MIPS Instruction Set

DSC 315: Computer Organization & Operating Systems

Dr. Laltu Sardar

School of Data Science,
Indian Institute of Science Education and Research Thiruvananthapuram (IISER TVM)

23rd January, 2026

Instructions

Instruction Set and Computer Language

Key Idea

To command a computer's hardware, you must speak its language.

- The **words** of a computer's language are called **instructions**.
- The **complete vocabulary** of these instructions is known as the **instruction set**.
- Hardware understands and executes only the instructions defined in its instruction set.

Example of Instruction Set and Instructions

Instruction Set

An instruction set is the complete collection of machine level commands that a processor can understand and execute.

- Example instruction set: **x86**, **ARMv7**, **ARMv8**, **RISC-V**, **MIPS**
- Each instruction set defines:
  - **Allowed operations:** Define what actions the processor can perform, such as arithmetic, logic, data movement, and control flow operations.
  - **Instruction formats:** Specify how an instruction is encoded, including the size and position of fields like opcode, registers, and immediate values.
  - **Addressing modes:** Describe how an instruction identifies the location of its operands in registers, memory, or constants.

We study MIPS Instruction Set

Program Representation Levels

High-Level Language
(C, C++, Python)

⇓ Compiler

Assembly Language
(Human-readable instructions)

⇓ Assembler

Machine Language
(Binary instructions executed by hardware)

- Programs move from abstract descriptions to hardware-executable form.
- Each level hides complexity while preserving program meaning.

Learning Flow: Assembly to Machine Instructions

Assumed Assembly Instructions

(Formats and Syntax Given)

⇓

Machine Instruction Format

(Encoding and Syntax)

⇓

Assembly to Machine Translation

(Instruction Encoding)

Example: From C to Assembly to Machine Instruction

C Code

```
c = a + b;
```

Assembly Code

```
ADD x3, x1, x2
```

Machine Instruction (Binary Encoding)

```
0000000 00010 00001 000 00011 0110011
```

- High level code expresses intent.
- Assembly shows explicit register operations.

Example: d = a + b + c

C Code

```
d = a + b + c;
```

Assembly Code

```
ADD x4, x1, x2
ADD x5, x4, x3  // xi's are addresses of the registers
```

Machine Instructions (Conceptual Encoding)

```
0000000 00010 00001 000 00100 0110011
0000000 00011 00100 000 00101 0110011
```

- First instruction computes a + b.
- Second instruction adds c to the intermediate result.
- Final value is stored as d.

Program Execution and Instructions

Key Idea

When you execute a program, you are actually executing a **sequence of instructions**. Each instruction is fetched and executed one at a time.

→ Program Counter (PC)	Instruction Array
1001 →	Instruction 1
1002 →	Instruction 2
1003 →	Instruction 3
1004 →	Instruction 4

- The Program Counter holds the address of the current instruction.
- After execution, the PC moves to the next instruction.

MIPS Instruction Operands

- MIPS instructions ‘ typically use **three operands**.
- Operands specify where data comes from and where results are stored.

Types of Operands in MIPS

- **Registers:** Most operations use registers, for example `$t0, $t1, $t2`.
- **Immediate values:** Constant values encoded directly in the instruction.
- **Memory operands:** Accessed using load and store instructions with base plus offset.

Example

```
1               add $t0, $t1, $t2
2               lw  $t0, 4($t1)
```

MIPS Registers and Their Names

Register	Name	Purpose
$zero	$0	Constant value 0
$at	$1	Assembler temporary
$v0–$v1	$2–$3	Function return values
$a0–$a3	$4–$7	Function arguments
$t0–$t7	$8–$15	Temporary registers
$s0–$s7	$16–$23	Saved registers
$t8–$t9	$24–$25	More temporaries
$k0–$k1	$26–$27	Operating system use
$gp	$28	Global pointer
$sp	$29	Stack pointer
$fp	$30	Frame pointer
$ra	$31	Return address

- MIPS has registers.

General Purpose vs Special Purpose Registers

General Purpose Registers

- Used for arithmetic and logic operations
- Hold operands and intermediate results
- Freely used by programmers and compilers
- Examples in MIPS:
  - $t0–$t9 (temporaries)
  - $s0–$s7 (saved registers)

Special Purpose Registers

- Have predefined roles in program execution
- Support control flow, memory, and system functions
- Usage follows strict conventions
- Examples in MIPS:
  - $sp (stack pointer)
  - $ra (return address)
  - $gp (global pointer)
  - $zero (constant zero)

- General purpose registers focus on computation.
- Special purpose registers support program structure and control.

Special Purpose Registers in MIPS

- **$zero** Always holds the constant value 0. Useful for comparisons and moves.
- **$at** Reserved for the assembler. Handles large constants.
- **$gp** Global pointer. Provides fast access to global and static data.
- **$sp** Stack pointer. Points to the top of the runtime stack.
- **$fp** Frame pointer. Helps access local variables and function parameters.
- **$ra** Return address. Stores the address to return after a function call.
- **$k0, $k1** Reserved for operating system and exception handling.

- These registers have predefined roles by convention.
- Programmers usually avoid modifying OS reserved registers.

IISER THIRUVANANTHAPURAM

Immediate Values in MIPS

- Immediate values are **constant operands embedded in the instruction.**
- They remove the need to load constants from memory.
- Immediate fields have limited size due to instruction width.

Example

```
1               addi $t0, $t1, 10
```

- Adds constant 10 directly to the contents of `$t1`.
- Result is stored in `$t0`.
- 16-bits kept for constants as signed int.

Memory Operands in MIPS

- MIPS uses **load** and **store** instructions to access memory.
- **Arithmetic instructions operate only on registers.**
- Memory addresses are computed using base plus offset.

Example

```
1               lw  $t0, 8($t1)
2               sw  $t0, 12($t1)
```

- `lw` loads data from memory into a register.
- `sw` stores data from a register into memory.
- MIPS uses a **32-bit address space**.
- Memory is **byte addressed**.

MIPS Assembly Instructions

MIPS Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
sub	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three register operands
addi	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants

MIPS Data Transfer Instructions

Instruction	Example	Meaning	Comments
lw	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word memory to register
sw	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word register to memory
lh	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword to register
lhu	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword unsigned
sh	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword to memory
lb	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte to register
lbu	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte unsigned
sb	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte to memory
ll	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Atomic load
sc	sc $s1,20($s2)	Memory[...] = $s1; $s1=0 or 1	Atomic store
lui	lui $s1,20	$s1 = 20×2^{16}	Load upper immediate

MIPS Logical Instructions

Instruction	Example	Meaning	Comments
and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Bitwise AND
or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Bitwise OR
nor	nor $s1,$s2,$s3	$s1 = $\sim$($s2 | $s3)	Bitwise NOR
andi	andi $s1,$s2,20	$s1 = $s2 & 20	AND with constant
ori	ori $s1,$s2,20	$s1 = $s2 | 20	OR with constant
sll	sll $s1,$s2,10	$s1 = $s2 $\ll$ 10	Shift left logical
srl	srl $s1,$s2,10	$s1 = $s2 $\gg$ 10	Shift right logical

MIPS Conditional Branch Instructions

Instruction	Example	Meaning	Comments
beq	beq $s1,$s2,25	if equal branch	PC relative branch
bne	bne $s1,$s2,25	if not equal branch	PC relative branch
slt	slt $s1,$s2,$s3	$s1=1 if $s2<$s3	Signed compare
sltu	sltu $s1,$s2,$s3	$s1=1 if $s2<$s3	Unsigned compare
slti	slti $s1,$s2,20	$s1=1 if $s2<20	Immediate compare
sltiu	sltiu $s1,$s2,20	$s1=1 if $s2<20	Unsigned immediate

MIPS Unconditional Jump Instructions

Instruction	Example	Meaning	Comments
j	j 2500	go to target	Jump instruction
jr	jr $ra	go to $ra	Procedure return
jal	jal 2500	$ra = PC+4; jump	Procedure call

MIPS Unconditional Jump Instructions

Important Note to Students

- You do **not** need to memorize individual instructions.
- Focus on understanding instruction categories and usage.
- Learn how operands and addressing work.
- In examinations, the required set of instructions will be provided.

Example: C to Assembly Translation

C Code

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

Assumed Register Mapping

- $s1 = g $s2 = h $s3 = i $s4 = j $s0 = f

Assembly Code

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

- First addition computes $g + h$.
- Second addition computes $i + j$.
- Final subtraction produces the result f.

What will happen when the addresses are in the memory?

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF(255_{10})
90000001	00			
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1_{10})
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFFFFF ($4.45015E-308_{10}$)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Memory Address Rules

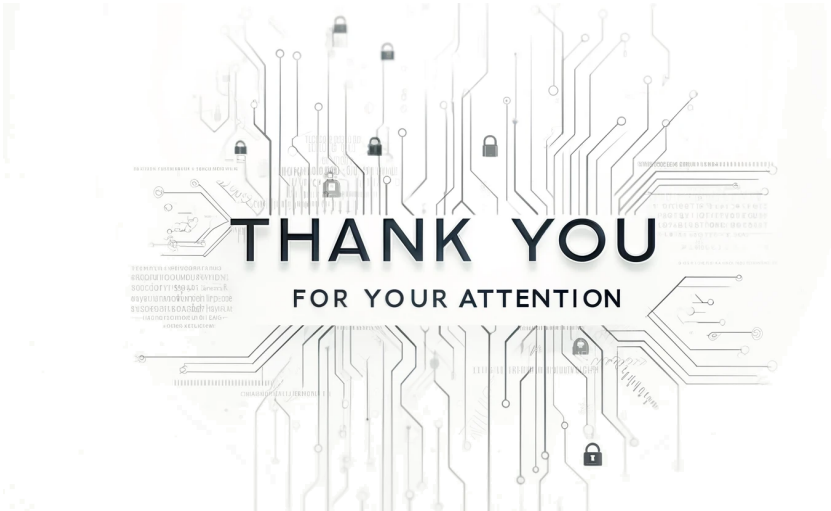
- Memory is **byte addressed**.
- Each memory address refers to exactly **one byte**.
- A word consists of **4 consecutive bytes**.

Alignment Rules

- Word addresses must be **aligned**.
- Word access addresses are multiples of 4.
- Misaligned accesses are either slow or not allowed.

Address Calculation

- Effective address = base register + offset.
- **Offset is specified in bytes**.



Dr. Laltu Sardar, Assistant Professor, IISER Thiruvananthapuram
laltu.sardar@iisertvm.ac.in, laltu.sardar.crypto@gmail.com
Course webpage: https://laltu-sardar.github.io/courses/corgos_2026.html.


Carl Hamacher, Zvonko Vranesic, and Safwat Zaky.
Computer Organization.
McGraw-Hill, 6th edition, 2012.


John L. Hennessy and David A. Patterson.
Computer Architecture: A Quantitative Approach.
Morgan Kaufmann, 6th edition, 2017.


Vincent P. Heuring and Harry F. Jordan.
Computer System Design and Architecture.
Pearson, 2nd edition, 2007.


David A. Patterson and John L. Hennessy.
Computer Organization and Design: The Hardware/Software Interface.
Morgan Kaufmann, 4th edition, 2014.


William Stallings.
Computer Organization and Architecture: Designing for Performance.
Pearson, 10th edition, 2016.