

DRAFT

Qualcomm uC extensions

Version 0.3,

Table of Contents

Licensing and Acknowledgements	1
Copyright and license information	2
Acknowledgements	3
Versions	4
1. Extension description	6
1.1. Sub-extensions	7
1.1.1. Xqcia	7
1.1.2. Xqciac	7
1.1.3. Xqcibi	7
1.1.4. Xqcibm	7
1.1.5. Xqcicli	7
1.1.6. Xqcicm	7
1.1.7. Xqcics	7
1.1.8. Xqcicsr	7
1.1.9. Xqciint	7
1.1.10. Xqcilb	8
1.1.11. Xqcili	8
1.1.12. Xqcilia	8
1.1.13. Xqcilo	8
1.1.14. Xqcilsm	8
1.1.15. Xqcisls	8
2. Instruction summary	9
2.1. Instructions by sub-extension	15
3. CSR summary	21
4. Csrs (in alphabetical order)	22
4.1. qc.mclcie0	23
4.1.1. Attributes	23
4.1.2. Format	23
4.1.3. Field Summary	23
4.1.4. Fields	24
IRQ0	24
IRQ1	25
IRQ2	25
IRQ3	25
IRQ4	26
IRQ5	26
IRQ6	26
IRQ7	27
IRQ8	27

IRQ9	27
IRQ10	28
IRQ11	28
IRQ12	28
IRQ13	29
IRQ14	29
IRQ15	29
IRQ16	30
IRQ17	30
IRQ18	30
IRQ19	31
IRQ20	31
IRQ21	31
IRQ22	32
IRQ23	32
IRQ24	32
IRQ25	33
IRQ26	33
IRQ27	33
IRQ28	34
IRQ29	34
IRQ30	34
IRQ31	35
4.2. qc.mclciel	36
4.2.1. Attributes	36
4.2.2. Format	36
4.2.3. Field Summary	36
4.2.4. Fields	37
IRQ32	37
IRQ33	38
IRQ34	38
IRQ35	38
IRQ36	39
IRQ37	39
IRQ38	39
IRQ39	40
IRQ40	40
IRQ41	40
IRQ42	41

IRQ43	41
IRQ44	41
IRQ45	42
IRQ46	42
IRQ47	42
IRQ48	43
IRQ49	43
IRQ50	43
IRQ51	44
IRQ52	44
IRQ53	44
IRQ54	45
IRQ55	45
IRQ56	45
IRQ57	46
IRQ58	46
IRQ59	46
IRQ60	47
IRQ61	47
IRQ62	47
IRQ63	48
4.3. qc.mclcie2	49
4.3.1. Attributes	49
4.3.2. Format	49
4.3.3. Field Summary	49
4.3.4. Fields	50
IRQ64	50
IRQ65	51
IRQ66	51
IRQ67	51
IRQ68	52
IRQ69	52
IRQ70	52
IRQ71	53
IRQ72	53
IRQ73	53
IRQ74	54
IRQ75	54
IRQ76	54

IRQ77	55
IRQ78	55
IRQ79	55
IRQ80	56
IRQ81	56
IRQ82	56
IRQ83	57
IRQ84	57
IRQ85	57
IRQ86	58
IRQ87	58
IRQ88	58
IRQ89	59
IRQ90	59
IRQ91	59
IRQ92	60
IRQ93	60
IRQ94	60
IRQ95	61
4.4. qc.mclicie3	62
4.4.1. Attributes	62
4.4.2. Format	62
4.4.3. Field Summary	62
4.4.4. Fields	63
IRQ96	63
IRQ97	64
IRQ98	64
IRQ99	64
IRQ100	65
IRQ101	65
IRQ102	65
IRQ103	66
IRQ104	66
IRQ105	66
IRQ106	67
IRQ107	67
IRQ108	67
IRQ109	68
IRQ110	68

IRQ111	68
IRQ112	69
IRQ113	69
IRQ114	69
IRQ115	70
IRQ116	70
IRQ117	70
IRQ118	71
IRQ119	71
IRQ120	71
IRQ121	72
IRQ122	72
IRQ123	72
IRQ124	73
IRQ125	73
IRQ126	73
IRQ127	74
4.5. qc.mclcie4	75
4.5.1. Attributes	75
4.5.2. Format	75
4.5.3. Field Summary	75
4.5.4. Fields	76
IRQ128	76
IRQ129	77
IRQ130	77
IRQ131	77
IRQ132	78
IRQ133	78
IRQ134	78
IRQ135	79
IRQ136	79
IRQ137	79
IRQ138	80
IRQ139	80
IRQ140	80
IRQ141	81
IRQ142	81
IRQ143	81
IRQ144	82

IRQ145.....	82
IRQ146.....	82
IRQ147.....	83
IRQ148.....	83
IRQ149.....	83
IRQ150.....	84
IRQ151.....	84
IRQ152.....	84
IRQ153.....	85
IRQ154.....	85
IRQ155.....	85
IRQ156.....	86
IRQ157.....	86
IRQ158.....	86
IRQ159.....	87
4.6. qc.mclicie5.....	88
4.6.1. Attributes	88
4.6.2. Format	88
4.6.3. Field Summary	88
4.6.4. Fields	89
IRQ160.....	89
IRQ161.....	90
IRQ162.....	90
IRQ163.....	90
IRQ164	91
IRQ165	91
IRQ166	91
IRQ167.....	92
IRQ168.....	92
IRQ169.....	92
IRQ170.....	93
IRQ171.....	93
IRQ172.....	93
IRQ173.....	94
IRQ174.....	94
IRQ175.....	94
IRQ176.....	95
IRQ177.....	95
IRQ178.....	95

IRQ179	96
IRQ180	96
IRQ181	96
IRQ182	97
IRQ183	97
IRQ184	97
IRQ185	98
IRQ186	98
IRQ187	98
IRQ188	99
IRQ189	99
IRQ190	99
IRQ191	100
4.7. qc.mclcie6	101
4.7.1. Attributes	101
4.7.2. Format	101
4.7.3. Field Summary	101
4.7.4. Fields	102
IRQ192	102
IRQ193	103
IRQ194	103
IRQ195	103
IRQ196	104
IRQ197	104
IRQ198	104
IRQ199	105
IRQ200	105
IRQ201	105
IRQ202	106
IRQ203	106
IRQ204	106
IRQ205	107
IRQ206	107
IRQ207	107
IRQ208	108
IRQ209	108
IRQ210	108
IRQ211	109
IRQ212	109

IRQ213	109
IRQ214	110
IRQ215	110
IRQ216	110
IRQ217	111
IRQ218	111
IRQ219	111
IRQ220	112
IRQ221	112
IRQ222	112
IRQ223	113
4.8. qc.mclcie7	114
4.8.1. Attributes	114
4.8.2. Format	114
4.8.3. Field Summary	114
4.8.4. Fields	115
IRQ224	115
IRQ225	116
IRQ226	116
IRQ227	116
IRQ228	117
IRQ229	117
IRQ230	117
IRQ231	118
IRQ232	118
IRQ233	118
IRQ234	119
IRQ235	119
IRQ236	119
IRQ237	120
IRQ238	120
IRQ239	120
IRQ240	121
IRQ241	121
IRQ242	121
IRQ243	122
IRQ244	122
IRQ245	122
IRQ246	123

IRQ247	123
IRQ248	123
IRQ249	124
IRQ250	124
IRQ251	124
IRQ252	125
IRQ253	125
IRQ254	125
IRQ255	126
4.9. qc.mclcip0	127
4.9.1. Attributes	127
4.9.2. Format	127
4.9.3. Field Summary	127
4.9.4. Fields	128
IRQ0	128
IRQ1	129
IRQ2	129
IRQ3	129
IRQ4	130
IRQ5	130
IRQ6	130
IRQ7	131
IRQ8	131
IRQ9	131
IRQ10	132
IRQ11	132
IRQ12	132
IRQ13	133
IRQ14	133
IRQ15	133
IRQ16	134
IRQ17	134
IRQ18	134
IRQ19	135
IRQ20	135
IRQ21	135
IRQ22	136
IRQ23	136
IRQ24	136

IRQ25	137
IRQ26	137
IRQ27	137
IRQ28	138
IRQ29	138
IRQ30	138
IRQ31	139
4.10. qc.mclcip1	140
4.10.1. Attributes	140
4.10.2. Format	140
4.10.3. Field Summary	140
4.10.4. Fields	141
IRQ32	141
IRQ33	142
IRQ34	142
IRQ35	142
IRQ36	143
IRQ37	143
IRQ38	143
IRQ39	144
IRQ40	144
IRQ41	144
IRQ42	145
IRQ43	145
IRQ44	145
IRQ45	146
IRQ46	146
IRQ47	146
IRQ48	147
IRQ49	147
IRQ50	147
IRQ51	148
IRQ52	148
IRQ53	148
IRQ54	149
IRQ55	149
IRQ56	149
IRQ57	150
IRQ58	150

IRQ59	150
IRQ60	151
IRQ61	151
IRQ62	151
IRQ63	152
4.11. qc.mclcip2	153
4.11.1. Attributes	153
4.11.2. Format	153
4.11.3. Field Summary	153
4.11.4. Fields	154
IRQ64	154
IRQ65	155
IRQ66	155
IRQ67	155
IRQ68	156
IRQ69	156
IRQ70	156
IRQ71	157
IRQ72	157
IRQ73	157
IRQ74	158
IRQ75	158
IRQ76	158
IRQ77	159
IRQ78	159
IRQ79	159
IRQ80	160
IRQ81	160
IRQ82	160
IRQ83	161
IRQ84	161
IRQ85	161
IRQ86	162
IRQ87	162
IRQ88	162
IRQ89	163
IRQ90	163
IRQ91	163
IRQ92	164

IRQ93	164
IRQ94	164
IRQ95	165
4.12. qc.mclcip3	166
4.12.1. Attributes	166
4.12.2. Format	166
4.12.3. Field Summary	166
4.12.4. Fields	167
IRQ96	167
IRQ97	168
IRQ98	168
IRQ99	168
IRQ100	169
IRQ101	169
IRQ102	169
IRQ103	170
IRQ104	170
IRQ105	170
IRQ106	171
IRQ107	171
IRQ108	171
IRQ109	172
IRQ110	172
IRQ111	172
IRQ112	173
IRQ113	173
IRQ114	173
IRQ115	174
IRQ116	174
IRQ117	174
IRQ118	175
IRQ119	175
IRQ120	175
IRQ121	176
IRQ122	176
IRQ123	176
IRQ124	177
IRQ125	177
IRQ126	177

IRQ127	178
4.13. qc.mclcip4	179
4.13.1. Attributes	179
4.13.2. Format	179
4.13.3. Field Summary	179
4.13.4. Fields	180
IRQ128	180
IRQ129	181
IRQ130	181
IRQ131	181
IRQ132	182
IRQ133	182
IRQ134	182
IRQ135	183
IRQ136	183
IRQ137	183
IRQ138	184
IRQ139	184
IRQ140	184
IRQ141	185
IRQ142	185
IRQ143	185
IRQ144	186
IRQ145	186
IRQ146	186
IRQ147	187
IRQ148	187
IRQ149	187
IRQ150	188
IRQ151	188
IRQ152	188
IRQ153	189
IRQ154	189
IRQ155	189
IRQ156	190
IRQ157	190
IRQ158	190
IRQ159	191
4.14. qc.mclcip5	192

4.14.1. Attributes	192
4.14.2. Format	192
4.14.3. Field Summary	192
4.14.4. Fields	193
IRQ160	193
IRQ161	194
IRQ162	194
IRQ163	194
IRQ164	195
IRQ165	195
IRQ166	195
IRQ167	196
IRQ168	196
IRQ169	196
IRQ170	197
IRQ171	197
IRQ172	197
IRQ173	198
IRQ174	198
IRQ175	198
IRQ176	199
IRQ177	199
IRQ178	199
IRQ179	200
IRQ180	200
IRQ181	200
IRQ182	201
IRQ183	201
IRQ184	201
IRQ185	202
IRQ186	202
IRQ187	202
IRQ188	203
IRQ189	203
IRQ190	203
IRQ191	204
4.15. qc.mclcip6	205
4.15.1. Attributes	205
4.15.2. Format	205

4.15.3. Field Summary	205
4.15.4. Fields	206
IRQ192	206
IRQ193	207
IRQ194	207
IRQ195	207
IRQ196	208
IRQ197	208
IRQ198	208
IRQ199	209
IRQ200	209
IRQ201	209
IRQ202	210
IRQ203	210
IRQ204	210
IRQ205	211
IRQ206	211
IRQ207	211
IRQ208	212
IRQ209	212
IRQ210	212
IRQ211	213
IRQ212	213
IRQ213	213
IRQ214	214
IRQ215	214
IRQ216	214
IRQ217	215
IRQ218	215
IRQ219	215
IRQ220	216
IRQ221	216
IRQ222	216
IRQ223	217
4.16. qc.mclcip7	218
4.16.1. Attributes	218
4.16.2. Format	218
4.16.3. Field Summary	218
4.16.4. Fields	219

IRQ224	219
IRQ225	220
IRQ226	220
IRQ227	220
IRQ228	221
IRQ229	221
IRQ230	221
IRQ231	222
IRQ232	222
IRQ233	222
IRQ234	223
IRQ235	223
IRQ236	223
IRQ237	224
IRQ238	224
IRQ239	224
IRQ240	225
IRQ241	225
IRQ242	225
IRQ243	226
IRQ244	226
IRQ245	226
IRQ246	227
IRQ247	227
IRQ248	227
IRQ249	228
IRQ250	228
IRQ251	228
IRQ252	229
IRQ253	229
IRQ254	229
IRQ255	230

5. Instructions (in alphabetical order)	231
5.1. qc.addsat	231
5.2. qc.addusat	233
5.3. qc.beqi	234
5.4. qc.bgei	235
5.5. qc.bgeui	236
5.6. qc.bltn	237

5.7. qc.bltoi	238
5.8. qc.bnei	239
5.9. qc.brev32	240
5.10. qc.c.bexti	241
5.11. qc.c.bseti	242
5.12. qc.c.clrint	243
5.13. qc.c.di	244
5.14. qc.c.dir	245
5.15. qc.c.ei	246
5.16. qc.c.eir	247
5.17. qc.c.extu	248
5.18. qc.c.mienter.nest	249
5.19. qc.c.mienter	251
5.20. qc.c.mileaveret	253
5.21. qc.c.muladdi	255
5.22. qc.c.mveqz	256
5.23. qc.c.setint	257
5.24. qc.clo	258
5.25. qc.clrinti	259
5.26. qc.compress2	260
5.27. qc.compress3	261
5.28. qc.csrrwr	262
5.29. qc.csrrwri	263
5.30. qc.cto	264
5.31. qc.e.addai	265
5.32. qc.e.addi	266
5.33. qc.e.andai	267
5.34. qc.e.andi	268
5.35. qc.e.beqi	269
5.36. qc.e.bgei	270
5.37. qc.e.bgeui	271
5.38. qc.e.blti	272
5.39. qc.e.bltoi	273
5.40. qc.e.bnei	274
5.41. qc.e.j	275
5.42. qc.e.jal	276
5.43. qc.e.lb	277
5.44. qc.e.lbu	278
5.45. qc.e.lh	279
5.46. qc.e.lhu	280
5.47. qc.e.li	281

5.48. qc.e.lw	282
5.49. qc.e.orai	283
5.50. qc.e.ori	284
5.51. qc.e.sb	285
5.52. qc.e.sh	286
5.53. qc.e.sw	287
5.54. qc.e.xorai	288
5.55. qc.e.xori	289
5.56. qc.expand2	290
5.57. qc.expand3	291
5.58. qc.ext	292
5.59. qc.extd	293
5.60. qc.extdpr	294
5.61. qc.extdprh	295
5.62. qc.extdr	296
5.63. qc.extdu	297
5.64. qc.extdupr	298
5.65. qc.extduprh	299
5.66. qc.extdur	300
5.67. qc.extu	301
5.68. qc.insb	302
5.69. qc.insbh	303
5.70. qc.insbhr	304
5.71. qc.insbi	305
5.72. qc.insbpr	306
5.73. qc.insbprh	307
5.74. qc.insbr	308
5.75. qc.insbri	309
5.76. qc.li	310
5.77. qc.lieq	311
5.78. qc.lieqi	312
5.79. qc.lige	313
5.80. qc.ligei	314
5.81. qc.ligeu	315
5.82. qc.ligeui	316
5.83. qc.lilt	317
5.84. qc.lilti	318
5.85. qc.liltu	319
5.86. qc.liltui	320
5.87. qc.line	321
5.88. qc.linei	322

5.89. qc.lrb	323
5.90. qc.lrbu	324
5.91. qc.lrh	325
5.92. qc.lrhu	326
5.93. qc.lrw	327
5.94. qc.lwm	328
5.95. qc.lwmi	329
5.96. qc.muladdi	330
5.97. qc.mveq	331
5.98. qc.mveqi	332
5.99. qc.mvge	333
5.100. qc.mvgei	334
5.101. qc.mvgeu	335
5.102. qc.mvgeui	336
5.103. qc.mvlt	337
5.104. qc.mvlti	338
5.105. qc.mvltu	339
5.106. qc.mvltui	340
5.107. qc.mvne	341
5.108. qc.mvnei	342
5.109. qc.norm	343
5.110. qc.normeu	344
5.111. qc.normu	345
5.112. qc.selecteqi	346
5.113. qc.selectieq	347
5.114. qc.selectieqi	348
5.115. qc.selectiieq	349
5.116. qc.selectiine	350
5.117. qc.selectine	351
5.118. qc.selectinei	352
5.119. qc.selectnei	353
5.120. qc.setinti	354
5.121. qc.setwm	355
5.122. qc.setwmi	356
5.123. qc.shladd	357
5.124. qc.slasat	358
5.125. qc.sllsat	359
5.126. qc.srb	360
5.127. qc.srh	361
5.128. qc.srw	362
5.129. qc.subsat	363

5.130. qc.subusat	364
5.131. qc.swm	365
5.132. qc.swmi	366
5.133. qc.wrap	367
5.134. qc.wrapi	368
6. IDL Functions	369
6.1. abort_current_instruction (builtin)	369
6.2. access_check	369
6.3. assert (builtin)	369
6.4. atomic_check_then_write_32 (builtin)	369
6.5. atomic_check_then_write_64 (builtin)	370
6.6. current_translation_mode	370
6.7. effective_ldst_mode	371
6.8. exception_handling_mode	371
6.9. highest_set_bit	372
6.10. ialign	372
6.11. implemented? (builtin)	372
6.12. in_naturally_aligned_region?	373
6.13. is_naturally_aligned	373
6.14. jump_halfword	373
6.15. lowest_set_bit	374
6.16. misaligned_is_atomic?	374
6.17. mode	374
6.18. mpv	375
6.19. mtval_for	375
6.20. mtval_readonly?	376
6.21. notify_mode_change (builtin)	376
6.22. page_walk	376
6.23. pma_applies? (builtin)	379
6.24. pmp_check	379
6.25. pmp_match	380
6.26. pmp_match_32	380
6.27. pmp_match_64	381
6.28. raise	382
6.29. read_memory	383
6.30. read_memory_aligned	383
6.31. read_physical_memory (builtin)	384
6.32. set_mode	384
6.33. sext	384
6.34. stval_for	385
6.35. stval_readonly?	385

6.36. translate	386
6.37. unpredictable (builtin)	387
6.38. virtual_mode?	387
6.39. vstval_for	387
6.40. vstval_readonly?	388
6.41. write_memory	388
6.42. write_memory_aligned	389
6.43. write_physical_memory (builtin)	389
6.44. xlen	390

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.)

Contributors to version 0.2 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.)

Contributors to version 0.3 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

Version

0.2

State

development

Changes

- Splits Xqci into sub-extensions based on instruction type
- Removed shXadd instructions in favor of generic shladd

Version

0.3

State

development

Changes

- Rename extension and sub extensions to match toolchain guidelines
- Rename mnemonics to match toolchain guidelines
- Ensure every instruction belongs to at least one sub extension

Implies

- Xqcia (0.1)
- Xqciac (0.1)
- Xqcibi (0.1)
- Xqcibm (0.1)
- Xqicli (0.1)
- Xqicm (0.1)
- Xqcics (0.1)
- Xqcicsr (0.1)
- Xqciint (0.1)
- Xqcilb (0.1)
- Xqcili (0.1)
- Xqcilia (0.1)

- Xqcilo (0.1)
- Xqcilsm (0.1)
- Xqcisls (0.1)

DRAFT

Chapter 1. Extension description

The Xqci 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

Xqci 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. Xqci adds conditional branch instructions that compare an immediate value to a register value, reducing a common code sequence into a single instruction.

Conditional instructions

Xqci 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

Xqci 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

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

Saturating arithmetic

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

Xqci 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.

1.1. Sub-extensions

The following sub-extensions are defined:

1.1.1. Xqcia

The Xqcia extension includes eleven instructions to perform integer arithmetic.

1.1.2. Xqciac

The Xqciac extension includes three instructions to accelerate common address calculations.

1.1.3. Xqcibi

The Xqcibi extension includes twelve conditional branch instructions that use an immediate operand for a source.

All instructions in Xqcibi following the same relocation type as standard RISC-V branches in the base architecture (e.g., BEQ, BNE, etc.).

1.1.4. Xqcibm

The Xqcibm extension includes thirty eight instructions that perform bit manipulation, include insertion and extraction.

1.1.5. Xqcicli

The Xqcicli extension includes twelve instructions that conditionally load an immediate value.

1.1.6. Xqcicm

The Xqcicm extension includes thirteen conditional move instructions.

1.1.7. Xqcics

The Xqcics extension includes eight conditional select instructions.

1.1.8. Xqcicsr

The Xqcicsr extension contains two instructions to read/write CSR which index is in register and not immediate.

1.1.9. Xqciint

The Xqciint extension includes eleven instructions to accelerate interrupt servicing by performing common actions during ISR prologue/epilogue.

1.1.10. Xqcilb

The Xqcilb extension includes two 48-bit instructions to encode a long branch.

1.1.11. Xqcili

The Xqcili extension includes a two instructions that load large immediates than is available with the base RISC-V ISA.

1.1.12. Xqcilia

The Xqcilia extension includes eight 48-bit instructions that perform arithmetic using large immediates.

1.1.13. Xqcilo

The Xqcilo extension includes eight 48-bit load/stores instructions that use an offset larger than can be found in the base RISC-V ISA.

1.1.14. Xqcilsm

The Xqcilsm extension includes six instructions that transfer multiple values between registers and memory.

1.1.15. Xqcisls

The Xqsls extension includes five load and three store instructions with a scaled index addressing mode.

Chapter 2. Instruction summary

The following 134 instructions are added by this extension:

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1	v0. 2	v0. 3
✓		qc.addsat rd, rs1, rs2	Saturating signed addition	✓	✓	✓
✓		qc.addusat rd, rs1, rs2	Saturating unsigned addition	✓	✓	✓
✓		qc.beqi rs1, imm, offset	Branch on equal (Immediate)	✓	✓	✓
✓		qc.bgei rs1, imm, offset	Branch on greater than or equal (immediate)	✓	✓	✓
✓		qc.bgeui rs1, imm, offset	Branch on greater than or equal unsigned (immediate)	✓	✓	✓
✓		qc.blti rs1, imm, offset	Branch on less than (immediate)	✓	✓	✓
✓		qc.bltui rs1, imm, offset	Branch on less than unsigned (immediate)	✓	✓	✓
✓		qc.bnei rs1, imm, offset	Branch on not equal (immediate)	✓	✓	✓
✓		qc.brev32 rd, rs1	Reverse bit order	✓	✓	✓
✓		qc.c.bexti rd, shamt	Single-Bit extract (Immediate)	✓	✓	✓
✓		qc.c.bseti rd, shamt	Single-Bit set (Immediate)	✓	✓	✓
✓		qc.c.clrint rs1	Clear interrupt (Register)	✓	✓	✓
✓		`qc.c.di`	Disable interrupts	✓	✓	✓
✓		qc.c.dir rd	Disable interrupts (Register)	✓	✓	✓
✓		`qc.c.ei`	Enable interrupts	✓	✓	✓
✓		qc.c.eir rs1	Restore interrupts (Register)	✓	✓	✓
✓		qc.c.extu rd, width	Extract bits unsigned	✓	✓	✓
✓		`qc.c.mienter.nest`	Machine mode interrupt enter	✓	✓	✓
✓		`qc.c.mienter`	Machine mode interrupt enter	✓	✓	✓
✓		`qc.c.mileaveret`	Machine mode interrupt exit	✓	✓	✓
✓		qc.c.muladdi rd, rs1, uimm	Multiply and accumulate (Immediate)	✓	✓	✓
✓		qc.c.mveqz rd, rs1	Conditional Move if equal to zero	✓	✓	✓
✓		qc.c.setint rs1	Set interrupt (Register)	✓	✓	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1	v0. 2	v0. 3
✓		qc.clo rd, rs1	Count leading ones	✓	✓	✓
✓		qc.clrinti imm	Clear interrupt (Immediate)	✓	✓	✓
✓		qc.compress2 rd, rs1	Bit compression (every 2nd bit)	✓	✓	✓
✓		qc.compress3 rd, rs1	Bit compression (every 3rd bit)	✓	✓	✓
✓	✓	qc.csrrwr rd, rs1, rs2	Atomic Read/Write CSR (Register)	✓	✓	✓
✓	✓	qc.csrrwri rd, imm, rs2	Atomic Read/Write CSR (Register) Immediate	✓	✓	✓
✓		qc.cto rd, rs1	Count trailing ones	✓	✓	✓
✓		qc.e.addai rd, imm	Add immediate	✓	✓	✓
✓		qc.e.addi rd, rs1, imm	Add immediate	✓	✓	✓
✓		qc.e.andai rd, imm	And immediate	✓	✓	✓
✓		qc.e.andi rd, rs1, imm	Add immediate	✓	✓	✓
✓		qc.e.beqi rs1, imm, offset	Branch on equal (immediate)	✓	✓	✓
✓		qc.e.bgei rs1, imm, offset	Branch on greater than or equal (immediate)	✓	✓	✓
✓		qc.e.bgeui rs1, imm, offset	Branch on greater than or equal unsigned (immediate)	✓	✓	✓
✓		qc.e.blتي rs1, imm, offset	Branch on less than (immediate)	✓	✓	✓
✓		qc.e.blتي rs1, imm, offset	Branch on less than unsigned (immediate)	✓	✓	✓
✓		qc.e.bnei rs1, imm, offset	Branch on not equal (immediate)	✓	✓	✓
✓		qc.e.j imm	Jump	✓	✓	✓
✓		qc.e.jal imm	Jump and link	✓	✓	✓
✓		qc.e.lb rd, imm(rs1)	Load byte	✓	✓	✓
✓		qc.e.lbu rd, imm(rs1)	Load byte unsigned	✓	✓	✓
✓		qc.e.lh rd, imm(rs1)	Load halfword	✓	✓	✓
✓		qc.e.lhu rd, imm(rs1)	Load halfword unsigned	✓	✓	✓
✓		qc.e.li rd, imm	Load immediate large	✓	✓	✓
✓		qc.e.lw rd, imm(rs1)	Load word	✓	✓	✓
✓		qc.e.orai rd, imm	Or immediate	✓	✓	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1	v0. 2	v0. 3
✓		qc.e.ori rd, rs1, imm	Or immediate	✓	✓	✓
✓		qc.e.sb rs2, imm(rs1)	Store byte	✓	✓	✓
✓		qc.e.sh rs2, imm(rs1)	Store halfword	✓	✓	✓
✓		qc.e.sw rs2, imm(rs1)	Store word	✓	✓	✓
✓		qc.e.xorai rd, imm	Exclusive Or immediate	✓	✓	✓
✓		qc.e.xori rd, rs1, imm	Exclusive Or immediate	✓	✓	✓
✓		qc.expand2 rd, rs1	Bit expansion (every 2nd bit)	✓	✓	✓
✓		qc.expand3 rd, rs1	Bit expansion (every 3rd bit)	✓	✓	✓
✓		qc.ext rd, rs1, width, shamt	Extract bits signed	✓	✓	✓
✓		qc.extd rd, rs1, width, shamt	Extract bits from pair signed (Immediate)	✓	✓	✓
✓		qc.extdpr rd, rs1, rs2	Extract bits from pair signed, packed descriptor (Register)	✓	✓	✓
✓		qc.extdprh rd, rs1, rs2	Extract bits from pair signed, packed descriptor high part (Register)	✓	✓	✓
✓		qc.extdr rd, rs1, rs2	Extract bits from pair signed (Register)	✓	✓	✓
✓		qc.extdu rd, rs1, width, shamt	Extract bits from pair unsigned (Immediate)	✓	✓	✓
✓		qc.extdupr rd, rs1, rs2	Extract bits from pair unsigned, packed descriptor (Register)	✓	✓	✓
✓		qc.extduprh rd, rs1, rs2	Extract bits from pair unsigned, packed descriptor high part (Register)	✓	✓	✓
✓		qc.extdur rd, rs1, rs2	Extract bits from pair unsigned (Register)	✓	✓	✓
✓		qc.extu rd, rs1, width, shamt	Extract bits unsigned	✓	✓	✓
✓		qc.insb rd, rs1, width, shamt	Insert bits (Register)	✓	✓	✓
✓		qc.insbh rd, rs1, width, shamt	Insert bits in 64-bit higher part (Register)	✓	✓	✓
✓		qc.insbhr rd, rs1, rs2	Insert bits in 64-bit higher part (Register)	✓	✓	✓
✓		qc.insbi rd, imm, width, shamt	Insert bits (Immediate)	✓	✓	✓
✓		qc.insbpr rd, rs1, rs2	Insert bits, packed descriptor (Register)	✓	✓	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1	v0. 2	v0. 3
✓		qc.insbprh rd, rs1, rs2	Insert bits, packed descriptor high part (Register)	✓	✓	✓
✓		qc.insbr rd, rs1, rs2	Insert bits (Register)	✓	✓	✓
✓		qc.insbri rd, rs1, imm	Insert bits (Immediate)	✓	✓	✓
✓		qc.li rd, imm	Load immediate large	✓	✓	✓
✓		qc.lieq rd, rs1, rs2, simm	Conditional load immediate if equal (Register)	✓	✓	✓
✓		qc.lieqi rd, rs1, imm, simm	Conditional load immediate if equal (Immediate)	✓	✓	✓
✓		qc.lige rd, rs1, rs2, simm	Conditional load immediate if great or equal than (Register)	✓	✓	✓
✓		qc.ligei rd, rs1, imm, simm	Conditional load immediate if great or equal than (Immediate)	✓	✓	✓
✓		qc.ligeu rd, rs1, rs2, simm	Conditional load immediate if great or equal than unsigned (Register)	✓	✓	✓
✓		qc.ligeui rd, rs1, imm, simm	Conditional load immediate if great or equal than unsigned (Immediate)	✓	✓	✓
✓		qc.lilt rd, rs1, rs2, simm	Conditional load immediate if less than (Register)	✓	✓	✓
✓		qc.lilti rd, rs1, imm, simm	Conditional load immediate if less than (Immediate)	✓	✓	✓
✓		qc.liltu rd, rs1, rs2, simm	Conditional load immediate if less than unsigned (Register)	✓	✓	✓
✓		qc.liltui rd, rs1, imm, simm	Conditional load immediate if less than unsigned (Immediate)	✓	✓	✓
✓		qc.line rd, rs1, rs2, simm	Conditional load immediate if not equal (Register)	✓	✓	✓
✓		qc.linei rd, rs1, imm, simm	Conditional load immediate if not equal (Immediate)	✓	✓	✓
✓		qc.lrb rd, rs1, rs2, shamt	Load indexed byte	✓	✓	✓
✓		qc.lrbu rd, rs1, rs2, shamt	Load indexed unsigned byte	✓	✓	✓
✓		qc.lrh rd, rs1, rs2, shamt	Load indexed halfword	✓	✓	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1	v0. 2	v0. 3
✓		qc.lrlhu rd, rs1, rs2, shamt	Load indexed unsigned halfword	✓	✓	✓
✓		qc.lrlw rd, rs1, rs2, shamt	Load indexed word	✓	✓	✓
✓		qc.lwm rd, rs2, imm(rs1)	Load word multiple	✓	✓	✓
✓		qc.lwmi rd, length, imm(rs1)	Load word multiple (Immediate)	✓	✓	✓
✓		qc.muladdi rd, rs1, imm	Multiply and accumulate (Immediate)	✓	✓	✓
✓		qc.mveq rd, rs1, rs2, rs3	Conditional move if equal (Register)	✓	✓	✓
✓		qc.mveqi rd, rs1, imm, rs3	Conditional move if equal (Immediate)	✓	✓	✓
✓		qc.mvge rd, rs1, rs2, rs3	Conditional move if great or equal than (Register)	✓	✓	✓
✓		qc.mvgei rd, rs1, imm, rs3	Conditional move if great or equal than (Immediate)	✓	✓	✓
✓		qc.mvgeu rd, rs1, rs2, rs3	Conditional move if great or equal than unsigned (Register)	✓	✓	✓
✓		qc.mvgeui rd, rs1, imm, rs3	Conditional move if great or equal than unsigned (Immediate)	✓	✓	✓
✓		qc.mvlt rd, rs1, rs2, rs3	Conditional move if less than (Register)	✓	✓	✓
✓		qc.mvlti rd, rs1, imm, rs3	Conditional move if less than (Immediate)	✓	✓	✓
✓		qc.mvltu rd, rs1, rs2, rs3	Conditional move if less than unsigned (Register)	✓	✓	✓
✓		qc.mvltui rd, rs1, imm, rs3	Conditional move if less than unsigned (Immediate)	✓	✓	✓
✓		qc.mvne rd, rs1, rs2, rs3	Conditional move if not equal (Register)	✓	✓	✓
✓		qc.mvnei rd, rs1, imm, rs3	Conditional move if not equal (Immediate)	✓	✓	✓
✓		qc.norm rd, rs1	Signed Normalization	✓	✓	✓
✓		qc.normeu rd, rs1	Unsigned Even Normalization	✓	✓	✓
✓		qc.normu rd, rs1	Unsigned Normalization	✓	✓	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1	v0. 2	v0. 3
✓		qc.selecteqi rd, imm, rs2, rs3	Select load immediate or register if equal (Immediate)	✓	✓	✓
✓		qc.selectieq rd, rs1, rs2, simm2	Select load immediate or register if equal (Register)	✓	✓	✓
✓		qc.selectieqi rd, imm, rs2, simm2	Select load immediate or register if equal (Immediate)	✓	✓	✓
✓		qc.selectiieq rd, rs1, simm1, simm2	Select load immediate if equal (Register)	✓	✓	✓
✓		qc.selectiine rd, rs1, simm1, simm2	Select load immediate if not equal (Register)	✓	✓	✓
✓		qc.selectine rd, rs1, rs2, simm2	Select load immediate or register if not equal (Register)	✓	✓	✓
✓		qc.selectinei rd, imm, rs2, simm2	Select load immediate or register if not equal (Immediate)	✓	✓	✓
✓		qc.selectnei rd, imm, rs2, rs3	Select load immediate or register if not equal (Immediate)	✓	✓	✓
✓		qc.setinti imm	Set interrupt (Immediate)	✓	✓	✓
✓		qc.setwm rs3, rs2, imm(rs1)	Set word multiple (Register)	✓	✓	✓
✓		qc.setwmi rs3, length, imm(rs1)	Set word multiple (Immediate)	✓	✓	✓
✓		qc.shladd rs1, rs2, shamt	Shift left and add (immediate)		✓	✓
✓		qc.slasat rd, rs1, rs2	Saturating arithmetic left shift	✓	✓	✓
✓		qc.sllsat rd, rs1, rs2	Saturating logical left shift	✓	✓	✓
✓		qc.srb rs3, rs1, rs2, shamt	Store indexed byte	✓	✓	✓
✓		qc.srh rs3, rs1, rs2, shamt	Store indexed halfword	✓	✓	✓
✓		qc.srw rs3, rs1, rs2, shamt	Store indexed word	✓	✓	✓
✓		qc.subsat rd, rs1, rs2	Saturating signed subtraction	✓	✓	✓
✓		qc.subusat rd, rs1, rs2	Saturating unsigned subtraction	✓	✓	✓
✓		qc.swm rs3, rs2, imm(rs1)	Store word multiple	✓	✓	✓

RV3 2	RV6 4	Mnemonic	Instruction	v0. 1	v0. 2	v0. 3
✓		qc.swmi rs3, length, imm(rs1)	Store word multiple (immediate)	✓	✓	✓
✓		qc.wrap rd, rs1, rs2	Wraparound (Register)	✓	✓	✓
✓		qc.wrapi rd, rs1, imm	Wraparound (Immediate)	✓	✓	✓

2.1. Instructions by sub-extension

Mnemonic	Xq ci	Xqc ia	Xqci ac	Xqc ibi	Xqci bm	Xqci cli	Xqci cm	Xqci cs	Xqci csr	Xqci int	Xqc ilb	Xq cili	Xqci lia	Xqc ilo	Xqcil sm	Xqci sls
qc.addsat	✓	✓														
qc.addsat	✓	✓														
qc.beqi	✓			✓												
qc.bgei	✓			✓												
qc.bgeui	✓			✓												
qc.blti	✓			✓												
qc.bltui	✓			✓												
qc.bnei	✓			✓												
qc.brev32	✓				✓											
qc.c.bexti	✓				✓											
qc.c.bseti	✓				✓											
qc.c.clrint	✓									✓						
qc.c.di	✓									✓						
qc.c.dir	✓									✓						
qc.c.ei	✓									✓						
qc.c.eir	✓									✓						
qc.c.extu	✓				✓											
qc.c.mienter.nest	✓									✓						
qc.c.mienter	✓									✓						

Mnemonic	Xqci	Xqcia	Xqciac	Xqciabi	Xqciabm	Xqciaccli	Xqciacm	Xqciacs	Xqciacsr	Xqciacint	Xqciacilb	Xqciacili	Xqciacilia	Xqciacilo	Xqciacilsm	Xqciacils
qc.c.mile averet	✓									✓						
qc.c.mul addi	✓		✓													
qc.c.mve qz	✓						✓									
qc.c.setin t	✓									✓						
qc.clo	✓				✓											
qc.clrinti	✓									✓						
qc.compress2	✓				✓											
qc.compress3	✓				✓											
qc.csrrwr	✓								✓							
qc.csrrwri	✓								✓							
qc.cto	✓				✓											
qc.e.addai	✓												✓			
qc.e.addi	✓												✓			
qc.e.andai	✓												✓			
qc.e.andi	✓												✓			
qc.e.beqi	✓			✓												
qc.e.bgei	✓			✓												
qc.e.bgeui	✓			✓												
qc.e.blti	✓			✓												
qc.e.bltui	✓			✓												
qc.e.bnei	✓			✓												
qc.e.j	✓										✓					
qc.e.jal	✓										✓					

Mnemonic	Xqci	Xqcia	Xqciac	Xqciibi	Xqciibm	Xqciicli	Xqciicm	Xqciics	Xqciicsr	Xqciint	Xqciilb	Xqcicili	Xqcilia	Xqcilo	Xqcilsm	Xqcisls
qc.e.lb	✓													✓		
qc.e.lbu	✓													✓		
qc.e.lh	✓													✓		
qc.e.lhu	✓													✓		
qc.e.li	✓											✓				
qc.e.lw	✓													✓		
qc.e.orai	✓												✓			
qc.e.ori	✓												✓			
qc.e.sb	✓													✓		
qc.e.sh	✓													✓		
qc.e.sw	✓													✓		
qc.e.xorai	✓												✓			
qc.e.xori	✓												✓			
qc.expanded2	✓				✓											
qc.expanded3	✓				✓											
qc.ext	✓				✓											
qc.extd	✓				✓											
qc.extdpr	✓				✓											
qc.extdprh	✓				✓											
qc.extdr	✓				✓											
qc.extdu	✓				✓											
qc.extdupr	✓				✓											
qc.extduprh	✓				✓											
qc.extdur	✓				✓											
qc.extu	✓				✓											
qc.insb	✓				✓											
qc.insbh	✓				✓											

Mnemonic	Xqci	Xqcia	Xqciac	Xqciibi	Xqciibm	Xqciicli	Xqciicm	Xqciics	Xqciicsr	Xqciint	Xqciilb	Xqcicili	Xqciilia	Xqcilo	Xqcilsm	Xqcisls
qc.insbhr	✓				✓											
qc.insbi	✓				✓											
qc.insbpr	✓				✓											
qc.insbprh	✓				✓											
qc.insbr	✓				✓											
qc.insbri	✓				✓											
qc.li	✓											✓				
qc.lieq	✓					✓										
qc.lieqi	✓					✓										
qc.lige	✓					✓										
qc.ligei	✓					✓										
qc.ligeu	✓					✓										
qc.ligeui	✓					✓										
qc.lilt	✓					✓										
qc.lilti	✓					✓										
qc.liltu	✓					✓										
qc.liltui	✓					✓										
qc.line	✓					✓										
qc.linei	✓					✓										
qc.lrb	✓															✓
qc.lrbu	✓															✓
qc.lrh	✓															✓
qc.lrhu	✓															✓
qc.lrw	✓															✓
qc.lwm	✓														✓	
qc.lwmi	✓														✓	
qc.muladdi	✓		✓													
qc.mveq	✓						✓									

Mnemonic	Xqci	Xqcia	Xqciao	Xqcib	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqcint	Xqcilb	Xqcicili	Xqcilia	Xqcilo	Xqcilsm	Xqcisls
qc.mveqi	✓						✓									
qc.mvge	✓						✓									
qc.mvgei	✓						✓									
qc.mvgeu	✓						✓									
qc.mvgeui	✓						✓									
qc.mvlt	✓						✓									
qc.mvlti	✓						✓									
qc.mvltu	✓						✓									
qc.mvltui	✓						✓									
qc.mvne	✓						✓									
qc.mvnei	✓						✓									
qc.norm	✓	✓														
qc.normeu	✓	✓														
qc.normu	✓	✓														
qc.selecteqi	✓							✓								
qc.selectieq	✓							✓								
qc.selectieqi	✓							✓								
qc.selectieiq	✓							✓								
qc.selectiine	✓							✓								
qc.selectiine	✓							✓								
qc.selectinei	✓							✓								
qc.selectnei	✓							✓								
qc.setinti	✓									✓						

Mnemonic	Xqci	Xqcia	Xqciac	Xqciibi	Xqciibm	Xqcicli	Xqci cm	Xqci cs	Xqci csr	Xqci int	Xqci ilb	Xqci cili	Xqci lia	Xqci ilo	Xqcil sm	Xqci sls
qc.setwm	✓														✓	
qc.setwmi	✓														✓	
qc.shlad	✓		✓													
qc.slasat	✓	✓														
qc.sllsat	✓	✓														
qc.srb	✓															✓
qc.srh	✓															✓
qc.srw	✓															✓
qc.subsat	✓	✓														
qc.subsat	✓	✓														
qc.swm	✓														✓	
qc.swmi	✓														✓	
qc.wrap	✓	✓														
qc.wrapi	✓	✓														

Chapter 3. CSR summary

The following 16 are added by this extension.

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

Chapter 4. Csrs (in alphabetical order)

DRAFT

4.1. qc.mclcie0

IRQ Enable 0

Enable bits for IRQs 0-31

4.1.1. Attributes

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

4.1.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24	IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8	IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

Figure 1. qc.mclcie0 format

4.1.3. 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

4.1.4. Fields

IRQ0

Location 0 Description IRQ0 enabled Type RW Reset value 0

IRQ1

Location

1

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

0

IRQ2

Location

2

Description*IRQ2 enabled***Type***RW***Reset value**

0

IRQ3

Location

3

Description*IRQ3 enabled***Type***RW***Reset value**

0

IRQ4

Location

4

Description*IRQ4 enabled***Type***RW***Reset value**

0

IRQ5

Location

5

Description*IRQ5 enabled***Type***RW***Reset value**

0

IRQ6

Location

6

Description*IRQ6 enabled***Type***RW***Reset value**

0

IRQ7

Location

7

Description*IRQ7 enabled***Type***RW***Reset value**

0

IRQ8

Location

8

Description*IRQ8 enabled***Type***RW***Reset value**

0

IRQ9

Location

9

Description*IRQ9 enabled***Type***RW***Reset value**

0

IRQ10

Location*10***Description***IRQ10 enabled***Type***RW***Reset value***0*

IRQ11

Location*11***Description***IRQ11 enabled***Type***RW***Reset value***0*

IRQ12

Location*12***Description***IRQ12 enabled***Type***RW***Reset value***0*

IRQ13

Location*13***Description***IRQ13 enabled***Type***RW***Reset value***0*

IRQ14

Location*14***Description***IRQ14 enabled***Type***RW***Reset value***0*

IRQ15

Location*15***Description***IRQ15 enabled***Type***RW***Reset value***0*

IRQ16

Location*16***Description***IRQ16 enabled***Type***RW***Reset value***0*

IRQ17

Location*17***Description***IRQ17 enabled***Type***RW***Reset value***0*

IRQ18

Location*18***Description***IRQ18 enabled***Type***RW***Reset value***0*

IRQ19

Location*19***Description***IRQ19 enabled***Type***RW***Reset value***0*

IRQ20

Location*20***Description***IRQ20 enabled***Type***RW***Reset value***0*

IRQ21

Location*21***Description***IRQ21 enabled***Type***RW***Reset value***0*

IRQ22

Location

22

Description*IRQ22 enabled***Type***RW***Reset value**

0

IRQ23

Location

23

Description*IRQ23 enabled***Type***RW***Reset value**

0

IRQ24

Location

24

Description*IRQ24 enabled***Type***RW***Reset value**

0

IRQ25

Location

25

Description*IRQ25 enabled***Type***RW***Reset value**

0

IRQ26

Location

26

Description*IRQ26 enabled***Type***RW***Reset value**

0

IRQ27

Location

27

Description*IRQ27 enabled***Type***RW***Reset value**

0

IRQ28

Location

28

Description*IRQ28 enabled***Type***RW***Reset value**

0

IRQ29

Location

29

Description*IRQ29 enabled***Type***RW***Reset value**

0

IRQ30

Location

30

Description*IRQ30 enabled***Type***RW***Reset value**

0

IRQ31

Location*31***Description***IRQ31 enabled***Type***RW***Reset value***0*

DRAFT

4.2. qc.mclcie1

IRQ Enable 1

Enable bits for IRQs 32-63

4.2.1. Attributes

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

4.2.2. Format

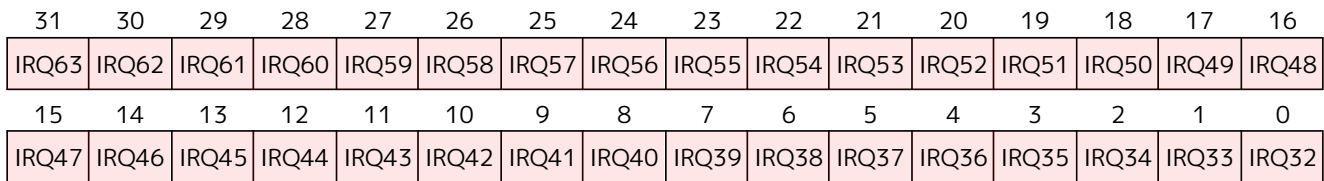


Figure 2. qc.mclcie1 format

4.2.3. 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

4.2.4. Fields

IRQ32

Location 0
Description IRQ32 enabled
Type RW
Reset value 0

IRQ33

Location

1

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

0

IRQ34

Location

2

Description*IRQ34 enabled***Type***RW***Reset value**

0

IRQ35

Location

3

Description*IRQ35 enabled***Type***RW***Reset value**

0

IRQ36

Location

4

Description*IRQ36 enabled***Type***RW***Reset value**

0

IRQ37

Location

5

Description*IRQ37 enabled***Type***RW***Reset value**

0

IRQ38

Location

6

Description*IRQ38 enabled***Type***RW***Reset value**

0

IRQ39

Location

7

Description*IRQ39 enabled***Type***RW***Reset value**

0

IRQ40

Location

8

Description*IRQ40 enabled***Type***RW***Reset value**

0

IRQ41

Location

9

Description*IRQ41 enabled***Type***RW***Reset value**

0

IRQ42

Location*10***Description***IRQ42 enabled***Type***RW***Reset value***0*

IRQ43

Location*11***Description***IRQ43 enabled***Type***RW***Reset value***0*

IRQ44

Location*12***Description***IRQ44 enabled***Type***RW***Reset value***0*

IRQ45

Location*13***Description***IRQ45 enabled***Type***RW***Reset value***0*

IRQ46

Location*14***Description***IRQ46 enabled***Type***RW***Reset value***0*

IRQ47

Location*15***Description***IRQ47 enabled***Type***RW***Reset value***0*

IRQ48

Location*16***Description***IRQ48 enabled***Type***RW***Reset value***0*

IRQ49

Location*17***Description***IRQ49 enabled***Type***RW***Reset value***0*

IRQ50

Location*18***Description***IRQ50 enabled***Type***RW***Reset value***0*

IRQ51

Location*19***Description***IRQ51 enabled***Type***RW***Reset value***0*

IRQ52

Location*20***Description***IRQ52 enabled***Type***RW***Reset value***0*

IRQ53

Location*21***Description***IRQ53 enabled***Type***RW***Reset value***0*

IRQ54

Location

22

Description*IRQ54 enabled***Type***RW***Reset value**

0

IRQ55

Location

23

Description*IRQ55 enabled***Type***RW***Reset value**

0

IRQ56

Location

24

Description*IRQ56 enabled***Type***RW***Reset value**

0

IRQ57

Location

25

Description*IRQ57 enabled***Type***RW***Reset value**

0

IRQ58

Location

26

Description*IRQ58 enabled***Type***RW***Reset value**

0

IRQ59

Location

27

Description*IRQ59 enabled***Type***RW***Reset value**

0

IRQ60

Location

28

Description*IRQ60 enabled***Type***RW***Reset value**

0

IRQ61

Location

29

Description*IRQ61 enabled***Type***RW***Reset value**

0

IRQ62

Location

30

Description*IRQ62 enabled***Type***RW***Reset value**

0

IRQ63

Location

31

Description

IRQ63 enabled

Type

RW

Reset value

0

DRAFT

4.3. qc.mclcie2

IRQ Enable 2

Enable bits for IRQs 64-95

4.3.1. Attributes

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

4.3.2. 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. qc.mclcie2 format

4.3.3. 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

4.3.4. Fields

IRQ64

Location 0
Description IRQ64 enabled
Type RW
Reset value 0

IRQ65

Location*1***Description***IRQ65 enabled***Type***RW***Reset value***0*

IRQ66

Location*2***Description***IRQ66 enabled***Type***RW***Reset value***0*

IRQ67

Location*3***Description***IRQ67 enabled***Type***RW***Reset value***0*

IRQ68

Location

4

Description*IRQ68 enabled***Type***RW***Reset value**

0

IRQ69

Location

5

Description*IRQ69 enabled***Type***RW***Reset value**

0

IRQ70

Location

6

Description*IRQ70 enabled***Type***RW***Reset value**

0

IRQ71

Location

7

Description*IRQ71 enabled***Type***RW***Reset value**

0

IRQ72

Location

8

Description*IRQ72 enabled***Type***RW***Reset value**

0

IRQ73

Location

9

Description*IRQ73 enabled***Type***RW***Reset value**

0

IRQ74

Location*10***Description***IRQ74 enabled***Type***RW***Reset value***0*

IRQ75

Location*11***Description***IRQ75 enabled***Type***RW***Reset value***0*

IRQ76

Location*12***Description***IRQ76 enabled***Type***RW***Reset value***0*

IRQ77

Location*13***Description***IRQ77 enabled***Type***RW***Reset value***0*

IRQ78

Location*14***Description***IRQ78 enabled***Type***RW***Reset value***0*

IRQ79

Location*15***Description***IRQ79 enabled***Type***RW***Reset value***0*

IRQ80

Location*16***Description***IRQ80 enabled***Type***RW***Reset value***0*

IRQ81

Location*17***Description***IRQ81 enabled***Type***RW***Reset value***0*

IRQ82

Location*18***Description***IRQ82 enabled***Type***RW***Reset value***0*

IRQ83

Location*19***Description***IRQ83 enabled***Type***RW***Reset value***0*

IRQ84

Location*20***Description***IRQ84 enabled***Type***RW***Reset value***0*

IRQ85

Location*21***Description***IRQ85 enabled***Type***RW***Reset value***0*

IRQ86

Location

22

Description*IRQ86 enabled***Type***RW***Reset value**

0

IRQ87

Location

23

Description*IRQ87 enabled***Type***RW***Reset value**

0

IRQ88

Location

24

Description*IRQ88 enabled***Type***RW***Reset value**

0

IRQ89

Location

25

Description*IRQ89 enabled***Type***RW***Reset value**

0

IRQ90

Location

26

Description*IRQ90 enabled***Type***RW***Reset value**

0

IRQ91

Location

27

Description*IRQ91 enabled***Type***RW***Reset value**

0

IRQ92

Location

28

Description*IRQ92 enabled***Type***RW***Reset value**

0

IRQ93

Location

29

Description*IRQ93 enabled***Type***RW***Reset value**

0

IRQ94

Location

30

Description*IRQ94 enabled***Type***RW***Reset value**

0

IRQ95

Location*31***Description***IRQ95 enabled***Type***RW***Reset value***0*

DRAFT

4.4. qc.mclcie3

IRQ Enable 3

Enable bits for IRQs 96-127

4.4.1. Attributes

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

4.4.2. 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. qc.mclcie3 format

4.4.3. 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

4.4.4. Fields

IRQ96

Location

0

Description

IRQ96 enabled

Type

RW

Reset value

0

IRQ97

Location

1

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

0

IRQ98

Location

2

Description*IRQ98 enabled***Type***RW***Reset value**

0

IRQ99

Location

3

Description*IRQ99 enabled***Type***RW***Reset value**

0

IRQ100

Location

4

Description*IRQ100 enabled***Type***RW***Reset value**

0

IRQ101

Location

5

Description*IRQ101 enabled***Type***RW***Reset value**

0

IRQ102

Location

6

Description*IRQ102 enabled***Type***RW***Reset value**

0

IRQ103

Location

7

Description*IRQ103 enabled***Type***RW***Reset value**

0

IRQ104

Location

8

Description*IRQ104 enabled***Type***RW***Reset value**

0

IRQ105

Location

9

Description*IRQ105 enabled***Type***RW***Reset value**

0

IRQ106

Location*10***Description***IRQ106 enabled***Type***RW***Reset value***0*

IRQ107

Location*11***Description***IRQ107 enabled***Type***RW***Reset value***0*

IRQ108

Location*12***Description***IRQ108 enabled***Type***RW***Reset value***0*

IRQ109

Location*13***Description***IRQ109 enabled***Type***RW***Reset value***0*

IRQ110

Location*14***Description***IRQ110 enabled***Type***RW***Reset value***0*

IRQ111

Location*15***Description***IRQ111 enabled***Type***RW***Reset value***0*

IRQ112

Location*16***Description***IRQ112 enabled***Type***RW***Reset value***0*

IRQ113

Location*17***Description***IRQ113 enabled***Type***RW***Reset value***0*

IRQ114

Location*18***Description***IRQ114 enabled***Type***RW***Reset value***0*

IRQ115

Location*19***Description***IRQ115 enabled***Type***RW***Reset value***0*

IRQ116

Location*20***Description***IRQ116 enabled***Type***RW***Reset value***0*

IRQ117

Location*21***Description***IRQ117 enabled***Type***RW***Reset value***0*

IRQ118

Location

22

Description*IRQ118 enabled***Type***RW***Reset value**

0

IRQ119

Location

23

Description*IRQ119 enabled***Type***RW***Reset value**

0

IRQ120

Location

24

Description*IRQ120 enabled***Type***RW***Reset value**

0

IRQ121

Location

25

Description*IRQ121 enabled***Type***RW***Reset value**

0

IRQ122

Location

26

Description*IRQ122 enabled***Type***RW***Reset value**

0

IRQ123

Location

27

Description*IRQ123 enabled***Type***RW***Reset value**

0

IRQ124

Location

28

Description*IRQ124 enabled***Type***RW***Reset value**

0

IRQ125

Location

29

Description*IRQ125 enabled***Type***RW***Reset value**

0

IRQ126

Location

30

Description*IRQ126 enabled***Type***RW***Reset value**

0

IRQ127

Location*31***Description***IRQ127 enabled***Type***RW***Reset value***0*

DRAFT

4.5. qc.mclcie4

IRQ Enable 4

Enable bits for IRQs 128-159

4.5.1. Attributes

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

4.5.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ159	IRQ158	IRQ157	IRQ156	IRQ155	IRQ154	IRQ153	IRQ152	IRQ151	IRQ150	IRQ149	IRQ148	IRQ147	IRQ146	IRQ145	IRQ144
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ143	IRQ142	IRQ141	IRQ140	IRQ139	IRQ138	IRQ137	IRQ136	IRQ135	IRQ134	IRQ133	IRQ132	IRQ131	IRQ130	IRQ129	IRQ128

Figure 5. qc.mclcie4 format

4.5.3. 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

4.5.4. Fields

IRQ128

Location 0
Description IRQ128 enabled
Type RW
Reset value 0

IRQ129

Location

1

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

0

IRQ130

Location

2

Description*IRQ130 enabled***Type***RW***Reset value**

0

IRQ131

Location

3

Description*IRQ131 enabled***Type***RW***Reset value**

0

IRQ132

Location

4

Description*IRQ132 enabled***Type***RW***Reset value**

0

IRQ133

Location

5

Description*IRQ133 enabled***Type***RW***Reset value**

0

IRQ134

Location

6

Description*IRQ134 enabled***Type***RW***Reset value**

0

IRQ135

Location

7

Description*IRQ135 enabled***Type***RW***Reset value**

0

IRQ136

Location

8

Description*IRQ136 enabled***Type***RW***Reset value**

0

IRQ137

Location

9

Description*IRQ137 enabled***Type***RW***Reset value**

0

IRQ138

Location*10***Description***IRQ138 enabled***Type***RW***Reset value***0*

IRQ139

Location*11***Description***IRQ139 enabled***Type***RW***Reset value***0*

IRQ140

Location*12***Description***IRQ140 enabled***Type***RW***Reset value***0*

IRQ141

Location*13***Description***IRQ141 enabled***Type***RW***Reset value***0*

IRQ142

Location*14***Description***IRQ142 enabled***Type***RW***Reset value***0*

IRQ143

Location*15***Description***IRQ143 enabled***Type***RW***Reset value***0*

IRQ144

Location*16***Description***IRQ144 enabled***Type***RW***Reset value***0*

IRQ145

Location*17***Description***IRQ145 enabled***Type***RW***Reset value***0*

IRQ146

Location*18***Description***IRQ146 enabled***Type***RW***Reset value***0*

IRQ147

Location*19***Description***IRQ147 enabled***Type***RW***Reset value***0*

IRQ148

Location*20***Description***IRQ148 enabled***Type***RW***Reset value***0*

IRQ149

Location*21***Description***IRQ149 enabled***Type***RW***Reset value***0*

IRQ150

Location

22

Description*IRQ150 enabled***Type***RW***Reset value**

0

IRQ151

Location

23

Description*IRQ151 enabled***Type***RW***Reset value**

0

IRQ152

Location

24

Description*IRQ152 enabled***Type***RW***Reset value**

0

IRQ153

Location

25

Description*IRQ153 enabled***Type***RW***Reset value**

0

IRQ154

Location

26

Description*IRQ154 enabled***Type***RW***Reset value**

0

IRQ155

Location

27

Description*IRQ155 enabled***Type***RW***Reset value**

0

IRQ156

Location

28

Description*IRQ156 enabled***Type***RW***Reset value**

0

IRQ157

Location

29

Description*IRQ157 enabled***Type***RW***Reset value**

0

IRQ158

Location

30

Description*IRQ158 enabled***Type***RW***Reset value**

0

IRQ159

Location*31***Description***IRQ159 enabled***Type***RW***Reset value***0*

DRAFT

4.6. qc.mclcie5

IRQ Enable 5

Enable bits for IRQs 160-191

4.6.1. Attributes

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

4.6.2. Format



Figure 6. qc.mclcie5 format

4.6.3. 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

4.6.4. Fields

IRQ160

Location

0

Description

IRQ160 enabled

Type

RW

Reset value

0

IRQ161

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

IRQ162

Location*2***Description***IRQ162 enabled***Type***RW***Reset value***0*

IRQ163

Location*3***Description***IRQ163 enabled***Type***RW***Reset value***0*

IRQ164

Location

4

Description*IRQ164 enabled***Type***RW***Reset value**

0

IRQ165

Location

5

Description*IRQ165 enabled***Type***RW***Reset value**

0

IRQ166

Location

6

Description*IRQ166 enabled***Type***RW***Reset value**

0

IRQ167

Location

7

Description*IRQ167 enabled***Type***RW***Reset value**

0

IRQ168

Location

8

Description*IRQ168 enabled***Type***RW***Reset value**

0

IRQ169

Location

9

Description*IRQ169 enabled***Type***RW***Reset value**

0

IRQ170

Location*10***Description***IRQ170 enabled***Type***RW***Reset value***0*

IRQ171

Location*11***Description***IRQ171 enabled***Type***RW***Reset value***0*

IRQ172

Location*12***Description***IRQ172 enabled***Type***RW***Reset value***0*

IRQ173

Location*13***Description***IRQ173 enabled***Type***RW***Reset value***0*

IRQ174

Location*14***Description***IRQ174 enabled***Type***RW***Reset value***0*

IRQ175

Location*15***Description***IRQ175 enabled***Type***RW***Reset value***0*

IRQ176

Location*16***Description***IRQ176 enabled***Type***RW***Reset value***0*

IRQ177

Location*17***Description***IRQ177 enabled***Type***RW***Reset value***0*

IRQ178

Location*18***Description***IRQ178 enabled***Type***RW***Reset value***0*

IRQ179

Location*19***Description***IRQ179 enabled***Type***RW***Reset value***0*

IRQ180

Location*20***Description***IRQ180 enabled***Type***RW***Reset value***0*

IRQ181

Location*21***Description***IRQ181 enabled***Type***RW***Reset value***0*

IRQ182

Location

22

Description*IRQ182 enabled***Type***RW***Reset value**

0

IRQ183

Location

23

Description*IRQ183 enabled***Type***RW***Reset value**

0

IRQ184

Location

24

Description*IRQ184 enabled***Type***RW***Reset value**

0

IRQ185

Location

25

Description*IRQ185 enabled***Type***RW***Reset value**

0

IRQ186

Location

26

Description*IRQ186 enabled***Type***RW***Reset value**

0

IRQ187

Location

27

Description*IRQ187 enabled***Type***RW***Reset value**

0

IRQ188

Location

28

Description*IRQ188 enabled***Type***RW***Reset value**

0

IRQ189

Location

29

Description*IRQ189 enabled***Type***RW***Reset value**

0

IRQ190

Location

30

Description*IRQ190 enabled***Type***RW***Reset value**

0

IRQ191

Location*31***Description***IRQ191 enabled***Type***RW***Reset value***0*

DRAFT

4.7. qc.mclcie6

IRQ Enable 6

Enable bits for IRQs 192-223

4.7.1. Attributes

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

4.7.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ223	IRQ222	IRQ221	IRQ220	IRQ219	IRQ218	IRQ217	IRQ216	IRQ215	IRQ214	IRQ213	IRQ212	IRQ211	IRQ210	IRQ209	IRQ208
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ207	IRQ206	IRQ205	IRQ204	IRQ203	IRQ202	IRQ201	IRQ200	IRQ199	IRQ198	IRQ197	IRQ196	IRQ195	IRQ194	IRQ193	IRQ192

Figure 7. qc.mclcie6 format

4.7.3. 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

4.7.4. Fields

IRQ192

Location

0

Description

IRQ192 enabled

Type

RW

Reset value

0

IRQ193

Location*1***Description***IRQ193 enabled***Type***RW***Reset value***0*

IRQ194

Location*2***Description***IRQ194 enabled***Type***RW***Reset value***0*

IRQ195

Location*3***Description***IRQ195 enabled***Type***RW***Reset value***0*

IRQ196

Location

4

Description*IRQ196 enabled***Type***RW***Reset value**

0

IRQ197

Location

5

Description*IRQ197 enabled***Type***RW***Reset value**

0

IRQ198

Location

6

Description*IRQ198 enabled***Type***RW***Reset value**

0

IRQ199

Location

7

Description*IRQ199 enabled***Type***RW***Reset value**

0

IRQ200

Location

8

Description*IRQ200 enabled***Type***RW***Reset value**

0

IRQ201

Location

9

Description*IRQ201 enabled***Type***RW***Reset value**

0

IRQ202

Location*10***Description***IRQ202 enabled***Type***RW***Reset value***0*

IRQ203

Location*11***Description***IRQ203 enabled***Type***RW***Reset value***0*

IRQ204

Location*12***Description***IRQ204 enabled***Type***RW***Reset value***0*

IRQ205

Location*13***Description***IRQ205 enabled***Type***RW***Reset value***0*

IRQ206

Location*14***Description***IRQ206 enabled***Type***RW***Reset value***0*

IRQ207

Location*15***Description***IRQ207 enabled***Type***RW***Reset value***0*

IRQ208

Location*16***Description***IRQ208 enabled***Type***RW***Reset value***0*

IRQ209

Location*17***Description***IRQ209 enabled***Type***RW***Reset value***0*

IRQ210

Location*18***Description***IRQ210 enabled***Type***RW***Reset value***0*

IRQ211

Location*19***Description***IRQ211 enabled***Type***RW***Reset value***0*

IRQ212

Location*20***Description***IRQ212 enabled***Type***RW***Reset value***0*

IRQ213

Location*21***Description***IRQ213 enabled***Type***RW***Reset value***0*

IRQ214

Location

22

Description*IRQ214 enabled***Type***RW***Reset value**

0

IRQ215

Location

23

Description*IRQ215 enabled***Type***RW***Reset value**

0

IRQ216

Location

24

Description*IRQ216 enabled***Type***RW***Reset value**

0

IRQ217

Location

25

Description*IRQ217 enabled***Type***RW***Reset value**

0

IRQ218

Location

26

Description*IRQ218 enabled***Type***RW***Reset value**

0

IRQ219

Location

27

Description*IRQ219 enabled***Type***RW***Reset value**

0

IRQ220

Location

28

Description*IRQ220 enabled***Type***RW***Reset value**

0

IRQ221

Location

29

Description*IRQ221 enabled***Type***RW***Reset value**

0

IRQ222

Location

30

Description*IRQ222 enabled***Type***RW***Reset value**

0

IRQ223

Location*31***Description***IRQ223 enabled***Type***RW***Reset value***0*

DRAFT

4.8. qc.mclcie7

IRQ Enable 7

Enable bits for IRQs 224-255

4.8.1. Attributes

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

4.8.2. Format



Figure 8. qc.mclcie7 format

4.8.3. 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

4.8.4. Fields

IRQ224

Location

0

Description

IRQ224 enabled

Type

RW

Reset value

0

IRQ225

Location

1

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

0

IRQ226

Location

2

Description*IRQ226 enabled***Type***RW***Reset value**

0

IRQ227

Location

3

Description*IRQ227 enabled***Type***RW***Reset value**

0

IRQ228

Location

4

Description*IRQ228 enabled***Type***RW***Reset value**

0

IRQ229

Location

5

Description*IRQ229 enabled***Type***RW***Reset value**

0

IRQ230

Location

6

Description*IRQ230 enabled***Type***RW***Reset value**

0

IRQ231

Location

7

Description*IRQ231 enabled***Type***RW***Reset value**

0

IRQ232

Location

8

Description*IRQ232 enabled***Type***RW***Reset value**

0

IRQ233

Location

9

Description*IRQ233 enabled***Type***RW***Reset value**

0

IRQ234

Location*10***Description***IRQ234 enabled***Type***RW***Reset value***0*

IRQ235

Location*11***Description***IRQ235 enabled***Type***RW***Reset value***0*

IRQ236

Location*12***Description***IRQ236 enabled***Type***RW***Reset value***0*

IRQ237

Location*13***Description***IRQ237 enabled***Type***RW***Reset value***0*

IRQ238

Location*14***Description***IRQ238 enabled***Type***RW***Reset value***0*

IRQ239

Location*15***Description***IRQ239 enabled***Type***RW***Reset value***0*

IRQ240

Location*16***Description***IRQ240 enabled***Type***RW***Reset value***0*

IRQ241

Location*17***Description***IRQ241 enabled***Type***RW***Reset value***0*

IRQ242

Location*18***Description***IRQ242 enabled***Type***RW***Reset value***0*

IRQ243

Location*19***Description***IRQ243 enabled***Type***RW***Reset value***0*

IRQ244

Location*20***Description***IRQ244 enabled***Type***RW***Reset value***0*

IRQ245

Location*21***Description***IRQ245 enabled***Type***RW***Reset value***0*

IRQ246

Location

22

Description*IRQ246 enabled***Type***RW***Reset value**

0

IRQ247

Location

23

Description*IRQ247 enabled***Type***RW***Reset value**

0

IRQ248

Location

24

Description*IRQ248 enabled***Type***RW***Reset value**

0

IRQ249

Location

25

Description*IRQ249 enabled***Type***RW***Reset value**

0

IRQ250

Location

26

Description*IRQ250 enabled***Type***RW***Reset value**

0

IRQ251

Location

27

Description*IRQ251 enabled***Type***RW***Reset value**

0

IRQ252

Location

28

Description*IRQ252 enabled***Type***RW***Reset value**

0

IRQ253

Location

29

Description*IRQ253 enabled***Type***RW***Reset value**

0

IRQ254

Location

30

Description*IRQ254 enabled***Type***RW***Reset value**

0

IRQ255

Location
31
Description
<i>IRQ255 enabled</i>
Type
<i>RW</i>
Reset value
0

DRAFT

4.9. qc.mclcip0

IRQ Pending 0

Pending bits for IRQs 0-31

4.9.1. Attributes

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

4.9.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ31	IRQ30	IRQ29	IRQ28	IRQ27	IRQ26	IRQ25	IRQ24	IRQ23	IRQ22	IRQ21	IRQ20	IRQ19	IRQ18	IRQ17	IRQ16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8	IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

Figure 9. qc.mclcip0 format

4.9.3. 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

4.9.4. Fields

IRQ0

Location 0
Description IRQ0 pending
Type RW
Reset value 0

IRQ1

Location

1

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

0

IRQ2

Location

2

Description*IRQ2 pending***Type***RW***Reset value**

0

IRQ3

Location

3

Description*IRQ3 pending***Type***RW***Reset value**

0

IRQ4

Location

4

Description*IRQ4 pending***Type***RW***Reset value**

0

IRQ5

Location

5

Description*IRQ5 pending***Type***RW***Reset value**

0

IRQ6

Location

6

Description*IRQ6 pending***Type***RW***Reset value**

0

IRQ7

Location

7

Description*IRQ7 pending***Type***RW***Reset value**

0

IRQ8

Location

8

Description*IRQ8 pending***Type***RW***Reset value**

0

IRQ9

Location

9

Description*IRQ9 pending***Type***RW***Reset value**

0

IRQ10

Location*10***Description***IRQ10 pending***Type***RW***Reset value***0*

IRQ11

Location*11***Description***IRQ11 pending***Type***RW***Reset value***0*

IRQ12

Location*12***Description***IRQ12 pending***Type***RW***Reset value***0*

IRQ13

Location*13***Description***IRQ13 pending***Type***RW***Reset value***0*

IRQ14

Location*14***Description***IRQ14 pending***Type***RW***Reset value***0*

IRQ15

Location*15***Description***IRQ15 pending***Type***RW***Reset value***0*

IRQ16

Location*16***Description***IRQ16 pending***Type***RW***Reset value***0*

IRQ17

Location*17***Description***IRQ17 pending***Type***RW***Reset value***0*

IRQ18

Location*18***Description***IRQ18 pending***Type***RW***Reset value***0*

IRQ19

Location*19***Description***IRQ19 pending***Type***RW***Reset value***0*

IRQ20

Location*20***Description***IRQ20 pending***Type***RW***Reset value***0*

IRQ21

Location*21***Description***IRQ21 pending***Type***RW***Reset value***0*

IRQ22

Location

22

Description*IRQ22 pending***Type***RW***Reset value**

0

IRQ23

Location

23

Description*IRQ23 pending***Type***RW***Reset value**

0

IRQ24

Location

24

Description*IRQ24 pending***Type***RW***Reset value**

0

IRQ25

Location

25

Description*IRQ25 pending***Type***RW***Reset value**

0

IRQ26

Location

26

Description*IRQ26 pending***Type***RW***Reset value**

0

IRQ27

Location

27

Description*IRQ27 pending***Type***RW***Reset value**

0

IRQ28

Location

28

Description*IRQ28 pending***Type***RW***Reset value**

0

IRQ29

Location

29

Description*IRQ29 pending***Type***RW***Reset value**

0

IRQ30

Location

30

Description*IRQ30 pending***Type***RW***Reset value**

0

IRQ31

Location*31***Description***IRQ31 pending***Type***RW***Reset value***0*

DRAFT

4.10. qc.mclicip1

IRQ Pending 1

Pending bits for IRQs 32-63

4.10.1. Attributes

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

4.10.2. Format

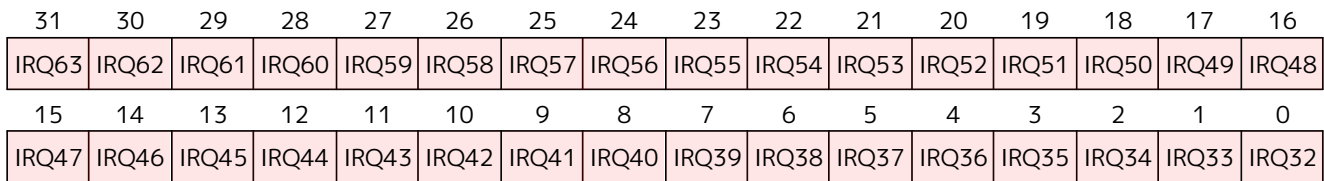


Figure 10. qc.mclicip1 format

4.10.3. 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

4.10.4. Fields

IRQ32

Location 0
Description IRQ32 pending
Type RW
Reset value 0

IRQ33

Location

1

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

0

IRQ34

Location

2

Description*IRQ34 pending***Type***RW***Reset value**

0

IRQ35

Location

3

Description*IRQ35 pending***Type***RW***Reset value**

0

IRQ36

Location

4

Description*IRQ36 pending***Type***RW***Reset value**

0

IRQ37

Location

5

Description*IRQ37 pending***Type***RW***Reset value**

0

IRQ38

Location

6

Description*IRQ38 pending***Type***RW***Reset value**

0

IRQ39

Location

7

Description*IRQ39 pending***Type***RW***Reset value**

0

IRQ40

Location

8

Description*IRQ40 pending***Type***RW***Reset value**

0

IRQ41

Location

9

Description*IRQ41 pending***Type***RW***Reset value**

0

IRQ42

Location*10***Description***IRQ42 pending***Type***RW***Reset value***0*

IRQ43

Location*11***Description***IRQ43 pending***Type***RW***Reset value***0*

IRQ44

Location*12***Description***IRQ44 pending***Type***RW***Reset value***0*

IRQ45

Location*13***Description***IRQ45 pending***Type***RW***Reset value***0*

IRQ46

Location*14***Description***IRQ46 pending***Type***RW***Reset value***0*

IRQ47

Location*15***Description***IRQ47 pending***Type***RW***Reset value***0*

IRQ48

Location*16***Description***IRQ48 pending***Type***RW***Reset value***0*

IRQ49

Location*17***Description***IRQ49 pending***Type***RW***Reset value***0*

IRQ50

Location*18***Description***IRQ50 pending***Type***RW***Reset value***0*

IRQ51

Location*19***Description***IRQ51 pending***Type***RW***Reset value***0*

IRQ52

Location*20***Description***IRQ52 pending***Type***RW***Reset value***0*

IRQ53

Location*21***Description***IRQ53 pending***Type***RW***Reset value***0*

IRQ54

Location

22

Description*IRQ54 pending***Type***RW***Reset value**

0

IRQ55

Location

23

Description*IRQ55 pending***Type***RW***Reset value**

0

IRQ56

Location

24

Description*IRQ56 pending***Type***RW***Reset value**

0

IRQ57

Location

25

Description*IRQ57 pending***Type***RW***Reset value**

0

IRQ58

Location

26

Description*IRQ58 pending***Type***RW***Reset value**

0

IRQ59

Location

27

Description*IRQ59 pending***Type***RW***Reset value**

0

IRQ60

Location

28

Description*IRQ60 pending***Type***RW***Reset value**

0

IRQ61

Location

29

Description*IRQ61 pending***Type***RW***Reset value**

0

IRQ62

Location

30

Description*IRQ62 pending***Type***RW***Reset value**

0

IRQ63

Location*31***Description***IRQ63 pending***Type***RW***Reset value***0*

DRAFT

4.11. qc.mclcip2

IRQ Pending 2

Pending bits for IRQs 64-95

4.11.1. Attributes

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

4.11.2. 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. qc.mclcip2 format

4.11.3. 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

4.11.4. Fields

IRQ64

Location 0
Description IRQ64 pending
Type RW
Reset value 0

IRQ65

Location

1

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

0

IRQ66

Location

2

Description*IRQ66 pending***Type***RW***Reset value**

0

IRQ67

Location

3

Description*IRQ67 pending***Type***RW***Reset value**

0

IRQ68

Location

4

Description*IRQ68 pending***Type***RW***Reset value**

0

IRQ69

Location

5

Description*IRQ69 pending***Type***RW***Reset value**

0

IRQ70

Location

6

Description*IRQ70 pending***Type***RW***Reset value**

0

IRQ71

Location

7

Description*IRQ71 pending***Type***RW***Reset value**

0

IRQ72

Location

8

Description*IRQ72 pending***Type***RW***Reset value**

0

IRQ73

Location

9

Description*IRQ73 pending***Type***RW***Reset value**

0

IRQ74

Location*10***Description***IRQ74 pending***Type***RW***Reset value***0*

IRQ75

Location*11***Description***IRQ75 pending***Type***RW***Reset value***0*

IRQ76

Location*12***Description***IRQ76 pending***Type***RW***Reset value***0*

IRQ77

Location*13***Description***IRQ77 pending***Type***RW***Reset value***0*

IRQ78

Location*14***Description***IRQ78 pending***Type***RW***Reset value***0*

IRQ79

Location*15***Description***IRQ79 pending***Type***RW***Reset value***0*

IRQ80

Location*16***Description***IRQ80 pending***Type***RW***Reset value***0*

IRQ81

Location*17***Description***IRQ81 pending***Type***RW***Reset value***0*

IRQ82

Location*18***Description***IRQ82 pending***Type***RW***Reset value***0*

IRQ83

Location*19***Description***IRQ83 pending***Type***RW***Reset value***0*

IRQ84

Location*20***Description***IRQ84 pending***Type***RW***Reset value***0*

IRQ85

Location*21***Description***IRQ85 pending***Type***RW***Reset value***0*

IRQ86

Location

22

Description*IRQ86 pending***Type***RW***Reset value**

0

IRQ87

Location

23

Description*IRQ87 pending***Type***RW***Reset value**

0

IRQ88

Location

24

Description*IRQ88 pending***Type***RW***Reset value**

0

IRQ89

Location

25

Description*IRQ89 pending***Type***RW***Reset value**

0

IRQ90

Location

26

Description*IRQ90 pending***Type***RW***Reset value**

0

IRQ91

Location

27

Description*IRQ91 pending***Type***RW***Reset value**

0

IRQ92

Location

28

Description*IRQ92 pending***Type***RW***Reset value**

0

IRQ93

Location

29

Description*IRQ93 pending***Type***RW***Reset value**

0

IRQ94

Location

30

Description*IRQ94 pending***Type***RW***Reset value**

0

IRQ95

Location*31***Description***IRQ95 pending***Type***RW***Reset value***0*

DRAFT

4.12. qc.mclcip3

IRQ Pending 3

Pending bits for IRQs 96-127

4.12.1. Attributes

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

4.12.2. Format



Figure 12. qc.mclcip3 format

4.12.3. 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

4.12.4. Fields

IRQ96

Location

0

Description

IRQ96 pending

Type

RW

Reset value

0

IRQ97

Location

1

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

0

IRQ98

Location

2

Description*IRQ98 pending***Type***RW***Reset value**

0

IRQ99

Location

3

Description*IRQ99 pending***Type***RW***Reset value**

0

IRQ100

Location

4

Description*IRQ100 pending***Type***RW***Reset value**

0

IRQ101

Location

5

Description*IRQ101 pending***Type***RW***Reset value**

0

IRQ102

Location

6

Description*IRQ102 pending***Type***RW***Reset value**

0

IRQ103

Location

7

Description*IRQ103 pending***Type***RW***Reset value**

0

IRQ104

Location

8

Description*IRQ104 pending***Type***RW***Reset value**

0

IRQ105

Location

9

Description*IRQ105 pending***Type***RW***Reset value**

0

IRQ106

Location*10***Description***IRQ106 pending***Type***RW***Reset value***0*

IRQ107

Location*11***Description***IRQ107 pending***Type***RW***Reset value***0*

IRQ108

Location*12***Description***IRQ108 pending***Type***RW***Reset value***0*

IRQ109

Location*13***Description***IRQ109 pending***Type***RW***Reset value***0*

IRQ110

Location*14***Description***IRQ110 pending***Type***RW***Reset value***0*

IRQ111

Location*15***Description***IRQ111 pending***Type***RW***Reset value***0*

IRQ112

Location*16***Description***IRQ112 pending***Type***RW***Reset value***0*

IRQ113

Location*17***Description***IRQ113 pending***Type***RW***Reset value***0*

IRQ114

Location*18***Description***IRQ114 pending***Type***RW***Reset value***0*

IRQ115

Location*19***Description***IRQ115 pending***Type***RW***Reset value***0*

IRQ116

Location*20***Description***IRQ116 pending***Type***RW***Reset value***0*

IRQ117

Location*21***Description***IRQ117 pending***Type***RW***Reset value***0*

IRQ118

Location

22

Description*IRQ118 pending***Type***RW***Reset value**

0

IRQ119

Location

23

Description*IRQ119 pending***Type***RW***Reset value**

0

IRQ120

Location

24

Description*IRQ120 pending***Type***RW***Reset value**

0

IRQ121

Location

25

Description*IRQ121 pending***Type***RW***Reset value**

0

IRQ122

Location

26

Description*IRQ122 pending***Type***RW***Reset value**

0

IRQ123

Location

27

Description*IRQ123 pending***Type***RW***Reset value**

0

IRQ124

Location

28

Description*IRQ124 pending***Type***RW***Reset value**

0

IRQ125

Location

29

Description*IRQ125 pending***Type***RW***Reset value**

0

IRQ126

Location

30

Description*IRQ126 pending***Type***RW***Reset value**

0

IRQ127

Location*31***Description***IRQ127 pending***Type***RW***Reset value***0*

DRAFT

4.13. qc.mclcip4

IRQ Pending 4

Pending bits for IRQs 128-159

4.13.1. Attributes

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

4.13.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ159	IRQ158	IRQ157	IRQ156	IRQ155	IRQ154	IRQ153	IRQ152	IRQ151	IRQ150	IRQ149	IRQ148	IRQ147	IRQ146	IRQ145	IRQ144
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ143	IRQ142	IRQ141	IRQ140	IRQ139	IRQ138	IRQ137	IRQ136	IRQ135	IRQ134	IRQ133	IRQ132	IRQ131	IRQ130	IRQ129	IRQ128

Figure 13. qc.mclcip4 format

4.13.3. 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

4.13.4. Fields

IRQ128

Location 0
Description IRQ128 pending
Type RW
Reset value 0

IRQ129

Location

1

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

0

IRQ130

Location

2

Description*IRQ130 pending***Type***RW***Reset value**

0

IRQ131

Location

3

Description*IRQ131 pending***Type***RW***Reset value**

0

IRQ132

Location

4

Description*IRQ132 pending***Type***RW***Reset value**

0

IRQ133

Location

5

Description*IRQ133 pending***Type***RW***Reset value**

0

IRQ134

Location

6

Description*IRQ134 pending***Type***RW***Reset value**

0

IRQ135

Location

7

Description*IRQ135 pending***Type***RW***Reset value**

0

IRQ136

Location

8

Description*IRQ136 pending***Type***RW***Reset value**

0

IRQ137

Location

9

Description*IRQ137 pending***Type***RW***Reset value**

0

IRQ138

Location*10***Description***IRQ138 pending***Type***RW***Reset value***0*

IRQ139

Location*11***Description***IRQ139 pending***Type***RW***Reset value***0*

IRQ140

Location*12***Description***IRQ140 pending***Type***RW***Reset value***0*

IRQ141

Location*13***Description***IRQ141 pending***Type***RW***Reset value***0*

IRQ142

Location*14***Description***IRQ142 pending***Type***RW***Reset value***0*

IRQ143

Location*15***Description***IRQ143 pending***Type***RW***Reset value***0*

IRQ144

Location*16***Description***IRQ144 pending***Type***RW***Reset value***0*

IRQ145

Location*17***Description***IRQ145 pending***Type***RW***Reset value***0*

IRQ146

Location*18***Description***IRQ146 pending***Type***RW***Reset value***0*

IRQ147

Location*19***Description***IRQ147 pending***Type***RW***Reset value***0*

IRQ148

Location*20***Description***IRQ148 pending***Type***RW***Reset value***0*

IRQ149

Location*21***Description***IRQ149 pending***Type***RW***Reset value***0*

IRQ150

Location

22

Description*IRQ150 pending***Type***RW***Reset value**

0

IRQ151

Location

23

Description*IRQ151 pending***Type***RW***Reset value**

0

IRQ152

Location

24

Description*IRQ152 pending***Type***RW***Reset value**

0

IRQ153

Location

25

Description*IRQ153 pending***Type***RW***Reset value**

0

IRQ154

Location

26

Description*IRQ154 pending***Type***RW***Reset value**

0

IRQ155

Location

27

Description*IRQ155 pending***Type***RW***Reset value**

0

IRQ156

Location

28

Description*IRQ156 pending***Type***RW***Reset value**

0

IRQ157

Location

29

Description*IRQ157 pending***Type***RW***Reset value**

0

IRQ158

Location

30

Description*IRQ158 pending***Type***RW***Reset value**

0

IRQ159

Location*31***Description***IRQ159 pending***Type***RW***Reset value***0*

DRAFT

4.14. qc.mclcip5

IRQ Pending 5

Pending bits for IRQs 160-191

4.14.1. Attributes

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

4.14.2. Format



Figure 14. qc.mclcip5 format

4.14.3. 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

4.14.4. Fields

IRQ160

Location

0

Description

IRQ160 pending

Type

RW

Reset value

0

IRQ161

Location

1

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

0

IRQ162

Location

2

Description*IRQ162 pending***Type***RW***Reset value**

0

IRQ163

Location

3

Description*IRQ163 pending***Type***RW***Reset value**

0

IRQ164

Location

4

Description*IRQ164 pending***Type***RW***Reset value**

0

IRQ165

Location

5

Description*IRQ165 pending***Type***RW***Reset value**

0

IRQ166

Location

6

Description*IRQ166 pending***Type***RW***Reset value**

0

IRQ167

Location

7

Description*IRQ167 pending***Type***RW***Reset value**

0

IRQ168

Location

8

Description*IRQ168 pending***Type***RW***Reset value**

0

IRQ169

Location

9

Description*IRQ169 pending***Type***RW***Reset value**

0

IRQ170

Location*10***Description***IRQ170 pending***Type***RW***Reset value***0*

IRQ171

Location*11***Description***IRQ171 pending***Type***RW***Reset value***0*

IRQ172

Location*12***Description***IRQ172 pending***Type***RW***Reset value***0*

IRQ173

Location*13***Description***IRQ173 pending***Type***RW***Reset value***0*

IRQ174

Location*14***Description***IRQ174 pending***Type***RW***Reset value***0*

IRQ175

Location*15***Description***IRQ175 pending***Type***RW***Reset value***0*

IRQ176

Location*16***Description***IRQ176 pending***Type***RW***Reset value***0*

IRQ177

Location*17***Description***IRQ177 pending***Type***RW***Reset value***0*

IRQ178

Location*18***Description***IRQ178 pending***Type***RW***Reset value***0*

IRQ179

Location*19***Description***IRQ179 pending***Type***RW***Reset value***0*

IRQ180

Location*20***Description***IRQ180 pending***Type***RW***Reset value***0*

IRQ181

Location*21***Description***IRQ181 pending***Type***RW***Reset value***0*

IRQ182

Location

22

Description*IRQ182 pending***Type***RW***Reset value**

0

IRQ183

Location

23

Description*IRQ183 pending***Type***RW***Reset value**

0

IRQ184

Location

24

Description*IRQ184 pending***Type***RW***Reset value**

0

IRQ185

Location

25

Description*IRQ185 pending***Type***RW***Reset value**

0

IRQ186

Location

26

Description*IRQ186 pending***Type***RW***Reset value**

0

IRQ187

Location

27

Description*IRQ187 pending***Type***RW***Reset value**

0

IRQ188

Location

28

Description*IRQ188 pending***Type***RW***Reset value**

0

IRQ189

Location

29

Description*IRQ189 pending***Type***RW***Reset value**

0

IRQ190

Location

30

Description*IRQ190 pending***Type***RW***Reset value**

0

IRQ191

Location*31***Description***IRQ191 pending***Type***RW***Reset value***0*

DRAFT

4.15. qc.mclcip6

IRQ Pending 6

Pending bits for IRQs 192-223

4.15.1. Attributes

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

4.15.2. Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IRQ223	IRQ222	IRQ221	IRQ220	IRQ219	IRQ218	IRQ217	IRQ216	IRQ215	IRQ214	IRQ213	IRQ212	IRQ211	IRQ210	IRQ209	IRQ208
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ207	IRQ206	IRQ205	IRQ204	IRQ203	IRQ202	IRQ201	IRQ200	IRQ199	IRQ198	IRQ197	IRQ196	IRQ195	IRQ194	IRQ193	IRQ192

Figure 15. qc.mclcip6 format

4.15.3. 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

4.15.4. Fields

IRQ192

Location

0

Description

IRQ192 pending

Type

RW

Reset value

0

IRQ193

Location*1***Description***IRQ193 pending***Type***RW***Reset value***0*

IRQ194

Location*2***Description***IRQ194 pending***Type***RW***Reset value***0*

IRQ195

Location*3***Description***IRQ195 pending***Type***RW***Reset value***0*

IRQ196

Location

4

Description*IRQ196 pending***Type***RW***Reset value**

0

IRQ197

Location

5

Description*IRQ197 pending***Type***RW***Reset value**

0

IRQ198

Location

6

Description*IRQ198 pending***Type***RW***Reset value**

0

IRQ199

Location

7

Description*IRQ199 pending***Type***RW***Reset value**

0

IRQ200

Location

8

Description*IRQ200 pending***Type***RW***Reset value**

0

IRQ201

Location

9

Description*IRQ201 pending***Type***RW***Reset value**

0

IRQ202

Location*10***Description***IRQ202 pending***Type***RW***Reset value***0*

IRQ203

Location*11***Description***IRQ203 pending***Type***RW***Reset value***0*

IRQ204

Location*12***Description***IRQ204 pending***Type***RW***Reset value***0*

IRQ205

Location*13***Description***IRQ205 pending***Type***RW***Reset value***0*

IRQ206

Location*14***Description***IRQ206 pending***Type***RW***Reset value***0*

IRQ207

Location*15***Description***IRQ207 pending***Type***RW***Reset value***0*

IRQ208

Location*16***Description***IRQ208 pending***Type***RW***Reset value***0*

IRQ209

Location*17***Description***IRQ209 pending***Type***RW***Reset value***0*

IRQ210

Location*18***Description***IRQ210 pending***Type***RW***Reset value***0*

IRQ211

Location*19***Description***IRQ211 pending***Type***RW***Reset value***0*

IRQ212

Location*20***Description***IRQ212 pending***Type***RW***Reset value***0*

IRQ213

Location*21***Description***IRQ213 pending***Type***RW***Reset value***0*

IRQ214

Location

22

Description*IRQ214 pending***Type***RW***Reset value**

0

IRQ215

Location

23

Description*IRQ215 pending***Type***RW***Reset value**

0

IRQ216

Location

24

Description*IRQ216 pending***Type***RW***Reset value**

0

IRQ217

Location

25

Description*IRQ217 pending***Type***RW***Reset value**

0

IRQ218

Location

26

Description*IRQ218 pending***Type***RW***Reset value**

0

IRQ219

Location

27

Description*IRQ219 pending***Type***RW***Reset value**

0

IRQ220

Location

28

Description*IRQ220 pending***Type***RW***Reset value**

0

IRQ221

Location

29

Description*IRQ221 pending***Type***RW***Reset value**

0

IRQ222

Location

30

Description*IRQ222 pending***Type***RW***Reset value**

0

IRQ223

Location*31***Description***IRQ223 pending***Type***RW***Reset value***0*

DRAFT

4.16. qc.mclcip7

IRQ Pending 7

Pending bits for IRQs 224-255

4.16.1. Attributes

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

4.16.2. Format



Figure 16. qc.mclcip7 format

4.16.3. 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

4.16.4. Fields

IRQ224

Location

0

Description

IRQ224 pending

Type

RW

Reset value

0

IRQ225

Location

1

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

0

IRQ226

Location

2

Description*IRQ226 pending***Type***RW***Reset value**

0

IRQ227

Location

3

Description*IRQ227 pending***Type***RW***Reset value**

0

IRQ228

Location

4

Description*IRQ228 pending***Type***RW***Reset value**

0

IRQ229

Location

5

Description*IRQ229 pending***Type***RW***Reset value**

0

IRQ230

Location

6

Description*IRQ230 pending***Type***RW***Reset value**

0

IRQ231

Location

7

Description*IRQ231 pending***Type***RW***Reset value**

0

IRQ232

Location

8

Description*IRQ232 pending***Type***RW***Reset value**

0

IRQ233

Location

9

Description*IRQ233 pending***Type***RW***Reset value**

0

IRQ234

Location*10***Description***IRQ234 pending***Type***RW***Reset value***0*

IRQ235

Location*11***Description***IRQ235 pending***Type***RW***Reset value***0*

IRQ236

Location*12***Description***IRQ236 pending***Type***RW***Reset value***0*

IRQ237

Location*13***Description***IRQ237 pending***Type***RW***Reset value***0*

IRQ238

Location*14***Description***IRQ238 pending***Type***RW***Reset value***0*

IRQ239

Location*15***Description***IRQ239 pending***Type***RW***Reset value***0*

IRQ240

Location*16***Description***IRQ240 pending***Type***RW***Reset value***0*

IRQ241

Location*17***Description***IRQ241 pending***Type***RW***Reset value***0*

IRQ242

Location*18***Description***IRQ242 pending***Type***RW***Reset value***0*

IRQ243

Location*19***Description***IRQ243 pending***Type***RW***Reset value***0*

IRQ244

Location*20***Description***IRQ244 pending***Type***RW***Reset value***0*

IRQ245

Location*21***Description***IRQ245 pending***Type***RW***Reset value***0*

IRQ246

Location

22

Description*IRQ246 pending***Type***RW***Reset value**

0

IRQ247

Location

23

Description*IRQ247 pending***Type***RW***Reset value**

0

IRQ248

Location

24

Description*IRQ248 pending***Type***RW***Reset value**

0

IRQ249

Location

25

Description*IRQ249 pending***Type***RW***Reset value**

0

IRQ250

Location

26

Description*IRQ250 pending***Type***RW***Reset value**

0

IRQ251

Location

27

Description*IRQ251 pending***Type***RW***Reset value**

0

IRQ252

Location

28

Description*IRQ252 pending***Type***RW***Reset value**

0

IRQ253

Location

29

Description*IRQ253 pending***Type***RW***Reset value**

0

IRQ254

Location

30

Description*IRQ254 pending***Type***RW***Reset value**

0

IRQ255

Location*31***Description***IRQ255 pending***Type***RW***Reset value***0*

DRAFT

Chapter 5. Instructions (in alphabetical order)

5.1. qc.addsat

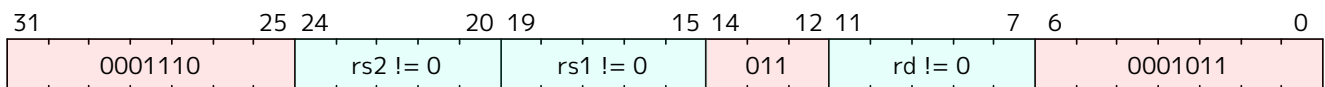
Synopsis

Saturating signed addition

Mnemonic

```
qc.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
Xqci	>= 0

Extension	Minimum version
Xqcia	≥ 0

DRAFT

5.2. qc.addusat

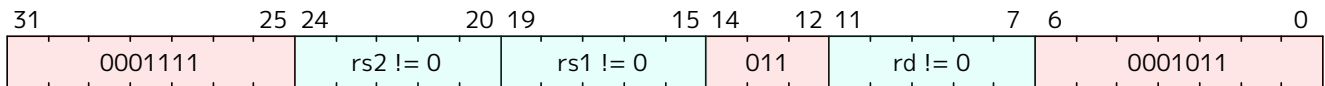
Synopsis

Saturating unsigned addition

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.3. qc.beqi

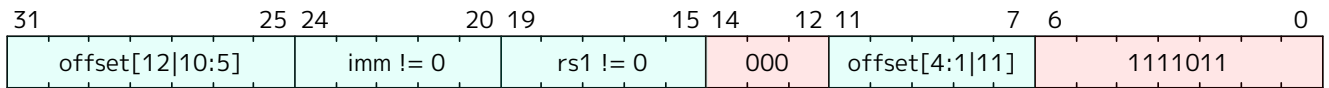
Synopsis

Branch on equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibi	>= 0

5.4. qc.bgei

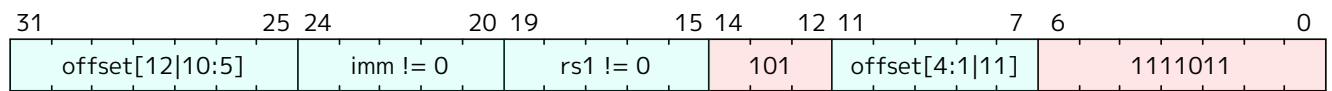
Synopsis

Branch on greater than or equal (immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibi	>= 0

5.5. qc.bgeui

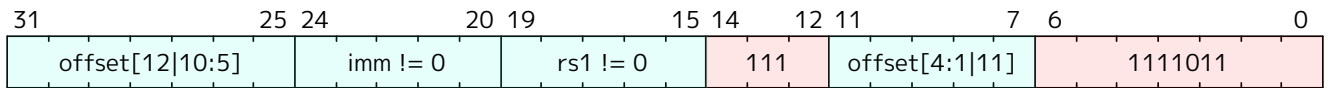
Synopsis

Branch on greater than or equal unsigned (immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibi	>= 0

5.6. qc.blti

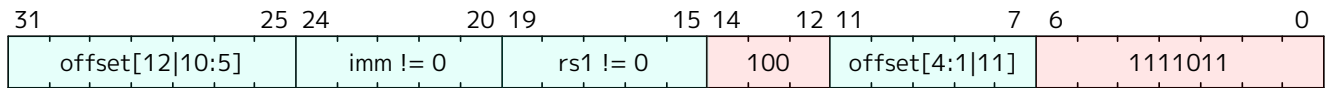
Synopsis

Branch on less than (immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibi	>= 0

5.7. qc.bltoi

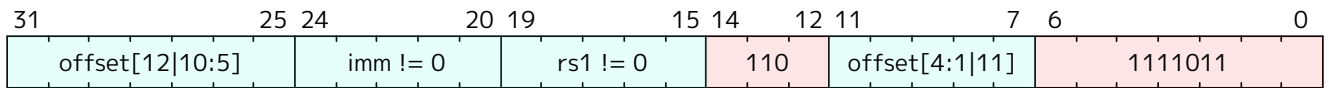
Synopsis

Branch on less than unsigned (immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibi	>= 0

5.8. qc.bnei

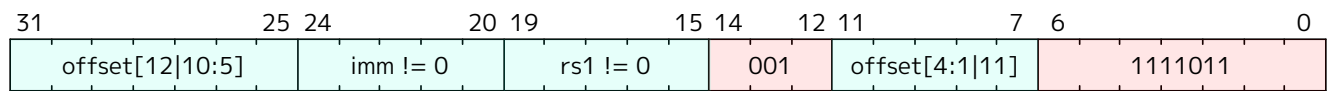
Synopsis

Branch on not equal (immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibi	>= 0

5.9. qc.brev32

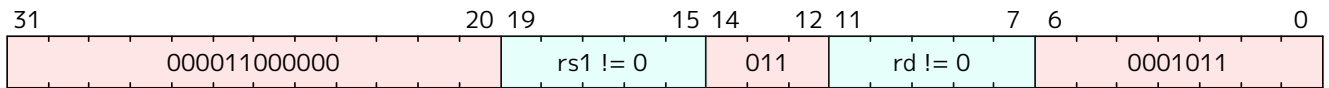
Synopsis

Reverse bit order

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.10. qc.c.bexti

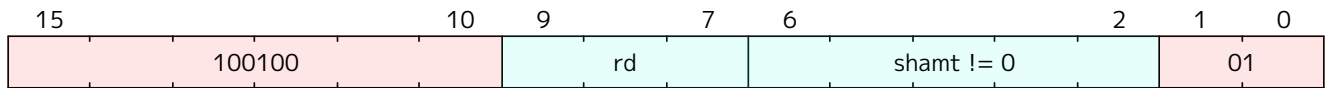
Synopsis

Single-Bit extract (Immediate)

Mnemonic

```
qc.c.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
Xqci	>= 0
Xqcibm	>= 0

5.11. qc.c.bseti

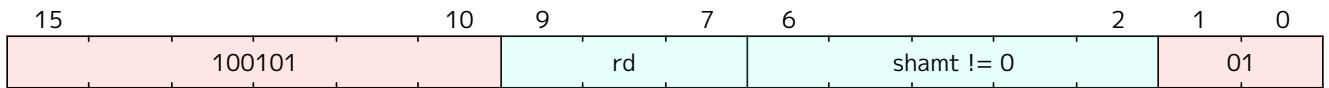
Synopsis

Single-Bit set (Immediate)

Mnemonic

```
qc.c.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 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] | (1 << index);
```

Included in

Extension	Minimum version
Xqci	>= 0
Xqcibm	>= 0

5.12. qc.c.clrint

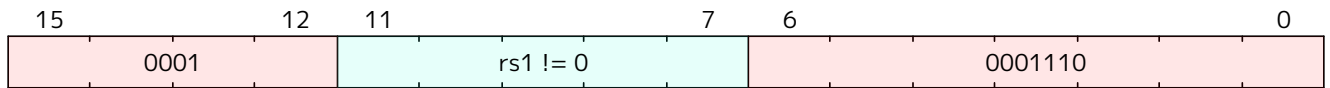
Synopsis

Clear interrupt (Register)

Mnemonic

```
qc.c.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[mclicip0 + idx].sw_read();
CSR[mclicip0 + idx].sw_write(pre_csr & ~(1 << bit));
```

Included in

Extension	Minimum version
Xqci	>= 0
Xqciint	>= 0

5.13. qc.c.di

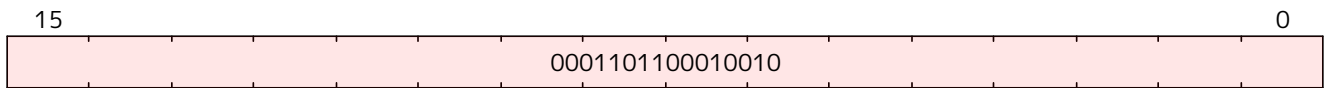
Synopsis

Disable interrupts

Mnemonic

qc.c.di

Encoding



Description

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

Decode Variables

qc.c.di has no decode variables.

Operation

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

Included in

Extension	Minimum version
Xqci	>= 0
Xqciint	>= 0

5.14. qc.c.dir

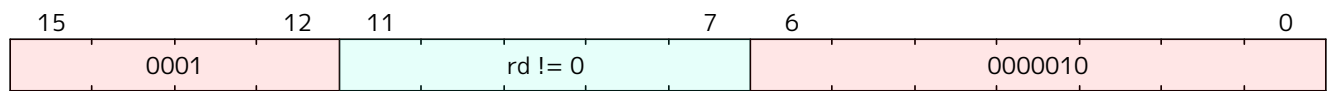
Synopsis

Disable interrupts (Register)

Mnemonic

```
qc.c.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
Xqci	>= 0
Xqciint	>= 0

5.15. qc.c.ei

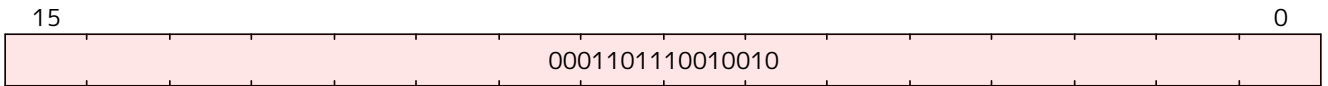
Synopsis

Enable interrupts

Mnemonic

qc.c.ei

Encoding



Description

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

Decode Variables

qc.c.ei has no decode variables.

Operation

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

Included in

Extension	Minimum version
Xqci	>= 0
Xqciint	>= 0

5.16. qc.c.eir

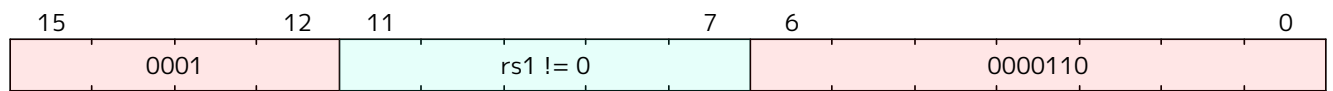
Synopsis

Restore interrupts (Register)

Mnemonic

```
qc.c.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
Xqci	>= 0
Xqciint	>= 0

5.17. qc.c.extu

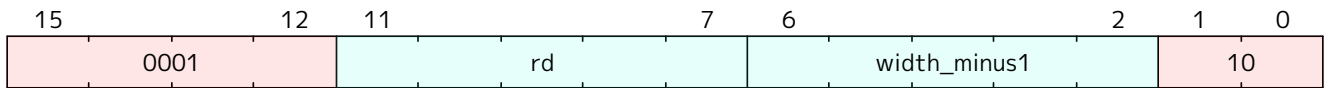
Synopsis

Extract bits unsigned

Mnemonic

```
qc.c.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
Xqci	>= 0
Xqcibm	>= 0

5.18. qc.c.mienter.nest

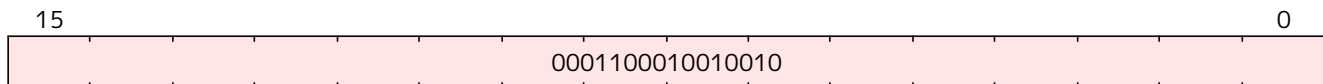
Synopsis

Machine mode interrupt enter

Mnemonic

```
qc.c.mienter.nest
```

Encoding



Description

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

Decode Variables

qc.c.mienter.nest has no 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
Xqci	>= 0
Xqciint	>= 0

DRAFT

5.19. qc.c.mienter

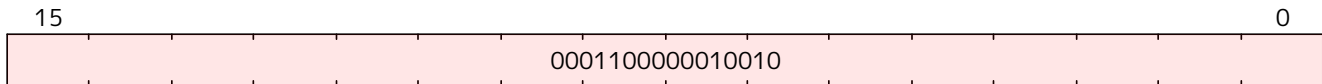
Synopsis

Machine mode interrupt enter

Mnemonic

qc.c.mienter

Encoding



Description

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

Decode Variables

qc.c.mienter has no 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
Xqci	>= 0
Xqciint	>= 0

DRAFT

5.20. qc.c.mileaveret

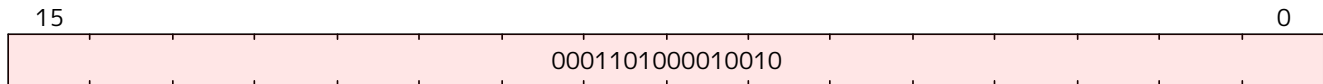
Synopsis

Machine mode interrupt exit

Mnemonic

qc.c.mileaveret

Encoding



Description

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

Decode Variables

qc.c.mileaveret has no 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
Xqci	>= 0
Xqciint	>= 0

DRAFT

5.21. qc.c.muladdi

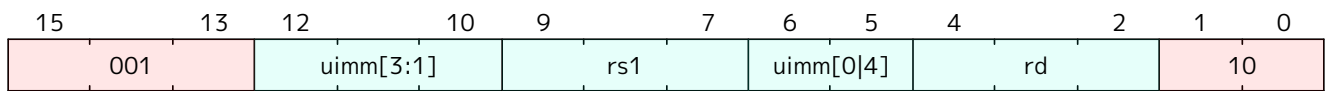
Synopsis

Multiply and accumulate (Immediate)

Mnemonic

```
qc.c.muladdi 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
Xqci	>= 0
Xqciac	>= 0

5.22. qc.c.mveqz

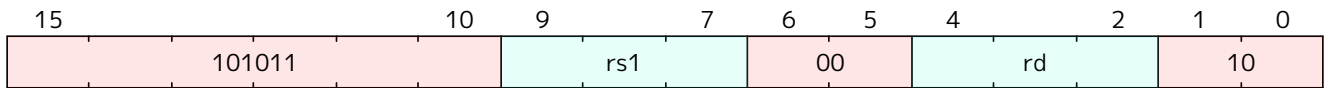
Synopsis

Conditional Move if equal to zero

Mnemonic

```
qc.c.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
Xqci	>= 0
Xqcicm	>= 0

5.23. qc.c.setint

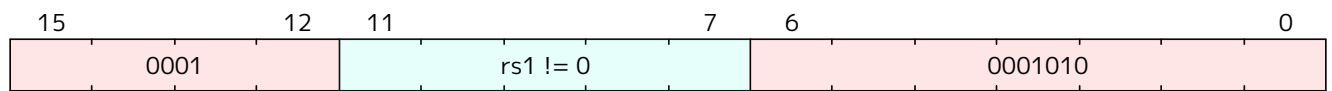
Synopsis

Set interrupt (Register)

Mnemonic

```
qc.c.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[mclicip0 + idx].sw_read();  
CSR[mclicip0 + idx].sw_write(pre_csr | (1 << bit));
```

Included in

Extension	Minimum version
Xqci	>= 0
Xqciint	>= 0

5.24. qc.clo

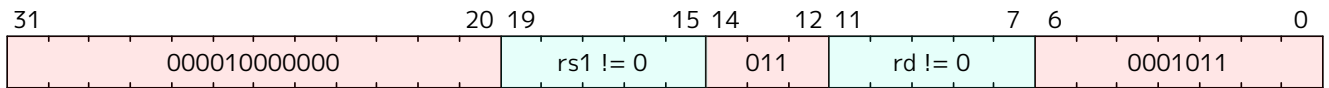
Synopsis

Count leading ones

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.25. qc.clrinti

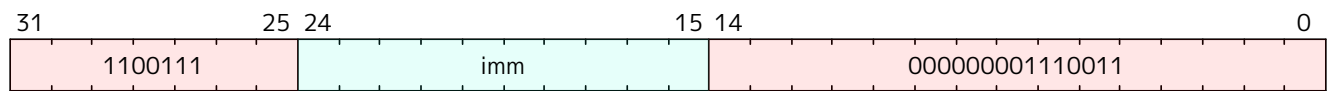
Synopsis

Clear interrupt (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqciint	>= 0

5.26. qc.compress2

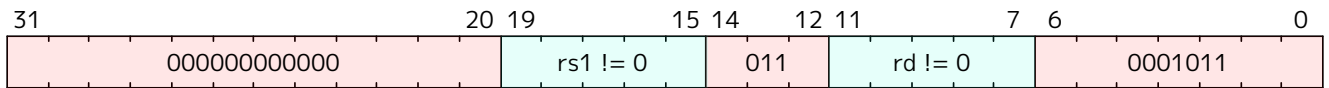
Synopsis

Bit compression (every 2nd bit)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.27. qc.compress3

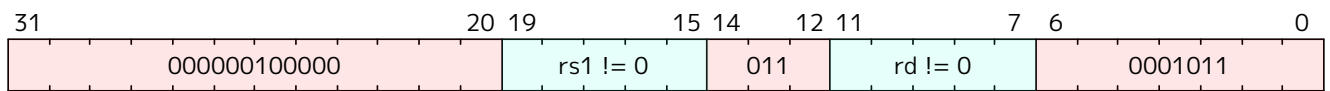
Synopsis

Bit compression (every 3rd bit)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.28. qc.csrrwr

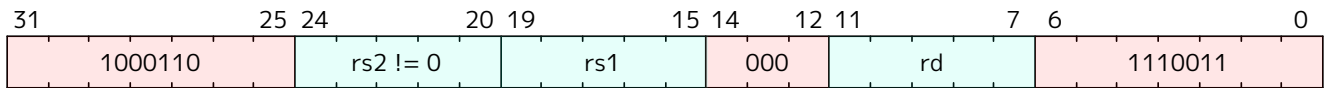
Synopsis

Atomic Read/Write CSR (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicsr	>= 0

5.29. qc.csrrwri

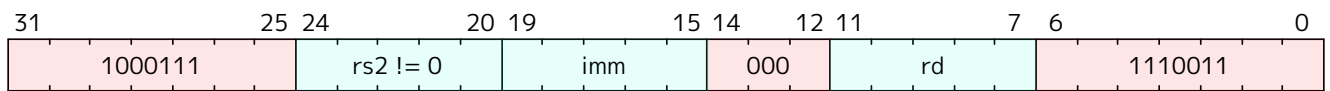
Synopsis

Atomic Read/Write CSR (Register) Immediate

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicsr	>= 0

5.30. qc.cto

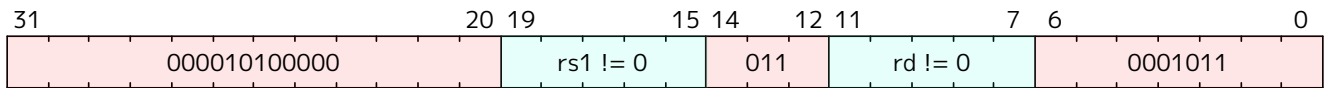
Synopsis

Count trailing ones

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.31. qc.e.addai

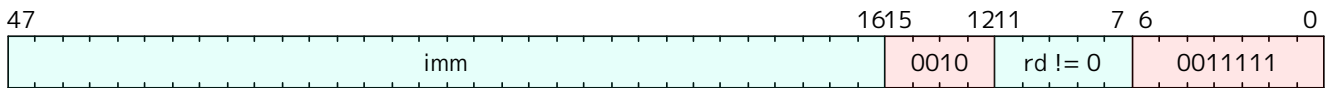
Synopsis

Add immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.32. qc.e.addi

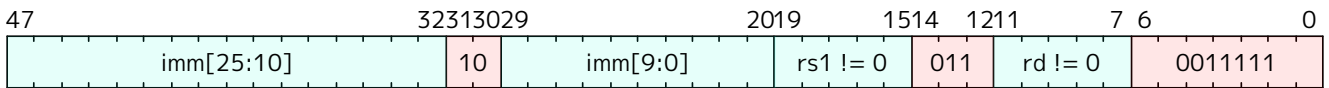
Synopsis

Add immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.33. qc.e.andai

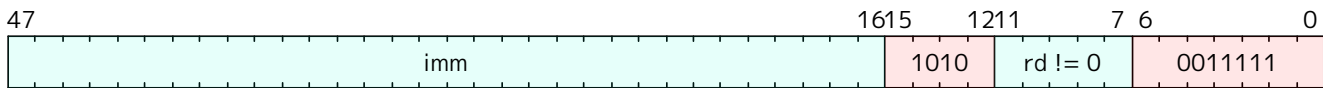
Synopsis

And immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.34. qc.e.andi

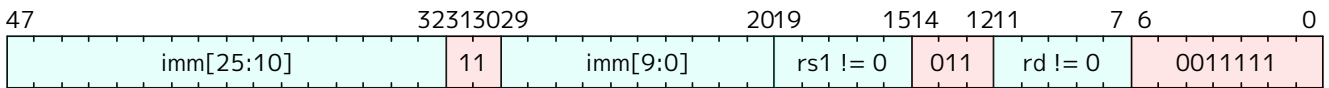
Synopsis

Add immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.35. qc.e.beqi

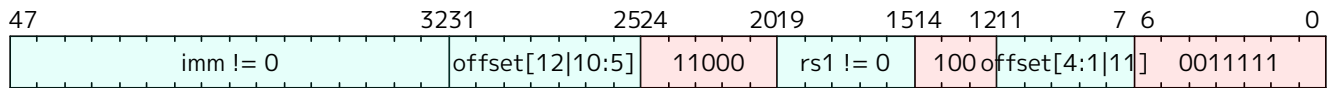
Synopsis

Branch on equal (immediate)

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcibi	>= 0

5.36. qc.e.bgei

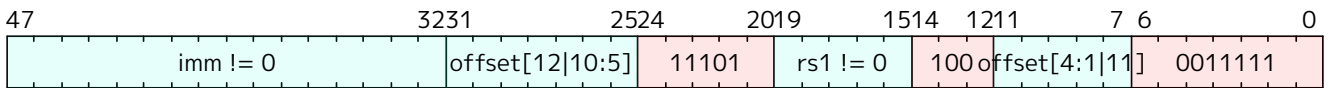
Synopsis

Branch on greater than or equal (immediate)

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcibi	>= 0

5.37. qc.e.bgeui

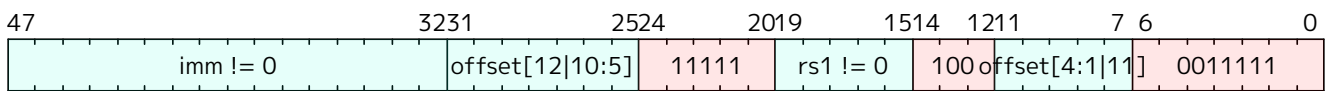
Synopsis

Branch on greater than or equal unsigned (immediate)

Mnemonic

qc.e.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
Xqci	>= 0
Xqcibi	>= 0

5.38. qc.e.blti

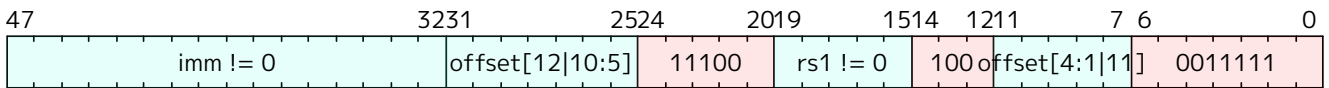
Synopsis

Branch on less than (immediate)

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcibi	>= 0

5.39. qc.e.bltui

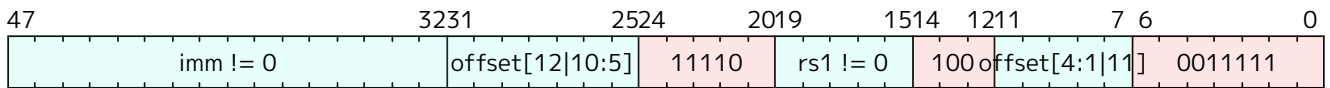
Synopsis

Branch on less than unsigned (immediate)

Mnemonic

```
qc.e.bltui  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
Xqci	>= 0
Xqcibi	>= 0

5.40. qc.e.bnei

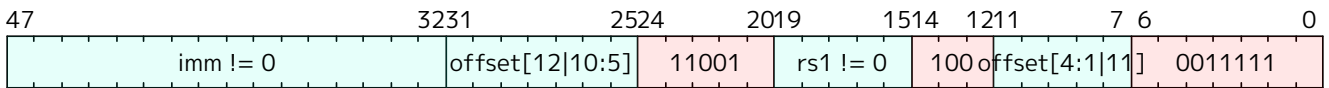
Synopsis

Branch on not equal (immediate)

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcibi	>= 0

5.41. qc.ej

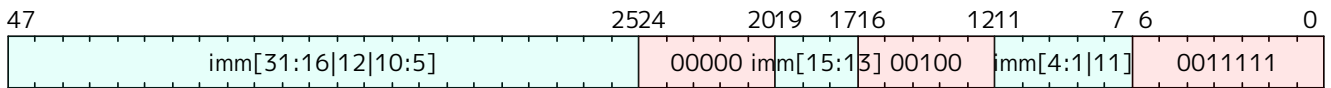
Synopsis

Jump

Mnemonic

qc.e.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
Xqci	>= 0
Xqcilb	>= 0

5.42. qc.e.jal

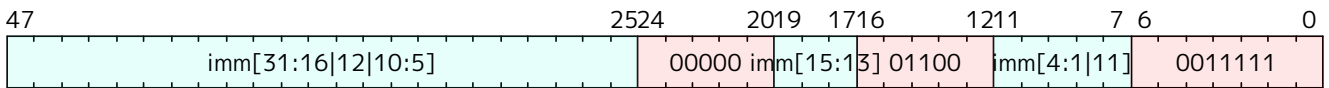
Synopsis

Jump and link

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilb	>= 0

5.43. qc.e.lb

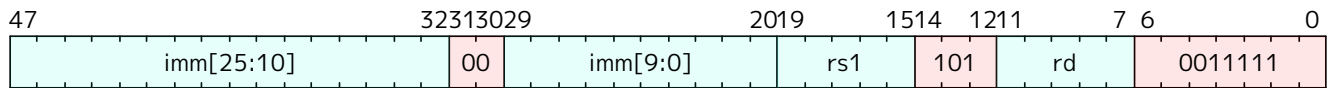
Synopsis

Load byte

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.44. qc.e.lbu

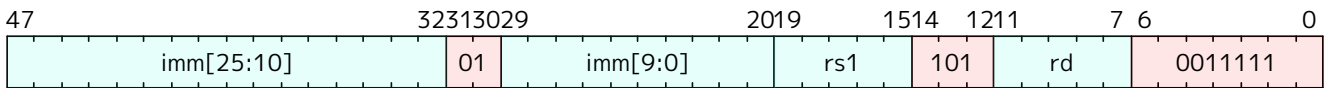
Synopsis

Load byte unsigned

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.45. qc.e.lh

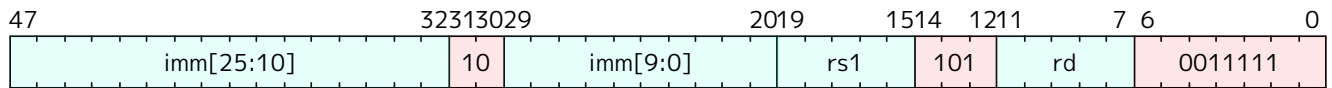
Synopsis

Load halfword

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.46. qc.e.lhu

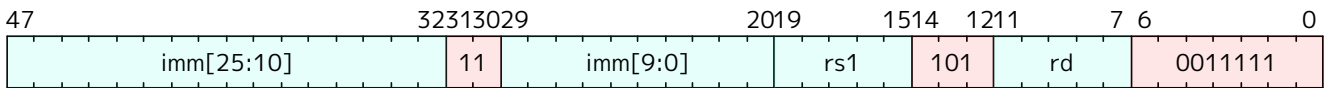
Synopsis

Load halfword unsigned

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.47. qc.e.li

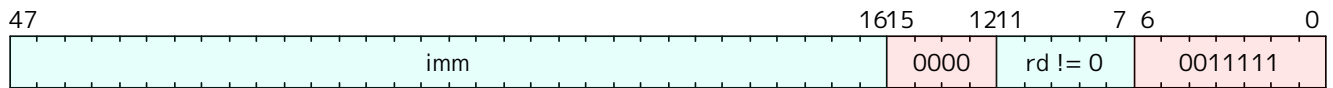
Synopsis

Load immediate large

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcili	>= 0

5.48. qc.e.lw

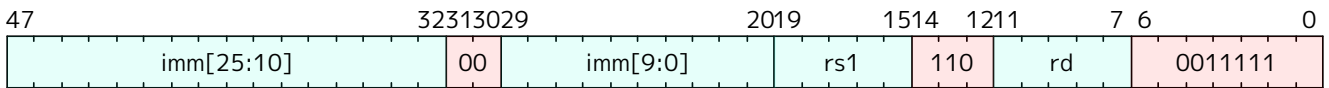
Synopsis

Load word

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.49. qc.e.orai

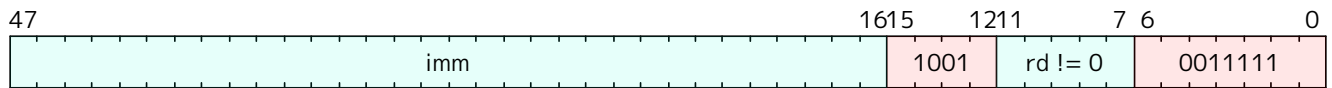
Synopsis

Or immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.50. qc.e.ori

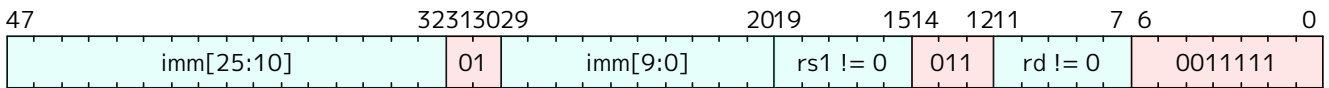
Synopsis

Or immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.51. qc.e.sb

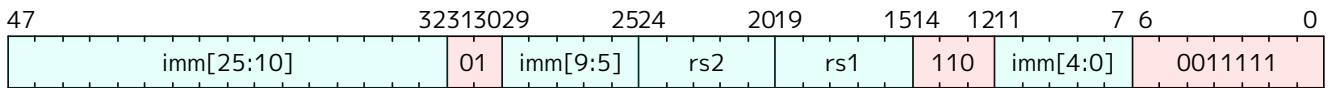
Synopsis

Store byte

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.52. qc.e.sh

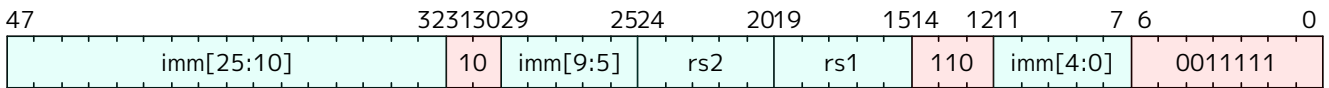
Synopsis

Store halfword

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.53. qc.e.sw

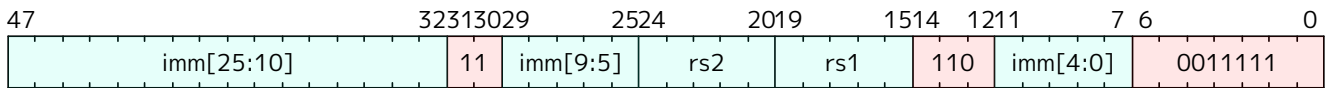
Synopsis

Store word

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilo	>= 0

5.54. qc.e.xorai

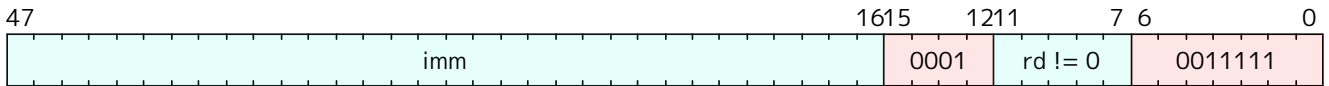
Synopsis

Exclusive Or immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.55. qc.e.xori

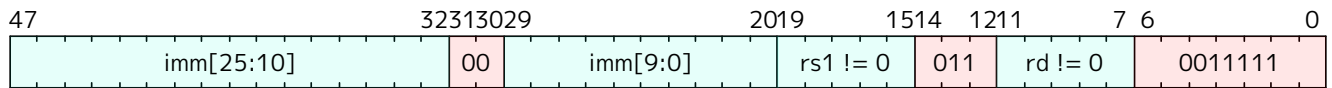
Synopsis

Exclusive Or immediate

Mnemonic

```
qc.e.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
Xqci	>= 0
Xqcilia	>= 0

5.56. qc.expand2

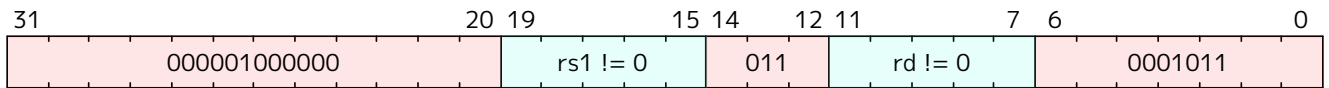
Synopsis

Bit expansion (every 2nd bit)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.57. qc.expand3

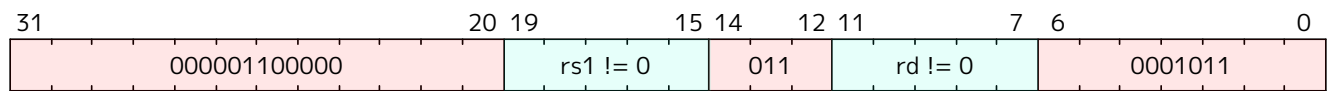
Synopsis

Bit expansion (every 3rd bit)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.58. qc.ext

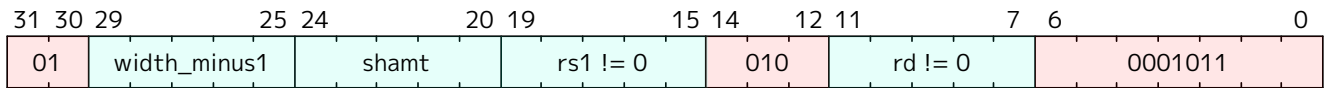
Synopsis

Extract bits signed

Mnemonic

```
qc.ext 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
Xqci	>= 0
Xqcibm	>= 0

5.59. qc.extd

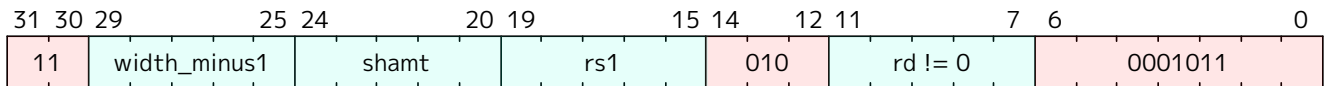
Synopsis

Extract bits from pair signed (Immediate)

Mnemonic

```
qc.extd 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
Xqci	≥ 0
Xqcibm	≥ 0

5.60. qc.extdpr

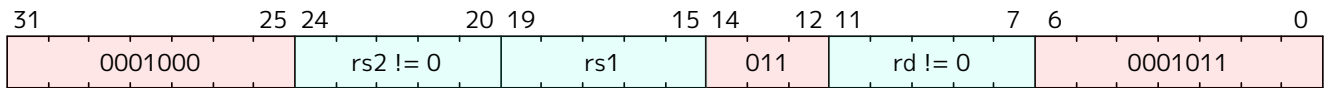
Synopsis

Extract bits from pair signed, packed descriptor (Register)

Mnemonic

```
qc.extdpr 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
Xqci	>= 0
Xqcibm	>= 0

5.61. qc.extdprh

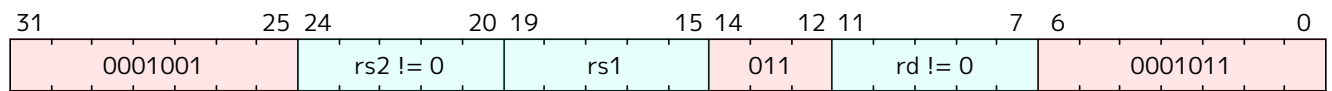
Synopsis

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

Mnemonic

```
qc.extdprh 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
Xqci	>= 0
Xqcibm	>= 0

5.62. qc.extdr

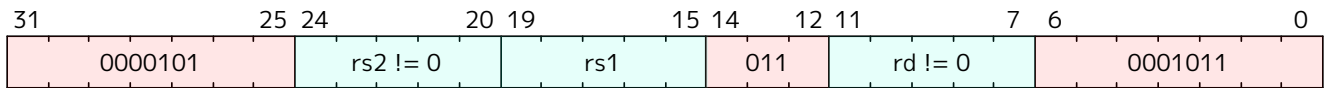
Synopsis

Extract bits from pair signed (Register)

Mnemonic

```
qc.extdr 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
Xqci	>= 0
Xqcibm	>= 0

5.63. qc.extdu

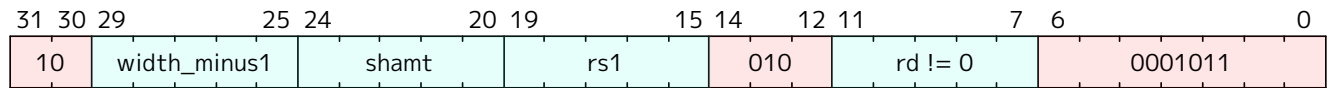
Synopsis

Extract bits from pair unsigned (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.64. qc.extdupr

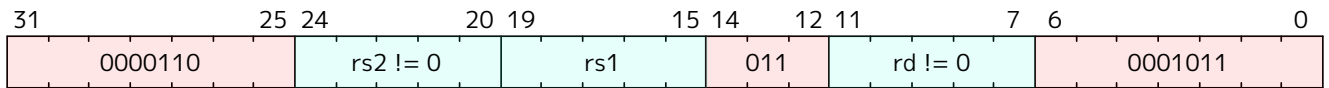
Synopsis

Extract bits from pair unsigned, packed descriptor (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.65. qc.extduprh

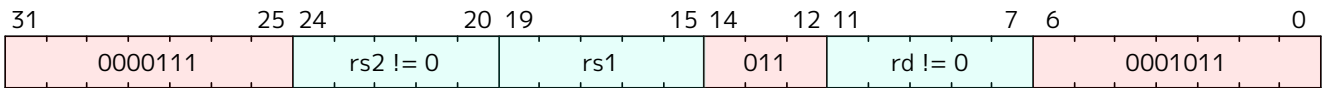
Synopsis

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

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.66. qc.extdur

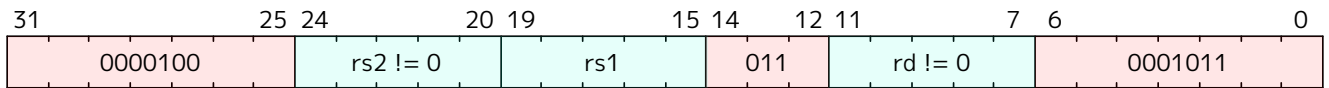
Synopsis

Extract bits from pair unsigned (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.67. qc.extu

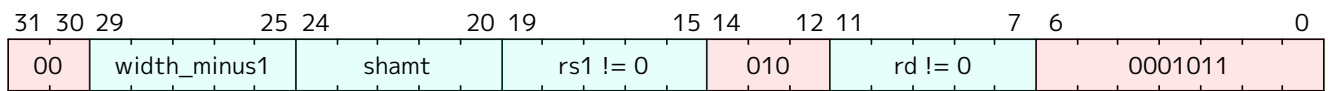
Synopsis

Extract bits unsigned

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.68. qc.insb

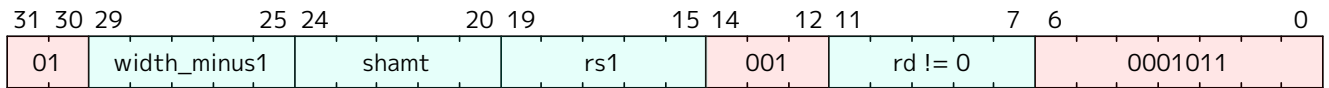
Synopsis

Insert bits (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.69. qc.insbh

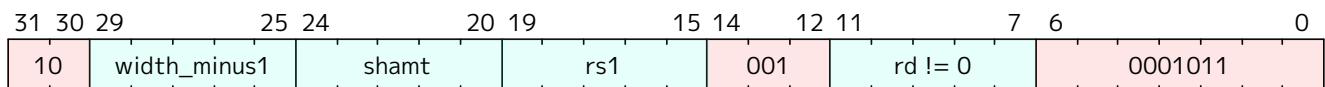
Synopsis

Insert bits in 64-bit higher part (Register)

Mnemonic

```
qc.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 $(\text{width_minus1} + 1)$ (1..32), and the offset of the subset is determined by `shamt`. In case when $\text{width} + \text{offset} \leq 32$, 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
Xqci	≥ 0
Xqcibm	≥ 0

5.70. qc.insbhr

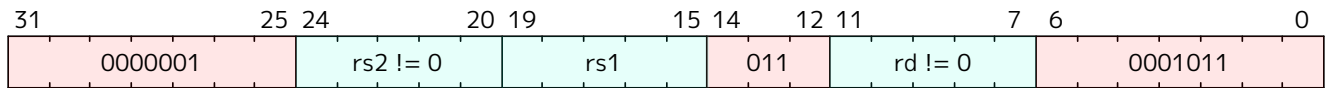
Synopsis

Insert bits in 64-bit higher part (Register)

Mnemonic

```
qc.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 bundary. 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
Xqci	>= 0
Xqcibm	>= 0

5.71. qc.insbi

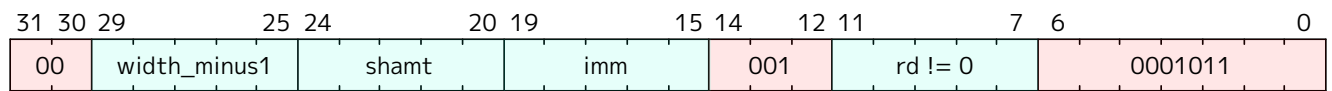
Synopsis

Insert bits (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.72. qc.insbpr

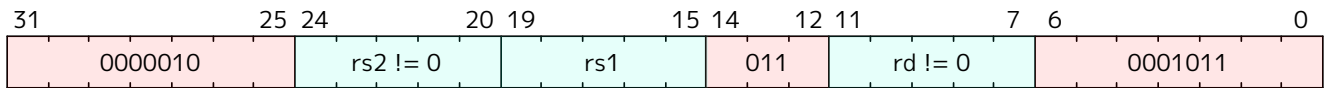
Synopsis

Insert bits, packed descriptor (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.73. qc.insbprh

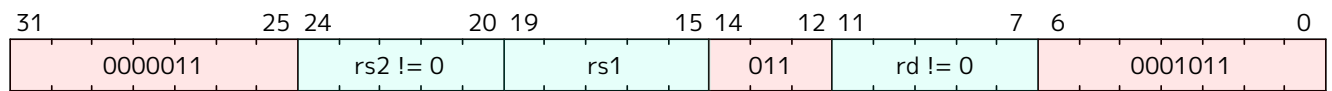
Synopsis

Insert bits, packed descriptor high part (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.74. qc.insbr

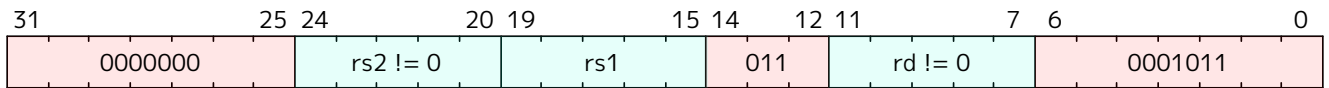
Synopsis

Insert bits (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.75. qc.insbri

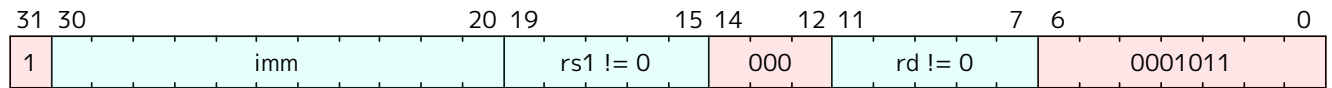
Synopsis

Insert bits (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcibm	>= 0

5.76. qc.li

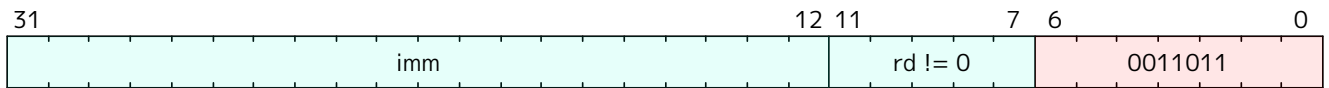
Synopsis

Load immediate large

Mnemonic

```
qc.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
Xqci	>= 0
Xqcili	>= 0

5.77. qc.lieq

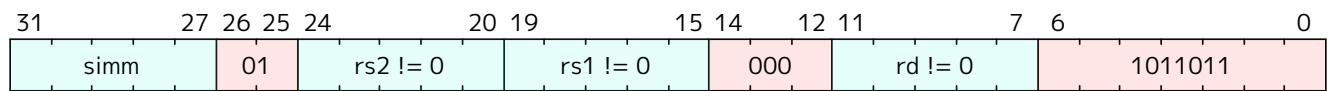
Synopsis

Conditional load immediate if equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.78. qc.lieqi

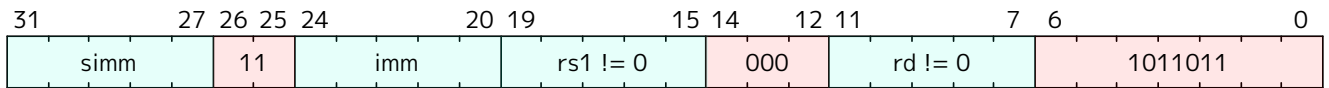
Synopsis

Conditional load immediate if equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.79. qc.lige

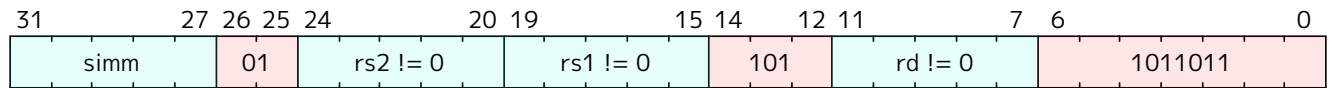
Synopsis

Conditional load immediate if great or equal than (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.80. qc.ligei

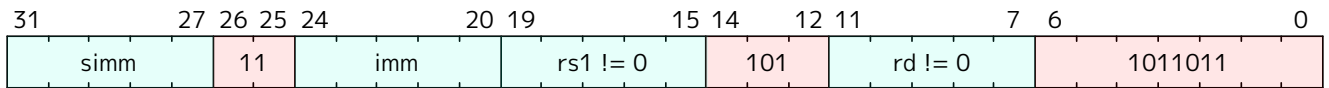
Synopsis

Conditional load immediate if great or equal than (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.81. qc.ligeu

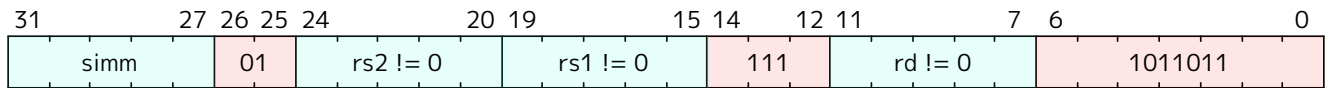
Synopsis

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

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.82. qc.ligeui

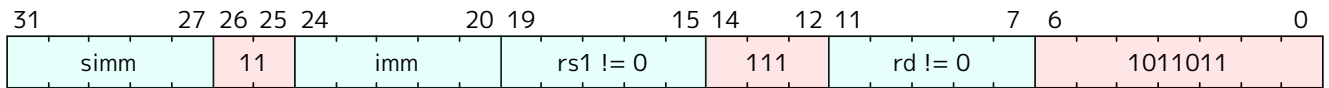
Synopsis

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

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.83. qc.lilt

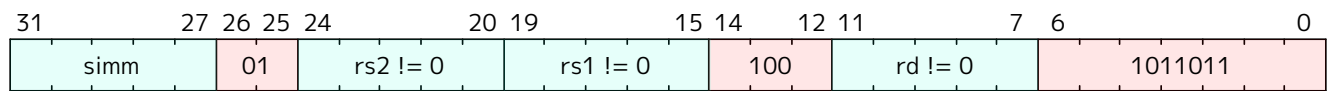
Synopsis

Conditional load immediate if less than (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.84. qc.lilti

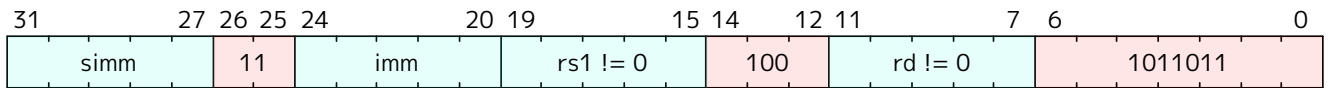
Synopsis

Conditional load immediate if less than (Immediate)

Mnemonic

```
qc.lilti 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
Xqci	>= 0
Xqcicli	>= 0

5.85. qc.liltu

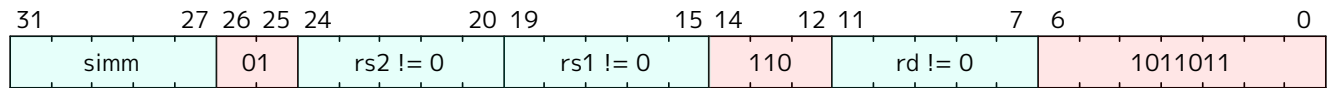
Synopsis

Conditional load immediate if less than unsigned (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.86. qc.liltui

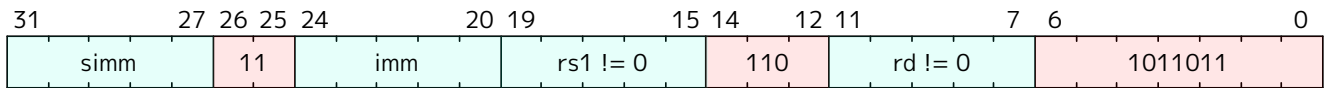
Synopsis

Conditional load immediate if less than unsigned (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.87. qc.line

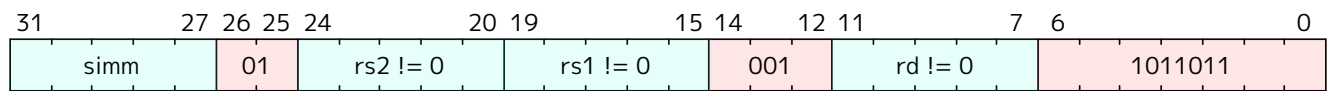
Synopsis

Conditional load immediate if not equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.88. qc.linei

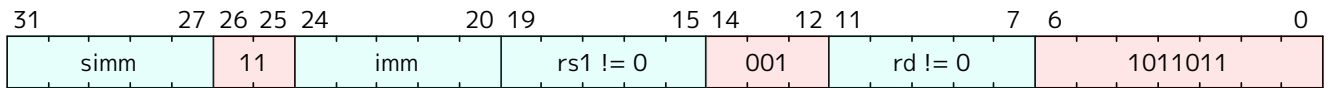
Synopsis

Conditional load immediate if not equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicli	>= 0

5.89. qc.lrb

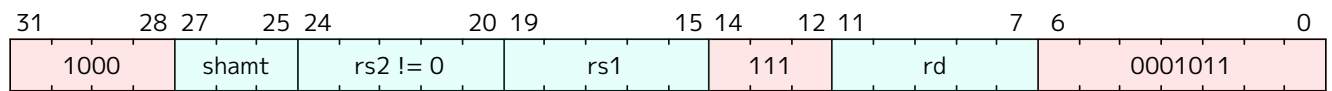
Synopsis

Load indexed byte

Mnemonic

```
qc.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
Xqci	>= 0
Xqcisls	>= 0

5.90. qc.lrbu

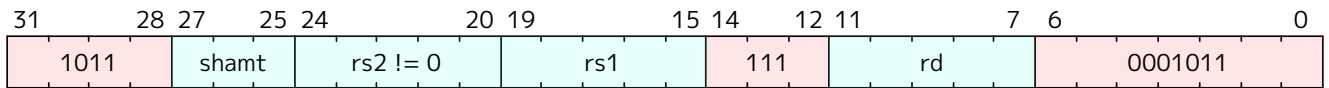
Synopsis

Load indexed unsigned byte

Mnemonic

```
qc.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
Xqci	>= 0
Xqcisls	>= 0

5.91. qc.lrh

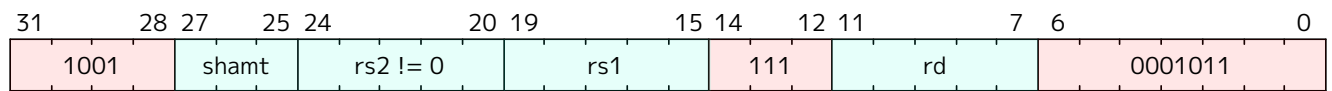
Synopsis

Load indexed halfword

Mnemonic

```
qc.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
Xqci	>= 0
Xqcisls	>= 0

5.92. qc.lrhu

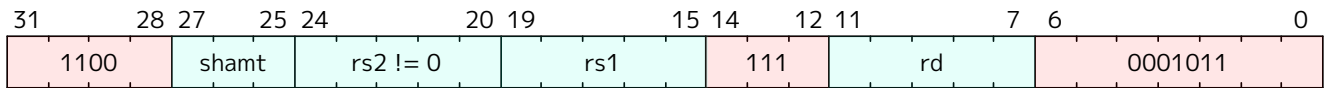
Synopsis

Load indexed unsigned halfword

Mnemonic

```
qc.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
Xqci	>= 0
Xqcisls	>= 0

5.93. qc.lrw

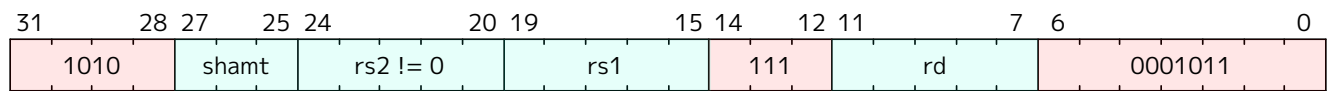
Synopsis

Load indexed word

Mnemonic

```
qc.lrw rd, rs1, rs2, shamt
```

Encoding



Description

Load 32 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
Xqci	>= 0
Xqcisls	>= 0

5.94. qc.lwm

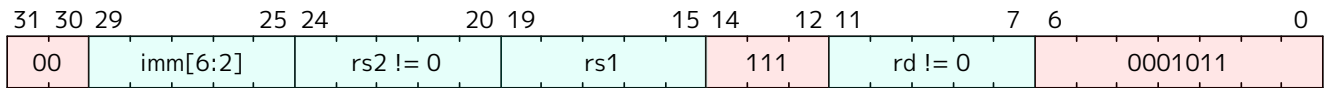
Synopsis

Load word multiple

Mnemonic

```
qc.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
Xqci	>= 0
Xqcilsm	>= 0

5.95. qc.lwmi

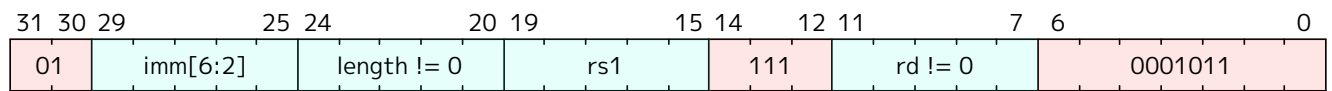
Synopsis

Load word multiple (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcilsm	>= 0

5.96. qc.muladdi

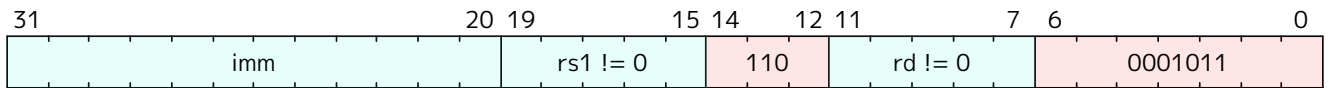
Synopsis

Multiply and accumulate (Immediate)

Mnemonic

```
qc.muladdi 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
Xqci	>= 0
Xqciac	>= 0

5.97. qc.mveq

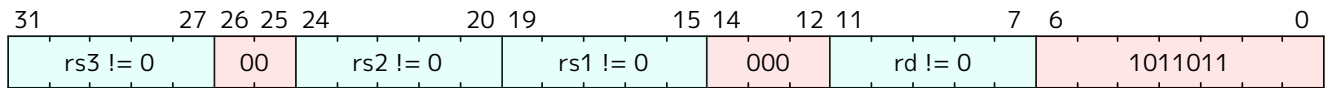
Synopsis

Conditional move if equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.98. qc.mveqi

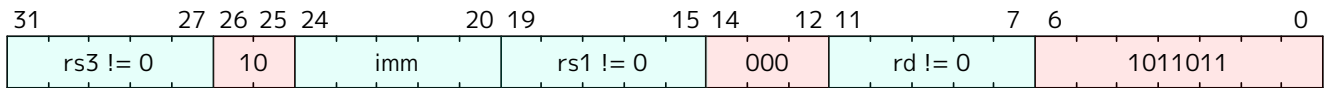
Synopsis

Conditional move if equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.99. qc.mvge

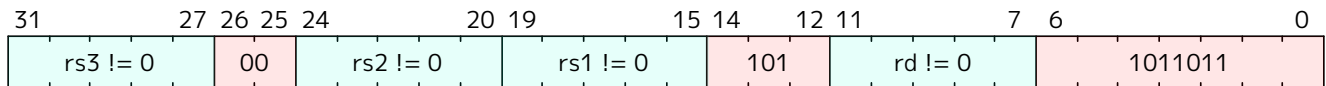
Synopsis

Conditional move if great or equal than (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.100. qc.mvgei

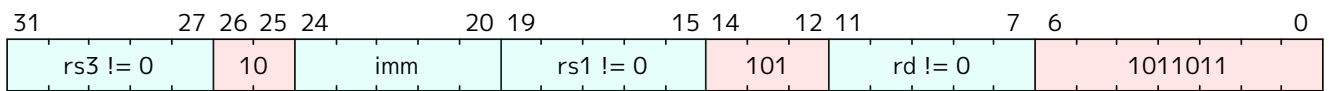
Synopsis

Conditional move if great or equal than (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.101. qc.mvgeu

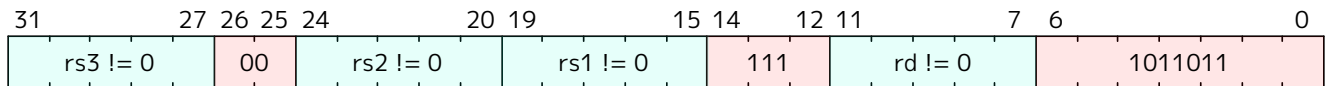
Synopsis

Conditional move if great or equal than unsigned (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.102. qc.mvgeui

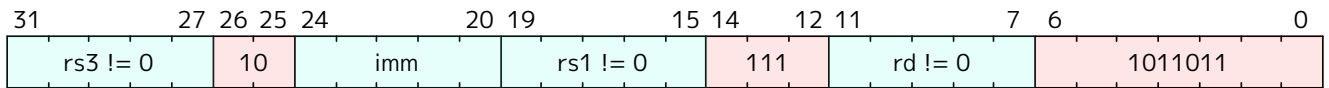
Synopsis

Conditional move if great or equal than unsigned (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.103. qc.mvlt

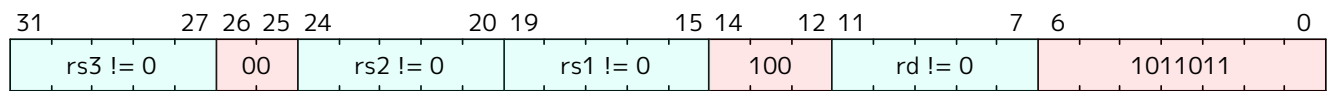
Synopsis

Conditional move if less than (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.104. qc.mvlti

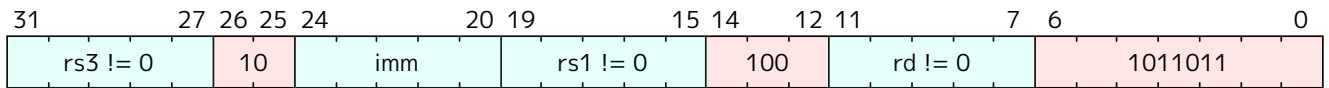
Synopsis

Conditional move if less than (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.105. qc.mvltu

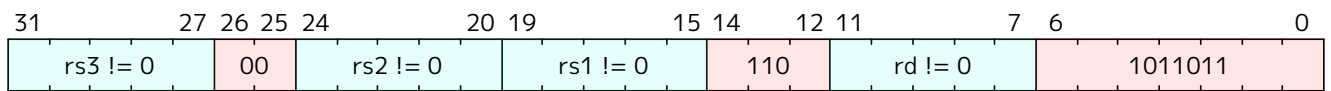
Synopsis

Conditional move if less than unsigned (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.106. qc.mvltui

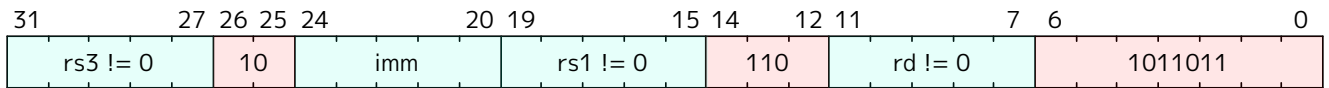
Synopsis

Conditional move if less than unsigned (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.107. qc.mvne

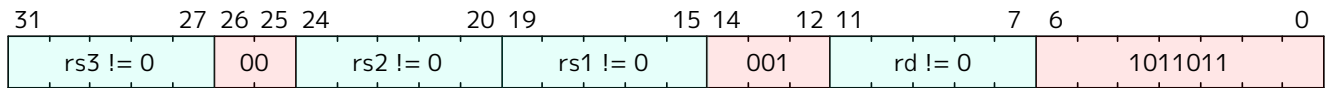
Synopsis

Conditional move if not equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.108. qc.mvnei

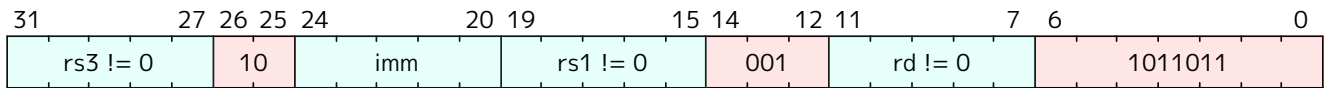
Synopsis

Conditional move if not equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcicm	>= 0

5.109. qc.norm

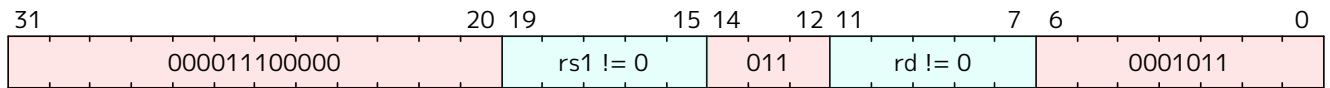
Synopsis

Signed Normalization

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.110. qc.normeu

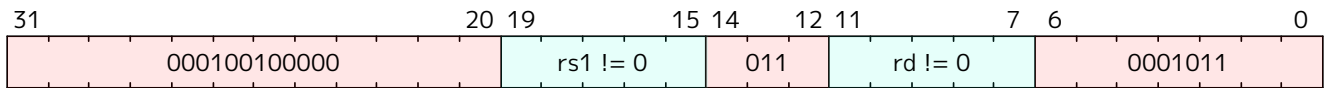
Synopsis

Unsigned Even Normalization

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.111. qc.normu

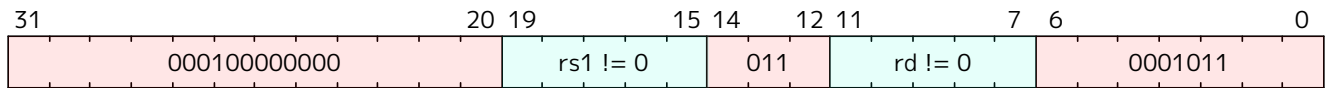
Synopsis

Unsigned Normalization

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.112. qc.selecteqi

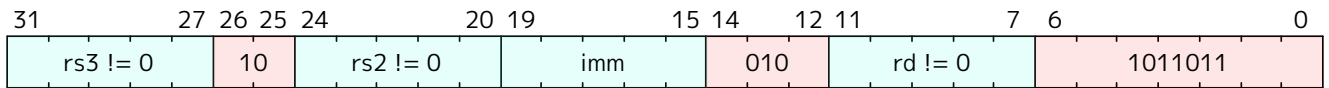
Synopsis

Select load immediate or register if equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.113. qc.selectieq

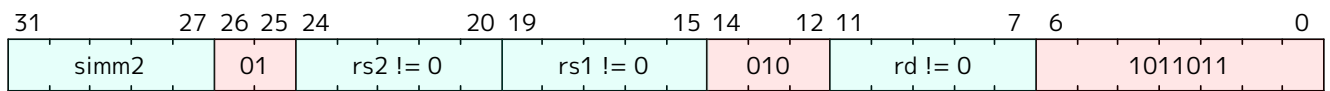
Synopsis

Select load immediate or register if equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.114. qc.selectieqi

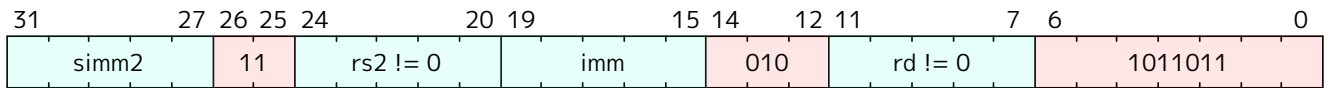
Synopsis

Select load immediate or register if equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.115. qc.selectiieq

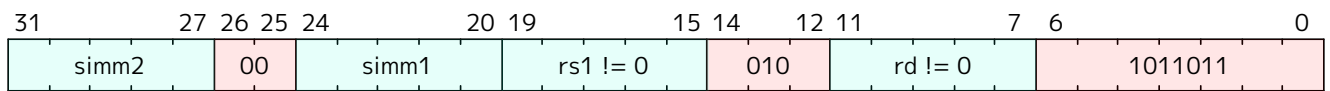
Synopsis

Select load immediate if equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.116. qc.selectiine

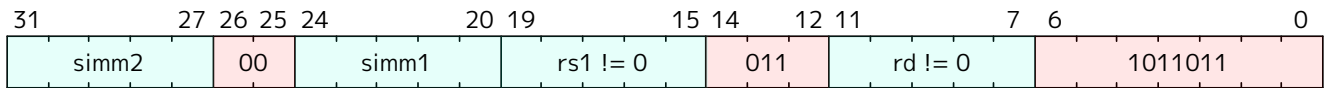
Synopsis

Select load immediate if not equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.117. qc.selectine

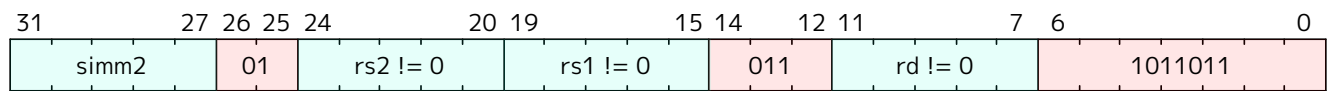
Synopsis

Select load immediate or register if not equal (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.118. qc.selectinei

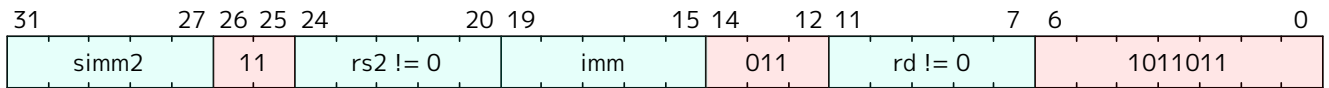
Synopsis

Select load immediate or register if not equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.119. qc.selectnei

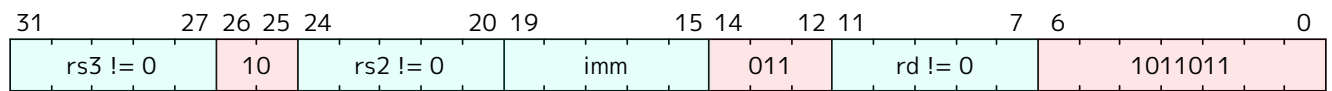
Synopsis

Select load immediate or register if not equal (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcics	>= 0

5.120. qc.setinti

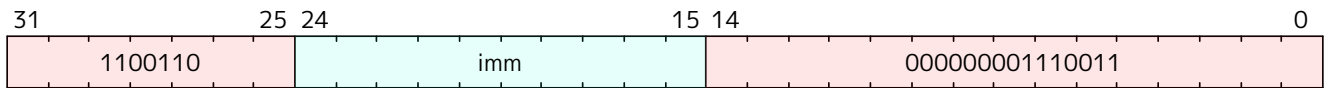
Synopsis

Set interrupt (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqciint	>= 0

5.121. qc.setwm

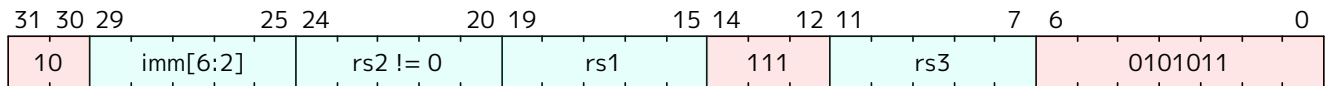
Synopsis

Set word multiple (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcilsm	>= 0

5.122. qc.setwmi

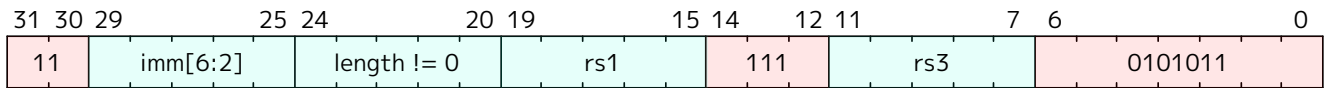
Synopsis

Set word multiple (Immediate)

Mnemonic

```
qc.setwmi rs3, length, 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
Xqci	>= 0
Xqcilsm	>= 0

5.123. qc.shladd

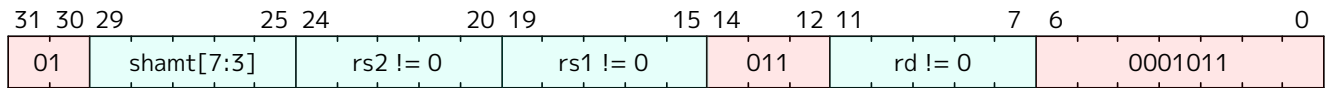
Synopsis

Shift left and add (immediate)

Mnemonic

```
qc.shladd  rs1, rs2, shamt
```

Encoding



Description

Left shift *rs1* by *shamt* and add the value in *rs2*.

Decode Variables

```
Bits<8> shamt = {$encoding[29:25], 3'd0};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = (X[rs1] << shamt) + X[rs2];
```

Included in

Extension	Minimum version
Xqci	>= 0.2
Xqciac	>= 0

5.124. qc.slasat

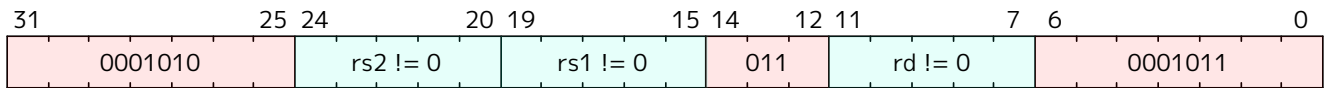
Synopsis

Saturating arithmetic left shift

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.125. qc.sllsat

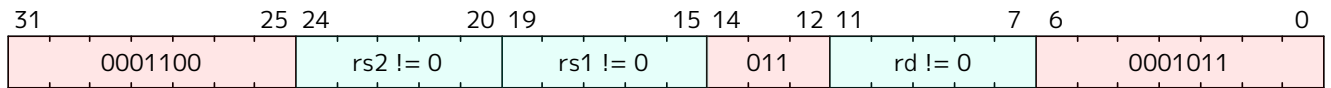
Synopsis

Saturating logical left shift

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.126. qc.srb

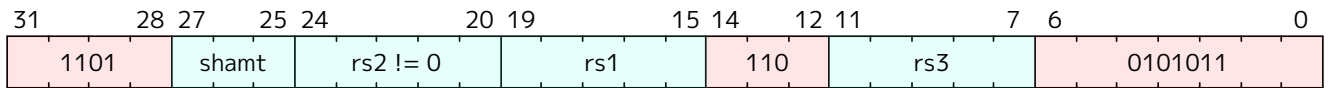
Synopsis

Store indexed byte

Mnemonic

```
qc.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
Xqci	>= 0
Xqcisls	>= 0

5.127. qc.srh

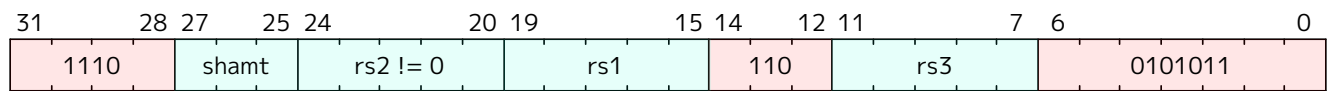
Synopsis

Store indexed halfword

Mnemonic

qc.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
Xqci	>= 0
Xqcisls	>= 0

5.128. qc.srw

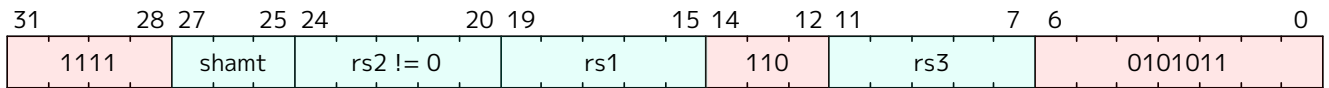
Synopsis

Store indexed word

Mnemonic

```
qc.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
Xqci	>= 0
Xqcisls	>= 0

5.129. qc.subsat

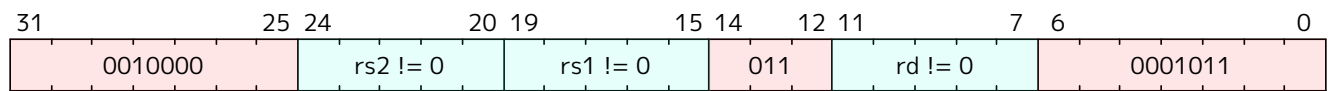
Synopsis

Saturating signed subtraction

Mnemonic

qc.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
Xqci	>= 0
Xqcia	>= 0

5.130. qc.subusat

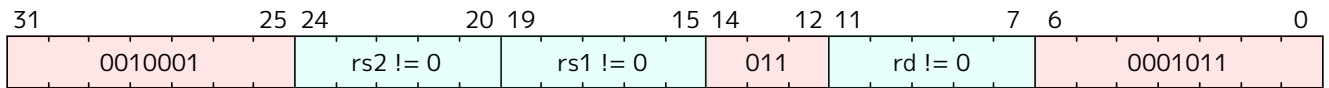
Synopsis

Saturating unsigned subtraction

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.131. qc.swm

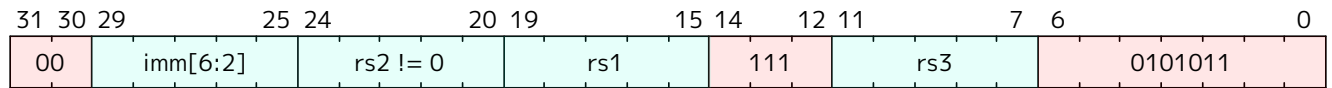
Synopsis

Store word multiple

Mnemonic

```
qc.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
Xqci	>= 0
Xqcilsm	>= 0

5.132. qc.swmi

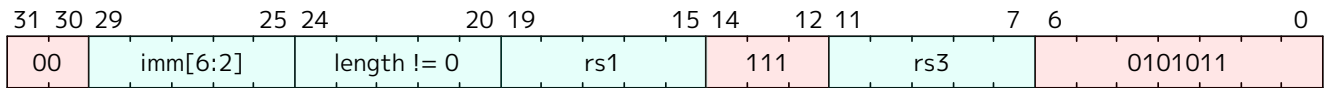
Synopsis

Store word multiple (immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcilsm	>= 0

5.133. qc.wrap

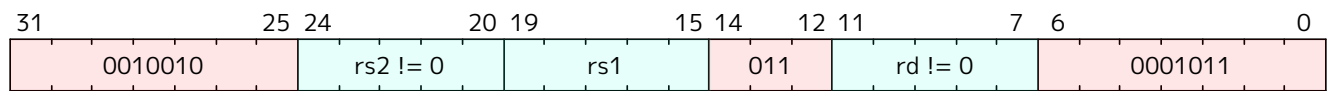
Synopsis

Wraparound (Register)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 0

5.134. qc.wrapi

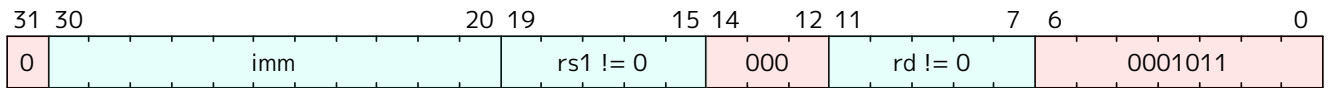
Synopsis

Wraparound (Immediate)

Mnemonic

```
qc.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
Xqci	>= 0
Xqcia	>= 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. 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();

```

6.8. 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;
}
}
```

6.9. 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;
```

6.10. 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;
}
```

6.11. implemented? (builtin)

Return true if the implementation supports extension.

Return Type	Boolean
Arguments	ExtensionName extension

6.12. in_naturally_aligned_region?

Checks if a length-bit access starting at address lies entirely within an N-bit naturally-aligned region.

Return Type	Boolean
Arguments	XReg address, U32 length

```
XReg Mask = (N / 8) - 1;
return (address & ~mask) == ((address + length - 1) & ~mask);
```

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. misaligned_is_atomic?

Returns true if an access starting at physical_address that is N bits long is atomic.

This function takes into account any Atomicity Granule PMAs, so **it should not be used for load-reserved/store-conditional**, since those PMAs do not apply to those accesses.

Return Type	Boolean
Arguments	Bits<PHYS_ADDR_WIDTH> physical_address

```
return false if (MAX_MISALIGNED_ATOMICITY_GRANULE_SIZE == 0);
if (pma_applies?(PmaAttribute::MAG16, physical_address, N) &&
    in_naturally_aligned_region?(M, 128>(physical_address_address, N)) {
    return true;
} else if (pma_applies?(PmaAttribute::MAG8, physical_address, N) &&
    in_naturally_aligned_region?(XLEN, 64>(physical_address, N)) {
    return true;
} else if (pma_applies?(PmaAttribute::MAG4, physical_address, N) &&
    in_naturally_aligned_region?(XLEN, 32>(physical_address, N)) {
    return true;
} else if (pma_applies?(PmaAttribute::MAG2, physical_address, N) &&
    in_naturally_aligned_region?(XLEN, 16>(physical_address, N)) {
    return true;
} else {
    return false;
}
```

6.17. 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;
}

```

6.18. 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");
}

```

6.19. 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;
}
```

6.20. 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));
```

6.21. 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

6.22. 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;
}
}
```

6.23. 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

6.24. 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;

```

6.25. 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);
}

```

6.26. 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[pmpcfg_idx]) >> 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, -;

```

6.27. 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[pmpcfg_idx]) >> 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, -;

```

6.28. 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();

```

6.29. read_memory

Read from virtual memory.

Return Type	Bits<len>
Arguments	XReg virtual_address

```

Boolean aligned = is_naturally_aligned<XLEN, len>(virtual_address);
XReg physical_address;
if (aligned) {
    return read_memory_aligned<len>(virtual_address);
}
if (MAX_MISALIGNED_ATOMICTY_GRANULE_SIZE > 0) {
    assert(MISALIGNED_LDST_EXCEPTION_PRIORITY == "low");
    physical_address = (CSR[misa].S == 1) ? translate(virtual_address, MemoryOperation
::Read) : virtual_address;
    if (misaligned_is_atomic?<len>(physical_address)) {
        access_check<len>(physical_address, virtual_address, MemoryOperation::Read,
ExceptionCode::LoadAccessFault);
        return read_physical_memory<len>(physical_address);
    }
}
if (!MISALIGNED_LDST) {
    if (MISALIGNED_LDST_EXCEPTION_PRIORITY == "low") {
        physical_address = (CSR[misa].S == 1) ? translate(virtual_address,
MemoryOperation::Read) : virtual_address;
        access_check<len>(physical_address, virtual_address, MemoryOperation::Read,
ExceptionCode::LoadAccessFault);
    }
    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");
    }
}
}

```

6.30. 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);
```

6.31. read_physical_memory (builtin)

Read from physical memory.

Return Type	Bits<len>
Arguments	XReg paddr

6.32. 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;
}
```

6.33. 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;
}

```

6.34. 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;
}

```

6.35. 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;
}

```

6.36. 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;

```

6.37. 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

6.38. virtual_mode?

Returns True if the current mode is virtual (VS or VU).

Return Type	Boolean
Arguments	

```
return (mode() == PrivilegeMode::VS) || (mode() == PrivilegeMode::VU);
```

6.39. 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;
}

```

6.40. 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;
}

```

6.41. 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);
XReg physical_address;
if (aligned) {
    write_memory_aligned<len>(virtual_address, value);
}
if (MAX_MISALIGNED_ATOMICITY_GRANULE_SIZE > 0) {
    assert(MISALIGNED_LDST_EXCEPTION_PRIORITY == "low");
    physical_address = (CSR[misa].S == 1) ? translate(virtual_address, MemoryOperation
::Write) : virtual_address;
    if (misaligned_is_atomic?<len>(physical_address)) {

```

```

    access_check<len>(physical_address, virtual_address, MemoryOperation::Write,
ExceptionCode::StoreAmoAccessFault);
    write_physical_memory<len>(physical_address, value);
}
}
if (!MISALIGNED_LDST) {
    if (MISALIGNED_LDST_EXCEPTION_PRIORITY == "low") {
        physical_address = (CSR[misa].S == 1) ? translate(virtual_address,
MemoryOperation::Write) : virtual_address;
        access_check<len>(physical_address, virtual_address, MemoryOperation::Write,
ExceptionCode::StoreAmoAccessFault);
    }
    raise(ExceptionCode::StoreAmoAddressMisaligned, virtual_address);
} else {
    if (MISALIGNED_SPLIT_STRATEGY == "by_byte") {
        for (U32 i = 0; i <= len; i++) {
            write_memory_aligned<88 * i))[7:0]);
        }
    } 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");
    }
}
}

```

6.42. 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);

```

6.43. write_physical_memory (builtin)

Write to physical memory.

Return Type	void
Arguments	XReg paddr, Bits<len> value

6.44. 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;
    }
}
}

```