CHAPTER 2-1

Computer Abstractions and Technology

By Pattama Longani
Collage of arts, media and Technology

Human Language

Diversity

Computer Language

Similar principle of H/W Techology

A few basic operation

Easy to built H/W and compiler

Maximize performance

Minimize cost & power

Similarity

instructions

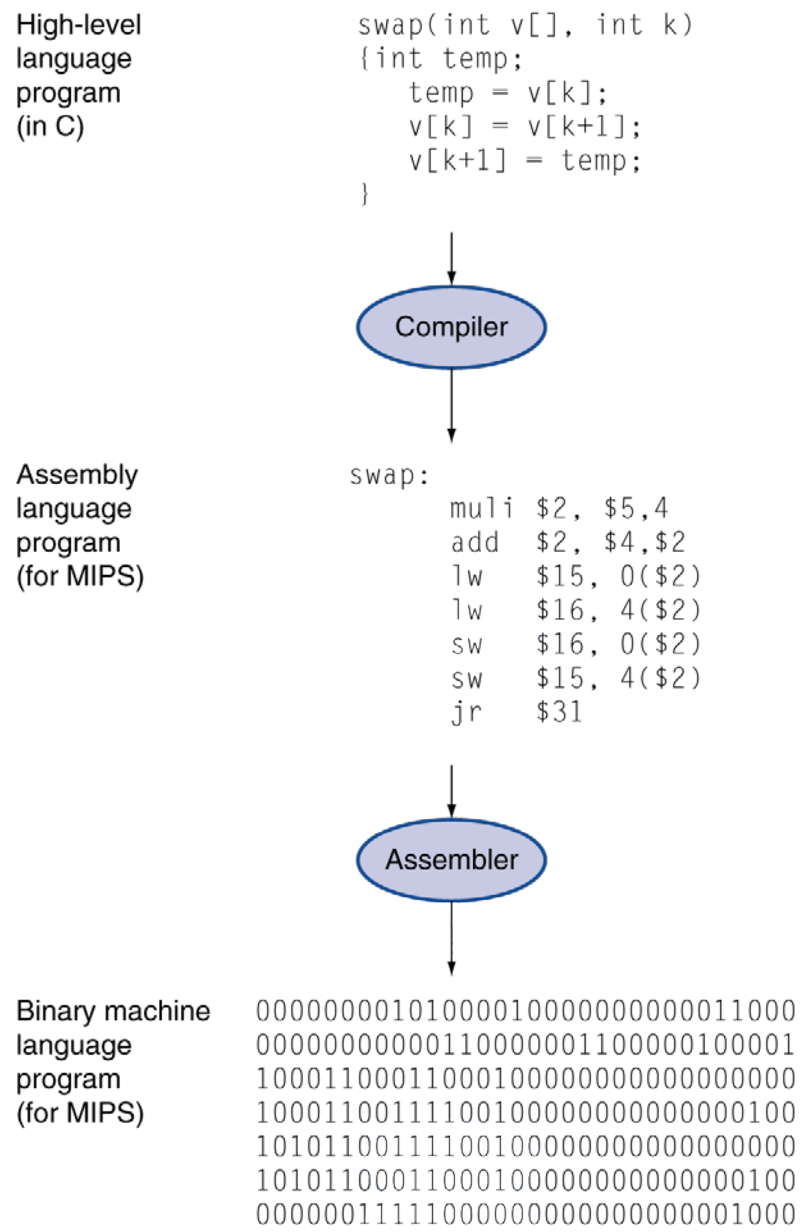
- The words of a computer's language
- add, sub, lw, …

instruction set

- computer's language's vocabulary
- {add, sub, lw, …}

We will learn **MISP instruction set.**

- showing both how it is represented in hardware
- the relationship between high-level and low-level programming languages



Arithmetic Operation

MIPS arithmetic instruction performs only **one operation** at a time.

Ex: adding the two variables b and c and to put their sum in a.

(a = b+c; in C)

MISP : **add** a, b, c

<u>Ex:</u> Subtracting the variables b from c and put the result in a.

(a = b-c)

MISP : $\boxed{\textbf{sub}\ a,\ b,\ c}$

- Instruction “add”, “sub” always have exactly three variables.

Ex: Sum four variables b, c, d, and e into variable a.

Solution:

```
add a, b, c
add a, a, d
add a, a, e
#a=b+c+d+e
```

Sharp symbol # is called **comments**
Computer ignore them

PRINCIPLES OF HARDWARE DESIGN

Design Principle 1:

Simplicity favors regularity.

- a variable number of operands is more complicated than hardware for a fixed number.

Ex: Compiling Two C Assignment Statements into MIPS

a = b + c;

d = a – e;

Solution:

add a, b, c
sub d, a, e

The translation from C to MIPS assembly language instructions is performed by

the compiler

Ex: Compiling C Assignment Statements into MIPS

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

Solution:

Hint: **compiler** creates a **temporary variable**, called t_0, t_1 :

add t_0, g, h

add t_1, i, j

sub f, t_0, t_1

VARIABLE

High – level language	Low – level language
unlimited	limited number of special locations built directly in hardware called *registers*.

The size of a register in the MIPS architecture is 32 bits;
And there are 32 registers

groups of 32 bits occur so frequently that they are given the name **word**

PRINCIPLES OF HARDWARE DESIGN

Design Principle 2:

Smaller is faster

- A very large number of registers **may increase the clock cycle time** simply because it **takes electronic signals longer** when they must travel farther.
- the designer must balance for the number of registers (increase) with the clock cycle time (decrese).

- the three operands of MIPS arithmetic instructions must each be chosen from one of the 32-bit registers.
- The reason for not using more than 32 is the number of bits it would take in the **instruction format**
- Effective use of registers is critical to program performance.

- There are 32 registers for MISP (0 to 31)
  - $s0, $s1, . . ., $s7 for registers that correspond to variables in C and Java programs
  - $t0, $t1, . . . , $t9 for **temporary registers** needed to compile the program into MIPS instructions.

Ex: Compiling C Assignment Statements using register. **f = (g + h) – (i + j);**

Solution:

- Hint:The variables f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $s4, respectively.
- **compiler** creates a **temporary variable** on temporary register $t0, $t1

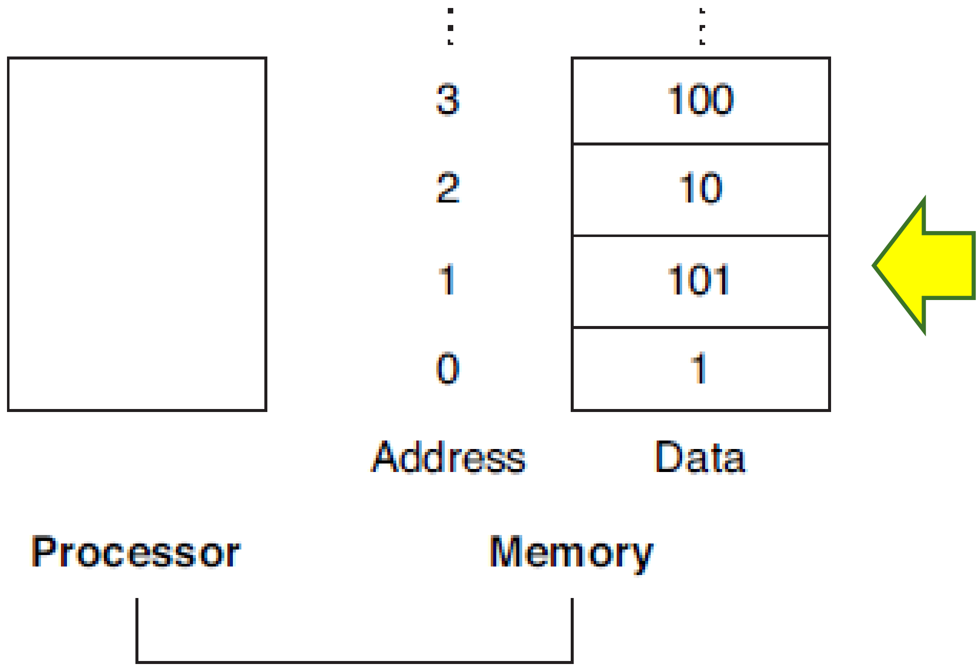
add t_0, g, h
add t_1, i, j
sub f, t_0, t_1

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

MEMORY OPERANDS

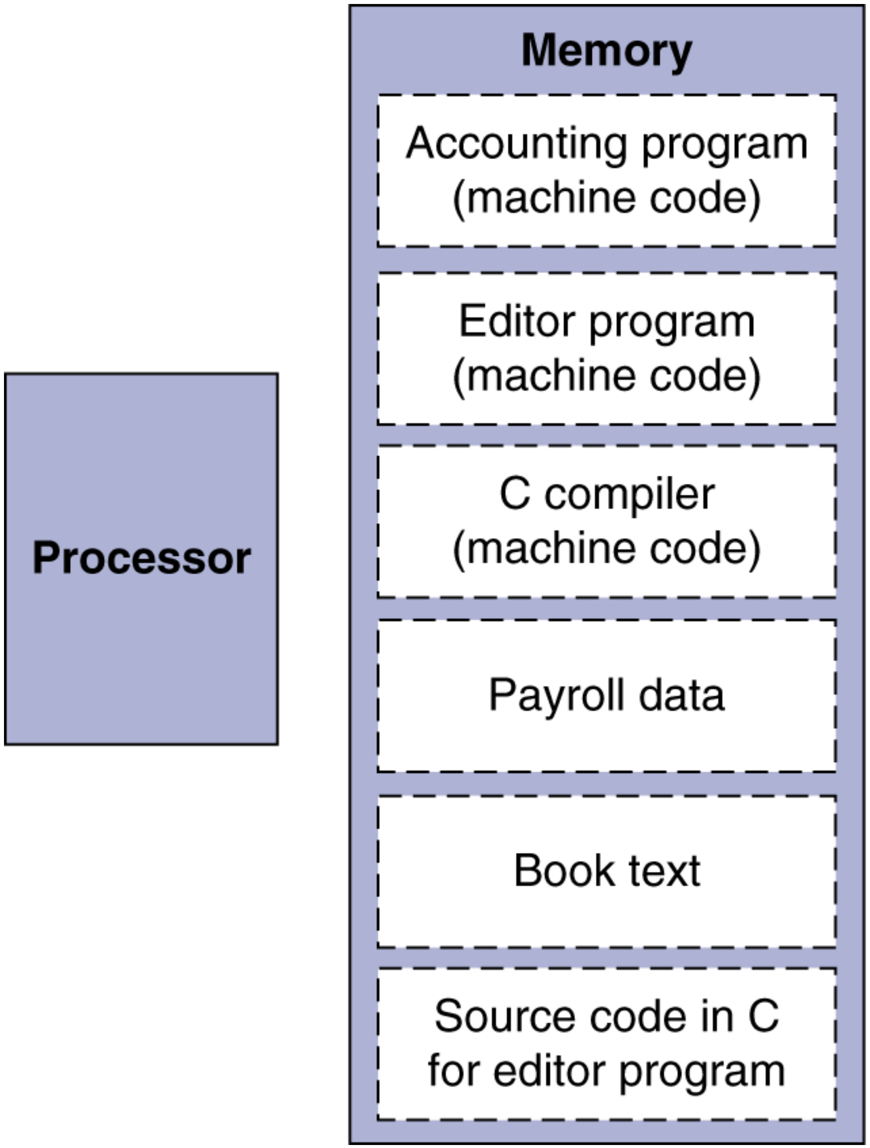
simple variables	arrays and structures
contain single data elements	that can contain more than one data elements
kept in register	kept in memory

- **arithmetic operations** occur **<u>only on registers</u>** in MIPS instructions; (add, sub)
- **data transfer instructions** transfer data between memory and registers. (lw, sw)



- the address of the third data element is 2
- the value of Memory[2] is 10.
- Base address is 0

- To access a word in memory, the instruction must supply the memory **address**
- **base address** is the starting address



1. Instructions are represented as numbers.
2. Programs are stored in memory to be read or written, just like numbers.

memory can contain

- the source code for an editor program,
- the corresponding compiled machine code, the text that the compiled program is using,
- even the compiler that generated the machine code.

BYTE ADDRESSING

- Processors can number bytes within a word so the byte with the lowest number is either the leftmost or rightmost one.
- The convention used by a machine is called its **byte order**.

big-endian

Byte #			
0	1	2	3

little-endian

Byte #			
3	2	1	0

Load Word = The data transfer instruction that copies data from memory to a register

<u>Ex:</u>

```
lw $s1,20($s2)
```

offset

Base register

<u>Mean:</u>

$s1 = Memory[$s2 + 20]

(a word from the memory is loaded to the register)

Compiling an Assignment When an Operand Is in Memory

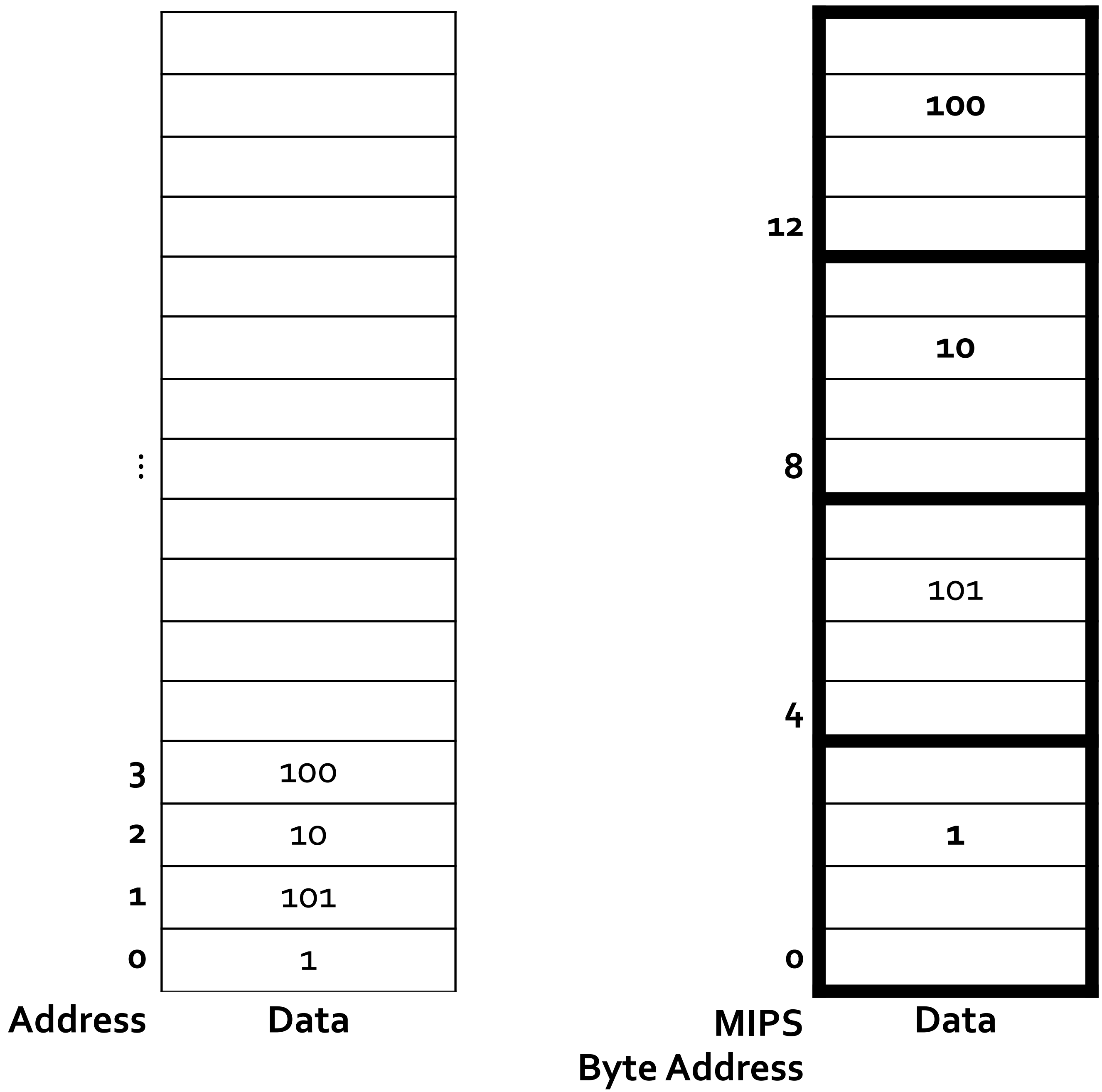
Ex: assume that A is an array of 100 words and that the compiler has associated the variables g and h with the registers $s1 and $s2. Let’s the base address of the array is in $s3.Compile this C assignment statement:

g = h + A[8]

Solution:

```
lw $t0, 8($s3)
add $s1, $s2, $t0
```

- 8-bit = 1 **bytes** are useful in many programs, most architectures address in bytes.
- The address of a word in MISP (32 bits) matches 4 bytes.
  - addresses of sequential words differ by 4.
- The **<u>alignment restriction</u>** in MIPS, words must start at addresses that are multiples of 4.



3
2
1
0
100
10
101
1
Address
Data
12
8
4
0
100
10
101
1
MIPS
Byte Address
Data

- Byte addressing also affects the array index.
- To get the proper byte address in the code above, the offset to be added to the base register $s3 must be 4 × 8, or 32.
- Therefore, From the previous instruction g = h + A[8] in C, the command **in MIPS** should be

~~**lw** $t0, 8($s3)~~
~~**add** $s1, $s2, $t0~~

lw $t0, 32($s3)
add $s1, $s2, $t0 √

Store Word = The data transfer instruction that copies data from register to a memory

Ex:

```
sw $s1,20($s2)
```

offset

Base register

Mean:

Memory[$s2 + 20] = $s1

(a word from the register is stored in the memory)

Compiling Using Load and Store

- **<u>Ex:</u>** Assume variable h is associated with register $s2 and the base address of the array A is in $s3. What is the **<u>MIPS</u>** assembly code for the C assignment statement:

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

<u>Solution:</u>

```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)
```

MEMORY VS. REGISTER

- register is faster than memory
  - data accesses are faster
  - data is more useful in a register.
  - Register have higher throughput than memory
  - accessing registers uses less energy than accessing memory.
- there are fewer registers

PRINCIPLES OF HARDWARE DESIGN

Design Principle 3:

Make the common case fast

- Many times a program will use a constant in an operation
  - Incrementing an index to point to the next element of an array.
  - more than half of the MIPS arithmetic instructions have a constant as an operand when running

- Ex: add the constant 4 to register $s3

```
lw $t0, AddrConstant4($s1)
add $s3, $s3, $t0
```

Register keeping base address of contrant location

- **addi** (*add immediate*) to add instruction with one **constant operand**

Ex:

addi $s3,$s3,4.

Mean:

$s3 = $s3 + 4

$zero

- Used when one operand is zero
- It is the hardwired register to the value zero.

Ex:

```
add $t2, $s1, $zero
```