

# Computer Architecture

## Chapter 2: MIPS

Dr. Phạm Quốc Cường

Adapted from Computer Organization the Hardware/Software Interface – 5th



Computer Engineering – CSE – HCMUT

# Introduction

- **Language:** a system of communication consisting of sounds, words, and grammar, or the system of communication used by people in a particular country or type of work (Oxford Dictionary)



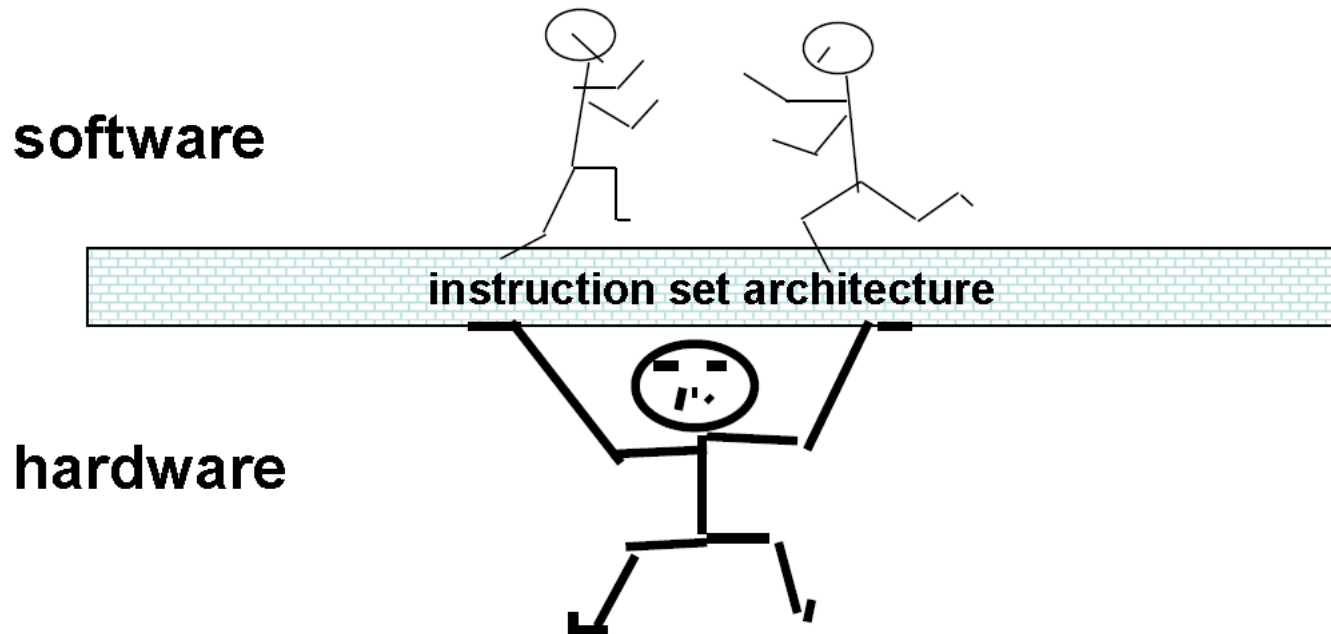


# Introduction (cont.)

- To command a computer's hardware: speak its language



# Instruction Set Architecture (ISA)



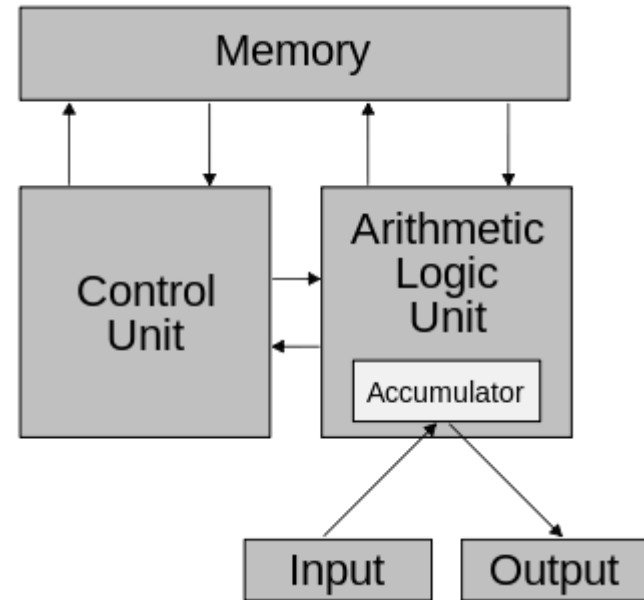
MIPS32 Add Immediate Instruction

001000	00001	00010	0000000101011110
OP Code	Addr 1	Addr 2	Immediate value

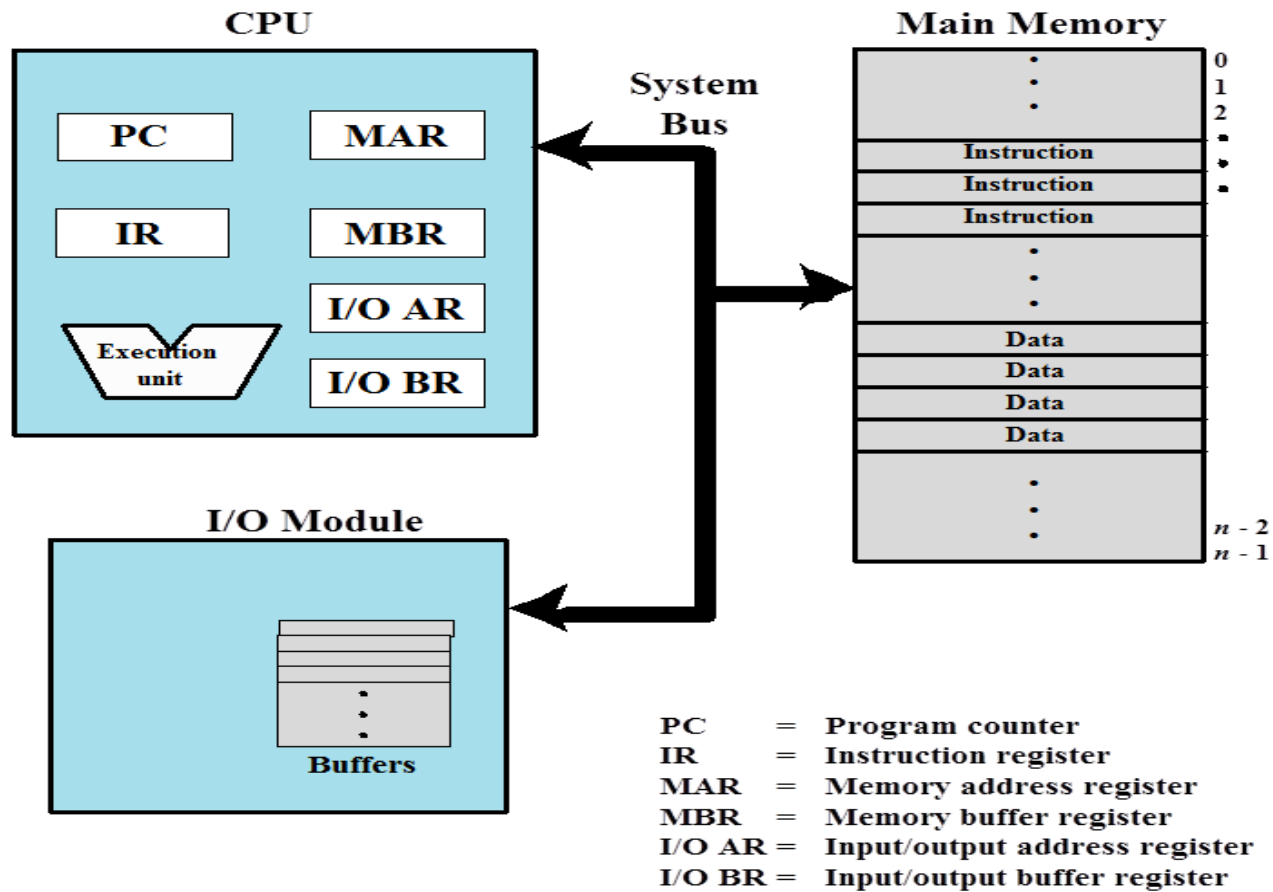
Equivalent mnemonic: **addi** \$r1, \$r2, 350

# Von Neumann Architecture

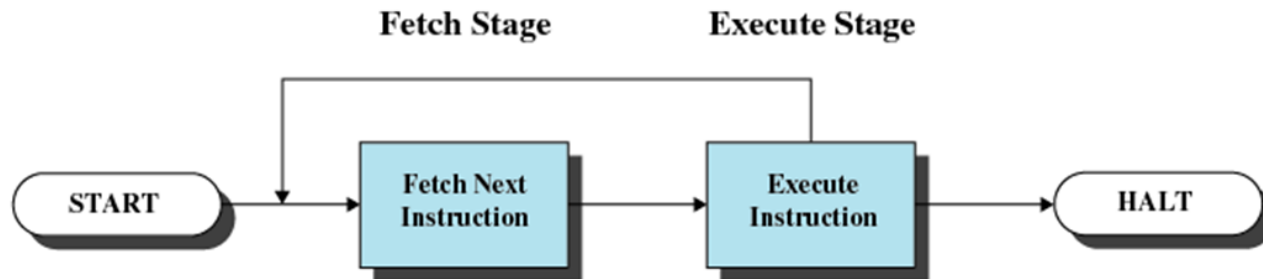
- Stored-program concept
- Instruction category:
  - Arithmetic
  - Data transfer
  - Logical
  - Conditional branch
  - Unconditional jump



# Computer Components



# Instruction Execution



- Instruction fetch: from the memory
  - PC increased
  - PC stores the next instruction
- Execution: decode and execute

# The MIPS Instruction Set

- MIPS architecture
- MIPS Assembly Inst.  $\Leftrightarrow$  MIPS Machine Instr.
- Assembly:
  - `add $t0, $s2, $t0`
- Machine:
  - `000000_10010_01000_01000_00000_100000`
- Only **one operation** is performed per MIPS instruction



# IS Design Principles

- Simplicity favors regularity
- Smaller is faster
- Make the common case fast
- Good design demands good compromises

# MIPS Operands

- 32 32-bit registers
  - $\$s0-\$s7$ : corresponding to variables
  - $\$t0-\$t9$ : storing temporary value
  - $\$a0-\$a3$
  - $\$v0-\$v1$
  - $\$gp, \$fp, \$sp, \$ra, \$at, \$zero, \$k0-\$k1$
- $2^{30}$  memory words (4 byte): accessed only by data transfer instructions (memory operand)
- Immediate

# Arithmetic Instructions



- Opcode:
  - add:  $DR = SR1 + SR2$
  - sub:  $DR = SR1 - SR2$
  - addi: SR2 is an immediate (e.g. 20),  $DR = SR1 + SR2$
- Three register operands

# Design Principle 1

- **Simplicity favours regularity**
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

# Arithmetic Instructions: Example

- **Q**: what is MIPS code for the following C code

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

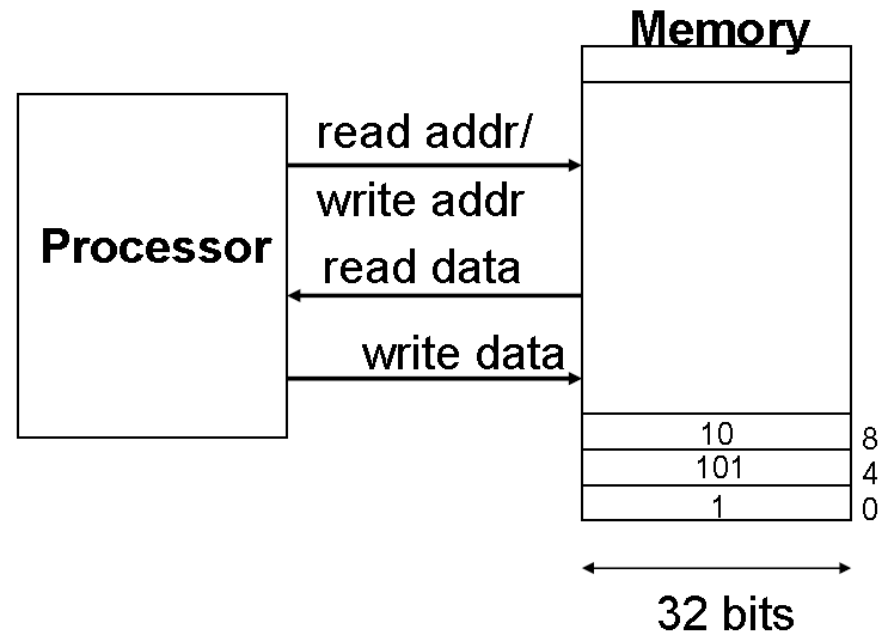
If the variables  $g$ ,  $h$ ,  $i$ ,  $j$ , and  $f$  are assigned to the register  $\$s0$ ,  $\$s1$ ,  $\$s2$ ,  $\$s3$ , and  $\$s4$ , respectively.

- **A**:

```
add $t0, $s0, $s1 # g + h
add $t1, $s2, $s3 # i + j
sub $s4, $t0, $t1 # t0 - t1
```

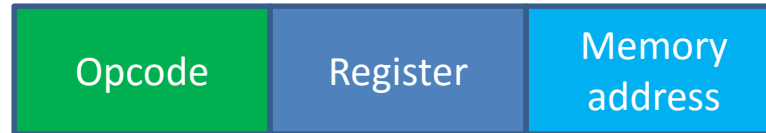
# Data Transfer Instructions

- Move data b/w memory and registers
  - Register
  - Address: a value used to delineate the location of a specific data element within a memory array
- Load: **copy** data from memory to a register
- Store: **copy** data from a register to memory





# Data Transfer Instructions (cont.)



- Memory address: *offset(base register)*
  - Byte address: each address identifies an 8-bit byte
  - “words” are aligned in memory (address must be multiple of 4)

# Data Transfer Instructions (cont.)

- Opcode:
  - lw: load word
  - sw: store word
  - lh: load half ( $\$s1 = \{16\{M[\$s2+imm][15]\}, M[\$s2 + imm]\}$ )
  - lhu: load half unsigned ( $\$s1 = \{16'b0, M[\$s2 + imm]\}$ )
  - sh: store half
  - lb: load byte
  - lbu: load byte unsigned
  - sb: store byte
  - ll: load linked word
  - sc: store conditional
  - lui: load upper immediate  $\$s1 = \{imm, 16'b0\}$

# Memory Operands

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory
- MIPS is Big Endian
  - Most-significant byte at least address of a word
  - *c.f.* Little Endian: least-significant byte at least address

# Memory Operand Example 1

- C code:

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

– g in \$s1, h in \$s2, base address of A in \$s3

- Compiled MIPS code:

– Index 8 requires offset of 32

- 4 bytes per word

```
lw    $t0, 32($s3)    # load word
add   $s1, $s2, $t0
```

# Memory Operand Example 2

- C code:

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

– `h` in `$s2`, base address of `A` in `$s3`

- Compiled MIPS code:

– Index 8 requires offset of 32

```
lw    $t0, 32($s3)    # load word
```

```
add   $t0, $s2, $t0
```

```
sw    $t0, 48($s3)    # store word
```

# 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!



# Immediate Operands

- Constant data specified in an instruction  
`addi $s3, $s3, 4`
- No subtract immediate instruction
  - Just use a negative constant  
`addi $s2, $s1, -1`
- *Design Principle 3: Make the common case fast*
  - Small constants are common
  - Immediate operand avoids a load instruction

# The Constant Zero

- MIPS register 0 (\$zero) is the constant 0
  - Cannot be overwritten
- Useful for common operations
  - E.g., move between registers  
add \$t2, \$s1, \$zero

# Unsigned Binary Integers

- Given an n-bit number

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

- Range: 0 to  $+2^n - 1$
- Example
  - 0000 0000 0000 0000 0000 0000 0000 1011<sub>2</sub>  
=  $0 + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$   
=  $0 + \dots + 8 + 0 + 2 + 1 = 11_{10}$
- Using 32 bits
  - 0 to +4,294,967,295

# 2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

- Range:  $-2^{n-1}$  to  $+2^{n-1} - 1$

- Example

$$\begin{aligned} & - 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2 \\ & = -1 \times 2^{31} + 1 \times 2^{30} + \dots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\ & = -2,147,483,648 + 2,147,483,644 = -4_{10} \end{aligned}$$

- Using 32 bits

$$- -2,147,483,648 \text{ to } +2,147,483,647$$

# 2s-Complement Signed Integers

- Bit 31 is sign bit
  - 1 for negative numbers
  - 0 for non-negative numbers
- $-(-2^{n-1})$  can't be represented
- Non-negative numbers have the same unsigned and 2s-complement representation
- Some specific numbers
  - 0: 0000 0000 ... 0000
  - -1: 1111 1111 ... 1111
  - Most-negative: 1000 0000 ... 0000
  - Most-positive: 0111 1111 ... 1111

# Signed Negation

- Complement and add 1
  - Complement means  $1 \rightarrow 0, 0 \rightarrow 1$

$$x + \bar{x} = 1111 \dots 111_2 = -1$$

$$\bar{x} + 1 = -x$$

- Example: negate +2
  - $+2 = 0000 \ 0000 \dots 0010_2$
  - $-2 = 1111 \ 1111 \dots 1101_2 + 1$   
 $= 1111 \ 1111 \dots 1110_2$



# Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- In MIPS instruction set
  - `addi`: extend immediate value
  - `lb`, `lh`: extend loaded byte/halfword
  - `beq`, `bne`: extend the displacement
- 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