MIPS Architecture

조교 박현재

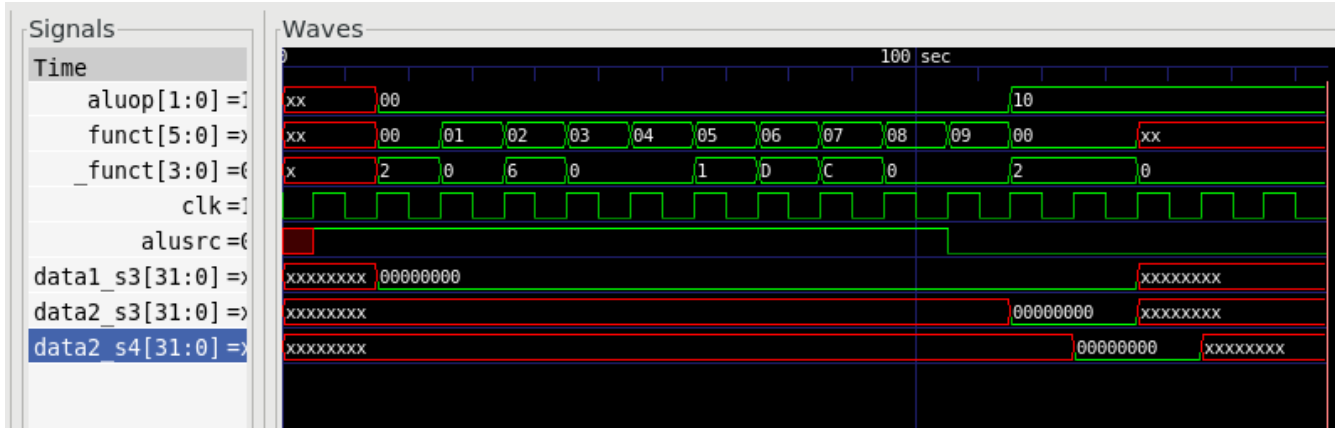
개요

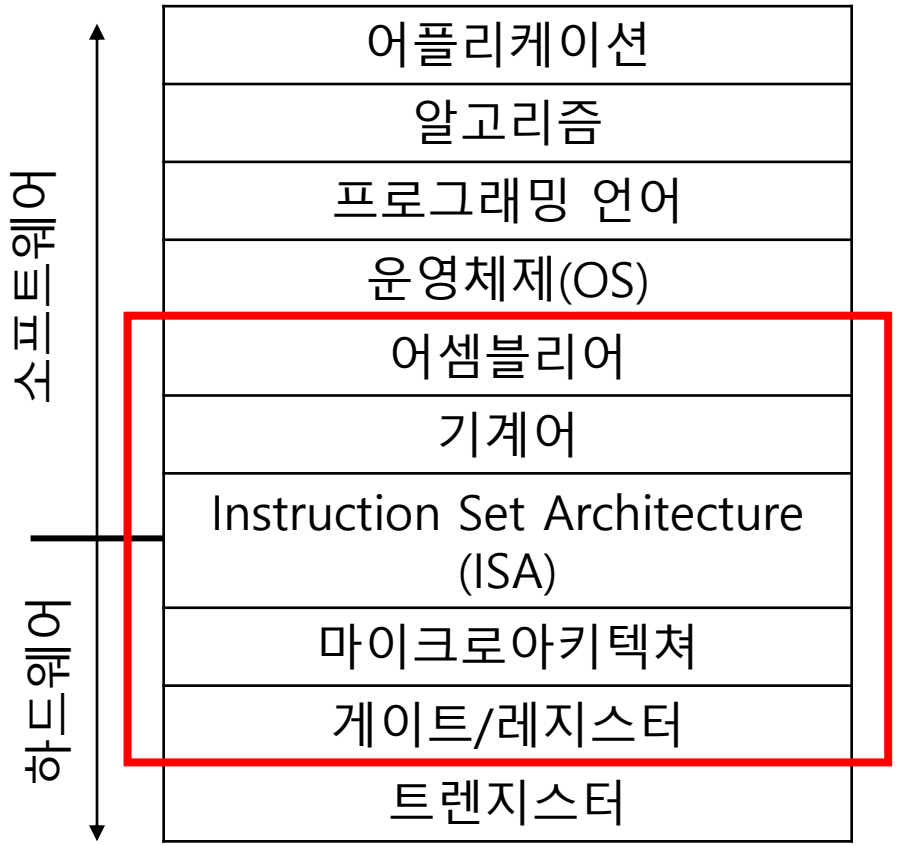
- Introduction
- MIPS
  - MIPS란
  - Pipeline
  - MIPS 5 Stage
- Instruction Fetch
  - PC(Program Counter)
- 과제

Introduction

학부 연구생 프로젝트 연구 목표

– MIPS Architecture에 대한 이해
– MIPS Architecture를 **RTL로 직접 설계**
– 설계한 MIPS를 WSL 환경에서 **직접 검증**
– 설계한 MIPS로 WSL 환경에서 **어셈블리어를 직접 구동**





컴퓨터의 추상화 계층

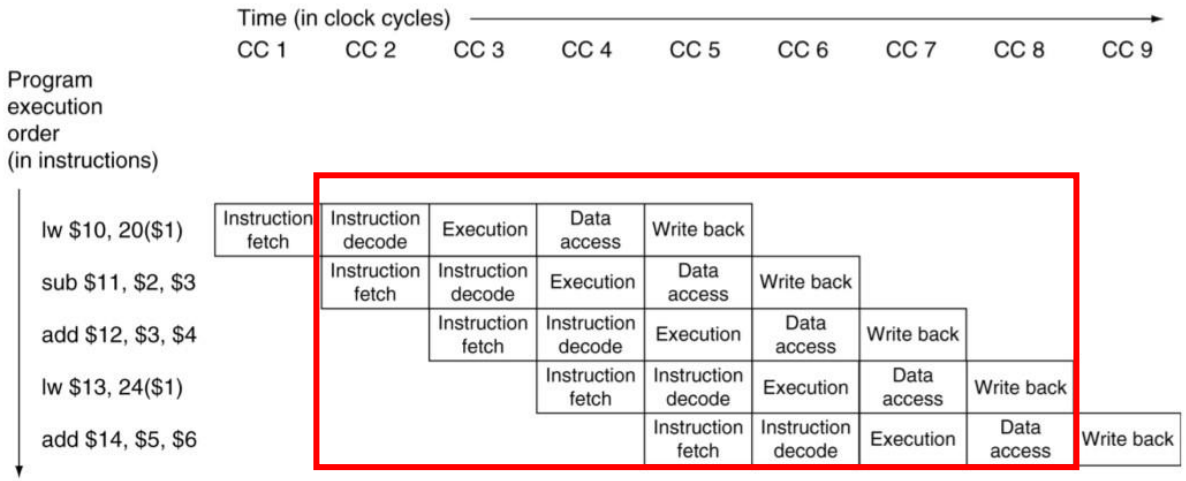
MIPS – MIPS란

- MIPS: Microprocessor without Interlocked Pipelined Stages
  - **실제 사용된 분야**

    R3000: 플레이스테이션, 플레이스테이션2
  - **디자인 원리**
    - 규칙적인 것이 간단함을 위해 좋다.
    - 많이 발생하는 사항을 빨리 처리한다.
    - 적을 수록 빠르다.
    - 좋은 설계는 좋은 절충안을 제시한다.

MIPS – Pipeline

- **Pipeline이란?**
  - 한 번에 여러 개의 명령어를 동시에 수행하여 전체 명령어 수행 시간을 감소.
    (개별 명령어 수행 시간 감소 X)
  - Pipeline으로 개선되는 최종 속도 향상



$k: Pipeline$ 단계 수, N: 실행 할 명령어 수

$$T_{No-Pipeline} = k * N, \qquad T_{Pipelined} = k + (N - 1)$$

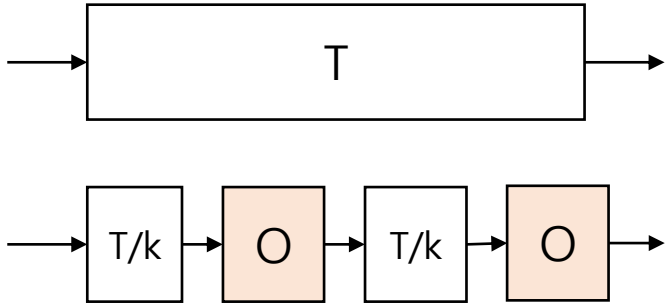
$$Speed\ Up = \frac{k * N}{k + N - 1}$$

$$Speed\ Up_{N \to \infty} \approx k$$

  - Pipeline 단계가 많을 수록 좋아 보이지만, 현실적인 한계는 20 Stage가 최대.

MIPS – Pipeline

- **왜 20 Stage가 한계인가?**



1. Stage별 균일한 작업 분배의 어려움

→ 가장 오래 걸리는 작업 기준으로 Pipelining이 이뤄짐.

2. Stage 증가에 따라 커지는 Overhead

→ Propagation Delay, Clock Skew, Setup/Hold Time등의 Overhead가 발생

$$Original\ CLK: T\ , k: Pipeline\ Stage$$

$$Speed\ Up = \frac{T}{\frac{T}{k} + O}\ , Speed\ Up_{k\to\infty} = \frac{T}{O}$$

→ 개선도는 결국 Overhead가 Dominant하게 됨.

3. Hazard 관리의 어려움

MIPS – Pipeline

- **Hazard**

매 사이클 마다 새로운 Instruction의 수행을 방해하는 요소들

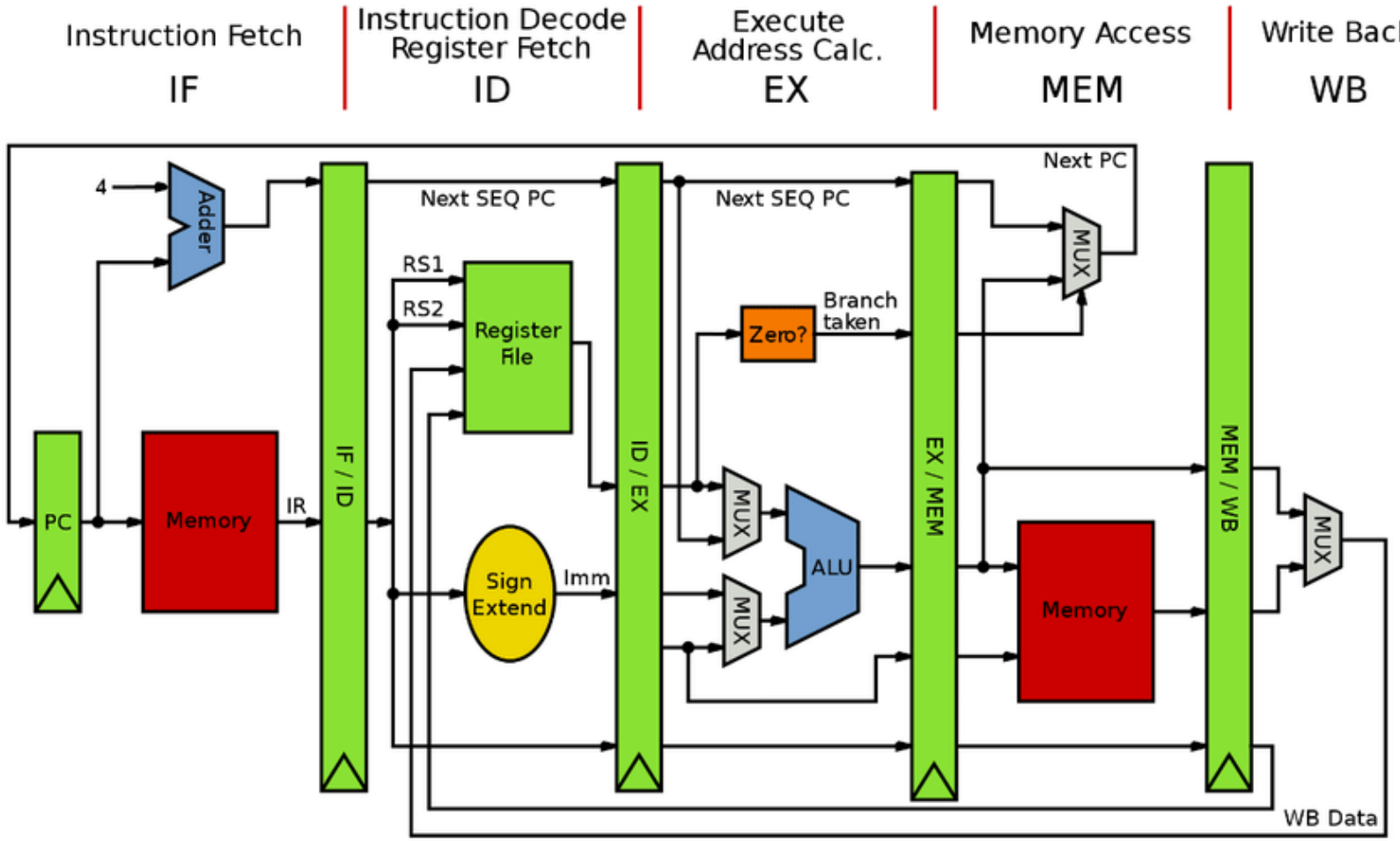
  - **Structural Hazards**
    HW 설계 이슈로 발생하는 Hazard

  - **Data Hazards**
    같은 Data를 두고 두 명령어가 Race를 할 경우 발생하는 Hazard

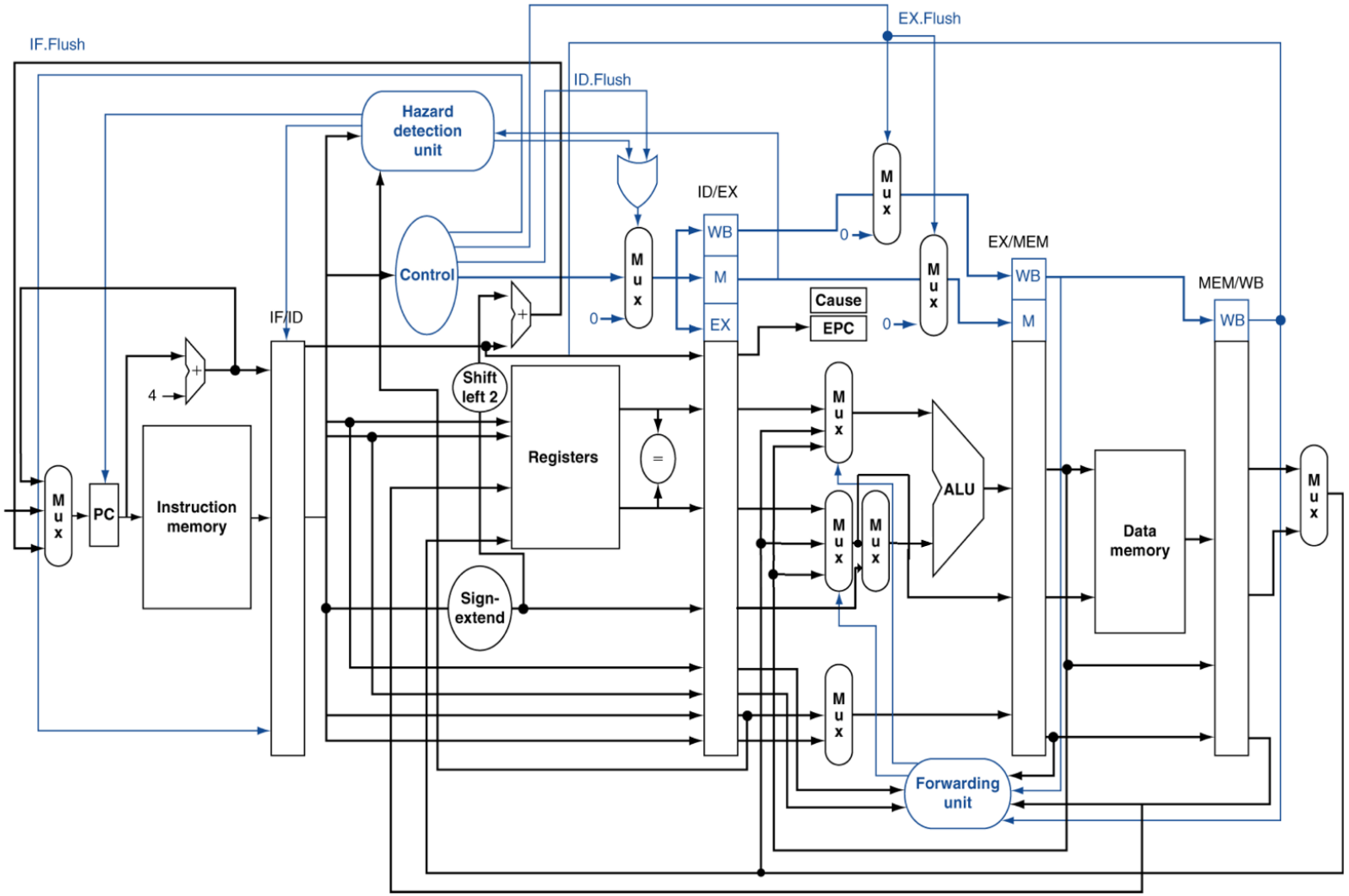
  - **Control(Branch) Hazards**
    Branch 명령어에서 발생하는 Hazard

MIPS – MIPS 5 Stage

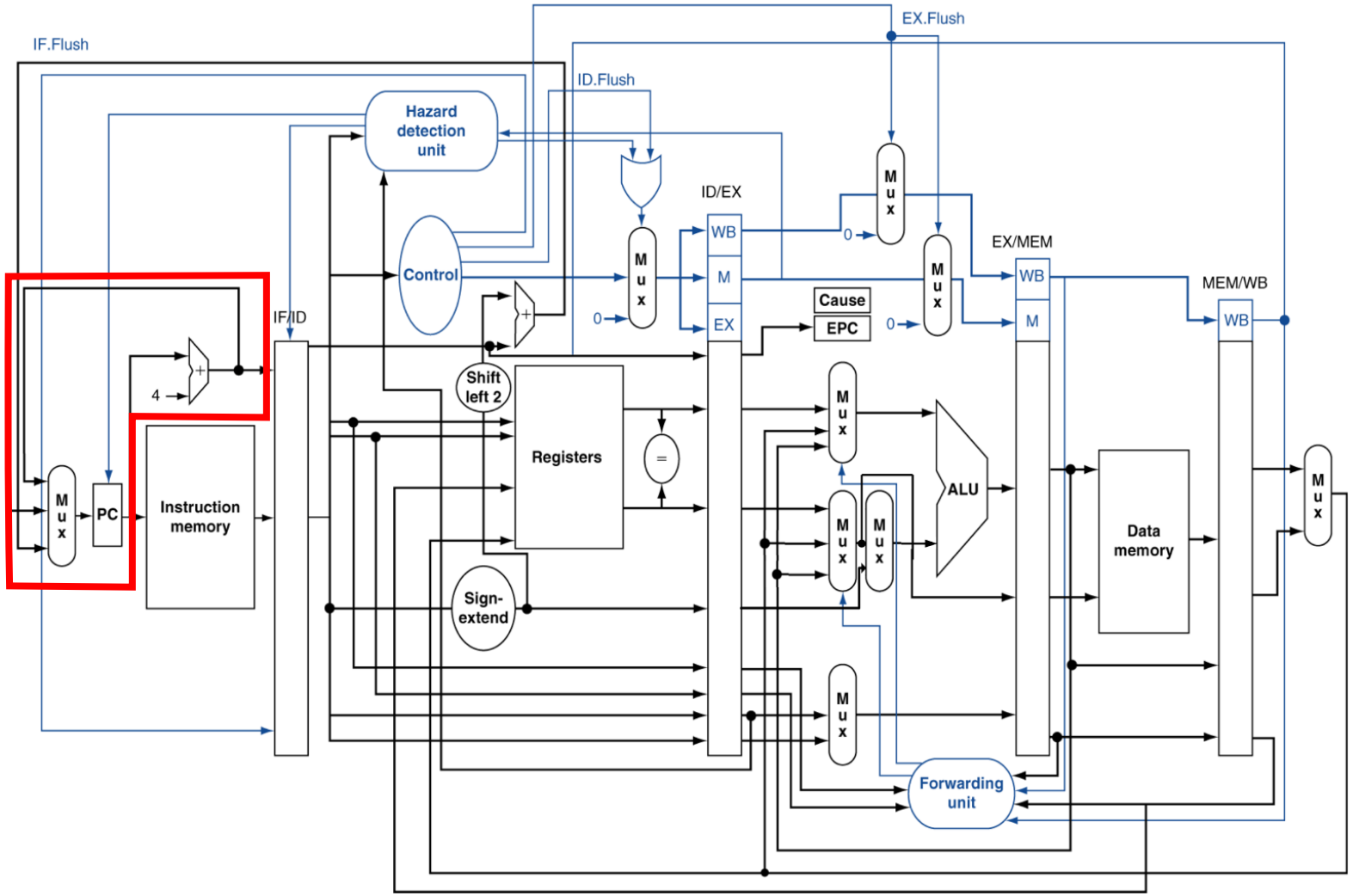
- **Instruction Fetch**
- **Instruction Decode**
- **Execution**
- **Memory Access**
- **Write Back**



MIPS – MIPS Architecture



MIPS – MIPS Instruction Fetch



System Integrated Circuit Design Lab.

MIPS – MIPS Instruction Fetch

- **PC(Program Counter): 메모리에서 가져올 명령어의 주소**

Program Counter의 Data Width는 **4 byte(32bit)**
다음 명령어의 주소 = 기존 명령어의 주소 **+ 4** 이다.
즉, PC(Next) = PC(Before)

- **PC의 초기값**

PC는 reset 될 때마다 **미리 지정된 주소**로 초기화

- **아키텍쳐 별 초기화 값**

MIPS: 0x0000_0000
x86(CISC 구조): 0xFFFF_FFF0
ARM: 0x000_0000
RISC-V: 사용자 지정

Program Counter

Address	Data
0000_0010	0xFF_DA_EA
0000_000C	0x3D_AB_BB
0000_0008	0x2B_AA_08
0000_0004	0xF2_F1_AC_07
0000_0000	0xAB_CD_EF_78

width= 4 bytes

MIPS – MIPS Instruction Fetch

- **PC(Program Counter)의 다음 값으로 올 수 있는 목록**

1. 이전 명령어가 branch였을 경우

➔ **branch 주소**

2. 이전 명령어가 jump였을 경우

➔ **jump 주소**

3. Stall이 걸린 경우

➔ **기존 PC**

4. Default

➔ **PC + 4**

과제

- Program Counter 설계

```
module program_counter(
    input clk,
    input rst_n,
    input stall_s1_s2, pcsrc, jump_s4,
    input [31:0] jaddr_s4, baddr_s4,
    output reg [31:0] pc_out
    );
```

- Pipeline Stage 동작을 위한 파라미터화 된 D F/F 설계

```
module regr (
    input clk,
    input clear,
    input hold,
    input wire          in,
    output reg          out);

    parameter
```