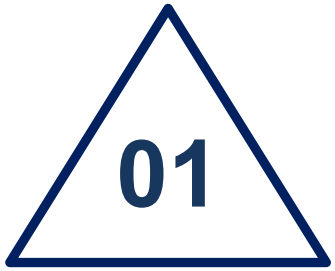
RISC-V ISA

Chapter 3
Standard Extensions

CONTENTS



Introduction

Definitions for standard extension

3.1 Introduction

RISC-V has defined base integer instructions and some privilege instructions, another goal of RISC-V is to provide a basis for more specialised instruction-set extensions or more customised accelerators.

- **Standard Extension**:

Designed for general usages, which is not conflict with another standard extensions.

- **Non-standard Extension**:

Usually designed highly specialised, which may cause conflict with other extensions. Not covered here.

3.1 Introduction

Standard Extension	Description
M	Integer Multiply/Divide extension
F	Single-precision floating-point extension
D	Double-precision floating-point extension
Q	Quad-precision floating-point extension
A	Atomic extension
C	Compressed extension

02

Standard Extensions

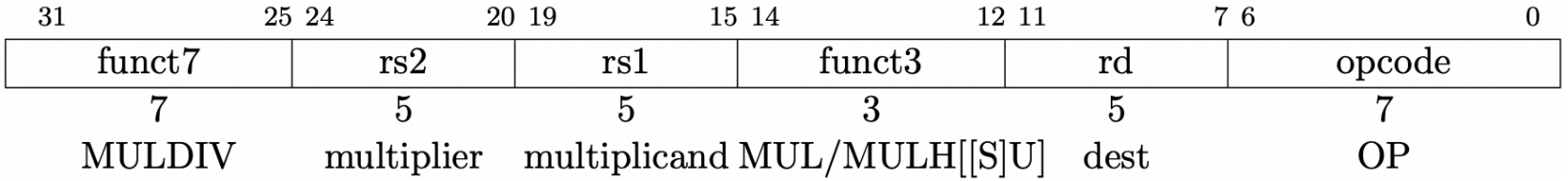
Cover extensions including “M”, ”F”, “A”, and “C” extensions

1. “M” Standard Extension

2. “F”, ”D”, ”Q” Standard Extension
3. “A” Standard Extension
4. “C” Standard Extension

3.2 Standard Extensions

3.2.1 “M” Standard Extension



MUL performs an *XLEN-bit* × *XLEN-bit* multiplication of rs_1 *by* rs_2, then places lower *XLEN-bit* of the product into destination register *rd*.

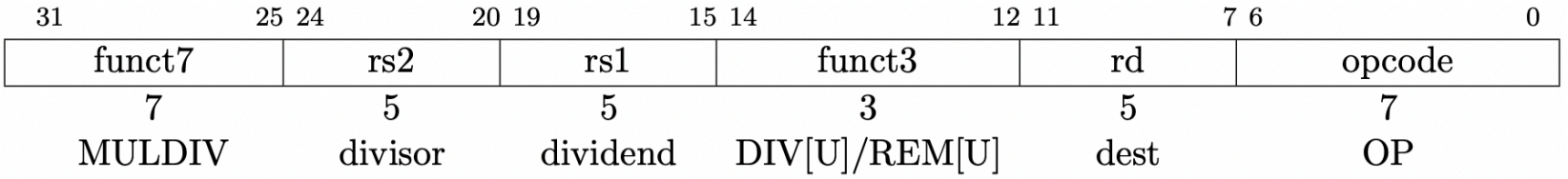
$$rd = lower(rs_1 \times rs_2)$$

MULH performs the similar computation but places upper *XLEN-bit* of the product into destination register *rd*.

$$rd = upper(rs_1 \times rs_2)$$

3.2 Standard Extensions

3.2.1 “M” Standard Extension



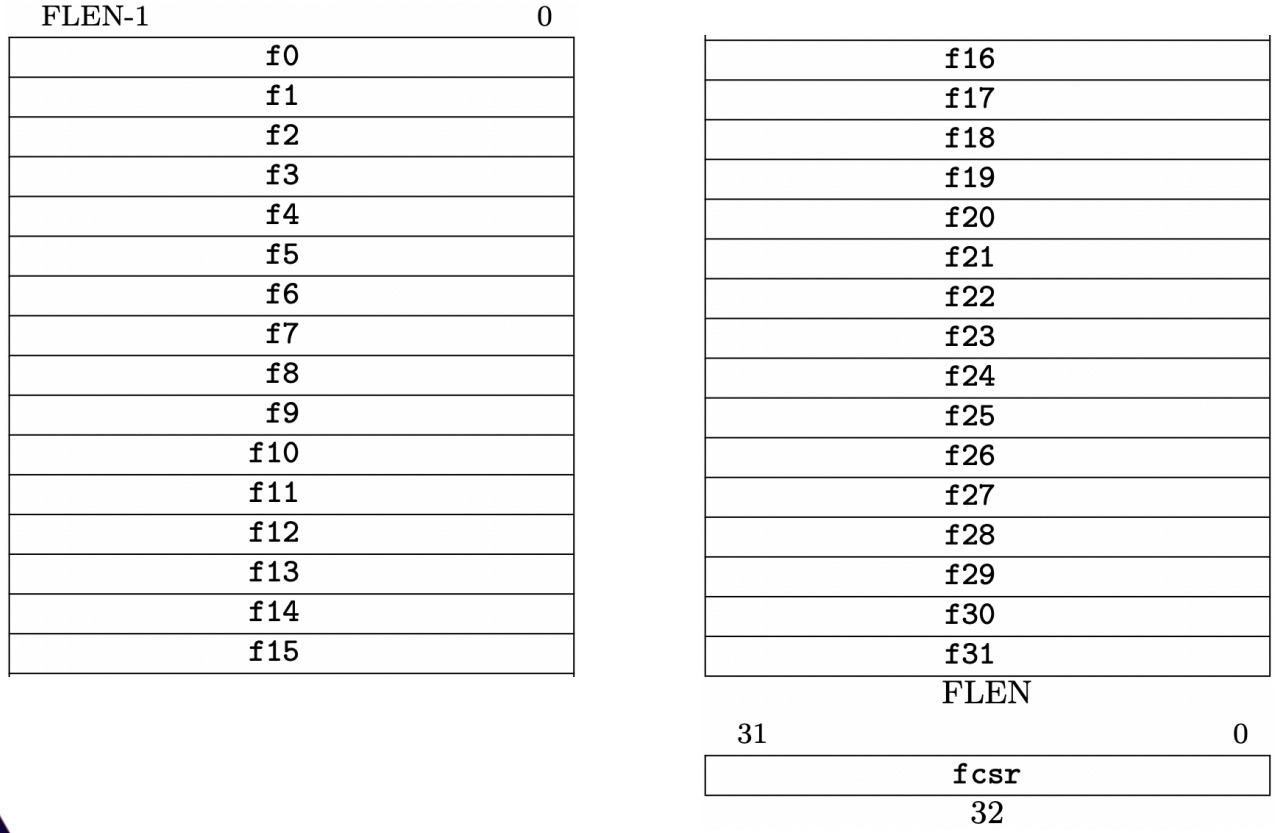
DIV performs an *XLEN-bit* by *XLEN-bit* signed integer division of rs_1 *by* rs_2, then places the quotient into destination register *rd*.

REM performs the similar computation, then places remainder into destination register *rd*.

3.2 Standard Extensions

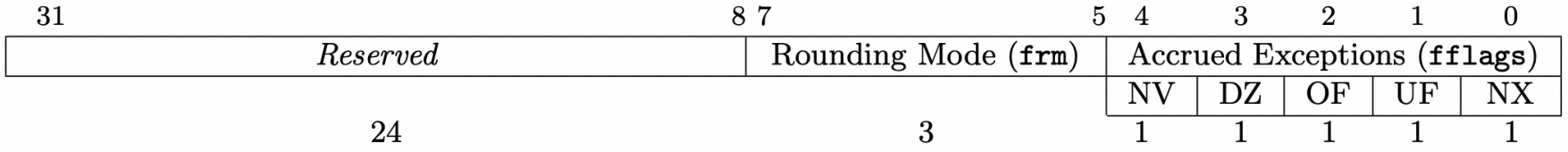
3.2.2 “F”, “D”, “Q” Standard Extension

“F”, ”D”, ”Q” represent the single-precision, double-precision, and quad-precision respectively.



3.2 Standard Extensions

3.2.2 “F”, “D”, “Q” Standard Extension



Rounding Mode	Mnemonic	Meaning
000	RNE	Round to Nearest, ties to Even
001	RTZ	Round towards Zero
010	RDN	Round Down (towards $-\infty$)
011	RUP	Round Up (towards $+\infty$)
100	RMM	Round to Nearest, ties to Max Magnitude
101		*Invalid. Reserved for future use.*
110		*Invalid. Reserved for future use.*
111	DYN	In instruction’s *rm* field, selects dynamic rounding mode; In Rounding Mode register, *Invalid.*

Flag Mnemonic	Flag Meaning
NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

3.2 Standard Extensions

3.2.2 “F”, “D”, “Q” Standard Extension

Floating-point operations can use either static rounding mode or dynamic rounding mode:

- **Static rounding mode**:

Select the *rm* field from 000 *to* 100. The specific instruction utilises the static rounding mode corresponding to *rm* field.

- **Dynamic rounding mode**:

Select the *rm* field as 111. The specific instruction utilises the dynamic rounding mode corresponding to *frm* field in *fcsr* status register.

3.2 Standard Extensions

3.2.2 “F”, “D”, “Q” Standard Extension

The *fcsr* can be read, written or copied by the CSR instructions:

FRCSR reads *fcsr* into integer register *rd.*

$$rd = fcsr$$

FSCSR swaps *fcsr* by coping into integer register *rd,* then writing a value from integer register rs_1 into *fcsr.*

$$rd = fcsr$$

$$fcsr = rs_1$$

The field within *fcsr* can be accessed individually through CSR addresses:

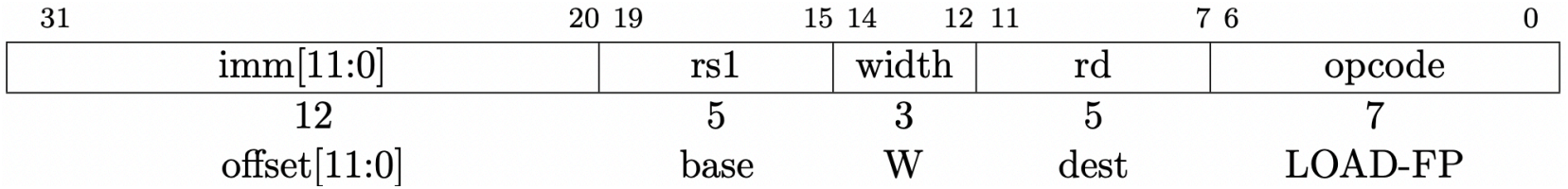
FRRM reads *frm* field and copies it into 3 LSBs of integer register *rd* with zeros in other bits.

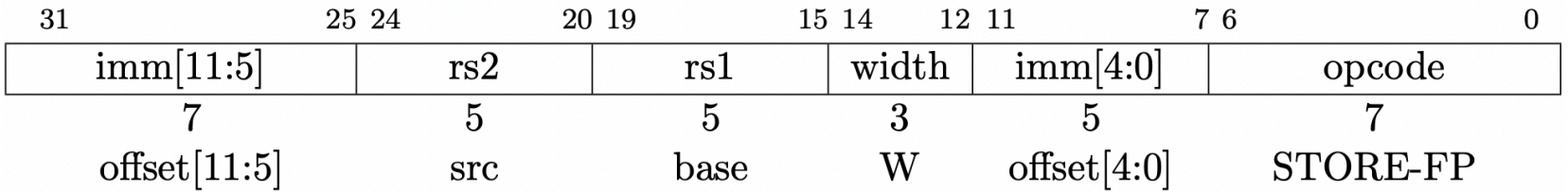
$$rd = zero\,.\,ext(frm)$$

FSRM swaps *frm* by coping into integer register *rd,* then writing a value from the 3 LSBs of integer register rs_1 into *frm* with zeros in other bits.

$$rd = frm$$
$$frm = rs_1[2:0]$$

Load and Store Instruction





FLW loads a single-precision floating-point value from memory into floating-point register *rd*.

$$rd = mem(rs_1 + offset)$$

FSW stores a single-precision floating-point value from floating-point register rs_2 into memory.

$$mem(rs_1 + offset) = rs_2$$

Computational Instructions

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FADD/FSUB	S	src2	src1	RM	dest	OP-FP
FMUL/FDIV	S	src2	src1	RM	dest	OP-FP
FSQRT	S	0	src	RM	dest	OP-FP
FMIN-MAX	S	src2	src1	MIN/MAX	dest	OP-FP

FADD/FMUL performs addition/multiplication between rs_1 *and* rs_2, then write the result into register *rd*.

$$rd = rs_1 \odot rs_2$$

FSUB/FDIV performs subtraction/division of rs_1 *by* rs_2, then write the result into register *rd*.

$$rd = rs_1 \odot rs_2$$

Computational Instructions

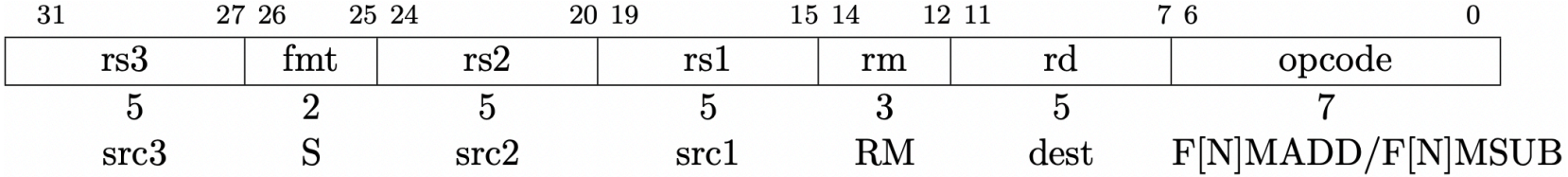
31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FADD/FSUB	S	src2	src1	RM	dest	OP-FP
FMUL/FDIV	S	src2	src1	RM	dest	OP-FP
FSQRT	S	0	src	RM	dest	OP-FP
FMIN-MAX	S	src2	src1	MIN/MAX	dest	OP-FP

FSQRT computes the square root of rs_1, then write the result into register *rd*.

FMIN/FMAX write the minimum/maximum of rs_1 and rs_2 into register *rd*.

* The value -0.0 is considered to be less than $+0.0$. If the both inputs are NaN, the result is canonical NaN.

Computational Instructions



Floating-point fused multiply-add instructions requires a new standard instruction format. A R4-type specifies three source registers, rs_1, rs_2, rs_3, and a destination register *rd*.

FMADD/FMSUB multiplies the value in rs_1 *and* rs_2, adds/subtracts the value in rs_3, then write the result into register *rd*.

$$rd = (rs_1 \times rs_2) \pm rs_3$$

FNMADD/FNMSUB multiplies the value in rs_1 *and* rs_2, negates the product, subtracts/adds the value in rs_3, then writes the result into register *rd*.

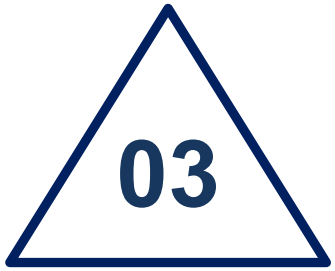
$$rd = -(rs_1 \times rs_2) \mp rs_3$$

3.2 Standard Extensions

3.2.3 “A” Standard Extension

3.2 Standard Extensions

3.2.4 “C” Standard Extension



Extension Not Covered

Brief introduce other extensions.

3.2 Extension Not Covered

References

[1] (2020) The RISC-V Instruction Set Manual, Volume I - Unprivileged ISA.

[2] (2020) The RISC-V Instruction Set Manual, Volume II - Privileged ISA.

[3] (2018) RISCV - An Overview of the Instruction Set Architecture.

THANKS
for your
ATTENTION