計算機結構
Computer Architecture
RISCV Supplementary Material (Inst.)

Speaker: Kuan-Heng Liu

Instructor: Prof. An-Yeu Wu

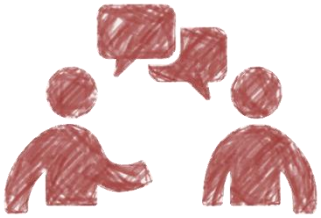
Date: 2025/03/10

Outline

- Introduction of RISC-V
- Comparison of RISC-V and MIPS

Instruction Set Architecture

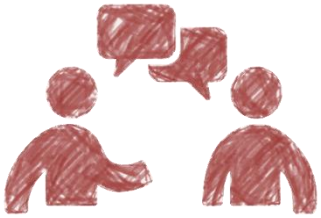
- The interface between hardware and software
  - Complexed Instruction Set Computer (CISC)
  - Reduced Instruction Set Computer (RISC)



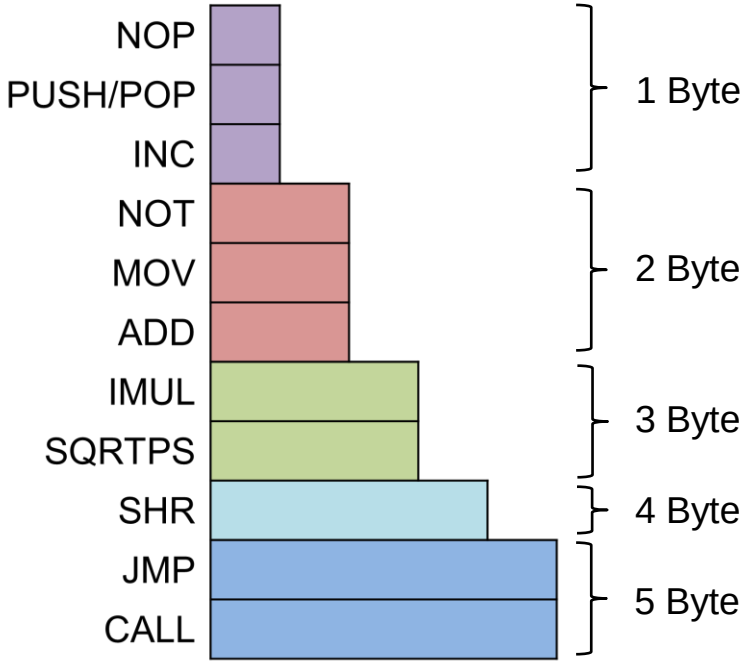
Comparison	CISC	RISC
Instruction count	<100	>200
Instruction length	**Not fixed**	**Fixed**
Area	High	**Low**
Timing	High	**Low**
Program size	**Small**	Large
Instruction cycle	Multiple cycle	Single cycle
Celebrity	x86 (PC)	ARM (Mobile)
Description	Execute 20% instruction in 80% time	As simple as possible

Instruction Set Architecture

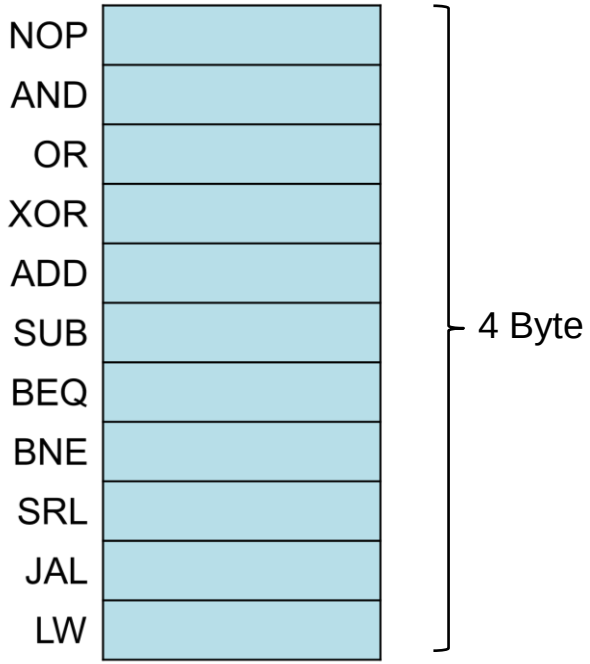
- The interface between hardware and software
  - Complexed Instruction Set Computer (CISC)
  - Reduced Instruction Set Computer (RISC)



CISC (Pentium III)



RISC (RISC-V)



RISC-V: Born of Hope

- Birth: UC Berkeley, Professor Krste Asanovic, 2010
- RISC-V Foundation in 2015
  - Comprises more than 100 member organizations, collaborative community of software and hardware innovators.
- Goals
  - Royalty-free for any purpose
  - Become a standard ISA that connect well in hardware and software.
- Design Concept
  - As Simple As Possible
  - Modularize function blocks
- Award
  - The Linley Group's Analyst's Choice Award for Best Technology (The instruction set, 2017) RISCV


Berkeley Architecture Research
DRAPER
bluespec
NVIDIA
DOVER MICROSYSTEMS
Hewlett Packard Enterprise
Mellanox TECHNOLOGIES
Microsoft
NXP
C-SKY
HUAWEI
cortus
Micron
SAMSUNG
SiFive
Rambus
Cryptography Research
ORACLE
IBM
WD Western Digital
Google
Microsemi
QUALCOMM
RISC-V Foundation: 65+ Members
ESPRESSIF
MEDIATEK
AMD
ANDES TECHNOLOGY
IDT
SEGGER It simply works!
BAE SYSTEMS
Esperanto Technologies
VeriSilicon
LATTICE SEMICONDUCTOR
GRAY RESEARCH
runtime.io
antmicro
codasip
GRIDOOG
SHC SH CONSULTING
INTRINSIX
ETH zürich
TRINAMIC
imperas
ISTUARY
GREENWAVES
PROCESSOR
MIT CSAIL
ROA LOGIC
Rumble Development
lowRISC
UBILITE Hello IoT World
ICT
EMDALO TECHNOLOGIES
EMBECOSM
Sur Technology
Syntacore
Blockstream
VectorBlox
Technolution
ultraSoC
csem
PRINCETON UNIVERSITY
BERKELEY LAB

RISC-V: Basic Design

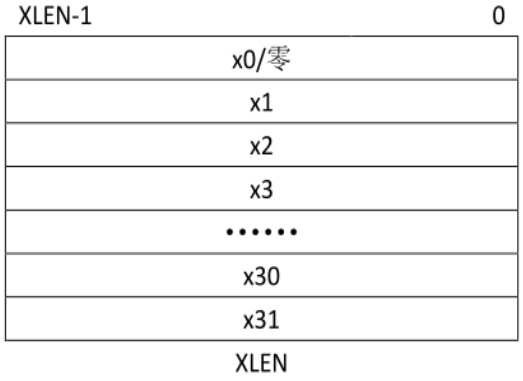
- Base Integer Instruction Set Architecture (ISA)
  - must be present in any implementation

Name	Number	Description
RV32I	47	Including arithmetic, branch, store/load. 32 bits user address space, 32*32-bits registers
RV32E	47	subset of RV32I, 16*32-bits registers ⟸ **Low power**
RV64I	59	64 bits user address space, 32*64-bits registers
RV128I	71	128bits user address space, 32*128-bits registers

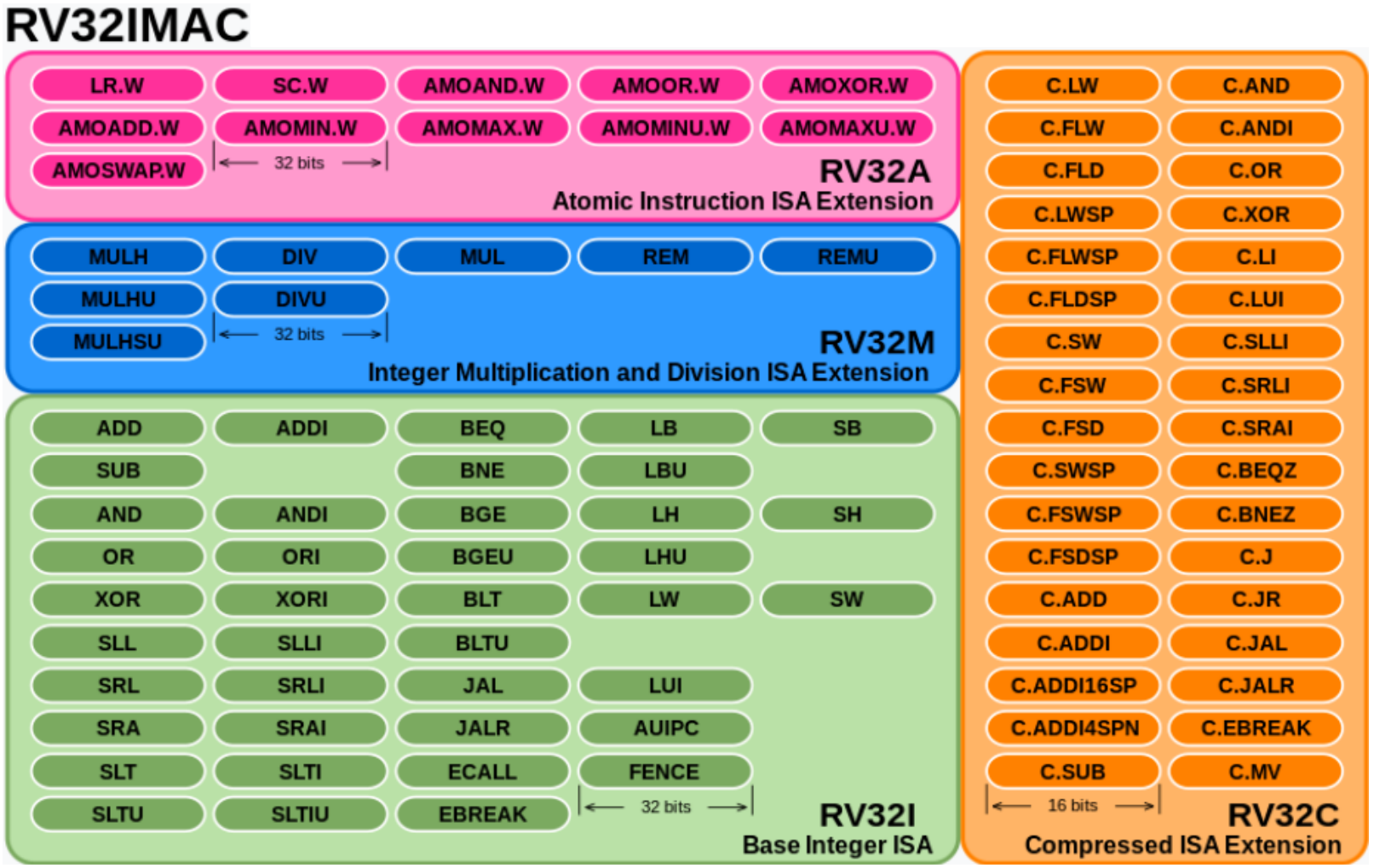
- Standard Extension **(Modular design)**

Name	Number	Description
C	**53**	**16bits compressed instruction**
M	8	Multiplication, division, mod
F	26	Floating type instruction
D	26	Double type instruction

- Integer Registers



RISC-V: Modular Design (Extension)



RISC-V: Register-type Instruction

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register (R-type)	funct7							rs2					rs1					funct3			rd					opcode						

- Register-type Instruction (R-type)
  - Purpose: used for **reg-to-reg ops**, such as arithmetic, logical, and shift
  - Operates solely on registers, **execution without memory access**
- Characteristics
  - Both source and destination are registers (rs1, rs2, rd)
  - Used for **ALU operations** (arithmetic, logic, shift, etc.)
  - funct3 and funct7 differentiate specific operations
- Common Instructions
  - add rd, rs1, rs2
  - sub rd, rs1, rs2
  - and rd, rs1, rs2
  - sll rd, rs1, rs2

RISC-V: Immediate-type Instruction

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Immediate (I-type)	imm[11:0]												rs1					funct3			rd					opcode						

- Immediate-type Instruction (I-type)
  - Purpose: used for **immediate operations, memory access (load)**
- Characteristics
  - rs1 is the source reg, rd is the destination reg, and imm is a 12-bit immediate value
  - Used for arithmetic, data access (load), and jumps
- Common Instructions
  - addi rd, rs1, imm
  - lw rd, imm(rs1)
  - jalr rd, imm(rs1)
  - slti rd, rs1, imm

RISC-V: Branch-type Instruction

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Branch (B-type)	[12]	imm[10:5]						rs2					rs1					func3			imm[4:1]				[11]	opcode						

- Branch-type Instruction (B-type)
  - Purpose: used for conditional branching
  - Support **loop control, if-else statements**
- Characteristics
  - rs1, rs2 are used for comparison
  - Immediate (imm) is a 12-bit signed **offset relative to PC**
  - Branch addresses are multiples of 2 (instruction alignment)
- Common Instructions
  - beq rs1, rs2, imm
  - bne rs1, rs2, imm
  - blt rs1, rs2, imm
  - bge rs1, rs2, imm

RISC-V: Jump-type Instruction

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Jump (J-type)	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode					

- Jump-type Instruction (J-type)
  - Purpose: used for unconditional jumps
  - Support **function calls & return**, PC modifications
- Characteristics
  - Immediate (imm) is a 20-bit signed **offset relative to PC**
  - rd stores the return address (used for function calls)
- Common Instructions
  - jal rd, imm

RISC-V: (Un)conditional Jump/Branch

- Notice that immediate values among instructions have different meanings

- Beq
  - Immediate is encoded in ***halfword*** offset
  - Branch to an ***relative*** address (PC + Immediate)
- Jal
  - Immediate is encoded in ***halfword*** offset
  - Jump to an ***relative*** address (PC + immediate)
  - Store PC+4 to an indicated register
- Jalr
  - Immediate is encoded in ***byte*** offset
  - Jump to an ***absolute*** address (rs + immediate)
  - Store PC+4 to an indicated register

Addr	Instr
0x00	Addi x0, x0, 0
0x04	Addi x0, x0, 0
0x08	Beq x0, x0, **0x08**
0x0C	…
0x10	…
0x14	…
0x18	Jal x2, **0x04**
0x1C	…
0x20	Jalr x3, x2, **0xFF0**

Register File		
	x0	Constant 0
	x2	**0x1C**
	x3	0x24

※ Due to 16 bit Compressed Instruction, half-word is used

RISC-V: Store-type Instruction

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Store (S-type)	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						

- Store-type Instruction (S-type)
  - Purpose: used for storing data into memory
- Characteristics
  - Requires **rs1 for the address** and **rs2 for the data**
  - The immediate value (imm[11:5] and imm[4:0]) forms a 12-bit offset to compute the memory address
- Common Instructions
  - sw rs2, imm(rs1)
  - sh rs2, imm(rs1)
  - sb rs2, imm(rs1)

RISC-V:
Upper immediate-type Instruction

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Upper immediate (U-type)	imm[31:12]																				rd					opcode					

- Upper immediate-type Instruction (U-type)
  - Purpose: used for loading large constants or global address calculations
  - **Enables large 32-bit constant loading**
- Characteristics
  - 20-bit immediate value (shifted left by 12 bits)
  - rd is the target register
- Common Instructions
  - lui rd, imm
  - auipc rd, imm

RISC-V vs. MIPS: Architecture & Design Comparison

- Both RISC-V and MIPS are RISC (Reduced Instruction Set Computer)
- They differ in instruction format, hardware design, extensibility, and software support

Feature	RISC-V	MIPS
Open-source	Yes (no license fees)	No (proprietary)
Instruction format	32-bit, 16-bit (compressed mode)	Fixed 32-bit
Immediate size	12-bit (I-type), 20-bit (U/J-type)	16-bit
Branch instructions	BEQ, BNE, **BLT, BGE**	BEQ, BNE, **SLT + BEQ/BNE**
Jump instructions	JAL, JALR	J, JAL, JR, JALR
Floating point	Optional (F/D extension)	Support MIPS II ↑

RISC-V vs. MIPS: Instruction Format & Extensibility

- Both RISC-V and MIPS are **regularly designed ISA**
- opcode, funct: define operation
- rs1, rs2: **source** registers (read), rd: **destination** registers (write)
- imm : **immediate** numbers (different meanings among instructions)

Instruction Formats		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RISCV	Register (R-type)	funct7							rs2					rs1					funct3			rd					opcode					
	Immediate (I-type)	imm[11:0]												rs1					funct3			rd					opcode					
	Jump (J-type)	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode					
	Branch (B-type)	[12]	imm[10:5]						rs2					rs1					func3			imm[4:1]				[11]	opcode					
	Store (S-type)	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode					
	Upper immediate (U-type)	imm[31:12]																				rd					opcode					
MIPS	Register (R-type)	opcode							rs					rt					rd			sa					function					
	Immediate (I-type)	opcode							rs					rt					immediate													
	Jump (J-type)	opcode							offset																							

RISC-V vs. MIPS: Instruction Format & Extensibility

- RISC-V: **6 basic formats** (R, I, J, B, S, U) → support extension
- MIPS: **3 basic formats** (R, I, J) → less flexible

Instruction Formats		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RISCV	Register (R-type)	funct7							rs2					rs1					funct3			rd					opcode					
	Immediate (I-type)	imm[11:0]												rs1					funct3			rd					opcode					
	Jump (J-type)	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode					
	Branch (B-type)	[12]	imm[10:5]						rs2					rs1					func3			imm[4:1]				[11]	opcode					
	Store (S-type)	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode					
	Upper immediate (U-type)	imm[31:12]																				rd					opcode					
MIPS	Register (R-type)	opcode						rs						rt					rd			sa					function					
	Immediate (I-type)	opcode						rs						rt					immediate													
	Jump (J-type)	opcode						offset																									

RISC-V vs. MIPS: Instruction Format & Extensibility

- RISC-V: **6 basic formats** (R, I, J, B, S, U) → support extension
- MIPS: **3 basic formats** (R, I, J) → less flexible
- **RISC-V supports compressed 16-bit instructions** (C extension) to improve code density

Feature	RISC-V	MIPS
Modularity	**Highly extensible** (I, M, A, F, D, C, V)	Fixed ISA (MIPS32, MIPS64)
Instruction Type	R, I, S, B, U, J	R, I, J
Compressed instructions	C extension (16-bit instructions)	MIPS16e (less commonly used)
Vector processing	V extension (dynamic vector length)	Fixed-length SIMD

RISC-V vs. MIPS: Register Convention

Register	RISC-V	MIPS	Description
Constant Zero	zero (x0)	$zero ($0)	Always **0**
Return Address	ra (x1)	$ra ($31)	Stores return address
Stack Pointer	sp (x2)	$sp ($29)	Points to stack
Global Pointer	gp (x3)	$gp ($28)	Points to global variables
Thread Pointer	tp (x4)	N/A	Points to thread-local storage (TLS)
Temp Register	t0-t6 (x5-x7, x28-x31)	$t0-$t9 ($8-$15, $24-$25)	Used for temporary values
Saved Register	s0-s11 (x8-x9, x18-x27)	$s0-$s7 ($16-$23)	Preserved across function calls
Function Arguments	a0-a7 (x10-x17)	$a0-$a3 ($4-$7)	Pass function arguments
Return Value	a0-a1 (x10-x11)	$v0-$v1 ($2-$3)	Return function values
Frame Pointer	fp (x8)	$fp ($30)	Used as frame pointer
Kernel Reserved	N/A	$k0-$k1 ($26--$27)	Reserved for OS kernel
Program Counter	PC	PC	Stores the next instruction address

RISC-V vs. MIPS: Instruction Comparison

Operation	Pseudo Instruction	RISC-V	MIPS
Arithmetic	Addition	add rd, rs1, rs2	add $rd, $rs, $rt
	Subtraction	sub rd, rs1, rs2	sub $rd, $rs, $rt
	Multiplication	mul rd, rs1, rs2	mul $rd, $rs, $rt
	Division	div rd, rs1, rs2	div $rd, $rs, $rt
Logical and Bitwise	Bitwise AND	and $rd, $rs, $rt	and $rd, $rs, $rt
	Bitwise OR	or $rd, $rs, $rt	sub $rd, $rs, $rt
	Bitwise XOR	xor $rd, $rs, $rt	xor $rd, $rs, $rt
	Shift Left Logical	sll $rd, $rs, $rt	sll $rd, $rs, $rt
	Shift Right Logical	srl $rd, $rs, $rt	srl $rd, $rs, $rt
Memory Access	Load Word	lw rd, offset (rs1)	lw $rt, offset ($rs)
	Load Byte	lb rd, offset (rs1)	lb $rt, offset ($rs)
	Store Word	sw rs2, offset (rs1)	sw $rt, offset ($rs)
	Store Byte	sb rs2, offset (rs1)	sb $rt, offset ($rs)
Branching & Jump	Branch if Equal	beq rs1, rs2, imm	beq $rs, $rt, offset
	Branch if Not Equal	bne rs1, rs2, imm	bne $rs, $rt, offset
	Jump and Link	jal rd, imm	jal $ra, offset
	Jump Register	jalr rd, offset (rs1)	js $rs

RISC-V vs. MIPS: Instruction Comparison

- Consider only basic integer computation, (un)conditional jump, and load/store
  - RISC-V: **37 instructions (winner)**
  - MIPS R2000: **51 instructions**
- Pseudo instruction (RISC-V ISA)

Pseudo Instruction	RISC-V	MIPS
nop	addi x0, x0, 0	sll, $0, $0, 0
not rd, rs	xor rd, rs, -1	nor rd, rs, $0
neg rd, rs	sub rd, x0, rs	sub rd, $0, rs
j offset	jal x0, offset	not pseudo instruction
jal offset	jal x1, offset	not pseudo instruction
jr rs	jalr x0, rs, 0	not pseudo instruction
jalr rs	jalr x1, rs, 0	jalr $31, rs

RISC-V vs. MIPS: Software Ecosystem & Application

- RISC-V is growing in all major areas, from AI to cloud and embedded computing
- MIPS is losing software and industry support, with fewer new product

Category	RISC-V	MIPS
Industry Adoption	IoT, AI, cloud, embedded	embedded
Compiler	GCC, LLVM/Clang (full support)	GCC, LLVM (limited support)
Operating Systems	Linux (Ubuntu, Fedora), RTOS	Linux (OpenWRT, FreeBSD)
Cloud Computing	QEMU, KVM, Firecracker (AWS), Docker (WIP)	N/A
AI/ML	TensorFlow Lite, TVM, RISC-V AI Chips	N/A

RISC-V vs. MIPS: MIPS Products (Application)

- MIPS is used in early embedded application







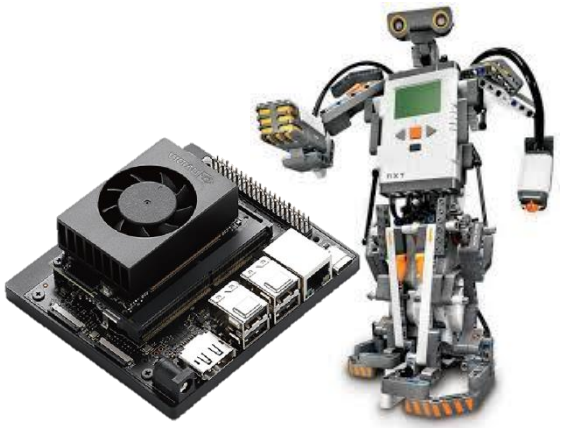


RISC-V vs. MIPS: RISC-V Product (Application)

- RISC-V is used in modern application
  - Embedded system, IoT device, cloud computing, AI, etc.





RISC-V vs. MIPS: Endianness

RISC-V (Little-endian)

- **add $r1, $r2, $r3** # **$r1 = $r2 + $r3**

↓31bit ↓0bit

- **0000000 00011 00010 000 00001 0110011**
  funct7 $rs2=$r3 $rs1=$r2 funct3 $rd=$r1 OP
- Instructions are stored in memory in a little-endian sequence of **bytes**
- **00 31 00 B3** (hex) —Little Endian→ **B3 00 31 00**(hex)

[31:16] [15:0] **[15:0] [31:16]**

128x32 SRAM	[31:24]	[23:16]	[15:8]	[7:0]
0	B3	00	31	00
:				
127				

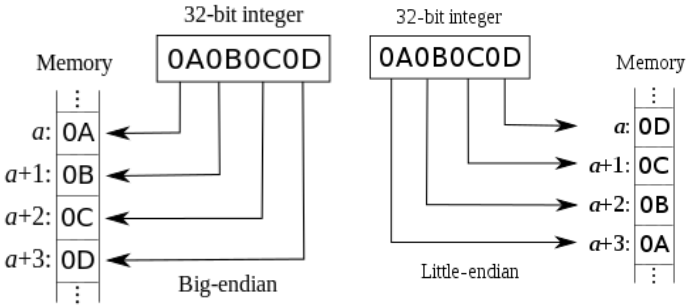
MIPS (Big-endian)

- **add $r1, $r2, r3** # **$r1 = $r2 + $r3**

↓31bit ↓0bit

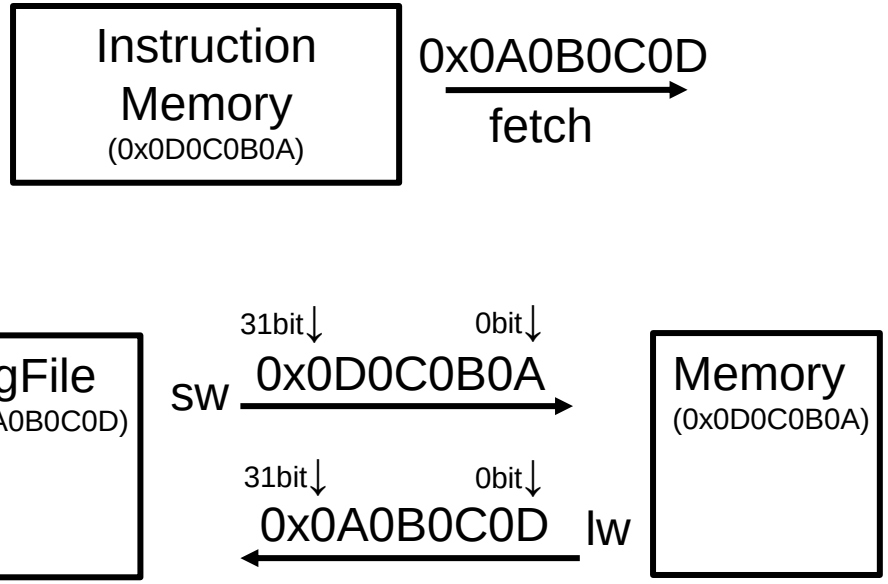
- **000000 00010 00011 00001 00000 000000**
  OP $rs = $r2 $rt = $r3 $rd = $r1 shamt=0 funct = 0
- **00 43 08 00** (**Instruction in Big Endian & Hex**)
  [31:16] [15:0]

128x32 SRAM	[31:24]	[23:16]	[15:8]	[7:0]
0	00	43	08	00
:				
127				



RISC-V vs. MIPS: Data Format

- Be cautious with endian when load/store data
  - Need to perform conversion when encountering the interface of SRAM
  - Assume data 0x0A0B0C0D



RISC-V vs. MIPS: Data Format

- Why Little endian?
  - Fetch with the same address if a given value is stored in different width
    - 32bit 0x0D0C0B0A
    - 64bit 0x000000000D0C0B0A
    - We can always fetch the lowest 32bit address
  - Mainstream
    - Intel x86