RISC-V Instruction Set Architecture I

Muhammad Tahir

Lecture 3-4

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

Competing ISAs

- x-86 Architecture – CISC, register-memory, used in desktop computers, servers, proprietary
- ARM Architecture – RISC, register-register, majority of embedded and cellular market, proprietary
- RISC-V Architecture – RISC, register-register, upcoming, open-source

RISC-V ISA

- RISC-V is a modular architecture requiring one base architecture (mandatory) that is extendable using multiple extensions (optional)

Table 1: RISC-V `Base` architectures.

ISA	Description
RV32I	Base Integer Instruction Set, 32-bit
RV64I	Base Integer Instruction Set, 64-bit
RV128I	Base Integer Instruction Set, 128-bit
RV32E	Base Integer Instruction Set for Embedded Systems, 32-bit (Subset of RV32I)

RISC-V ISA Cont'd

Table 2: List of RISC-V ISA `extensions`.

Extension	Description
M	Standard Extension for Integer Multiplication, Division
A	Standard Extension for Atomic Instructions
F	Standard Extension for Single-Precision Floating-Point
D	Standard Extension for Double-Precision Floating-Point
Q	Standard Extension for Quad-Precision Floating-Point
L	Standard Extension for Decimal Floating-Point
C	Standard Extension for Compressed Instructions
B	Standard Extension for Bit Manipulation
J	Standard Extension for Dynamically Translated Languages
T	Standard Extension for Transactional Memory
P	Standard Extension for Packed-SIMD Instructions
V	Standard Extension for Vector Operations
N	Standard Extension for User-Level Interrupts

RISC-V ISA Cont'd

- `Instruction Encoding` ∼ variable length, e.g. RV32IC will have 32-bit base and 16-bit compressed extension
- `Addressing Modes` ∼ four addressing modes, register, immediate, displacement (including PC-relative)
- `Instruction Operands` ∼ number of operands can be from zero to three

RV32I Registers

- General Purpose Registers (GPRs) (32 registers)
  - x0, x1, ..., x31
  - 32-bit wide integer registers
  - x0 is hard-wired to zero
- Control and Status Registers (CSRs)
  - User Level CSRs
  - Supervisor Level CSRs
  - Hypervisor Level CSRs
  - Machine Level CSRs
- Program counter (PC)

RV32I Instruction Set

- Three groups of instructions and corresponding tree
  - Data Processing Instructions
    - Register-Register instructions (R-type)
    - Register-immediate instructions (I-type)
    - Register-immediate instructions (U-type)

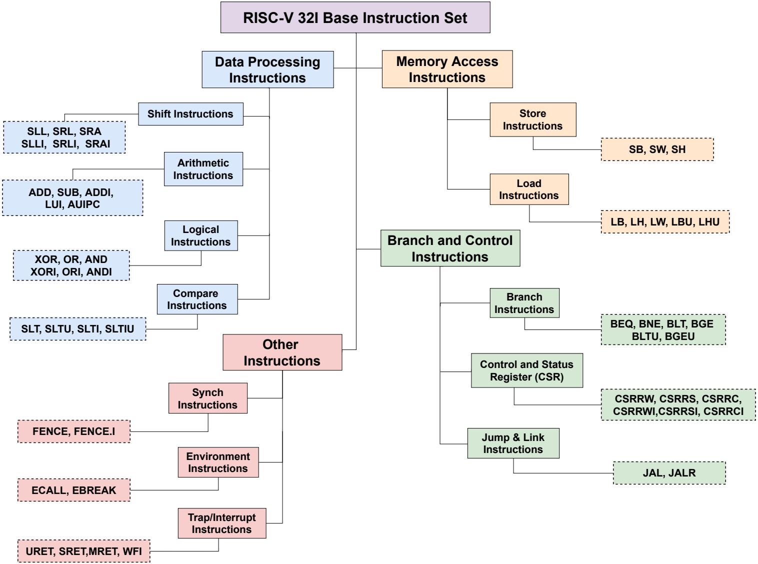
RV32I Instruction Set

- Three groups of instructions and corresponding tree
    - Data Processing Instructions
        - Register-Register instructions (R-type)
        - Register-immediate instructions (I-type)
        - Register-immediate instructions (U-type)
    - Memory Access Instructions
        - Load (I-type)
        - Store (S-type)

RV32I Instruction Set

- Three groups of instructions and corresponding tree
  - Data Processing Instructions
    - Register-Register instructions (R-type)
    - Register-immediate instructions (I-type)
    - Register-immediate instructions (U-type)
  - Memory Access Instructions
    - Load (I-type)
    - Store (S-type)
  - Flow Control Instructions
    - Unconditional jump and link (J-type)
    - Unconditional jump via register and link (I-type)
    - Conditional branches (B-type)

RV32I Instruction Set Cont'd



RISC-V Base Instruction Formats: R, I, S, U

- R Format

func7	rs2	rs1	func3	rd	opcode

- I Format

imm[11:0]	rs1	func3	rd	opcode

- S Format

imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode

- U Format

imm[31:12]	rd	opcode

Opcode field   Operand field

RISC-V Base Instruction Formats: B Format

- Similar to S Format
    - S Format

imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode

- B Format

imm[12:10:5]	rs2	rs1	func3	imm[4:1:11]	opcode

RISC-V Base Instruction Formats: J Format

- Similar to U Format
  - U Format

imm[31:12]	rd	opcode

  - J Format

imm[20]:imm[10:1]:imm[11]:imm[19:12]	rd	opcode

Data Processing Instructions: R Type

- Format: R-type

7	5	5	3	5	7
func7	rs2	rs1	func3	rd	opcode

Table 3: R type data processing instructions.

Instruction	Operation	Type	Illustration
SLL	Shift Left Logical	R	SLL rd,rs1,rs2
SRL	Shift Right Logical	R	SRL rd,rs1,rs2
SRA	Shift Right Arithmetic	R	SRA rd,rs1,rs2
ADD	Addition	R	ADD rd,rs1,rs2
SUB	Subtraction	R	SUB rd,rs1,rs2
XOR	XOR	R	XOR rd,rs1,rs2
OR	OR	R	OR rd,rs1,rs2
AND	AND	R	AND rd,rs1,rs2

Data Processing Instructions: R Type Cont'd

Table 4: R type data processing instructions.

Instruction	Operation	Type	Illustration
MUL	Multiply	R	MUL rd,rs1,rs2
MULH	Multiply upper Half	R	MULH rd,rs1,rs2
MULHSU	Multiply Half Sign/Uns	R	MULHSU rd,rs1,rs2
MULHU	Multiply upper Half Uns	R	MULHU rd,rs1,rs2
DIV	Divide	R	DIV rd,rs1,rs2
DIVU	Divide Unsigned	R	DIVU rd,rs1,rs2

- Format: R-type

7	5	5	3	5	7
func7	rs2	rs1	func3	rd	opcode

- The destination register: `rd` $\leftarrow$ rs1(func3, func7)rs2

Data Processing Instructions: I Type

- Format: I-type

12	5	3	5	7
imm[11:0]	rs1	func3	rd	opcode

Table 5: I type data processing instructions.

Instruction	Operation	Type	Illustration
ADDI	Add immediate	I	ADDI rd, rs1, imm
ANDI	AND immediate	I	ANDI rd, rs1, imm
ORI	OR immediate	I	ORI rd, rs1, imm
XORI	XOR immediate	I	XORI rd, rs1, imm
SLTI	Set less than immediate	I	SLTI rd, rs1, imm
SLTIU	Set less than immediate unsigned	I	SLTIU rd, rs1, imm

Data Processing Instructions: I Type Cont'd

Table 6: I type data processing instructions.

Instruction	Operation	Type	Illustration
SLLI	Shift left logical immediate	I	SLLI rd, rs1, shamt
SRLI	Shift right logical immediate	I	SRLI rd, rs1, shamt
SRAI	Shift right arithmetic immediate	I	SRAI rd, rs1, shamt

- Format: I-type

7 : 5	5	3	5	7
imm[11:5] : imm[4:0]	rs1	func3	rd	opcode

- The shift amount: imm[4:0] $\leftarrow$ `shamt`

Data Processing Instructions: U Type

- Format: U-type

20	5	7
imm[31:12]	rd	opcode

Table 7: U type data processing instructions.

Instruction	Operation	Type	Illustration
LUI	Load Upper Immediate	U	LUI rd, imm
AUIPC	Add Upper Immediate to PC	U	AUIPC rd, imm

- `U-imm` = imm[31:12] : 12'b0
- `opcode` = LUI : rd $\leftarrow$ U-imm
- `opcode` = AUIPC : rd $\leftarrow$ pc + U-imm

Memory Access Instructions: I Type

- Format: I-type

12	5	3	5	7
imm[11:0]	rs1	func3	rd	opcode

Table 8: I type memory access instructions.

Instruction	Operation	Type	Illustration
LB	Load Byte	I	LB rd, rs1, imm
LH	Load Halfword	I	LH rd, rs1, imm
LW	Load Word	I	LW rd, rs1, imm
LBU	Load Byte Unsigned	I	LBU rd, rs1, imm
LHU	Load Half Unsigned	I	LHU rd, rs1, imm

Memory Access Instructions: I Type Cont'd

- Format: I-type

12	5	3	5	7
imm[11:0]	rs1	func3	rd	opcode

- `I-imm` = signExtend(imm[11:0])
- `opcode` = LOAD: rd $\leftarrow$ mem[rs1 + I-imm]
- `funct3` = LW/LB/LBU/LH/LHU

Memory Access Instructions: S Type

- Format: S-type

7	5	5	3	5	7
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode

Table 9: S type memory access instructions.

Instruction	Operation	Type	Illustration
SB	Store Byte	I	SB rs1, rs2, imm
SH	Store Halfword	I	SH rs1, rs2, imm
SW	Store Word	I	SW rs1, rs2, imm

Memory Access Instructions: S Type Cont'd

- Format: S-type

7	5	5	3	5	7
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode

- `S-imm` = signExtend(imm[11:5], imm[4:0])
- `opcode` = STORE: mem[rs1 + S-imm] $\leftarrow$ rs2
- `func3` = SW/SB/SH

Memory Access Instructions: Addressing Modes

- PC-relative Addressing: Memory address is the sum of `PC` and an offset (e.g. auipc, jal and branch instructions). The offset is specified in the instruction.
- Register-offset Addressing: Memory address is the sum of a `Register` and an offset (e.g. jalr, addi and all memory instructions). The offset is specified in the instruction.
- Absolute Addressing: Memory address is a constant (32-bit) (e.g the lui instruction). Though this addressing can be interpreted as register-offset addressing with x0 being the register (x0-offset).

Flow Control Instructions: J Type

- Format: J-type

1 : 10 : 1 : 8	5	7
imm[20]:imm[10:1]:imm[11]:imm[19:12]	rd	opcode

Table 10: J type flow control instructions.

Instruction	Operation	Type	Illustration
JAL	Jump and Link	J	JAL rd, Label

- `J-imm` = signExtend(inst[31], inst[19:12], inst[20], inst[30:21], 1'b0)
- `opcode` = JAL: rd $\leftarrow$ pc + 4;
  pc $\leftarrow$ pc + J-imm (= Label)
- `Jump` = $\pm$1MB range

Flow Control Instructions: I Type

- Format: I-type

12	5	3	5	7
imm[11:0]	rs1	func3	rd	opcode

Table 11: I type flow control instructions.

Instruction	Operation	Type	Illustration
JALR	Jump and Link	I	JALR rd, imm(rs1)

- `I-imm` = signExtend(imm[11:0])
- `opcode` = JALR: rd $\leftarrow$ pc + 4;
  pc $\leftarrow$ ((rs1 + I-imm) & ~(0x01))

Flow Control Instructions: B Type

- Conditional branches: B-type

1 : 6	5	5	3	4 : 1	7
imm[12:10:5]	rs2	rs1	func3	imm[4:1:11]	opcode

Table 12: B type flow control instructions.

Instruction	Operation	Type	Illustration
BEQ	Branch Equal	B	BEQ rs1, rs2, imm
BNE	Branch Unequal	B	BNE rs1, rs2, imm
BLT	Branch Lessthan	B	BLT rs1, rs2, imm
BGE	Branch Greater equal	B	BGE rs1, rs2, imm
BLTU	Branch Lessthan Unsigned	B	BLTU rs1, rs2, imm
BGEU	Branch Greater equal Unsigned	B	BGEU rs1, rs2, imm

Flow Control Instructions: B Type Cont'd

- Format: B-type

1 : 6	5	5	3	4 : 1	7
imm[12:10:5]	rs2	rs1	func3	imm[4:1:11]	opcode

- `B-imm` = signExtend(inst[31], inst[7], inst[30:25], inst[11:8], 1'b0)
- `opcode` = BRANCH:
  pc $\leftarrow$ compare(funct3, rs1, rs2) ? pc + B-imm : pc + 4
- `funct3` = BEQ/BNE/BLT/BLTU/BGE/BGEU

Suggested Reading

- Read relevant sections of Chapter 2 of [Patterson and Hennessy, 2021].
- Read User Manual for the instruction set and its architecture [Waterman et al., 2016].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Patterson, D. and Hennessy, J. (2021).
Computer Organization and Design RISC-V Edition: The Hardware Software Interface, 2nd Edition.
Morgan Kaufmann.

Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2016).
The risc-v instruction set manual, volume i: User-level isa, version 2.1.
EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129.