컴퓨터 구조
(LD001600)

김준철
greensday@sungshin.ac.kr

11주차 강의

주차	강의 목차
1	과목소개 / 시스템 개론
2	컴퓨터 추상화 및 관련기술
3	컴퓨터 구조 정리, ISA, RISC, MIPS
4	컴퓨터 명령어 1_ISA,하드웨어연산, 명령어의 컴퓨터 내부표현 등
5	컴퓨터 명령어 2_명령어의 컴퓨터 내부표현,논리 명령어 등
6	컴퓨터 명령어 3_판단 명령어,프로시저
7	컴퓨터 명령어 4_MIPS 주소지정
8	중간고사
9	컴퓨터연산, 프로세서 1_기초, 설계 관례
10	프로세서 2_데이터패스, 단순 구현
11	프로세서 3_파이프라이닝
12	프로세서 4_데이터해저드1
13	프로세서 5_데이터해저드2, 제어해저드
14	메모리 계층 구조 1_메모리,캐시(cache)
15	기말고사

Ch.04 프로세서 3 The Processor 3

4.3 파이프라이닝에 대한 개관

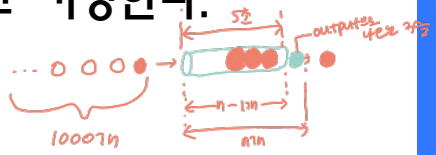
(일부는 12,13주차의 다른 chapter에서 배움)

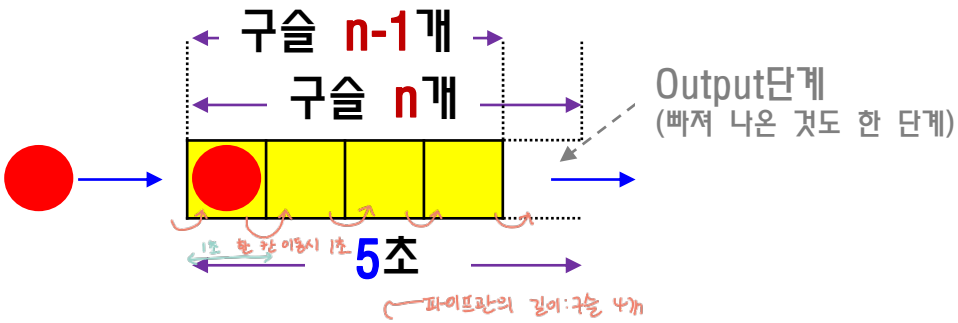
4.4 파이프라인 데이터패스 및 제어

Pipelining Analogy

어떤 일을 할 수가 출출을 이어서 하면 따로따로 하나씩 하는 것보다 훨씬 이득

- [예제] 구슬 1000개를 아래 파이프관을 통과 시켜서 전달하려고 한다.
  파이프 라인의 길이에는 구슬 n-1개가 채워질 수 있는 길이이고,
  구슬이 파이프라인을 통과해서 밖으로 나오는데 5초가 걸린다고 가정한다.





1. 구슬이 1000개, n=5 일 때,

   (1)구슬 1개가 파이프라인을 통과해야 다음 구슬이 파이프 라인으로 들어간다면 → single cycle
   구슬 1000개가 모두 파이프라인을 통과해서 나올 때 까지 걸리는 시간은?

   ①번 구슬의 input~output : 5초
   ⓝ번 구슬의 input ~ output : 5초 　　1000×5초 = 5000초

   (2)구슬 1개가 파이프라인을 들어가면 이어서 바로 다음 구슬이 줄지어서 들어 갈 수 있다면,
   구슬 1000개가 모두 파이프라인을 통과해서 나올 때 까지 걸리는 시간은?

   → pinelining

구슬이 입구쪽으로 들어가서 이동하면서 시간이 지나 4개의 구슬이 다 채워짐. 그리고 나서 구슬 output으로 나옴.
구슬이 파이프관에 다 채워지는 시간(4s)
→ pipelineing fillup time.
└ pipelineing에서는 이 fillup time 지나면 그후부터는 단위시간마다 결과가 출출출 하나씩 나옴

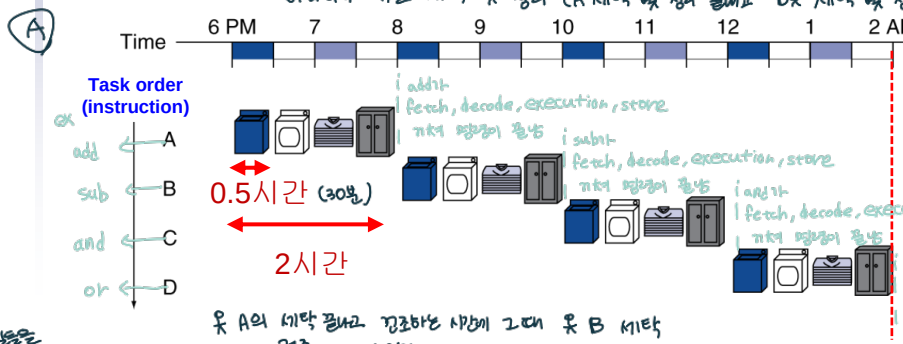
①번 구슬의 input~output : 5초
②번 구슬의 input~output : ①구슬 output으로 나온 1초 후
③번 구슬의 input~output : ②구슬 output으로 나온 1초 후
⋮
1000번 구슬의 input~output : 999구슬 output으로 나온 1초 후

5초 + 999×1초
1번구슬의 input~output
= 1004초

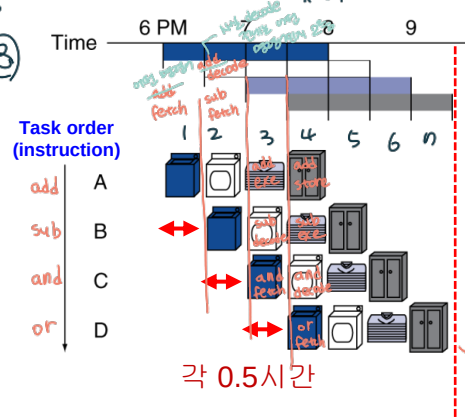
Pipelining Analogy

- Pipelining: ILP(Instruction Level Parallelism)의 하나
  - Multiple instruction을 overlapping하여 실행
    → performance를 개선 (throughput 개선)

Ex. 4가지 종류의 옷의 세탁 및 정리 프로세스의 예 세탁 → 건조 → 옷정리 → 옷장정리 (각 단계는 30분 소요)



4종류의 세탁을 끝내는데 총 8시간소요



4종류의 세탁을 끝내는데 총 3.5시간소요

speedup = $\frac{\text{명령어 수행시간}_{\text{pipeline 아님}}}{\text{명령어 수행시간}_{\text{pipeline}}}$

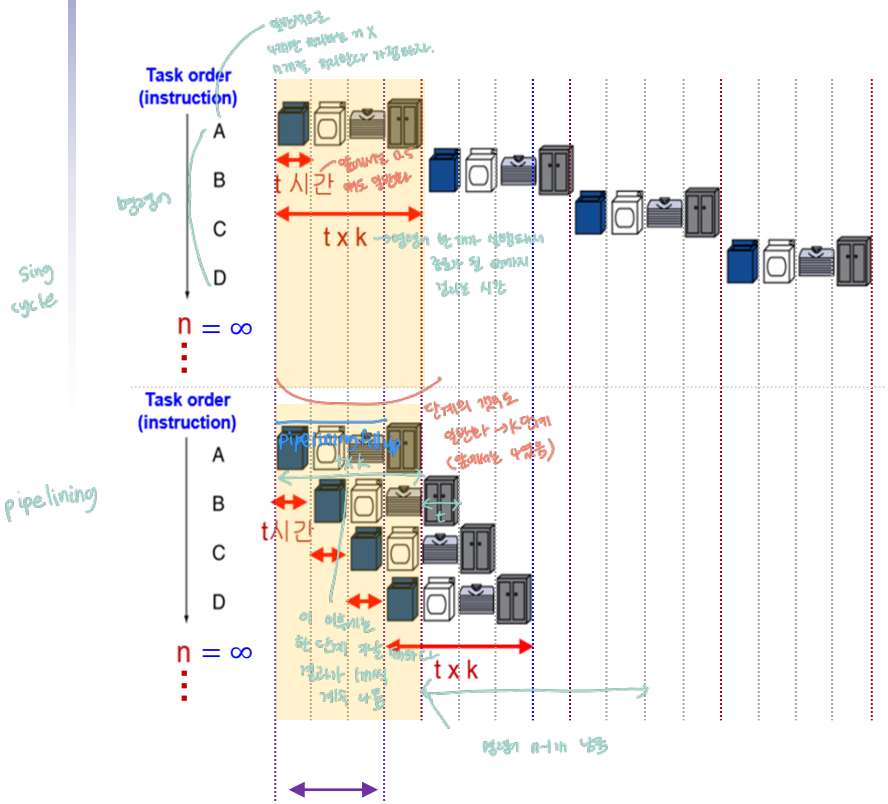
* 4단계(stage)의 연속실행 시 속도증가 :

Speedup = {(0.5x4)x4} ÷ {0.5x4+1.5}

= 8 ÷ 3.5 = 2.3

Pipelining Analogy

- n : instruction 수
- k : pipeline stage 수
- t : stage 하나를 수행하는데 걸리는 시간



pipeline fill-up time

= t x (k-1)

- Task order가 매우 클 때, (A,B,C,D,E,F,…) speedup을 수식으로 표현

$$\text{Speedup} = \frac{\text{명령어 수행시간}_{\text{pipeline 아님}}}{\text{명령어 수행시간}_{\text{pipeline}}}$$

$$= \frac{t \times k \times n}{t \times (k-1) + t \times n} = \frac{t \times k}{t} = k$$

$n = \infty$

$$\text{Speedup} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Time between instructions}_{\text{pipelined}}}$$

= number of states (단계 수)

MIPS Pipeline

- Five stages, one step per stage
  1. **IF** : Instruction fetch from memory
  2. **ID** : Instruction decode & register read
  3. **EX** : Execute operation or calculate address
  4. **MEM** : Access memory operand
  5. **WB** : Write result back to register

single cycle이랑 같음.

lw

data Memory에 있는 data를 cpu 안에 있는 register로 가지고 돌아가는 것.

cpu

data Memory

data Memory쪽으로 접근. 해당 주소에 있는 data가 register 쪽으로 가져 저장함.

single cycle 할 때는 안 봄

사실 메모리 엑세스를 했는데 메모리 엑세스 단계라고 이야기하지 않는 것

메모리 결과물을 register쪽으로 다시 가져가서 저장하는 것.

R-type 명령에 필요한가?

ex) add

data Memory에 접근 안 함.

R-type 명령에 필요 없으나,
단계를 없앨 수는 없음 → pipeline 구조에서는 자기가 하지 않는 단계 (ex: MEM)도 하진 않지만 그 단계 지나가야함.

register에서 가져오는 값

add

Data Memory

register

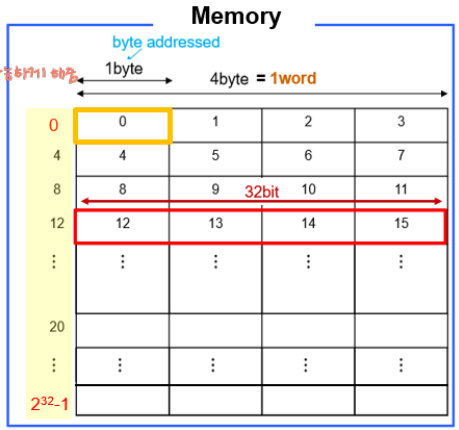
throughput 개선. (single cycle 대비) → 성능
but 모든 컴퓨터가 pipelining 지원하지는 못함

Pipelining and ISA Design

컴퓨터 구조 크게 CISK와 RISK 2가지로 나눌 수 있음

일반적인 MIPS → pipe line 구조를 재현하기 어려움. / ISA 굉장히 복잡 — 명령어마다 길이 정수 다름.

- Pipeline을 위해서는 ISA가 단순 해야함
  - 그래서 요즘 대세 RISK (MIPS도 리스크 아키텍)
  - MIPS ISA의 모든 Instruction은 32bit (4word)
    - ISA 굉장히 단순.
      종류 3가지. 모든 명령어 32bit
- Pipelining을 위한 MIPS ISA 디자인
  - 모든 instructions은 32-bits — 일찍 인식하면 미리 데이터를 read 한 것.
    - 또 decode 과정에서 prefetch도 가능. (ISA가 뒤바뀌지 않았던 것) → ISA 뒤바뀜 안되면 decode 한 후에 prefetch 하면.. 시간이 더 걸림
    - Fetch / decode : one cycle 안에 쉽게 가능 — 여러가지 일을
  - instruction formats이 단순하고 간단
    - 1 step으로 register들을 decode / read(prefetch)
  - Load/store addressing도 단순하고 간단
    - mips는 주소계산이 간단 (base address + offset 이 한가지 뿐)
  - Alignment of memory operands
    - → alignment restriction (메모리에 접근할 때 변수(word) 단위로 접근하도록) → 단일한 접근 → 1 cycle 안에 Memory Access 가능하게 하기
    - Memory access는 단지 one cycle 소요



MIPS Pipelined Datapath

MIPS의 pipeline 특징)
각 stage 별로 소요되는 시간이 동일(t)

나중에라면 얘기하지
사실 pipeline 하고
앞에서 본 MIPS의 single cycle 하고
→이 그림처럼 잘라지기 쉽지않음
그렇게 그린 것
→pipeline 지원하지 않는 하드웨어 각 단계별로 자르려면 쉽지 X
사실 MIPS가 single cycle로 돌아가는 하드웨어
아니라 pipeline으로 돌아가는 하드웨어
→쉽게 잘라서 단계단계로 구분할 수 있는 것

하드웨어는
어쨌거나 명령어마다 길이 달라짐.

그림 : singlecycle 구조의 하드. pipeline에서도 이 하드 동일하게 사용

하드웨어에서 명령어 1개만 돌아감.

사실 소요시간은 다 동일하지
연산량이 있는데 동일하게 맞추어야 하지 X

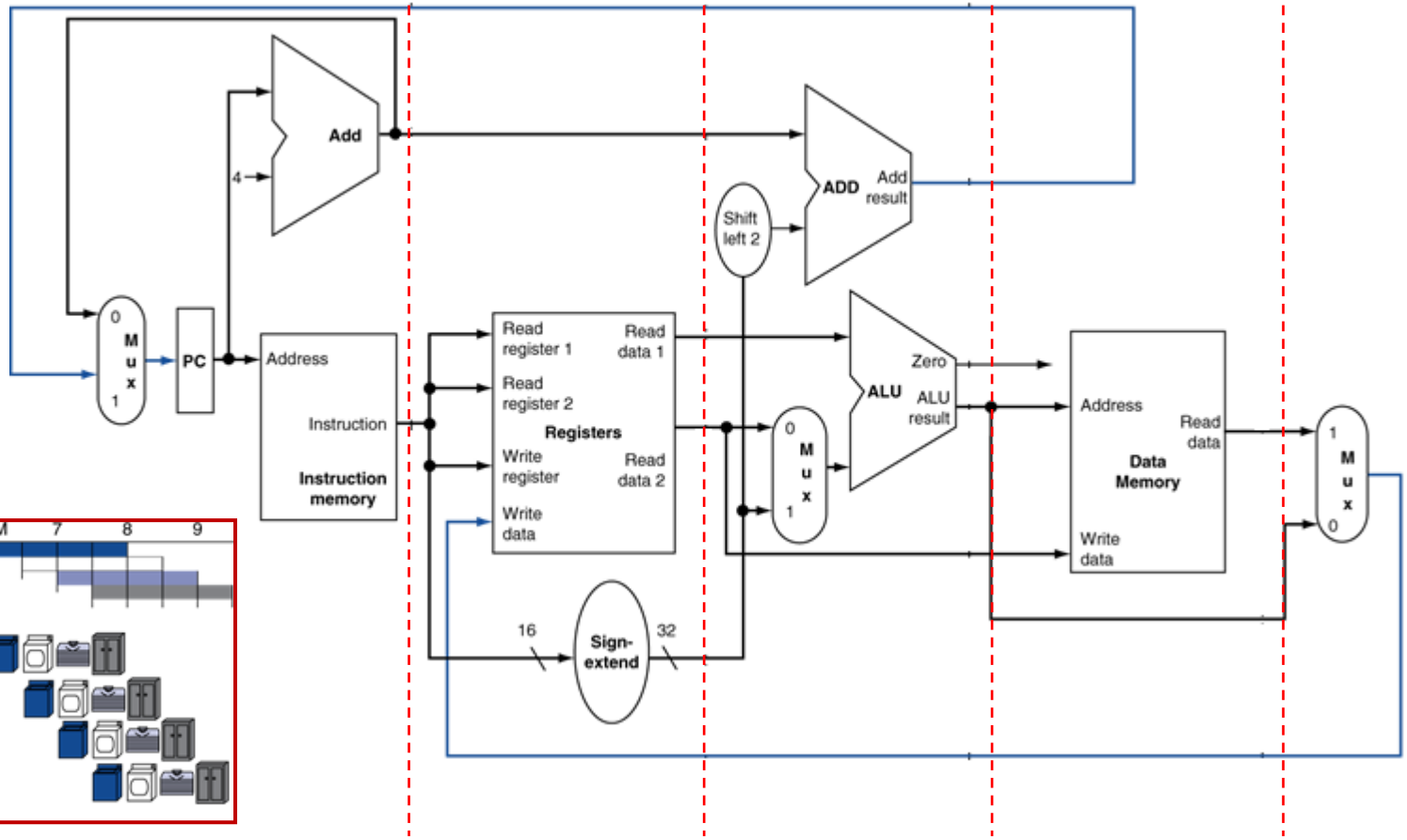
ALU 필요

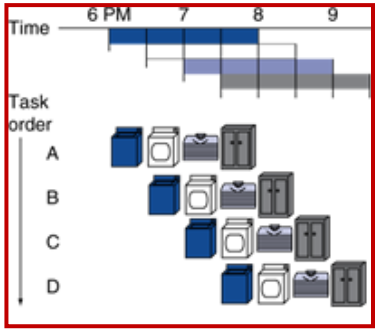
Memory 있어야

IF: instruction fetch	ID: instruction decode/ register file read	EX: Execute/ address calculation	MEM: Memory access	WB: Write back

모든 stage 가 동일시간

t t t t t





Pipeline Performance

- stages가 아래 시간과 같다고 가정

  register read or write : 100ps
  기타 stages: 200ps

- single-cycle datapath

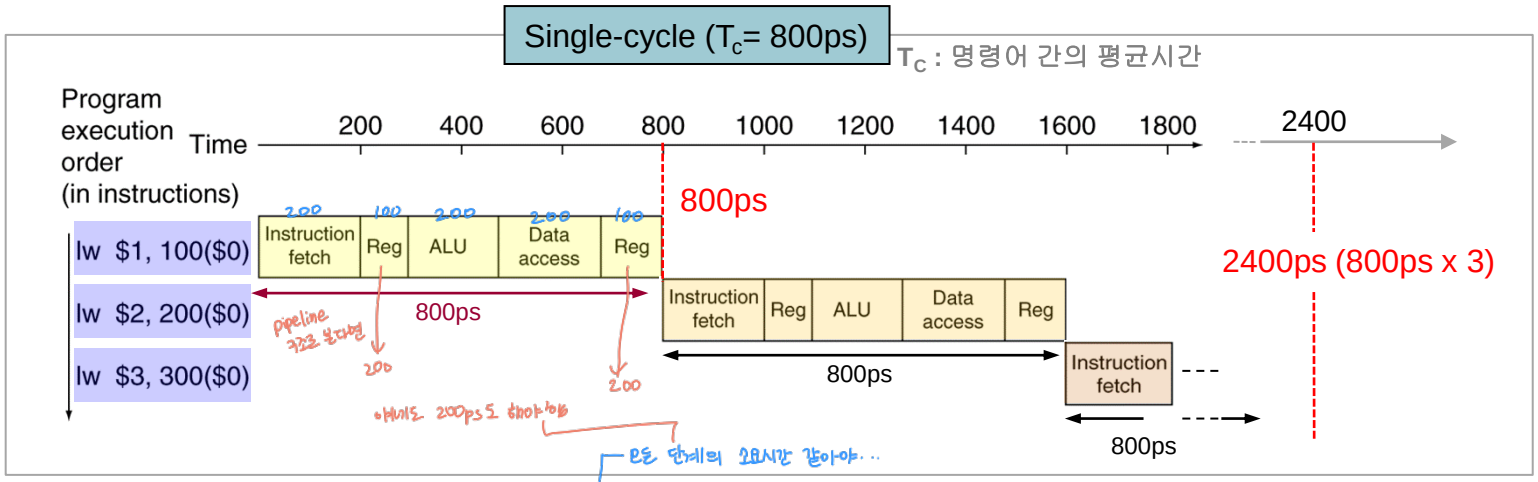
Instruction	Instruction fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100ps	200ps	200ps	100ps	800ps
sw	200ps	100ps	200ps	200ps		700ps
R-format	200ps	100ps	200ps		100ps	600ps
beq	200ps	100ps	200ps			500ps

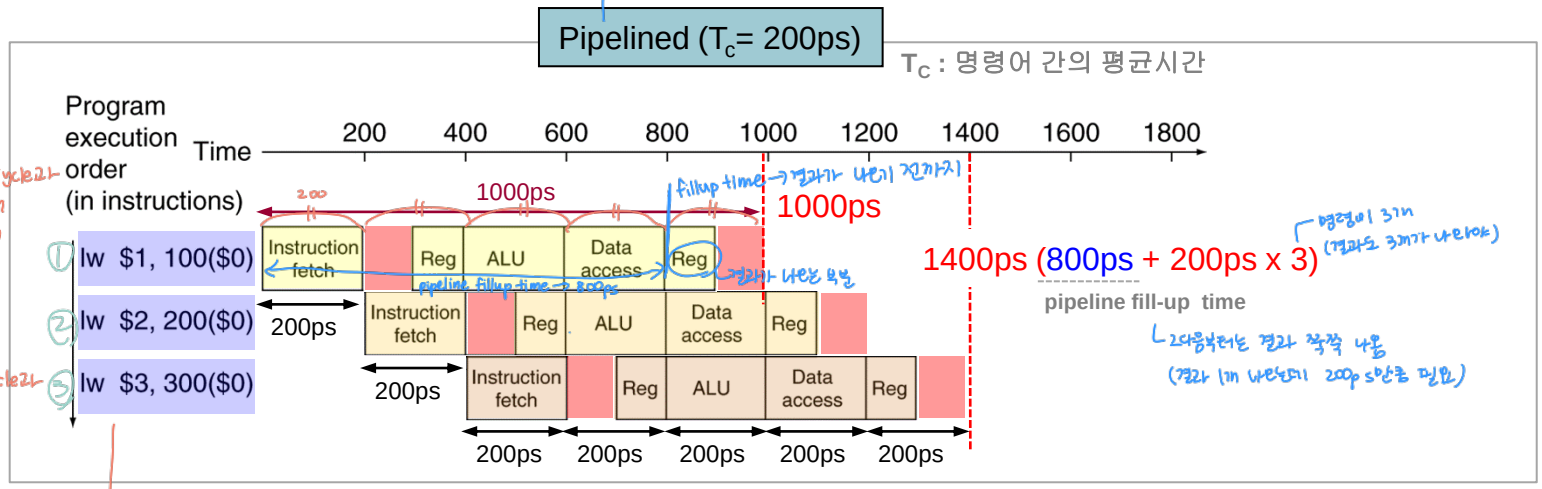
* Longest delay determines clock period
* critical path (longest delay path): **load instruction**

- pipelined datapath와 single-cycle datapath를 비교하면 어떤 결과를 얻을 수 있을까?

Pipeline Performance

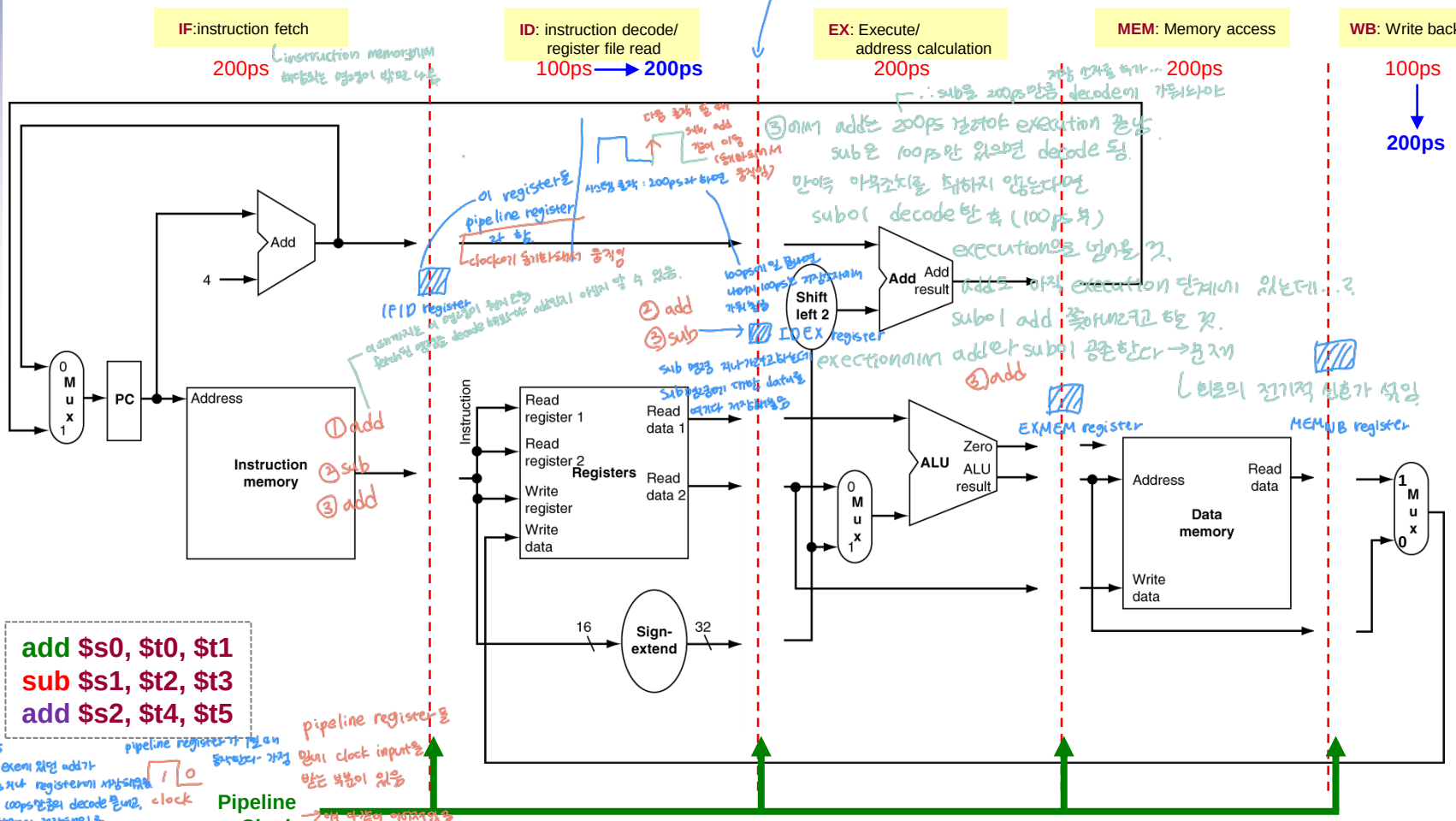
- pipelined datapath와 single-cycle datapath의 비교





Pipeline registers

- previous cycle의 정보를 **저장** 필요가 있음
  → stage 간에 (**registers**) 필요

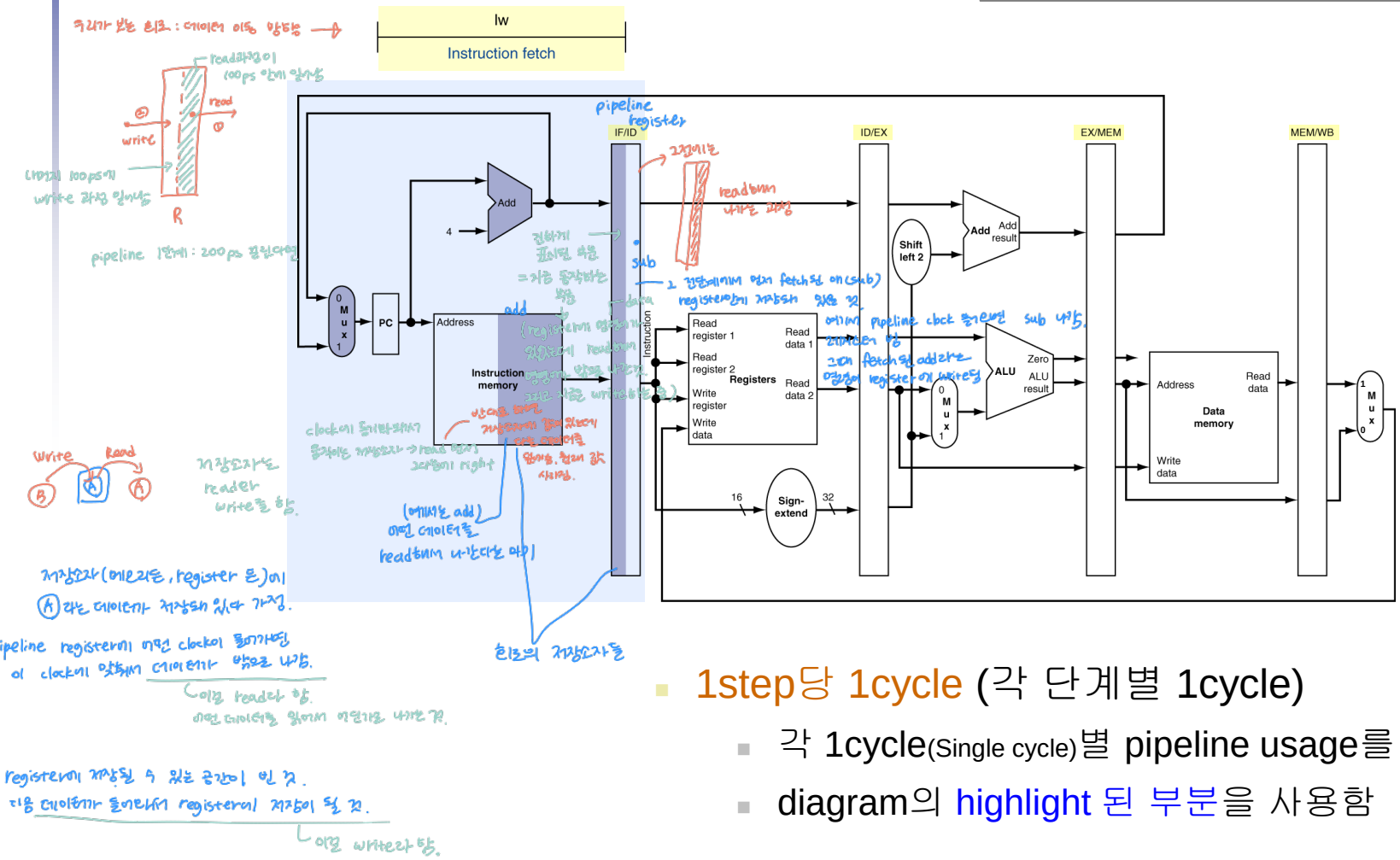


add $s0, $t0, $t1
sub $s1, $t2, $t3
add $s2, $t4, $t5

Pipeline Operation

- 1 clock-cycle pipeline diagram

교재의 diagram 보는 방법



- 1step당 1cycle (각 단계별 1cycle)
  - 각 1cycle(Single cycle)별 pipeline usage를 보여줌
  - diagram의 highlight 된 부분을 사용함

그림 한 개에 multi cycle을 다 표현
일종의 회로: 촬영하면 꼭 snapshot한 형태(or 순간에 표시) / multi-cycle pipeline diagram은 여러개의 명령이 시간이 흐르면서 어느 단계에 있게 되는지 다 표시

Diagram 1 Multi-Cycle Pipeline Diagram

(명령 수행과정을 시간단위로 전체적으로 묘사할거)

- Form showing resource usage

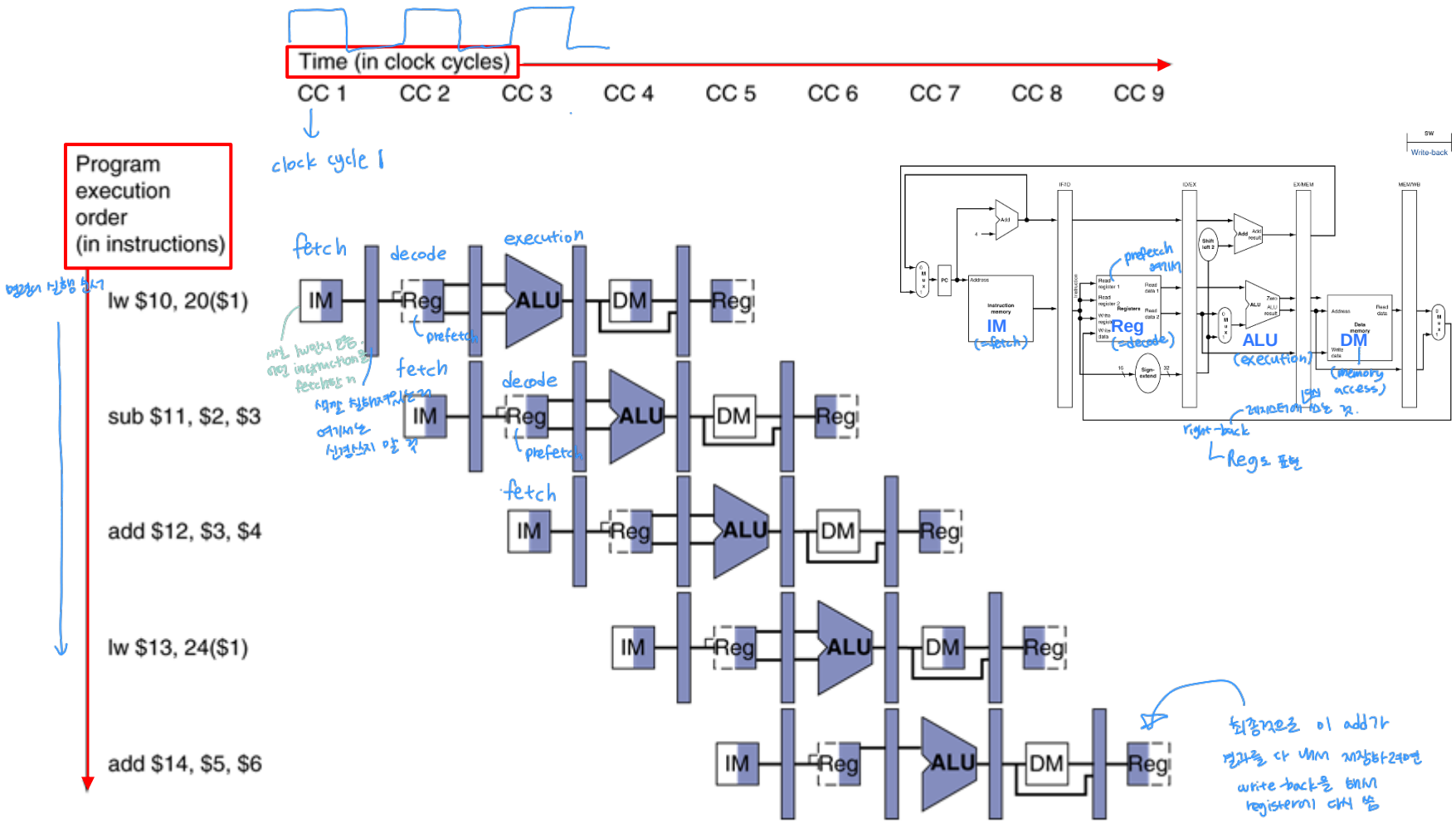


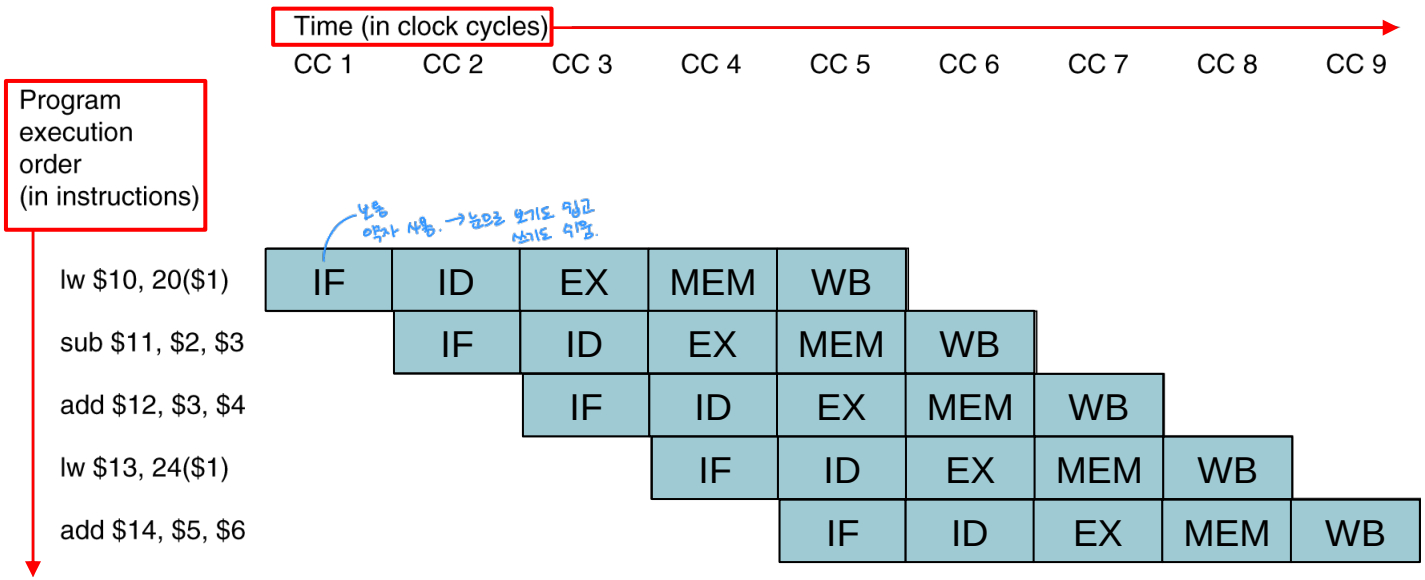
Diagram 2 Multi-Cycle Pipeline Diagram

여러 한 화면으로 각 clock에 대한 명령어들이 어느 위치에 있는지 다 확인가능

Traditional form

그림 형태나 글 형태나 이것만 다른 거

앞에서 본거랑 좀 차근의 form

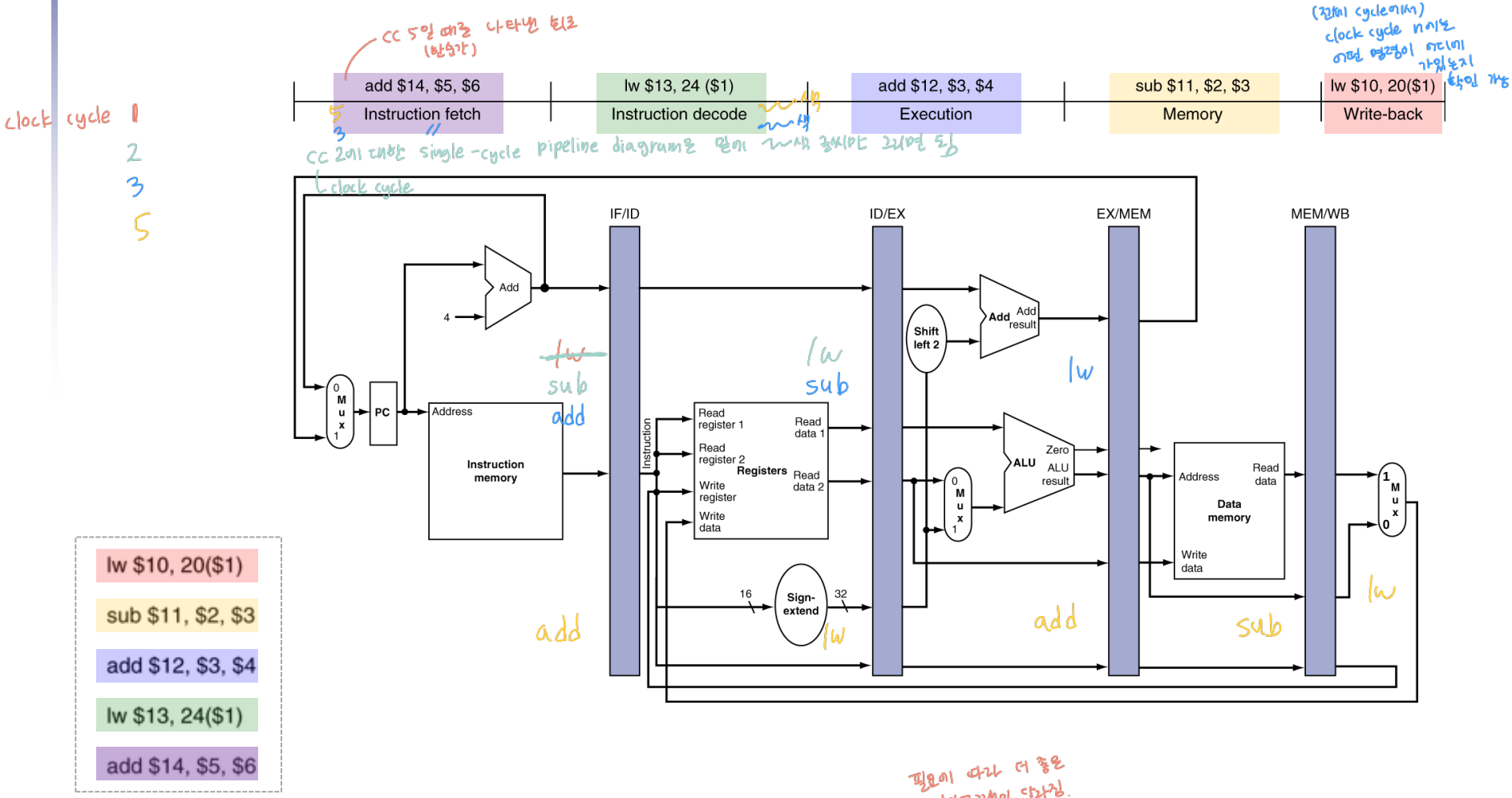


single-cycle 회로인 거 아님!

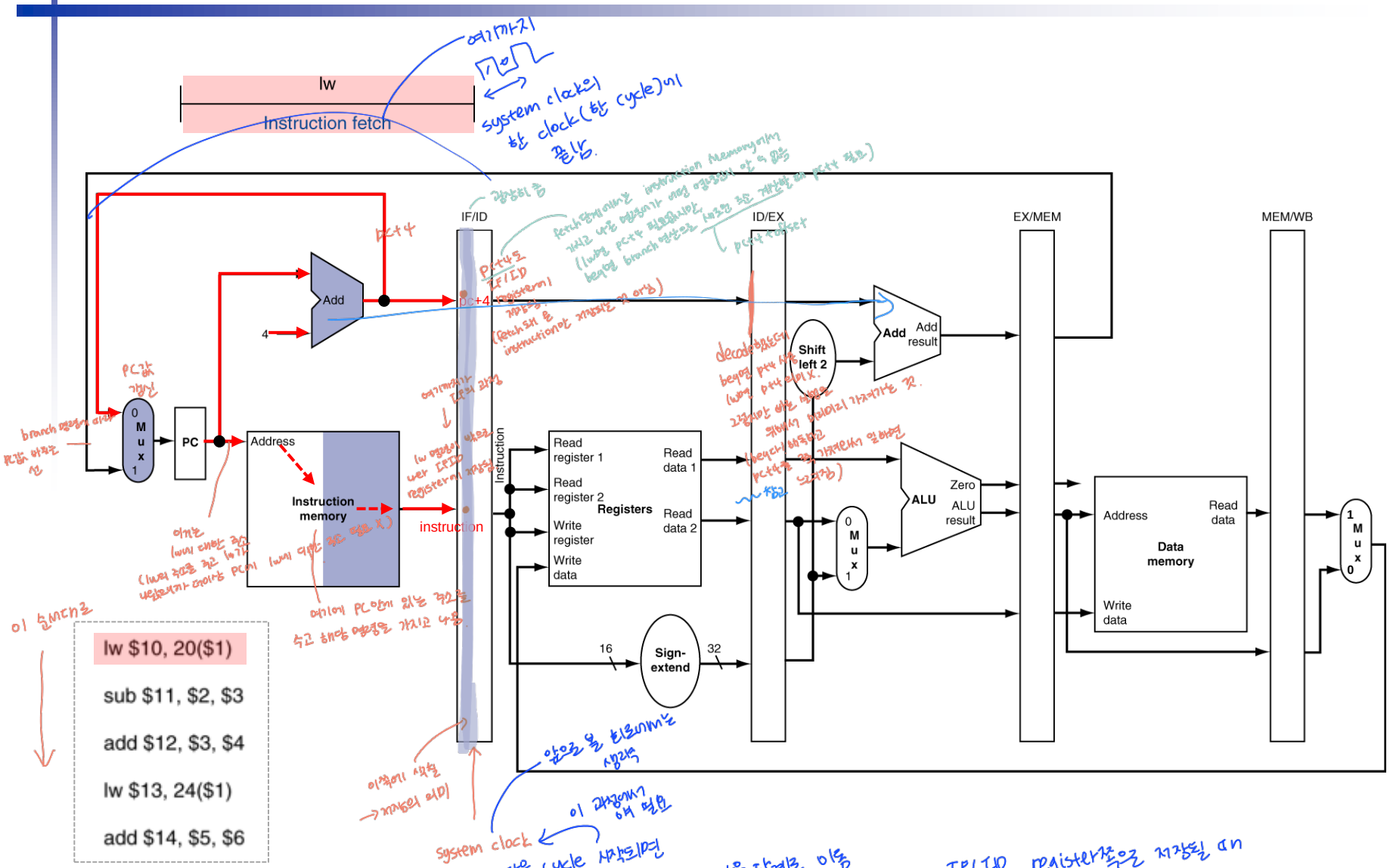
Diagram 3 Single-Cycle Pipeline Diagram

특정 1cycle의 모습
(clock cycle 5)

- State of pipeline in a given cycle

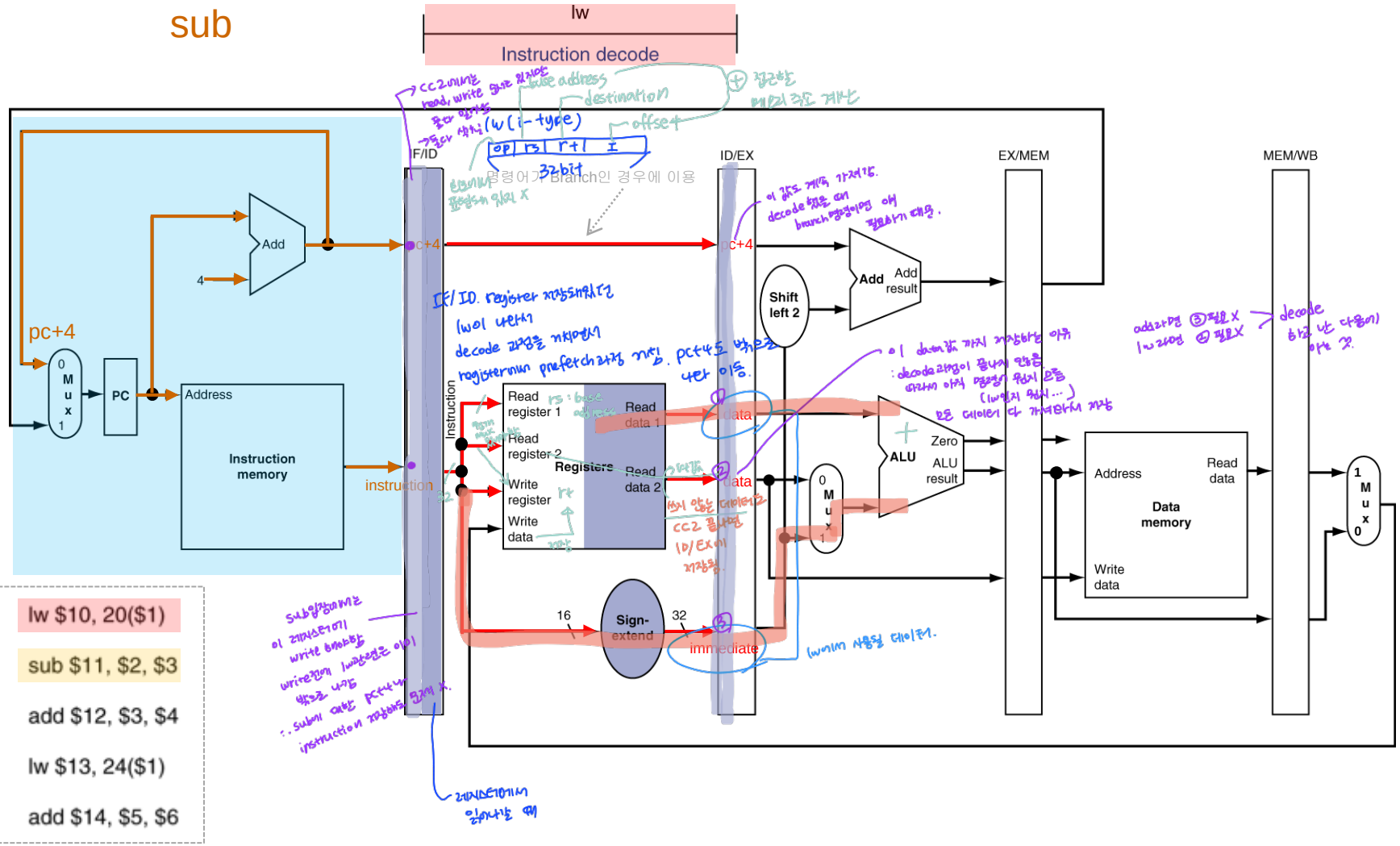


IF (instruction fetch) Single-Cycle Pipeline Diagram



ID (instruction decode)

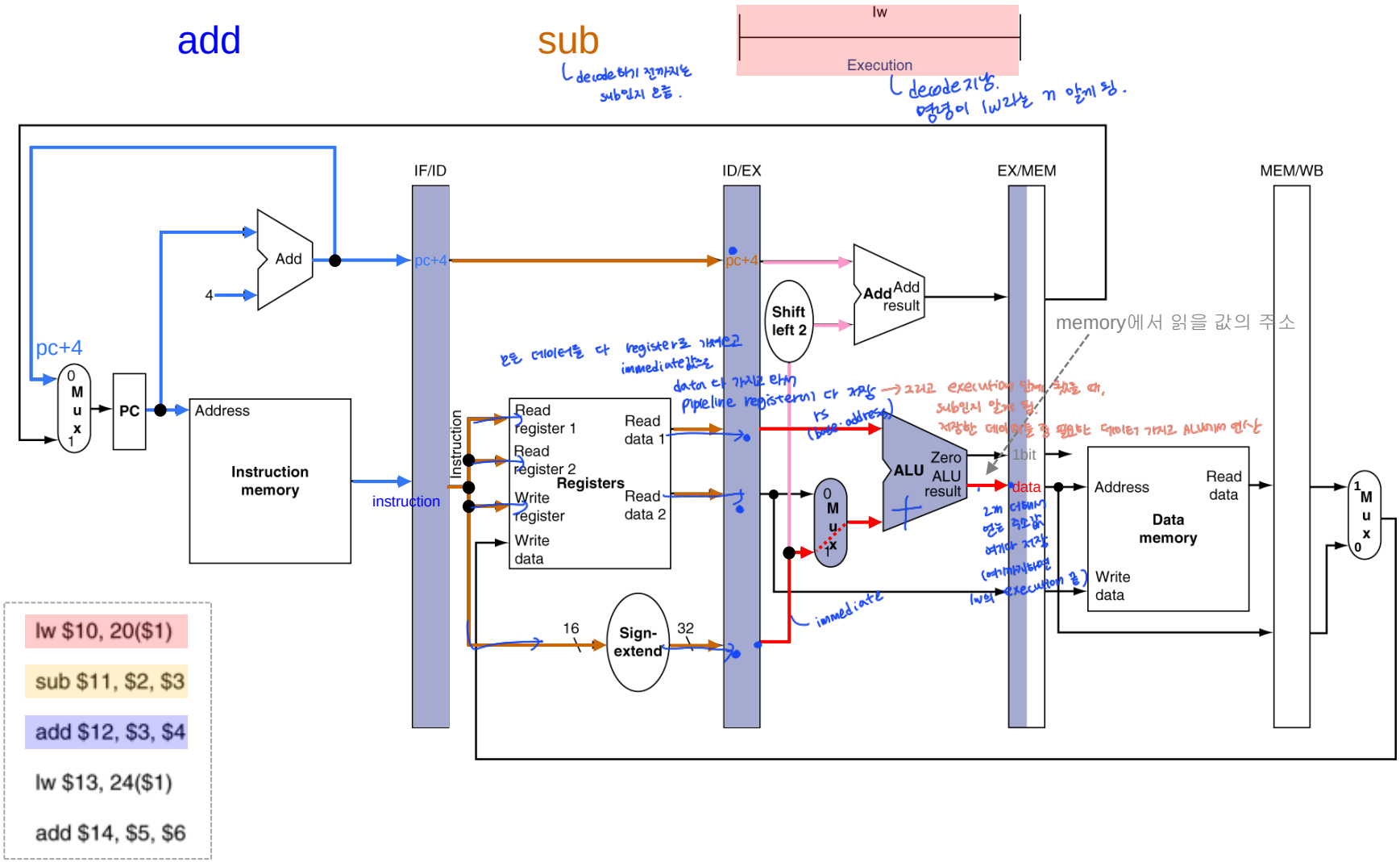
Single-Cycle Pipeline Diagram



EX(execution) for Load

Single-Cycle Pipeline Diagram

instrucion이 lw 임을 명확히 인지하고 난 후의 단계

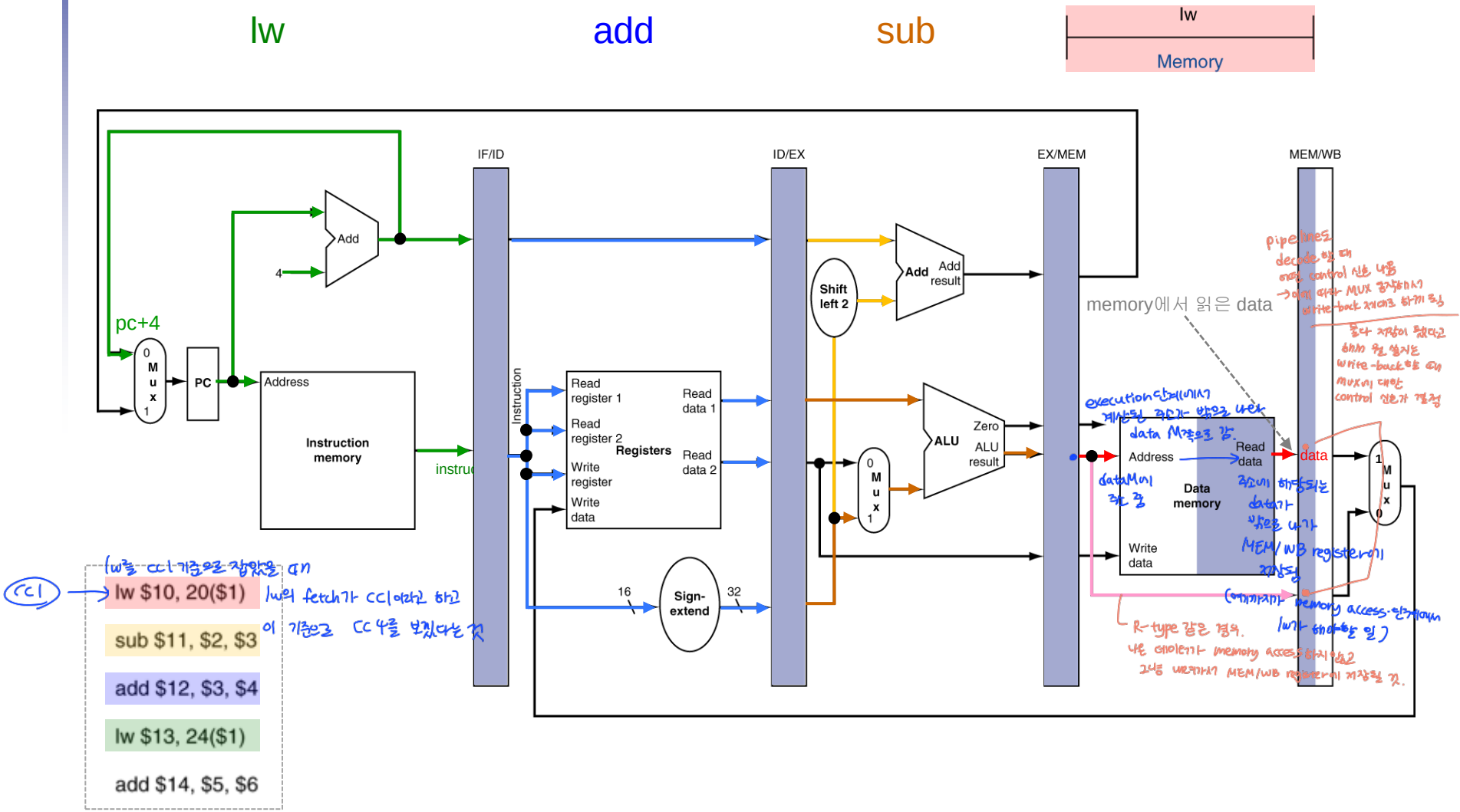


MEM(memory) for Load

CC 4

(lw → memory access

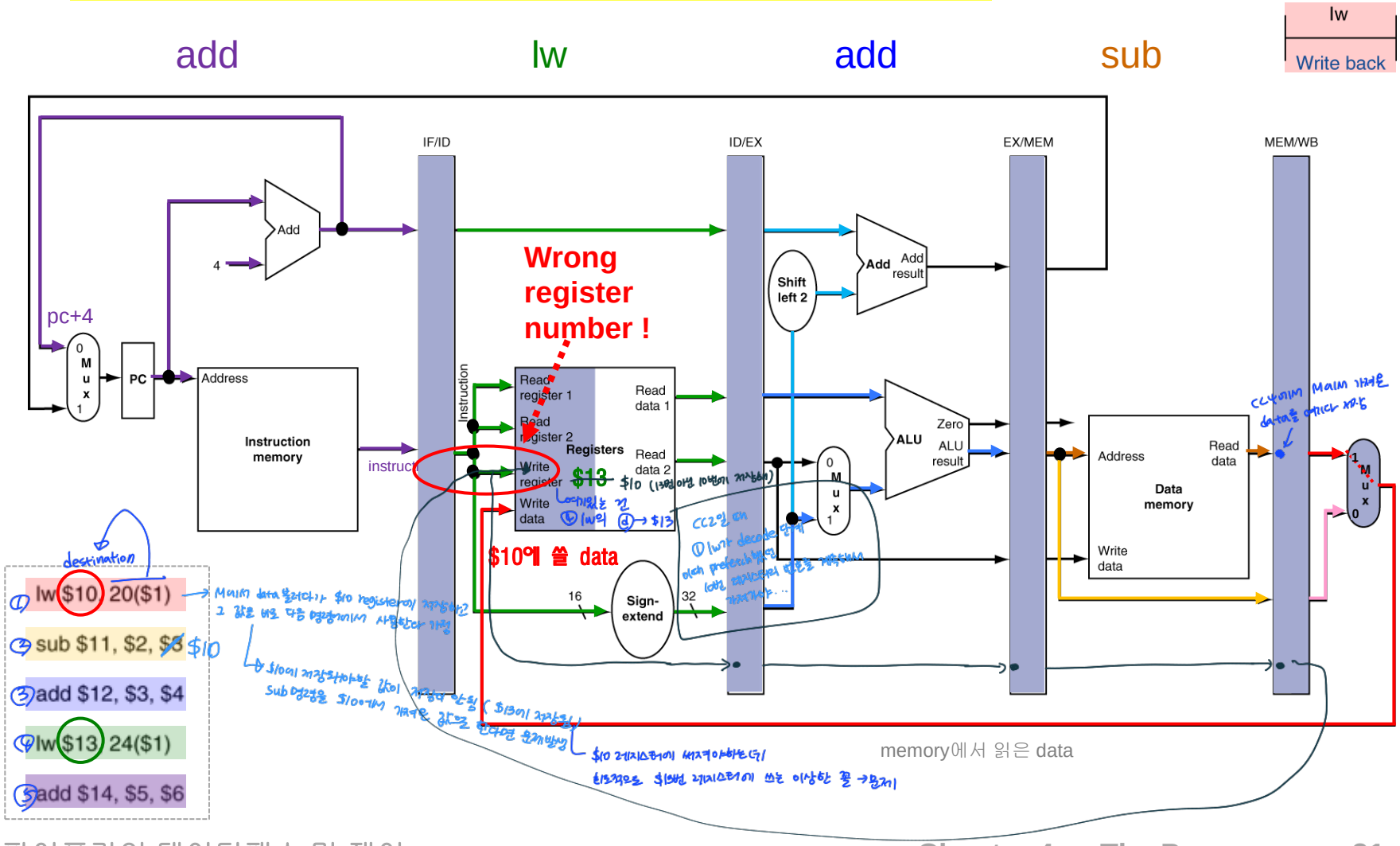
Single-Cycle Pipeline Diagram



WB(write back) for Load

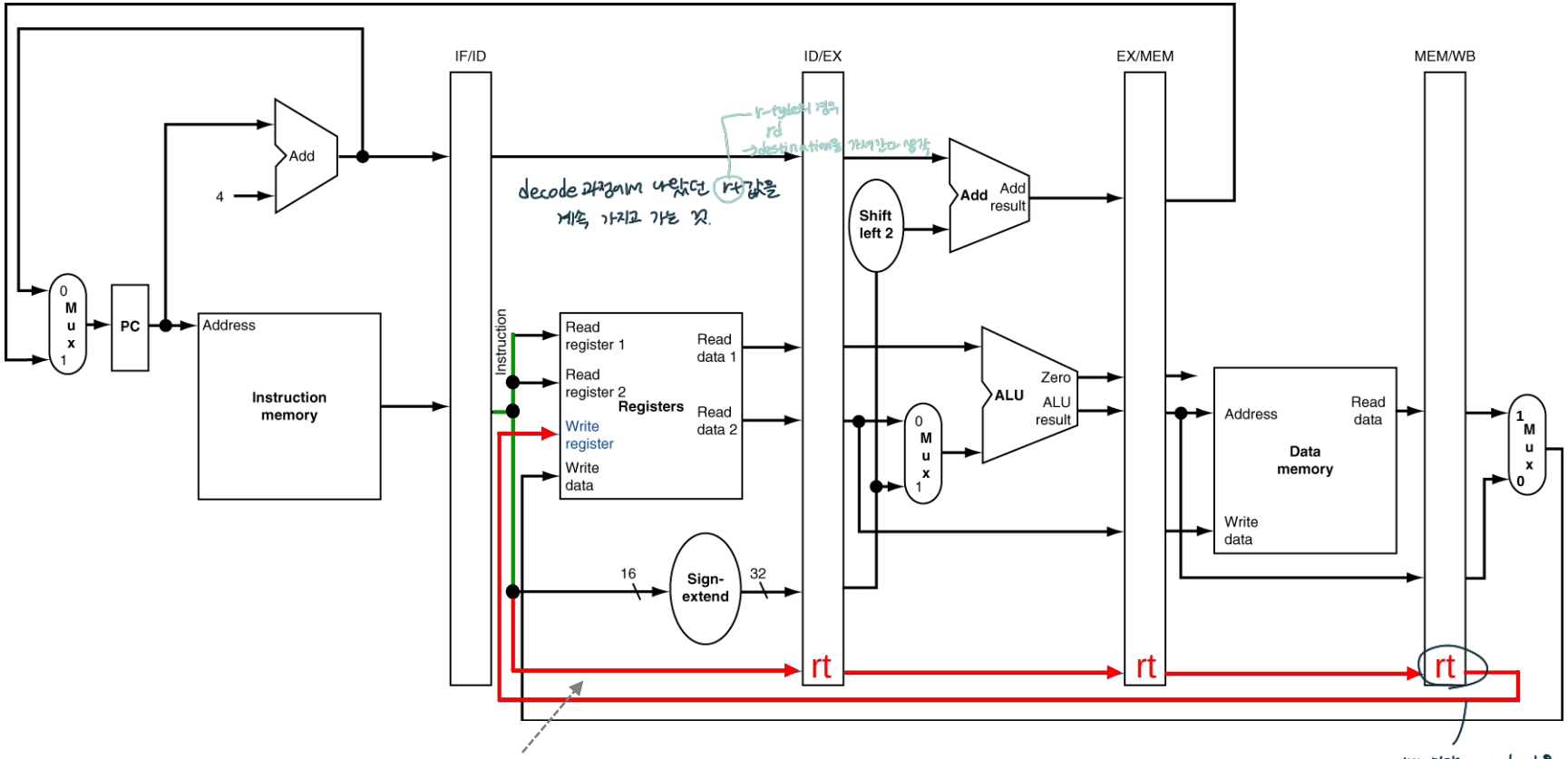
Single-Cycle Pipeline Diagram

- 아래 lw instruction의 WB단계 diagram에서의 문제점



Corrected Datapath for Load

타이밍 정상적으로 동작되지 않는 문제
→ 해저드라고 함.
나중에 나옴.



(data / control) Hazard

아마 해결이 안 된다면 pipeline 쓰는 사용 못함.

data Hazard (앞에 밝힌 문제)

control Hazard ex) beq

program counter 값에 대한 문제

IF: instruction fetch

ID: instruction decode/ register file read

EX: Execute/ address calculation

MEM: Memory access

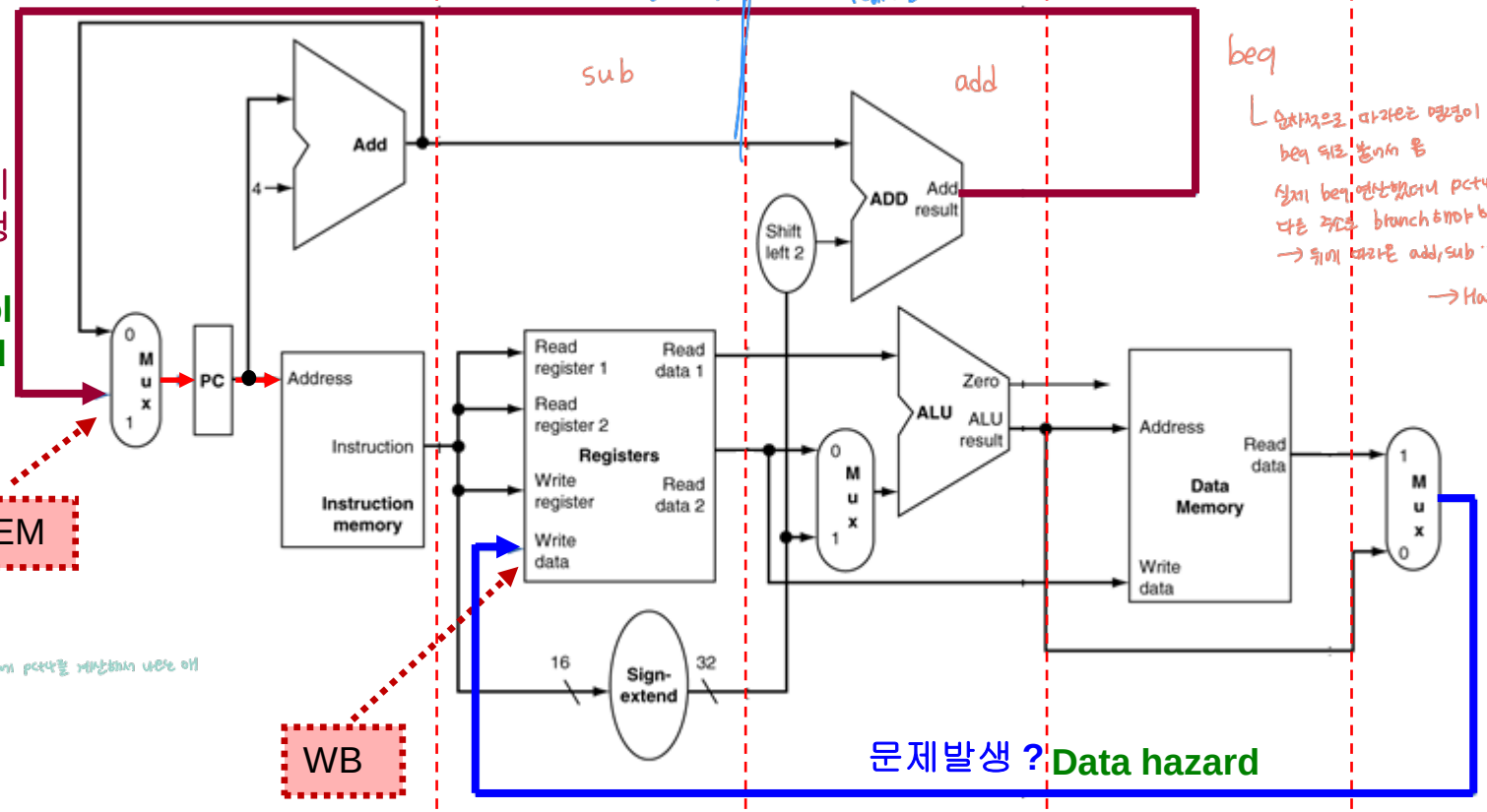
WB: Write back

decode 해봤더니 명령 beq

beq가 execution단계로 오면 PC값 바꿀 주소 계산

계산한 주소 PC에 가져다 쓰는 단계 → MEM 단계에 와야 가능.

문제 발생 ? **Control hazard**



beq

└ 순차적으로 마지막은 명령이 beq 되고 봤더니 뭐 실제 beq 연산했더니 pc+4가 아닌 다른 주소로 branch해야 함.

→ 뒤에 따라온 add, sub … → 다 잘못된 애들

→ Hazard

MEM

WB

문제발생 ? **Data hazard**

순차적

beq

add

sub

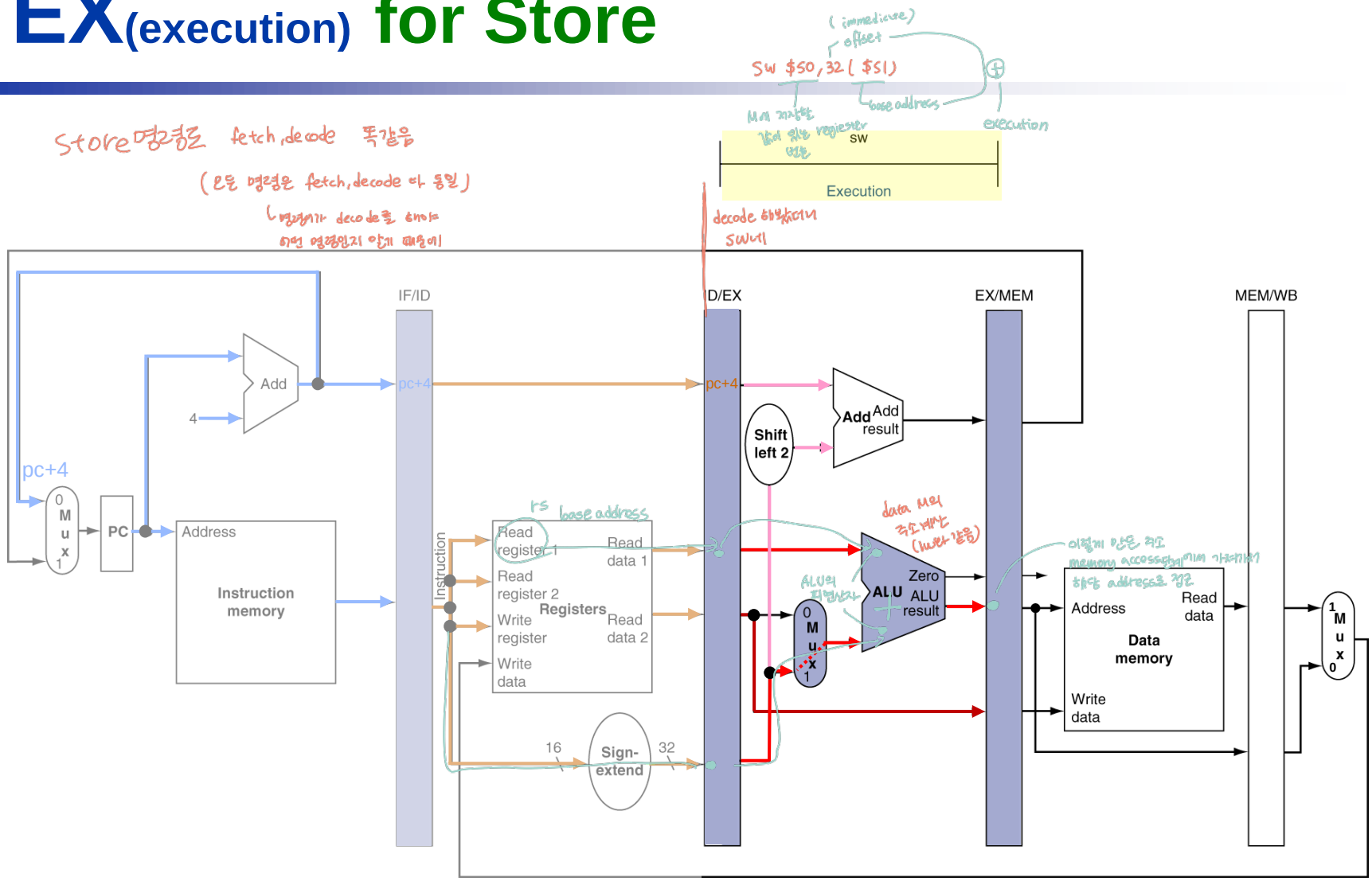
lw

⋮

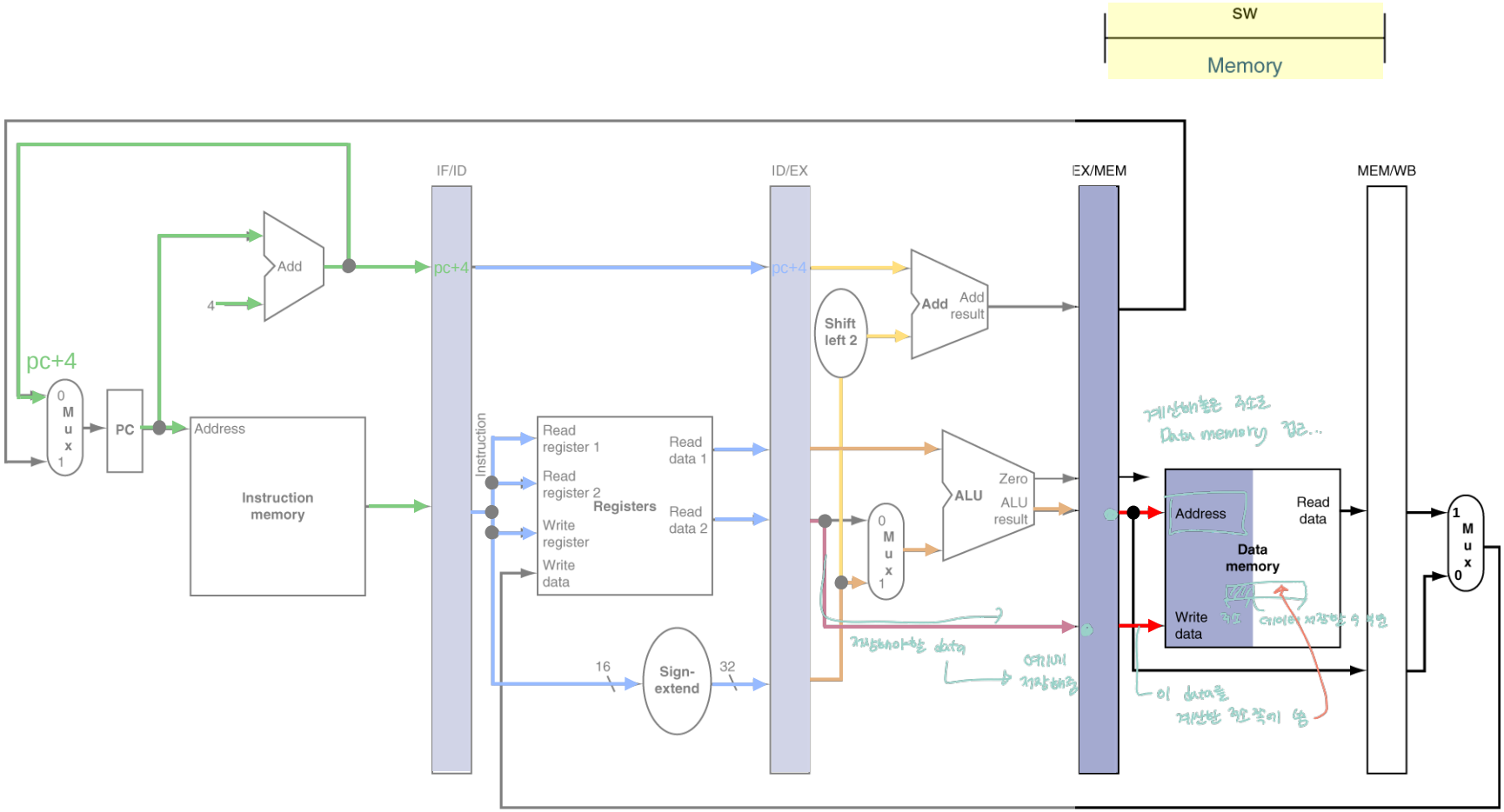
beq 다음에 pc+4를 계산해서 나온 애

└ MEM단계에서 그 주소를 PC에 넣어줬다면 …

EX(execution) for Store



MEM(memory) for Store

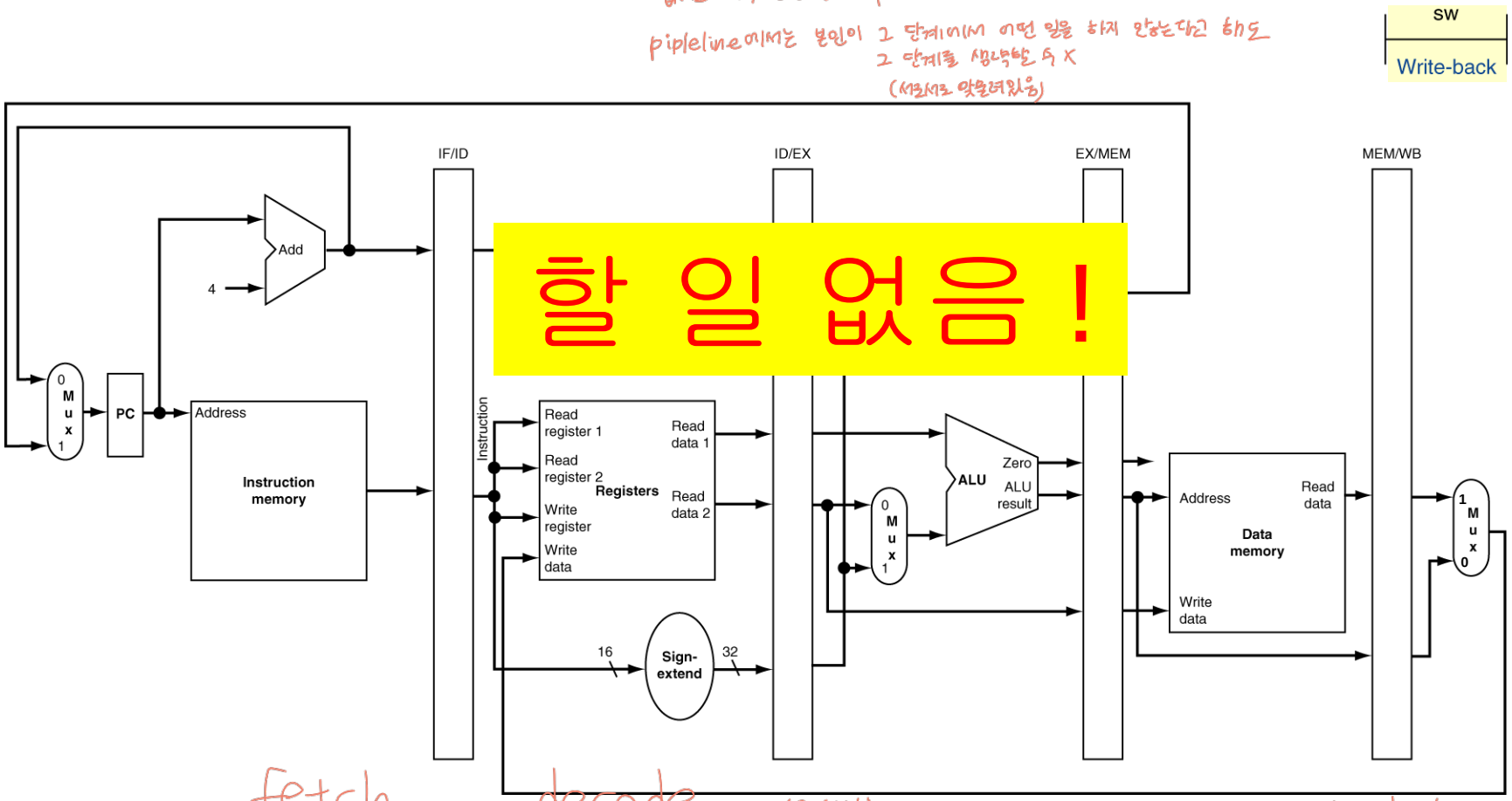


WB(write back) for Store

→ write-back 단계가 없어도 아무 상관 없음

but 그래도 writeback 단계에서 아무것도 안하는 상태로 그냥 기다리고 있어야 함

pipeline에서는 본인이 그 단계에서 어떤 일을 하지 않는다고 해도 그 단계를 생략할 수 X (서로서로 맞물려있음)



할 일 없음 !

	fetch	decode	execution	memory access	write-back
sw	200ps	200ps	200ps	200ps	

decode 사실 100ps 걸림.
but pipelining에서
register나 있지 때문에 200ps 동안 동작하게 만들어줌

결과적으로 single cycle보다 pipeline 방법이 좋음... 일반적으로 더

Pipelined Control (Simplified)

single cycle에서 봤던 회로와 거의 동일 (stage마다 pipeline register가 있는 것만 다름)

pipeline은 clock cycle길이가 clock cycle 짧음을 1개마다 한 단계의 일을 함.

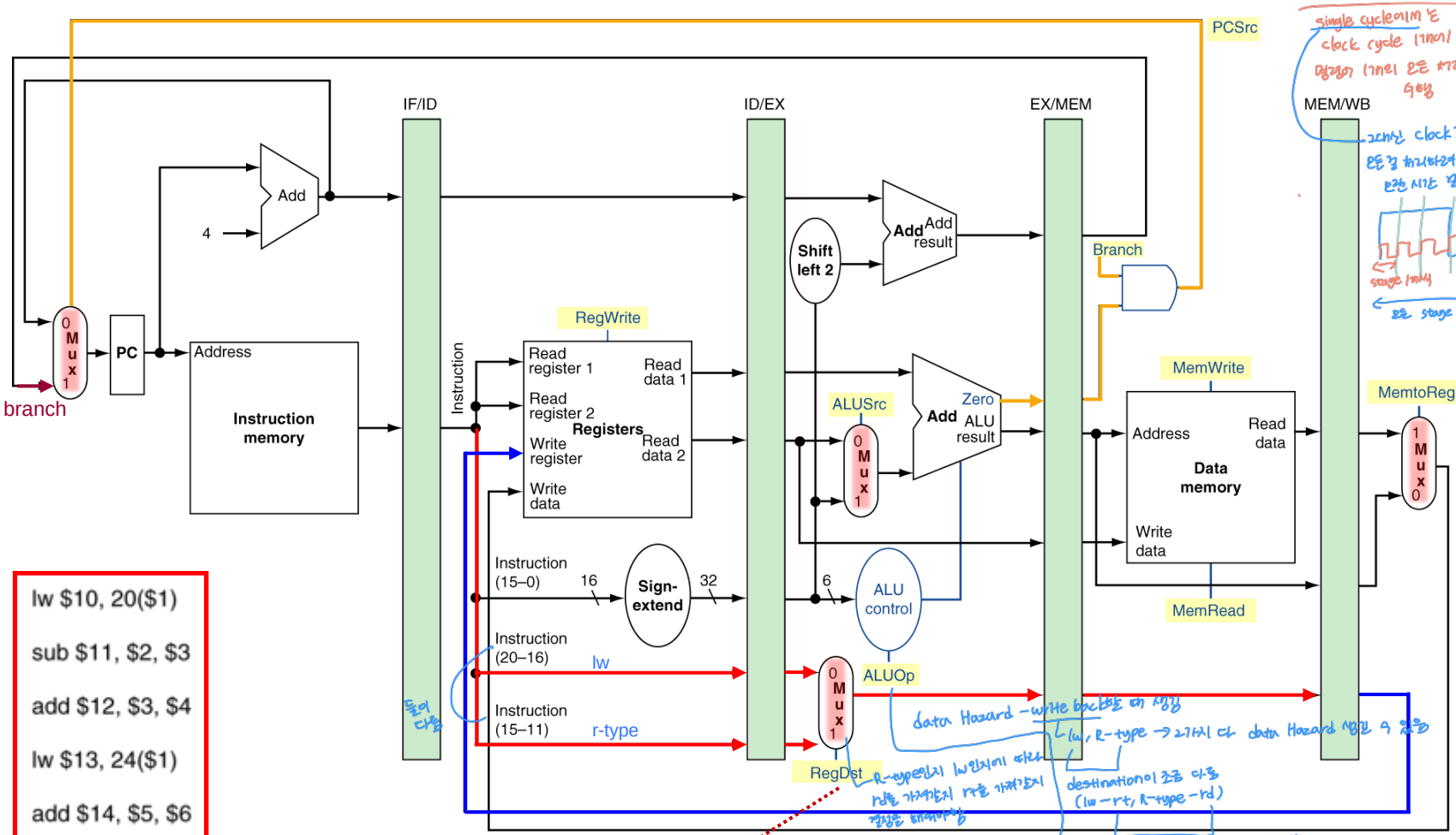
이 회로는 control signal이 생략되어 있음.

single cycle에서는 clock cycle 1개에 명령어 1개의 모든 처리 단계 수행

그래서 clock 길이가 길 모든 걸 처리하려면 모든 시간 필요

stage 1개씩

모든 stage 처리



lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6

둘이 다름

data Hazard - write back단계 때 생김

lw, R-type → 2가지 다 data Hazard 생길 수 있음

destination이 조금 다름 (lw - rt, R-type - rd)

R-type인지 lw인지에 따라 rd를 가져갈지 rt를 가져갈지 결정을 해야하는데

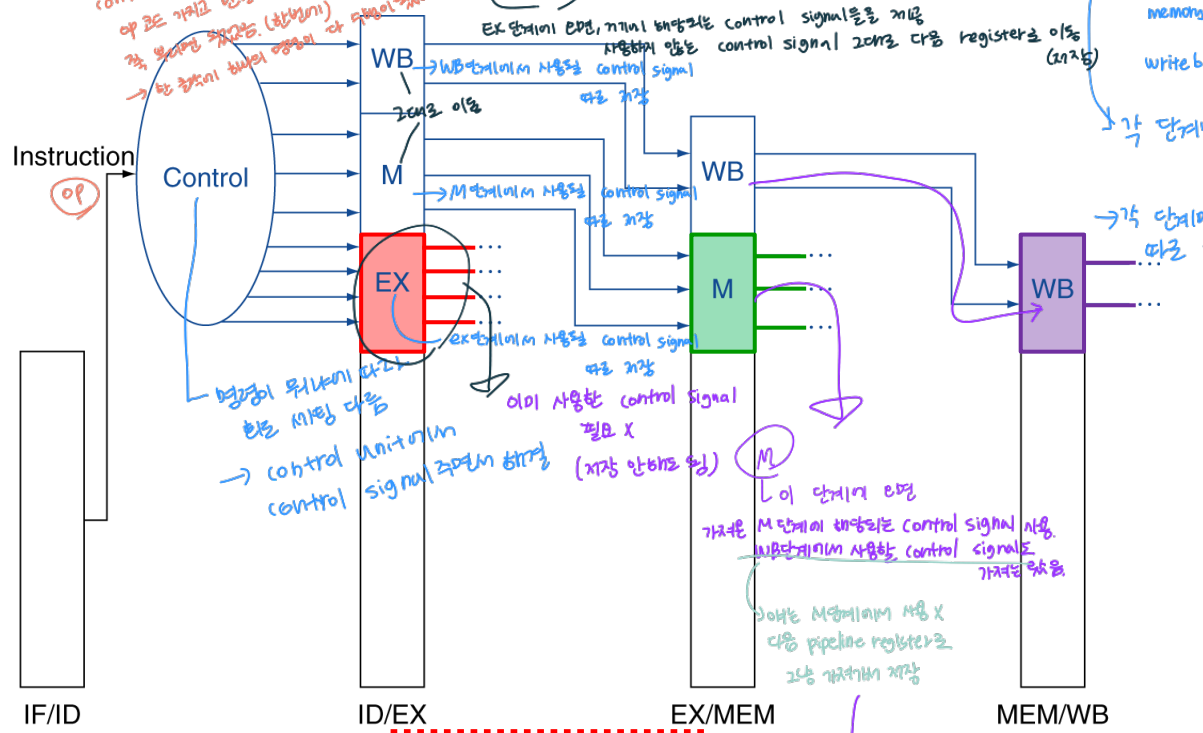
decode 결과를 ID에서 모르므로
EX에서 rt / rd 중 write할 register 선택

명령어를 보는 데 명령어에서의 ⓓ 위치가 lw나 r-type이 다름

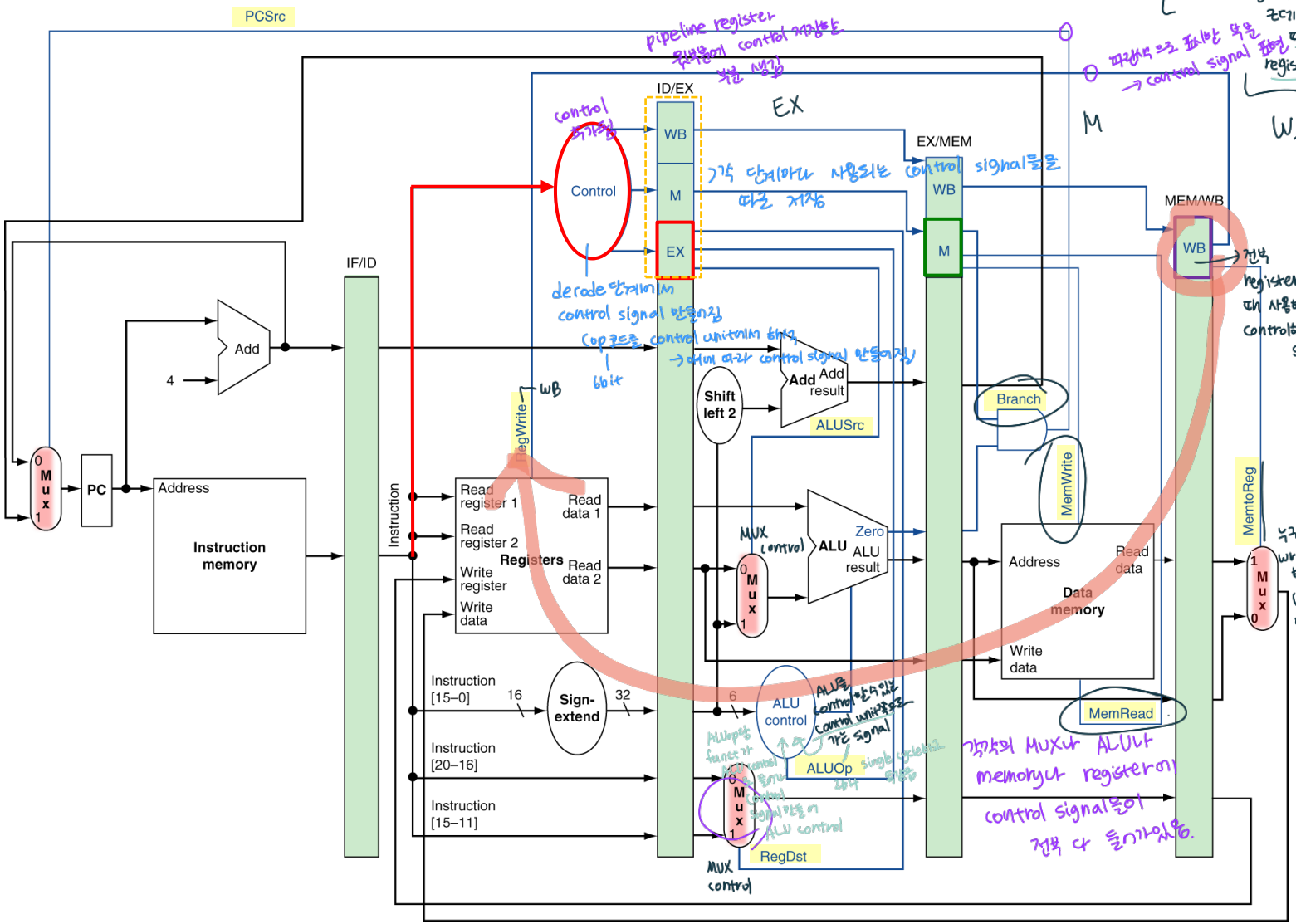
lw 경우 20-16 / r-type 15-11

Pipelined Control

- instruction에 따라 Control signals이 달라짐
  - 각 단계는 single-cycle implementation과 비슷함



Complete Pipelined Datapath(with control)



감사합니다.