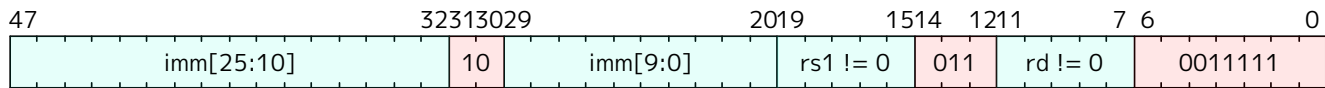
Synopsis

Add immediate

Mnemonic

```
qc48.addi rd, rs1, imm
```

Encoding



Description

Add a sign-extended 26-bit immediate `imm` to the value in `rs1`, and store the result in `rd`.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5>  rs1 = $encoding[19:15];
Bits<5>  rd  = $encoding[11:7];
```

Operation

```
X[rd] = X[rs1] + sext(imm, 26);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.126. qc48.andai

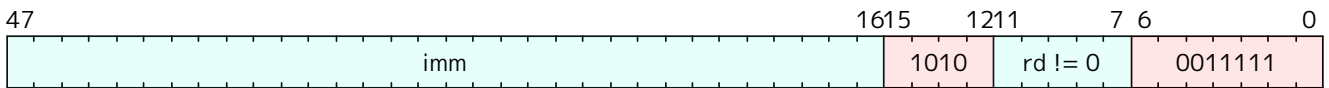
Synopsis

And immediate

Mnemonic

```
qc48.andai rd, imm
```

Encoding



Description

And a 32-bit immediate `imm` to the value in `rd`, and store the result back in `rd`.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] & imm;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.127. qc48.andi

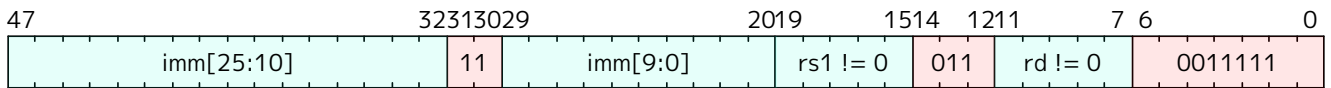
Synopsis

Add immediate

Mnemonic

```
qc48.andi rd, rs1, imm
```

Encoding



Description

And a sign-extended 26-bit immediate `imm` to the value in `rs1`, and store the result in `rd`.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5>  rs1 = $encoding[19:15];
Bits<5>  rd  = $encoding[11:7];
```

Operation

```
X[rd] = X[rs1] & sext(imm, 26);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.128. qc48.beqi

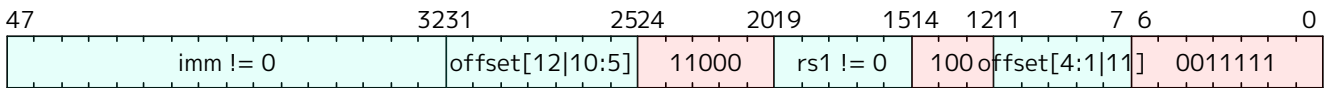
Synopsis

Branch on equal (immediate)

Mnemonic

```
qc48.beqi rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is equal to the signed immediate imm

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<16> imm = $encoding[47:32];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if ($signed(X[rs1]) == $signed(imm)) {  
  jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.129. qc48.bgei

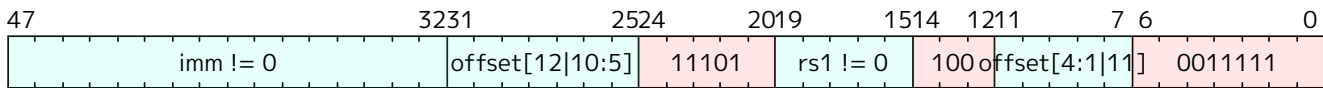
Synopsis

Branch on greater than or equal (immediate)

Mnemonic

```
qc48.bgei rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is greater than or equal to the immediate imm.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<16> imm = $encoding[47:32];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if ($signed(X[rs1]) >= $signed(imm)) {  
  jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.130. qc48.bgeui

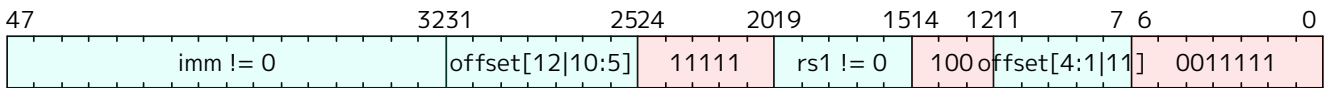
Synopsis

Branch on greater than or equal unsigned (immediate)

Mnemonic

```
qc48.bgeui rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the unsigned value in rs1 is greater than or equal to the unsigned immediate imm.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],
$encoding[11:8], 1'd0});
Bits<16> imm = $encoding[47:32];
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if (X[rs1] >= imm) {
    jump_halfword(PC + $signed(offset));
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.131. qc48.blti

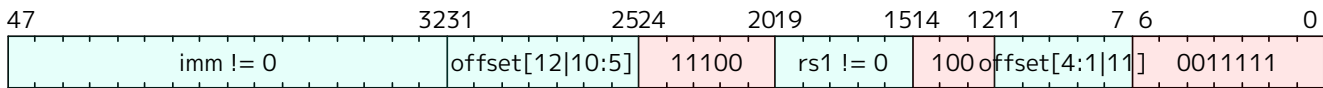
Synopsis

Branch on less than (immediate)

Mnemonic

```
qc48.blti rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is less than the immediate imm.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],  
$encoding[11:8], 1'd0});  
Bits<16> imm = $encoding[47:32];  
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if ($signed(X[rs1]) < $signed(imm)) {  
  jump_halfword(PC + $signed(offset));  
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.132. qc48.bltoi

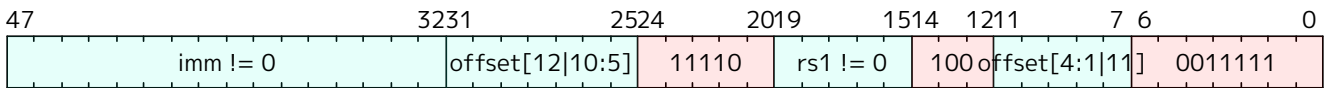
Synopsis

Branch on less than unsigned (immediate)

Mnemonic

```
qc48.bltoi rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the unsigned value in rs1 is less than the unsigned immediate imm.

Decode Variables

```
signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],
$encoding[11:8], 1'd0});
Bits<16> imm = $encoding[47:32];
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
if (X[rs1] < imm) {
  jump_halfword(PC + $signed(offset));
}
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.133. qc48.bnei

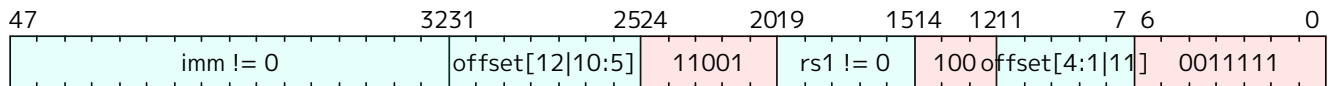
Synopsis

Branch on not equal (immediate)

Mnemonic

```
qc48.bnei rs1, imm, offset
```

Encoding



Description

Branches to PC + offset if the value in rs1 is not equal to the signed immediate imm.

Decode Variables

```

signed Bits<13> offset = sext({$encoding[31], $encoding[7], $encoding[30:25],
$encoding[11:8], 1'd0});
Bits<16> imm = $encoding[47:32];
Bits<5> rs1 = $encoding[19:15];

```

Operation

```

if ($signed(X[rs1]) != $signed(imm)) {
    jump_halfword(PC + $signed(offset));
}

```

Included in

Extension	Minimum version
Xqciu	>= 0

5.134. qc48j

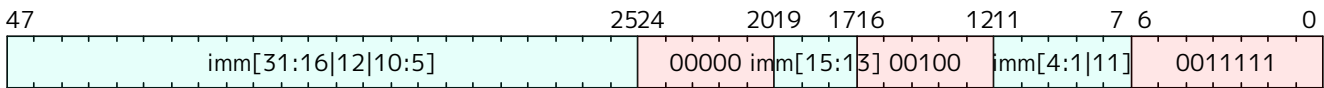
Synopsis

Jump

Mnemonic

qc48.j imm

Encoding



Description

Jump to a PC-relative offset

Decode Variables

signed **Bits**<32> imm = sext({\$encoding[47:32], \$encoding[19:17], \$encoding[31], \$encoding[7], \$encoding[30:25], \$encoding[11:8], 1'd0});

Operation

XReg retrun_addr = **PC** + 6;
jump_halfword(**PC** + imm);

Included in

Extension	Minimum version
Xqciu	>= 0

5.135. qc48.jal

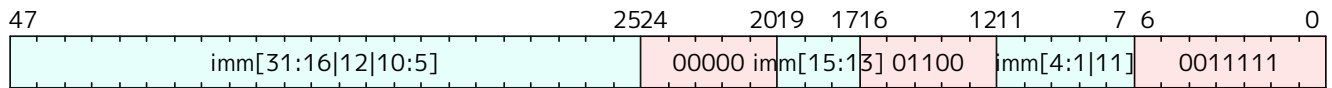
Synopsis

Jump and link

Mnemonic

qc48.jal imm

Encoding



Description

Jump to a PC-relative offset and store the return address in x1.

Decode Variables

```
signed Bits<32> imm = sext({$encoding[47:32], $encoding[19:17], $encoding[31],
$encoding[7], $encoding[30:25], $encoding[11:8], 1'd0});
```

Operation

```
XReg retrun_addr = PC + 6;
jump_halfword(PC + imm);
X[1] = retrun_addr;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.136. qc48.lb

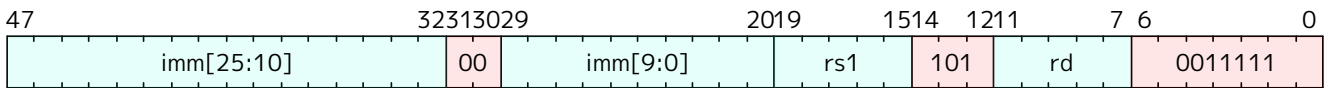
Synopsis

Load byte

Mnemonic

```
qc48.lb rd, imm(rs1)
```

Encoding



Description

Load 8 bits of data into register rd from an address formed by adding rs1 to a signed offset imm. Sign extend the result.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + imm;
X[rd] = sext(read_memory<8>(virtual_address), 7);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.137. qc48.lbu

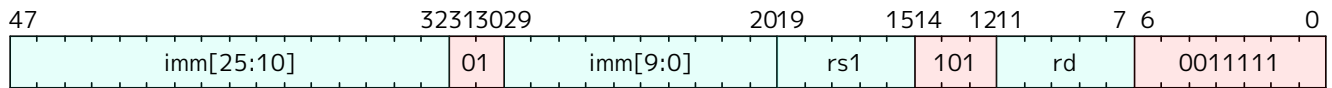
Synopsis

Load byte unsigned

Mnemonic

```
qc48.lbu rd, imm(rs1)
```

Encoding



Description

Load 8 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset `imm`. Zero extend the result.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<8>(virtual_address);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.138. qc48.lh

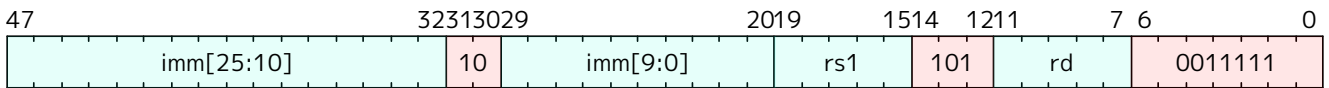
Synopsis

Load halfword

Mnemonic

```
qc48.lh rd, imm(rs1)
```

Encoding



Description

Load 16 bits of data into register rd from an address formed by adding rs1 to a signed offset imm. Sign extend the result.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + imm;
X[rd] = sext(read_memory<16>(virtual_address), 15);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.139. qc48.lhu

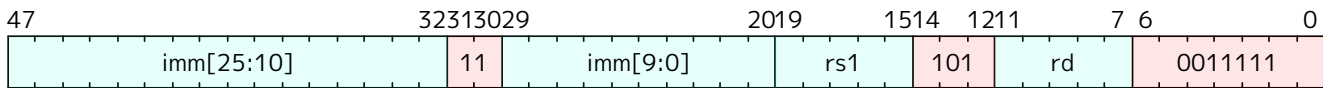
Synopsis

Load halfword unsigned

Mnemonic

```
qc48.lhu rd, imm(rs1)
```

Encoding



Description

Load 16 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset `imm`. Zero extend the result.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<16>(virtual_address);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.140. qc48.li

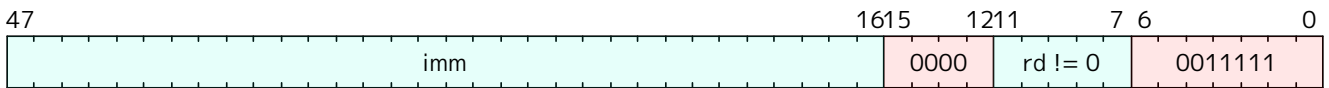
Synopsis

Load immediate large

Mnemonic

```
qc48.li rd, imm
```

Encoding



Description

Loads the 32-bit immediate `imm` into `rd`.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = imm;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.141. qc48.lw

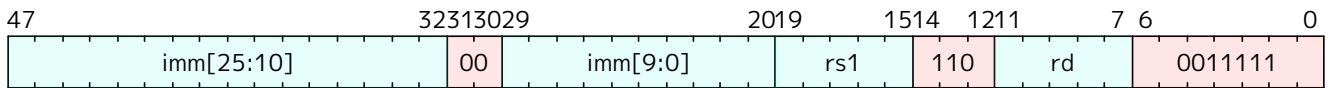
Synopsis

Load word

Mnemonic

```
qc48.lw rd, imm(rs1)
```

Encoding



Description

Load 32 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset `imm`. Sign extend the result.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<32>(virtual_address);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.142. qc48.orai

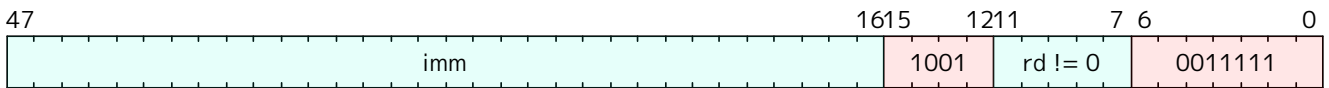
Synopsis

Or immediate

Mnemonic

```
qc48.orai rd, imm
```

Encoding



Description

Or a 32-bit immediate `imm` to the value in `rd`, and store the result back in `rd`.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] | imm;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.143. qc48.ori

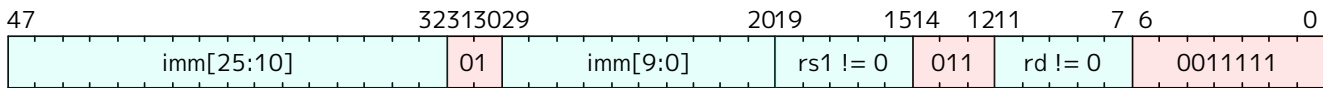
Synopsis

Or immediate

Mnemonic

```
qc48.ori rd, rs1, imm
```

Encoding



Description

Or a sign-extended 26-bit immediate `imm` to the value in `rs1`, and store the result in `rd`.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5>  rs1 = $encoding[19:15];
Bits<5>  rd  = $encoding[11:7];
```

Operation

```
X[rd] = X[rs1] | sext(imm, 26);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.144. qc48.sb

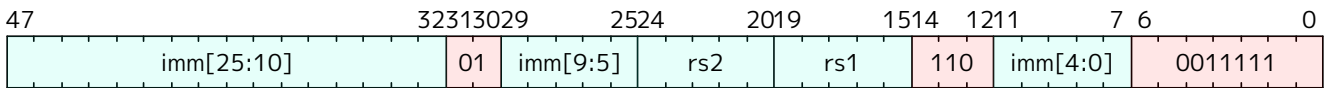
Synopsis

Store byte

Mnemonic

```
qc48.sb rs2, imm(rs1)
```

Encoding



Description

Store 8 bits of data from register rs2 to an address formed by adding rs1 to a signed offset imm.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:25], $encoding[11:7] };
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
XReg virtual_address = X[rs1] + $signed(imm);
write_memory<8>(virtual_address, X[rs2][7:0]);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.145. qc48.sh

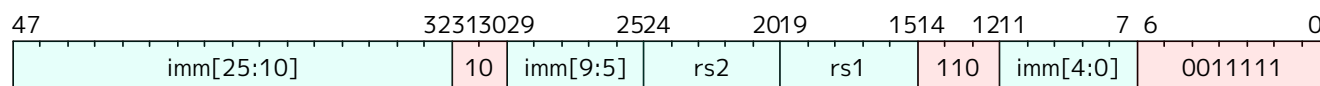
Synopsis

Store halfword

Mnemonic

```
qc48.sh rs2, imm(rs1)
```

Encoding



Description

Store 16 bits of data from register rs2 to an address formed by adding rs1 to a signed offset imm.

Decode Variables

```
Bits<26> imm = {$encoding[47:32], $encoding[29:25], $encoding[11:7]};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
XReg virtual_address = X[rs1] + $signed(imm);
write_memory<16>(virtual_address, X[rs2][15:0]);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.146. qc48.sw

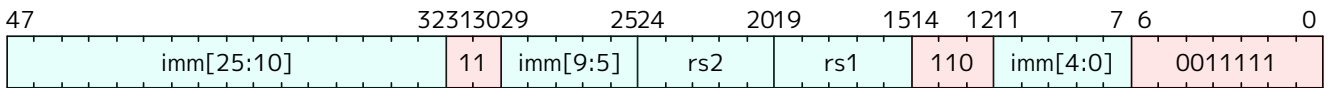
Synopsis

Store word

Mnemonic

```
qc48.sw rs2, imm(rs1)
```

Encoding



Description

Store 32 bits of data from register rs2 to an address formed by adding rs1 to a signed offset imm.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:25], $encoding[11:7] };
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

Operation

```
XReg virtual_address = X[rs1] + $signed(imm);
write_memory<32>(virtual_address, X[rs2][31:0]);
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.147. qc48.xorai

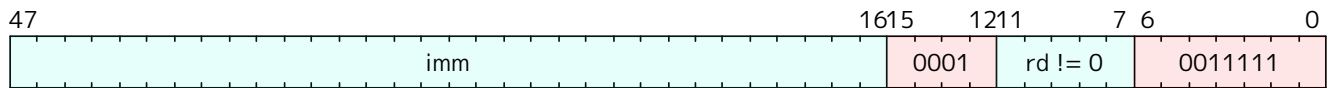
Synopsis

Exclusive Or immediate

Mnemonic

```
qc48.xorai rd, imm
```

Encoding



Description

Exclusive or a 32-bit immediate `imm` to the value in `rd`, and store the result back in `rd`.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] ^ imm;
```

Included in

Extension	Minimum version
Xqciu	>= 0

5.148. qc48.xori

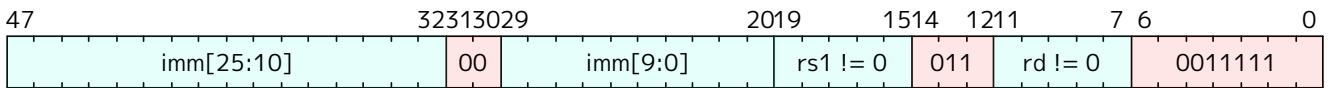
Synopsis

Exclusive Or immediate

Mnemonic

```
qc48.xori rd, rs1, imm
```

Encoding



Description

Exclusive or a sign-extended 26-bit immediate `imm` to the value in `rs1`, and store the result in `rd`.

Decode Variables

```
Bits<26> imm = { $encoding[47:32], $encoding[29:20] };
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rs1] ^ sext(imm, 26);
```

Included in

Extension	Minimum version
Xqciu	>= 0

Chapter 6. IDL Functions

6.1. abort_current_instruction (builtin)

Abort the current instruction, and start refetching from $\$pc$.

Return Type	void
Arguments	

6.2. access_check

Checks if the physical address `paddr` is able to access memory, and raises the appropriate exception if not.

Return Type	void
Arguments	Bits<PHYS_ADDR_WIDTH> <code>paddr</code> , XReg <code>vaddr</code> , MemoryOperation <code>type</code> , ExceptionCode <code>fault_type</code>

```
if (paddr > 1 << PHYS_ADDR_WIDTH) - access_size {
    raise(fault_type, vaddr);
}
if (!pmp_check<access_size>(paddr[PHYS_ADDR_WIDTH - 1:0], type)) {
    raise(fault_type, vaddr);
}
```

6.3. assert (builtin)

Assert that a condition is true. Failure represents an error in the IDL model.

Return Type	void
Arguments	Boolean <code>test</code> , String <code>message</code>

6.4. atomic_check_then_write_32 (builtin)

Atomically:

- Reads 32-bits from `paddr`
- Compares the read value to `compare_value`
- Writes `write_value` to `paddr` if the comparison was bitwise-equal

returns true if the write occurs, and false otherwise

Preconditions:

- paddr must be aligned to 32-bits

Return Type	Boolean
Arguments	Bits<PHYS_ADDR_WIDTH> paddr, Bits<32> compare_value, Bits<32> write_value

6.5. atomic_check_then_write_64 (builtin)

Atomically:

- Reads 64-bits from paddr
- Compares the read value to compare_value
- Writes write_value to paddr if the comparison was bitwise-equal

returns true if the write occurs, and false otherwise

Preconditions:

- paddr must be aligned to 64-bits

Return Type	Boolean
Arguments	Bits<PHYS_ADDR_WIDTH> paddr, Bits<64> compare_value, Bits<64> write_value

6.6. current_translation_mode

Returns the current translation mode for a load or store given the machine state (e.g., value of satp csr).

Return Type	SatpMode
Arguments	

```

PrivilegeMode effective_mode = effective_ldst_mode();
SatpMode translation_mode;
if (effective_mode == PrivilegeMode::M) {
    return SatpMode::Bare;
} else if (CSR[misa].S == 1'b1) {
    return CSR[satp].MODE;
} else {
    return SatpMode::Bare;
}

```

6.7. delay (builtin)

Delay the processor by cycles cycles.

Return Type	void
Arguments	XReg cycles

6.8. effective_ldst_mode

Returns the effective privilege mode for normal explicit loads and stores, taking into account the current actual privilege mode and modifications from mstatus.MPRV.

Return Type	PrivilegeMode
Arguments	

```

if (mode() == PrivilegeMode::M) {
    if (implemented?(ExtensionName::U) && CSR[mstatus].MPRV == 1) {
        if (CSR[mstatus].MPP == 0b00) {
            if (implemented?(ExtensionName::H) && mpv() == 0b1) {
                return PrivilegeMode::VU;
            } else {
                return PrivilegeMode::U;
            }
        } else if (implemented?(ExtensionName::S) && CSR[mstatus].MPP == 0b01) {
            if (implemented?(ExtensionName::H) && mpv() == 0b1) {
                return PrivilegeMode::VS;
            } else {
                return PrivilegeMode::S;
            }
        }
    }
}
return mode();

```

6.9. exception_handling_mode

Returns the target privilege mode that will handle synchronous exception exception_code

Return Type	PrivilegeMode
Arguments	ExceptionCode exception_code

```

if (mode() == PrivilegeMode::M) {
    return PrivilegeMode::M;
} else if (implemented?(ExtensionName::S) && mode() == PrivilegeMode::HS) || (mode()
== PrivilegeMode::U) {
    if ((CSR[medeleg] & (1 << $bits(exception_code))) != 0) {

```

```

    return PrivilegeMode::HS;
} else {
    return PrivilegeMode::M;
}
} else {
    assert(implemented?(ExtensionName::H) && mode() == PrivilegeMode::VS) || (mode() ==
PrivilegeMode::VU, "Unexpected mode");
    if ((CSR[medeleg] & (1 << $bits(exception_code))) != 0) {
        if ((CSR[hedeleg] & (1 << $bits(exception_code))) != 0) {
            return PrivilegeMode::VS;
        } else {
            return PrivilegeMode::HS;
        }
    } else {
        return PrivilegeMode::M;
    }
}
}

```

6.10. highest_set_bit

Returns the position of the highest (nearest MSB) bit that is '1', or -1 if value is zero.

Return Type	XReg
Arguments	XReg value

```

for (U32 i = xlen() - 1; i >= 0; i--) {
    if (value[i] == 1) {
        return i;
    }
}
return -'sd1;

```

6.11. ialign

Returns IALIGN, the smallest instruction encoding size, in bits.

Return Type	Bits<6>
Arguments	

```

if (implemented?(ExtensionName::C) && (CSR[misa].C == 0x1)) {
    return 16;
} else {
    return 32;
}

```

6.12. implemented? (builtin)

Return true if the implementation supports extension.

Return Type	Boolean
Arguments	ExtensionName extension

6.13. is_naturally_aligned

Checks if M-bit value is naturally aligned to N bits.

Return Type	Boolean
Arguments	Bits<M> value

```
return true if (N == 8);
Bits<M> mask = (N / 8) - 1;
return (value & ~mask) == value;
```

6.14. jump_halfword

Jump to virtual halfword address target_hw_addr.

If target address is misaligned, raise a MisalignedAddress exception.

Return Type	void
Arguments	XReg target_hw_addr

```
assert((target_hw_addr & 0x1) == 0x0, "Expected halfword-aligned address in
jump_halfword");
if (ialign() != 16) {
    if ((target_hw_addr & 0x3) != 0) {
        raise(ExceptionCode::InstructionAddressMisaligned, target_hw_addr);
    }
}
PC = target_hw_addr;
```

6.15. lowest_set_bit

Returns the position of the lowest (nearest LSB) bit that is '1', or XLEN if value is zero.

Return Type	XReg
Arguments	XReg value

```
for (U32 i = 0; i < xlen(); i++) {
    if (value[i] == 1) {
        return i;
    }
}
return xlen();
```

6.16. mode

Returns the current active privilege mode.

Return Type	PrivilegeMode
Arguments	

```
if (!implemented?(ExtensionName::S) && !implemented?(ExtensionName::U) && !
implemented?(ExtensionName::H)) {
    return PrivilegeMode::M;
} else {
    return current_mode;
}
```

6.17. mpv

Returns the current value of CSR[mstatus].MPV (when MXLEN == 64) of CSR[mstatush].MPV (when MXLEN == 32)

Return Type	Bits<1>
Arguments	

```
if (implemented?(ExtensionName::H)) {
    return (XLEN == 32) ? CSR[mstatush].MPV : CSR[mstatus].MPV;
} else {
    assert(false, "TODO");
}
```

6.18. mtval_for

Given an exception code and a **legal** non-zero value for mtval, returns the value to be written in mtval considering implementation options

Return Type	XReg
Arguments	ExceptionCode exception_code, XReg tval

```

if (exception_code == ExceptionCode::Breakpoint) {
    return REPORT_VA_IN_MTVAL_ON_BREAKPOINT ? tval : 0;
} else if (exception_code == ExceptionCode::LoadAddressMisaligned) {
    return REPORT_VA_IN_MTVAL_ON_LOAD_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoAddressMisaligned) {
    return REPORT_VA_IN_MTVAL_ON_STORE_AMO_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionAddressMisaligned) {
    return REPORT_VA_IN_MTVAL_ON_INSTRUCTION_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::LoadAccessFault) {
    return REPORT_VA_IN_MTVAL_ON_LOAD_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoAccessFault) {
    return REPORT_VA_IN_MTVAL_ON_STORE_AMO_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionAccessFault) {
    return REPORT_VA_IN_MTVAL_ON_INSTRUCTION_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::LoadPageFault) {
    return REPORT_VA_IN_MTVAL_ON_LOAD_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoPageFault) {
    return REPORT_VA_IN_MTVAL_ON_STORE_AMO_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionPageFault) {
    return REPORT_VA_IN_MTVAL_ON_INSTRUCTION_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::IllegalInstruction) {
    return REPORT_ENCODING_IN_MTVAL_ON_ILLEGAL_INSTRUCTION ? tval : 0;
} else {
    return 0;
}

```

6.19. mtval_readonly?

Returns whether or not CSR[mtval] is read-only based on implementation options

Return Type	Boolean
Arguments	

```

return !(REPORT_VA_IN_MTVAL_ON_BREAKPOINT || REPORT_VA_IN_MTVAL_ON_LOAD_MISALIGNED ||
REPORT_VA_IN_MTVAL_ON_STORE_MISALIGNED || (implemented?(ExtensionName::Zaamo) &&
REPORT_VA_IN_MTVAL_ON_AMO_MISALIGNED) || REPORT_VA_IN_MTVAL_ON_INSTRUCTION_MISALIGNED
|| REPORT_VA_IN_MTVAL_ON_LOAD_ACCESS_FAULT || REPORT_VA_IN_MTVAL_ON_STORE_ACCESS_FAULT
|| (implemented?(ExtensionName::Zaamo) && REPORT_VA_IN_MTVAL_ON_AMO_ACCESS_FAULT) ||
REPORT_VA_IN_MTVAL_ON_INSTRUCTION_ACCESS_FAULT ||
REPORT_VA_IN_MTVAL_ON_LOAD_PAGE_FAULT || REPORT_VA_IN_MTVAL_ON_STORE_PAGE_FAULT ||
(implemented?(ExtensionName::Zaamo) && REPORT_VA_IN_MTVAL_ON_AMO_PAGE_FAULT) ||
REPORT_VA_IN_MTVAL_ON_INSTRUCTION_PAGE_FAULT ||
REPORT_ENCODING_IN_MTVAL_ON_ILLEGAL_INSTRUCTION || implemented?(ExtensionName::
Sdext));

```

6.20. notify_mode_change (builtin)

Called whenever the privilege mode changes. Downstream tools can use this to hook events.

Return Type	void
Arguments	PrivilegeMode new_mode, PrivilegeMode old_mode

6.21. page_walk

Translate virtual address through a page walk.

May raise a Page Fault if an error involving the page table structure occurs along the walk.

Implicit reads of the page table are accessed check, and may raise Access Faults. Implicit writes (updates of A/D) are also accessed checked, and may raise Access Faults

The translated address *is not* accessed checked.

Returns the translated physical address.

Return Type	Bits<PA_SIZE>
Arguments	Bits<XLEN> vaddr, MemoryOperation op

```

Bits<PA_SIZE> ppn;
U32 VPN_SIZE = (LEVELS == 2) ? 10 : 9;
ExceptionCode access_fault_code = op == MemoryOperation::Read ? ExceptionCode
::LoadAccessFault : (op == MemoryOperation::Fetch ? ExceptionCode
::InstructionAccessFault : ExceptionCode::StoreAmoAccessFault);
ExceptionCode page_fault_code = op == MemoryOperation::Read ? ExceptionCode
::LoadPageFault : (op == MemoryOperation::Fetch ? ExceptionCode::InstructionPageFault
: ExceptionCode::StoreAmoPageFault);
ppn = CSR[satp].PPN;
if (vaddr[xlen() - 1:VA_SIZE] != {xlen() - VA_SIZE{vaddr[VA_SIZE - 1]}}) {
    raise(page_fault_code, vaddr);
}
for (U32 i = (LEVELS - 1); i >= 0; i--) {
    U32 vpn = (vaddr >> (12 + VPN_SIZE * i)) & 1 << VPN_SIZE) - 1);    Bits<PA_SIZE>
pte_addr = (ppn << 12) + (vpn * (PTESIZE / 8);
    access_check<PTESIZE>(pte_addr, vaddr, MemoryOperation::Read, access_fault_code);
    if (!pma_applies?(PmaAttribute::HardwarePageTableRead, pte_addr, PTESIZE)) {
        raise(access_fault_code, vaddr);
    }
    Bits<PTESIZE> pte = read_physical_memory<PTESIZE>(pte_addr);
    PteFlags pte_flags = pte[9:0];
    if ((VA_SIZE != 32) && (pte[60:54] != 0)) {
        raise(page_fault_code, vaddr);
    }
    if (pte_flags.V == 0 || (pte_flags.R == 0 && pte_flags.W == 1)) {
        raise(page_fault_code, vaddr);
    } else if (pte_flags.R == 1 || pte_flags.X == 1) {
        if (op == MemoryOperation::Read || op == MemoryOperation::ReadModifyWrite) {
            if ((CSR[mstatus].MXR == 0 && pte_flags.R == 0) || (CSR[mstatus].MXR == 1 &&
pte_flags.X == 0 && pte_flags.R == 0)) {
                raise(page_fault_code, vaddr);
            }
        }
    }
}

```

```

    }
    if ((mode() == PrivilegeMode::U && pte_flags.U == 0) || (mode() ==
PrivilegeMode::M && CSR[mstatus].MPRV == 1 && CSR[mstatus].MPP == $bits(
PrivilegeMode::HS) && pte_flags.U == 1 && CSR[mstatus].SUM == 0) || (mode() ==
PrivilegeMode::S && pte_flags.U == 1 && CSR[mstatus].SUM == 0)) {
        raise(page_fault_code, vaddr);
    }
}
if (op == MemoryOperation::Write) || (op == MemoryOperation::ReadModifyWrite &&
(pte_flags.W == 0)) {
    raise(page_fault_code, vaddr);
} else if ((op == MemoryOperation::Fetch) && (pte_flags.X == 0)) {
    raise(page_fault_code, vaddr);
}
if (pte_flags.U == 0) {
    if (mode() == PrivilegeMode::U) {
        raise(page_fault_code, vaddr);
    }
} else {
    if (mode() == PrivilegeMode::S || (mode() == PrivilegeMode::M && CSR[mstatus]
.MPRV == 1 && CSR[mstatus].MPP == $bits(PrivilegeMode::S))) {
        if (op == MemoryOperation::Read || op == MemoryOperation::ReadModifyWrite) {
            if (CSR[mstatus].SUM == 0) {
                raise(page_fault_code, vaddr);
            }
        } else {
            raise(page_fault_code, vaddr);
        }
    }
}
raise(page_fault_code, vaddr) if ;
if ((pte_flags.A == 0) || pte_flags.D == 0) && ((op == MemoryOperation::Write) ||
(op == MemoryOperation::ReadModifyWrite)) {
    if (CSR[menvcfg].ADUE == 1'b1) {
        if (!pma_applies?(PmaAttribute::RsrvEventual, pte_addr, PTESIZE)) {
            raise(access_fault_code, vaddr);
        }
        if (!pma_applies?(PmaAttribute::HardwarePageTableWrite, pte_addr, PTESIZE)) {
            raise(access_fault_code, vaddr);
        }
        access_check<PTESIZE>(pte_addr, vaddr, MemoryOperation::Write,
access_fault_code);
        Boolean success;
        Bits<PTESIZE> updated_pte;
        if (pte_flags.D == 0 && op == MemoryOperation::Write) {
            updated_pte = pte | 0b11000000;
        } else {
            updated_pte = pte | 0b01000000;
        }
        if (PTESIZE == 32) {
            success = atomic_check_then_write_32(pte_addr, pte, updated_pte);
        } else if (PTESIZE == 64) {
            success = atomic_check_then_write_64(pte_addr, pte, updated_pte);
        } else {
            assert(false, "Unexpected PTESIZE");

```

```
    }
    if (!success) {
        i = i + 1;
    } else {
        return {(pte[PA_SIZE - 3:(i * VPN_SIZE) + 10] << 2), vaddr[11:0]};
    }
}
if ((CSR[menvcfg].ADUE == 1'b0) && implemented?(ExtensionName::Svade)) {
    raise(page_fault_code, vaddr);
}
}
return {(pte[PA_SIZE - 3:(i * VPN_SIZE) + 10] << 2), vaddr[11:0]};
} else {
    if (i == 0) {
        raise(page_fault_code, vaddr);
    }
    if (pte_flags.D == 1 || pte_flags.A == 1 || pte_flags.U == 1) {
        raise(page_fault_code, vaddr);
    }
    ppn = pte[PA_SIZE - 3:10] << 12;
}
}
```

6.22. pma_applies? (builtin)

Checks if *attr* is applied to the entire physical address region between [*paddr*, *paddr* + *len*).

Return Type	Boolean
Arguments	PmaAttribute attr, Bits<PHYS_ADDR_WIDTH> paddr, U32 len

6.23. pmp_check

Given a physical address and operation type, return whether or not the access is allowed by PMP.

Return Type	Boolean
Arguments	Bits<PHYS_ADDR_WIDTH> paddr, MemoryOperation type

```
PrivilegeMode mode = effective_ldst_mode();
PmpMatchResult match_result;
PmpCfg cfg;
(match_result, cfg = pmp_match<access_size>(paddr));
if (match_result == PmpMatchResult::FullMatch) {
    if (mode == PrivilegeMode::M && (cfg.L == 0)) {
        return true;
    }
    if (type == MemoryOperation::Write && (cfg.W == 0)) {
```

```
    return false;
} else if (type == MemoryOperation::Read && (cfg.R == 0)) {
    return false;
} else if (type == MemoryOperation::Fetch && (cfg.X == 0)) {
    return false;
}
} else if (match_result == PmpMatchResult::NoMatch) {
    if (mode == PrivilegeMode::M) {
        return true;
    } else {
        return false;
    }
} else {
    assert(match_result == PmpMatchResult::PartialMatch, "PMP matching logic error");
    return false;
}
return true;
```

6.24. pmp_match

Given a physical address, see if any PMP entry matches.

If there is a complete match, return the PmpCfg that guards the region. If there is no match or a partial match, report that result.

Return Type	PmpMatchResult, PmpCfg
Arguments	Bits<PHYS_ADDR_WIDTH> paddr

```
if (XLEN == 64) {
    return pmp_match_64<access_size>(paddr);
} else {
    return pmp_match_32<access_size>(paddr);
}
```

6.25. pmp_match_32

Given a physical address, see if any PMP entry matches.

If there is a complete match, return the PmpCfg that guards the region. If there is no match or a partial match, report that result.

Return Type	PmpMatchResult, PmpCfg
Arguments	Bits<PHYS_ADDR_WIDTH> paddr

```
Bits<12> pmpcfg0_addr = 0x3a0;
Bits<12> pmpaddr0_addr = 0x3b0;
```

```

for (U32 i = 0; i < NUM_PMP_ENTRIES; i++) {
    Bits<12> pmpcfg_idx = pmpcfg0_addr + (i / 4);
    Bits<6> shamt = (i % 4) * 8;
    PmpCfg cfg = ($bits(CSR[pmpcfg0_addr]) >> shamt)[7:0];
    Bits<12> pmpaddr_idx = pmpaddr0_addr + i;
    Bits<PHYS_ADDR_WIDTH> range_hi = 0;
    Bits<PHYS_ADDR_WIDTH> range_lo = 0;
    if (cfg.A == $bits(PmpCfg_A::TOR)) {
        if (i == 0) {
            range_lo = 0;
        } else {
            range_lo = (CSR[pmpaddr_idx - 1] << 2)[PHYS_ADDR_WIDTH - 1:0];
        }
        range_hi = (CSR[pmpaddr_idx] << 2)[PHYS_ADDR_WIDTH - 1:0];
    } else if (cfg.A == $bits(PmpCfg_A::NAPOT)) {
        Bits<PHYS_ADDR_WIDTH - 2> pmpaddr_value = CSR[pmpaddr_idx].sw_read()[
PHYS_ADDR_WIDTH - 3:0];
        Bits<PHYS_ADDR_WIDTH - 2> mask = pmpaddr_value ^ (pmpaddr_value + 1);
        range_lo = (pmpaddr_value & ~mask) << 2;
        Bits<PHYS_ADDR_WIDTH - 2> len = mask + 1;
        range_hi = pmpaddr_value & ~mask + len << 2;    } else if (cfg.A == $bits(
PmpCfg_A::NA4) {
        range_lo = ($bits(CSR[pmpaddr_idx]) << 2)[PHYS_ADDR_WIDTH - 1:0];
        range_hi = range_lo + 4;
    }
    if ((paddr >= range_lo) && paddr + (access_size / 8 < range_hi)) {
        return PmpMatchResult::FullMatch, cfg;
    } else if (! {
        return PmpMatchResult::PartialMatch, -;
    }
}
return PmpMatchResult::NoMatch, -;

```

6.26. pmp_match_64

Given a physical address, see if any PMP entry matches.

If there is a complete match, return the PmpCfg that guards the region. If there is no match or a partial match, report that result.

Return Type	PmpMatchResult, PmpCfg
Arguments	Bits<PHYS_ADDR_WIDTH> paddr

```

Bits<12> pmpcfg0_addr = 0x3a0;
Bits<12> pmpaddr0_addr = 0x3b0;
for (U32 i = 0; i < NUM_PMP_ENTRIES; i++) {
    Bits<12> pmpcfg_idx = pmpcfg0_addr + (i / 8) * 2;
    Bits<6> shamt = (i % 8) * 8;
    PmpCfg cfg = ($bits(CSR[pmpcfg0_addr]) >> shamt)[7:0];
    Bits<12> pmpaddr_idx = pmpaddr0_addr + i;
    Bits<PHYS_ADDR_WIDTH> range_hi = 0;

```

```

Bits<PHYS_ADDR_WIDTH> range_lo = 0;
if (cfg.A == $bits(PmpCfg_A::TOR)) {
    if (i == 0) {
        range_lo = 0;
    } else {
        range_lo = (CSR[pmpaddr_idx - 1] << 2)[PHYS_ADDR_WIDTH - 1:0];
    }
    range_hi = (CSR[pmpaddr_idx] << 2)[PHYS_ADDR_WIDTH - 1:0];
} else if (cfg.A == $bits(PmpCfg_A::NAPOT)) {
    Bits<PHYS_ADDR_WIDTH - 2> pmpaddr_value = CSR[pmpaddr_idx].sw_read()[
PHYS_ADDR_WIDTH - 3:0];
    Bits<PHYS_ADDR_WIDTH - 2> mask = pmpaddr_value ^ (pmpaddr_value + 1);
    range_lo = (pmpaddr_value & ~mask) << 2;
    Bits<PHYS_ADDR_WIDTH - 2> len = mask + 1;
    range_hi = pmpaddr_value & ~mask + len << 2; } else if (cfg.A == $bits(
PmpCfg_A::NA4 {
    range_lo = (CSR[pmpaddr_idx] << 2)[PHYS_ADDR_WIDTH - 1:0];
    range_hi = range_lo + 4;
}
if ((paddr >= range_lo) && paddr + (access_size / 8 < range_hi)) {
    return PmpMatchResult::FullMatch, cfg;
} else if (! {
    return PmpMatchResult::PartialMatch, -;
}
}
return PmpMatchResult::NoMatch, -;

```

6.27. raise

Raise synchronous exception number exception_code.

Return Type	void
Arguments	ExceptionCode exception_code, XReg tval

```

PrivilegeMode handling_mode = exception_handling_mode(exception_code);
if (handling_mode == PrivilegeMode::M) {
    CSR[mepc].PC = PC;
    if (!mtval_readonly?()) {
        CSR[mtval].VALUE = mtval_for(exception_code, tval);
    }
    PC = {CSR[mtvec].BASE, 2'b00};
    CSR[mcause].INT = 1'b0;
    CSR[mcause].CAUSE = $bits(exception_code);
} else if (implemented?(ExtensionName::S) && (handling_mode == PrivilegeMode::HS)) {
    CSR[sepc].PC = PC;
    if (!stval_readonly?()) {
        CSR[stval].VALUE = stval_for(exception_code, tval);
    }
    PC = {CSR[stvec].BASE, 2'b00};
    CSR[scause].INT = 1'b0;
    CSR[scause].CODE = $bits(exception_code);
}

```

```

} else if (implemented?(ExtensionName::H) && (handling_mode == PrivilegeMode::VS)) {
  CSR[vsepc].PC = PC;
  if (!vstval_readonly?()) {
    CSR[vstval].VALUE = vstval_for(exception_code, tval);
  }
  PC = {CSR[vstvec].BASE, 2'b00};
  CSR[vscause].INT = 1'b0;
  CSR[vscause].CODE = $bits(exception_code);
}
set_mode(handling_mode);
abort_current_instruction();

```

6.28. read_memory

Read from virtual memory

Return Type	Bits<len>
Arguments	XReg virtual_address

```

Boolean aligned = is_naturally_aligned<XLEN, len>(virtual_address);
if (aligned) {
  return read_memory_aligned<len>(virtual_address);
} else if (!aligned && !MISALIGNED_LDST) {
  raise(ExceptionCode::LoadAddressMisaligned, virtual_address);
} else {
  if (MISALIGNED_SPLIT_STRATEGY == "by_byte") {
    Bits<len> result = 0;
    for (U32 i = 0; i <= len; i++) {
      result = result | (read_memory_aligned<8>(virtual_address + i) << (8 * i));
    }
    return result;
  } else if (MISALIGNED_SPLIT_STRATEGY == "custom") {
    unpredictable("An implementation is free to break a misaligned access any way,
leading to unpredictable behavior when any part of the misaligned access causes an
exception");
  }
}

```

6.29. read_memory_aligned

Read from virtual memory using a known aligned address.

Return Type	Bits<len>
Arguments	XReg virtual_address

```

XReg physical_address;
physical_address = (CSR[misa].S == 1) ? translate(virtual_address, MemoryOperation

```

```

::Read) : virtual_address;
access_check<len>(physical_address, virtual_address, MemoryOperation::Read,
ExceptionCode::LoadAccessFault);
return read_physical_memory<len>(physical_address);

```

6.30. read_physical_memory (builtin)

Read from physical memory.

Return Type	Bits<len>
Arguments	XReg paddr

6.31. set_mode

Set the current privilege mode to new_mode

Return Type	void
Arguments	PrivilegeMode new_mode

```

if (new_mode != current_mode) {
    notify_mode_change(new_mode, current_mode);
    current_mode = new_mode;
}

```

6.32. sext

Sign extend value starting at first_extended_bit.

Bits [XLEN-1:first_extended_bit] of the return value should get the value of bit (first_extended_bit - 1).

Return Type	XReg
Arguments	XReg value, XReg first_extended_bit

```

if (first_extended_bit == XLEN) {
    return value;
} else {
    Bits<1> sign = value[first_extended_bit - 1];
    for (U32 i = XLEN - 1; i >= first_extended_bit; i--) {
        value[i] = sign;
    }
    return value;
}

```

6.33. stval_for

Given an exception code and a **legal** non-zero value for stval, returns the value to be written in stval considering implementation options

Return Type	XReg
Arguments	ExceptionCode exception_code, XReg tval

```

if (exception_code == ExceptionCode::Breakpoint) {
    return REPORT_VA_IN_STVAL_ON_BREAKPOINT ? tval : 0;
} else if (exception_code == ExceptionCode::LoadAddressMisaligned) {
    return REPORT_VA_IN_STVAL_ON_LOAD_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoAddressMisaligned) {
    return REPORT_VA_IN_STVAL_ON_STORE_AMO_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionAddressMisaligned) {
    return REPORT_VA_IN_STVAL_ON_INSTRUCTION_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::LoadAccessFault) {
    return REPORT_VA_IN_STVAL_ON_LOAD_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoAccessFault) {
    return REPORT_VA_IN_STVAL_ON_STORE_AMO_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionAccessFault) {
    return REPORT_VA_IN_STVAL_ON_INSTRUCTION_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::LoadPageFault) {
    return REPORT_VA_IN_STVAL_ON_LOAD_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoPageFault) {
    return REPORT_VA_IN_STVAL_ON_STORE_AMO_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionPageFault) {
    return REPORT_VA_IN_STVAL_ON_INSTRUCTION_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::IllegalInstruction) {
    return REPORT_ENCODING_IN_STVAL_ON_ILLEGAL_INSTRUCTION ? tval : 0;
} else {
    return 0;
}

```

6.34. stval_readonly?

Returns whether or not CSR[stval] is read-only based on implementation options

Return Type	Boolean
Arguments	

```

if (implemented?(ExtensionName::S)) {
    return !(REPORT_VA_IN_STVAL_ON_BREAKPOINT || REPORT_VA_IN_STVAL_ON_LOAD_MISALIGNED
    || REPORT_VA_IN_STVAL_ON_STORE_AMO_MISALIGNED ||
    REPORT_VA_IN_STVAL_ON_INSTRUCTION_MISALIGNED ||
    REPORT_VA_IN_STVAL_ON_LOAD_ACCESS_FAULT ||
    REPORT_VA_IN_STVAL_ON_STORE_AMO_ACCESS_FAULT ||
    REPORT_VA_IN_STVAL_ON_INSTRUCTION_ACCESS_FAULT ||
    REPORT_VA_IN_STVAL_ON_LOAD_PAGE_FAULT || REPORT_VA_IN_STVAL_ON_STORE_AMO_PAGE_FAULT ||

```

```

REPORT_VA_IN_STVAL_ON_INSTRUCTION_PAGE_FAULT ||
REPORT_ENCODING_IN_STVAL_ON_ILLEGAL_INSTRUCTION ||
REPORT_CAUSE_IN_STVAL_ON_SOFTWARE_CHECK || implemented?(ExtensionName::Sdext));
} else {
    return true;
}

```

6.35. translate

Translate a virtual address for a load, returning a physical address. May raise a Page Fault or Access Fault.

The final physical address is **not** access checked (for PMP, PMA, etc., violations).

Return Type	Bits<PHYS_ADDR_WIDTH>
Arguments	XReg vaddr, MemoryOperation op

```

SatpMode translation_mode = current_translation_mode();
XReg paddr;
if (implemented?(ExtensionName::H) && virtual_mode?()) {
    return 0;
} else {
    if (translation_mode == SatpMode::Bare) {
        paddr = vaddr;
    } else if (implemented?(ExtensionName::Sv32) && translation_mode == SatpMode::Sv32) {
        paddr = page_walk<32, 34, 32, 2>(vaddr, op);
    } else if (implemented?(ExtensionName::Sv39) && translation_mode == SatpMode::Sv39) {
        paddr = page_walk<39, 56, 64, 3>(vaddr, op);
    } else if (implemented?(ExtensionName::Sv48) && translation_mode == SatpMode::Sv48) {
        paddr = page_walk<48, 56, 64, 4>(vaddr, op);
    } else if (implemented?(ExtensionName::Sv57) && translation_mode == SatpMode::Sv57) {
        paddr = page_walk<57, 56, 64, 5>(vaddr, op);
    }
}
return paddr;

```

6.36. unpredictable (builtin)

Indicate that the hart has reached a state that is unpredictable because the RISC-V spec allows multiple behaviors. Generally, this will be a fatal condition to any emulation, since it is unclear what to do next.

The single argument *why* is a string describing why the hart entered an unpredictable state.

Return Type	void
Arguments	String why

6.37. virtual_mode?

Returns True if the current mode is virtual (VS or VU).

Return Type	Boolean
Arguments	

```
return (mode() == PrivilegeMode::VS) || (mode() == PrivilegeMode::VU);
```

6.38. vstval_for

Given an exception code and a **legal** non-zero value for vstval, returns the value to be written in vstval considering implementation options

Return Type	XReg
Arguments	ExceptionCode exception_code, XReg tval

```
if (exception_code == ExceptionCode::Breakpoint) {
    return REPORT_VA_IN_VSTVAL_ON_BREAKPOINT ? tval : 0;
} else if (exception_code == ExceptionCode::LoadAddressMisaligned) {
    return REPORT_VA_IN_VSTVAL_ON_LOAD_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoAddressMisaligned) {
    return REPORT_VA_IN_VSTVAL_ON_STORE_AMO_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionAddressMisaligned) {
    return REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_MISALIGNED ? tval : 0;
} else if (exception_code == ExceptionCode::LoadAccessFault) {
    return REPORT_VA_IN_VSTVAL_ON_LOAD_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoAccessFault) {
    return REPORT_VA_IN_VSTVAL_ON_STORE_AMO_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionAccessFault) {
    return REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_ACCESS_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::LoadPageFault) {
    return REPORT_VA_IN_VSTVAL_ON_LOAD_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::StoreAmoPageFault) {
    return REPORT_VA_IN_VSTVAL_ON_STORE_AMO_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::InstructionPageFault) {
    return REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_PAGE_FAULT ? tval : 0;
} else if (exception_code == ExceptionCode::IllegalInstruction) {
    return REPORT_ENCODING_IN_VSTVAL_ON_ILLEGAL_INSTRUCTION ? tval : 0;
} else {
    return 0;
}
```

6.39. vstval_readonly?

Returns whether or not CSR[vstval] is read-only based on implementation options

Return Type	Boolean
Arguments	

```

if (implemented?(ExtensionName::H)) {
    return !(REPORT_VA_IN_VSTVAL_ON_BREAKPOINT || REPORT_VA_IN_VSTVAL_ON_LOAD_MISALIGNED
    || REPORT_VA_IN_VSTVAL_ON_STORE_AMO_MISALIGNED ||
    REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_MISALIGNED ||
    REPORT_VA_IN_VSTVAL_ON_LOAD_ACCESS_FAULT ||
    REPORT_VA_IN_VSTVAL_ON_STORE_AMO_ACCESS_FAULT ||
    REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_ACCESS_FAULT ||
    REPORT_VA_IN_VSTVAL_ON_LOAD_PAGE_FAULT || REPORT_VA_IN_VSTVAL_ON_STORE_AMO_PAGE_FAULT
    || REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_PAGE_FAULT ||
    REPORT_ENCODING_IN_VSTVAL_ON_ILLEGAL_INSTRUCTION ||
    REPORT_CAUSE_IN_VSTVAL_ON_SOFTWARE_CHECK || implemented?(ExtensionName::Sdext));
} else {
    return true;
}

```

6.40. write_memory

Write to virtual memory

Return Type	void
Arguments	XReg virtual_address, Bits<len> value

```

Boolean aligned = is_naturally_aligned<XLEN, len>(virtual_address);
if (aligned) {
    write_memory_aligned<len>(virtual_address, value);
} else if (!aligned && !MISALIGNED_LDST) {
    raise(ExceptionCode::StoreAmoAddressMisaligned, virtual_address);
} else {
    for (U32 i = 0; i <= len; i++) {
        write_memory_aligned<8>(virtual_address + i, (value >> (8 * i))[7:0]);
    }
}

```

6.41. write_memory_aligned

Write to virtual memory using a known aligned address.

Return Type	void
-------------	------

Arguments	XReg virtual_address, Bits<len> value
------------------	---------------------------------------

```

XReg physical_address;
physical_address = (CSR[misa].S == 1) ? translate(virtual_address, MemoryOperation::Write) : virtual_address;
access_check<len>(physical_address, virtual_address, MemoryOperation::Write, ExceptionCode::StoreAmoAccessFault);
write_physical_memory<len>(physical_address, value);

```

6.42. write_physical_memory (builtin)

Write to physical memory.

Return Type	void
Arguments	XReg paddr, Bits<len> value

6.43. xlen

Returns the effective XLEN for the current privilege mode.

Return Type	Bits<8>
Arguments	

```

if (XLEN == 32) {
    return 32;
}
if (mode() == PrivilegeMode::M) {
    if (CSR[misa].MXL == $bits(XRegWidth::XLEN32)) {
        return 32;
    } else if (CSR[misa].MXL == $bits(XRegWidth::XLEN64)) {
        return 64;
    }
} else if (implemented?(ExtensionName::S) && mode() == PrivilegeMode::S) {
    if (CSR[mstatus].SXL == $bits(XRegWidth::XLEN32)) {
        return 32;
    } else if (CSR[mstatus].SXL == $bits(XRegWidth::XLEN64)) {
        return 64;
    }
} else if (implemented?(ExtensionName::U) && mode() == PrivilegeMode::U) {
    if (CSR[mstatus].UXL == $bits(XRegWidth::XLEN32)) {
        return 32;
    } else if (CSR[mstatus].UXL == $bits(XRegWidth::XLEN64)) {
        return 64;
    }
} else if (implemented?(ExtensionName::H) && mode() == PrivilegeMode::VS) {
    if (CSR[hstatus].VSXL == $bits(XRegWidth::XLEN32)) {
        return 32;
    }
}

```

```
} else if (CSR[hstatus].VSXL == $bits(XRegWidth::XLEN64)) {  
    return 64;  
}  
} else if (implemented?(ExtensionName::H) && mode() == PrivilegeMode::VU) {  
    if (CSR[vsstatus].UXL == $bits(XRegWidth::XLEN32)) {  
        return 32;  
    } else if (CSR[vsstatus].UXL == $bits(XRegWidth::XLEN64)) {  
        return 64;  
    }  
}
```

DRAFT