Qualcomm uC extensions

Version 0.6.0, 2025-01-29

Table of Contents

Licensing and Acknowledgements **1**

Copyright and license information **2**

Acknowledgements **3**

Versions **4**

Version History 4

1. Conventions **8**

2. Extension description **9**

2.1. Sub-extensions 11

2.1.1. Xqcia (0.4.0) 11

2.1.2. Xqciac (0.2.0) 11

2.1.3. Xqcibi (0.2.0) 11

2.1.4. Xqcibm (0.4.0) 11

2.1.5. Xqcicli (0.2.0) 12

2.1.6. Xqcicm (0.2.0) 12

2.1.7. Xqcics (0.2.0) 12

2.1.8. Xqcicsr (0.3.0) 12

2.1.9. Xqciint (0.2.0) 12

2.1.10. Xqcilb (0.2.0) 12

2.1.11. Xqcili (0.2.0) 12

2.1.12. Xqcilia (0.2.0) 12

2.1.13. Xqcilo (0.2.0) 13

2.1.14. Xqcilsm (0.4.0) 13

2.1.15. Xqcisim (0.2.0) 13

2.1.16. Xqcisls (0.2.0) 13

2.1.17. Xqcisync (0.2.0) 13

3. Instruction summary **14**

3.1. Instructions by sub-extension 19

4. CSR summary **26**

5. Csrs (in alphabetical order) **27**

5.1. qc_mclicie0 28

5.1.1. Attributes 28

5.1.2. Format 28

5.1.3. Field Summary 28

5.1.4. Fields 29

IRQ0 29

IRQ1 29

IRQ2 30

IRQ3 30

IRQ4 30

IRQ5 31

IRQ6 31

IRQ7 31
IRQ8 32
IRQ9 32
IRQ10 32
IRQ11 33
IRQ12 33
IRQ13 33
IRQ14 34
IRQ15 34
IRQ16 34
IRQ17 35
IRQ18 35
IRQ19 35
IRQ20 36
IRQ21 36
IRQ22 36
IRQ23 37
IRQ24 37
IRQ25 37
IRQ26 38
IRQ27 38
IRQ28 38
IRQ29 39
IRQ30 39
IRQ31 39
5.2. qc_mclicie1 40
5.2.1. Attributes 40
5.2.2. Format 40
5.2.3. Field Summary 40
5.2.4. Fields 41
IRQ32 41
IRQ33 41
IRQ34 42
IRQ35 42
IRQ36 42
IRQ37 43
IRQ38 43
IRQ39 43
IRQ40 44
IRQ41 44
IRQ42 44
IRQ43 45
IRQ44 45

IRQ45 45
IRQ46 46
IRQ47 46
IRQ48 46
IRQ49 47
IRQ50 47
IRQ51 47
IRQ52 48
IRQ53 48
IRQ54 48
IRQ55 49
IRQ56 49
IRQ57 49
IRQ58 50
IRQ59 50
IRQ60 50
IRQ61 51
IRQ62 51
IRQ63 51
5.3. qc_mclicie2 52
5.3.1. Attributes 52
5.3.2. Format 52
5.3.3. Field Summary 52
5.3.4. Fields 53
IRQ64 53
IRQ65 53
IRQ66 54
IRQ67 54
IRQ68 54
IRQ69 55
IRQ70 55
IRQ71 55
IRQ72 56
IRQ73 56
IRQ74 56
IRQ75 57
IRQ76 57
IRQ77 57
IRQ78 58
IRQ79 58
IRQ80 58
IRQ81 59
IRQ82 59

IRQ83 59
IRQ84 60
IRQ85 60
IRQ86 60
IRQ87 61
IRQ88 61
IRQ89 61
IRQ90 62
IRQ91 62
IRQ92 62
IRQ93 63
IRQ94 63
IRQ95 63
5.4. qc_mclicie3 64
5.4.1. Attributes 64
5.4.2. Format 64
5.4.3. Field Summary 64
5.4.4. Fields 65
IRQ96 65
IRQ97 65
IRQ98 66
IRQ99 66
IRQ100 66
IRQ101 67
IRQ102 67
IRQ103 67
IRQ104 68
IRQ105 68
IRQ106 68
IRQ107 69
IRQ108 69
IRQ109 69
IRQ110 70
IRQ111 70
IRQ112 70
IRQ113 71
IRQ114 71
IRQ115 71
IRQ116 72
IRQ117 72
IRQ118 72
IRQ119 73
IRQ120 73

IRQ121 73
IRQ122 74
IRQ123 74
IRQ124 74
IRQ125 75
IRQ126 75
IRQ127 75
5.5. qc_mclicie4 76
5.5.1. Attributes 76
5.5.2. Format 76
5.5.3. Field Summary 76
5.5.4. Fields 77
IRQ128 77
IRQ129 77
IRQ130 78
IRQ131 78
IRQ132 78
IRQ133 79
IRQ134 79
IRQ135 79
IRQ136 80
IRQ137 80
IRQ138 80
IRQ139 81
IRQ140 81
IRQ141 81
IRQ142 82
IRQ143 82
IRQ144 82
IRQ145 83
IRQ146 83
IRQ147 83
IRQ148 84
IRQ149 84
IRQ150 84
IRQ151 85
IRQ152 85
IRQ153 85
IRQ154 86
IRQ155 86
IRQ156 86
IRQ157 87
IRQ158 87

IRQ159 87
5.6. qc_mclicie5 88
5.6.1. Attributes 88
5.6.2. Format 88
5.6.3. Field Summary 88
5.6.4. Fields 89
IRQ160 89
IRQ161 89
IRQ162 90
IRQ163 90
IRQ164 90
IRQ165 91
IRQ166 91
IRQ167 91
IRQ168 92
IRQ169 92
IRQ170 92
IRQ171 93
IRQ172 93
IRQ173 93
IRQ174 94
IRQ175 94
IRQ176 94
IRQ177 95
IRQ178 95
IRQ179 95
IRQ180 96
IRQ181 96
IRQ182 96
IRQ183 97
IRQ184 97
IRQ185 97
IRQ186 98
IRQ187 98
IRQ188 98
IRQ189 99
IRQ190 99
IRQ191 99
5.7. qc_mclicie6 100
5.7.1. Attributes 100
5.7.2. Format 100
5.7.3. Field Summary 100
5.7.4. Fields 101

IRQ192 101
IRQ193 101
IRQ194 102
IRQ195 102
IRQ196 102
IRQ197 103
IRQ198 103
IRQ199 103
IRQ200 104
IRQ201 104
IRQ202 104
IRQ203 105
IRQ204 105
IRQ205 105
IRQ206 106
IRQ207 106
IRQ208 106
IRQ209 107
IRQ210 107
IRQ211 107
IRQ212 108
IRQ213 108
IRQ214 108
IRQ215 109
IRQ216 109
IRQ217 109
IRQ218 110
IRQ219 110
IRQ220 110
IRQ221 111
IRQ222 111
IRQ223 111
5.8. qc_mclicie7 112
5.8.1. Attributes 112
5.8.2. Format 112
5.8.3. Field Summary 112
5.8.4. Fields 113
IRQ224 113
IRQ225 113
IRQ226 114
IRQ227 114
IRQ228 114
IRQ229 115

IRQ230 115
IRQ231 115
IRQ232 116
IRQ233 116
IRQ234 116
IRQ235 117
IRQ236 117
IRQ237 117
IRQ238 118
IRQ239 118
IRQ240 118
IRQ241 119
IRQ242 119
IRQ243 119
IRQ244 120
IRQ245 120
IRQ246 120
IRQ247 121
IRQ248 121
IRQ249 121
IRQ250 122
IRQ251 122
IRQ252 122
IRQ253 123
IRQ254 123
IRQ255 123
5.9. qc_mclicip0 124
5.9.1. Attributes 124
5.9.2. Format 124
5.9.3. Field Summary 124
5.9.4. Fields 125
IRQ0 125
IRQ1 125
IRQ2 126
IRQ3 126
IRQ4 126
IRQ5 127
IRQ6 127
IRQ7 127
IRQ8 128
IRQ9 128
IRQ10 128
IRQ11 129

IRQ12 129
IRQ13 129
IRQ14 130
IRQ15 130
IRQ16 130
IRQ17 131
IRQ18 131
IRQ19 131
IRQ20 132
IRQ21 132
IRQ22 132
IRQ23 133
IRQ24 133
IRQ25 133
IRQ26 134
IRQ27 134
IRQ28 134
IRQ29 135
IRQ30 135
IRQ31 135
5.10. qc_mclicip1 136
5.10.1. Attributes 136
5.10.2. Format 136
5.10.3. Field Summary 136
5.10.4. Fields 137
IRQ32 137
IRQ33 137
IRQ34 138
IRQ35 138
IRQ36 138
IRQ37 139
IRQ38 139
IRQ39 139
IRQ40 140
IRQ41 140
IRQ42 140
IRQ43 141
IRQ44 141
IRQ45 141
IRQ46 142
IRQ47 142
IRQ48 142
IRQ49 143

IRQ50 143
IRQ51 143
IRQ52 144
IRQ53 144
IRQ54 144
IRQ55 145
IRQ56 145
IRQ57 145
IRQ58 146
IRQ59 146
IRQ60 146
IRQ61 147
IRQ62 147
IRQ63 147
5.11. qc_mclicip2 148
5.11.1. Attributes 148
5.11.2. Format 148
5.11.3. Field Summary 148
5.11.4. Fields 149
IRQ64 149
IRQ65 149
IRQ66 150
IRQ67 150
IRQ68 150
IRQ69 151
IRQ70 151
IRQ71 151
IRQ72 152
IRQ73 152
IRQ74 152
IRQ75 153
IRQ76 153
IRQ77 153
IRQ78 154
IRQ79 154
IRQ80 154
IRQ81 155
IRQ82 155
IRQ83 155
IRQ84 156
IRQ85 156
IRQ86 156
IRQ87 157

IRQ88 157
IRQ89 157
IRQ90 158
IRQ91 158
IRQ92 158
IRQ93 159
IRQ94 159
IRQ95 159
5.12. qc_mclicip3 160
5.12.1. Attributes 160
5.12.2. Format 160
5.12.3. Field Summary 160
5.12.4. Fields 161
IRQ96 161
IRQ97 161
IRQ98 162
IRQ99 162
IRQ100 162
IRQ101 163
IRQ102 163
IRQ103 163
IRQ104 164
IRQ105 164
IRQ106 164
IRQ107 165
IRQ108 165
IRQ109 165
IRQ110 166
IRQ111 166
IRQ112 166
IRQ113 167
IRQ114 167
IRQ115 167
IRQ116 168
IRQ117 168
IRQ118 168
IRQ119 169
IRQ120 169
IRQ121 169
IRQ122 170
IRQ123 170
IRQ124 170
IRQ125 171

IRQ126 171
IRQ127 171
5.13. qc_mclicip4 172
5.13.1. Attributes 172
5.13.2. Format 172
5.13.3. Field Summary 172
5.13.4. Fields 173
IRQ128 173
IRQ129 173
IRQ130 174
IRQ131 174
IRQ132 174
IRQ133 175
IRQ134 175
IRQ135 175
IRQ136 176
IRQ137 176
IRQ138 176
IRQ139 177
IRQ140 177
IRQ141 177
IRQ142 178
IRQ143 178
IRQ144 178
IRQ145 179
IRQ146 179
IRQ147 179
IRQ148 180
IRQ149 180
IRQ150 180
IRQ151 181
IRQ152 181
IRQ153 181
IRQ154 182
IRQ155 182
IRQ156 182
IRQ157 183
IRQ158 183
IRQ159 183
5.14. qc_mclicip5 184
5.14.1. Attributes 184
5.14.2. Format 184
5.14.3. Field Summary 184

5.14.4. Fields 185
IRQ160 185
IRQ161 185
IRQ162 186
IRQ163 186
IRQ164 186
IRQ165 187
IRQ166 187
IRQ167 187
IRQ168 188
IRQ169 188
IRQ170 188
IRQ171 189
IRQ172 189
IRQ173 189
IRQ174 190
IRQ175 190
IRQ176 190
IRQ177 191
IRQ178 191
IRQ179 191
IRQ180 192
IRQ181 192
IRQ182 192
IRQ183 193
IRQ184 193
IRQ185 193
IRQ186 194
IRQ187 194
IRQ188 194
IRQ189 195
IRQ190 195
IRQ191 195
5.15. qc_mclicip6 196
5.15.1. Attributes 196
5.15.2. Format 196
5.15.3. Field Summary 196
5.15.4. Fields 197
IRQ192 197
IRQ193 197
IRQ194 198
IRQ195 198
IRQ196 198

IRQ197 199
IRQ198 199
IRQ199 199
IRQ200 200
IRQ201 200
IRQ202 200
IRQ203 201
IRQ204 201
IRQ205 201
IRQ206 202
IRQ207 202
IRQ208 202
IRQ209 203
IRQ210 203
IRQ211 203
IRQ212 204
IRQ213 204
IRQ214 204
IRQ215 205
IRQ216 205
IRQ217 205
IRQ218 206
IRQ219 206
IRQ220 206
IRQ221 207
IRQ222 207
IRQ223 207
5.16. qc_mclicip7 208
5.16.1. Attributes 208
5.16.2. Format 208
5.16.3. Field Summary 208
5.16.4. Fields 209
IRQ224 209
IRQ225 209
IRQ226 210
IRQ227 210
IRQ228 210
IRQ229 211
IRQ230 211
IRQ231 211
IRQ232 212
IRQ233 212
IRQ234 212

IRQ235 213
IRQ236 213
IRQ237 213
IRQ238 214
IRQ239 214
IRQ240 214
IRQ241 215
IRQ242 215
IRQ243 215
IRQ244 216
IRQ245 216
IRQ246 216
IRQ247 217
IRQ248 217
IRQ249 217
IRQ250 218
IRQ251 218
IRQ252 218
IRQ253 219
IRQ254 219
IRQ255 219
5.17. qc_mncause 220
5.17.1. Attributes 220
5.17.2. Format 220
5.17.3. Field Summary 220
5.17.4. Fields 221
INT 221
NMI 221
MPP 221
MPIE 221
MIE 222
EXCP 222
RESP 223
MPIL 223
MIL 223
NMICODE 224
6. Instructions (in alphabetical order) 225
6.1. qc.addsat 225
6.2. qc.addusat 227
6.3. qc.beqi 228
6.4. qc.bgei 229
6.5. qc.bgeui 230
6.6. qc.blti 231

6.7. qc.bltui 232
6.8. qc.bnei 233
6.9. qc.brev32 234
6.10. qc.c.bexti 235
6.11. qc.c.bseti 236
6.12. qc.c.clrint 237
6.13. qc.c.delay 238
6.14. qc.c.di 239
6.15. qc.c.dir 240
6.16. qc.c.ei 241
6.17. qc.c.eir 242
6.18. qc.c.extu 243
6.19. qc.c.mienter.nest 244
6.20. qc.c.mienter 246
6.21. qc.c.mileaveret 248
6.22. qc.c.mnret 250
6.23. qc.c.mret 251
6.24. qc.c.muliadd 252
6.25. qc.c.mveqz 253
6.26. qc.c.ptrace 254
6.27. qc.c.setint 255
6.28. qc.c.sync 256
6.29. qc.c.syncr 257
6.30. qc.c.syncwf 258
6.31. qc.c.syncwl 259
6.32. qc.clo 260
6.33. qc.clrinti 261
6.34. qc.compress2 262
6.35. qc.compress3 263
6.36. qc.csrrwr 264
6.37. qc.csrrwri 265
6.38. qc.cto 266
6.39. qc.e.addai 267
6.40. qc.e.addi 268
6.41. qc.e.andai 269
6.42. qc.e.andi 270
6.43. qc.e.beqi 271
6.44. qc.e.bgei 272
6.45. qc.e.bgeui 273
6.46. qc.e.blti 274
6.47. qc.e.bltui 275
6.48. qc.e.bnei 276
6.49. qc.e.j 277

6.50. qc.e.jal 278
6.51. qc.e.lb 279
6.52. qc.e.lbu 280
6.53. qc.e.lh 281
6.54. qc.e.lhu 282
6.55. qc.e.li 283
6.56. qc.e.lw 284
6.57. qc.e.orai 285
6.58. qc.e.ori 286
6.59. qc.e.sb 287
6.60. qc.e.sh 288
6.61. qc.e.sw 289
6.62. qc.e.xorai 290
6.63. qc.e.xori 291
6.64. qc.expand2 292
6.65. qc.expand3 293
6.66. qc.ext 294
6.67. qc.extd 295
6.68. qc.extdpr 296
6.69. qc.extdprh 297
6.70. qc.extdr 298
6.71. qc.extdu 299
6.72. qc.extdupr 300
6.73. qc.extduprh 301
6.74. qc.extdur 302
6.75. qc.extu 303
6.76. qc.insb 304
6.77. qc.insbh 305
6.78. qc.insbhr 306
6.79. qc.insbi 307
6.80. qc.insbpr 308
6.81. qc.insbprh 309
6.82. qc.insbr 310
6.83. qc.insbri 311
6.84. qc.inw 312
6.85. qc.li 313
6.86. qc.lieq 314
6.87. qc.lieqi 315
6.88. qc.lige 316
6.89. qc.ligei 317
6.90. qc.ligeu 318
6.91. qc.ligeui 319
6.92. qc.lilt 320

6.93. qc.lilti 321
6.94. qc.liltu 322
6.95. qc.liltui 323
6.96. qc.line 324
6.97. qc.linei 325
6.98. qc.lrb 326
6.99. qc.lrbu 327
6.100. qc.lrh 328
6.101. qc.lrhu 329
6.102. qc.lrw 330
6.103. qc.lwm 331
6.104. qc.lwmi 332
6.105. qc.muliadd 333
6.106. qc.mveq 334
6.107. qc.mveqi 335
6.108. qc.mvge 336
6.109. qc.mvgei 337
6.110. qc.mvgeu 338
6.111. qc.mvgeui 339
6.112. qc.mvlt 340
6.113. qc.mvlti 341
6.114. qc.mvltu 342
6.115. qc.mvltui 343
6.116. qc.mvne 344
6.117. qc.mvnei 345
6.118. qc.norm 346
6.119. qc.normeu 347
6.120. qc.normu 348
6.121. qc.outw 349
6.122. qc.pcoredump 350
6.123. qc.pexit 351
6.124. qc.ppreg 352
6.125. qc.ppregs 353
6.126. qc.pputc 354
6.127. qc.pputci 355
6.128. qc.pputs 356
6.129. qc.psyscall 357
6.130. qc.psyscalli 358
6.131. qc.selecteqi 359
6.132. qc.selectieq 360
6.133. qc.selectieqi 361
6.134. qc.selectiieq 362
6.135. qc.selectiine 363

6.136. qc.selectine 364
6.137. qc.selectinei 365
6.138. qc.selectnei 366
6.139. qc.setinti 367
6.140. qc.setwm 368
6.141. qc.setwmi 369
6.142. qc.shladd 370
6.143. qc.shlsat 371
6.144. qc.shlusat 372
6.145. qc.srb 373
6.146. qc.srh 374
6.147. qc.srw 375
6.148. qc.subsat 376
6.149. qc.subusat 377
6.150. qc.swm 378
6.151. qc.swmi 379
6.152. qc.sync 380
6.153. qc.syncr 381
6.154. qc.syncwf 382
6.155. qc.syncwl 383
6.156. qc.wrap 384
6.157. qc.wrapi 385
7. IDL Functions 386
7.1. abort_current_instruction (builtin) 386
7.2. access_check 386
7.3. assert (builtin) 386
7.4. atomic_check_then_write_32 (builtin) 386
7.5. atomic_check_then_write_64 (builtin) 387
7.6. cached_translation (builtin) 387
7.7. current_translation_mode 387
7.8. delay (builtin) 390
7.9. effective_ldst_mode 390
7.10. exception_handling_mode 391
7.11. fence_tso (builtin) 391
7.12. gstage_page_walk 392
7.13. highest_set_bit 395
7.14. implemented? (builtin) 395
7.15. is_naturally_aligned 395
7.16. iss_syscall (builtin) 395
7.17. jump_halfword 395
7.18. lowest_set_bit 396
7.19. maybe_cache_translation (builtin) 396
7.20. misaligned_is_atomic? 396

7.21. mode 397
7.22. mtval_for 397
7.23. notify_mode_change (builtin) 398
7.24. raise 398
7.25. raise_guest_page_fault 399
7.26. raise_precise 400
7.27. read_device (builtin) 401
7.28. read_memory 402
7.29. read_memory_aligned 403
7.30. read_physical_memory (builtin) 403
7.31. set_mode 403
7.32. sext 403
7.33. stage1_page_walk 404
7.34. stval_for 408
7.35. sync_read_after_write_device (builtin) 409
7.36. sync_write_after_read_device (builtin) 409
7.37. tinst_transform 409
7.38. tinst_value_for_guest_page_fault 410
7.39. translate 411
7.40. translate_gstage 412
7.41. unpredictable (builtin) 413
7.42. vstval_for 413
7.43. write_device (builtin) 414
7.44. write_memory 414
7.45. write_memory_aligned 415
7.46. write_physical_memory (builtin) 416
7.47. xlen 416

Licensing and Acknowledgements

This document is in the Frozen state.

Change is extremely unlikely. A high threshold will be used, and a change will only occur because of some truly critical issue being identified during the public review cycle. Any other desired or needed changes can be the subject of a follow-on new extension.

Copyright and license information

Acknowledgements

Contributors to version 0.6.0 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.

Versions

This specification documents version 0.6.0 of Xqci.

Version History

Version

0.1.0

State

development

Version

0.2.0

State

development

Changes

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

Version

0.3.0

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 (0.1.0)
- Xqciac (0.1.0)
- Xqcibi (0.1.0)
- Xqcibm (0.1.0)
- Xqcicli (0.1.0)
- Xqcicm (0.1.0)
- Xqcics (0.1.0)
- Xqcicsr (0.1.0)
- Xqciint (0.1.0)
- Xqcilb (0.1.0)
- Xqcili (0.1.0)

- Xqcilia (0.1.0)
- Xqcilo (0.1.0)
- Xqcilsm (0.1.0)
- Xqcisls (0.1.0)

Version

0.4.0

State

frozen

Changes

- Add information about instruction formats of each instruction
- Fix description and functionality of qc.c.extu instruction
- Fix description and functionality of qc.shladd instruction

Implies

- Xqcia (0.2.0)
- Xqciac (0.2.0)
- Xqcibi (0.2.0)
- Xqcibm (0.2.0)
- Xqcicli (0.2.0)
- Xqcicm (0.2.0)
- Xqcics (0.2.0)
- Xqcicsr (0.2.0)
- Xqciint (0.2.0)
- Xqcilb (0.2.0)
- Xqcili (0.2.0)
- Xqcilia (0.2.0)
- Xqcilo (0.2.0)
- Xqcilsm (0.2.0)
- Xqcisls (0.2.0)

Requires

- Zca, version >= 1.0.0

Version

0.5.0

State

frozen

Changes

- Added Xqciio sub-extension

- Added Xqcisim sub-extension
- Added Xqcisync sub-extension
- Rename instruction of qc.muladdi to qc.muliadd
- Rename instruction of qc.c.muladdi to qc.c.muliadd
- Fix description of qc.shladd instruction
- Fix description and functionality of qc.c.extu instruction
- Fix description and functionality of qc.wrapi instruction
- Fix description of qc.swmi, qc.lwmi and qc.setwmi instructions
- Fix description of qc.mclici* CSRs to reflect being part of Xqciint custom extension
- Fix description of qc.setinti and qc.clrinti instructions

Implies

- Xqcia (0.3.0)
- Xqciac (0.3.0)
- Xqcibi (0.2.0)
- Xqcibm (0.3.0)
- Xqcicli (0.2.0)
- Xqcicm (0.2.0)
- Xqcics (0.2.0)
- Xqcicsr (0.2.0)
- Xqciint (0.3.0)
- Xqciio (0.1.0)
- Xqcilb (0.2.0)
- Xqcili (0.2.0)
- Xqcilia (0.2.0)
- Xqcilo (0.2.0)
- Xqcilsm (0.3.0)
- Xqcisim (0.1.0)
- Xqcisls (0.2.0)
- Xqcisync (0.1.0)

Requires

- Zca, version >= 1.0.0

Version

0.6.0

State

frozen

Changes

- Fix encoding of qc.c.extu instruction
- Fix encoding of qc.swmi instruction
- Fix decoding of qc.pputci instruction
- Fix decoding of qc.delay instruction (state that immediate cannot be 0)
- Rename qc.slasat → qc.shlsat
- Rename qc.sllsat → qc.shlusat
- Add requirement to include Zca extension for Xqcisim since it has 16-bit instructions
- Add requirement to include Zca extension for Xqcisync since it has 16-bit instructions
- Remove qc.flags CSR

Implies

- Xqcia (0.4.0)
- Xqciac (0.2.0)
- Xqcibi (0.2.0)
- Xqcibm (0.4.0)
- Xqcicli (0.2.0)
- Xqcicm (0.2.0)
- Xqcics (0.2.0)
- Xqcicsr (0.3.0)
- Xqciint (0.2.0)
- Xqcilb (0.2.0)
- Xqcili (0.2.0)
- Xqcilia (0.2.0)
- Xqcilo (0.2.0)
- Xqcilsm (0.4.0)
- Xqcisim (0.2.0)
- Xqcisls (0.2.0)
- Xqcisync (0.2.0)

Requires

- Zca, version >= 1.0.0

Chapter 1. Conventions

Version requirements are specified as conditions using the following operators:

Operator	Meaning
`~> VERSION`	Accepts any version that is *compatible* with `VERSION`. By default, a version A is compatible with a version B if A's version number is greater than or equal to version B. If that default assumption is ever broken, it will be noted in the list of extension versions.
`= VERSION`	Accepts only version `VERSION`.
`>= VERSION`	Accepts any version greater than or equal to `VERSION`.
`<= VERSION`	Accepts any version less than or equal to `VERSION`.
`> VERSION`	Accepts any version greater than `VERSION`.
`< VERSION`	Accepts any version less than `VERSION`.

Chapter 2. Extension description

The Xqci extension includes a set of instructions that improve RISC-V code density and performance in microontrollers. 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.

Extension-specific instruction formats

QC.EAI format used for 48-bit instructions that operate on 32-bit immediate argument.

Field	Start bit	Width	Description
opcode	0	7	Opcode field of 48-bit instructions is 0x1F
rd	7	5	Destination register
func3	12	3	Function field identifying instruction group
f1	15	1	Secondary function field
imm[31:0]	16	32	Immediate operand of 32 bits

QC.EI format used for 48-bit instructions that operate on 26-bit immediate argument, including loads.

Field	Start bit	Width	Description
opcode	0	7	Opcode field of 48-bit instructions is 0x1F
rd	7	5	Destination register
func3	12	3	Function field identifying instruction group
rs1	15	5	Register argument
imm[9:0]	20	10	Immediate operand of 26 bits, the 10 LSBs
func2	30	2	Secondary function field
imm[25:10]	32	16	Immediate operand of 26 bits, the 16 MSBs

QC.EB format used for 48-bit branch instructions that compare register with 16-bit immediate.

Field	Start bit	Width	Description
opcode	0	7	Opcode field of 48-bit instructions is 0x1F
imm[4:1,11]	7	5	Bits of immediate value of branch target
func3	12	3	Function field identifying instruction group
rs1	15	5	Register argument
func5	20	5	Secondary function field
imm[12,10:5]	25	7	Bits of immediate value of branch target
simm[15:0]	32	16	Immediate operand of 16 bits to compare with register

QC.EJ format used for 48-bit jump/call instructions with 32-bit immediate target address.

Field	Start bit	Width	Description
opcode	0	7	Opcode field of 48-bit instructions is 0x1F
imm[4:1,11]	7	5	Bits of immediate value of branch target
func3	12	3	Function field identifying instruction group
func2	15	2	Secondary function field
imm[15:13]	17	3	Bits of immediate value of branch target
func5	20	5	Secondary function field
imm[12,10:5]	25	7	Bits of immediate value of branch target

Field	Start bit	Width	Description
imm[31:16]	32	16	The 16 MSBs of immediate value of branch target

QC.ES format used for 48-bit store instructions with 26-bit immediate offset.

Field	Start bit	Width	Description
opcode	0	7	Opcode field of 48-bit instructions is 0x1F
imm[4:0]	7	5	Immediate operand of 26 bits offset, the 5 LSBs
func3	12	3	Function field identifying instruction group
rs1	15	5	Register argument used as base address
rs2	20	5	Register argument to be saved
imm[9:5]	25	5	Immediate operand of 26 bits offset, the 5 bits
func2	30	2	Secondary function field
imm[25:10]	32	16	Immediate operand of 26 bits offset, the 16 MSBs

2.1. Sub-extensions

Xqci defines the following #{implications.size} sub-extensions:

2.1.1. Xqcia (0.4.0)

The Xqcia extension includes eleven instructions to perform integer arithmetic.

2.1.2. Xqciac (0.2.0)

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

Xqciac requires:

- Zca, version >= 1.0.0

2.1.3. Xqcibi (0.2.0)

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.).

Xqcibi requires:

- Zca, version >= 1.0.0

2.1.4. Xqcibm (0.4.0)

The Xqcibm extension includes thirty eight instructions that perform bit manipulation, include

insertion and extraction.

Xqcibm requires:

- Zca, version >= 1.0.0

2.1.5. Xqcicli (0.2.0)

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

2.1.6. Xqcicm (0.2.0)

The Xqcicm extension includes thirteen conditional move instructions.

Xqcicm requires:

- Zca, version >= 1.0.0

2.1.7. Xqcics (0.2.0)

The Xqcics extension includes eight conditional select instructions.

2.1.8. Xqcicsr (0.3.0)

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

2.1.9. Xqciint (0.2.0)

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

Xqciint requires:

- Zca, version >= 1.0.0

2.1.10. Xqcilb (0.2.0)

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

Xqcilb requires:

- Zca, version >= 1.0.0

2.1.11. Xqcili (0.2.0)

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

2.1.12. Xqcilia (0.2.0)

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

Xqcilia requires:

- Zca, version >= 1.0.0

2.1.13. Xqcilo (0.2.0)

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.

Xqcilo requires:

- Zca, version >= 1.0.0

2.1.14. Xqcilsm (0.4.0)

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

2.1.15. Xqcisim (0.2.0)

The Xqcisim extension includes ten hint instructions to interface simulation environment. On real target any instruction from this extension executed as "no-operation" and have no effect.

Xqcisim requires:

- Zca, version >= 1.0.0

2.1.16. Xqcisls (0.2.0)

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

2.1.17. Xqcisync (0.2.0)

The Xqcisync extension includes nine instructions, eight for non-memory-mapped devices synchronization and delay instruction. Synchronization instructions are kind of IO fences that work with special devices synchronization signals.

Xqcisync requires:

- Zca, version >= 1.0.0

Chapter 3. Instruction summary

The following 157 instructions are added by this extension:

RV32	RV64	Mnemonic	Instruction
✓		`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.delay `	Delay execution for immediate amount of cycles
✓		`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.mnret `	Machine NMI Return
✓		`qc.c.mret `	Machine Exception Return
✓		`qc.c.muliadd rd, rs1, uimm`	Multiply and accumulate (Immediate)
✓		`qc.c.mveqz rd, rs1`	Conditional Move if equal to zero
✓		`qc.c.ptrace `	Tracing pseudo-instruction (hint) working only in simulation environment
✓		`qc.c.setint rs1`	Set interrupt (Register)

✓		qc.c.sync slist	Non-memory-mapped device read-after-write synchronization to completion
✓		qc.c.syncr slist	Non-memory-mapped device read-after-write synchronization
✓		qc.c.syncwf slist	Non-memory-mapped device write-after-read synchronization
✓		qc.c.syncwl slist	Non-memory-mapped device write-after-read synchronization to completion
✓		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.blti rs1, imm, offset	Branch on less than (immediate)
✓		qc.e.bltui 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
✓		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)
✓		`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.inw rd, imm(rs1)`	Input word from non-memory-mapped device
✓		`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
✓		`qc.lrhu rd, rs1, rs2, shamt`	Load indexed unsigned halfword
✓		`qc.lrw 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.muliadd 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
✓		`qc.outw rs2, imm(rs1)`	Output word to non-memory-mapped device
✓		`qc.pcoredump `	Print core dump pseudo-instruction (hint) working only in simulation environment
✓		`qc.pexit rs1`	Exit call with register argument pseudo-instruction (hint) working only in simulation environment
✓		`qc.ppreg rs1`	Print register pseudo-instruction (hint) working only in simulation environment
✓		`qc.ppregs `	Print all registers pseudo-instruction (hint) working only in simulation environment
✓		`qc.pputc rs1`	Print character passed in register argument pseudo-instruction (hint) working only in simulation environment
✓		`qc.pputci imm`	Print character passed in the immediate argument pseudo-instruction (hint) working only in simulation environment
✓		`qc.pputs rs1`	Print string pseudo-instruction (hint) working only in simulation environment
✓		`qc.psyscall rs1`	System call with register argument pseudo-instruction (hint) working only in simulation environment
✓		`qc.psyscalli imm`	System call with immediate argument pseudo-instruction (hint) working only in simulation environment
✓		`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 rd, rs1, rs2, shamt`	Shift left and add (immediate)
✓		`qc.shlsat rd, rs1, rs2`	Saturating signed left shift
✓		`qc.shlusat rd, rs1, rs2`	Saturating unsigned 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
✓		`qc.swmi rs3, length, imm(rs1)`	Store word multiple (immediate)
✓		`qc.sync imm`	Non-memory-mapped device read-after-write synchronization to completion
✓		`qc.syncr imm`	Non-memory-mapped device read-after-write synchronization
✓		`qc.syncwf imm`	Non-memory-mapped device write-after-read synchronization
✓		`qc.syncwl imm`	Non-memory-mapped device write-after-read synchronization to completion
✓		`qc.wrap rd, rs1, rs2`	Wraparound (Register)
✓		`qc.wrapi rd, rs1, imm`	Wraparound (Unsigned Immediate)

3.1. Instructions by sub-extension

Mnemonic	Xqcia	Xqciac	Xqcibi	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqciint	Xqcilb	Xqcili	Xqcilia	Xqcilo	Xqcilsm	Xqcisim	Xqcisls	Xqcisync
qc.addsat	✓																
qc.addusat	✓																
qc.beqi			✓														
qc.bgei			✓														
qc.bgeui			✓														

Mnemonic	Xqcia	Xqciac	Xqcibi	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqciint	Xqcilb	Xqcili	Xqcilia	Xqcilo	Xqcilsm	Xqcisim	Xqcisls	Xqcisync
qc.blti			✓														
qc.bltui			✓														
qc.bnei			✓														
qc.brev32				✓													
qc.c.bexti				✓													
qc.c.bseti				✓													
qc.c.clrint									✓								
qc.c.delay																	✓
qc.c.di									✓								
qc.c.dir									✓								
qc.c.ei									✓								
qc.c.eir									✓								
qc.c.extu				✓													
qc.c.mienter.nest									✓								
qc.c.mienter									✓								
qc.c.mileaveret									✓								
qc.c.mnret									✓								
qc.c.mret									✓								
qc.c.muliadd		✓															
qc.c.mveqz		✓															
qc.c.ptrace															✓		
qc.c.setint									✓								
qc.c.sync																	✓
qc.c.syncr																	✓
qc.c.syncwf																	✓

Mnemonic	Xqcia	Xqciac	Xqcibi	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqciint	Xqcilb	Xqcili	Xqcilia	Xqcilo	Xqcilsm	Xqcisim	Xqcisls	Xqcisync
qc.c.syncwl																	✓
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										✓							
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													✓				

Mnemonic	Xqcia	Xqciac	Xqcibi	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqciint	Xqcilb	Xqcili	Xqcilia	Xqcilo	Xqcilsm	Xqcisim	Xqcisls	Xqcisync
qc.e.sw													✓				
qc.e.xorai												✓					
qc.e.xori												✓					
qc.expand2				✓													
qc.expand3				✓													
qc.ext				✓													
qc.extd				✓													
qc.extdpr				✓													
qc.extdprh				✓													
qc.extdr				✓													
qc.extdu				✓													
qc.extdupr				✓													
qc.extduprh				✓													
qc.extdur				✓													
qc.extu				✓													
qc.insb				✓													
qc.insbh				✓													
qc.insbhr				✓													
qc.insbi				✓													
qc.insbpr				✓													
qc.insbprh				✓													
qc.insbr				✓													
qc.insbri				✓													
qc.inw																	
qc.li											✓						
qc.lieq					✓												
qc.lieqi					✓												
qc.lige					✓												
qc.ligei					✓												

Mnemonic	Xqcia	Xqciac	Xqcibi	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqciint	Xqcilb	Xqcili	Xqcilia	Xqcilo	Xqcilsm	Xqcisim	Xqcisls	Xqcisync
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.muliadd		✓															
qc.mveq						✓											
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	✓																

Mnemonic	Xqcia	Xqciac	Xqcibi	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqciint	Xqcilb	Xqcili	Xqcilia	Xqcilo	Xqcilsm	Xqcisim	Xqcisls	Xqcisync
qc.outw																	
qc.pcoredump															✓		
qc.pexit															✓		
qc.ppreg															✓		
qc.ppregs															✓		
qc.pputc															✓		
qc.pputci															✓		
qc.pputs															✓		
qc.psyscall															✓		
qc.psyscalli															✓		
qc.selecteqi							✓										
qc.selectieq							✓										
qc.selectieqi							✓										
qc.selectiieq							✓										
qc.selectiine							✓										
qc.selectine							✓										
qc.selectinei							✓										
qc.selectnei							✓										
qc.setinti									✓								
qc.setwm														✓			
qc.setwmi														✓			
qc.shladd		✓															
qc.shlsat	✓																
qc.shlusat	✓																

Mnemonic	Xqcia	Xqciac	Xqcibi	Xqcibm	Xqcicli	Xqcicm	Xqcics	Xqcicsr	Xqciint	Xqcilb	Xqcili	Xqcilia	Xqcilo	Xqcilsm	Xqcisim	Xqcisls	Xqcisync
qc.srb																✓	
qc.srh																✓	
qc.srw																✓	
qc.subsat	✓																
qc.subusat	✓																
qc.swm														✓			
qc.swmi														✓			
qc.sync																	✓
qc.syncr																	✓
qc.syncwf																	✓
qc.syncwl																	✓
qc.wrap	✓																
qc.wrapi	✓																

Chapter 4. CSR summary

The following 17 are added by this extension.

RV32	RV64	CSR	Name
✓		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
✓		qc_mncause	Machine NMI Cause

Chapter 5. Csrs (in alphabetical order)

5.1. qc_mclicie0

IRQ Enable 0

Enable bits for IRQs 0-31

5.1.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f0
Length	32-bit
Privilege Mode	M

5.1.2. Format

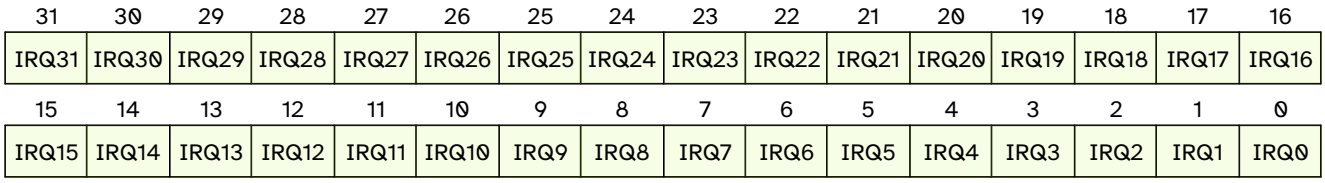


Figure 1. qc_mclicie0 format

5.1.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie0.IRQ0	0	RW	0
qc_mclicie0.IRQ1	1	RW	0
qc_mclicie0.IRQ2	2	RW	0
qc_mclicie0.IRQ3	3	RW	0
qc_mclicie0.IRQ4	4	RW	0
qc_mclicie0.IRQ5	5	RW	0
qc_mclicie0.IRQ6	6	RW	0
qc_mclicie0.IRQ7	7	RW	0
qc_mclicie0.IRQ8	8	RW	0
qc_mclicie0.IRQ9	9	RW	0
qc_mclicie0.IRQ10	10	RW	0
qc_mclicie0.IRQ11	11	RW	0
qc_mclicie0.IRQ12	12	RW	0
qc_mclicie0.IRQ13	13	RW	0
qc_mclicie0.IRQ14	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie0.IRQ15	15	RW	0
qc_mclicie0.IRQ16	16	RW	0
qc_mclicie0.IRQ17	17	RW	0
qc_mclicie0.IRQ18	18	RW	0
qc_mclicie0.IRQ19	19	RW	0
qc_mclicie0.IRQ20	20	RW	0
qc_mclicie0.IRQ21	21	RW	0
qc_mclicie0.IRQ22	22	RW	0
qc_mclicie0.IRQ23	23	RW	0
qc_mclicie0.IRQ24	24	RW	0
qc_mclicie0.IRQ25	25	RW	0
qc_mclicie0.IRQ26	26	RW	0
qc_mclicie0.IRQ27	27	RW	0
qc_mclicie0.IRQ28	28	RW	0
qc_mclicie0.IRQ29	29	RW	0
qc_mclicie0.IRQ30	30	RW	0
qc_mclicie0.IRQ31	31	RW	0

5.1.4. 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

5.2. qc_mclicie1

IRQ Enable 1

Enable bits for IRQs 32-63

5.2.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f1
Length	32-bit
Privilege Mode	M

5.2.2. Format

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

Figure 2. qc_mclicie1 format

5.2.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie1.IRQ32	0	RW	0
qc_mclicie1.IRQ33	1	RW	0
qc_mclicie1.IRQ34	2	RW	0
qc_mclicie1.IRQ35	3	RW	0
qc_mclicie1.IRQ36	4	RW	0
qc_mclicie1.IRQ37	5	RW	0
qc_mclicie1.IRQ38	6	RW	0
qc_mclicie1.IRQ39	7	RW	0
qc_mclicie1.IRQ40	8	RW	0
qc_mclicie1.IRQ41	9	RW	0
qc_mclicie1.IRQ42	10	RW	0
qc_mclicie1.IRQ43	11	RW	0
qc_mclicie1.IRQ44	12	RW	0
qc_mclicie1.IRQ45	13	RW	0
qc_mclicie1.IRQ46	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie1.IRQ47	15	RW	0
qc_mclicie1.IRQ48	16	RW	0
qc_mclicie1.IRQ49	17	RW	0
qc_mclicie1.IRQ50	18	RW	0
qc_mclicie1.IRQ51	19	RW	0
qc_mclicie1.IRQ52	20	RW	0
qc_mclicie1.IRQ53	21	RW	0
qc_mclicie1.IRQ54	22	RW	0
qc_mclicie1.IRQ55	23	RW	0
qc_mclicie1.IRQ56	24	RW	0
qc_mclicie1.IRQ57	25	RW	0
qc_mclicie1.IRQ58	26	RW	0
qc_mclicie1.IRQ59	27	RW	0
qc_mclicie1.IRQ60	28	RW	0
qc_mclicie1.IRQ61	29	RW	0
qc_mclicie1.IRQ62	30	RW	0
qc_mclicie1.IRQ63	31	RW	0

5.2.4. 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

5.3. qc_mclicie2

IRQ Enable 2

Enable bits for IRQs 64-95

5.3.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f2
Length	32-bit
Privilege Mode	M

5.3.2. 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_mclicie2 format

5.3.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie2.IRQ64	0	RW	0
qc_mclicie2.IRQ65	1	RW	0
qc_mclicie2.IRQ66	2	RW	0
qc_mclicie2.IRQ67	3	RW	0
qc_mclicie2.IRQ68	4	RW	0
qc_mclicie2.IRQ69	5	RW	0
qc_mclicie2.IRQ70	6	RW	0
qc_mclicie2.IRQ71	7	RW	0
qc_mclicie2.IRQ72	8	RW	0
qc_mclicie2.IRQ73	9	RW	0
qc_mclicie2.IRQ74	10	RW	0
qc_mclicie2.IRQ75	11	RW	0
qc_mclicie2.IRQ76	12	RW	0
qc_mclicie2.IRQ77	13	RW	0
qc_mclicie2.IRQ78	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie2.IRQ79	15	RW	0
qc_mclicie2.IRQ80	16	RW	0
qc_mclicie2.IRQ81	17	RW	0
qc_mclicie2.IRQ82	18	RW	0
qc_mclicie2.IRQ83	19	RW	0
qc_mclicie2.IRQ84	20	RW	0
qc_mclicie2.IRQ85	21	RW	0
qc_mclicie2.IRQ86	22	RW	0
qc_mclicie2.IRQ87	23	RW	0
qc_mclicie2.IRQ88	24	RW	0
qc_mclicie2.IRQ89	25	RW	0
qc_mclicie2.IRQ90	26	RW	0
qc_mclicie2.IRQ91	27	RW	0
qc_mclicie2.IRQ92	28	RW	0
qc_mclicie2.IRQ93	29	RW	0
qc_mclicie2.IRQ94	30	RW	0
qc_mclicie2.IRQ95	31	RW	0

5.3.4. 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

5.4. qc_mclicie3

IRQ Enable 3

Enable bits for IRQs 96-127

5.4.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f3
Length	32-bit
Privilege Mode	M

5.4.2. Format

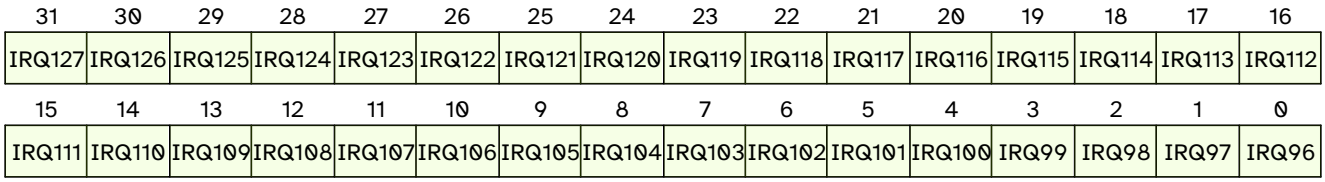


Figure 4. qc_mclicie3 format

5.4.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie3.IRQ96	0	RW	0
qc_mclicie3.IRQ97	1	RW	0
qc_mclicie3.IRQ98	2	RW	0
qc_mclicie3.IRQ99	3	RW	0
qc_mclicie3.IRQ100	4	RW	0
qc_mclicie3.IRQ101	5	RW	0
qc_mclicie3.IRQ102	6	RW	0
qc_mclicie3.IRQ103	7	RW	0
qc_mclicie3.IRQ104	8	RW	0
qc_mclicie3.IRQ105	9	RW	0
qc_mclicie3.IRQ106	10	RW	0
qc_mclicie3.IRQ107	11	RW	0
qc_mclicie3.IRQ108	12	RW	0
qc_mclicie3.IRQ109	13	RW	0
qc_mclicie3.IRQ110	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie3.IRQ111	15	RW	0
qc_mclicie3.IRQ112	16	RW	0
qc_mclicie3.IRQ113	17	RW	0
qc_mclicie3.IRQ114	18	RW	0
qc_mclicie3.IRQ115	19	RW	0
qc_mclicie3.IRQ116	20	RW	0
qc_mclicie3.IRQ117	21	RW	0
qc_mclicie3.IRQ118	22	RW	0
qc_mclicie3.IRQ119	23	RW	0
qc_mclicie3.IRQ120	24	RW	0
qc_mclicie3.IRQ121	25	RW	0
qc_mclicie3.IRQ122	26	RW	0
qc_mclicie3.IRQ123	27	RW	0
qc_mclicie3.IRQ124	28	RW	0
qc_mclicie3.IRQ125	29	RW	0
qc_mclicie3.IRQ126	30	RW	0
qc_mclicie3.IRQ127	31	RW	0

5.4.4. 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

5.5. qc_mclicie4

IRQ Enable 4

Enable bits for IRQs 128-159

5.5.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f4
Length	32-bit
Privilege Mode	M

5.5.2. Format

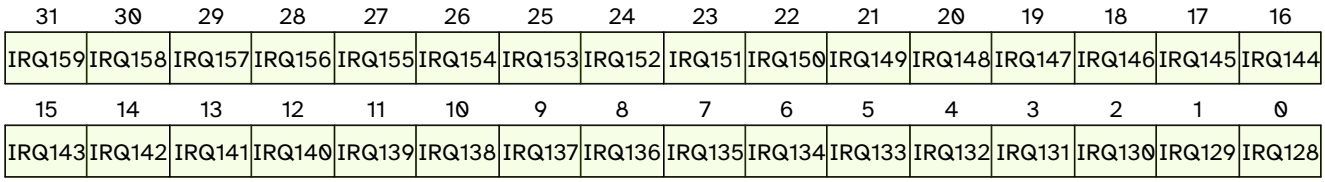


Figure 5. qc_mclicie4 format

5.5.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie4.IRQ128	0	RW	0
qc_mclicie4.IRQ129	1	RW	0
qc_mclicie4.IRQ130	2	RW	0
qc_mclicie4.IRQ131	3	RW	0
qc_mclicie4.IRQ132	4	RW	0
qc_mclicie4.IRQ133	5	RW	0
qc_mclicie4.IRQ134	6	RW	0
qc_mclicie4.IRQ135	7	RW	0
qc_mclicie4.IRQ136	8	RW	0
qc_mclicie4.IRQ137	9	RW	0
qc_mclicie4.IRQ138	10	RW	0
qc_mclicie4.IRQ139	11	RW	0
qc_mclicie4.IRQ140	12	RW	0
qc_mclicie4.IRQ141	13	RW	0
qc_mclicie4.IRQ142	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie4.IRQ143	15	RW	0
qc_mclicie4.IRQ144	16	RW	0
qc_mclicie4.IRQ145	17	RW	0
qc_mclicie4.IRQ146	18	RW	0
qc_mclicie4.IRQ147	19	RW	0
qc_mclicie4.IRQ148	20	RW	0
qc_mclicie4.IRQ149	21	RW	0
qc_mclicie4.IRQ150	22	RW	0
qc_mclicie4.IRQ151	23	RW	0
qc_mclicie4.IRQ152	24	RW	0
qc_mclicie4.IRQ153	25	RW	0
qc_mclicie4.IRQ154	26	RW	0
qc_mclicie4.IRQ155	27	RW	0
qc_mclicie4.IRQ156	28	RW	0
qc_mclicie4.IRQ157	29	RW	0
qc_mclicie4.IRQ158	30	RW	0
qc_mclicie4.IRQ159	31	RW	0

5.5.4. 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

5.6. qc_mclicie5

IRQ Enable 5

Enable bits for IRQs 160-191

5.6.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f5
Length	32-bit
Privilege Mode	M

5.6.2. Format

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

Figure 6. qc_mclicie5 format

5.6.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie5.IRQ160	0	RW	0
qc_mclicie5.IRQ161	1	RW	0
qc_mclicie5.IRQ162	2	RW	0
qc_mclicie5.IRQ163	3	RW	0
qc_mclicie5.IRQ164	4	RW	0
qc_mclicie5.IRQ165	5	RW	0
qc_mclicie5.IRQ166	6	RW	0
qc_mclicie5.IRQ167	7	RW	0
qc_mclicie5.IRQ168	8	RW	0
qc_mclicie5.IRQ169	9	RW	0
qc_mclicie5.IRQ170	10	RW	0
qc_mclicie5.IRQ171	11	RW	0
qc_mclicie5.IRQ172	12	RW	0
qc_mclicie5.IRQ173	13	RW	0
qc_mclicie5.IRQ174	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie5.IRQ175	15	RW	0
qc_mclicie5.IRQ176	16	RW	0
qc_mclicie5.IRQ177	17	RW	0
qc_mclicie5.IRQ178	18	RW	0
qc_mclicie5.IRQ179	19	RW	0
qc_mclicie5.IRQ180	20	RW	0
qc_mclicie5.IRQ181	21	RW	0
qc_mclicie5.IRQ182	22	RW	0
qc_mclicie5.IRQ183	23	RW	0
qc_mclicie5.IRQ184	24	RW	0
qc_mclicie5.IRQ185	25	RW	0
qc_mclicie5.IRQ186	26	RW	0
qc_mclicie5.IRQ187	27	RW	0
qc_mclicie5.IRQ188	28	RW	0
qc_mclicie5.IRQ189	29	RW	0
qc_mclicie5.IRQ190	30	RW	0
qc_mclicie5.IRQ191	31	RW	0

5.6.4. 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

5.7. qc_mclicie6

IRQ Enable 6

Enable bits for IRQs 192-223

5.7.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f6
Length	32-bit
Privilege Mode	M

5.7.2. Format

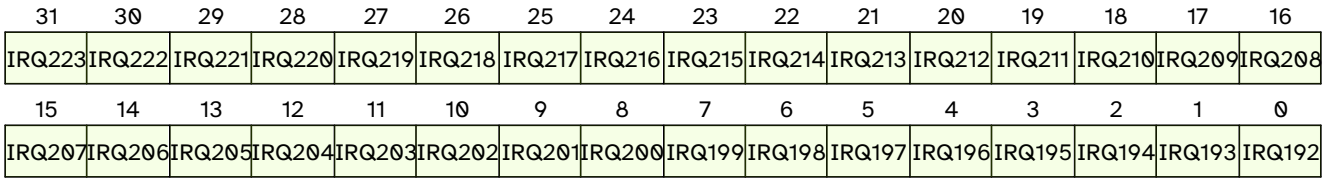


Figure 7. qc_mclicie6 format

5.7.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie6.IRQ192	0	RW	0
qc_mclicie6.IRQ193	1	RW	0
qc_mclicie6.IRQ194	2	RW	0
qc_mclicie6.IRQ195	3	RW	0
qc_mclicie6.IRQ196	4	RW	0
qc_mclicie6.IRQ197	5	RW	0
qc_mclicie6.IRQ198	6	RW	0
qc_mclicie6.IRQ199	7	RW	0
qc_mclicie6.IRQ200	8	RW	0
qc_mclicie6.IRQ201	9	RW	0
qc_mclicie6.IRQ202	10	RW	0
qc_mclicie6.IRQ203	11	RW	0
qc_mclicie6.IRQ204	12	RW	0
qc_mclicie6.IRQ205	13	RW	0
qc_mclicie6.IRQ206	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie6.IRQ207	15	RW	0
qc_mclicie6.IRQ208	16	RW	0
qc_mclicie6.IRQ209	17	RW	0
qc_mclicie6.IRQ210	18	RW	0
qc_mclicie6.IRQ211	19	RW	0
qc_mclicie6.IRQ212	20	RW	0
qc_mclicie6.IRQ213	21	RW	0
qc_mclicie6.IRQ214	22	RW	0
qc_mclicie6.IRQ215	23	RW	0
qc_mclicie6.IRQ216	24	RW	0
qc_mclicie6.IRQ217	25	RW	0
qc_mclicie6.IRQ218	26	RW	0
qc_mclicie6.IRQ219	27	RW	0
qc_mclicie6.IRQ220	28	RW	0
qc_mclicie6.IRQ221	29	RW	0
qc_mclicie6.IRQ222	30	RW	0
qc_mclicie6.IRQ223	31	RW	0

5.7.4. 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

5.8. qc_mclicie7

IRQ Enable 7

Enable bits for IRQs 224-255

5.8.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f7
Length	32-bit
Privilege Mode	M

5.8.2. Format

Figure 8. qc_mclicie7 format

5.8.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicie7.IRQ224	0	RW	0
qc_mclicie7.IRQ225	1	RW	0
qc_mclicie7.IRQ226	2	RW	0
qc_mclicie7.IRQ227	3	RW	0
qc_mclicie7.IRQ228	4	RW	0
qc_mclicie7.IRQ229	5	RW	0
qc_mclicie7.IRQ230	6	RW	0
qc_mclicie7.IRQ231	7	RW	0
qc_mclicie7.IRQ232	8	RW	0
qc_mclicie7.IRQ233	9	RW	0
qc_mclicie7.IRQ234	10	RW	0
qc_mclicie7.IRQ235	11	RW	0
qc_mclicie7.IRQ236	12	RW	0
qc_mclicie7.IRQ237	13	RW	0
qc_mclicie7.IRQ238	14	RW	0

Name	Location	Type	Reset Value
qc_mclicie7.IRQ239	15	RW	0
qc_mclicie7.IRQ240	16	RW	0
qc_mclicie7.IRQ241	17	RW	0
qc_mclicie7.IRQ242	18	RW	0
qc_mclicie7.IRQ243	19	RW	0
qc_mclicie7.IRQ244	20	RW	0
qc_mclicie7.IRQ245	21	RW	0
qc_mclicie7.IRQ246	22	RW	0
qc_mclicie7.IRQ247	23	RW	0
qc_mclicie7.IRQ248	24	RW	0
qc_mclicie7.IRQ249	25	RW	0
qc_mclicie7.IRQ250	26	RW	0
qc_mclicie7.IRQ251	27	RW	0
qc_mclicie7.IRQ252	28	RW	0
qc_mclicie7.IRQ253	29	RW	0
qc_mclicie7.IRQ254	30	RW	0
qc_mclicie7.IRQ255	31	RW	0

5.8.4. 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

IRQ255 enabled

Type

RW

Reset value

0

5.9. qc_mclicip0

IRQ Pending 0

Pending bits for IRQs 0-31

5.9.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f0
Length	32-bit
Privilege Mode	M

5.9.2. 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_mclicip0 format

5.9.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip0.IRQ0	0	RW	0
qc_mclicip0.IRQ1	1	RW	0
qc_mclicip0.IRQ2	2	RW	0
qc_mclicip0.IRQ3	3	RW	0
qc_mclicip0.IRQ4	4	RW	0
qc_mclicip0.IRQ5	5	RW	0
qc_mclicip0.IRQ6	6	RW	0
qc_mclicip0.IRQ7	7	RW	0
qc_mclicip0.IRQ8	8	RW	0
qc_mclicip0.IRQ9	9	RW	0
qc_mclicip0.IRQ10	10	RW	0
qc_mclicip0.IRQ11	11	RW	0
qc_mclicip0.IRQ12	12	RW	0
qc_mclicip0.IRQ13	13	RW	0
qc_mclicip0.IRQ14	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip0.IRQ15	15	RW	0
qc_mclicip0.IRQ16	16	RW	0
qc_mclicip0.IRQ17	17	RW	0
qc_mclicip0.IRQ18	18	RW	0
qc_mclicip0.IRQ19	19	RW	0
qc_mclicip0.IRQ20	20	RW	0
qc_mclicip0.IRQ21	21	RW	0
qc_mclicip0.IRQ22	22	RW	0
qc_mclicip0.IRQ23	23	RW	0
qc_mclicip0.IRQ24	24	RW	0
qc_mclicip0.IRQ25	25	RW	0
qc_mclicip0.IRQ26	26	RW	0
qc_mclicip0.IRQ27	27	RW	0
qc_mclicip0.IRQ28	28	RW	0
qc_mclicip0.IRQ29	29	RW	0
qc_mclicip0.IRQ30	30	RW	0
qc_mclicip0.IRQ31	31	RW	0

5.9.4. 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

5.10. qc_mclicip1

IRQ Pending 1

Pending bits for IRQs 32-63

5.10.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f1
Length	32-bit
Privilege Mode	M

5.10.2. Format

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

Figure 10. qc_mclicip1 format

5.10.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip1.IRQ32	0	RW	0
qc_mclicip1.IRQ33	1	RW	0
qc_mclicip1.IRQ34	2	RW	0
qc_mclicip1.IRQ35	3	RW	0
qc_mclicip1.IRQ36	4	RW	0
qc_mclicip1.IRQ37	5	RW	0
qc_mclicip1.IRQ38	6	RW	0
qc_mclicip1.IRQ39	7	RW	0
qc_mclicip1.IRQ40	8	RW	0
qc_mclicip1.IRQ41	9	RW	0
qc_mclicip1.IRQ42	10	RW	0
qc_mclicip1.IRQ43	11	RW	0
qc_mclicip1.IRQ44	12	RW	0
qc_mclicip1.IRQ45	13	RW	0
qc_mclicip1.IRQ46	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip1.IRQ47	15	RW	0
qc_mclicip1.IRQ48	16	RW	0
qc_mclicip1.IRQ49	17	RW	0
qc_mclicip1.IRQ50	18	RW	0
qc_mclicip1.IRQ51	19	RW	0
qc_mclicip1.IRQ52	20	RW	0
qc_mclicip1.IRQ53	21	RW	0
qc_mclicip1.IRQ54	22	RW	0
qc_mclicip1.IRQ55	23	RW	0
qc_mclicip1.IRQ56	24	RW	0
qc_mclicip1.IRQ57	25	RW	0
qc_mclicip1.IRQ58	26	RW	0
qc_mclicip1.IRQ59	27	RW	0
qc_mclicip1.IRQ60	28	RW	0
qc_mclicip1.IRQ61	29	RW	0
qc_mclicip1.IRQ62	30	RW	0
qc_mclicip1.IRQ63	31	RW	0

5.10.4. 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

5.11. qc_mclicip2

IRQ Pending 2

Pending bits for IRQs 64-95

5.11.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f2
Length	32-bit
Privilege Mode	M

5.11.2. 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_mclicip2 format

5.11.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip2.IRQ64	0	RW	0
qc_mclicip2.IRQ65	1	RW	0
qc_mclicip2.IRQ66	2	RW	0
qc_mclicip2.IRQ67	3	RW	0
qc_mclicip2.IRQ68	4	RW	0
qc_mclicip2.IRQ69	5	RW	0
qc_mclicip2.IRQ70	6	RW	0
qc_mclicip2.IRQ71	7	RW	0
qc_mclicip2.IRQ72	8	RW	0
qc_mclicip2.IRQ73	9	RW	0
qc_mclicip2.IRQ74	10	RW	0
qc_mclicip2.IRQ75	11	RW	0
qc_mclicip2.IRQ76	12	RW	0
qc_mclicip2.IRQ77	13	RW	0
qc_mclicip2.IRQ78	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip2.IRQ79	15	RW	0
qc_mclicip2.IRQ80	16	RW	0
qc_mclicip2.IRQ81	17	RW	0
qc_mclicip2.IRQ82	18	RW	0
qc_mclicip2.IRQ83	19	RW	0
qc_mclicip2.IRQ84	20	RW	0
qc_mclicip2.IRQ85	21	RW	0
qc_mclicip2.IRQ86	22	RW	0
qc_mclicip2.IRQ87	23	RW	0
qc_mclicip2.IRQ88	24	RW	0
qc_mclicip2.IRQ89	25	RW	0
qc_mclicip2.IRQ90	26	RW	0
qc_mclicip2.IRQ91	27	RW	0
qc_mclicip2.IRQ92	28	RW	0
qc_mclicip2.IRQ93	29	RW	0
qc_mclicip2.IRQ94	30	RW	0
qc_mclicip2.IRQ95	31	RW	0

5.11.4. 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

5.12. qc_mclicip3

IRQ Pending 3

Pending bits for IRQs 96-127

5.12.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f3
Length	32-bit
Privilege Mode	M

5.12.2. Format

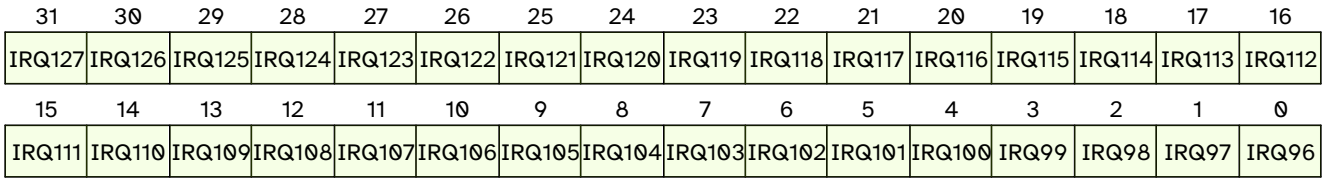


Figure 12. qc_mclicip3 format

5.12.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip3.IRQ96	0	RW	0
qc_mclicip3.IRQ97	1	RW	0
qc_mclicip3.IRQ98	2	RW	0
qc_mclicip3.IRQ99	3	RW	0
qc_mclicip3.IRQ100	4	RW	0
qc_mclicip3.IRQ101	5	RW	0
qc_mclicip3.IRQ102	6	RW	0
qc_mclicip3.IRQ103	7	RW	0
qc_mclicip3.IRQ104	8	RW	0
qc_mclicip3.IRQ105	9	RW	0
qc_mclicip3.IRQ106	10	RW	0
qc_mclicip3.IRQ107	11	RW	0
qc_mclicip3.IRQ108	12	RW	0
qc_mclicip3.IRQ109	13	RW	0
qc_mclicip3.IRQ110	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip3.IRQ111	15	RW	0
qc_mclicip3.IRQ112	16	RW	0
qc_mclicip3.IRQ113	17	RW	0
qc_mclicip3.IRQ114	18	RW	0
qc_mclicip3.IRQ115	19	RW	0
qc_mclicip3.IRQ116	20	RW	0
qc_mclicip3.IRQ117	21	RW	0
qc_mclicip3.IRQ118	22	RW	0
qc_mclicip3.IRQ119	23	RW	0
qc_mclicip3.IRQ120	24	RW	0
qc_mclicip3.IRQ121	25	RW	0
qc_mclicip3.IRQ122	26	RW	0
qc_mclicip3.IRQ123	27	RW	0
qc_mclicip3.IRQ124	28	RW	0
qc_mclicip3.IRQ125	29	RW	0
qc_mclicip3.IRQ126	30	RW	0
qc_mclicip3.IRQ127	31	RW	0

5.12.4. 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

5.13. qc_mclicip4

IRQ Pending 4

Pending bits for IRQs 128-159

5.13.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f4
Length	32-bit
Privilege Mode	M

5.13.2. 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_mclicip4 format

5.13.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip4.IRQ128	0	RW	0
qc_mclicip4.IRQ129	1	RW	0
qc_mclicip4.IRQ130	2	RW	0
qc_mclicip4.IRQ131	3	RW	0
qc_mclicip4.IRQ132	4	RW	0
qc_mclicip4.IRQ133	5	RW	0
qc_mclicip4.IRQ134	6	RW	0
qc_mclicip4.IRQ135	7	RW	0
qc_mclicip4.IRQ136	8	RW	0
qc_mclicip4.IRQ137	9	RW	0
qc_mclicip4.IRQ138	10	RW	0
qc_mclicip4.IRQ139	11	RW	0
qc_mclicip4.IRQ140	12	RW	0
qc_mclicip4.IRQ141	13	RW	0
qc_mclicip4.IRQ142	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip4.IRQ143	15	RW	0
qc_mclicip4.IRQ144	16	RW	0
qc_mclicip4.IRQ145	17	RW	0
qc_mclicip4.IRQ146	18	RW	0
qc_mclicip4.IRQ147	19	RW	0
qc_mclicip4.IRQ148	20	RW	0
qc_mclicip4.IRQ149	21	RW	0
qc_mclicip4.IRQ150	22	RW	0
qc_mclicip4.IRQ151	23	RW	0
qc_mclicip4.IRQ152	24	RW	0
qc_mclicip4.IRQ153	25	RW	0
qc_mclicip4.IRQ154	26	RW	0
qc_mclicip4.IRQ155	27	RW	0
qc_mclicip4.IRQ156	28	RW	0
qc_mclicip4.IRQ157	29	RW	0
qc_mclicip4.IRQ158	30	RW	0
qc_mclicip4.IRQ159	31	RW	0

5.13.4. 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

5.14. qc_mclicip5

IRQ Pending 5

Pending bits for IRQs 160-191

5.14.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f5
Length	32-bit
Privilege Mode	M

5.14.2. Format

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

Figure 14. qc_mclicip5 format

5.14.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip5.IRQ160	0	RW	0
qc_mclicip5.IRQ161	1	RW	0
qc_mclicip5.IRQ162	2	RW	0
qc_mclicip5.IRQ163	3	RW	0
qc_mclicip5.IRQ164	4	RW	0
qc_mclicip5.IRQ165	5	RW	0
qc_mclicip5.IRQ166	6	RW	0
qc_mclicip5.IRQ167	7	RW	0
qc_mclicip5.IRQ168	8	RW	0
qc_mclicip5.IRQ169	9	RW	0
qc_mclicip5.IRQ170	10	RW	0
qc_mclicip5.IRQ171	11	RW	0
qc_mclicip5.IRQ172	12	RW	0
qc_mclicip5.IRQ173	13	RW	0
qc_mclicip5.IRQ174	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip5.IRQ175	15	RW	0
qc_mclicip5.IRQ176	16	RW	0
qc_mclicip5.IRQ177	17	RW	0
qc_mclicip5.IRQ178	18	RW	0
qc_mclicip5.IRQ179	19	RW	0
qc_mclicip5.IRQ180	20	RW	0
qc_mclicip5.IRQ181	21	RW	0
qc_mclicip5.IRQ182	22	RW	0
qc_mclicip5.IRQ183	23	RW	0
qc_mclicip5.IRQ184	24	RW	0
qc_mclicip5.IRQ185	25	RW	0
qc_mclicip5.IRQ186	26	RW	0
qc_mclicip5.IRQ187	27	RW	0
qc_mclicip5.IRQ188	28	RW	0
qc_mclicip5.IRQ189	29	RW	0
qc_mclicip5.IRQ190	30	RW	0
qc_mclicip5.IRQ191	31	RW	0

5.14.4. 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

5.15. qc_mclicip6

IRQ Pending 6

Pending bits for IRQs 192-223

5.15.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f6
Length	32-bit
Privilege Mode	M

5.15.2. Format

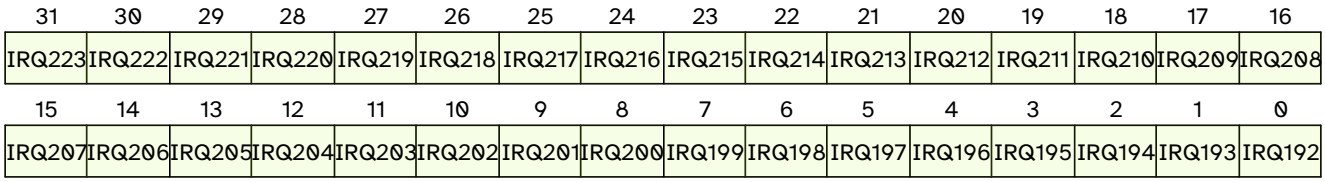


Figure 15. qc_mclicip6 format

5.15.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip6.IRQ192	0	RW	0
qc_mclicip6.IRQ193	1	RW	0
qc_mclicip6.IRQ194	2	RW	0
qc_mclicip6.IRQ195	3	RW	0
qc_mclicip6.IRQ196	4	RW	0
qc_mclicip6.IRQ197	5	RW	0
qc_mclicip6.IRQ198	6	RW	0
qc_mclicip6.IRQ199	7	RW	0
qc_mclicip6.IRQ200	8	RW	0
qc_mclicip6.IRQ201	9	RW	0
qc_mclicip6.IRQ202	10	RW	0
qc_mclicip6.IRQ203	11	RW	0
qc_mclicip6.IRQ204	12	RW	0
qc_mclicip6.IRQ205	13	RW	0
qc_mclicip6.IRQ206	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip6.IRQ207	15	RW	0
qc_mclicip6.IRQ208	16	RW	0
qc_mclicip6.IRQ209	17	RW	0
qc_mclicip6.IRQ210	18	RW	0
qc_mclicip6.IRQ211	19	RW	0
qc_mclicip6.IRQ212	20	RW	0
qc_mclicip6.IRQ213	21	RW	0
qc_mclicip6.IRQ214	22	RW	0
qc_mclicip6.IRQ215	23	RW	0
qc_mclicip6.IRQ216	24	RW	0
qc_mclicip6.IRQ217	25	RW	0
qc_mclicip6.IRQ218	26	RW	0
qc_mclicip6.IRQ219	27	RW	0
qc_mclicip6.IRQ220	28	RW	0
qc_mclicip6.IRQ221	29	RW	0
qc_mclicip6.IRQ222	30	RW	0
qc_mclicip6.IRQ223	31	RW	0

5.15.4. 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

5.16. qc_mclicip7

IRQ Pending 7

Pending bits for IRQs 224-255

5.16.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7f7
Length	32-bit
Privilege Mode	M

5.16.2. Format

Figure 16. qc_mclicip7 format

5.16.3. Field Summary

Name	Location	Type	Reset Value
qc_mclicip7.IRQ224	0	RW	0
qc_mclicip7.IRQ225	1	RW	0
qc_mclicip7.IRQ226	2	RW	0
qc_mclicip7.IRQ227	3	RW	0
qc_mclicip7.IRQ228	4	RW	0
qc_mclicip7.IRQ229	5	RW	0
qc_mclicip7.IRQ230	6	RW	0
qc_mclicip7.IRQ231	7	RW	0
qc_mclicip7.IRQ232	8	RW	0
qc_mclicip7.IRQ233	9	RW	0
qc_mclicip7.IRQ234	10	RW	0
qc_mclicip7.IRQ235	11	RW	0
qc_mclicip7.IRQ236	12	RW	0
qc_mclicip7.IRQ237	13	RW	0
qc_mclicip7.IRQ238	14	RW	0

Name	Location	Type	Reset Value
qc_mclicip7.IRQ239	15	RW	0
qc_mclicip7.IRQ240	16	RW	0
qc_mclicip7.IRQ241	17	RW	0
qc_mclicip7.IRQ242	18	RW	0
qc_mclicip7.IRQ243	19	RW	0
qc_mclicip7.IRQ244	20	RW	0
qc_mclicip7.IRQ245	21	RW	0
qc_mclicip7.IRQ246	22	RW	0
qc_mclicip7.IRQ247	23	RW	0
qc_mclicip7.IRQ248	24	RW	0
qc_mclicip7.IRQ249	25	RW	0
qc_mclicip7.IRQ250	26	RW	0
qc_mclicip7.IRQ251	27	RW	0
qc_mclicip7.IRQ252	28	RW	0
qc_mclicip7.IRQ253	29	RW	0
qc_mclicip7.IRQ254	30	RW	0
qc_mclicip7.IRQ255	31	RW	0

5.16.4. 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

5.17. qc_mncause

Machine NMI Cause

Reports the cause of the latest non-maskable interrupt.

5.17.1. Attributes

Defining Extension	• anyOf:
	  • Xqci, version >= 0
	  • Xqciint, version >= 0

CSR Address	0x7c2
Length	32-bit
Privilege Mode	M

5.17.2. Format

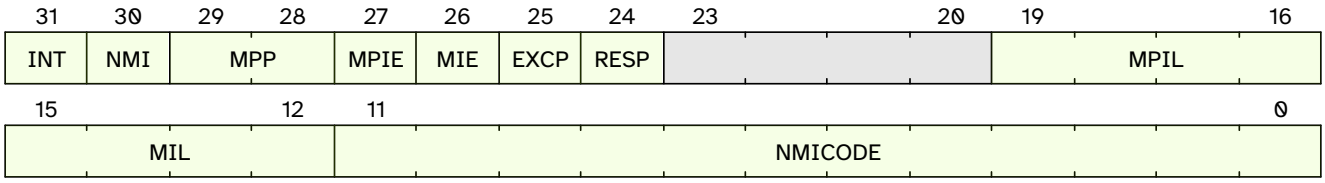


Figure 17. qc_mncause format

5.17.3. Field Summary

Name	Location	Type	Reset Value
qc_mncause.INT	31	RW-H	0
qc_mncause.NMI	30	RW-H	0
qc_mncause.MPP	29:28	RW-H	3
qc_mncause.MPIE	27	RW-H	0
qc_mncause.MIE	26	RW-H	0
qc_mncause.EXCP	25	RW-H	0
qc_mncause.RESP	24	RW-H	0
qc_mncause.MPIL	19:16	RW-H	0
qc_mncause.MIL	15:12	RW-H	15
qc_mncause.NMICODE	11:0	RW-H	0

5.17.4. Fields

`INT`

Location

31

Description

Interrupt bit copied from mcause.INT at the moment of NMI

Type

RW-H

Reset value

0

`NMI`

Location

30

Description

If 1'b1, currently processing NMI.

Type

RW-H

Reset value

0

`MPP`

Location

29:28

Description

M-mode Previous Privilege.

Type

RW-H

Reset value

3

MPIE

Location

27

Description

M-mode Previous Interrupt Enable.

Type

RW-H

Reset value

0

MIE

Location

26

Description

M-mode Interrupt Enable.

Type

RW-H

Reset value

0

EXCP

Location

25

Description

Exception Pending Bit.

Type

RW-H

Reset value

0

`RESP`

Location

24

Description

Resume Pending Bit.

Type

RW-H

Reset value

0

`MPIL`

Location

19:16

Description

M-mode Previous Interrupt Level.

Type

RW-H

Reset value

0

`MIL`

Location

15:12

Description

M-mode Interrupt Level.

Type

RW-H

Reset value

15

NMICODE

Location

11:0

Description

NMI code ID.

Type

RW-H

Reset value

0

Chapter 6. Instructions (in alphabetical order)

6.1. qc.addsat

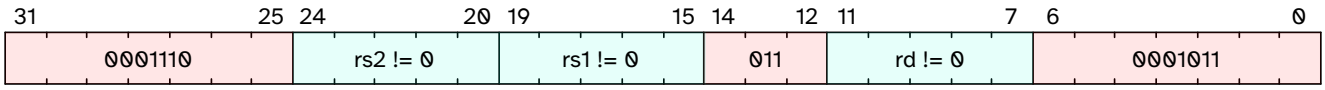
Synopsis

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`. Instruction encoded in R instruction format

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	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.2. qc.addusat

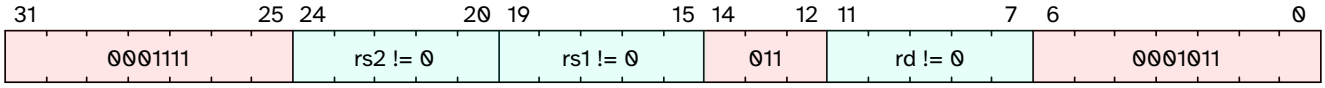
Synopsis

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`. Instruction encoded in R instruction format

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	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.3. qc.beqi

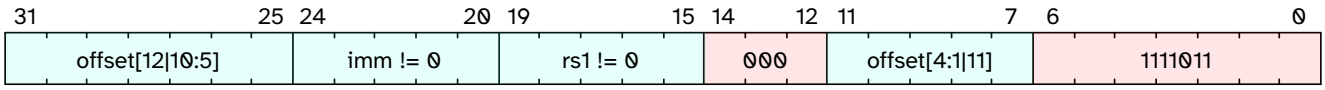
Synopsis

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 Instruction encoded in BI instruction format

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.4. qc.bgei

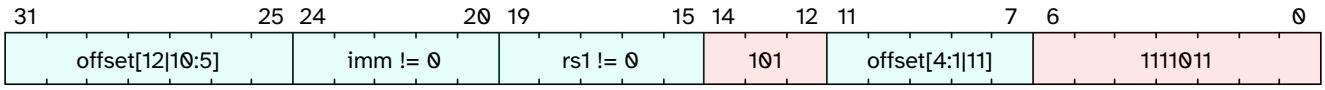
Synopsis

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. Instruction encoded in BI instruction format

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.5. qc.bgeui

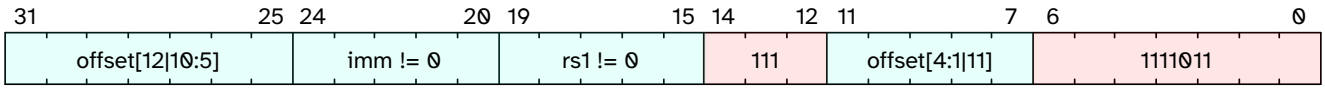
Synopsis

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. Instruction encoded in BI instruction format

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.6. qc.blti

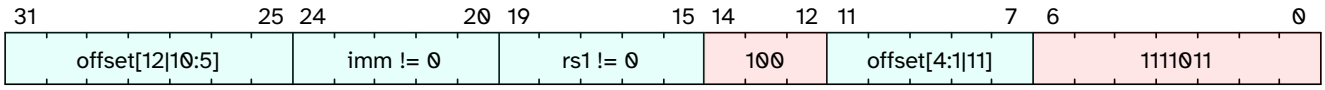
Synopsis

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. Instruction encoded in BI instruction format

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.7. qc.bltui

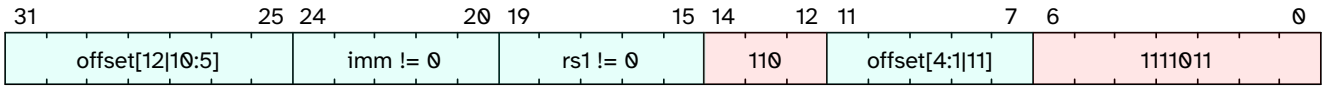
Synopsis

Branch on less than unsigned (immediate)

Mnemonic

```
qc.bltui  rs1, imm, offset
```

Encoding



Description

Branches to `PC + offset` if the unsigned value in `rs1` is less than the unsigned immediate. Instruction encoded in BI instruction format

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.8. qc.bnei

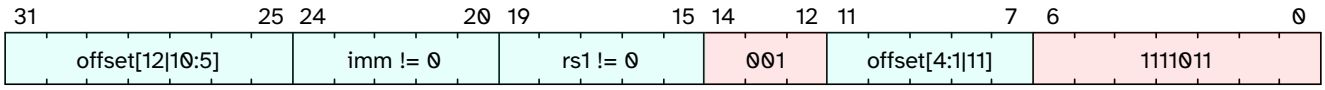
Synopsis

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. Instruction encoded in BI instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.9. qc.brev32

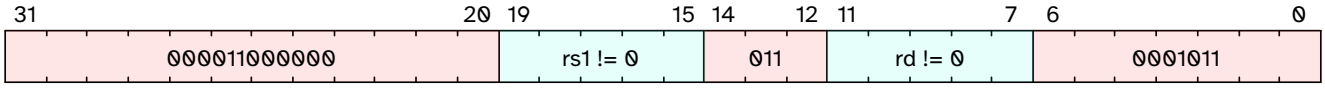
Synopsis

Reverse bit order

Mnemonic

```
qc.brev32  rd, rs1
```

Encoding



Description

Reverses the bit order of `rs1` and writes the result to `rd`. Instruction encoded in I instruction format

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.10. qc.c.bexti

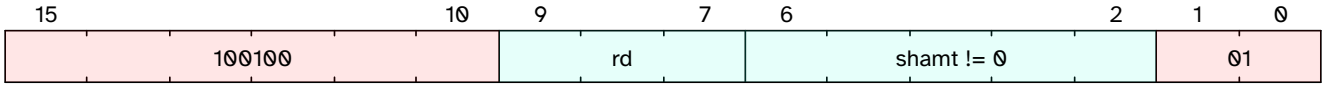
Synopsis

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`. Instruction encoded in CB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.11. qc.c.bseti

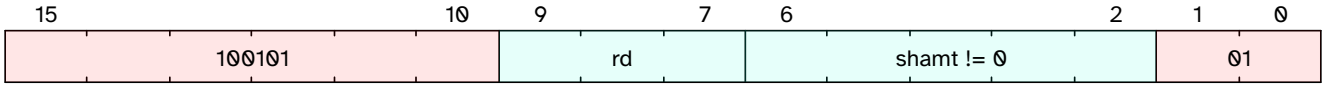
Synopsis

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`. Instruction encoded in CB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.12. qc.c.clrint

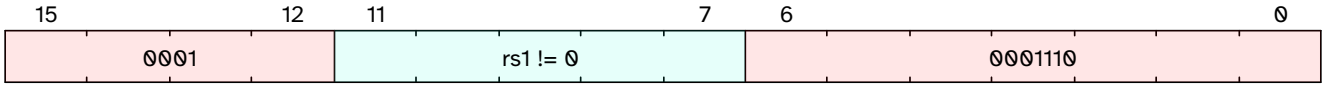
Synopsis

Clear interrupt (Register)

Mnemonic

```
qc.c.clrint  rs1
```

Encoding



Description

Clear interrupt, interrupt number is in `rs1`. Instruction encoded in CI instruction format.

Decode Variables

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

Operation

```
Bits<12> MCLICIP0_ADDR = CSR[qc_mclicip0].address();
XReg idx = rs1 / 32;
XReg bit = rs1 % 32;
XReg pre_csr = CSR[MCLICIP0_ADDR + idx].sw_read();
CSR[MCLICIP0_ADDR + idx].sw_write(pre_csr & ~(1 << bit));
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.13. qc.c.delay

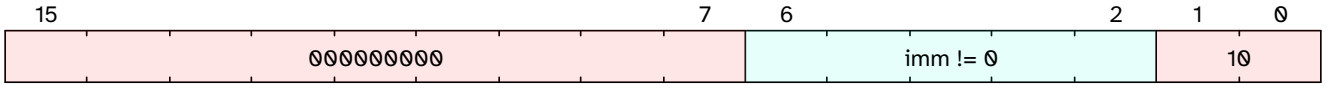
Synopsis

Delay execution for immediate amount of cycles

Mnemonic

```
qc.c.delay
```

Encoding



Description

Delay execution for amount of cycles provided as immediate argument Instruction encoded in CI instruction format.

Decode Variables

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

Operation

```
delay(imm);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.14. qc.c.di

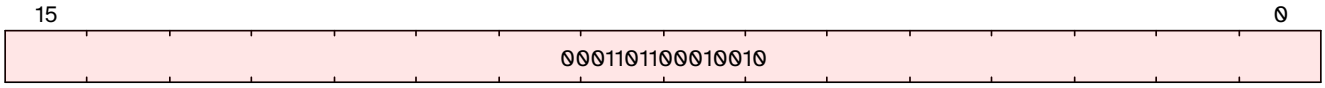
Synopsis

Disable interrupts

Mnemonic

```
qc.c.di
```

Encoding



Description

Globally disable interrupts. Equivalent to "csrrci `zero`, mstatus, 8". Instruction encoded in CI instruction format.

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	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.15. qc.c.dir

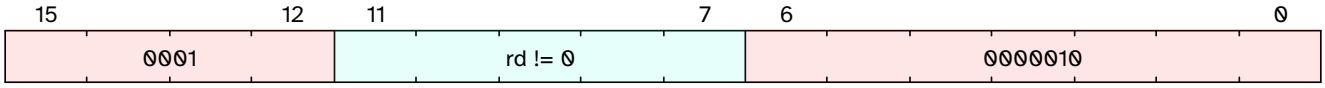
Synopsis

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". Instruction encoded in CI instruction format.

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	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.16. qc.c.ei

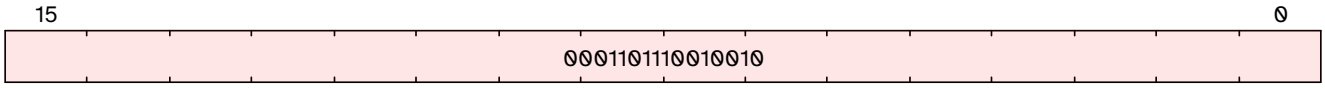
Synopsis

Enable interrupts

Mnemonic

```
qc.c.ei
```

Encoding



Description

Globally enable interrupts. Equivalent to "csrrsi `zero`, mstatus, 8". Instruction encoded in CI instruction format.

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	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.17. qc.c.eir

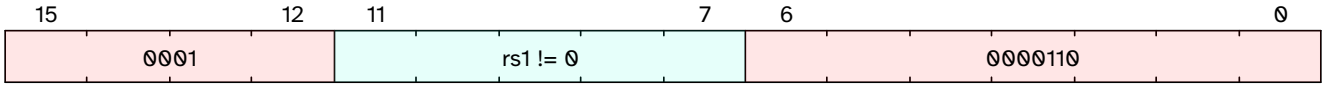
Synopsis

Restore interrupts (Register)

Mnemonic

```
qc.c.eir  rs1
```

Encoding



Description

Globally restore interrupts, write `rs1` to mstatus. Equivalent to "csrrs `zero`, mstatus, `rs1`". Instruction encoded in CI instruction format.

Decode Variables

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

Operation

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

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.18. qc.c.extu

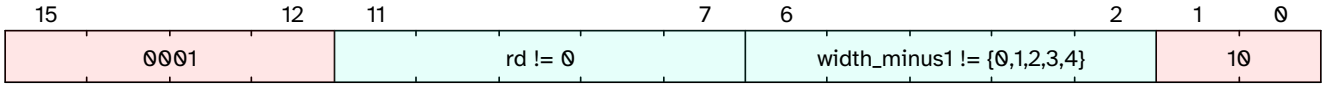
Synopsis

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). Instruction encoded in CI instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.19. qc.c.mienter.nest

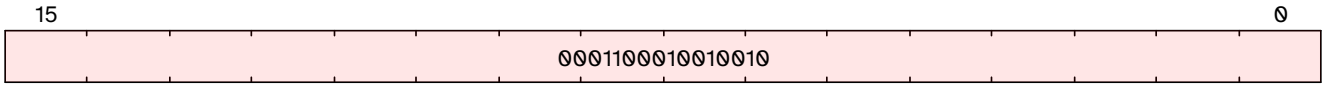
Synopsis

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. Instruction encoded in CI instruction format.

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

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

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.20. qc.c.mienter

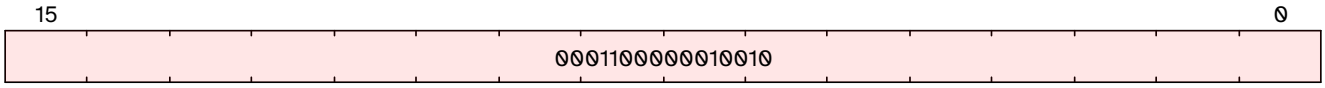
Synopsis

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. Instruction encoded in CI instruction format.

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

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

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.21. qc.c.mileaveret

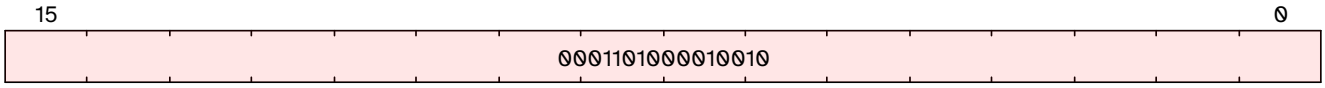
Synopsis

Machine mode interrupt exit

Mnemonic

```
qc.c.mileaveret
```

Encoding



Description

Machine mode interrupt exit. Interrupt frame is restored from the stack. Instruction encoded in CI instruction format.

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

```
X[31] = read_memory<32>(virtual_address - 80, $encoding);
X[2] = X[2] + 96;
$pc = curr_retpc;
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.22. qc.c.mnret

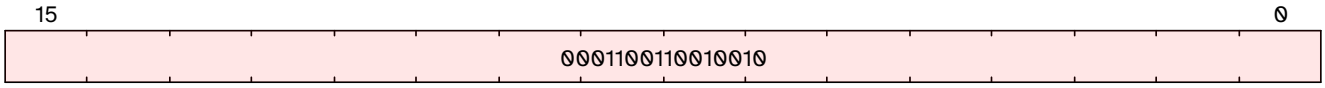
Synopsis

Machine NMI Return

Mnemonic

```
qc.c.mnret
```

Encoding



Description

Returns from an NMI in M-mode. Instruction encoded in CI instruction format.

Decode Variables

qc.c.mnret has no decode variables.

Operation

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

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.23. qc.c.mret

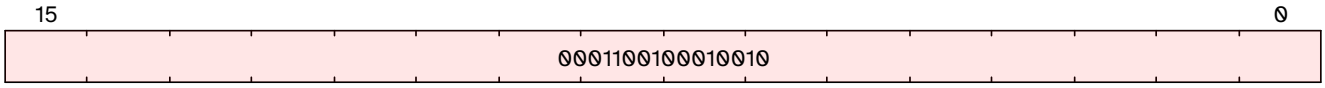
Synopsis

Machine Exception Return

Mnemonic

```
qc.c.mret
```

Encoding



Description

Returns from an exception in M-mode. Instruction encoded in CI instruction format.

Decode Variables

qc.c.mret has no decode variables.

Operation

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

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.24. qc.c.muliadd

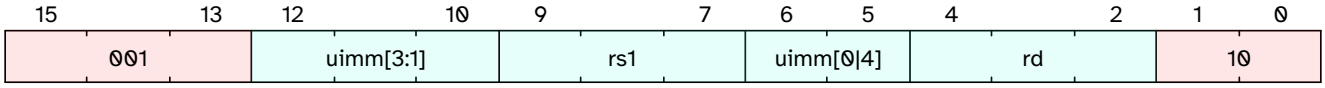
Synopsis

Multiply and accumulate (Immediate)

Mnemonic

```
qc.c.muliadd  rd, rs1, uimm
```

Encoding



Description

Increments `rd` by the multiplication of `rs1` and an unsigned immediate Instruction encoded in CL instruction format.

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	Version
Xqci	~> 0.1.0
Xqciac	~> 0.1.0

6.25. qc.c.mveqz

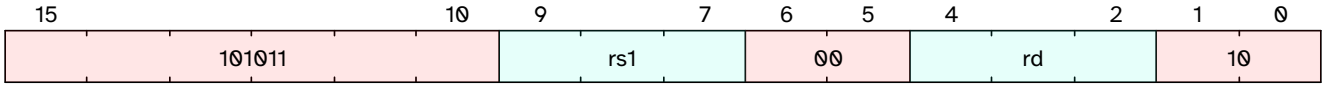
Synopsis

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 Instruction encoded in CL instruction format.

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	Version
Xqci	~> 0.1.0
Xqciac	~> 0.1.0

6.26. qc.c.ptrace

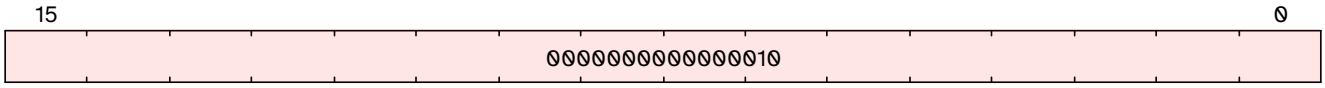
Synopsis

Tracing pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.c.ptrace
```

Encoding



Description

The tracing instruction have no explicit arguments. Implicit arguments defined by simulation environment implementation. Instruction is used to signal simulator to collect some tracing information. Instruction encoded in CI instruction format.

Decode Variables

qc.c.ptrace has no decode variables.

Operation

```
XReg func = 9;
XReg arg = 0;
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.27. qc.c.setint

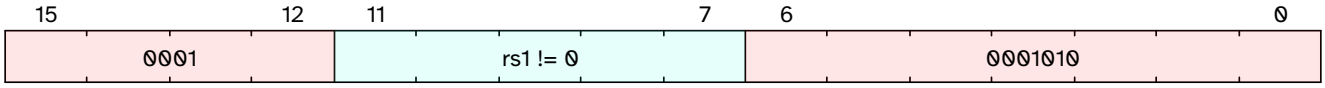
Synopsis

Set interrupt (Register)

Mnemonic

```
qc.c.setint  rs1
```

Encoding



Description

Set interrupt, interrupt number is in `rs1`. Instruction encoded in CI instruction format.

Decode Variables

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

Operation

```
Bits<12> MCLICIP0_ADDR = CSR[qc_mclicip0].address();
XReg idx = rs1 / 32;
XReg bit = rs1 % 32;
XReg pre_csr = CSR[MCLICIP0_ADDR + idx].sw_read();
CSR[MCLICIP0_ADDR + idx].sw_write(pre_csr | (1 << bit));
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.28. qc.c.sync

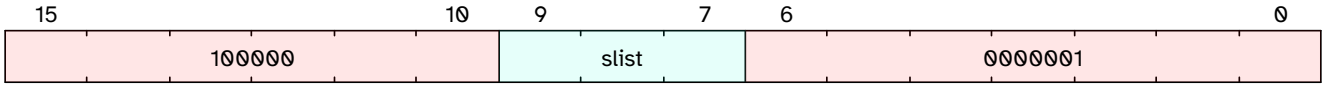
Synopsis

Non-memory-mapped device read-after-write synchronization to completion

Mnemonic

```
qc.c.sync  slist
```

Encoding



Description

This synchronization instruction delays execution till last read (input) from non-memory-mapped device transactions are completed for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 3-bit encoding of 5-bit bitmask field that specifies up to five pre-defined devices. Values of bitmask encoding supported: 0,1,2,4,8,16,15,31 Instruction encoded in CB instruction format.

Decode Variables

```
Bits<3> slist = $encoding[9:7];
```

Operation

```
Bits<5> bitmask;
if (slist == 0) {
  bitmask = 0;
} else if (slist < 6) {
  XReg shift = slist - 1;
  bitmask = (1 << shift);
} else {
  XReg shift = slist - 2;
  bitmask = (1 << shift) - 1;
}
fence_tso();
sync_read_after_write_device(true, bitmask);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.29. qc.c.syncr

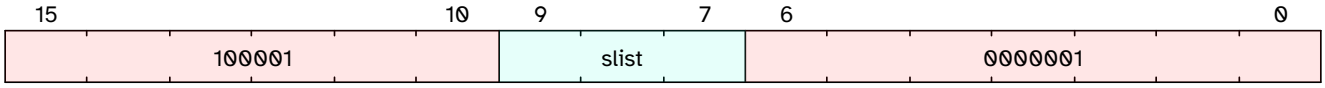
Synopsis

Non-memory-mapped device read-after-write synchronization

Mnemonic

```
qc.c.syncr  slist
```

Encoding



Description

This synchronization instruction delays execution till last read (input) from non-memory-mapped device transactions are started for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 3-bit encoding of 5-bit bitmask field that specifies up to five pre-defined devices. Values of bitmask encoding supported: 0,1,2,4,8,16,15,31 Instruction encoded in CB instruction format.

Decode Variables

```
Bits<3> slist = $encoding[9:7];
```

Operation

```
Bits<5> bitmask;
if (slist == 0) {
  bitmask = 0;
} else if (slist < 6) {
  XReg shift = slist - 1;
  bitmask = (1 << shift);
} else {
  XReg shift = slist - 2;
  bitmask = (1 << shift) - 1;
}
fence_tso();
sync_read_after_write_device(false, bitmask);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.30. qc.c.syncwf

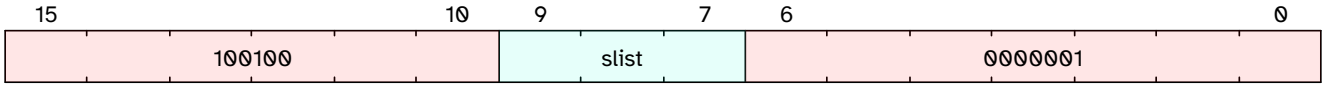
Synopsis

Non-memory-mapped device write-after-read synchronization

Mnemonic

```
qc.c.syncwf  slist
```

Encoding



Description

This synchronization instruction delays execution till last write (output) to non-memory-mapped device transactions are started for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 3-bit encoding of 5-bit bitmask field that specifies up to five pre-defined devices. Values of bitmask encoding supported: 0,1,2,4,8,16,15,31 Instruction encoded in CB instruction format.

Decode Variables

```
Bits<3> slist = $encoding[9:7];
```

Operation

```
Bits<5> bitmask;
if (slist == 0) {
  bitmask = 0;
} else if (slist < 6) {
  XReg shift = slist - 1;
  bitmask = (1 << shift);
} else {
  XReg shift = slist - 2;
  bitmask = (1 << shift) - 1;
}
fence_tso();
sync_write_after_read_device(false, bitmask);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.31. qc.c.syncwl

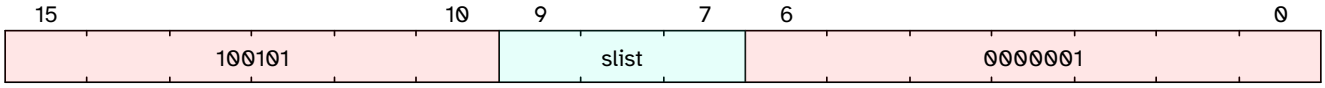
Synopsis

Non-memory-mapped device write-after-read synchronization to completion

Mnemonic

```
qc.c.syncwl  slist
```

Encoding



Description

This synchronization instruction delays execution till last write (output) to non-memory-mapped device transactions are completed for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 3-bit encoding of 5-bit bitmask field that specifies up to five pre-defined devices. Values of bitmask encoding supported: 0,1,2,4,8,16,15,31 Instruction encoded in CB instruction format.

Decode Variables

```
Bits<3> slist = $encoding[9:7];
```

Operation

```
Bits<5> bitmask;
if (slist == 0) {
  bitmask = 0;
} else if (slist < 6) {
  XReg shift = slist - 1;
  bitmask = (1 << shift);
} else {
  XReg shift = slist - 2;
  bitmask = (1 << shift) - 1;
}
fence_tso();
sync_write_after_read_device(true, bitmask);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.32. qc.clo

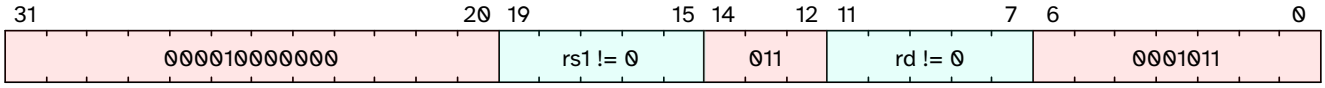
Synopsis

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` Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.33. qc.clrinti

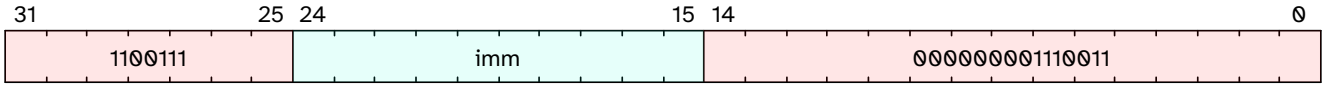
Synopsis

Clear interrupt (Immediate)

Mnemonic

```
qc.clrinti  imm
```

Encoding



Description

Clear interrupt, interrupt number is in `imm` (0 - 1023). Instruction encoded in R instruction format.

Decode Variables

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

Operation

```
Bits<12> MCLICIP0_ADDR = CSR[qc_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	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.34. qc.compress2

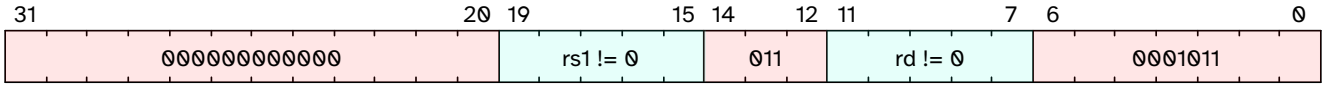
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.35. qc.compress3

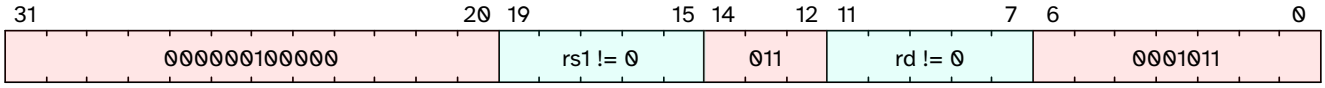
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.36. qc.csrrwr

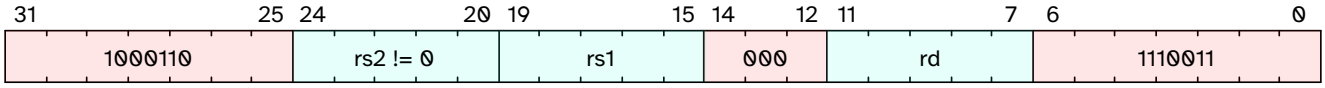
Synopsis

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. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicsr	~> 0.1.0

6.37. qc.csrrwri

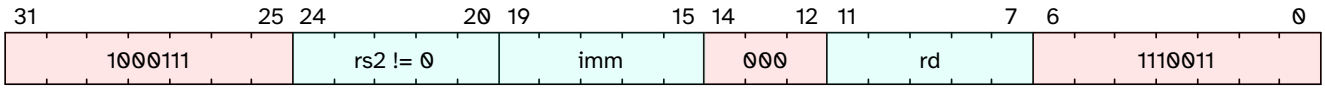
Synopsis

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. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicsr	~> 0.1.0

6.38. qc.cto

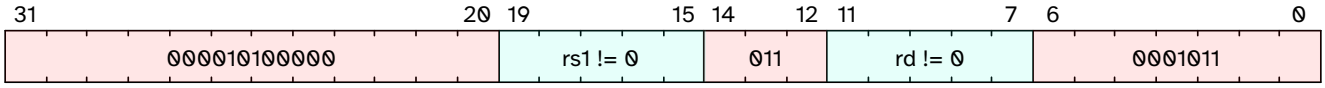
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.39. qc.e.addai

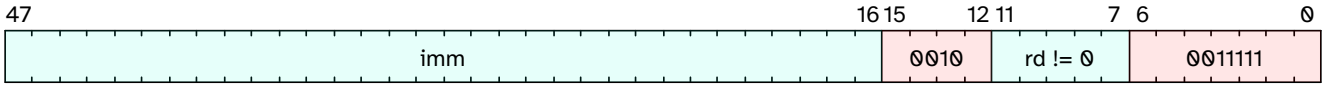
Synopsis

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`. Instruction encoded in QC.EAI instruction format.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] + imm;
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.40. qc.e.addi

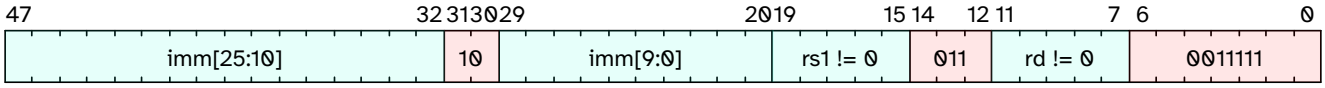
Synopsis

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`. Instruction encoded in QC.EI instruction format.

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	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.41. qc.e.andai

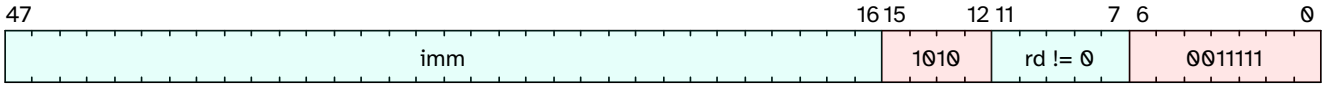
Synopsis

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`. Instruction encoded in QC.EAI instruction format.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] & imm;
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.42. qc.e.andi

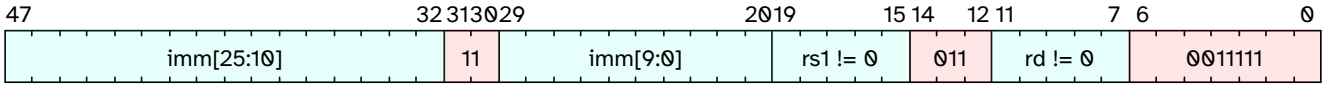
Synopsis

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`. Instruction encoded in QC.EI instruction format.

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	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.43. qc.e.beqi

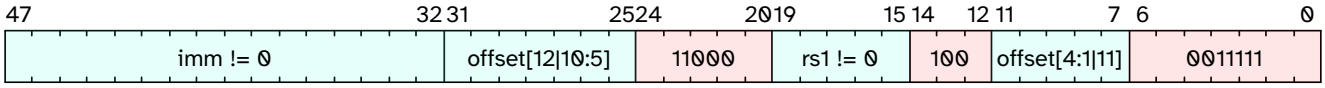
Synopsis

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` Instruction encoded in QC.EB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.44. qc.e.bgei

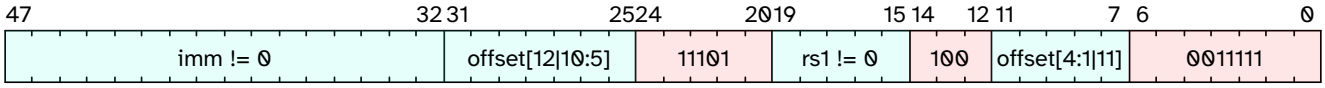
Synopsis

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`. Instruction encoded in QC.EB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.45. qc.e.bgeui

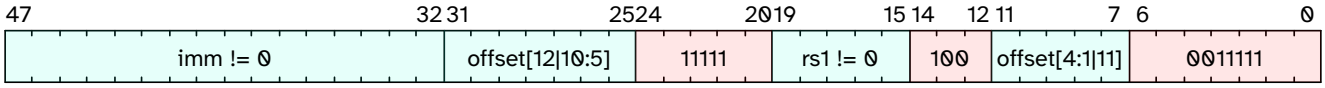
Synopsis

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`. Instruction encoded in QC.EB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.46. qc.e.blti

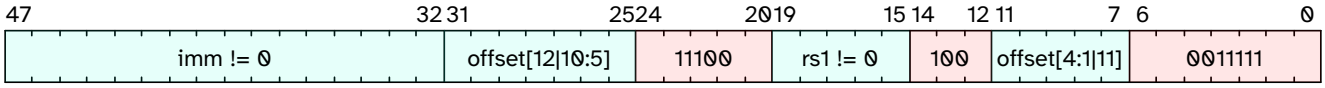
Synopsis

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`. Instruction encoded in QC.EB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.47. qc.e.bltui

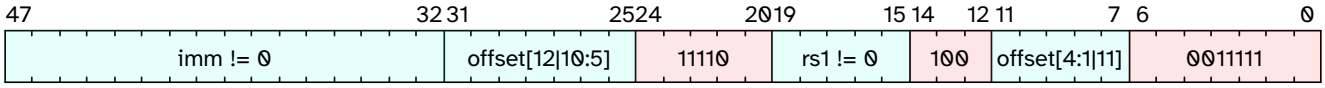
Synopsis

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`. Instruction encoded in QC.EB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.48. qc.e.bnei

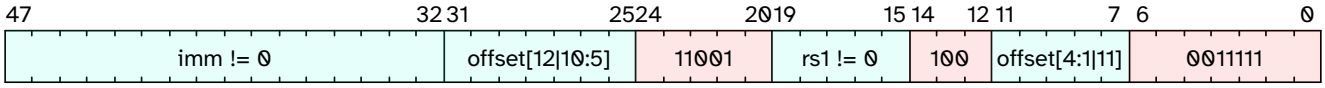
Synopsis

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`. Instruction encoded in QC.EB instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibi	~> 0.1.0

6.49. qc.e.j

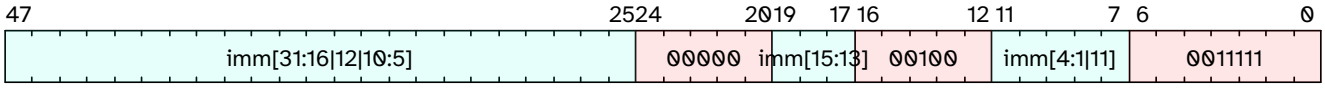
Synopsis

Jump

Mnemonic

```
qc.e.j  imm
```

Encoding



Description

Jump to a PC-relative offset. Instruction encoded in QC.EJ instruction format.

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

```
jump_halfword($pc + imm);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilb	~> 0.1.0

6.50. qc.e.jal

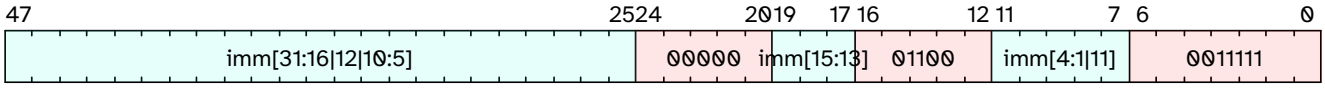
Synopsis

Jump and link

Mnemonic

```
qc.e.jal  imm
```

Encoding



Description

Jump to a PC-relative offset and store the return. Instruction encoded in QC.EJ instruction format. 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	Version
Xqci	~> 0.1.0
Xqcilb	~> 0.1.0

6.51. qc.e.lb

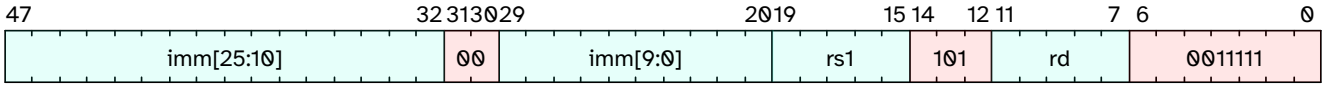
Synopsis

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. Instruction encoded in QC.EI instruction format.

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, $encoding), 7);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.52. qc.e.lbu

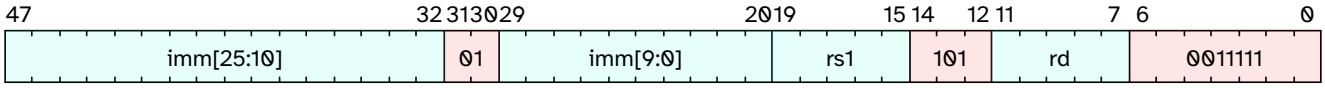
Synopsis

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. Instruction encoded in QC.EI instruction format.

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, $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.53. qc.e.lh

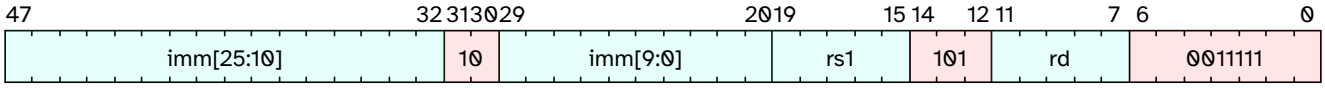
Synopsis

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. Instruction encoded in QC.EI instruction format.

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, $encoding), 15);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.54. qc.e.lhu

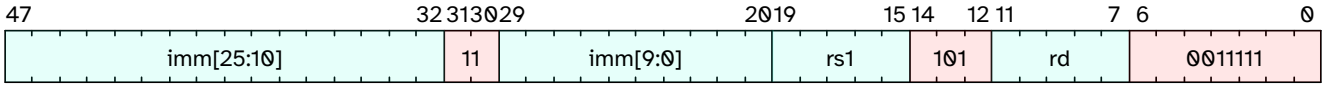
Synopsis

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. Instruction encoded in QC.EI instruction format.

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, $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.55. qc.e.li

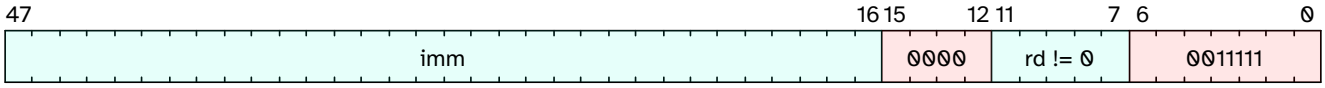
Synopsis

Load immediate large

Mnemonic

```
qc.e.li  rd, imm
```

Encoding



Description

Loads the 32-bit immediate `imm` into `rd`. Instruction encoded in QC.EAI instruction format.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = imm;
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcili	~> 0.1.0

6.56. qc.e.lw

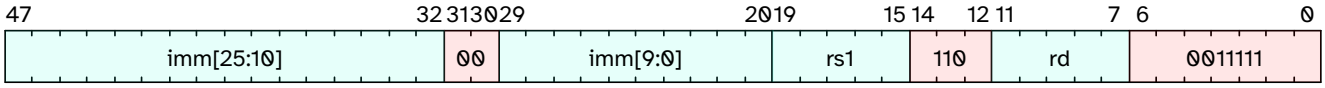
Synopsis

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. Instruction encoded in QC.EI instruction format.

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, $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.57. qc.e.orai

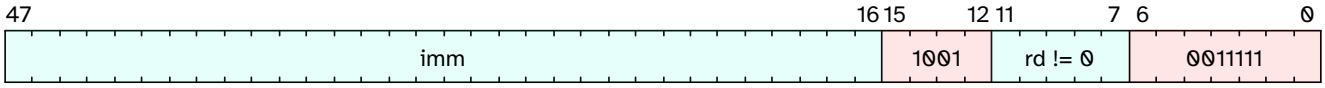
Synopsis

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`. Instruction encoded in QC.EAI instruction format.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] | imm;
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.58. qc.e.ori

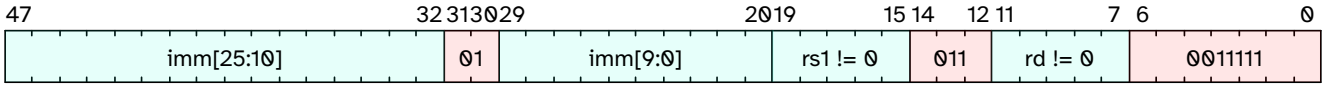
Synopsis

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`. Instruction encoded in QC.EI instruction format.

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	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.59. qc.e.sb

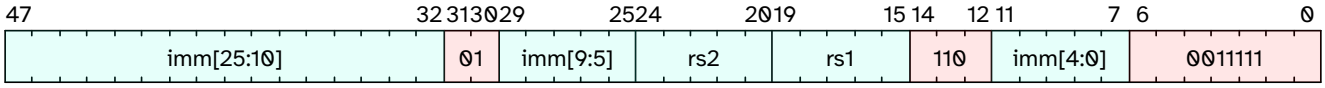
Synopsis

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`. Instruction encoded in QC.ES instruction format.

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], $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.60. qc.e.sh

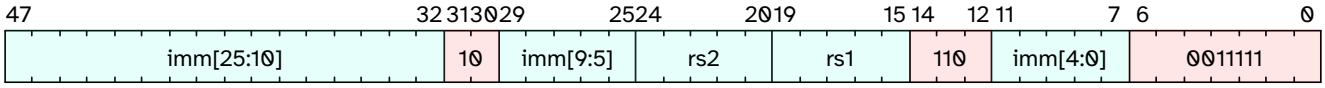
Synopsis

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`. Instruction encoded in QC.ES instruction format.

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], $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.61. qc.e.sw

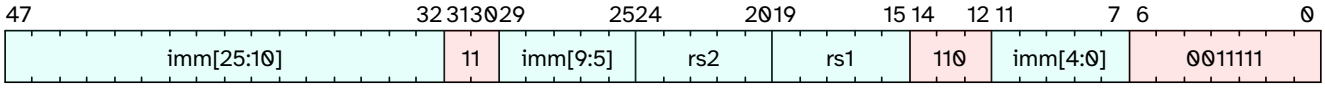
Synopsis

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`. Instruction encoded in QC.ES instruction format.

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], $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilo	~> 0.1.0

6.62. qc.e.xorai

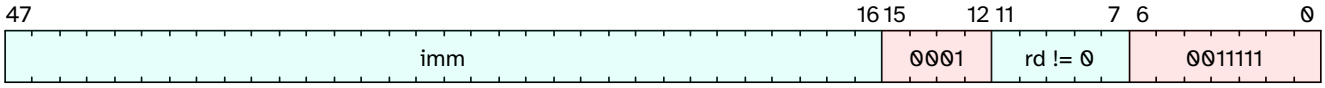
Synopsis

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`. Instruction encoded in QC.EAI instruction format.

Decode Variables

```
Bits<32> imm = $encoding[47:16];
Bits<5> rd = $encoding[11:7];
```

Operation

```
X[rd] = X[rd] ^ imm;
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.63. qc.e.xori

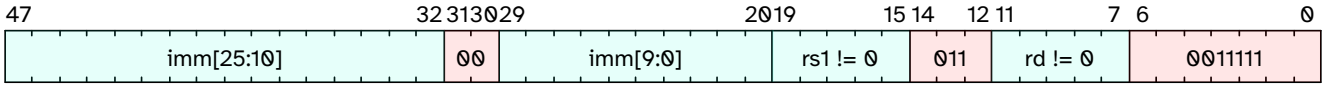
Synopsis

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`. Instruction encoded in QC.EI instruction format.

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	Version
Xqci	~> 0.1.0
Xqcilia	~> 0.1.0

6.64. qc.expand2

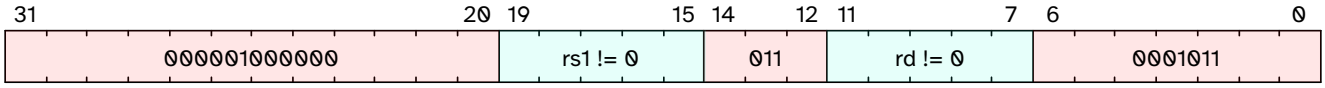
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.65. qc.expand3

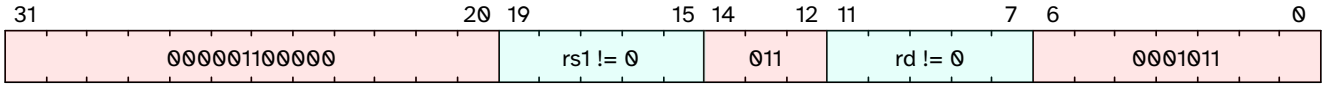
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.66. qc.ext

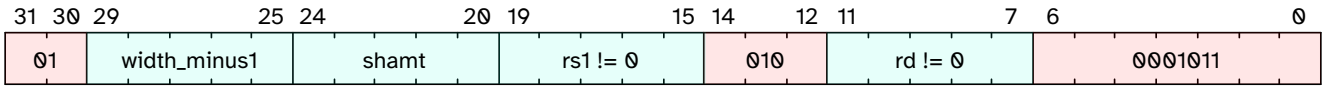
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.67. qc.extd

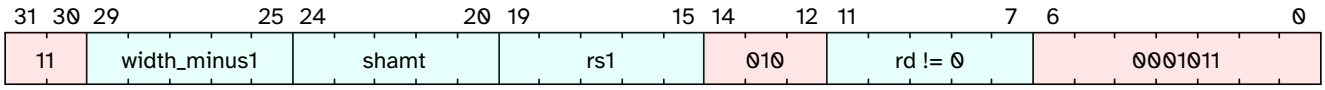
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.68. qc.extdpr

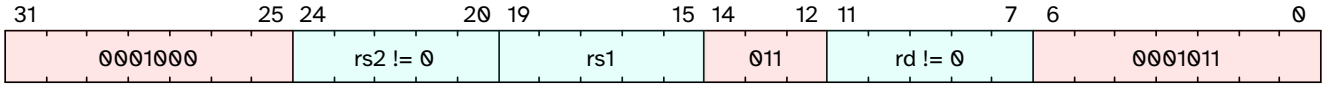
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.69. qc.extdprh

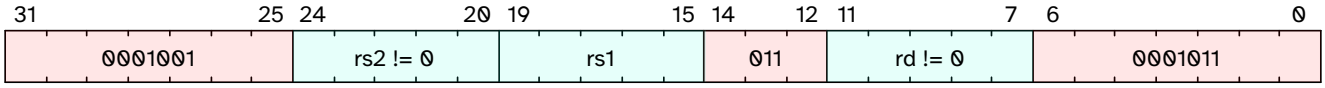
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.70. qc.extdr

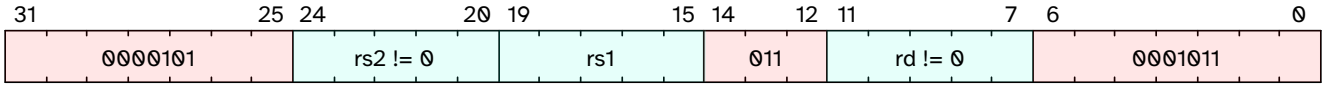
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.71. qc.extdu

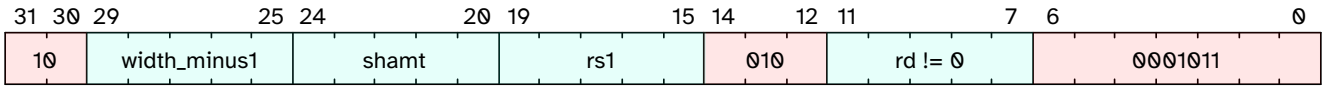
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.72. qc.extdupr

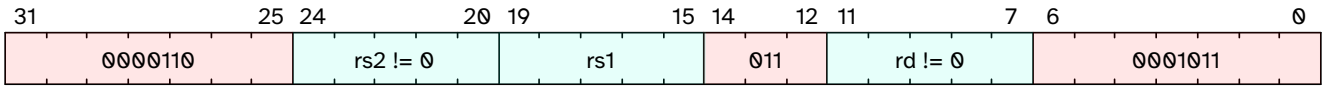
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.73. qc.extduprh

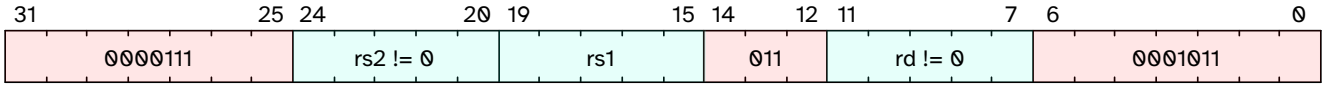
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.74. qc.extdur

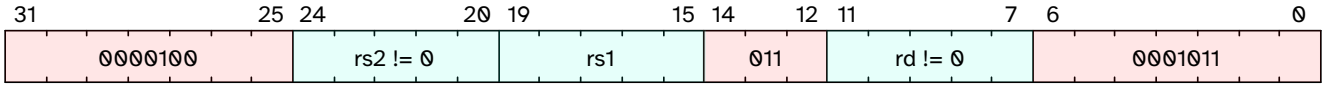
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.75. qc.extu

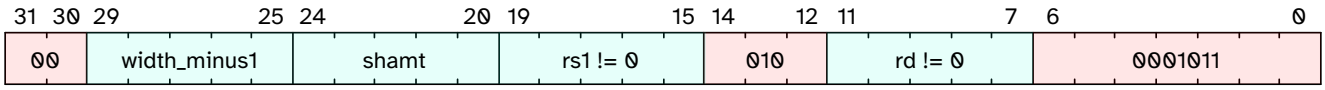
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.76. qc.insb

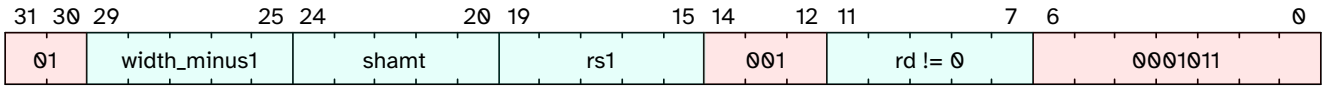
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.77. qc.insbh

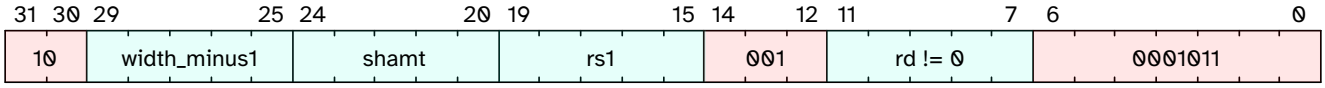
Synopsis

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 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 (`width_minus1` + 1) (1..32), and the offset of the subset is determined by `shamt`. In case when width + offset ⇐ 32, the destination register is left unchanged. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.78. qc.insbhr

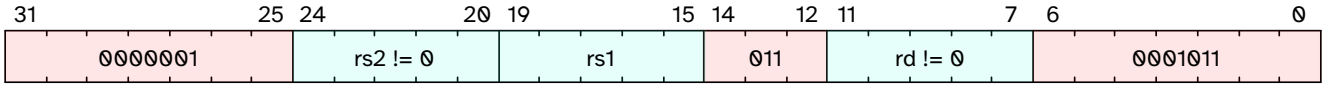
Synopsis

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 ⇐ 32, the destination register is left unchanged. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.79. qc.insbi

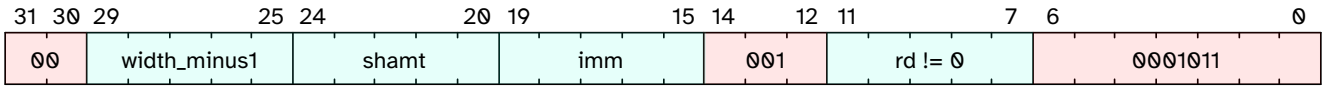
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.80. qc.insbpr

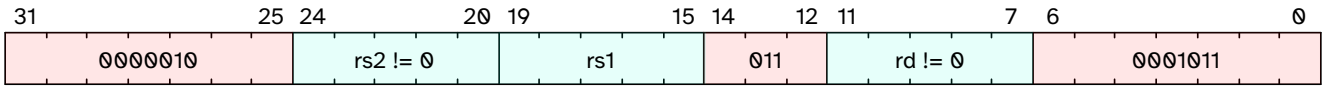
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.81. qc.insbprh

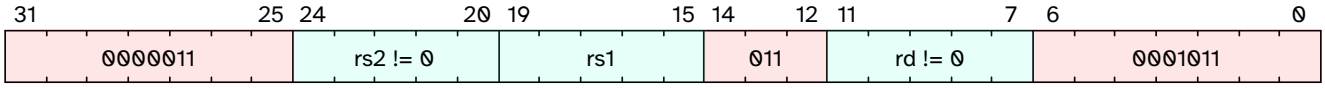
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.82. qc.insbr

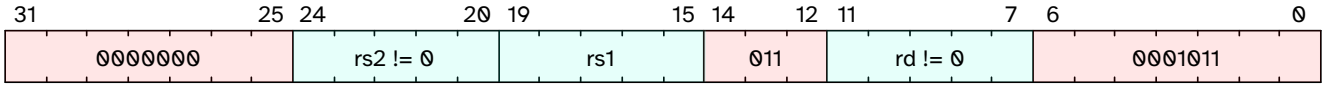
Synopsis

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]. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.83. qc.insbri

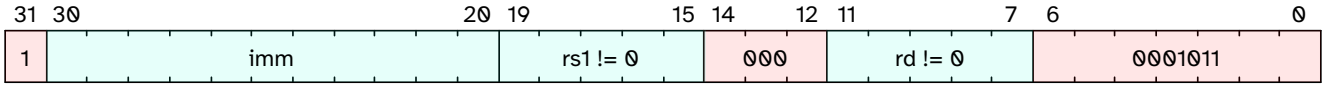
Synopsis

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]. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcibm	~> 0.1.0

6.84. qc.inw

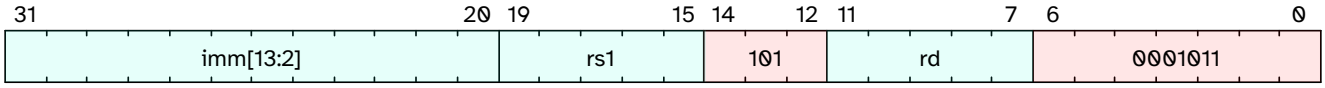
Synopsis

Input word from non-memory-mapped device

Mnemonic

```
qc.inw rd, imm(rs1)
```

Encoding



Description

Input 32 bits of data into register `rd` from a non-memory-mapped device. Such devices have own address space, unrelated to memory map. Device space address formed by adding `rs1` to to a unsigned offset `imm`. Instruction encoded in I instruction format.

Decode Variables

```
Bits<14> imm = {$encoding[31:20], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

Operation

```
XReg device_address = X[rs1] + imm;
X[rd] = read_device<32>(device_address);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciio	~> 0.1.0

6.85. qc.li

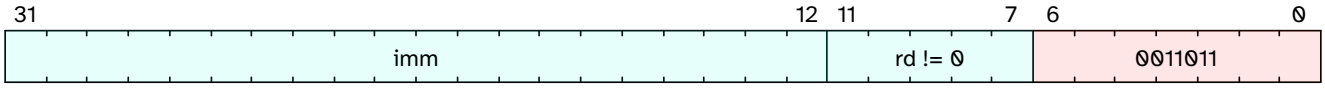
Synopsis

Load immediate large

Mnemonic

```
qc.li  rd, imm
```

Encoding



Description

Loads the 20-bit immediate `imm` into `rd`. Instruction encoded in U instruction format.

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	Version
Xqci	~> 0.1.0
Xqcili	~> 0.1.0

6.86. qc.lieq

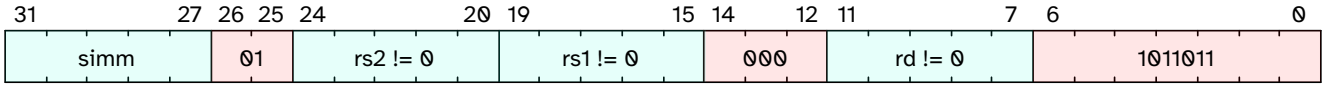
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.87. qc.lieqi

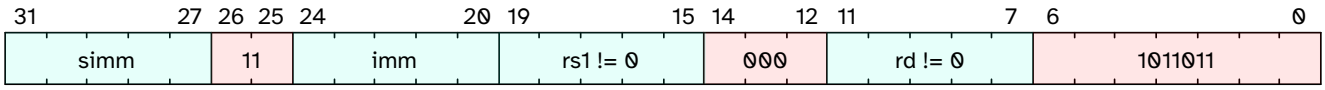
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.88. qc.lige

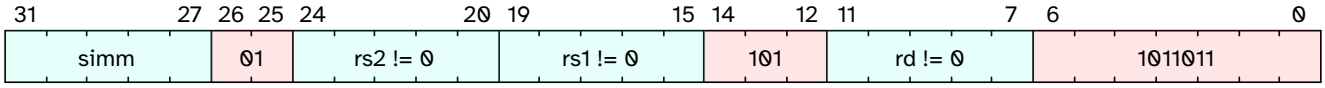
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.89. qc.ligei

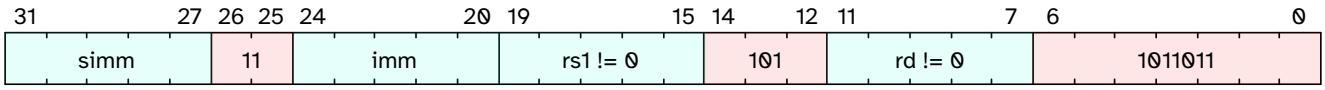
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.90. qc.ligeu

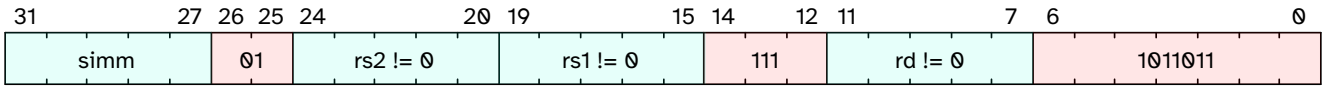
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.91. qc.ligeui

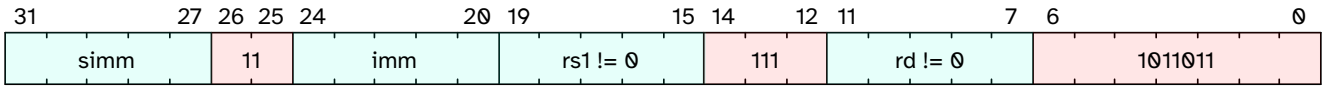
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.92. qc.lilt

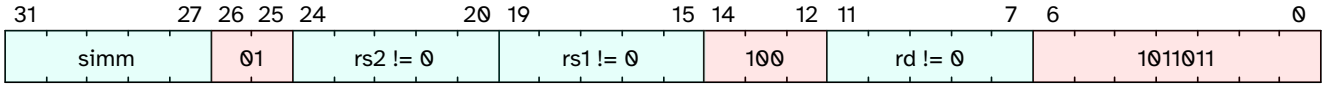
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.93. qc.lilti

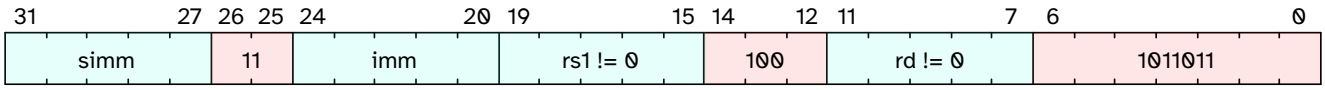
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.94. qc.liltu

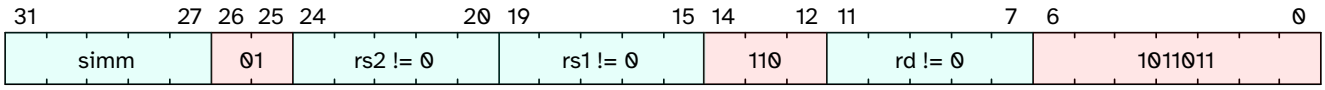
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.95. qc.liltui

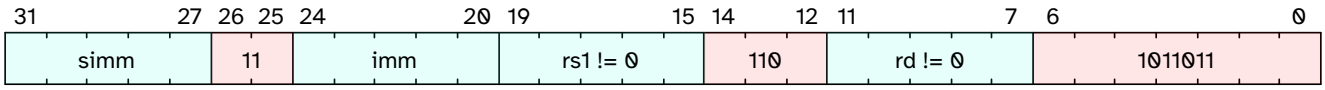
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.96. qc.line

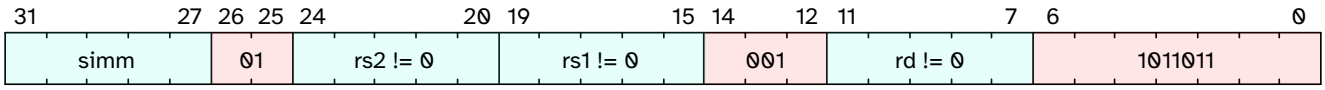
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.97. qc.linei

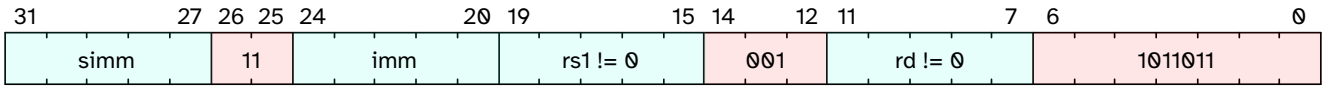
Synopsis

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`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicli	~> 0.1.0

6.98. qc.lrb

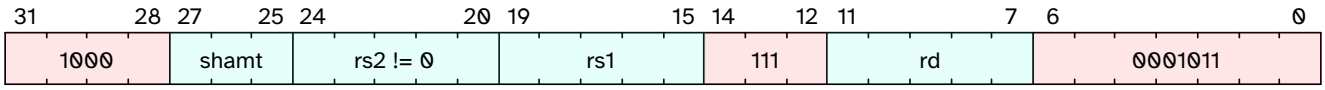
Synopsis

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. Instruction encoded in R instruction format.

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, $encoding), 8);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.99. qc.lrbu

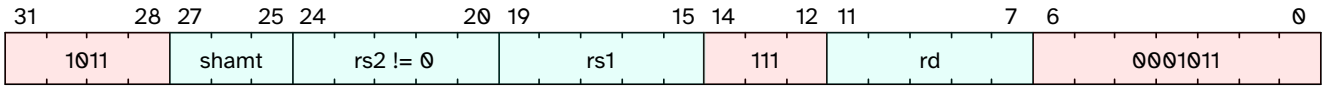
Synopsis

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`. Instruction encoded in R instruction format.

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, $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.100. qc.lrh

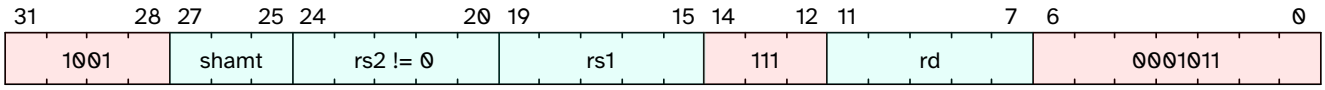
Synopsis

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. Instruction encoded in R instruction format.

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, $encoding), 16);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.101. qc.lrhu

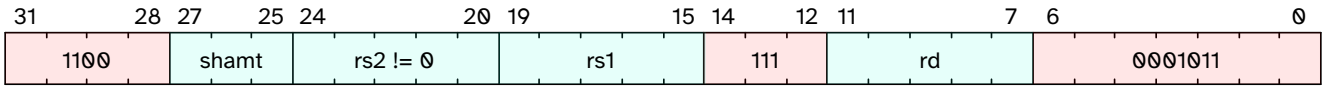
Synopsis

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`. Instruction encoded in R instruction format.

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, $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.102. qc.lrw

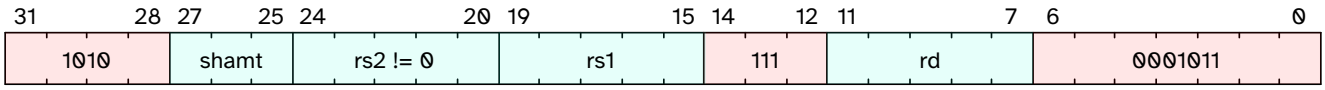
Synopsis

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`. Instruction encoded in R instruction format.

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, $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.103. qc.lwm

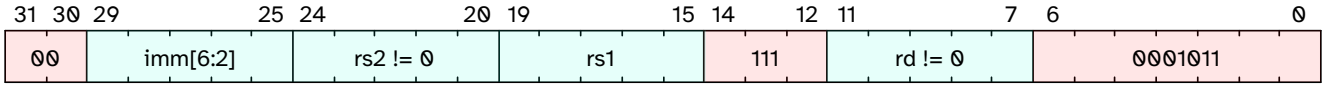
Synopsis

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`. Instruction encoded in R instruction format.

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, effective_ldst_mode(), $encoding) if
(rd + num_words) > 32;
for (U32 i = 0; i < num_words; i++) {
  X[rd + i] = read_memory<32>(vaddr, $encoding);
  vaddr = vaddr + 4;
}
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilsm	~> 0.1.0

6.104. qc.lwmi

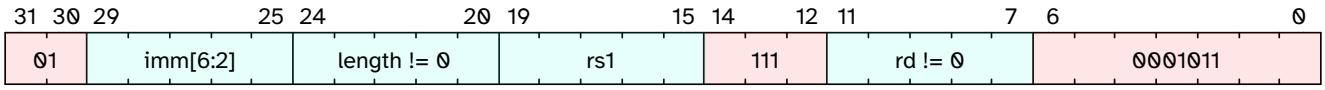
Synopsis

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. Instruction encoded in R instruction format.

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, effective_ldst_mode(), $encoding) if
(rd + length) > 32;
for (U32 i = 0; i < length; i++) {
  X[rd + i] = read_memory<32>(vaddr, $encoding);
  vaddr = vaddr + 4;
}
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilsm	~> 0.1.0

6.105. qc.muliadd

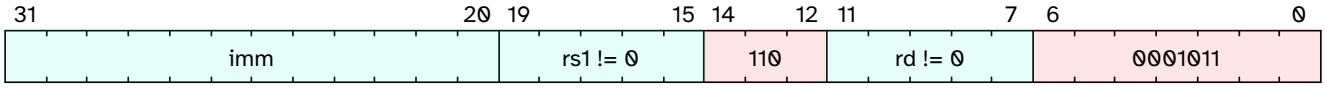
Synopsis

Multiply and accumulate (Immediate)

Mnemonic

```
qc.muliadd  rd, rs1, imm
```

Encoding



Description

Increments `rd` by the multiplication of `rs1` and a signed immediate `imm`. Instruction encoded in I instruction format.

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	Version
Xqci	~> 0.1.0
Xqciac	~> 0.1.0

6.106. qc.mveq

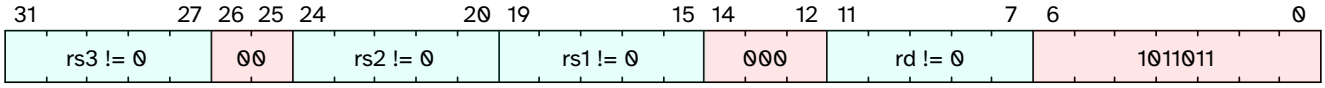
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.107. qc.mveqi

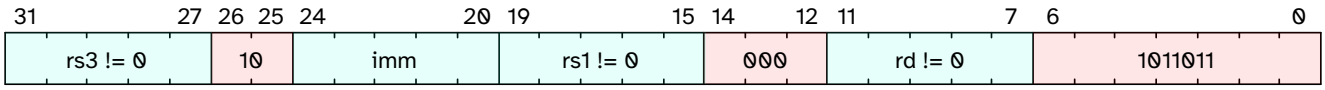
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.108. qc.mvge

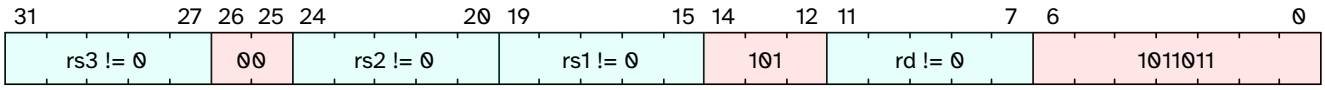
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.109. qc.mvgei

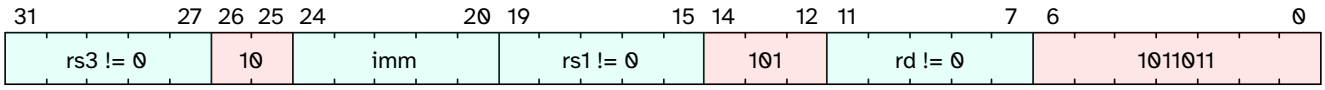
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.110. qc.mvgeu

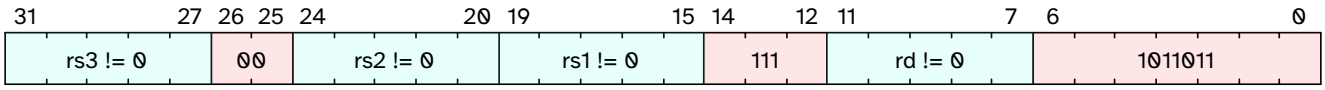
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.111. qc.mvgeui

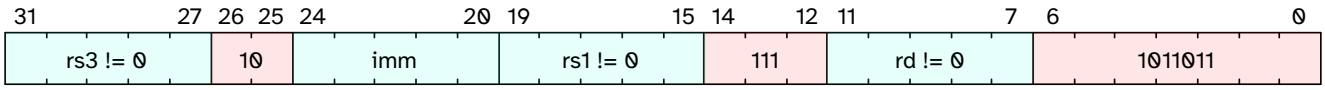
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.112. qc.mvlt

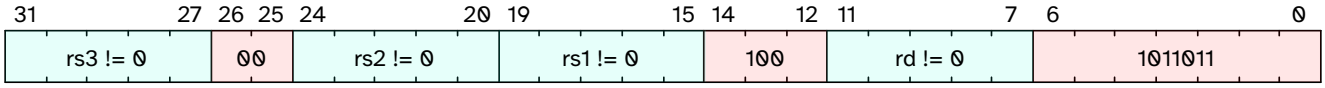
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.113. qc.mvlti

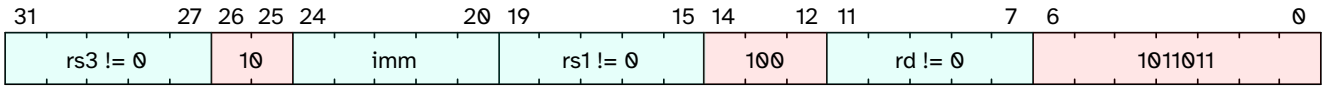
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.114. qc.mvltu

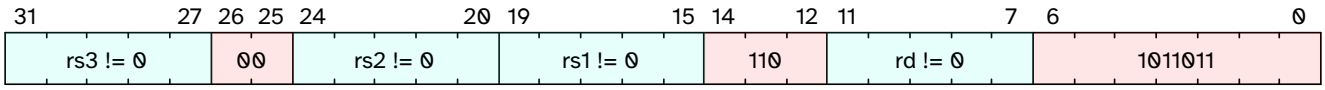
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.115. qc.mvltui

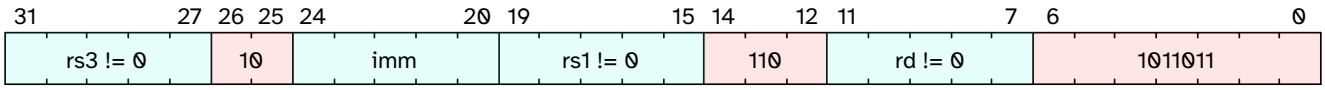
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.116. qc.mvne

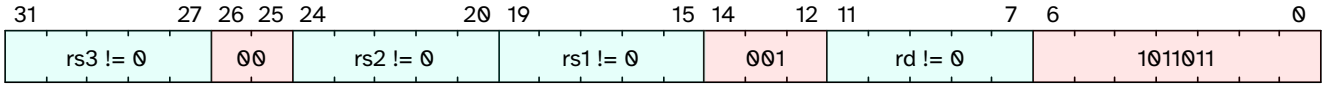
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.117. qc.mvnei

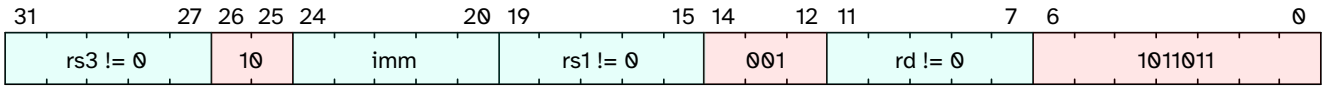
Synopsis

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`. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcicm	~> 0.1.0

6.118. qc.norm

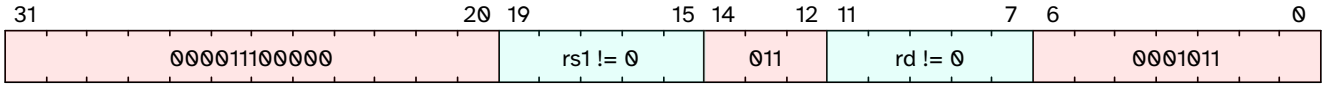
Synopsis

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`. Instruction encoded in I instruction format.

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 mnt = (X[rs1][31] == 1) ? (X[rs1] << (clo - 1)) : (X[rs1] << (clz - 1));
XReg exp = (X[rs1][31] == 1) ? (-(clo - 1)) : (-(clz - 1));
X[rd] = {mnt[23:0], exp[7:0]};
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.119. qc.normeu

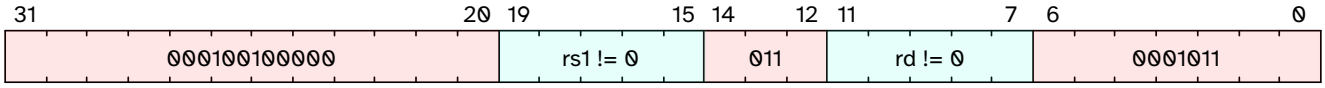
Synopsis

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`. Instruction encoded in I instruction format.

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 mnt = (X[rs1] << (clz & ~1));
XReg exp = ((-clz) & ~1);
X[rd] = {mnt[23:0], exp[7:0]};
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.120. qc.normu

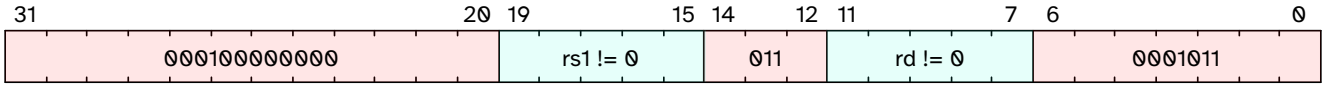
Synopsis

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`. Instruction encoded in I instruction format.

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 mnt = (X[rs1] << clz);
XReg exp = (-clz);
X[rd] = {mnt[23:0], exp[7:0]};
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.121. qc.outw

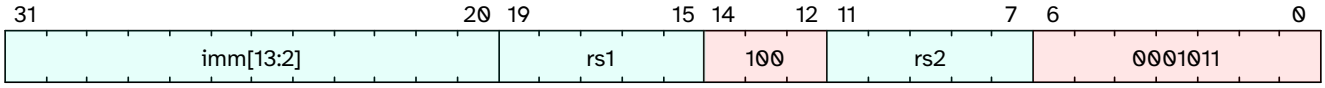
Synopsis

Output word to non-memory-mapped device

Mnemonic

```
qc.outw rs2, imm(rs1)
```

Encoding



Description

Output 32 bits of data from register `rs2` to a non-memory-mapped device. Such devices have own address space, unrelated to memory map. Device space address formed by adding `rs1` to to a unsigned offset `imm`. Instruction encoded in I instruction format.

Decode Variables

```
Bits<14> imm = {$encoding[31:20], 2'd0};
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[11:7];
```

Operation

```
XReg device_address = X[rs1] + imm;
write_device<32>(device_address, X[rs2]);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqciio	~> 0.1.0

6.122. qc.pcoredump

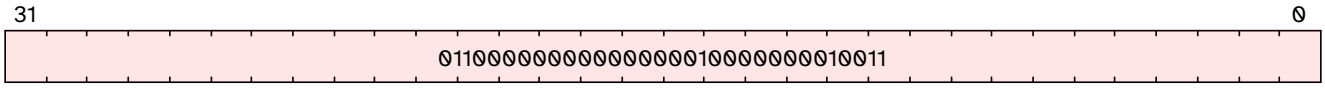
Synopsis

Print core dump pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.pcoredump
```

Encoding



Description

The print core dump instruction calls simulation environment with no explicit arguments. Implicit arguments are all general purpose registers and CSRs. Simulation environment expected to print the core dump on its console or standard output. The core dump format and content are defined by simulation environment. Instruction encoded in I instruction format.

Decode Variables

qc.pcoredump has no decode variables.

Operation

```
XReg func = 8;
XReg arg = 0;
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.123. qc.pexit

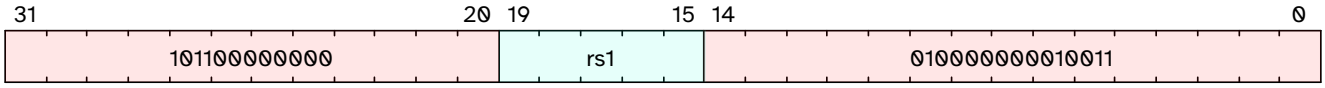
Synopsis

Exit call with register argument pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.pexit  rs1
```

Encoding



Description

The Exit call instruction calls simulation environment with unsigned `rs1` explicit argument. Simulation environment is expected to complete its execution and return to the system with exit code provided in `rs1`. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
XReg func = 12;
XReg arg = X[rs1];
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.124. qc.ppreg

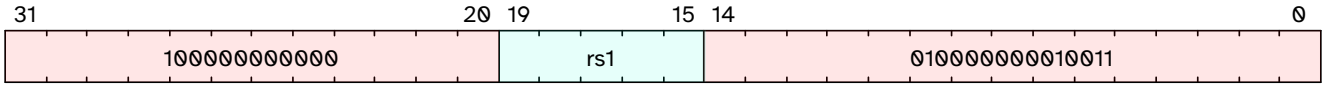
Synopsis

Print register pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.ppreg  rs1
```

Encoding



Description

The print register instruction calls simulation environment with `rs1` explicit argument. Simulation environment expected to print the register value on its console or standard output. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
XReg func = 2;
XReg arg = X[rs1];
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.125. qc.ppregs

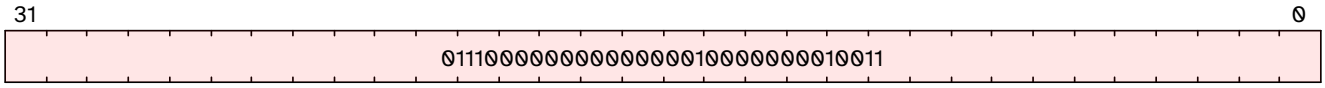
Synopsis

Print all registers pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.ppregs
```

Encoding



Description

The print registers instruction calls simulation environment with no explicit arguments. Implicit arguments are all general purpose registers. Simulation environment expected to print the all registers value on its console or standard output. Instruction encoded in I instruction format.

Decode Variables

qc.ppregs has no decode variables.

Operation

```
XReg func = 3;
XReg arg = 0;
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.126. qc.pputc

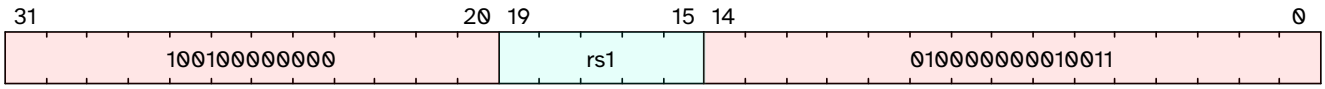
Synopsis

Print character passed in register argument pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.pputc  rs1
```

Encoding



Description

The print character instruction calls simulation environment with `rs1` explicit argument. Simulation environment expected to print the character on its console or standard output. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
XReg func = 4;
XReg arg = X[rs1];
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.127. qc.pputci

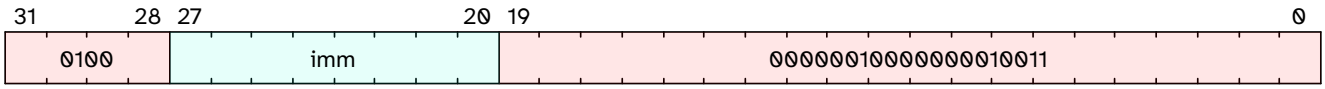
Synopsis

Print character passed in the immediate argument pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.pputci  imm
```

Encoding



Description

The print character instruction calls simulation environment with unsigned `imm` explicit argument. Simulation environment expected to print the 8-bit character on its console or standard output. Instruction encoded in I instruction format.

Decode Variables

```
Bits<8> imm = $encoding[27:20];
```

Operation

```
XReg func = 5;
XReg arg = imm;
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.128. qc.pputs

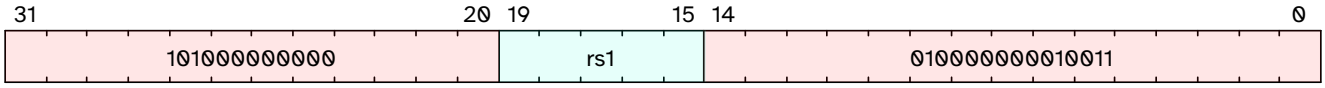
Synopsis

Print string pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.pputs  rs1
```

Encoding



Description

The print string instruction calls simulation environment with `rs1` explicit argument. The argument assumed to be pointer to the string in target simulated memory. Simulation environment expected to print the string on its console or standard output. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
XReg func = 6;
XReg arg = X[rs1];
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.129. qc.psyscall

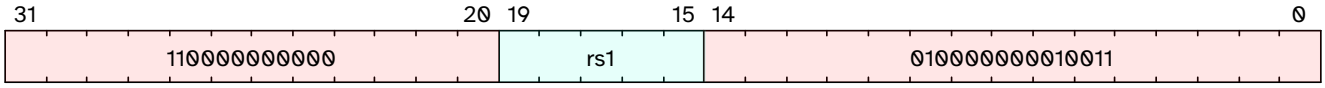
Synopsis

System call with register argument pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.psyscall  rs1
```

Encoding



Description

The System call instruction calls simulation environment with unsigned `rs1` explicit argument. Additional implicit arguments defined by simulation environment implementation. Instruction is used to call simulator to execute function according to the argument. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
XReg func = 10;
XReg arg = X[rs1];
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.130. qc.psyscalli

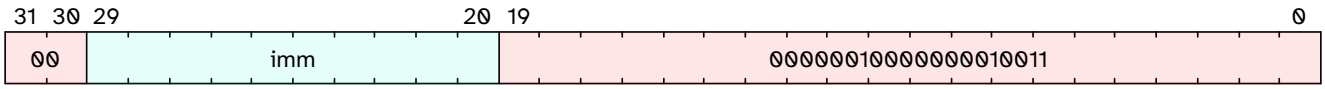
Synopsis

System call with immediate argument pseudo-instruction (hint) working only in simulation environment

Mnemonic

```
qc.psyscalli  imm
```

Encoding



Description

The System call instruction calls simulation environment with unsigned `imm` explicit argument. Additional implicit arguments defined by simulation environment implementation. Instruction is used to call simulator to execute function according to the argument. Instruction encoded in I instruction format.

Decode Variables

```
Bits<10> imm = $encoding[29:20];
```

Operation

```
XReg func = 11;
XReg arg = imm;
iss_syscall(func, arg);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisim	~> 0.1.0

6.131. qc.selecteqi

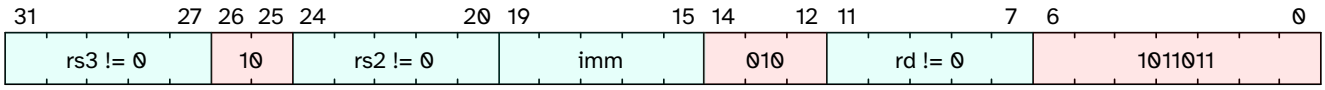
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.132. qc.selectieq

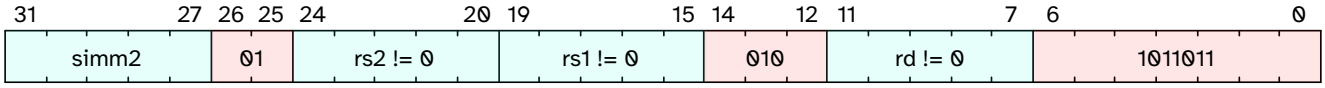
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.133. qc.selectieqi

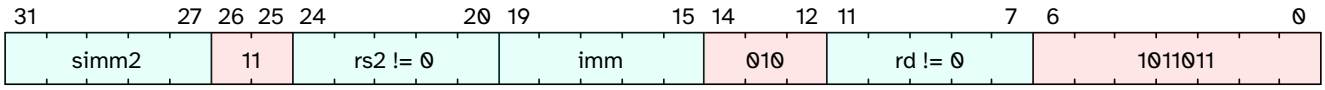
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.134. qc.selectiieq

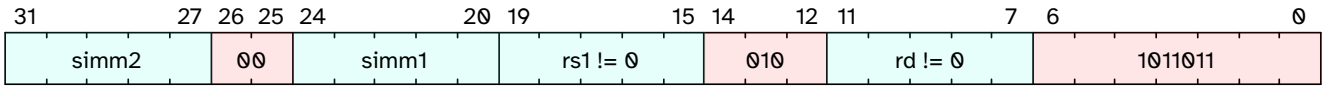
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.135. qc.selectiine

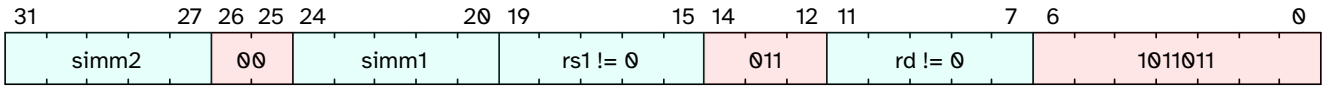
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.136. qc.selectine

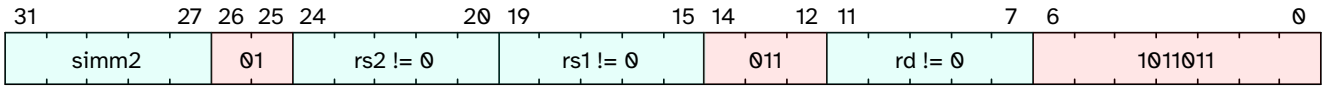
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.137. qc.selectinei

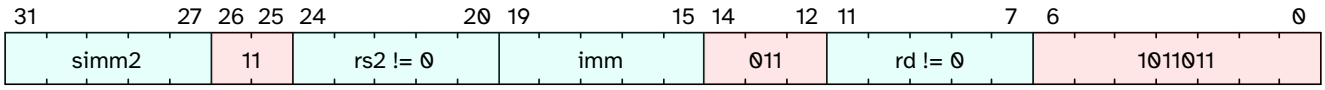
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.138. qc.selectnei

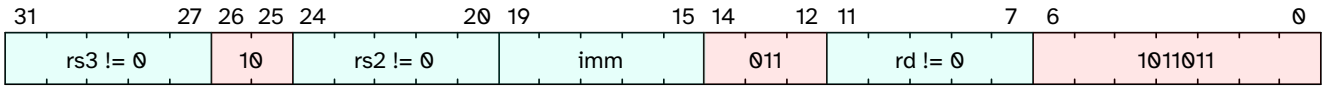
Synopsis

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. Instruction encoded in R4 instruction format.

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	Version
Xqci	~> 0.1.0
Xqcics	~> 0.1.0

6.139. qc.setinti

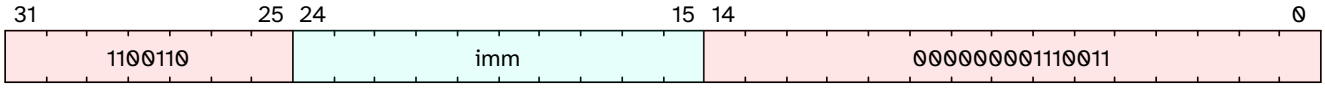
Synopsis

Set interrupt (Immediate)

Mnemonic

```
qc.setinti  imm
```

Encoding



Description

Set interrupt, interrupt number is in `imm` (0 - 1023). Instruction encoded in R instruction format.

Decode Variables

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

Operation

```
Bits<12> MCLICIP0_ADDR = CSR[qc_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	Version
Xqci	~> 0.1.0
Xqciint	~> 0.1.0

6.140. qc.setwm

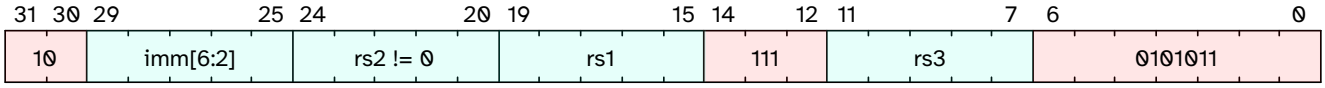
Synopsis

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`. Instruction encoded in R instruction format.

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, $encoding);
  vaddr = vaddr + 4;
}
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilsm	~> 0.1.0

6.141. qc.setwmi

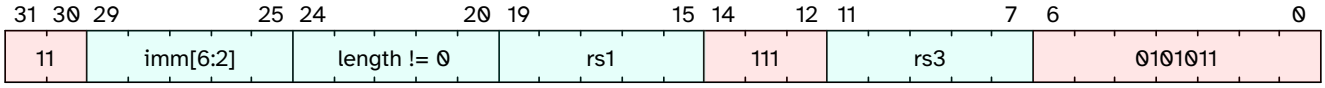
Synopsis

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. Instruction encoded in R instruction format.

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, $encoding);
  vaddr = vaddr + 4;
}
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilsm	~> 0.1.0

6.142. qc.shladd

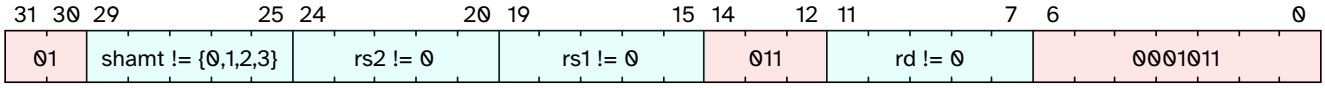
Synopsis

Shift left and add (immediate)

Mnemonic

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

Encoding



Description

Left shift *rs1* by *shamt* and add the value in *rs2*. Instruction encoded in R instruction format.

Decode Variables

```
Bits<5> shamt = $encoding[29:25];
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	Version
Xqci	>= 0.2
Xqciac	~> 0.1.0

6.143. qc.shlsat

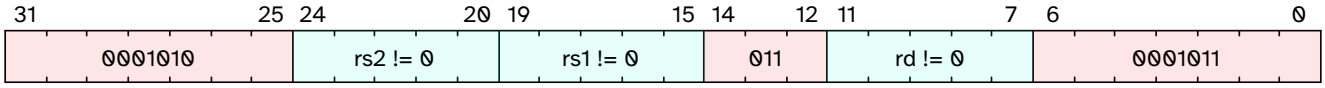
Synopsis

Saturating signed left shift

Mnemonic

```
qc.shlsat  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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.144. qc.shlusat

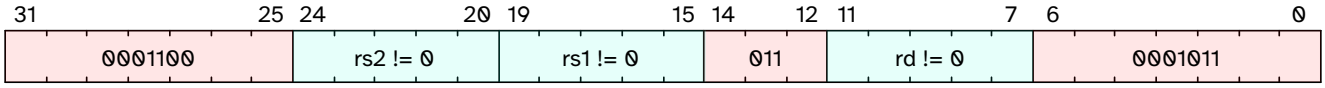
Synopsis

Saturating unsigned left shift

Mnemonic

```
qc.shlusat  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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.145. qc.srb

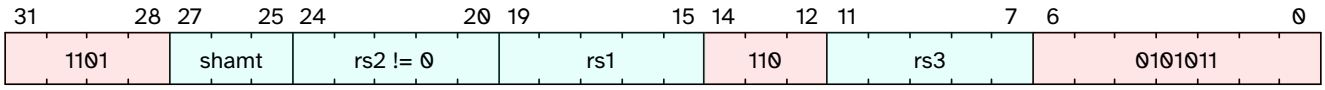
Synopsis

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`. Instruction encoded in R instruction format.

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], $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.146. qc.srh

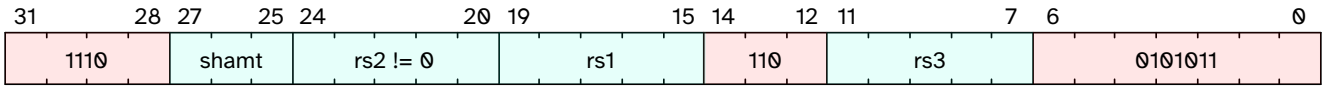
Synopsis

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`. Instruction encoded in R instruction format.

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], $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.147. qc.srw

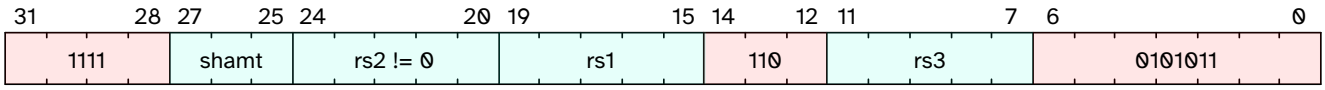
Synopsis

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`. Instruction encoded in R instruction format.

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], $encoding);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisls	~> 0.1.0

6.148. qc.subsat

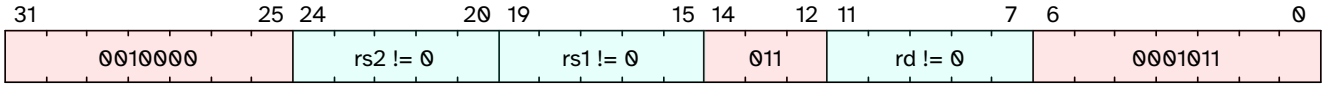
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.149. qc.subusat

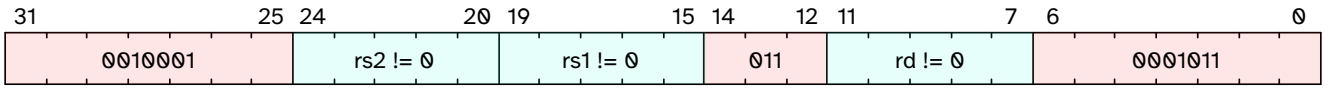
Synopsis

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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.150. qc.swm

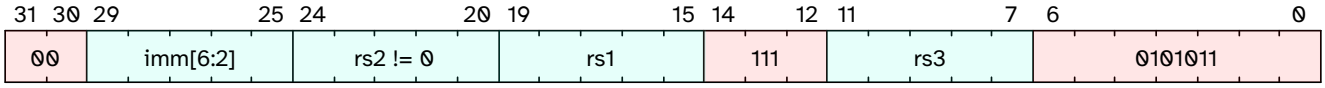
Synopsis

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`. Instruction encoded in R instruction format.

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, effective_ldst_mode(), $encoding) if
(rs3 + num_words) > 32;
for (U32 i = 0; i < num_words; i++) {
  write_memory<32>(vaddr, X[rs3 + i], $encoding);
  vaddr = vaddr + 4;
}
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilsm	~> 0.1.0

6.151. qc.swmi

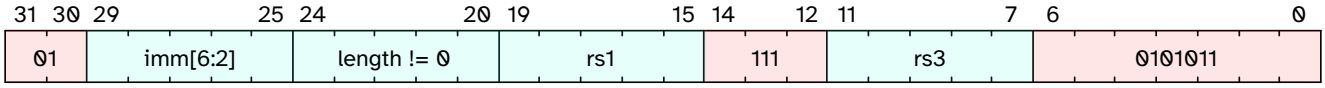
Synopsis

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. Instruction encoded in R instruction format.

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, effective_ldst_mode(), $encoding) if
(rs3 + length) > 32;
for (U32 i = 0; i < length; i++) {
  write_memory<32>(vaddr, X[rs3 + i], $encoding);
  vaddr = vaddr + 4;
}
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcilsm	~> 0.1.0

6.152. qc.sync

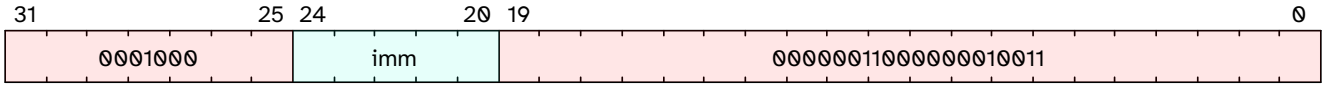
Synopsis

Non-memory-mapped device read-after-write synchronization to completion

Mnemonic

```
qc.sync  imm
```

Encoding



Description

This synchronization instruction delays execution till last read (input) from non-memory-mapped device transactions are completed for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 5-bit bitmask field that specifies up to five pre-defined devices. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
fence_tso();
sync_read_after_write_device(true, imm);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.153. qc.syncr

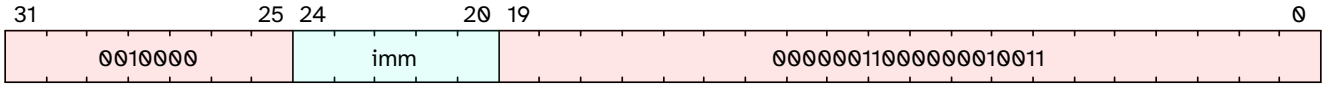
Synopsis

Non-memory-mapped device read-after-write synchronization

Mnemonic

```
qc.syncr  imm
```

Encoding



Description

This synchronization instruction delays execution till last read (input) from non-memory-mapped device transactions are started for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 5-bit bitmask field that specifies up to five pre-defined devices. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
fence_tso();
sync_read_after_write_device(false, imm);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.154. qc.syncwf

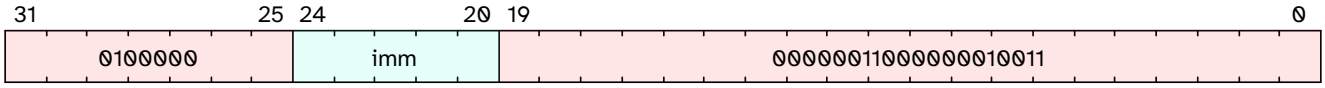
Synopsis

Non-memory-mapped device write-after-read synchronization

Mnemonic

```
qc.syncwf  imm
```

Encoding



Description

This synchronization instruction delays execution till last write (output) to non-memory-mapped device transactions are started for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 5-bit bitmask field that specifies up to five pre-defined devices. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
fence_tso();
sync_write_after_read_device(false, imm);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.155. qc.syncwl

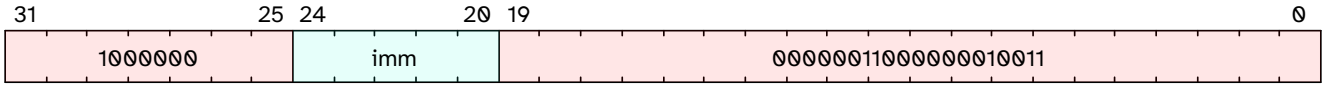
Synopsis

Non-memory-mapped device write-after-read synchronization to completion

Mnemonic

```
qc.syncwl  imm
```

Encoding



Description

This synchronization instruction delays execution till last write (output) to non-memory-mapped device transactions are completed for specified devices. As well, instruction implies fence.tso functionality for all memory accesses. Immediate argument is 5-bit bitmask field that specifies up to five pre-defined devices. Instruction encoded in I instruction format.

Decode Variables

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

Operation

```
fence_tso();
sync_write_after_read_device(true, imm);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcisync	~> 0.1.0

6.156. qc.wrap

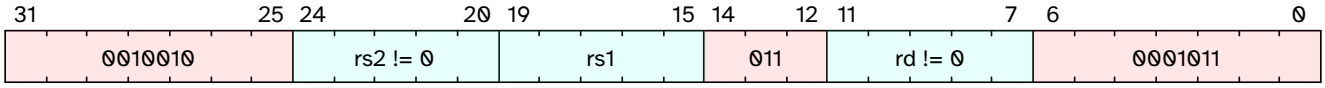
Synopsis

Wraparound (Register)

Mnemonic

```
qc.wrap  rd, rs1, rs2
```

Encoding



Description

If `rs1` >= `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`. Instruction encoded in R instruction format.

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	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

6.157. qc.wrapi

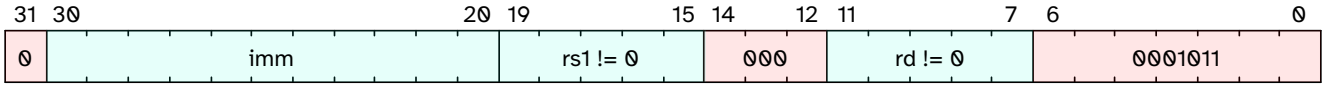
Synopsis

Wraparound (Unsigned Immediate)

Mnemonic

```
qc.wrapi  rd, rs1, imm
```

Encoding



Description

If `rs1` >= `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`. Instruction encoded in I instruction format. The `imm` is an unsigned immediate.

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] = ($signed(rs1_value) >= imm) ? rs1_value - imm : (($signed(rs1_value) <
0) ? ($signed(rs1_value) + imm) : rs1_value);
```

Included in

Extension	Version
Xqci	~> 0.1.0
Xqcia	~> 0.1.0

Chapter 7. IDL Functions

7.1. abort_current_instruction (builtin)

Abort the current instruction, and start refetching from $pc.

Return Type	`void`
Arguments	

7.2. 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> paddr, U32 access_size, XReg vaddr, MemoryOperation type, ExceptionCode fault_type, PrivilegeMode from_mode`

```
if (paddr > 1 << PHYS_ADDR_WIDTH) - access_size {
  raise(fault_type, from_mode, vaddr);
}
if (!pmp_check<access_size>(paddr[PHYS_ADDR_WIDTH - 1:0], type)) {
  raise(fault_type, from_mode, vaddr);
}
```

7.3. assert (builtin)

Assert that a condition is true. Failure represents an error in the IDL model.

Return Type	`void`
Arguments	`Boolean test, String message`

7.4. 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 will be aligned to 32-bits

Return Type	Boolean
Arguments	Bits<PHYS_ADDR_WIDTH> paddr, Bits<32> compare_value, Bits<32> write_value

7.5. 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 will be aligned to 64-bits

Return Type	Boolean
Arguments	Bits<PHYS_ADDR_WIDTH> paddr, Bits<64> compare_value, Bits<64> write_value

7.6. cached_translation (builtin)

Possibly returns a cached translation result matching vaddr.

If the result is value, the first return value will be true. paddr is the full translation of vaddr, including the page offset pbmt is the result of any paged-based attributes for the translation

Return Type	Boolean, TranslationResult
Arguments	XReg vaddr, MemoryOperation op

7.7. current_translation_mode

Returns the current first-stage translation mode for an explicit load or store from mode given the machine state (e.g., value of satp or vsatp csr).

Returns SatpMode::Reserved if the setting found in satp or vsatp is invalid.

Return Type	SatpMode
Arguments	PrivilegeMode mode

```
PrivilegeMode effective_mode = effective_ldst_mode();
```

```
if (effective_mode == PrivilegeMode::M) {
  return SatpMode::Bare;
}
if (CSR[misa].H == 1'b1) {
  if (effective_mode == PrivilegeMode::VS || effective_mode == PrivilegeMode
::VU) {
    Bits<4> mode_val = CSR[vsatp].MODE;
    if (mode_val == $bits(SatpMode::Sv32)) {
      if (XLEN == 64) {
        if ((effective_mode == PrivilegeMode::VS) && (CSR[hstatus].VSXL !=
$bits(XRegWidth::XLEN32))) {
          return SatpMode::Reserved;
        }
        if ((effective_mode == PrivilegeMode::VU) && (CSR[vsstatus].UXL !=
$bits(XRegWidth::XLEN32))) {
          return SatpMode::Reserved;
        }
      }
      if (!SV32_VSMODE_TRANSLATION) {
        return SatpMode::Reserved;
      }
      return SatpMode::Sv32;
    } else if ((XLEN == 64) && (mode_val == $bits(SatpMode::Sv39))) {
      if (effective_mode == PrivilegeMode::VS && CSR[hstatus].VSXL != $bits
(XRegWidth::XLEN64)) {
        return SatpMode::Reserved;
      }
      if (effective_mode == PrivilegeMode::VU && CSR[vsstatus].UXL != $bits
(XRegWidth::XLEN64)) {
        return SatpMode::Reserved;
      }
      if (!SV39_VSMODE_TRANSLATION) {
        return SatpMode::Reserved;
      }
      return SatpMode::Sv39;
    } else if ((XLEN == 64) && (mode_val == $bits(SatpMode::Sv48))) {
      if (effective_mode == PrivilegeMode::VS && CSR[hstatus].VSXL != $bits
(XRegWidth::XLEN64)) {
        return SatpMode::Reserved;
      }
      if (effective_mode == PrivilegeMode::VU && CSR[vsstatus].UXL != $bits
(XRegWidth::XLEN64)) {
        return SatpMode::Reserved;
      }
      if (!SV48_VSMODE_TRANSLATION) {
        return SatpMode::Reserved;
      }
      return SatpMode::Sv48;
    } else if ((XLEN == 64) && (mode_val == $bits(SatpMode::Sv57))) {
      if (effective_mode == PrivilegeMode::VS && CSR[hstatus].VSXL != $bits
(XRegWidth::XLEN64)) {
```

```
        return SatpMode::Reserved;
      }
      if (effective_mode == PrivilegeMode::VU && CSR[vsstatus].UXL != $bits
(XRegWidth::XLEN64)) {
        return SatpMode::Reserved;
      }
      if (!SV57_VSMODE_TRANSLATION) {
        return SatpMode::Reserved;
      }
      return SatpMode::Sv57;
    } else {
      return SatpMode::Reserved;
    }
  }
}
assert(effective_mode == PrivilegeMode::S || effective_mode == PrivilegeMode::
U, "unexpected priv mode");
Bits<4> mode_val = CSR[vsatp].MODE;
if (mode_val == $bits(SatpMode::Sv32)) {
  if (XLEN == 64) {
    if (effective_mode == PrivilegeMode::S && CSR[mstatus].SXL != $bits
(XRegWidth::XLEN32)) {
      return SatpMode::Reserved;
    }
    if (effective_mode == PrivilegeMode::U && CSR[sstatus].UXL != $bits
(XRegWidth::XLEN32)) {
      return SatpMode::Reserved;
    }
  }
  if (!implemented?(ExtensionName::Sv32)) {
    return SatpMode::Reserved;
  }
} else if ((XLEN == 64) && (mode_val == $bits(SatpMode::Sv39))) {
  if (effective_mode == PrivilegeMode::S && CSR[mstatus].SXL != $bits(
XRegWidth::XLEN64)) {
    return SatpMode::Reserved;
  }
  if (effective_mode == PrivilegeMode::U && CSR[sstatus].UXL != $bits(
XRegWidth::XLEN64)) {
    return SatpMode::Reserved;
  }
  if (!implemented?(ExtensionName::Sv39)) {
    return SatpMode::Reserved;
  }
  return SatpMode::Sv39;
} else if ((XLEN == 64) && (mode_val == $bits(SatpMode::Sv48))) {
  if (effective_mode == PrivilegeMode::S && CSR[mstatus].SXL != $bits(
XRegWidth::XLEN64)) {
    return SatpMode::Reserved;
  }
  if (effective_mode == PrivilegeMode::U && CSR[sstatus].UXL != $bits(
```

```
XRegWidth::XLEN64)) {
    return SatpMode::Reserved;
  }
  if (!implemented?(ExtensionName::Sv48)) {
    return SatpMode::Reserved;
  }
  return SatpMode::Sv48;
} else if ((XLEN == 64) && (mode_val == $bits(SatpMode::Sv57))) {
  if (effective_mode == PrivilegeMode::S && CSR[mstatus].SXL != $bits(
XRegWidth::XLEN64)) {
    return SatpMode::Reserved;
  }
  if (effective_mode == PrivilegeMode::U && CSR[sstatus].UXL != $bits(
XRegWidth::XLEN64)) {
    return SatpMode::Reserved;
  }
  if (!implemented?(ExtensionName::Sv57)) {
    return SatpMode::Reserved;
  }
  return SatpMode::Sv57;
} else {
  return SatpMode::Reserved;
}
```

7.8. delay (builtin)

Delay the processor by cycles cycles.

Return Type	void
Arguments	XReg cycles

7.9. 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 (CSR[misa].U == 1 && CSR[mstatus].MPRV == 1) {
    if (CSR[mstatus].MPP == 0b00) {
      if (CSR[misa].H == 1 && mpv() == 0b1) {
        return PrivilegeMode::VU;
      } else {
        return PrivilegeMode::U;
      }
```

```
    } else if (CSR[misa].S == 1 && CSR[mstatus].MPP == 0b01) {
      if (CSR[misa].H == 1 && mpv() == 0b1) {
        return PrivilegeMode::VS;
      } else {
        return PrivilegeMode::S;
      }
    }
  }
}
return mode();
```

7.10. 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 (($bits(CSR[medeleg]) & (1 << $bits(exception_code))) != 0) {
    return PrivilegeMode::HS;
  } else {
    return PrivilegeMode::M;
  }
} else {
  assert(implemented?(ExtensionName::H) && mode() == PrivilegeMode::VS) ||
(mode() == PrivilegeMode::VU, "Unexpected mode");
  if (($bits(CSR[medeleg]) & (1 << $bits(exception_code))) != 0) {
    if (($bits(CSR[hedeleg]) & (1 << $bits(exception_code))) != 0) {
      return PrivilegeMode::VS;
    } else {
      return PrivilegeMode::HS;
    }
  } else {
    return PrivilegeMode::M;
  }
}
```

7.11. fence_tso (builtin)

Execute a TSO memory ordering fence.(according to the FENCE instruction).

Return Type	void

Arguments	

7.12. gstage_page_walk

Translate guest physical address to physical address through a page walk.

May raise a Guest 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	TranslationResult
Arguments	XReg gpaddr, XReg vaddr, MemoryOperation op, PrivilegeMode effective_mode, Boolean for_final_vs_pte, Bits<INSTR_ENC_SIZE> encoding

```
Bits<PA_SIZE> ppn;
TranslationResult result;
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
::LoadGuestPageFault : (op == MemoryOperation::Fetch ? ExceptionCode
::InstructionGuestPageFault : ExceptionCode::StoreAmoGuestPageFault);
Boolean mxr = for_final_vs_pte && (CSR[mstatus].MXR == 1);
Boolean pbmte = CSR[menvcfg].PBMTE == 1;
Boolean adue = CSR[menvcfg].ADUE == 1;
Bits<32> tinst = tinst_value_for_guest_page_fault(op, encoding,
for_final_vs_pte);
U32 max_gpa_width = LEVELS * VPN_SIZE + 2 + 12;
if (gpaddr >> max_gpa_width != 0) {
  raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
}
ppn = CSR[hgatp].PPN;
for (U32 i = (LEVELS - 1); i >= 0; i--) {
  U32 this_vpn_size = (i == (LEVELS - 1)) ? VPN_SIZE + 2 : VPN_SIZE;
  U32 vpn = (gpaddr >> (12 + VPN_SIZE * i)) & 1 << this_vpn_size) - 1);   Bits
<PA_SIZE> pte_paddr = (ppn << 12) + (vpn * (PTESIZE / 8;
  if (!pma_applies?(PmaAttribute::HardwarePageTableRead, pte_paddr, PTESIZE)) {
    raise(access_fault_code, PrivilegeMode::U, vaddr);
  }
  access_check(pte_paddr, PTESIZE, vaddr, MemoryOperation::Read,
access_fault_code, effective_mode);
  Bits<PTESIZE> pte = read_physical_memory<PTESIZE>(pte_paddr);
```

```
  PteFlags pte_flags = pte[9:0];
  if ((VA_SIZE != 32) && (pte[60:54] != 0)) {
    raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
  }
  if (!implemented?(ExtensionName::Svnapot)) {
    if ((PTESIZE >= 64) && pte[63] != 0) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    }
  }
  if ((PTESIZE >= 64) && !pbmte && (pte[62:61] != 0)) {
    raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
  }
  if ((PTESIZE >= 64) && pbmte && (pte[62:61] == 3)) {
    raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
  }
  if (pte_flags.V == 0) {
    raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
  }
  if (pte_flags.R == 0 && pte_flags.W == 1) {
    raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
  }
  if (pte_flags.R == 1 || pte_flags.X == 1) {
    if (pte_flags.U == 0) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    }
    if (op == MemoryOperation::Write) || (op == MemoryOperation
::ReadModifyWrite && (pte_flags.W == 0)) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    } else if ((op == MemoryOperation::Fetch) && (pte_flags.X == 0)) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    } else if ((op == MemoryOperation::Read) || (op == MemoryOperation
::ReadModifyWrite)) {
      if (!mxr) && (pte_flags.R == 0 || mxr) && (pte_flags.X == 0 && pte_flags
.R == 0) {
        raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
      }
    }
    if ((i > 0) && (pte[(i - 1) * VPN_SIZE:10] != 0)) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    }
    if ((pte_flags.A == 0) || pte_flags.D == 0) && ((op == MemoryOperation
::Write) || (op == MemoryOperation::ReadModifyWrite)) {
      if (adue) {
        if (!pma_applies?(PmaAttribute::RsrvEventual, pte_paddr, PTESIZE)) {
          raise(access_fault_code, PrivilegeMode::U, vaddr);
        }
        if (!pma_applies?(PmaAttribute::HardwarePageTableWrite, pte_paddr,
PTESIZE)) {
          raise(access_fault_code, PrivilegeMode::U, vaddr);
        }
        access_check(pte_paddr, PTESIZE, vaddr, MemoryOperation::Write,
```

```
access_fault_code, effective_mode);
        Boolean success;
        Bits<PTESIZE> updated_pte;
        if (pte_flags.D == 0 && (op == MemoryOperation::Write || op ==
MemoryOperation::ReadModifyWrite)) {
          updated_pte = pte | 0b11000000;
        } else {
          updated_pte = pte | 0b01000000;
        }
        if (PTESIZE == 32) {
          success = atomic_check_then_write_32(pte_paddr, pte, updated_pte);
        } else if (PTESIZE == 64) {
          success = atomic_check_then_write_64(pte_paddr, pte, updated_pte);
        } else {
          assert(false, "Unexpected PTESIZE");
        }
        if (!success) {
          i = i + 1;
        } else {
          result.paddr = pte_paddr;
          if (PTESIZE >= 64) {
            result.pbmt = pte[62:61];
          }
          result.pte_flags = pte_flags;
          return result;
        }
      } else {
        raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
      }
    }
  } else {
    if (i == 0) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    }
    if (pte_flags.D == 1 || pte_flags.A == 1 || pte_flags.U == 1) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    }
    if ((VA_SIZE != 32) && (pte[62:61] != 0)) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    }
    if ((VA_SIZE != 32) && pte[63] != 0) {
      raise_guest_page_fault(op, gpaddr, vaddr, tinst, effective_mode);
    }
    ppn = pte[PA_SIZE - 3:10] << 12;
  }
}
```

7.13. 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;
```

7.14. implemented? (builtin)

Return true if the implementation supports `extension`.

Return Type	Boolean
Arguments	ExtensionName extension

7.15. is_naturally_aligned

Checks if value is naturally aligned to N bits.

Return Type	Boolean
Arguments	XReg value

```
return true if (N == 8);
XReg Mask = (N / 8) - 1;
return (value & ~Mask) == value;
```

7.16. iss_syscall (builtin)

Instruction set simulator system call.

Return Type	void
Arguments	XReg id, XReg arg

7.17. 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, mode(), target_hw_addr);
  }
}
$pc = target_hw_addr;
```

7.18. 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();
```

7.19. maybe_cache_translation (builtin)

Given a translation result, potentially cache the result for later use. This function models a TLB fill operation. A valid implementation does nothing.

Return Type	void
Arguments	XReg vaddr, MemoryOperation op, TranslationResult result

7.20. 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 (MISALIGNED_MAX_ATOMICITY_GRANULE_SIZE == 0);
if (pma_applies?(PmaAttribute::MAG16, physical_address, N) &&
in_naturally_aligned_region?<128>(physical_address, N)) {
  return true;
} else if (pma_applies?(PmaAttribute::MAG8, physical_address, N) &&
in_naturally_aligned_region?<4>(physical_address, N)) {
  return true;
} else if (pma_applies?(PmaAttribute::MAG4, physical_address, N) &&
in_naturally_aligned_region?<32>(physical_address, N)) {
  return true;
} else if (pma_applies?(PmaAttribute::MAG2, physical_address, N) &&
in_naturally_aligned_region?<16>(physical_address, N)) {
  return true;
} else {
  return false;
}
```

7.21. 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;
}
```

7.22. 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 if (exception_code == ExceptionCode::SoftwareCheck) {
  return tval;
} else {
  return 0;
}
```

7.23. 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`

7.24. raise

Raise synchronous exception number `exception_code`.

The exception may be imprecise, and will cause execution to enter an unpredictable state, if PRECISE_SYNCHRONOUS_EXCEPTIONS is false.

Otherwise, the exception will be precise.

Return Type	`void`
Arguments	`ExceptionCode exception_code, PrivilegeMode from_mode, XReg tval`

```
if (!PRECISE_SYNCHRONOUS_EXCEPTIONS) {
  unpredictable("Imprecise synchronous exception");
} else {
  raise_precise(exception_code, from_mode, tval);
```

```
}
```

7.25. raise_guest_page_fault

Raise a guest page fault exception.

Return Type	void
Arguments	MemoryOperation op, XReg gpa, XReg gva, XReg tinst_value, PrivilegeMode from_mode

```
ExceptionCode code;
Boolean write_gpa_in_tval;
if (op == MemoryOperation::Read) {
  code = ExceptionCode::LoadGuestPageFault;
  write_gpa_in_tval = REPORT_GPA_IN_TVAL_ON_LOAD_GUEST_PAGE_FAULT;
} else if (op == MemoryOperation::Write || op == MemoryOperation
::ReadModifyWrite) {
  code = ExceptionCode::StoreAmoGuestPageFault;
  write_gpa_in_tval = REPORT_GPA_IN_TVAL_ON_STORE_AMO_GUEST_PAGE_FAULT;
} else {
  assert(op == MemoryOperation::Fetch, "unexpected memory operation");
  code = ExceptionCode::InstructionGuestPageFault;
  write_gpa_in_tval = REPORT_GPA_IN_TVAL_ON_INSTRUCTION_GUEST_PAGE_FAULT;
}
PrivilegeMode handling_mode = exception_handling_mode(code);
if (handling_mode == PrivilegeMode::S) {
  CSR[htval].VALUE = write_gpa_in_tval ? (gpa >> 2) : 0;
  CSR[htinst].VALUE = tinst_value;
  CSR[sepc].PC = $pc;
  if (!stval_readonly?()) {
    CSR[stval].VALUE = stval_for(code, gva);
  }
  $pc = {CSR[stvec].BASE, 2'b00};
  CSR[scause].INT = 1'b0;
  CSR[scause].CODE = $bits(code);
  CSR[hstatus].GVA = 1;
  CSR[hstatus].SPV = 1;
  CSR[hstatus].SPVP = $bits(from_mode)[0];
  CSR[mstatus].SPP = $bits(from_mode)[0];
} else {
  assert(handling_mode == PrivilegeMode::M, "unexpected privilege mode");
  CSR[mtval2].VALUE = write_gpa_in_tval ? (gpa >> 2) : 0;
  CSR[mtinst].VALUE = tinst_value;
  CSR[mstatus].MPP = $bits(from_mode)[1:0];
  if (XLEN == 64) {
    CSR[mstatus].MPV = 1;
  } else {
    CSR[mstatush].MPV = 1;
```

```
  }
}
set_mode(handling_mode);
abort_current_instruction();
```

7.26. raise_precise

Raise synchronous exception number `exception_code`.

Return Type	void
Arguments	ExceptionCode exception_code, PrivilegeMode from_mode, 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].CODE = $bits(exception_code);
  if (CSR[misa].H == 1) {
    CSR[mtval2].VALUE = 0;
    CSR[mtinst].VALUE = 0;
    if (from_mode == PrivilegeMode::VU || from_mode == PrivilegeMode::VS) {
      if (XLEN == 32) {
        CSR[mstatush].MPV = 1;
      } else {
        CSR[mstatus].MPV = 1;
      }
    } else {
      if (XLEN == 32) {
        CSR[mstatush].MPV = 0;
      } else {
        CSR[mstatus].MPV = 0;
      }
    }
  }
  CSR[mstatus].MPP = $bits(from_mode);
} else if (CSR[misa].S == 1 && (handling_mode == PrivilegeMode::S)) {
  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);
```

```
  CSR[mstatus].SPP = $bits(from_mode)[0];
  if (CSR[misa].H == 1) {
    CSR[htval].VALUE = 0;
    CSR[htinst].VALUE = 0;
    CSR[hstatus].SPV = $bits(from_mode)[2];
    if (from_mode == PrivilegeMode::VU || from_mode == PrivilegeMode::VS) {
      CSR[hstatus].SPV = 1;
      if (exception_code == ExceptionCode::Breakpoint) && (
REPORT_VA_IN_STVAL_ON_BREAKPOINT || exception_code == ExceptionCode
::LoadAddressMisaligned) && (REPORT_VA_IN_STVAL_ON_LOAD_MISALIGNED ||
exception_code == ExceptionCode::StoreAmoAddressMisaligned) && (
REPORT_VA_IN_STVAL_ON_STORE_AMO_MISALIGNED || exception_code == ExceptionCode
::InstructionAddressMisaligned) && (
REPORT_VA_IN_STVAL_ON_INSTRUCTION_MISALIGNED || exception_code ==
ExceptionCode::LoadAccessFault) && (REPORT_VA_IN_STVAL_ON_LOAD_ACCESS_FAULT ||
exception_code == ExceptionCode::StoreAmoAccessFault) && (
REPORT_VA_IN_STVAL_ON_STORE_AMO_ACCESS_FAULT || exception_code ==
ExceptionCode::InstructionAccessFault) && (
REPORT_VA_IN_STVAL_ON_INSTRUCTION_ACCESS_FAULT || exception_code ==
ExceptionCode::LoadPageFault) && (REPORT_VA_IN_STVAL_ON_LOAD_PAGE_FAULT ||
exception_code == ExceptionCode::StoreAmoPageFault) && (
REPORT_VA_IN_STVAL_ON_STORE_AMO_PAGE_FAULT || exception_code == ExceptionCode
::InstructionPageFault) && (REPORT_VA_IN_STVAL_ON_INSTRUCTION_PAGE_FAULT) {
        CSR[hstatus].GVA = 1;
      } else {
        CSR[hstatus].GVA = 0;
      }
      CSR[hstatus].SPVP = $bits(from_mode)[0];
    } else {
      CSR[hstatus].SPV = 0;
      CSR[hstatus].GVA = 0;
    }
  }
} else if (CSR[misa].H == 1 && (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);
  CSR[vsstatus].SPP = $bits(from_mode)[0];
}
set_mode(handling_mode);
abort_current_instruction();
```

7.27. read_device (builtin)

Read from non-memory-mapped device. Such devices have own addresses not related to memory

map.

Return Type	Bits<len>
Arguments	XReg dev_addr

7.28. read_memory

Read from virtual memory.

Return Type	Bits<LEN>
Arguments	XReg virtual_address, Bits<INSTR_ENC_SIZE> encoding

```
Boolean aligned = is_naturally_aligned<LEN>(virtual_address);
XReg physical_address;
if (aligned) {
  return read_memory_aligned<LEN>(virtual_address, encoding);
}
if (MISALIGNED_MAX_ATOMICITY_GRANULE_SIZE > 0) {
  assert(MISALIGNED_LDST_EXCEPTION_PRIORITY == "low", "Invalid config: can't
mix low-priority misaligned exceptions with large atomicity granule");
  physical_address = (CSR[misa].S == 1) ? translate(virtual_address,
MemoryOperation::Read, effective_ldst_mode(), encoding).paddr :
virtual_address;
  if (misaligned_is_atomic?<LEN>(physical_address)) {
    access_check(physical_address, LEN, virtual_address, MemoryOperation::Read,
ExceptionCode::LoadAccessFault, effective_ldst_mode());
    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, effective_ldst_mode(), encoding).paddr :
virtual_address;
    access_check(physical_address, LEN, virtual_address, MemoryOperation::Read,
ExceptionCode::LoadAccessFault, effective_ldst_mode());
  }
  raise(ExceptionCode::LoadAddressMisaligned, effective_ldst_mode(),
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, encoding)
<< (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");
  }
}
```

7.29. read_memory_aligned

Read from virtual memory using a known aligned address.

Return Type	Bits<LEN>
Arguments	XReg virtual_address, Bits<INSTR_ENC_SIZE> encoding

```
TranslationResult result;
if (CSR[misa].S == 1) {
  result = translate(virtual_address, MemoryOperation::Read,
effective_ldst_mode(), encoding);
} else {
  result.paddr = virtual_address;
}
access_check(result.paddr, LEN, virtual_address, MemoryOperation::Read,
ExceptionCode::LoadAccessFault, effective_ldst_mode());
return read_physical_memory<LEN>(result.paddr);
```

7.30. read_physical_memory (builtin)

Read from physical memory.

Return Type	Bits<len>
Arguments	XReg paddr

7.31. set_mode

Set the current privilege mode to `new_mode`

Return Type	void
Arguments	PrivilegeMode new_mode

```
if (new_mode != current_mode) {
  notify_mode_change(new_mode, current_mode);
  current_mode = new_mode;
}
```

7.32. sext

Sign extend `value` starting at `first_extended_bit`.

Bits [`XLEN-1`:`first_extended_bit`] of the return value should get the value of bit (`first_extended bit - 1`).

Return Type	`XReg`
Arguments	`XReg value, XReg first_extended_bit`

```
if (first_extended_bit == XLEN) {
  return value;
} else {
  Bits<1> sign = value[first_extended_bit - 1];
  for (U32 i = XLEN - 1; i >= first_extended_bit; i--) {
    value[i] = sign;
  }
  return value;
}
```

7.33. stage1_page_walk

Translate virtual address to physical 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 guest physical address.

Return Type	`TranslationResult`
Arguments	`Bits<XLEN> vaddr, MemoryOperation op, PrivilegeMode effective_mode, Bits<INSTR_ENC_SIZE> encoding`

```
Bits<PA_SIZE> ppn;
TranslationResult result;
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);
Boolean sse = false;
Boolean adue;
```

```
if (CSR[misa].H == 1 && (effective_mode == PrivilegeMode::VS || effective_mode
== PrivilegeMode::VU)) {
  adue = CSR[henvcfg].ADUE == 1;
} else {
  adue = CSR[menvcfg].ADUE == 1;
}
Boolean pbmte;
if (VA_SIZE == 32) {
  pbmte = false;
} else {
  if (CSR[misa].H == 1 && (effective_mode == PrivilegeMode::VS ||
effective_mode == PrivilegeMode::VU)) {
    pbmte = CSR[henvcfg].PBMTE == 1;
  } else {
    pbmte = CSR[menvcfg].PBMTE == 1;
  }
}
Boolean mxr;
if (CSR[misa].H == 1 && (effective_mode == PrivilegeMode::VS || effective_mode
== PrivilegeMode::VU)) {
  mxr = (CSR[mstatus].MXR == 1) || (CSR[vsstatus].MXR == 1);
} else {
  mxr = CSR[mstatus].MXR == 1;
}
Boolean sum;
if (CSR[misa].H == 1 && (effective_mode == PrivilegeMode::VS)) {
  sum = CSR[vsstatus].SUM == 1;
} else {
  sum = CSR[mstatus].SUM == 1;
}
ppn = CSR[vsatp].PPN;
if ((VA_SIZE < xlen()) && (vaddr[xlen() - 1:VA_SIZE] != {xlen() - VA_SIZE{
vaddr[VA_SIZE - 1]}})) {
  raise(page_fault_code, effective_mode, vaddr);
}
for (U32 i = (LEVELS - 1); i >= 0; i--) {
  U32 vpn = (vaddr >> (12 + VPN_SIZE * i)) & 1 << VPN_SIZE) - 1);   Bits<
PA_SIZE> pte_gpaddr = (ppn << 12) + (vpn * (PTESIZE / 8;
  TranslationResult pte_phys = translate_gstage(pte_gpaddr, vaddr,
MemoryOperation::Read, effective_mode, encoding);
  if (!pma_applies?(PmaAttribute::HardwarePageTableRead, pte_phys.paddr,
PTESIZE)) {
    raise(access_fault_code, effective_mode, vaddr);
  }
  access_check(pte_phys.paddr, PTESIZE, vaddr, MemoryOperation::Read,
access_fault_code, effective_mode);
  Bits<PTESIZE> pte = read_physical_memory<PTESIZE>(pte_phys.paddr);
  PteFlags pte_flags = pte[9:0];
  Boolean ss_page = (pte_flags.R == 0) && (pte_flags.W == 1) && (pte_flags.X ==
0);
  if ((VA_SIZE != 32) && (pte[60:54] != 0)) {
```

```
    raise(page_fault_code, effective_mode, vaddr);
  }
  if (pte_flags.V == 0) {
    raise(page_fault_code, effective_mode, vaddr);
  }
  if (!sse) {
    if ((pte_flags.R == 0) && (pte_flags.W == 1)) {
      raise(page_fault_code, effective_mode, vaddr);
    }
  }
  if (pbmte) {
    if (pte[62:61] == 3) {
      raise(page_fault_code, effective_mode, vaddr);
    }
  } else {
    if ((PTESIZE >= 64) && (pte[62:61] != 0)) {
      raise(page_fault_code, effective_mode, vaddr);
    }
  }
  if (!implemented?(ExtensionName::Svnapot)) {
    if ((PTESIZE >= 64) && (pte[63] != 0)) {
      raise(page_fault_code, effective_mode, vaddr);
    }
  }
  if (pte_flags.R == 1 || pte_flags.X == 1) {
    if (op == MemoryOperation::Read || op == MemoryOperation::ReadModifyWrite)
{
      if (!mxr) && (pte_flags.R == 0 || mxr) && (pte_flags.X == 0 && pte_flags
.R == 0) {
        raise(page_fault_code, effective_mode, vaddr);
      }
      if (effective_mode == PrivilegeMode::U && pte_flags.U == 0) {
        raise(page_fault_code, effective_mode, vaddr);
      } else if (CSR[misa].H == 1 && effective_mode == PrivilegeMode::VU &&
pte_flags.U == 0) {
        raise(page_fault_code, effective_mode, vaddr);
      } else if (effective_mode == PrivilegeMode::S && pte_flags.U == 1 && !
sum) {
        raise(page_fault_code, effective_mode, vaddr);
      } else if (effective_mode == PrivilegeMode::VS && pte_flags.U == 1 &&
!sum) {
        raise(page_fault_code, effective_mode, vaddr);
      }
    }
    if (op == MemoryOperation::Write) || (op == MemoryOperation
::ReadModifyWrite && (pte_flags.W == 0)) {
      raise(page_fault_code, effective_mode, vaddr);
    } else if ((op == MemoryOperation::Fetch) && (pte_flags.X == 0)) {
      raise(page_fault_code, effective_mode, vaddr);
    } else if ((op == MemoryOperation::Fetch) && ss_page) {
      raise(page_fault_code, effective_mode, vaddr);
```

```
    }
    raise(page_fault_code, effective_mode, vaddr) if ;
    if ((pte_flags.A == 0) || pte_flags.D == 0) && ((op == MemoryOperation
::Write) || (op == MemoryOperation::ReadModifyWrite)) {
      if (adue) {
        TranslationResult pte_phys = translate_gstage(pte_gpaddr, vaddr,
MemoryOperation::Write, effective_mode, encoding);
        if (!pma_applies?(PmaAttribute::RsrvEventual, pte_phys.paddr, PTESIZE))
{
          raise(access_fault_code, effective_mode, vaddr);
        }
        if (!pma_applies?(PmaAttribute::HardwarePageTableWrite, pte_phys.paddr,
PTESIZE)) {
          raise(access_fault_code, effective_mode, vaddr);
        }
        access_check(pte_phys.paddr, PTESIZE, vaddr, MemoryOperation::Write,
access_fault_code, effective_mode);
        Boolean success;
        Bits<PTESIZE> updated_pte;
        if (pte_flags.D == 0 && (op == MemoryOperation::Write || op ==
MemoryOperation::ReadModifyWrite)) {
          updated_pte = pte | 0b11000000;
        } else {
          updated_pte = pte | 0b01000000;
        }
        if (PTESIZE == 32) {
          success = atomic_check_then_write_32(pte_phys.paddr, pte,
updated_pte);
        } else if (PTESIZE == 64) {
          success = atomic_check_then_write_64(pte_phys.paddr, pte,
updated_pte);
        } else {
          assert(false, "Unexpected PTESIZE");
        }
        if (!success) {
          i = i + 1;
        } else {
          TranslationResult pte_phys = translate_gstage({(pte[PA_SIZE - 3:(i *
VPN_SIZE) + 10] << 2), vaddr[11:0]}, vaddr, op, effective_mode, encoding);
          result.paddr = pte_phys.paddr;
          result.pbmt = pte_phys.pbmt == 0 ? pte[62:61] : pte_phys.pbmt;
          result.pte_flags = pte_flags;
          return result;
        }
      } else {
        raise(page_fault_code, effective_mode, vaddr);
      }
    }
    TranslationResult pte_phys = translate_gstage({(pte[PA_SIZE - 3:(i *
VPN_SIZE) + 10] << 2), vaddr[11:0]}, vaddr, op, effective_mode, encoding);
    result.paddr = pte_phys.paddr;
```

```
    if (PTESIZE >= 64) {
      result.pbmt = pte_phys.pbmt == Pbmt::PMA ? $enum(Pbmt, pte[62:61]) :
pte_phys.pbmt;
    }
    result.pte_flags = pte_flags;
    return result;
  } else {
    if (i == 0) {
      raise(page_fault_code, effective_mode, vaddr);
    }
    if (pte_flags.D == 1 || pte_flags.A == 1 || pte_flags.U == 1) {
      raise(page_fault_code, effective_mode, vaddr);
    }
    if ((VA_SIZE != 32) && (pte[62:61] != 0)) {
      raise(page_fault_code, effective_mode, vaddr);
    }
    if ((VA_SIZE != 32) && pte[63] != 0) {
      raise(page_fault_code, effective_mode, vaddr);
    }
    ppn = pte[PA_SIZE - 3:10] << 12;
  }
}
```

7.34. 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 if (exception_code == ExceptionCode::SoftwareCheck) {
  return tval;
} else {
  return 0;
}
```

7.35. sync_read_after_write_device (builtin)

Synchronize non-memory-mapped device read-after-write. Such devices have own addresses not related to memory map. Devices specified by device bitmask (up to 5 devices). Stalls processor till condtions met. Conditions are that last device output started or completed (depends on argument).

Return Type	`void`
Arguments	`Boolean completed, Bits<5> device_bitmask`

7.36. sync_write_after_read_device (builtin)

Synchronize non-memory-mapped device write-after-read. Such devices have own addresses not related to memory map. Devices specified by device bitmask (up to 5 devices). Stalls processor till condtions met. Conditions are that last device input started or completed (depends on argument).

Return Type	`void`
Arguments	`Boolean completed, Bits<5> device_bitmask`

7.37. tinst_transform

Returns the standard transformation of an encoding for htinst/mtinst

Return Type	`Bits<INSTR_ENC_SIZE>`
Arguments	`Bits<INSTR_ENC_SIZE> encoding, Bits<5> addr_offset`

```
if (encoding[1:0] == 0b11) {
  if (encoding[6:2] == 5'b00001) {
    return {{12{1'b0}}, addr_offset, encoding[14:0]};
  } else if (encoding[6:2] == 5'b01000) {
    return {{7{1'b0}}, encoding[24:20], addr_offset, encoding[14:12], {5{
1'b0}}, encoding[6:0]};
  } else if (encoding[6:2] == 5'b01011) {
    return {encoding[31:20], addr_offset, encoding[14:0]};
  } else if (encoding[6:2] == 5'b00011) {
```

```
    return {encoding[31:20], addr_offset, encoding[14:0]};
  } else {
    assert(false, "Bad transform");
  }
} else {
  assert(false, "TODO: compressed instruction");
}
```

7.38. tinst_value_for_guest_page_fault

Returns the value of htinst/mtinst for a Guest Page Fault

Return Type	XReg
Arguments	MemoryOperation op, Bits<INSTR_ENC_SIZE> encoding, Boolean for_final_vs_pte

```
if (for_final_vs_pte) {
  if (op == MemoryOperation::Fetch) {
    if (TINST_VALUE_ON_FINAL_INSTRUCTION_GUEST_PAGE_FAULT == "always zero") {
      return 0;
    } else {
      assert(TINST_VALUE_ON_FINAL_INSTRUCTION_GUEST_PAGE_FAULT == "always
pseudoinstruction", "Instruction guest page faults can only report zero/pseudo
instruction in tval");
      return 0x00002000;
    }
  } else if (op == MemoryOperation::Read) {
    if (TINST_VALUE_ON_FINAL_LOAD_GUEST_PAGE_FAULT == "always zero") {
      return 0;
    } else if (TINST_VALUE_ON_FINAL_LOAD_GUEST_PAGE_FAULT == "always
pseudoinstruction") {
      if ((VSXLEN == 32) || XLEN == 64) && (CSR[hstatus].VSXL == $bits
(XRegWidth::XLEN32)) {
        return 0x00002000;
      } else {
        return 0x00003000;
      }
    } else if (TINST_VALUE_ON_FINAL_LOAD_GUEST_PAGE_FAULT == "always
transformed standard instruction") {
      return tinst_transform(encoding, 0);
    } else {
      unpredictable("Custom value written into htinst/mtinst");
    }
  } else if (op == MemoryOperation::Write || op == MemoryOperation
::ReadModifyWrite) {
    if (TINST_VALUE_ON_FINAL_STORE_AMO_GUEST_PAGE_FAULT == "always zero") {
      return 0;
    } else if (TINST_VALUE_ON_FINAL_STORE_AMO_GUEST_PAGE_FAULT == "always
```

```
pseudoinstruction") {
      if ((VSXLEN == 32) || XLEN == 64) && (CSR[hstatus].VSXL == $bits
(XRegWidth::XLEN32)) {
        return 0x00002020;
      } else {
        return 0x00003020;
      }
    } else if (TINST_VALUE_ON_FINAL_STORE_AMO_GUEST_PAGE_FAULT == "always
transformed standard instruction") {
      return tinst_transform(encoding, 0);
    } else {
      unpredictable("Custom value written into htinst/mtinst");
    }
  }
} else {
  if (REPORT_GPA_IN_TVAL_ON_INTERMEDIATE_GUEST_PAGE_FAULT) {
    if ((VSXLEN == 32) || XLEN == 64) && (CSR[hstatus].VSXL == $bits(
XRegWidth::XLEN32)) {
      return 0x00002000;
    } else if ((VSXLEN == 64) || XLEN == 64) && (CSR[hstatus].VSXL == $bits
(XRegWidth::XLEN64)) {
      return 0x00003000;
    }
  }
}
```

7.39. translate

Translate a virtual address for operation type op that appears to execute at effective_mode.

The translation will depend on the effective privilege mode.

May raise a Page Fault or Access Fault.

The final physical address is **not** access checked (for PMP, PMA, etc., violations). (though intermediate page table reads will be)

Return Type	`TranslationResult`
Arguments	`XReg vaddr, MemoryOperation op, PrivilegeMode effective_mode, Bits<INSTR_ENC_SIZE> encoding`

```
Boolean cached_translation_valid;
TranslationResult cached_translation_result;
(cached_translation_valid, cached_translation_result = cached_translation(
vaddr, op));
if (cached_translation_valid) {
  return cached_translation_result;
}
```

```
TranslationResult result;
if (effective_mode == PrivilegeMode::M) {
  return vaddr;
}
SatpMode translation_mode = current_translation_mode(effective_mode);
if (translation_mode == SatpMode::Reserved) {
  if (op == MemoryOperation::Read) {
    raise(ExceptionCode::LoadPageFault, effective_mode, vaddr);
  } else if (op == MemoryOperation::Write || op == MemoryOperation
::ReadModifyWrite) {
    raise(ExceptionCode::StoreAmoPageFault, effective_mode, vaddr);
  } else {
    assert(op == MemoryOperation::Fetch, "Unexpected memory operation");
    raise(ExceptionCode::InstructionPageFault, effective_mode, vaddr);
  }
}
if (translation_mode == SatpMode::Sv32) {
  result = stage1_page_walk<32, 34, 32, 2>(vaddr, op, effective_mode,
encoding);
} else if (translation_mode == SatpMode::Sv39) {
  result = stage1_page_walk<39, 56, 64, 3>(vaddr, op, effective_mode,
encoding);
} else if (translation_mode == SatpMode::Sv48) {
  result = stage1_page_walk<48, 56, 64, 4>(vaddr, op, effective_mode,
encoding);
} else if (translation_mode == SatpMode::Sv57) {
  result = stage1_page_walk<57, 56, 64, 5>(vaddr, op, effective_mode,
encoding);
} else {
  assert(false, "Unexpected SatpMode");
}
maybe_cache_translation(vaddr, op, result);
return result;
```

7.40. translate_gstage

Translates a guest physical address to a physical address.

Return Type	`TranslationResult`
Arguments	`XReg gpaddr, XReg vaddr, MemoryOperation op, PrivilegeMode effective_mode, Bits<INSTR_ENC_SIZE> encoding`

```
TranslationResult result;
if (effective_mode == PrivilegeMode::S || effective_mode == PrivilegeMode::U) {
  result.paddr = gpaddr;
  return result;
}
Boolean mxr = CSR[mstatus].MXR == 1;
```

```
if (GSTAGE_MODE_BARE && CSR[hgatp].MODE == $bits(HgatpMode::Bare)) {
  result.paddr = gpaddr;
  return result;
} else if (SV32X4_TRANSLATION && CSR[hgatp].MODE == $bits(HgatpMode::Sv32x4)) {
  return gstage_page_walk<32, 34, 32, 2>(gpaddr, vaddr, op, effective_mode,
false, encoding);
} else if (SV39X4_TRANSLATION && CSR[hgatp].MODE == $bits(HgatpMode::Sv39x4)) {
  return gstage_page_walk<39, 56, 64, 3>(gpaddr, vaddr, op, effective_mode,
false, encoding);
} else if (SV48X4_TRANSLATION && CSR[hgatp].MODE == $bits(HgatpMode::Sv48x4)) {
  return gstage_page_walk<48, 56, 64, 4>(gpaddr, vaddr, op, effective_mode,
false, encoding);
} else if (SV57X4_TRANSLATION && CSR[hgatp].MODE == $bits(HgatpMode::Sv57x4)) {
  return gstage_page_walk<57, 56, 64, 5>(gpaddr, vaddr, op, effective_mode,
false, encoding);
} else {
  if (op == MemoryOperation::Read) {
    raise_guest_page_fault(op, gpaddr, vaddr, tinst_value_for_guest_page_fault
(op, encoding, true), effective_mode);
  } else if (op == MemoryOperation::Write || op == MemoryOperation
::ReadModifyWrite) {
    raise_guest_page_fault(op, gpaddr, vaddr, tinst_value_for_guest_page_fault
(op, encoding, true), effective_mode);
  } else {
    assert(op == MemoryOperation::Fetch, "unexpected memory op");
    raise_guest_page_fault(op, gpaddr, vaddr, tinst_value_for_guest_page_fault
(op, encoding, true), effective_mode);
  }
}
```

7.41. 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`

7.42. 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 if (exception_code == ExceptionCode::SoftwareCheck) {
  return tval;
} else {
  return 0;
}
```

7.43. write_device (builtin)

Write to non-memory-mapped device. Such devices have own addresses not related to memory map.

Return Type	`void`
Arguments	`XReg dev_addr, Bits<len> value`

7.44. write_memory

Write to virtual memory

Return Type	`void`
Arguments	`XReg virtual_address, Bits<LEN> value, Bits<INSTR_ENC_SIZE> encoding`

```
Boolean aligned = is_naturally_aligned<LEN>(virtual_address);
XReg physical_address;
if (aligned) {
  write_memory_aligned<LEN>(virtual_address, value, encoding);
```

```
}
if (MISALIGNED_MAX_ATOMICITY_GRANULE_SIZE > 0) {
  assert(MISALIGNED_LDST_EXCEPTION_PRIORITY == "low", "Invalid config: can't
mix low-priority misaligned exceptions with large atomicity granule");
  physical_address = (CSR[misa].S == 1) ? translate(virtual_address,
MemoryOperation::Write, effective_ldst_mode(), encoding).paddr :
virtual_address;
  if (misaligned_is_atomic?<LEN>(physical_address)) {
    access_check(physical_address, LEN, virtual_address, MemoryOperation::
Write, ExceptionCode::StoreAmoAccessFault, effective_ldst_mode());
    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, effective_ldst_mode(), encoding).paddr :
virtual_address;
    access_check(physical_address, LEN, virtual_address, MemoryOperation::
Write, ExceptionCode::StoreAmoAccessFault, effective_ldst_mode());
  }
  raise(ExceptionCode::StoreAmoAddressMisaligned, effective_ldst_mode(),
virtual_address);
} else {
  if (MISALIGNED_SPLIT_STRATEGY == "by_byte") {
    for (U32 i = 0; i <= LEN; i++) {
      write_memory_aligned<8>(virtual_address + i, (value >> (8 * i))[7:0],
encoding);
    }
  } 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");
  }
}
```

7.45. write_memory_aligned

Write to virtual memory using a known aligned address.

Return Type	`void`
Arguments	`XReg virtual_address, Bits<LEN> value, Bits<INSTR_ENC_SIZE> encoding`

```
XReg physical_address;
physical_address = (CSR[misa].S == 1) ? translate(virtual_address,
MemoryOperation::Write, effective_ldst_mode(), encoding).paddr :
virtual_address;
```

```
access_check(physical_address, LEN, virtual_address, MemoryOperation::Write,
ExceptionCode::StoreAmoAccessFault, effective_ldst_mode());
write_physical_memory<LEN>(physical_address, value);
```

7.46. write_physical_memory (builtin)

Write to physical memory.

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

7.47. xlen

Returns the effective XLEN for the current privilege mode.

Return Type	`Bits<8>`
Arguments	

```
if (XLEN == 32) {
  return 32;
} else {
  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;
    }
  }
}
```