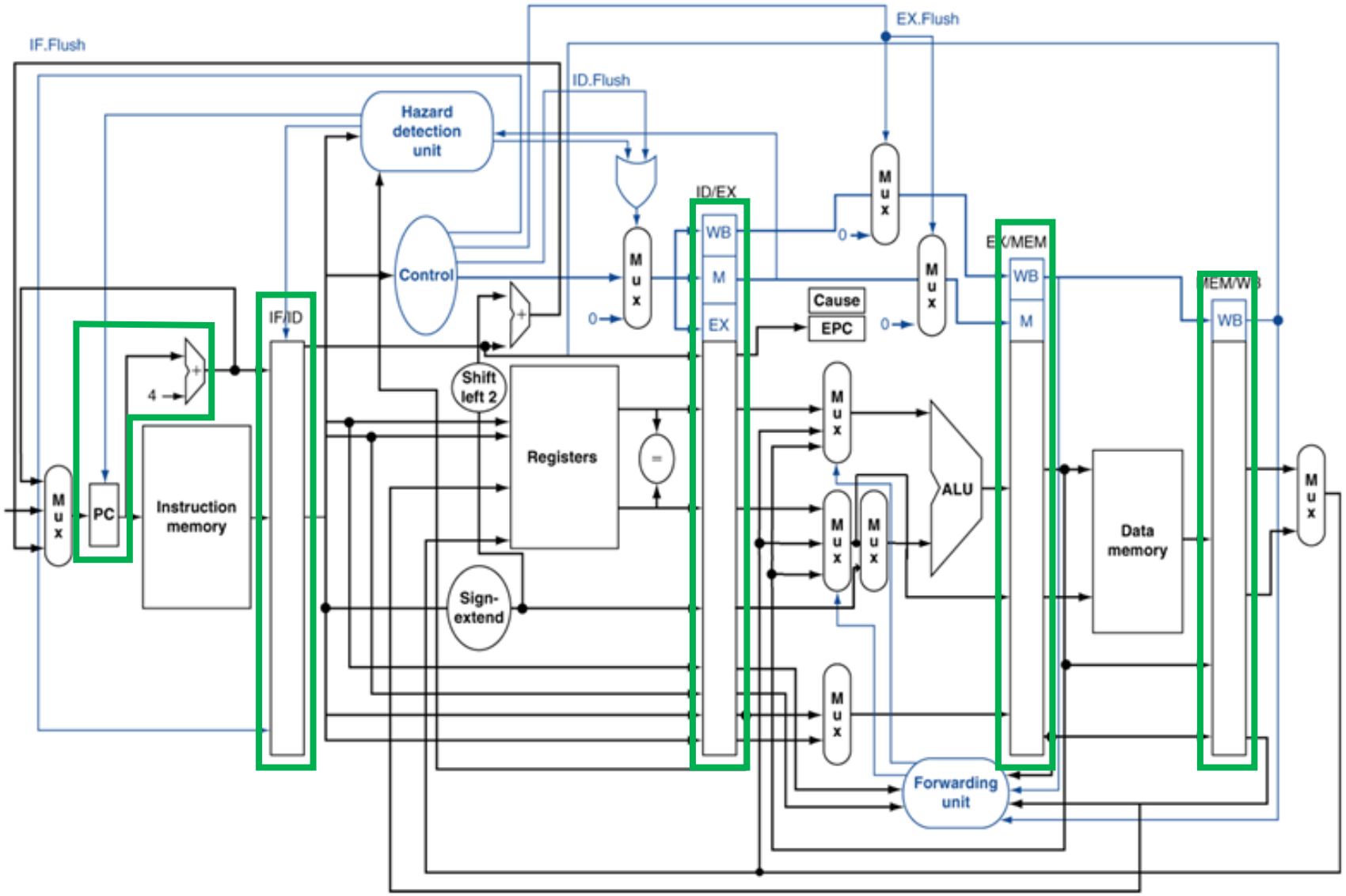
MIPS Execute Stage

조교 서유권

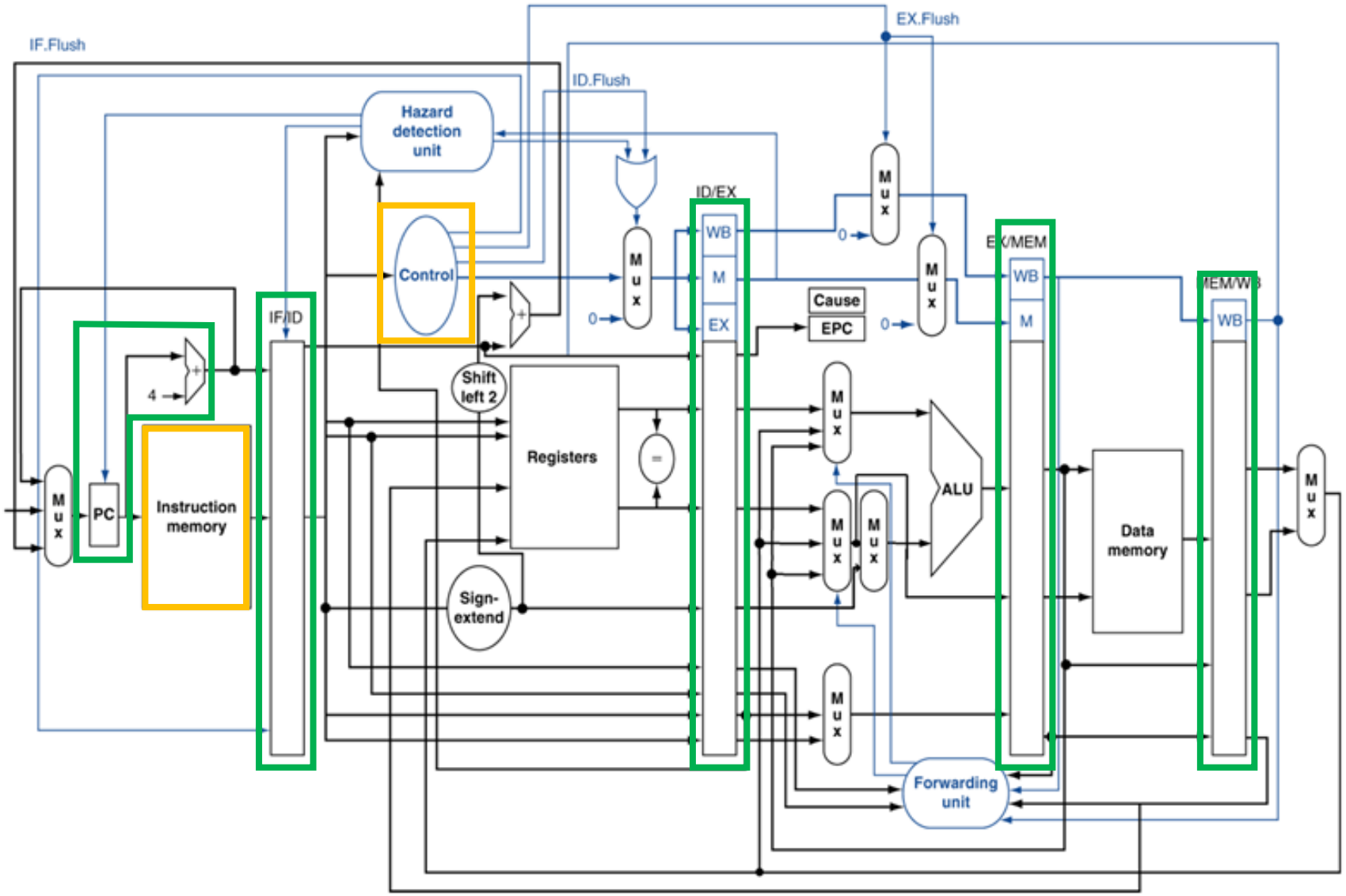
개요

- Introduction
- ID stage – Code correction
- Execute Stage
  - ALU Controller
  - ALU
- 과제

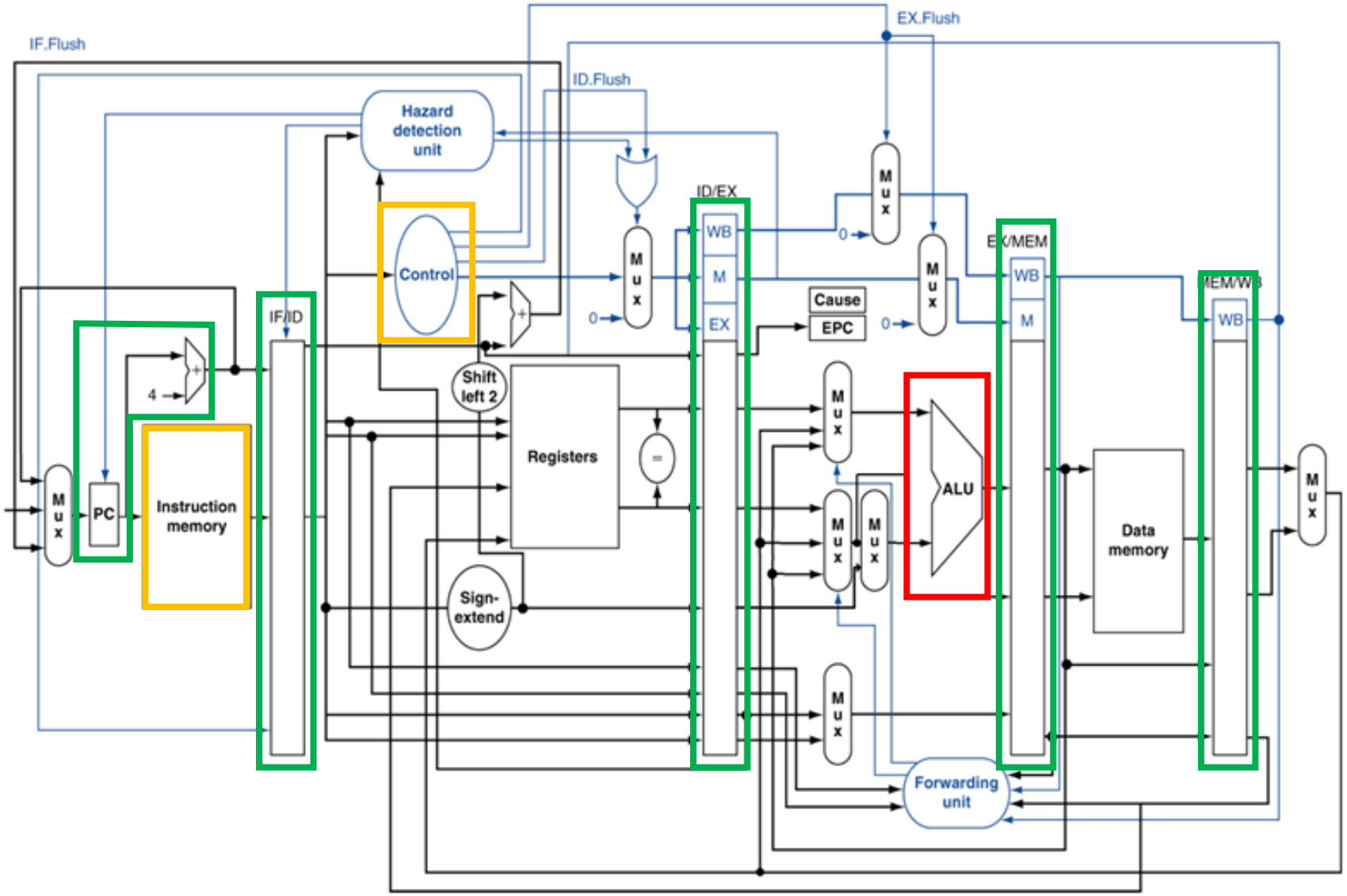
Introduction



Introduction



Introduction



ID stage – Code correction

- Controller 동작 수정

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**

ID stage – Code correction

- Controller 동작 수정

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		**1**	
sw	43	0				1			0	1	
beq	4	1	1					~~0~~	**0**	~~1~~	
bne	3	1		1					0		
j	2										1
R-Format	0										
Defaults	**-**	**2**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

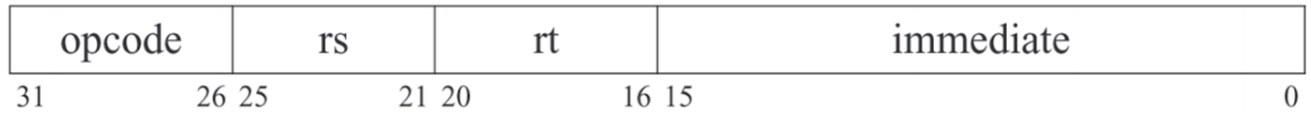
Execute Stage

- ALU를 이용하여 필요한 연산 수행
- ALU(Arithmetic and Logical Unit)란?
  - 사칙연산, 논리연산 등 여러 연산을 진행하는 장치
  - 출력
    - 연산 결과
    - 플래그
      - 부호 플래그
      - 제로 플래그
      - 오버플로우 플래그

Execute Stage

- ALU Controller
  - 주어진 명령어에 따라 ALU가 어떤 동작을 할지 제어
  - **ALU_OP**

```
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);
```



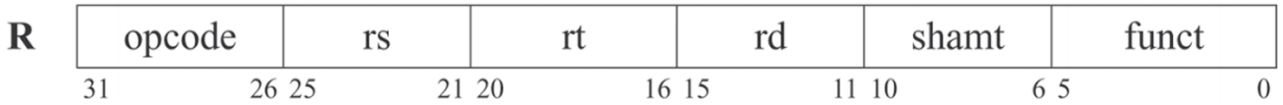
Instruction	OP-Code	alu_op
addi	8	0
lw	35	0
sw	43	0
beq	4	1
bne	3	1

- addi : 덧셈 연산
- lw / sw : 메모리 주소 연산(lw: 메모리 → 레지스터, sw: 레지스터 → 메모리)
  - lw $t0, **20($s3)** → $s3 + 20 위치의 값을 load
- beq / bne : 비교 연산
  - 두 값을 **빼서** 0인지 아닌지 확인
  - 0이면 eq, 아니면 ne 조건 성립
    - 조건 성립하면, **pcsrc** 신호 발생

```
module program_counter(
    input clk,
    input stall_s1_s2, pcsrc, jump_s4,
    input [31:0] pc,
    output [31:0] pc_out
    );
```

Execute Stage

- ALU Controller
  - 주어진 명령어에 따라 ALU가 어떤 동작을 할지 제어
  - **funct**



Instruction	OP-Code	alu_op									
R-Format	0										
Defaults	**-**	**2**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**

	add	sub	or	xor	nor	slt	and
funct	0	2	5	6	7	10	Default
aluctl	2	6	1	13	12	7	0

Execute Stage

- ALU
  - 32비트 데이터 2개를 입력으로 받아 ALU controller의 ctl 신호에 따라 연산 수행 후, 결과 출력
  - 총 7가지 연산 수행

ctl	연산
2	Add
6	Sub
0	AND
1	OR
12	NOR
13	XOR
7	Set less than
default	0 출력

- Set less than
  - A가 B보다 작으면, 1 아니면 0 출력

과제

- ALU Controller 설계 및 Testbench 작성 (9~10 페이지)

```
module alu_control(
    input wire [5:0] funct,
    input wire [1:0] aluop,
    output reg [3:0] aluctl
    );
```

```
module tb_alu_control();

    // inputs
    reg [5:0]   funct;
    reg [1:0]   aluop;

    // outputs
    wire [3:0]  aluctl;
```

- ALU 설계 및 Testbench 작성 (11 페이지)

```
module alu(
    input       [3:0]   ctl,
    input       [31:0]  a, b,
    output reg  [31:0]  out,
    output              zero
    );
```

- 덧셈, 뺄셈: **+, - 사용**하여 구현
- **Overflow Detect** 신호 구현(내부적으로만 구현)
- Set less than: **>, < 쓰지 않고** 구현
  - **A, B, (A-B)의 부호 비트**만 사용
  - **==, !=** 사용 가능

```
module tb_alu();

    // inputs
    reg [3:0]   ctl;
    reg [31:0]  a, b;

    // outputs
    wire [31:0] out;
    wire        zero;
```

과제(선택)

- ALU + ALU Controller 설계 및 Testbench 작성

```
module alu_alu_control_integration(
    input wire [5:0] funct,
    input wire [1:0] aluop,
    input      [31:0]  a, b,
    output reg [31:0]  out,
    output             zero
    );
```

```
module tb_alu_aluctl();

    // inputs - ALU Controller
    reg [5:0]   funct;
    reg [1:0]   aluop;

    // inputs - ALU
    reg [31:0]  a, b;

    // outputs
    wire [31:0]  out;
    wire         zero;
```