

---

# MIPS Forwarding & Hazard Detection

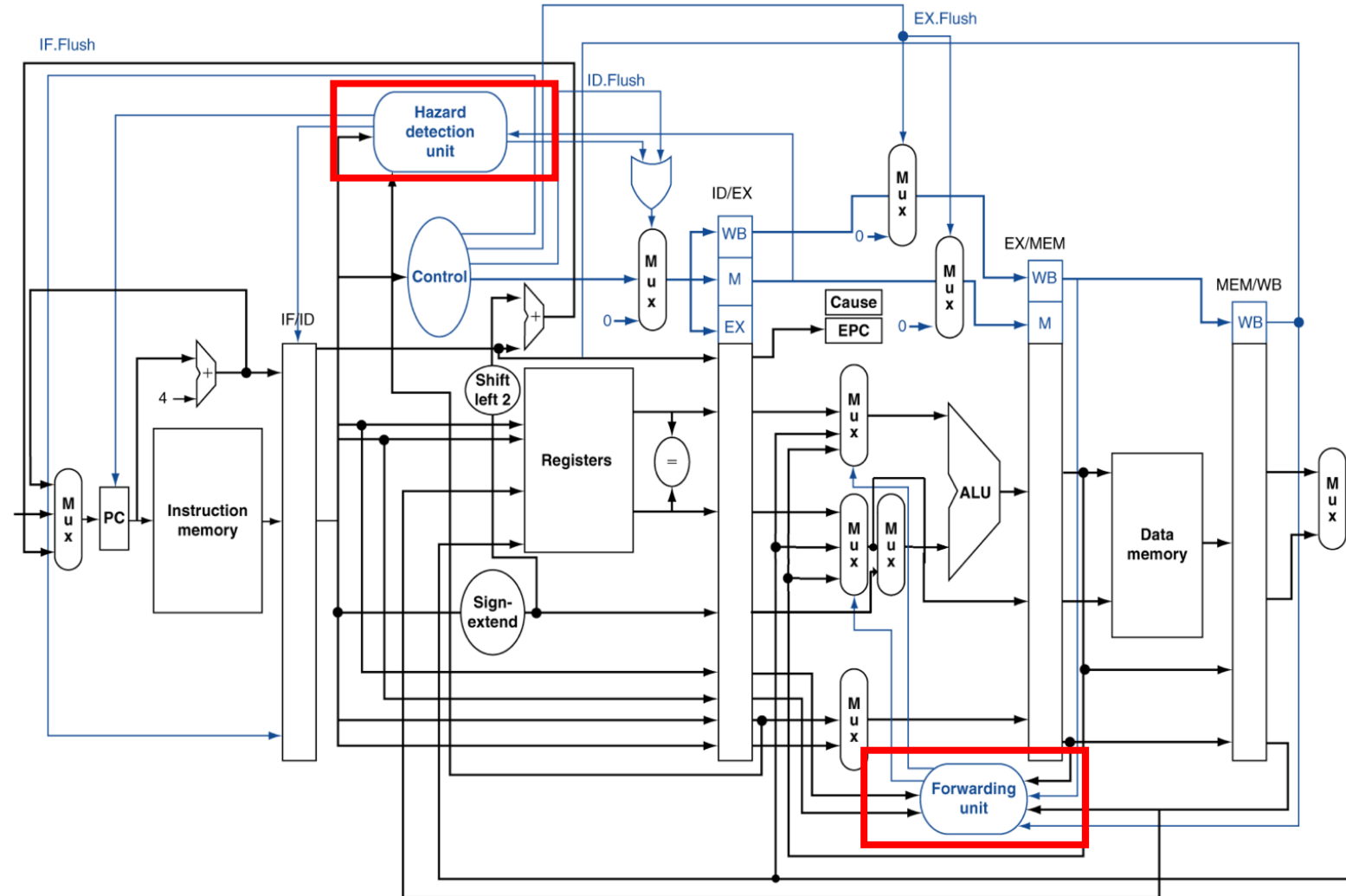
조교 박현재

# 개요

---

- Introduction
- Types of Hazard
  - Structural Hazard
  - Data Hazard by Dependency & Forwarding
  - Data Hazard by Load-Use & Stall
  - Control Hazards & Stall
  - Branch Prediction & Flush
- 과제

# Introduction



# Hazard – Types of Hazards

---

- **Hazard**

매 사이클마다 새로운 Instruction의 수행을 방해하는 요소들

- **Structural Hazards**

HW 설계 이슈로 발생하는 Hazard

- **Data Hazards**

같은 Data를 두고 두 명령어가 Race를 할 경우 발생하는 Hazard

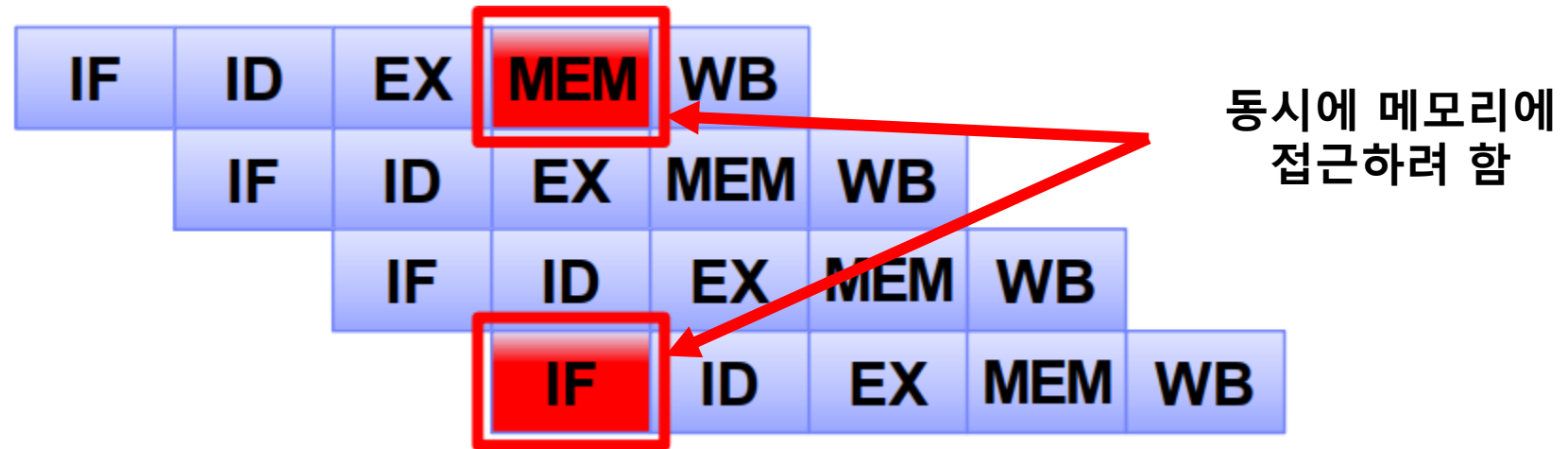
- **Control(Branch) Hazards**

Branch 명령어에서 발생하는 Hazard

# Structural Hazard

- **Structural Hazard란?**

- HW상 동일한 resource를 두고 경쟁을 벌이는 경우.



- Stall로 해결하기엔 너무 많은 Stall들이 요구됨.  
➔ HW 상으로 IF(Instr. Fetch) Memory와 MEM Memory를 분리하여 해결

or **Cache**를 사용

# Data Hazard by Dependency & Forwarding

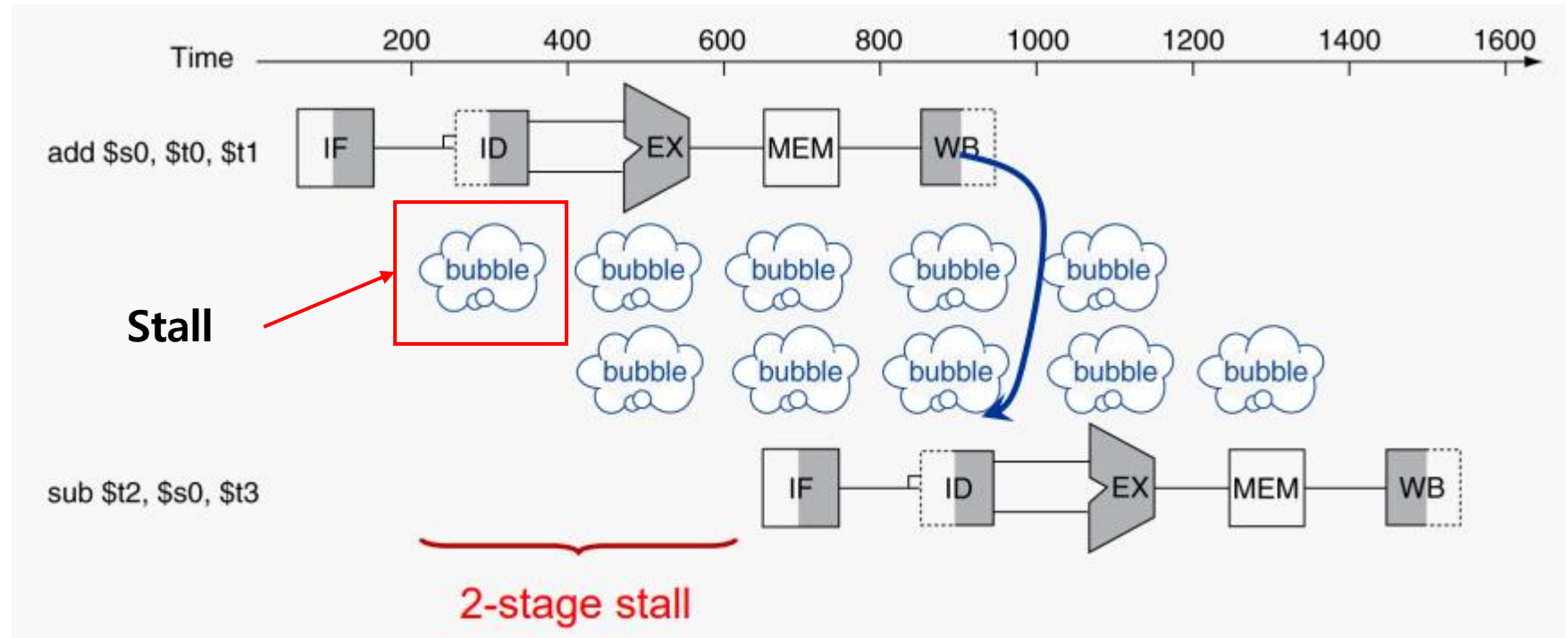
- **Dependency**

앞서 수행한 Instr.의 결과가 나중에 수행되는 Instr.에 영향을 미치는 경우

- **Data Hazard**

→ Dependency로 인해 정확한 명령어를 읽어오지 못하는 상황

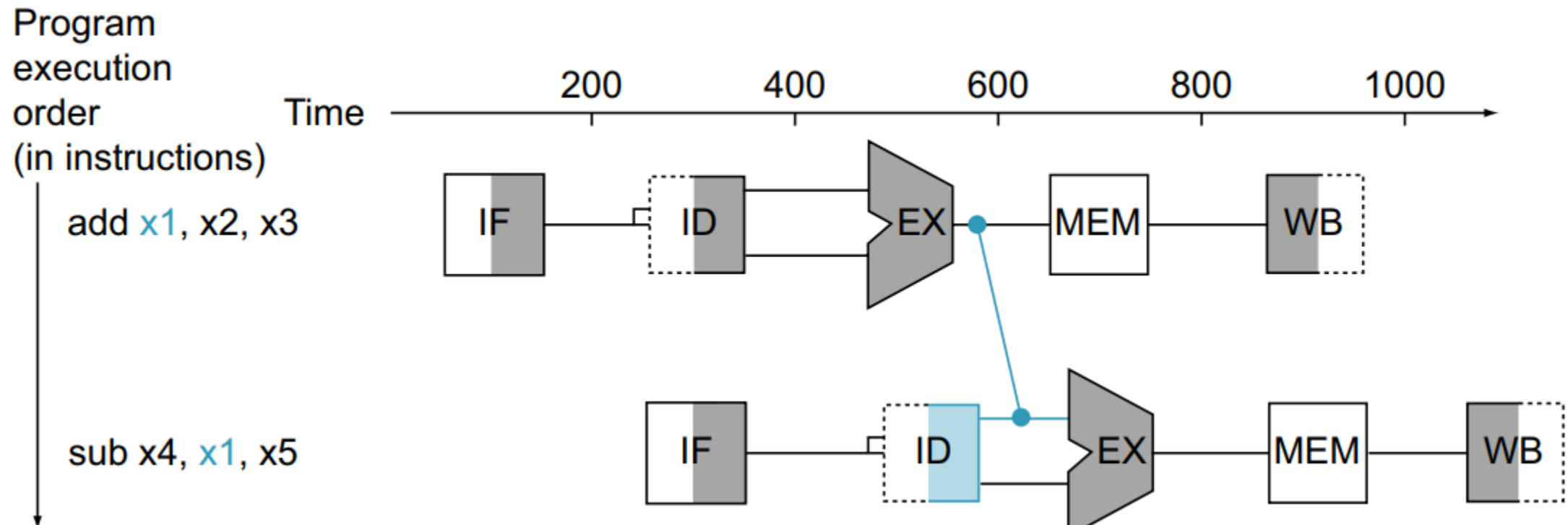
```
add $s0, $t0, $t1  
sub $t2, $s0, $t3
```



# Data Hazard by Dependency & Forwarding

- **Forwarding**

Data의 실제 값이 결정되는 순간에 Data를 전달



# Data Hazard by Dependency & Forwarding

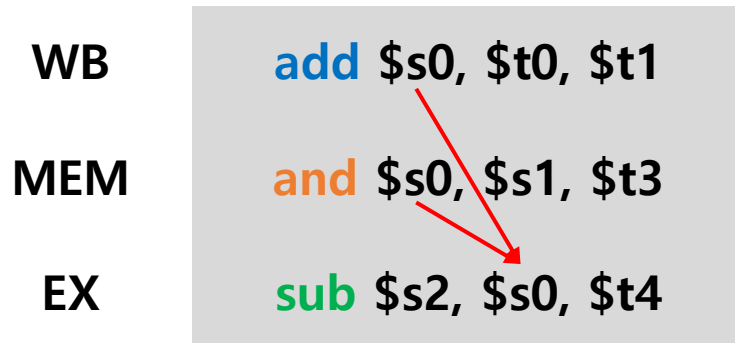
- Forwarding Truth Table

Ex/Mem의 regWrite	Ex/Mem의 Destination = ID/Ex의 Source	Mem/WB의 regWrite	MEM/WB의 Destination = ID/EX의 Source	Forward A	Source
1	1	X	X	2	Ex/Mem
x	0	1	1	1	Mem/WB
0	x	1	1	1	Mem/WB
Otherwise				0	ID/EX



# Data Hazard by Dependency & Forwarding

- Double Data Hazard



`add`와 `and`의 Forwarding 값

WB의 RegWrite = 1

WB의 rd = EX의 Source

→ Forwarding = 01

`and`와 `sub`의 Forwarding 값

Mem의 RegWrite = 1

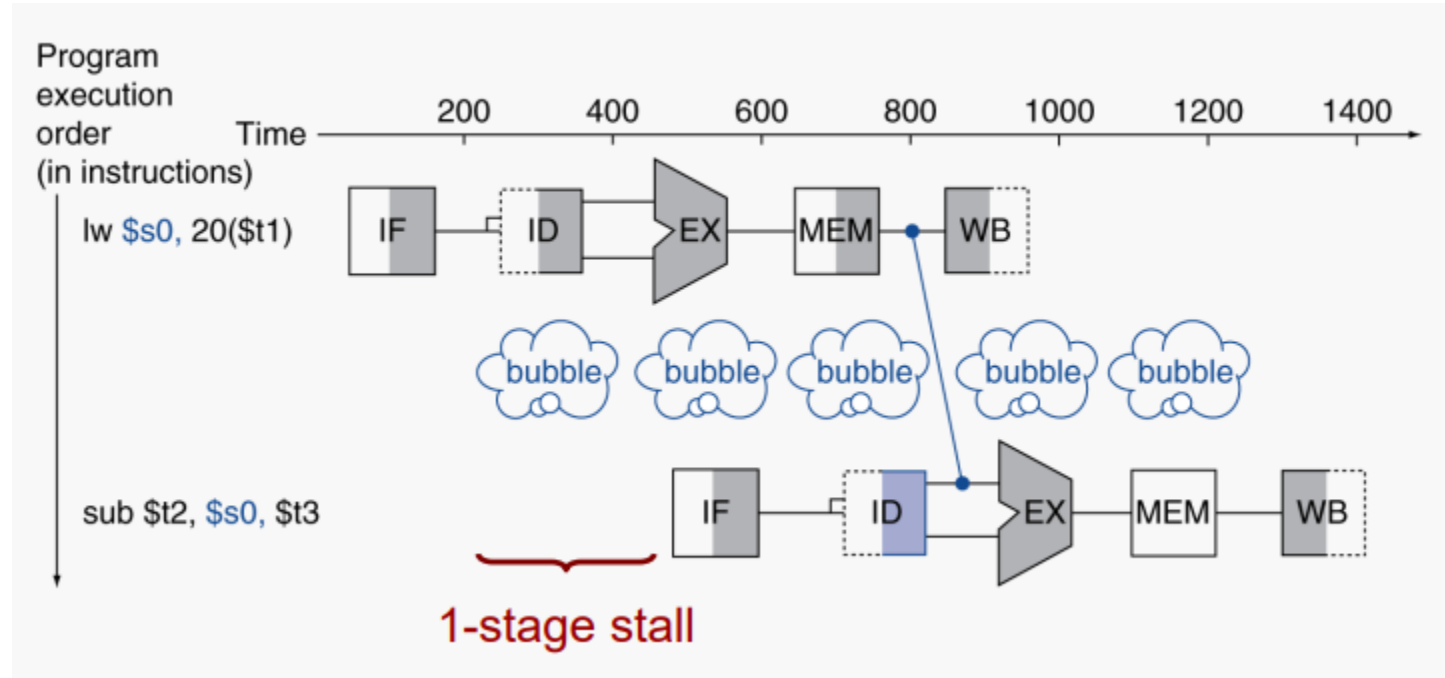
Mem의 rd = EX의 Source

→ Forwarding = 2

Forwarding 값과 관계 없이  
가까운 Forward를 수행

# Data Hazard by Load-Use & Stall

- Load Instr.의 한계와 Stall




# Data Hazard by Load-Use & Stall

- Stall 조건

lw \$1, 16(\$3) ; I-type, rt\_s3 = \$1, memread\_s3 = 1

add \$2, \$1, \$1 ; R-type, rs\_s2 = \$1, rt\_s2 = \$1, memread\_s2 = 0



**Stage 3(Memory) memread = True**  
**Stage 3's Target = Target or Source**

**➔ Stall = 1**

# Data Hazard by Load-Use & Stall

## • Stall의 SW적 해결 방법

```
a = b + e;  
$t3  $t1  $t5  
c = b + f;  
$t5  $t1  $t4
```

Interpret

```
lw $t1, 0($t0)  
lw $t2, 4($t0)  
(Stall)  
add $t3, $t1, $t2  
sw $t3, 12($t0)  
lw $t4, 8($t0)  
(Stall)  
add $t5, $t1, $t4  
sw $t5, 16($t0)
```

13 Cycle

Modify

Compiler의  
역할이 중요

```
lw $t1, 0($t0)  
lw $t2, 4($t0)  
lw $t4, 8($t0)  
  
add $t3, $t1, $t2  
sw $t3, 12($t0)  
add $t5, $t1, $t4  
sw $t5, 16($t0)
```

11 Cycle

# Control Hazards & Stall

- Control Hazard

address	instruction		
-----			
36:		sub	\$10, \$4, \$8
40:		beq	\$1, \$3, L1
44:		and	\$12, \$2, \$5
48:		or	\$13, \$12, \$13
52:		add	\$14, \$4, \$2
56:		slt	\$15, \$6, \$7
...			
72:	L1:	lw	\$4, 50(\$7)

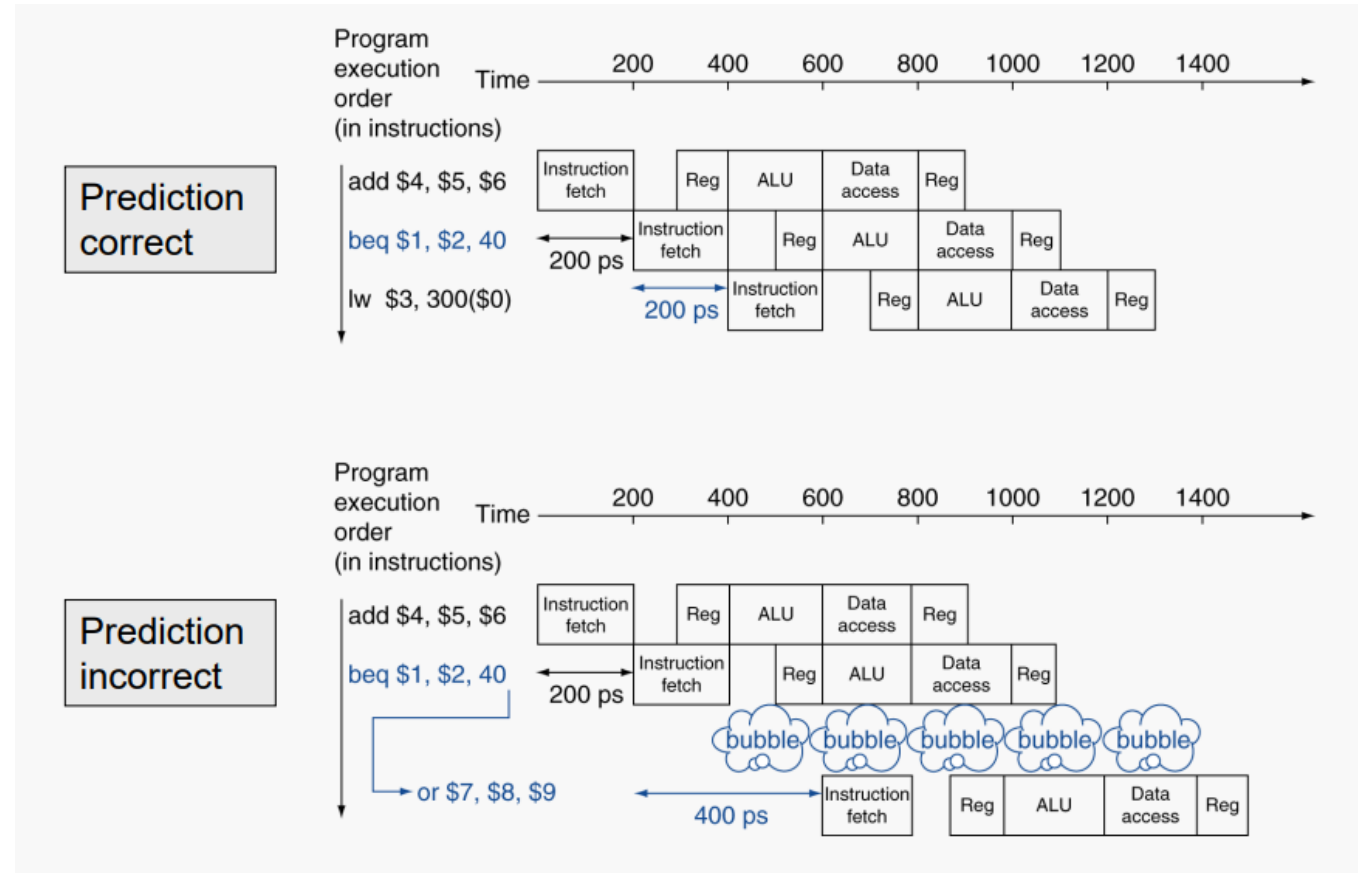
Stall 삽입

cycle	action	beq
-----		
0:	fetch sub	
1:	fetch beq	IF
2:	fetch and	ID
3:	stall	EX
4:	stall	MEM
5:	fetch and/lw	

- Branch의 조건이 맞는지 안 맞는지에 따라 Instr. Fetch가 결정되지 못 함.

# Control Hazards & Branch Prediction/Flush

- MIPS의 Static Branch Prediction ( Predict Not Taken)



# 과제

- Data Forwarding 설계

```
module DataForwarding(  
    input wire regwrite_s4, regwrite_s5,  
    input wire [4:0] wrreg_s4, wrreg_s5, rs_s3, rt_s3,  
    output reg [1:0] forward_a, forward_b  
);
```

- Pipeline Stall 설계

```
module PipelineStall(  
    input wire memread_s3,  
    input wire [4:0] rt, rt_s3, rs,  
    output reg stall_s1_s2  
);
```

```
module DataForwarding_tb;
```

```
    reg regwrite_s4_tb, regwrite_s5_tb;  
    reg [4:0] wrreg_s4_tb, wrreg_s5_tb, rs_s3_tb, rt_s3_tb;  
    wire [1:0] forward_a_tb, forward_b_tb;
```

```
module PipelineStall_tb;
```

```
// Inputs
```

```
    reg memread_s3_tb;  
    reg [4:0] rt_tb, rt_s3_tb, rs_tb;
```

```
// Output
```

```
    wire stall_s1_s2_tb;
```

# 과제

- Stage 별 연결 (CPU 생성)

```
module cpu(  
    input wire clk);  
  
parameter NMEM = 20; // number in instruction memory  
parameter IM_DATA = "im_data.txt";
```

**설날 연휴 동안 과제 수행  
(CPU 코드는 수요일 배포 예정)**