중간고사 관련

- **가반**
  - 4월 28일 (수) 13:30 - 14:45
- **나반**
  - 4월 28일 (수) 15:00 – 16:15
- **다반**
  - 4월 28일 (수) 09:00 – 10:15

숭실대학교
Soongsil University

Computer Architecture

과제 #1: ARM Instructions 분석

2021년 1학기
Young Geun Kim (김영근)

Example

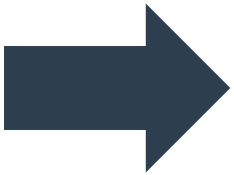
◆ E5820010

- ➢ Instruction을 Binary로 변환
  - 1110 01**0**1 1000 0010 0000 0000 0001 0000 (2)
- ➢ 어떤 Instruction인지 Reference File을 통해 확인
  - STR $0, [$2, #0x010];
  - 25번째 bit인 I bit가 0이므로, #으로 표기 (= Immediate Offset)
- ➢ Instruction이 어떤 의미를 가지는지 서술
  - 2번 Register에 저장된 값에 #0x010값을 더해 주소 값 계산
  - 0번 Register에 저장되어 있던 값을 위에서 계산한 메모리 주소에 저장

General Execution Flow

```c
#include <stdio.h>

int function(int a);

int main()
{
  int x=5;
  int y;

  y = function(x);

  printf("y=%i\n", y);

  return 0;
}

int function(int a)
{
  return 3*a+5;
}
```

C언어 프로그램
(example.c)



컴파일러

```
# Simple routine to demo functions
# USING a stack in this example to preserve
# values of calling function

# ----------------------------------------------------------------

        .text

        .globl  main
main:
        # Register assignments
        # $s0 = x
        # $s1 = y

        # Initialize registers
        lw      $s0, x          # Reg $s0 = x
        lw      $s1, y          # Reg $s1 = y

        # Call function
        move    $a0, $s0        # Argument 1: x ($s0)
        jal     fun             # Save current PC in $ra, and jump to fun
        move    $s1,$v0         # Return value saved in $v0. This is y ($s1)

        # Print msg1
        li      $v0, 4          # print_string syscall code = 4
        la      $a0, msg1
        syscall

        # Print result (y)
        li      $v0,1           # print_int syscall code = 1
        move    $a0, $s1        # Load integer to print in $a0
        syscall

        # Print newline
        li      $v0,4           # print_string syscall code = 4
        la      $a0, lf
        syscall

        # Exit
        li      $v0,10          # exit
        syscall

# ----------------------------------------------------------------

        # FUNCTION: int fun(int a)
        # Arguments are stored in $a0
        # Return value is stored in $v0
        # Return address is stored in $ra (put there by jal instruction)
        # Typical function operation is:

fun:    # This function overwrites $s0 and $s1
        # We should save those on the stack
        # This is PUSH'ing onto the stack
        addi $sp,$sp,-4         # Adjust stack pointer
        sw $s0,0($sp)           # Save $s0
        addi $sp,$sp,-4         # Adjust stack pointer
        sw $s1,0($sp)           # Save $s1

        # Do the function math
        li $s0, 3
        mul $s1,$s0,$a0         # s1 = 3*$a0  (i.e. 3*a)
        addi $s1,$s1,5          # 3*a+5

        # Save the return value in $v0
        move $v0,$s1

        # Restore saved register values from stack in opposite order
        # This is POP'ing from the stack
        lw $s1,0($sp)           # Restore $s1
        addi $sp,$sp,4          # Adjust stack pointer
        lw $s0,0($sp)           # Restore $s0
        addi $sp,$sp,4          # Adjust stack pointer

        # Return from function
        jr $ra                  # Jump to addr stored in $ra

# ----------------------------------------------------------------

        # Start .data segment (data!)
        .data
x:      .word 5
y:      .word 0
msg1:   .asciiz "y="
lf:     .asciiz "\n"
```

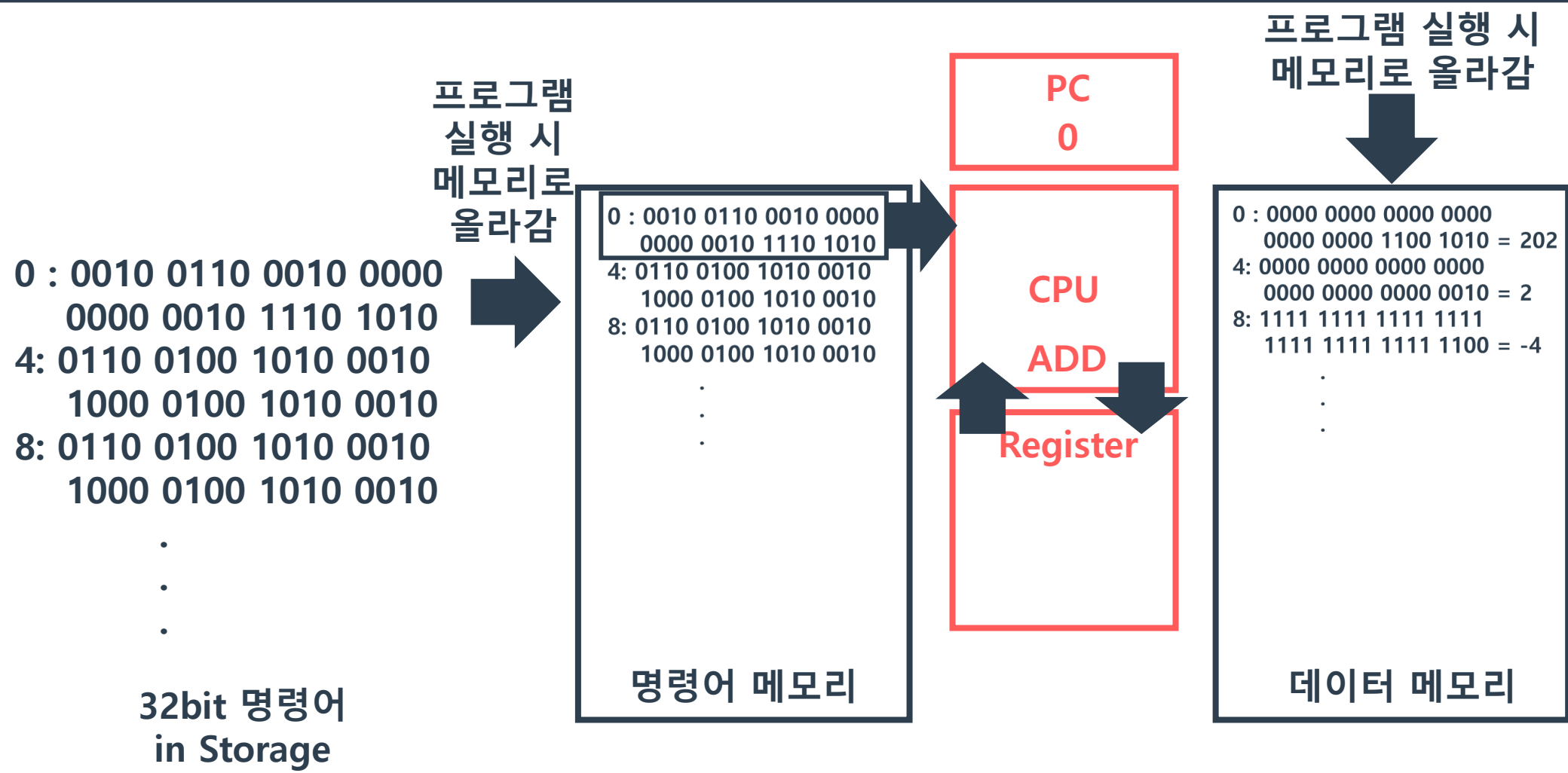
```
0 : 0010 0110 0010 0000
    0000 0010 1110 1010
4: 0110 0100 1010 0010
    1000 0100 1010 0010
8: 0110 0100 1010 0010
    1000 0100 1010 0010
    .
    .
    .
```

MIPS ISA

MIPS Assembly (example.asm)

32bit 명령어

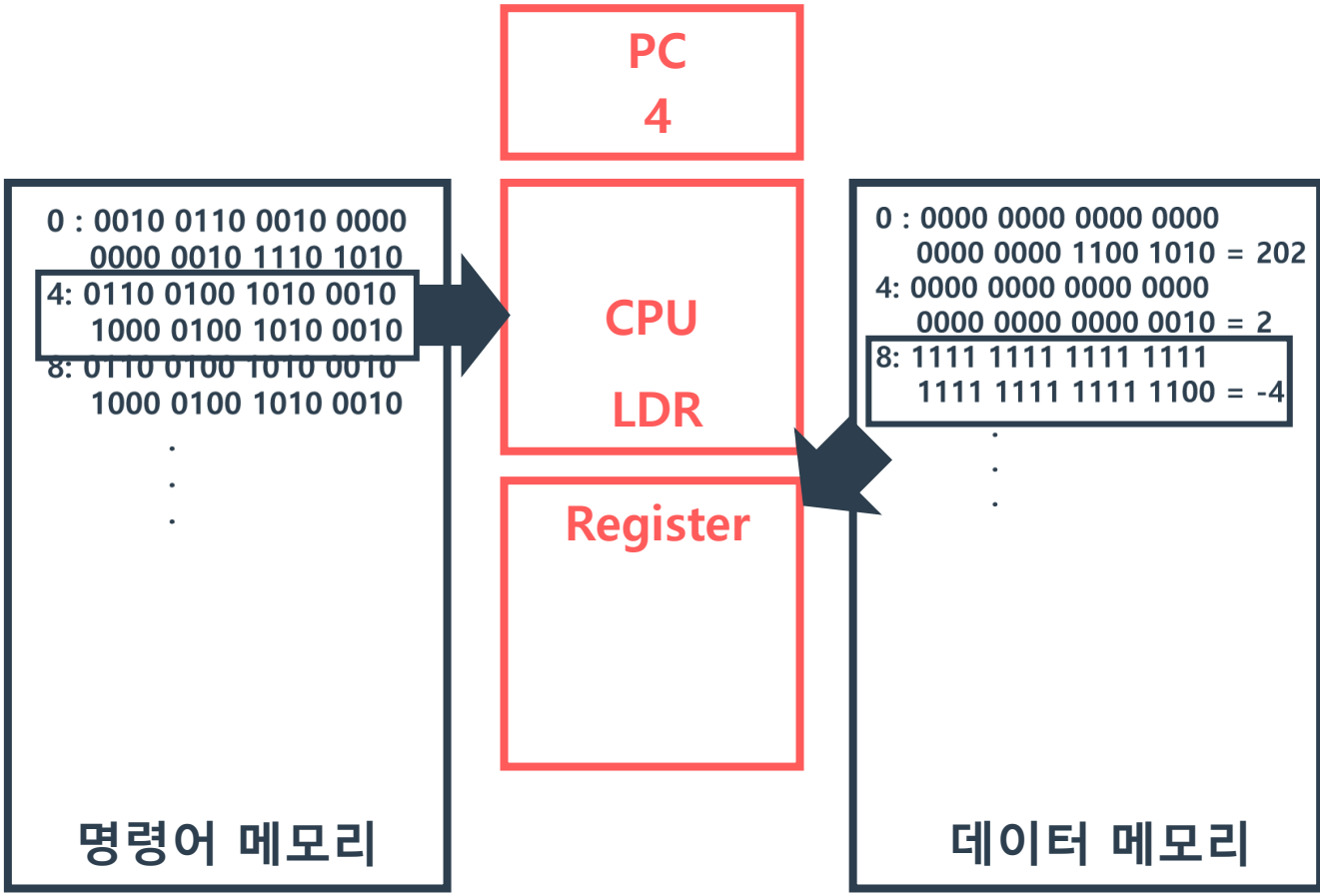
General Execution Flow (Cont'd)



General Execution Flow (Cont'd)

0 : 0010 0110 0010 0000
0000 0010 1110 1010
4: 0110 0100 1010 0010
1000 0100 1010 0010
8: 0110 0100 1010 0010
1000 0100 1010 0010
.
.
.

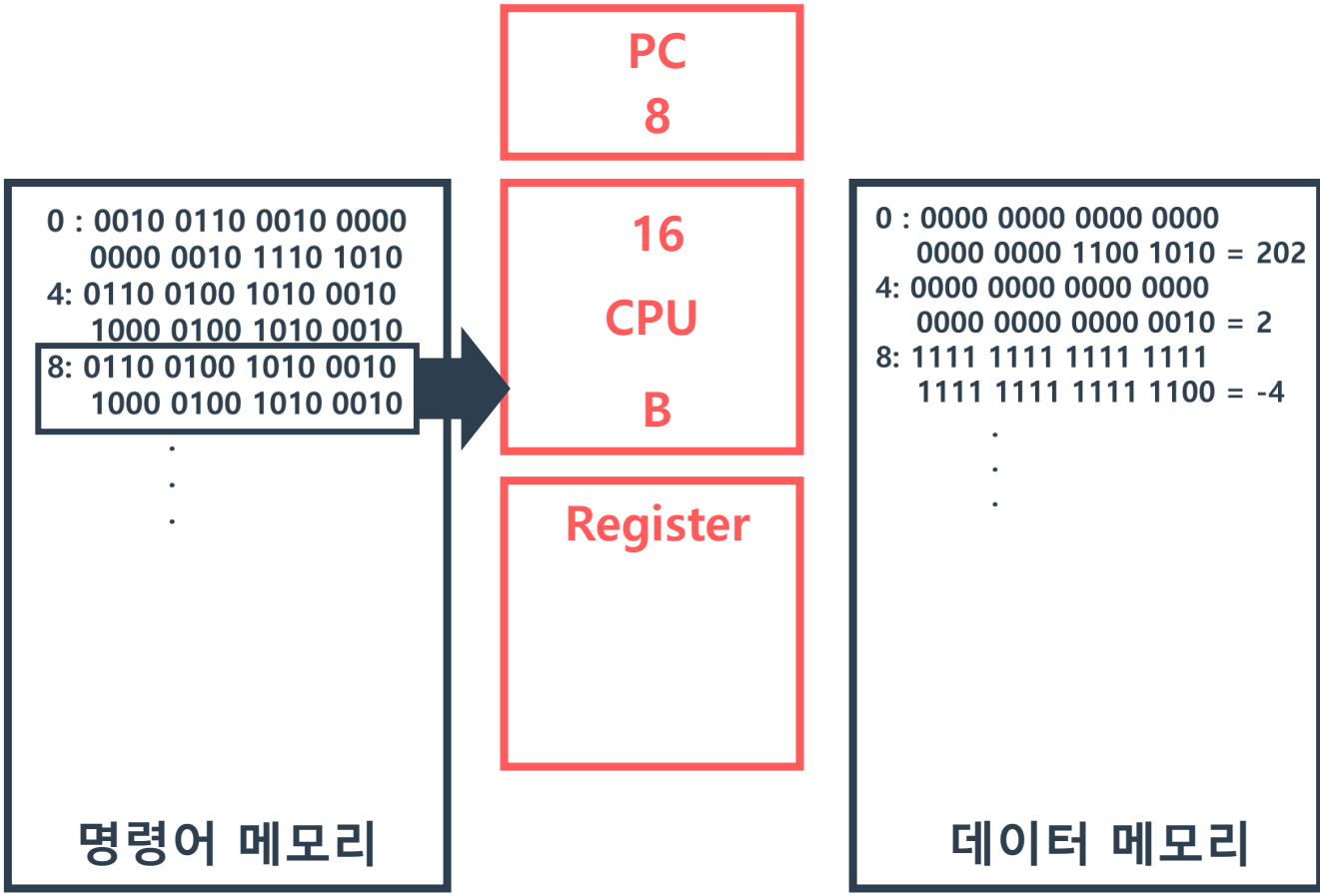
32bit 명령어
in Storage



General Execution Flow (Cont'd)

0 : 0010 0110 0010 0000
0000 0010 1110 1010
4: 0110 0100 1010 0010
1000 0100 1010 0010
8: 0110 0100 1010 0010
1000 0100 1010 0010
.
.
.

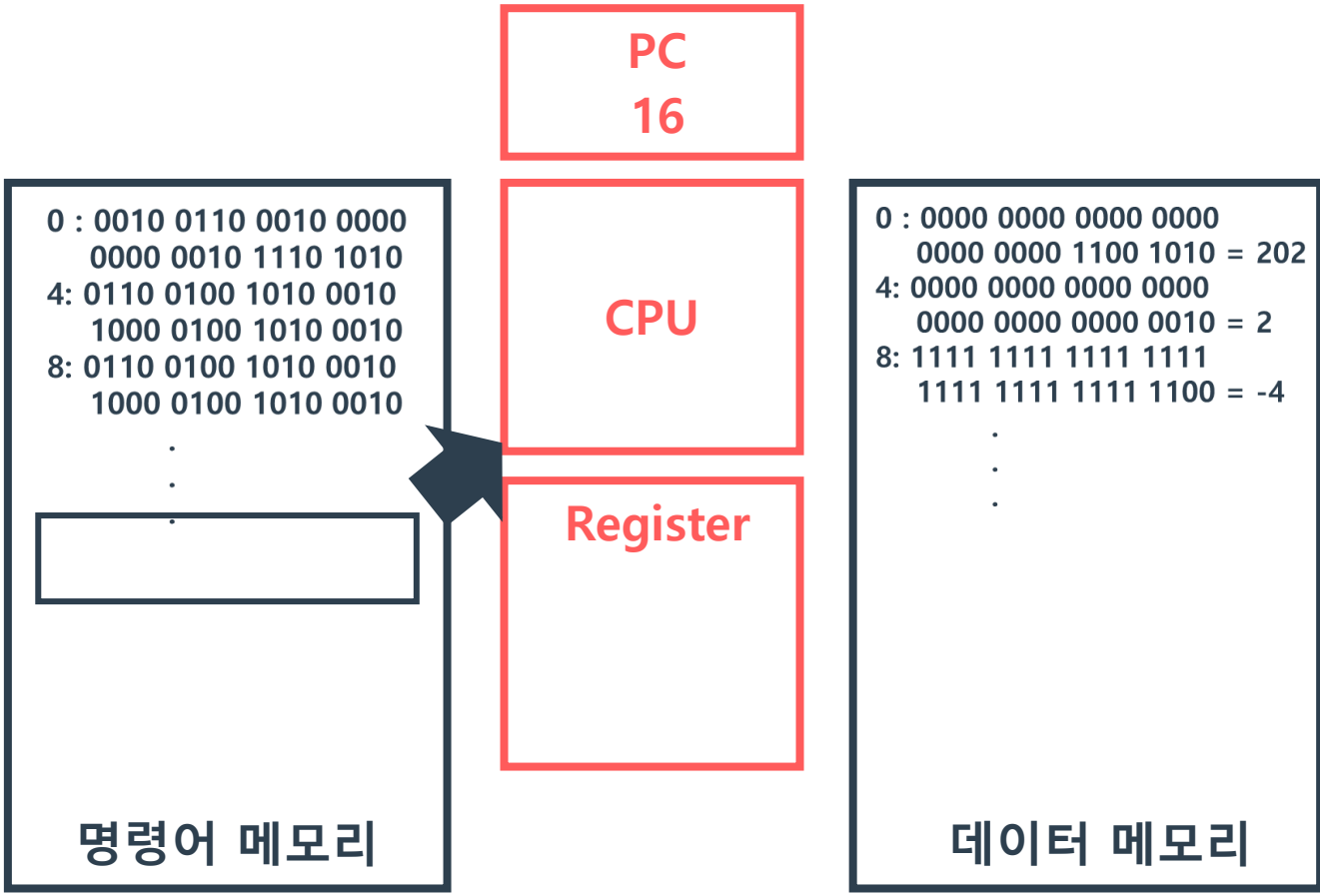
32bit 명령어
in Storage



General Execution Flow (Cont'd)

0 : 0010 0110 0010 0000
0000 0010 1110 1010
4: 0110 0100 1010 0010
1000 0100 1010 0010
8: 0110 0100 1010 0010
1000 0100 1010 0010
.
.
.

32bit 명령어
in Storage



Example

◆ EA000006

- Instruction을 Binary로 변환
  - 1110 **101**0 0000 0000 0000 0000 0000 0110 $_{(2)}$
- 어떤 Instruction인지 Reference File을 통해 확인
  - B **#6**;
- Instruction이 어떤 의미를 가지는지 서술
  - PC+8+**6**x4 주소로 이동 (MIPS의 경우 +4이므로 주의)
    - PC는 현재 실행 중인 명령어의 주소를 의미함
    - 주소 단위가 4 Byte (= 1 Word)이기 때문임
  - 따라서, (0 + 8 + 24) / 4 = 008번 주소로 이동
    - 다음 Instruction은 008번 주소에 있는 E59F2EC8이 됨

A4.1.5 B, BL