Computer Structure and Language

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

1

RISC-V Base & Extensions

- To support a wide range of scientific and industrial use cases, RISC-V ISA is divided into Bases & standard Extensions
- Each processor has exactly one base and an arbitrary number of extensions
- There are also privileged instructions; these instructions are necessary for an operating system
- ISA could be extended beyond the standard as well, if a designer or vendor finds it necessary
- Currently there are 2 ratified bases and 8 ratified extensions (4 bases and 16 extensions in total; including frozen and draft)

2

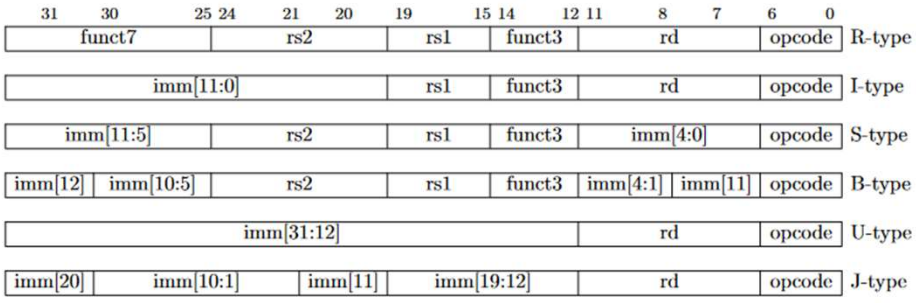
RV32I

- 32-bit base integer instruction set
- Contains 40 instructions
- 32-bit registers and address space
- 32-bit long Instructions
- Covers most important operations
- Can emulate almost any other operation (Except privileged and atomic)

3

RV32I

- Has 6 instruction formats



4

RISC-V Standard Extensions

- There are 8 ratified standard extensions (as of Dec. 2023)
- Each extension provides a specific functionality that is missing from base and other extensions
- Not all extensions are as useful as others
- Most useful extensions are grouped into “G” or “GC” subset
  - “M” for integer multiplication and division
  - “F” & “D” for single and double precision floating point operations
  - “A” for atomic operations
  - “C” for compressed instructions

5

RISC-V Standard Extensions

- Extensions are somewhat related to the base
- Some instructions are only available in larger extensions
- Generally they depend on the size of base registers (XLEN)
- For example:
  - In RV32I “mul” instruction from “M” extension multiplies two 32 bit numbers and stores lower 32 bits of result
  - In RV64I same instruction multiplies two 64 bit numbers and stores lower 64 bits of result
  - To get same functionality in RV64I “mulw” should be used (which does not exist with RV32I base)
- In rest of class extensions are explained in the context of RV32I base

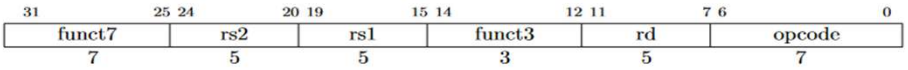
6

"M" EXTENSION

7

Instructions Formant

Integer Register-Immediate Instructions



Inst	Name	Opcode	funct7	funct3	
mul	MUL	0110011	0x01	0x0	rd = (rs1 * rs2)[31:0]
mulh	MUL High	0110011	0x01	0x1	rd = (rs1 * rs2)[63:32]
mulhsu	MUL High (S) (U)	0110011	0x01	0x2	rd = (rs1 * rs2)[63:32]
mulhu	MUL High (U)	0110011	0x01	0x3	rd = (rs1 * rs2)[63:32]
div	DIV	0110011	0x01	0x4	rd = rs1 / rs2
divu	DIV (U)	0110011	0x01	0x5	rd = rs1 / rs2
rem	Remainder	0110011	0x01	0x6	rd = rs1 % rs2
remu	Remainder (U)	0110011	0x01	0x7	rd = rs1 % rs2

8

Multiplication

- There is no instructions to get both high and low 32 bits of result (unlike s370 & x86)
- "**mul**" instructions multiplies values of source registers and stores lower 32 bits in destination registers
- "**mulh**", "**mulhu**" & "**mulhsu**" multiply value of source registers and store higher 32 bits of result in destination register.
  - mulh is used when both sources are signed
  - mulhu is used when both sources are unsigned
  - mulhsu is used when rs1 is signed and rs2 is unsigned
- Why mul doesn't care about sign?

9

Multiplication

- If both the high and low bits of the same product are required, then the recommended code sequence is: MULH[[S]U] rdh, rs1, rs2; MUL rdl, rs1, rs2 (source register specifiers must be in same order and rdh cannot be the same as rs1 or rs2). Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

10

Division

- There is no instructions to get both quotient and remainder (unlike s370 & x86)
- "**div**" & "**divu**" instructions divide rs1 by rs2 and save quotient in rd (signed and unsigned division respectively)
- "**rem**" & "**remu**" instructions divide rs1 by rs2 and save remainder in rd (signed and unsigned division respectively)

11

Division

- If both the quotient and remainder are required from the same division, the recommended code sequence is: DIV[U] rdq, rs1, rs2; REM[U] rdr, rs1, rs2 (rdq cannot be the same as rs1 or rs2). Microarchitectures can then fuse these into a single divide operation instead of performing two separate divides.
- Signed division overflow occurs only when the most-negative integer is divided by −1
- Results of division by zero and overflow are as follows

Condition	Dividend	Divisor	DIVU[W]	REMU[W]	DIV[W]	REM[W]
Division by zero	x	0	$2^L - 1$	x	-1	x
Overflow (signed only)	-2^{L-1}	-1	–	–	-2^{L-1}	0

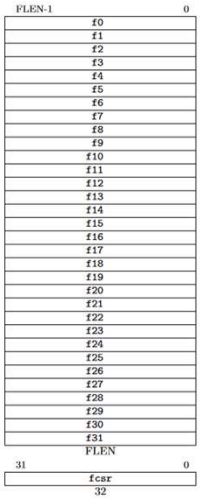
12

"F" EXTENSION

13

Registers

- F Extensions add 32, 32-bit floating point registers (f0-f31) and fcsr (control and status register)



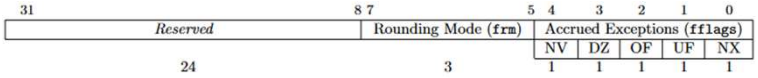
14

F Extension

- F extension instructions are very similar to base instructions
- We get special load, store and arithmetic instructions
- 26 instructions in total
- NaNs are canonical NaN (positive sign, only MSB equal to 1 from fraction bits)

15

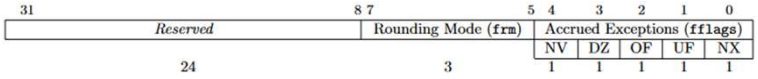
FCSR



- FRCSR & FSCSR pseudoinstructions can read and write fcsr
- FRCSR reads fcsr by copying it into integer register rd.
- FSCSR swaps the value in fcsr by copying the original value into integer register rd, and then writing a new value obtained from integer register rs1 into fcsr

16

FCSR



The fields within the fcsr can also be accessed individually through different CSR addresses, and separate assembler pseudoinstructions are defined for these accesses. The FRRM instruction reads the Rounding Mode field frm and copies it into the least-significant three bits of integer register rd, with zero in all other bits. FSRM swaps the value in frm by copying the original value into integer register rd, and then writing a new value obtained from the three least-significant bits of integer register rs1 into frm. FRFLAGS and FSFLAGS are defined analogously for the Accrued Exception Flags field fflags.

17

Rounding Modes

- Rounding mode can be selected statically or dynamically
- Static rounding mode is encoded in instruction itself
- Dynamic rounding mode is encoded in fcsr
- 3 bits are used for encoding rounding mode (both in instruction and fcsr). Selecting 111 in instruction, means using dynamic mode.
- If frm is set to an invalid value (101–111), any subsequent attempt to execute a floating-point operation with a dynamic rounding mode will raise an illegal instruction exception.
- Some instructions, including widening conversions, have the rm field but are nevertheless unaffected by the rounding mode; software should set their rm field to RNE (000).

18

Rounding Modes

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.*

19

Accrued Exception Flag

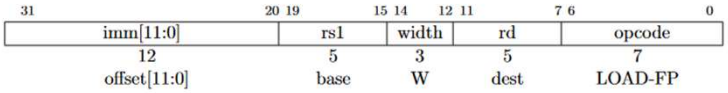
- The base RISC-V ISA does not support generating a trap on the setting of a floating-point exception flag.
- Instead the accrued exception flags indicate the exception conditions that have arisen on any floating-point instructions

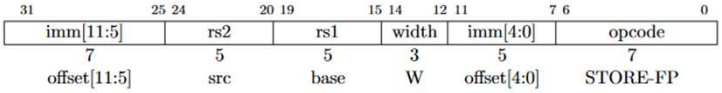
Flag Mnemonic	Flag Meaning
NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

20

Load & Store

- Same addressing mode as integer ISA (base + offset)
- Content of non-canonical NaNs are preserved (no modification to bits)
- Share same format with integer load and store (W is always 010)



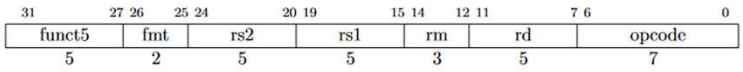


Inst	Opcode	W	Description
flw	0000111	010	Load a s.p. float to rd
fsw	0100111	010	Store s.p. float from rs2 to memory

21

Arithmetic Instructions

- Floating-point arithmetic instructions with one or two source operands use the R-type format with the OP-FP major opcode.
- The 2-bit floating-point format field fmt is always set to S (00) for all F instructions
- All floating-point operations that perform rounding can select the rounding mode using the rm field

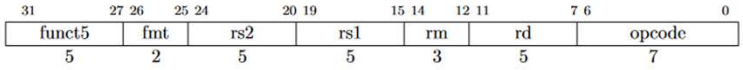


Inst	Opcode	funct5	Description
fadd.s	1010011	0000000	rd = rs1 + rs2
fsub.s	1010011	0000100	rd = rs1 – rs2
fmul.s	1010011	0001000	rd = rs1 * rs2
fdiv.s	1010011	0001100	rd = rs1 / rs2

22

Arithmetic Instructions

- **fsqrt.s** only takes one source, rs2 should be set to 0
- **fmin.s** and **fmax.s**
  - consider -0.0 less than +0.0
  - If both inputs are NaN, output is canonical NaN
  - If one input is NaN, out is the non-NaN value.

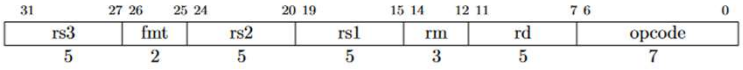


Inst	Opcode	funct5	Description
fsqrt.s	1010011	0101100	rd = sqrt(rs1)
fmin.s	0010100	0000100	rd = min(rs1, rs2)
fmax.s	0010100	0001000	rd = max(rs1, rs2)

23

Arithmetic Instructions – Fused Multiply, Add

- Take three sources
- Different encoding and opcodes
- Because rs3 replaces the funct5, only the opcodes distinguish them

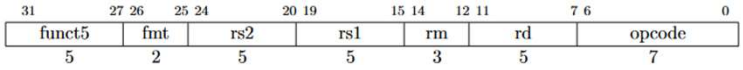


Inst	Opcode	Description
fmadd.s	1000011	rd = (rs1 x rs2) + rs3
fmsub.s	1001011	rd = (rs1 x rs2) - rs3
fnmsub.s	1001111	rd = -(rs1 x rs2) + rs3

24

Conversion and Move Instructions - Convert

- Convert values of float registers and move them o integer registers or vice versa.



Inst	Opcode	funct5	rs2	Description
fcvt.w.s	1010011	1100000	00000	rd(int) = float2int(rs1(float))
fcvt.wu.s	1010011	1100000	00001	rd(int) = float2uint(rs1(float))
fcvt.s.w	1010011	1101000	00000	rd(float) = int2float(rs1(int))
fcvt.s.wu	1010011	1101000	00001	rd(float) = uint2float(rs1(int))

25

Conversion and Move Instructions - Convert

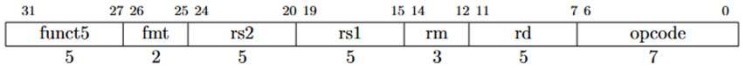
- If the rounded result is not representable in the destination format, it is clipped to the nearest value and the invalid flag is set. (valid inputs and behavior on invalid input is as follows)
- All floating-point to integer and integer to floating-point conversion instructions round according to the rm field.
- A floating-point register can be initialized to floating-point positive zero using FCVT.S.W rd, x0, which will never set any exception flags.

	FCVT.W.S	FCVT.WU.S	FCVT.L.S	FCVT.LU.S
Minimum valid input (after rounding)	-2^{31}	0	-2^{63}	0
Maximum valid input (after rounding)	$2^{31}-1$	$2^{32}-1$	$2^{63}-1$	$2^{64}-1$
Output for out-of-range negative input	-2^{31}	0	-2^{63}	0
Output for $-\infty$	-2^{31}	0	-2^{63}	0
Output for out-of-range positive input	$2^{31}-1$	$2^{32}-1$	$2^{63}-1$	$2^{64}-1$
Output for $+\infty$ or NaN	$2^{31}-1$	$2^{32}-1$	$2^{63}-1$	$2^{64}-1$

26

Conversion and Move Instructions - Sign Injection

- Take two floating point registers as source.
- Take all bits except sign from rs1
- Sign is selected based on following table
- Only rm differs between these instructions

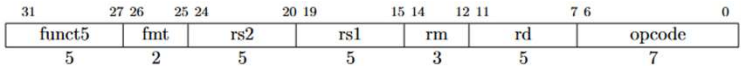


Inst	Opcode	funct5	rm	Description
fsgnj.s	1010011	0010000	000	Sign = sign(rs2)
fsgnjn.s	1010011	0010000	001	Sign = -sign(rs2)
fsgnjx.s	1010011	0010000	010	Sign = sign(rs1) xor sign(rs2)

27

Conversion and Move Instructions - Move

- Move instructions only move bits from integer to float registers or vice versa.
- They do not convert nor modify the bits (content of non canonical NaNs are preserved)

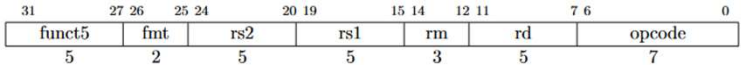


Inst	Opcode	funct5	rs2	rm	Description
fmv.x.w	1010011	1110000	00000	000	Move rs1(float) to rd(int)
fmv.w.x	1010011	1111000	00000	000	Move rs1(int) to rd(float)

28

Compare Instructions

- Compare rs1 & rs2, set rd based on result
- rd = 1 on true condition, rd = 0 on false condition
- rs1 & rs2 are floating point registers, rd is an integer register
- rm chooses which comparison is used



Inst	Opcode	funct5	rm	Description
feq.s	1010011	1010000	010	rd = (rs1 == rs2) ? 1 : 0
flt.s	1010011	1010000	001	rd = (rs1 < rs2) ? 1 : 0
fle.s	1010011	1010000	000	rd = (rs1 <= rs2) ? 1 : 0

29

Classify Instruction

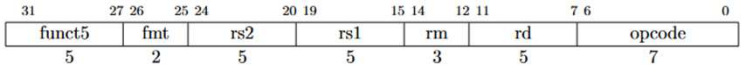
- The fclass.s instruction examines the value in floating-point register rs1 and writes to integer register rd a 10-bit mask that indicates the class of the floating-point number.
- The corresponding bit in rd will be set if the property is true and clear otherwise. All other bits in rd are cleared.
- Exactly one bit in rd will be set.
- fclass.s does not set the floating-point exception flags.

rd bit	Meaning
0	*rs1* is $-\infty$.
1	*rs1* is a negative normal number.
2	*rs1* is a negative subnormal number.
3	*rs1* is -0.
4	*rs1* is $+0$.
5	*rs1* is a positive subnormal number.
6	*rs1* is a positive normal number.
7	*rs1* is $+\infty$.
8	*rs1* is a signaling NaN.
9	*rs1* is a quiet NaN.

30

Classify Instruction

- fclass.s has only one source, rs2 must be set to 0.
- rm must be set to 001(binary)



Inst	Opcode	funct5	rm	rs2	Description
fclass.s	1010011	1110000	001	00000	Sets rd based on rs1

31

“D” EXTENSION

32

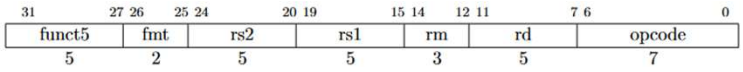
D Extension

- D extension relies on F extension and can not be included without F.
- Extends all 32 floating-point registers to 64-bit registers. (registers can either hold 32 or 64 bit floating-point values)
- All F instructions are also available in D
  - fmt field must be set to D(01)
  - Mnemonics swap s for d

33

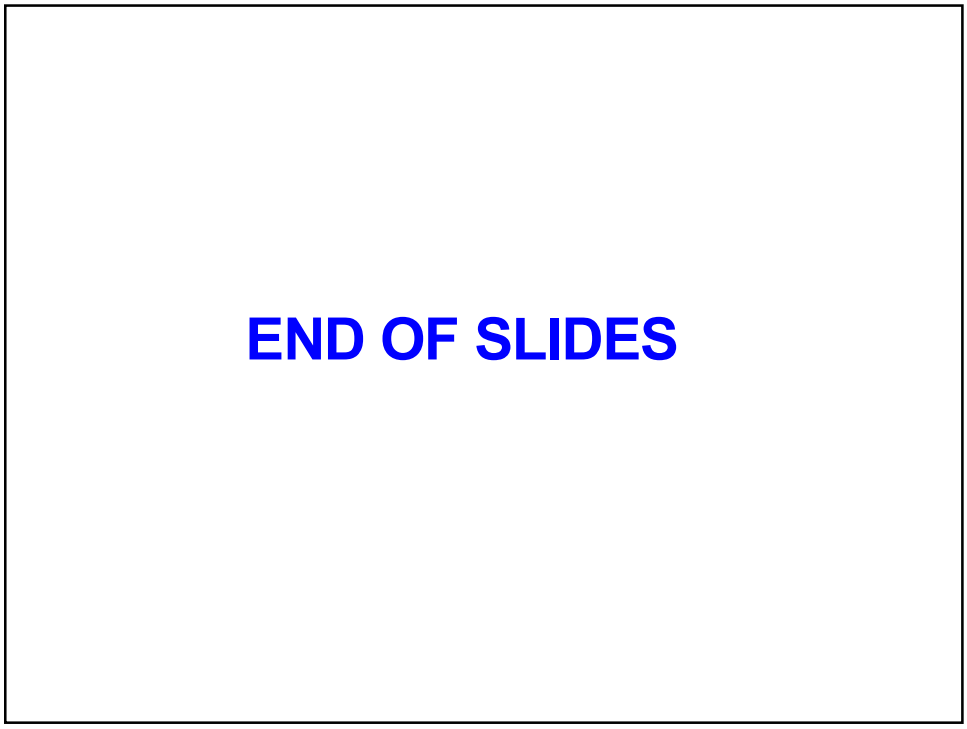
Conversion Instructions – Single Precision to Double Precision

- Convert values of float registers between single and double precision representations.
- Has only one source. rs2 must be set accordingly.



Inst	Opcode	funct5	rs2	Description
fcvt.s.d	1010011	0100000	00001	Double to single conversion
fcvt.d.s	1010011	0100001	00000	Single to double conversion

34



35