Lecture 2
MIPS Instructions

School of Computer Science and Engineering
Soongsil University

2. Instruction: Language of the Compute

2.1 Introduction

- **Instructions**
  - The words of a computer's language
  - A single operation of a processor defined by an instruction set architecture
  - Opcode + operand specifiers
- **Instruction set**
  - The vocabulary of commands understood by a given architecture
  - Set of all instructions that a computer understands
- **RISC vs. CISC**
  - Reduced Instruction Set Computer
  - Complex Instruction Set Computer

Common Goal of the Computer Designers

- To find a language that makes it easy to build the hardware and the compiler, while maximizing performance and minimizing cost and energy.
- **"Simplicity of the equipment"**
  - KISS (Keep it short and simple or Keep it simple, stupid) principle
- **MIPS** (Microprocessor without Interlocked Pipeline Stages)
- **MIPS Technologies, Inc.**
  - 1984: MIPS Computer Systems, Inc. (Prof. Hennessy)
  - 1992: SGI acquires MIPS Computer Systems -> MIPS Technologies, Inc.
  - 1998: SGI decided to migrate to the Itanium architecture
    -> spun out as an intellectual property licensing company
  - 2000: SGI가 완전히 손을 뗌
  - 2013: acquired by Imagination Technologies

MIPS Architectures

Instruction set	Year	Model	Clock	Cache	Transistors
MIPS I	1987	R2000	16 MHz	External (4K+4K ~ 32K+32K)	0.115 M
MIPS III	1991	R4000	100 MHz	8K + 8K	1.35 M
MIPS IV	1992	R8000	90 MHz	16K + 16K	2.6 M
	2002	R14000	500 MHz	32K + 32K and 512K	7.2 M
MIPS V	H1, H2 (planned, but not manufactured)				
MIPS 32	1999	based on MIPS II with some additional features from MIPS III, MIPS IV, and MIPS V			
MIPS 64	1999	based on MIPS V			

2.2 Operations of the Computer Hardware

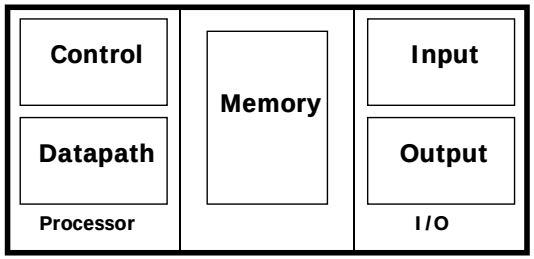
- **MIPS arithmetic instructions**
  - `add a,b,c  # a = b + c`
  - `sub a,b,c  # a = b - c`
- **Example: `f=(g+h)-(i+j);`**

```
add  t0,g,h     # t0 = g + h
add  t1,i,j     # t1 = i + j
sub  f,t0,t1    # f = to - t1 = (g+h)-(i+j)
```

Design Principle 1 : Simplicity favors regularity.
- Always three operands
- Keeping the hardware simple

2.3 Operands of the Computer Hardware

- **Arithmetic instruction's operands must be registers**
  - Only 32 registers provided
- **Compiler associates variables with registers**
- **What about programs with lots of variables**



MIPS Registers: Figure 2.18

- **32 registers**
- **Variables**
  - `$s0, $s1, $s2, … $s7`
- **Temporary registers**
  - `$t0, $t1, $t2, … $t9`

Design Principle 2 : Smaller is faster.
- A very large number of registers
  - Convenient for the programmers
  - But, longer distance to travel for the electronic signals
  - Increased clock cycle time

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

- `int f, g, h, i, j;`
- `f, g, h, i, j  →  $s0, $s1, $s2, $s3, $s4`

[Answer]

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


Memory Operands

- **Arrays and structures**
  - Too many data elements
  - Stored in memory, not in registers
- **Data transfer instructions**
  - load: **`lw`** (load word) in MIPS
  - store: **`sw`** (store word) in MIPS
- **Example: g = h + A[8];**

  base address of A → $s3

  `int g, h, A[100];`

  [Answer]

```
lw   $t0, 8($s3)    # $t0 gets A[8] (actually wrong!)
add  $s1, $s2, $t0  # g = h + A[8]
```


Actual MIPS Memory Structure

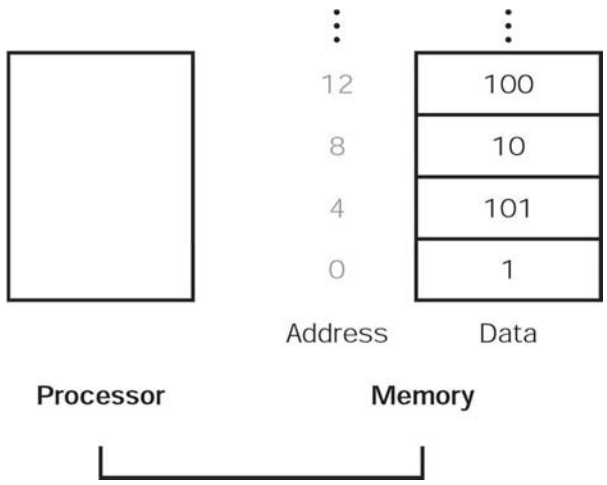


Figure 2.3

Memory and Processing Subsystems

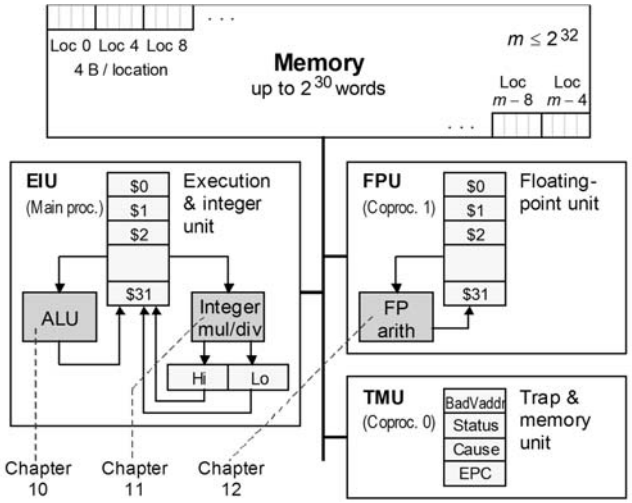
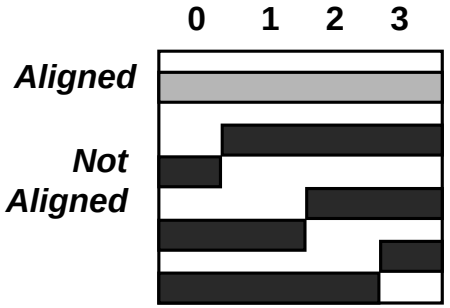


Figure 5.1 of Parhami

Hardware/Software Interface (1/2)

- **Byte addressing**
  - Hardware architectures which support accessing individual bytes of data rather than only words
  - ↔ Word addressing
- **Alignment restriction (aka memory alignment)**
  - Putting the data at a memory address equal to some multiple of the data size
  - Increasing the system's performance due to the way the CPU handles memory



Hardware/Software Interface (2/2)

- **Endianness**
  - From Gulliver's Travels by Jonathan Swift
  - Little endian
    - Using the address of the rightmost (or "little end") byte as the word address
    - Intel IA-32, DEC PDP 11, VAX-11, Alpha, Atmel AVR
  - Big endian
    - Using the address of the leftmost (or "big end") byte as the word address
    - MIPS, IBM S/360 and S/370, Motorola 680x0, SPARC, PA-RISC
  - Bi-endian
    - ARM, IA-64, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC

Endianness and Load

- **Load $s1 from M[200]**
  - `lw $s1,200($zero)`

199	
200	**81**
201	**C5**
202	**3B**
203	**02**
204	
205	

- **Little endian**

$s1	**02**	**3B**	**C5**	**81**

- **Big endian**

$s1	**81**	**C5**	**3B**	**02**

Endianness and Store

- **Store $s2 into M[100]**
  - `sw $s2,100($zero)`

$s2	**0000 0010**	**0011 1011**	**1100 0101**	**1000 0001**

- **Little endian**

99	
100	1000 0001
101	1100 0101
102	0011 1011
103	0000 0010
104	
105	

- **Big endian**

99	
100	0000 0010
101	0011 1011
102	1100 0101
103	1000 0001
104	
105	

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

- **base address of `A` → `$s3`**
- **h → `$s3`**

[Answer]

```
lw   $t0,32($s3)  # Temporary reg. $t0 gets A[8]
add  $t0,$s2,$t0  # Temporary reg. $t0 gets h+A[8]
sw   $t0,48($s3)  # Stores h+A[8] back into A[12]
```

Hardware/Software Interface

- Register spilling

Constant or Immediate Operands

- **Constant operands**
  - More than half of the SPEC2000 operands
- **Adding 4 to $s3 without constant operand**

```
lw  $t0,AddrConstant4($s1) # $t0=constant 4
add $s3,$s3,$t0            # $s3=$s3+$t0 ($t0=4)
```

- **Faster version**

`addi(add immediate)` instruction

```
addi $s3,$s3,4             # $s3=$s3+4
```

- **Making the common case fast.**
  - Constant operands are frequently used.
  - Arithmetic operations with constant operands.