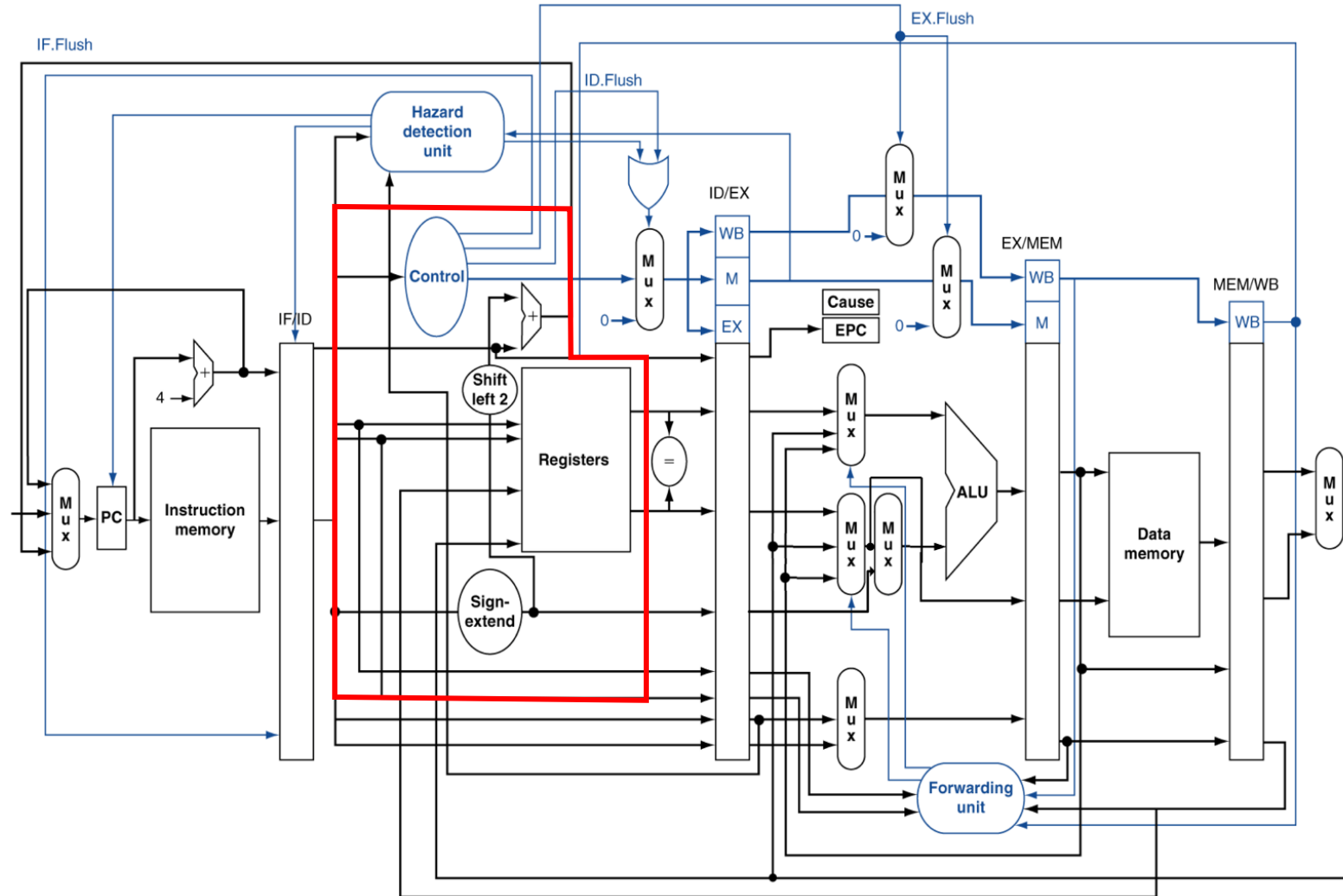

MIPS Instruction Decode Stage

조교 박현재

개요

- Introduction
- Instruction Decode Stage
 - Instruction Register(Memory)
 - Instruction Controller
 - Sign Extension
- 과제

Introduction



Instruction Decode Stage

•Instruction Decode Stage

•명령어 메모리에 접근 후 불러오기

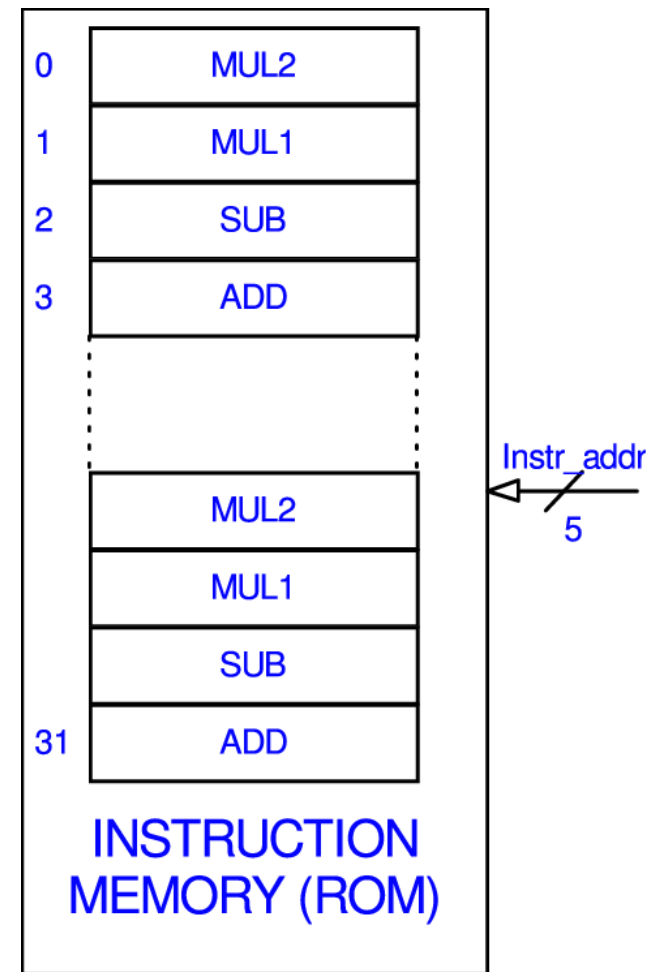
메모리의 Address: PC(Program Counter)
메모리의 Data: Instruction(실행 할 명령어)

• 불러온 명령어에 따라 제어 신호 할당

- RISC 구조는 정해진 명령어 Format에 맞춰 신호가 발생

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt	immediate		
	31 26 25	21 20	16 15	0		
J	opcode	address				
	31 26 25	0				



Instruction Register(Memory)

- **Instruction Register란?**

- 가장 최근에 Fetch된 명령어를 저장하는 레지스터.
- 현재 실행 중인 명령어와는 PC - 4만큼의 차이가 있음.

Instruction Memory	
메모리 번호	내부 Data(32bit)
0번째	0x012A_4020(add \$t1 \$t2 \$t3)
1번째	0x01AC_5022(sub \$t4 \$t5 \$t6)
2번째	0x8F07_0064(lw \$t7 100(\$t8))
...	...
128번째	0xAD39_00C8(sw \$t9 200(\$t1))

Instruction Register(Memory)

- PC(Program Counter)를 이용하여 Instruction Memory에 접근하기
 1. PC는 총 32bit, 하지만 하위 4bit는 실제 Addressing에 사용되지 않음.
 2. 메모리 번호는 최대 127까지만(7bit) 허용

➔ PC(Program Counter)의 [8:2]만 Addressing에 사용

PC(Program Counter) Binary

Init. 0000_0000_0000_0000_0000_0000_0000_0000

Next 0000_0000_0000_0000_0000_0000_0000_0100

Next 0000_0000_0000_0000_0000_0000_0000_1000

...

Next 0000_0000_0000_0000_0000_0000_1_1111_1100

PC+4



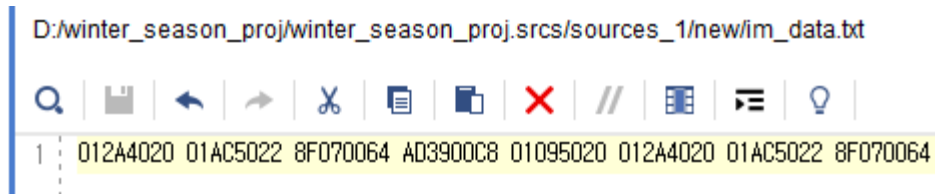
1_1111_11

Instruction Memory	
메모리 번호	내부 Data(32bit)
0번째	0x012A_4020(add \$t1 \$t2 \$t3)
1번째	0x01AC_5022(sub \$t4 \$t5 \$t6)
2번째	0x8F07_0064(lw \$t7 100(\$t8))
...	...
127번째	0xAD39_00C8(sw \$t9 200(\$t1))



Instruction Register(Memory)

•Verilog 상에서 Instruction Memory를 읽기



D:/winter_season_proj/winter_season_proj.srscs/sources_1/new/im_data.txt

1 012A4020 01AC5022 8F070064 AD3900C8 01095020 012A4020 01AC5022 8F070064

공백으로 단어를 구분

\$readmh

Verilog Test Bench

```
parameter NMEM = 128;
parameter IM_DATA = "im_data.txt";

reg [31:0] mem [0:127];

initial begin
    $readmemh(IM_DATA, mem, 0, NMEM-1);
end
```

32bit의 128개의 배열을 가진 mem register 내에 im_data.txt의 정보를 저장.

Instruction Controller

- MIPS 설계에서 사용할 Instruction Set Architecture

R-Format(Register Format): Register에 값을 쓰거나 읽는 명령어

OpCode: Format을 구분해주는 값

rs rt: Source Register

rd: Destination Register

shamt: Shift Amount(Shift가 아닐 시 0)

I-Format(Immediate Format): Immediate(정수)를 활용하는 명령어

OpCode: Format을 구분해주는 값

rs rt: Source Register

Immediate: 16bit의 정수 값

J-Format(Jump Format): Jump/Branch와 관련된 명령어

Address: 이동할 Address 주소

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt	immediate		
	31 26 25	21 20	16 15	0		
J	opcode	address				
	31 26 25					

Instruction Controller

- MIPS 설계에서 사용할 Instruction Set Architecture

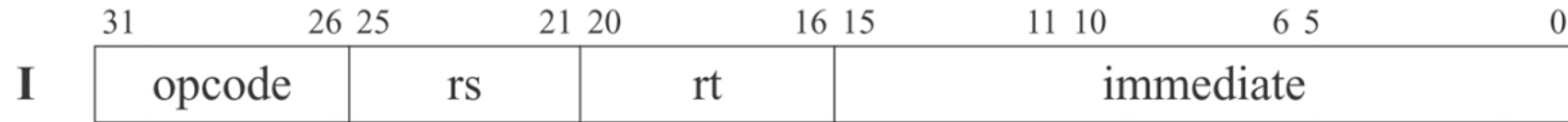
I-Format(Immediate Format): addi, lw, sw

J-Format(Jump Format): beq, bne, j(jump)

Instruction	OP-Code	alu_op	branch_eq	branch_ne	memread	memwrite	memtoreg	regdst	regwrite	alusrc	jump
addi	8	0						0		1	
lw	35	0			1		1	0			
sw	43	0				1			0	1	
beq	4	1	1					0		1	
bne	3	1		1					0		
j	2										1
R-Format	0										
Defaults	-	2	0	0	0	0	0	1	1	0	0

Sign Extension

- MIPS의 Sign Extension



I-Format상 16bit의 Immediate Constant → 32bit로 계산 할 필요가 있음.
따라서 MSB(Most Significant Bit)를 확장하여 사용.

0010 -> 0000 0010

1010 -> 1111 1010

과제

- Control 설계 및 테스트 벤치 작성

```
module control(  
    input wire [5:0] opcode,  
    output reg branch_eq, branch_ne,  
    output reg [1:0] aluop,  
    output reg memread, memwrite, memtoreg,  
    output reg regdst, regwrite, alusrc,  
    output reg jump);
```

```
module control_tb;  
  
    // Inputs  
    reg [5:0] opcode;  
  
    // Outputs  
    wire branch_eq, branch_ne;  
    wire [1:0] aluop;  
    wire memread, memwrite, memtoreg;  
    wire regdst, regwrite, alusrc;  
    wire jump;
```

- Instruction Memory와 control 연결 모듈 설계 및 테스트 벤치 작성

```
module im_control_integration(  
    input wire clk,  
    input wire [31:0] addr,  
    output wire branch_eq, branch_ne,  
    output wire [1:0] aluop,  
    output wire memread, memwrite, memtoreg,  
    output wire regdst, regwrite, alusrc,  
    output wire jump  
);
```

```
module im_control_integration_tb;  
  
    // Inputs  
    reg clk;  
    reg [31:0] addr;  
  
    // Outputs  
    wire branch_eq, branch_ne;  
    wire [1:0] aluop;  
    wire memread, memwrite, memtoreg;  
    wire regdst, regwrite, alusrc;  
    wire jump;
```

과제(선택)

- Instruction Memory/control/Program Counter 연결 모듈 설계 및 테스트 벤치 작성

```
module pc_im_control_integration(  
    input wire clk,  
    input wire rst_n,  
    input wire stall_s1_s2, pcsrc, jump_s4,  
    input wire [31:0] jaddr_s4, baddr_s4,  
    output wire branch_eq, branch_ne,  
    output wire [1:0] aluop,  
    output wire memread, memwrite, memtoreg,  
    output wire regdst, regwrite, alusrc,  
    output wire jump  
);
```

```
module pc_im_control_integration_tb;  
  
    // Inputs  
    reg clk;  
    reg rst_n;  
    reg stall_s1_s2;  
    reg pcsrc;  
    reg jump_s4;  
    reg [31:0] jaddr_s4;  
    reg [31:0] baddr_s4;  
  
    // Outputs  
    wire branch_eq, branch_ne;  
    wire [1:0] aluop;  
    wire memread, memwrite, memtoreg;  
    wire regdst, regwrite, alusrc;  
    wire jump;
```