

DRAFT

16-bit Push/Pop instructions and double-moves

Version 0.2.0, 2025-02-26

Table of Contents

Licensing and Acknowledgements	1
Copyright and license information	2
Acknowledgements	3
Versions	4
Version History	4
1. Conventions	5
2. Extension description	6
3. Instruction summary	7
4. CSR summary	8
5. Csrs (in alphabetical order)	9
5.1. qc.mstkbottmaddr	10
5.1.1. Attributes	10
5.1.2. Format	10
5.1.3. Field Summary	10
5.1.4. Fields	10
STKADDR	10
RESERVED	11
5.2. qc.mstktopaddr	12
5.2.1. Attributes	12
5.2.2. Format	12
5.2.3. Field Summary	12
5.2.4. Fields	12
STKADDR	12
RESERVED	13
6. Instructions (in alphabetical order)	14
6.1. qc.cm.mva01s	14
6.2. qc.cm.mvsa01	15
6.3. qc.cm.pop	16
6.4. qc.cm.popret	18
6.5. qc.cm.popretz	20
6.6. qc.cm.push	22
6.7. qc.cm.pushfp	24
7. IDL Functions	26

Licensing and Acknowledgements



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

Change should be expected

DRAFT

Copyright and license information

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

Copyright 2025 by Qualcomm Technologies, Inc..

DRAFT

Acknowledgements

Contributors to version 0.2.0 of the specification (in alphabetical order) include:

- James Ball <jameball@qti.qualcomm.com> (Qualcomm Technologies, Inc.)
- Derek Hower <dhower@qti.qualcomm.com> (Qualcomm Technologies, Inc.)
- Ana Pazos <apazos@qti.qualcomm.com> (Qualcomm Technologies, Inc.)
- Albert Yosher <ayosher@qti.qualcomm.com> (Qualcomm Technologies, Inc.)

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

DRAFT

Versions

This specification documents version 0.2.0 of Xqccmp.

Version History

Version

0.1.0

State

development

Version

0.2.0

State

development

Changes

- Fix all push and pop instructions description for RV64 (IDL code fix)
- Fix qc.cm.pushfp instruction to avoid rlist == 4, since not saving s0 == fp but updating it, should not occur
- Add CSRs qc.mstkbottomaddr and qc.mstktopaddr to support cache range checking
- Add list of supported custom exceptions - stack alignment check and stack range check
- Add stack exception checks in all push/pop instructions

Chapter 1. Conventions

Version requirements are specified as conditions using the following operators:

Operator	Meaning
~> VERSION	Accepts any version that is <i>compatible</i> 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.

DRAFT

Chapter 2. Extension description

The Xqccmp extension is a set of instructions which may be executed as a series of existing 32-bit RISC-V instructions.

This extension reuses some encodings from *c.fsdsp*. Therefore it is *incompatible* with [\[Zcd\]](#), which is included when C and D extensions are both present.



Xqccmp is primarily targeted at embedded class CPUs due to implementation complexity. Additionally, it is not compatible with architecture class profiles.

The Xqccmp extension depends on the [\[Zca\]](#) extension.

The Xqccmp extension using same encodings as Zcmp extension for similar instructions, with 2 major differences:

- Order of load and store of registers is opposite to Zcmp order, but compliant with SW ABI
- Xqccmp on top of Zcmp instructions defines qc.cm.pushfp instruction to manage frame pointer

The PUSH/POP assembly syntax uses several variables, the meaning of which are:

- *reg_list* is a list containing 1 to 13 registers (ra and 0 to 12 s registers)
 - valid values: {ra}, {ra, s0}, {ra, s0-s1}, {ra, s0-s2}, ..., {ra, s0-s8}, {ra, s0-s9}, {ra, s0-s11}
 - note that {ra, s0-s10} is *not* valid, giving 12 lists not 13 for better encoding
- *stack_adj* is the total size of the stack frame.
 - valid values vary with register list length and the specific encoding, see the instruction pages for details.

RV32	RV64	Mnemonic	Instruction
yes	yes	qc.cm.push {reg_list}, -stack_adj	[insns-qc_cm_push]
yes	yes	qc.cm.pushfp {reg_list}, -stack_adj	[insns-qc_cm_pushfp]
yes	yes	qc.cm.pop {reg_list}, stack_adj	[insns-qc_cm_pop]
yes	yes	qc.cm.popret {reg_list}, stack_adj	[insns-qc_cm_popret]
yes	yes	qc.cm.popretz {reg_list}, stack_adj	[insns-qc_cm_popretz]
yes	yes	qc.cm.mva01s rs1', rs2'	[insns-qc_cm_mva01s]
yes	yes	qc.cm.mvsa01 r1s', r2s'	[insns-qc_cm_mvsa01]

Chapter 3. Instruction summary

The following 7 instructions are added by this extension:

RV32	RV64	Mnemonic	Instruction
✓	✓	<code>qc.cm.mva01s r1s, r2s</code>	Move two s0-s7 registers into a0-a1
✓	✓	<code>qc.cm.mvsa01 r1s, r2s</code>	Move a0-a1 into two registers of s0-s7
✓	✓	<code>qc.cm.pop reg_list, stack_adj</code>	Destroy function call stack frame
✓	✓	<code>qc.cm.popret reg_list, stack_adj</code>	Destroy function call stack frame and return to ra.
✓	✓	<code>qc.cm.popretz reg_list, stack_adj</code>	Destroy function call stack frame, move zero to a0 and return to ra.
✓	✓	<code>qc.cm.push reg_list, -stack_adj</code>	Create function call stack frame
✓	✓	<code>qc.cm.pushfp reg_list, -stack_adj</code>	Create function call stack frame with frame pointer

DRAFT

Chapter 4. CSR summary

The following 2 are added by this extension.

RV32	RV64	CSR	Name
✓		qc.mstkbottomaddr	Machine Stack Bottom Limit
✓		qc.mstktopaddr	Machine Stack Top Limit

DRAFT

Chapter 5. Csrs (in alphabetical order)

DRAFT

5.1. qc.mstkbottomaddr

Machine Stack Bottom Limit

Stack bottom limit register. If the stack pointer (sp == x2) lesser then its value - SpOutOfRange exception occurs

5.1.1. Attributes

Defining Extension	- anyOf: - Xqccmp, version >= 0 - Xqci, version >= 0 - Xqciint, version >= 0
CSR Address	0x7c5
Length	32-bit
Privilege Mode	M

5.1.2. Format

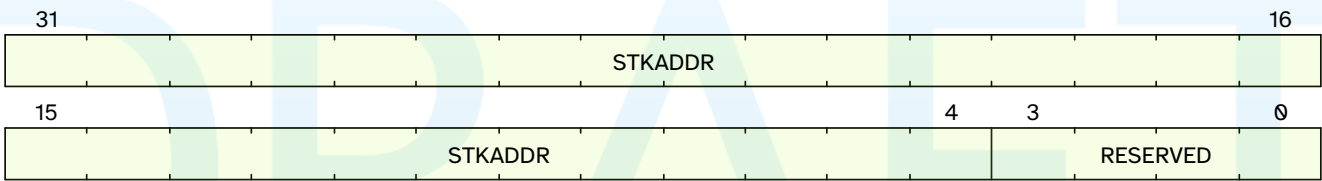


Figure 1. qc.mstkbottomaddr format

5.1.3. Field Summary

Name	Location	Type	Reset Value
qc.mstkbottomaddr.STKADDR	31:4	RW	0
qc.mstkbottomaddr.RESERVED	3:0	RO	0

5.1.4. Fields

STKADDR

Location

31:4

Description

If the stack pointer (sp == x2) lesser then STKADDR*16 - SpOutOfRange exception occurs

Type

RW

Reset value

0

RESERVED**Location**

3:0

Description*Reserved and must be zero***Type***RO***Reset value**

0

DRAFT

5.2. qc.mstktopaddr

Machine Stack Top Limit

Stack top limit register. If the stack pointer (sp == x2) greater then its value - SpOutOfRange exception occurs

5.2.1. Attributes

Defining Extension	- anyOf: - Xqccmp, version >= 0 - Xqci, version >= 0 - Xqciint, version >= 0
CSR Address	0x7c4
Length	32-bit
Privilege Mode	M

5.2.2. Format

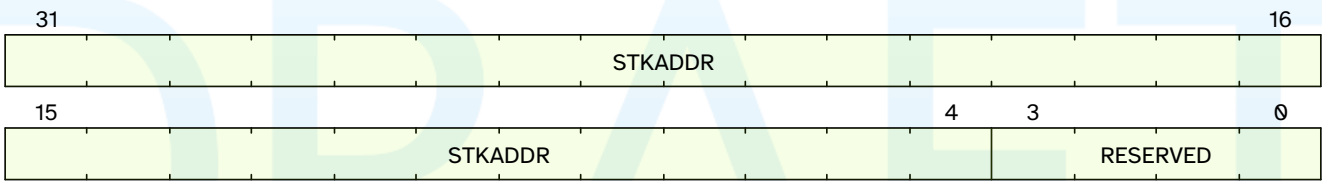


Figure 2. qc.mstktopaddr format

5.2.3. Field Summary

Name	Location	Type	Reset Value
qc.mstktopaddr.STKADDR	31:4	RW	0
qc.mstktopaddr.RESERVED	3:0	RO	0

5.2.4. Fields

STKADDR

Location

31:4

Description

If the stack pointer (sp == x2) greater then STKADDR*16 - SpOutOfRange exception occurs

Type

RW

Reset value

0

RESERVED**Location**

3:0

Description*Reserved and must be zero***Type***RO***Reset value**

0

DRAFT

Chapter 6. Instructions (in alphabetical order)

6.1. qc.cm.mva01s

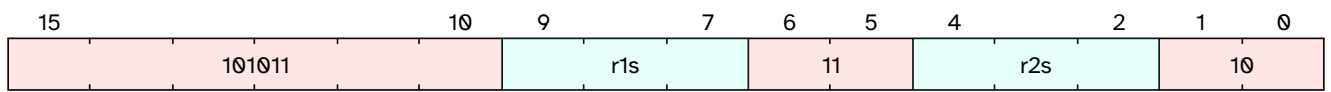
Synopsis

Move two s0-s7 registers into a0-a1

Mnemonic

```
qc.cm.mva01s r1s, r2s
```

Encoding



Description

This instruction moves r1s' into a0 and r2s' into a1. The execution is atomic, so it is not possible to observe state where only one of a0 or a1 have been updated. The encoding uses sreg number specifiers instead of xreg number specifiers to save encoding space. The mapping between them is specified in the pseudo-code below.

Decode Variables

```
Bits<3> r1s = $encoding[9:7];
Bits<3> r2s = $encoding[4:2];
```

Operation

```
if (implemented?(ExtensionName::Zcmp) && (CSR[misa].C == 1'b0)) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg xreg1 = (r1s[2:1] > 0) ? {1, 0, r1s[2:0]} : {0, 1, r1s[2:0]};
XReg xreg2 = (r2s[2:1] > 0) ? {1, 0, r2s[2:0]} : {0, 1, r2s[2:0]};
X[10] = X[xreg1];
X[11] = X[xreg2];
```

Included in

Extension	Version
Xqccmp	~> 0.1.0

6.2. qc.cm.mvsa01

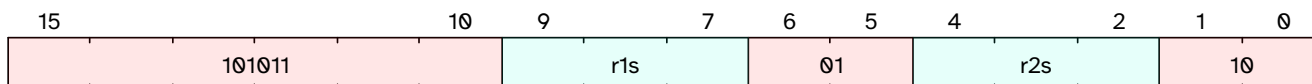
Synopsis

Move a0-a1 into two registers of s0-s7

Mnemonic

```
qc.cm.mvsa01 r1s, r2s
```

Encoding



Description

This instruction moves a0 into r1s' and a1 into r2s'. r1s' and r2s' must be different. The execution is atomic, so it is not possible to observe state where only one of r1s' or r2s' has been updated. The encoding uses sreg number specifiers instead of xreg number specifiers to save encoding space. The mapping between them is specified in the pseudo-code below.

Decode Variables

```
Bits<3> r1s = $encoding[9:7];
Bits<3> r2s = $encoding[4:2];
```

Operation

```
if (implemented?(ExtensionName::Zcmp) && (CSR[misa].C == 1'b0)) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg xreg1 = (r1s[2:1] > 0) ? {1, 0, r1s[2:0]} : {0, 1, r1s[2:0]};
XReg xreg2 = (r2s[2:1] > 0) ? {1, 0, r2s[2:0]} : {0, 1, r2s[2:0]};
X[xreg1] = X[10];
X[xreg2] = X[11];
```

Included in

Extension	Version
Xqccmp	~> 0.1.0

6.3. qc.cm.pop

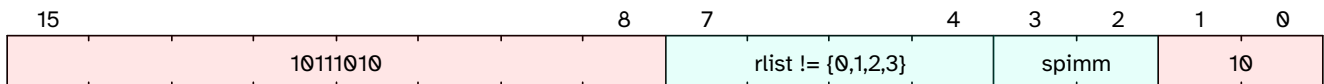
Synopsis

Destroy function call stack frame

Mnemonic

```
qc.cm.pop reg_list, stack_adj
```

Encoding



Description

Destroy stack frame: load ra and 0 to 12 saved registers from the stack frame, deallocate the stack frame. This instruction pops (loads) the registers in `reg_list` from stack memory, and then adjusts the stack pointer by `stack_adj`.

Restrictions on `stack_adj`:

- it must be enough to store all of the listed registers
- it must be a multiple of 16 (bytes):
 - for RV32 the allowed values are: 16, 32, 48, 64, 80, 96, 112
 - for RV64 the allowed values are: 16, 32, 48, 64, 80, 96, 112, 128, 144, 160

Decode Variables

```
Bits<4> rlist = $encoding[7:4];
Bits<2> spimm = $encoding[3:2];
```

Operation

```
if (implemented?(ExtensionName::Xqccmp) && (CSR[misa].C == 1'b0)) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (X[2] < CSR[qc.mstkbottomaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] > CSR[qc.mstktopaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] & 0xf != 0) {
    raise(ExceptionCode::IllegalStackPointer, mode(), $encoding);
}
XReg size = xlen();
XReg nreg = (rlist == 15) ? 13 : (rlist - 3);
```

```

XReg stack_aligned_adj = (nreg * size + 15) & ~0xF;
XReg virtual_address_sp = X[2];
XReg virtual_address_new_sp = virtual_address_sp + stack_aligned_adj + spimm;
XReg virtual_address_base = virtual_address_new_sp - size;
X[1] = read_memory_xlen(virtual_address_base - 0 * size, $encoding);
if (nreg > 1) {
    X[8] = read_memory_xlen(virtual_address_base - 1 * size, $encoding);
}
if (nreg > 2) {
    X[9] = read_memory_xlen(virtual_address_base - 2 * size, $encoding);
}
if (nreg > 3) {
    X[18] = read_memory_xlen(virtual_address_base - 3 * size, $encoding);
}
if (nreg > 4) {
    X[19] = read_memory_xlen(virtual_address_base - 4 * size, $encoding);
}
if (nreg > 5) {
    X[20] = read_memory_xlen(virtual_address_base - 5 * size, $encoding);
}
if (nreg > 6) {
    X[21] = read_memory_xlen(virtual_address_base - 6 * size, $encoding);
}
if (nreg > 7) {
    X[22] = read_memory_xlen(virtual_address_base - 7 * size, $encoding);
}
if (nreg > 8) {
    X[23] = read_memory_xlen(virtual_address_base - 8 * size, $encoding);
}
if (nreg > 9) {
    X[24] = read_memory_xlen(virtual_address_base - 9 * size, $encoding);
}
if (nreg > 10) {
    X[25] = read_memory_xlen(virtual_address_base - 10 * size, $encoding);
}
if (nreg > 11) {
    X[26] = read_memory_xlen(virtual_address_base - 11 * size, $encoding);
    X[27] = read_memory_xlen(virtual_address_base - 12 * size, $encoding);
}
X[2] = virtual_address_new_sp;

```

Included in

Extension	Version
Xqccmp	~> 0.1.0

6.4. qc.cm.popret

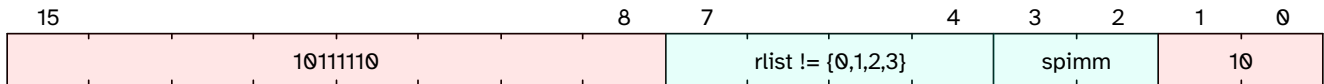
Synopsis

Destroy function call stack frame and return to ra.

Mnemonic

```
qc.cm.popret reg_list, stack_adj
```

Encoding



Description

Destroy stack frame: load ra and 0 to 12 saved registers from the stack frame, deallocate the stack frame, return to ra. This instruction pops (loads) the registers in reg_list from stack memory, and then adjusts the stack pointer by stack_adj and then return to ra.

Restrictions on stack_adj:

- it must be enough to store all of the listed registers
- it must be a multiple of 16 (bytes):
 - for RV32 the allowed values are: 16, 32, 48, 64, 80, 96, 112
 - for RV64 the allowed values are: 16, 32, 48, 64, 80, 96, 112, 128, 144, 160

Decode Variables

```
Bits<4> rlist = $encoding[7:4];
Bits<2> spimm = $encoding[3:2];
```

Operation

```
if (implemented?(ExtensionName::Xqccmp) && (CSR[misa].C == 1'b0)) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (X[2] < CSR[qc.mstkbottomaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] > CSR[qc.mstktopaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] & 0xf != 0) {
    raise(ExceptionCode::IllegalStackPointer, mode(), $encoding);
}
XReg size = xlen();
XReg nreg = (rlist == 15) ? 13 : (rlist - 3);
```

```

XReg stack_aligned_adj = (nreg * size + 15) & ~0xF;
XReg virtual_address_sp = X[2];
XReg virtual_address_new_sp = virtual_address_sp + stack_aligned_adj + spimm;
XReg virtual_address_base = virtual_address_new_sp - size;
X[1] = read_memory_xlen(virtual_address_base - 0 * size, $encoding);
if (nreg > 1) {
    X[8] = read_memory_xlen(virtual_address_base - 1 * size, $encoding);
}
if (nreg > 2) {
    X[9] = read_memory_xlen(virtual_address_base - 2 * size, $encoding);
}
if (nreg > 3) {
    X[18] = read_memory_xlen(virtual_address_base - 3 * size, $encoding);
}
if (nreg > 4) {
    X[19] = read_memory_xlen(virtual_address_base - 4 * size, $encoding);
}
if (nreg > 5) {
    X[20] = read_memory_xlen(virtual_address_base - 5 * size, $encoding);
}
if (nreg > 6) {
    X[21] = read_memory_xlen(virtual_address_base - 6 * size, $encoding);
}
if (nreg > 7) {
    X[22] = read_memory_xlen(virtual_address_base - 7 * size, $encoding);
}
if (nreg > 8) {
    X[23] = read_memory_xlen(virtual_address_base - 8 * size, $encoding);
}
if (nreg > 9) {
    X[24] = read_memory_xlen(virtual_address_base - 9 * size, $encoding);
}
if (nreg > 10) {
    X[25] = read_memory_xlen(virtual_address_base - 10 * size, $encoding);
}
if (nreg > 11) {
    X[26] = read_memory_xlen(virtual_address_base - 11 * size, $encoding);
    X[27] = read_memory_xlen(virtual_address_base - 12 * size, $encoding);
}
X[2] = virtual_address_new_sp;
jump(X[1]);

```

Included in

Extension	Version
Xqccmp	~> 0.1.0

6.5. qc.cm.popretz

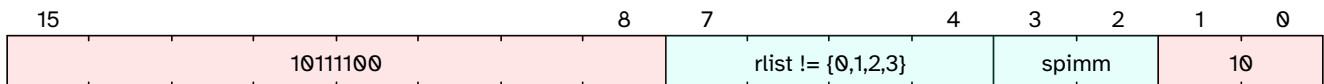
Synopsis

Destroy function call stack frame, move zero to a0 and return to ra.

Mnemonic

```
qc.cm.popretz reg_list, stack_adj
```

Encoding



Description

Destroy stack frame: load ra and 0 to 12 saved registers from the stack frame, deallocate the stack frame, move zero to a0, return to ra. This instruction pops (loads) the registers in reg_list from stack memory, and then adjusts the stack pointer by stack_adj, move zero to a0 and then return to ra.

Restrictions on stack_adj:

- it must be enough to store all of the listed registers
- it must be a multiple of 16 (bytes):
 - for RV32 the allowed values are: 16, 32, 48, 64, 80, 96, 112
 - for RV64 the allowed values are: 16, 32, 48, 64, 80, 96, 112, 128, 144, 160

Decode Variables

```
Bits<4> rlist = $encoding[7:4];
Bits<2> spimm = $encoding[3:2];
```

Operation

```
if (implemented?(ExtensionName::Xqccmp) && (CSR[misa].C == 1'b0)) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (X[2] < CSR[qc.mstkbottmaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] > CSR[qc.mstktopaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] & 0xf != 0) {
    raise(ExceptionCode::IllegalStackPointer, mode(), $encoding);
}
XReg size = xlen();
XReg nreg = (rlist == 15) ? 13 : (rlist - 3);
```

```

XReg stack_aligned_adj = (nreg * size + 15) & ~0xF;
XReg virtual_address_sp = X[2];
XReg virtual_address_new_sp = virtual_address_sp + stack_aligned_adj + spimm;
XReg virtual_address_base = virtual_address_new_sp - size;
X[1] = read_memory_xlen(virtual_address_base - 0 * size, $encoding);
if (nreg > 1) {
    X[8] = read_memory_xlen(virtual_address_base - 1 * size, $encoding);
}
if (nreg > 2) {
    X[9] = read_memory_xlen(virtual_address_base - 2 * size, $encoding);
}
if (nreg > 3) {
    X[18] = read_memory_xlen(virtual_address_base - 3 * size, $encoding);
}
if (nreg > 4) {
    X[19] = read_memory_xlen(virtual_address_base - 4 * size, $encoding);
}
if (nreg > 5) {
    X[20] = read_memory_xlen(virtual_address_base - 5 * size, $encoding);
}
if (nreg > 6) {
    X[21] = read_memory_xlen(virtual_address_base - 6 * size, $encoding);
}
if (nreg > 7) {
    X[22] = read_memory_xlen(virtual_address_base - 7 * size, $encoding);
}
if (nreg > 8) {
    X[23] = read_memory_xlen(virtual_address_base - 8 * size, $encoding);
}
if (nreg > 9) {
    X[24] = read_memory_xlen(virtual_address_base - 9 * size, $encoding);
}
if (nreg > 10) {
    X[25] = read_memory_xlen(virtual_address_base - 10 * size, $encoding);
}
if (nreg > 11) {
    X[26] = read_memory_xlen(virtual_address_base - 11 * size, $encoding);
    X[27] = read_memory_xlen(virtual_address_base - 12 * size, $encoding);
}
X[2] = virtual_address_new_sp;
X[10] = 0;
jump(X[1]);

```

Included in

Extension	Version
Xqccmp	~> 0.1.0

6.6. qc.cm.push

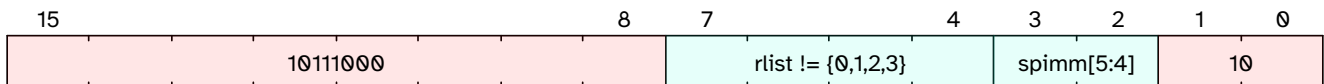
Synopsis

Create function call stack frame

Mnemonic

```
qc.cm.push reg_list, -stack_adj
```

Encoding



Description

Create stack frame: store `ra` and `0` to 12 saved registers to the stack frame, optionally allocate additional stack space. This instruction pushes (stores) the registers in `rlist` to the memory below the stack pointer, and then creates the stack frame by decrementing the stack pointer by `stack_adj`, including any additional stack space requested by the value of `spimm`.

Restrictions on `stack_adj`:

- it must be enough to store all of the listed registers
- it must be a multiple of 16 (bytes):
 - for RV32 the allowed values are: 16, 32, 48, 64, 80, 96, 112
 - for RV64 the allowed values are: 16, 32, 48, 64, 80, 96, 112, 128, 144, 160

Decode Variables

```
Bits<4> rlist = $encoding[7:4];
Bits<6> spimm = {$encoding[3:2], 4'd0};
```

Operation

```
if (implemented?(ExtensionName::Xqccmp) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (X[2] < CSR[qc.mstkbottombottomaddr]) {
  raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] > CSR[qc.mstktopaddr]) {
  raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] & 0xf != 0) {
  raise(ExceptionCode::IllegalStackPointer, mode(), $encoding);
}
XReg size = xlen();
```

```

XReg nreg = (rlist == 15) ? 13 : (rlist - 3);
XReg stack_aligned_adj = (nreg * size + 15) & ~0xF;
XReg virtual_address_sp = X[2];
XReg virtual_address_new_sp = virtual_address_sp - stack_aligned_adj - spimm;
XReg virtual_address_base = virtual_address_sp - size;
write_memory_xlen(virtual_address_base - 0 * size, X[1], $encoding);
if (nreg > 1) {
    write_memory_xlen(virtual_address_base - 1 * size, X[8], $encoding);
}
if (nreg > 2) {
    write_memory_xlen(virtual_address_base - 2 * size, X[9], $encoding);
}
if (nreg > 3) {
    write_memory_xlen(virtual_address_base - 3 * size, X[18], $encoding);
}
if (nreg > 4) {
    write_memory_xlen(virtual_address_base - 4 * size, X[19], $encoding);
}
if (nreg > 5) {
    write_memory_xlen(virtual_address_base - 5 * size, X[20], $encoding);
}
if (nreg > 6) {
    write_memory_xlen(virtual_address_base - 6 * size, X[21], $encoding);
}
if (nreg > 7) {
    write_memory_xlen(virtual_address_base - 7 * size, X[22], $encoding);
}
if (nreg > 8) {
    write_memory_xlen(virtual_address_base - 8 * size, X[23], $encoding);
}
if (nreg > 9) {
    write_memory_xlen(virtual_address_base - 9 * size, X[24], $encoding);
}
if (nreg > 10) {
    write_memory_xlen(virtual_address_base - 10 * size, X[25], $encoding);
}
if (nreg > 11) {
    write_memory_xlen(virtual_address_base - 11 * size, X[26], $encoding);
    write_memory_xlen(virtual_address_base - 12 * size, X[27], $encoding);
}
X[2] = virtual_address_new_sp;

```

Included in

Extension	Version
Xqccmp	~> 0.1.0

6.7. qc.cm.pushfp

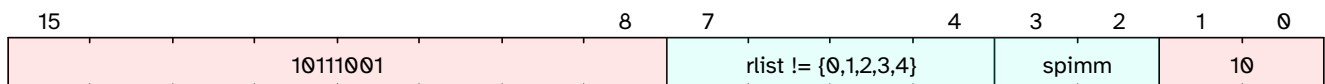
Synopsis

Create function call stack frame with frame pointer

Mnemonic

```
qc.cm.pushfp reg_list, -stack_adj
```

Encoding



Description

Create stack frame: store `ra` and `0` to `12` saved registers to the stack frame, optionally allocate additional stack space. As well, instruction initializes frame pointer (`fp`, aka `x8`) to the Canonical Frame Address (`sp` aka `x2`) on function entry. This instruction pushes (stores) the registers in `rlist` to the memory below the stack pointer, and then creates the stack frame by decrementing the stack pointer by `stack_adj`, including any additional stack space requested by the value of `spimm`.

Restrictions on `stack_adj`:

- it must be enough to store all of the listed registers
- it must be a multiple of 16 (bytes):
 - for RV32 the allowed values are: 16, 32, 48, 64, 80, 96, 112
 - for RV64 the allowed values are: 16, 32, 48, 64, 80, 96, 112, 128, 144, 160

Decode Variables

```
Bits<4> rlist = $encoding[7:4];
Bits<2> spimm = $encoding[3:2];
```

Operation

```
if (implemented?(ExtensionName::Xqccmp) && (CSR[misa].C == 1'b0)) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (X[2] < CSR[qc.mstkbottompaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] > CSR[qc.mstktoppaddr]) {
    raise(ExceptionCode::SpOutOfRange, mode(), $encoding);
}
if (X[2] & 0xf != 0) {
    raise(ExceptionCode::IllegalStackPointer, mode(), $encoding);
}
```

```

XReg size = xlen();
XReg nreg = (rlist == 15) ? 13 : (rlist - 3);
XReg stack_aligned_adj = (nreg * size + 15) & ~0xF;
XReg virtual_address_sp = X[2];
XReg virtual_address_new_sp = virtual_address_sp - stack_aligned_adj - spimm;
XReg virtual_address_base = virtual_address_sp - size;
write_memory_xlen(virtual_address_base - 0 * size, X[1], $encoding);
if (nreg > 1) {
    write_memory_xlen(virtual_address_base - 1 * size, X[8], $encoding);
}
if (nreg > 2) {
    write_memory_xlen(virtual_address_base - 2 * size, X[9], $encoding);
}
if (nreg > 3) {
    write_memory_xlen(virtual_address_base - 3 * size, X[18], $encoding);
}
if (nreg > 4) {
    write_memory_xlen(virtual_address_base - 4 * size, X[19], $encoding);
}
if (nreg > 5) {
    write_memory_xlen(virtual_address_base - 5 * size, X[20], $encoding);
}
if (nreg > 6) {
    write_memory_xlen(virtual_address_base - 6 * size, X[21], $encoding);
}
if (nreg > 7) {
    write_memory_xlen(virtual_address_base - 7 * size, X[22], $encoding);
}
if (nreg > 8) {
    write_memory_xlen(virtual_address_base - 8 * size, X[23], $encoding);
}
if (nreg > 9) {
    write_memory_xlen(virtual_address_base - 9 * size, X[24], $encoding);
}
if (nreg > 10) {
    write_memory_xlen(virtual_address_base - 10 * size, X[25], $encoding);
}
if (nreg > 11) {
    write_memory_xlen(virtual_address_base - 11 * size, X[26], $encoding);
    write_memory_xlen(virtual_address_base - 12 * size, X[27], $encoding);
}
X[8] = virtual_address_sp;
X[2] = virtual_address_new_sp;

```

Included in

Extension	Version
Xqccmp	~> 0.1.0

Chapter 7. IDL Functions

DRAFT