Redistribution without instructor's consent is prohibited.

N26F300
VLSI SYSTEM DESIGN
(GRADUATE LEVEL)

Fall 2022

RISC-V (I) – ISA Basics

Outline

- History of RISC-V
- Instruction Set – R-type, I-type, S-type
- Instruction Set – J-type

Redistribution without instructor's consent is prohibited.

History of RISC-V

Source: Computer Organization and Design (RISC-V ed.)

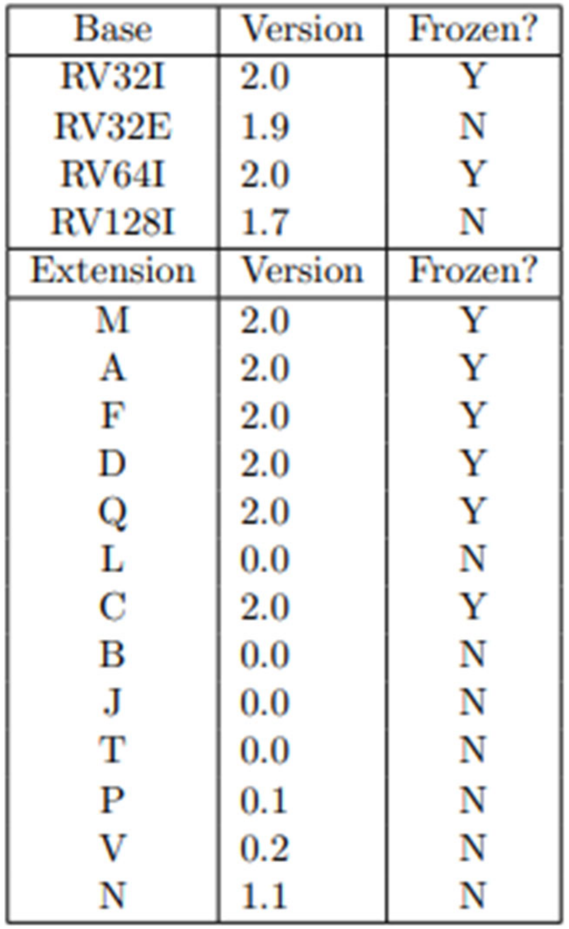
Redistribution without instructor's consent is prohibited.

RISC-V Instruction Set

- Developed at UC Berkeley as open ISA
- Now managed by the RISC-V Foundation (riscv.org)
- Typical of many modern ISAs
- Similar ISAs have a large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …

Redistribution without instructor's consent is prohibited.

RISC-V ISA Modules



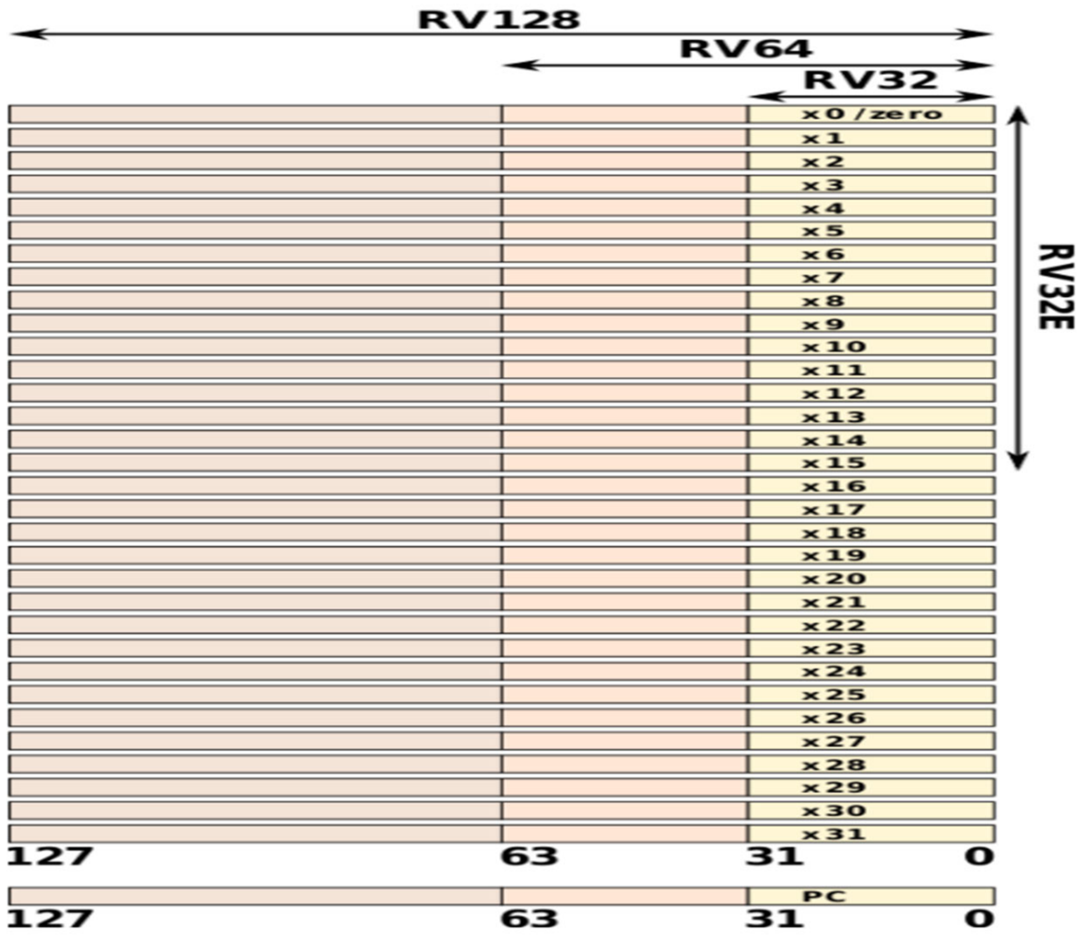
Base	Version	Frozen?
RV32I	2.0	Y
RV32E	1.9	N
RV64I	2.0	Y
RV128I	1.7	N
Extension	**Version**	**Frozen?**
M	2.0	Y
A	2.0	Y
F	2.0	Y
D	2.0	Y
Q	2.0	Y
L	0.0	N
C	2.0	Y
B	0.0	N
J	0.0	N
T	0.0	N
P	0.1	N
V	0.2	N
N	1.1	N

I: Integer; E (embedded)

M: Integer Multiplication and Division
A: Atomic Instructions
F: Single-Precision Floating-Point (FP)
D: Double-Precision FP
G: base integer set (MAFD)
Q: Quad-Precision FP
L: Decimal FP
C: Compressed Instructions
B: Bit Manipulation
J: Dynamically Translated Langurages
T: Transactional Memory
P: Packed-SIM Instruction
V: Vector Operations
N: User-Level Interrupts

Redistribution without instructor's consent is prohibited.

User-Level Base Integer Registers



Source: https://en.wikichip.org/wiki/risc-v/registers

Redistribution without instructor's consent is prohibited.

Instruction Set

- R-type Operations
- I-type Operations
- S-type Operations

Source: Computer Organization and Design (RISC-V ed.)

Redistribution without instructor's consent is prohibited.

Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c  // a gets b + c
```

- All arithmetic operations have this form
- *Design Principle 1:* Simplicity favours regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Arithmetic Example

- C code:

```
f = (g + h) - (i + j);
```

- Compiled RISC-V code:

```
add t0, g, h   // temp t0 = g + h
add t1, i, j   // temp t1 = i + j
sub f, t0, t1  // f = t0 - t1
```

Redistribution without instructor's consent is prohibited.

Register Operands

- Arithmetic instructions use register operands

- RISC-V has a 32 × 32-bit register file (RV32I)
  - Use for frequently accessed data
  - 32-bit data is called a “word”
    - 32 x 32-bit general purpose registers x0 to x30
  - 64-bit data is called a “doubleword” for extension “D”

Redistribution without instructor's consent is prohibited.

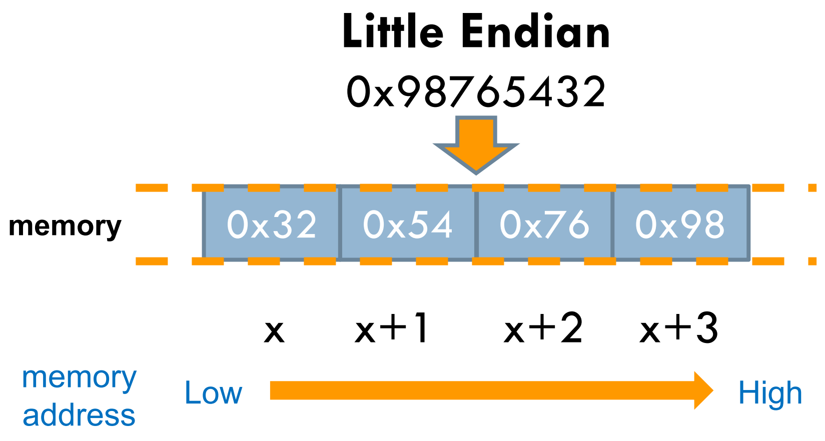
RISC-V Registers

- x0: the constant value 0
- x1: return address
- x2: stack pointer
- x3: global pointer
- x4: thread pointer
- x5 – x7, x28 – x31: temporaries
- x8: frame pointer
- x9, x18 – x27: saved registers
- x10 – x11: function arguments/results
- x12 – x17: function arguments

Redistribution without instructor's consent is prohibited.

Memory Operands

- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory
- Memory is byte addressed
  - Each address identifies an 8-bit byte
- RISC-V is Little Endian
  - Least-significant byte at the least address of a word (lowest address #)
  - *c.f.* Big Endian: most-significant byte at least address
- RISC-V does not require words to be aligned in memory



Memory Operand Example

- C code:

```
A[12] = h + A[8];
```

  - h in x21, base address of A in x22
- Compiled RISC-V code:
  - Index 8 requires offset of 64
    - 8 bytes per doubleword

```
ld      x9, 64(x22)
add     x9, x21, x9
sd      x9, 96(x22)
```

Redistribution without instructor's consent is prohibited.

Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Redistribution without instructor's consent is prohibited.

Immediate Operands

- Constant data specified in an instruction

```
addi x22, x22, 4
```

- Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction

Redistribution without instructor's consent is prohibited.

Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- Replicate the sign bit to the left
  - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110
- In RISC-V instruction set
  - `lb`: sign-extend loaded byte
  - `lbu`: zero-extend loaded byte

Redistribution without instructor's consent is prohibited.

Representing Instructions

- Instructions are encoded in binary
  - Called machine code
- RISC-V instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Regularity!

Redistribution without instructor's consent is prohibited.

Hexadecimal

- Base 16
  - Compact representation of bit strings
  - 4 bits per hex digit

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

- Example: eca8 6420
  - 1110 1100 1010 1000 0110 0100 0010 0000

Redistribution without instructor's consent is prohibited.

RISC-V R-type Instructions

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- Instruction fields
  - opcode: operation code
  - rd: destination register number
  - funct3: 3-bit function code (additional opcode)
  - rs1: the first source register number
  - rs2: the second source register number
  - funct7: 7-bit function code (additional opcode)

Redistribution without instructor's consent is prohibited.

Example of R-type Instructions

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

SLT and SLTU: signed and unsigned comparison; write 1 to *rd* if *rs1* < *rs2*, 0 otherwise.
SLL, SRL, and SRA: logical left, logical right, arithmetic right shifts on the value in *rs1* by the shift amount in the lower 5 bits of register *rs2*

Redistribution without instructor's consent is prohibited.

R-type Example

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

```
add x9,x20,x21
```

0	21	20	0	9	51

0000000	10101	10100	000	01001	0110011

$0000\ 0001\ 0101\ 1010\ 0000\ 0100\ 1011\ 0011_{two} =$
$015A04B3_{16}$

funct7 and *funct3* fields select the type of operation.

Redistribution without instructor's consent is prohibited.

RISC-V I-type Instructions

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

- Immediate arithmetic and load instructions
  - rs1: source or base address register number
  - immediate: constant operand, or offset added to base address
    - 2s-complement, sign extended
- *Design Principle 3:* Good design demands good compromises
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible

Redistribution without instructor's consent is prohibited.

RISC-V S-type Instructions

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- Different immediate format for store instructions
  - rs1: base address register number
  - rs2: source operand register number
  - immediate: offset added to base address
    - Split so that rs1 and rs2 fields always in the same place

Redistribution without instructor's consent is prohibited.

Logical Operations

- Instructions for bitwise manipulation

Operation	C	Java	RISC-V
Shift left	<<	<<	`slli`
Shift right	>>	>>>	`srli`
Bit-by-bit AND	&	&	`and, andi`
Bit-by-bit OR	|	|	`or, ori`
Bit-by-bit XOR	^	^	`xor, xori`
Bit-by-bit NOT	~	~	

- Useful for extracting and inserting groups of bits in a word

Redistribution without instructor's consent is prohibited.

Shift Operations

funct6	immed	rs1	funct3	rd	opcode
6 bits	6 bits	5 bits	3 bits	5 bits	7 bits

- immed: how many positions to shift
- Shift left logical
  - Shift left and fill with 0 bits
  - `slli` by *i* bits multiplies by 2^i
- Shift right logical
  - Shift right and fill with 0 bits
  - `srli` by *i* bits divides by 2^i (unsigned only)

Redistribution without instructor's consent is prohibited.

AND Operations

- Useful to mask bits in a word
  - Select some bits, clear others to 0

```
and x9,x10,x11
```

```
x10  00000000 00000000 00000000 00000000 00000000 00000000 00001101 11000000
x11  00000000 00000000 00000000 00000000 00000000 00000000 00111100 00000000
x9   00000000 00000000 00000000 00000000 00000000 00000000 00001100 00000000
```

Redistribution without instructor's consent is prohibited.

OR Operations

- Useful to include bits in a word
  - Set some bits to 1, leave others unchanged

```
or x9,x10,x11
```

```
x10  00000000 00000000 00000000 00000000 00000000 00000000 00001101 11000000
x11  00000000 00000000 00000000 00000000 00000000 00000000 00111100 00000000
x9   00000000 00000000 00000000 00000000 00000000 00000000 00111101 11000000
```

Redistribution without instructor's consent is prohibited.

XOR Operations

- Differencing operation
  - Set some bits to 1, leave others unchanged

```
xor x9,x10,x12  // NOT operation
```

x10	00000000 00000000 00000000 00000000 00000000 00000000 00001101 11000000
x12	11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
x9	11111111 11111111 11111111 11111111 11111111 11111111 11110010 00111111

Redistribution without instructor's consent is prohibited.