



Ch.4-3 The Processor - pipeline hazard

* pipeline을 활용하면 다양한 문제 발생

* Pipeline Hazards

① Structural Hazards : 동시에 두나의 resource에 2개 이상의 접근 발생

↳ Resource Conflict

② Data Hazards : 한 item이 준비가 되기 전인데 사용하려는 경우

add	H, t2, t3
sub	t4, t2, H

↳ dependency 때문에 발생

③ Control Hazards : 조건이 결정되기 전에 decision 만들어내면

↳ loop/jump을 판단하기 전에 이미 add를 실행해 버리면 문제가 생김.

beq	H, t4, loop
add	H, t2, t3

⇒ waiting을 통한 해결 가능

① Structural Hazard

resource를 사용함에 있어 conflict가 발생해서 생기는 Hazard

ex) MIPS pipeline ↳ single memory

instruction마다 1.3 access [1 : Ins Mem
 0 : data Mem]

average CPI = 1.3

↳ But 한 cycle마다 1만 가능 ⇒ 잠시 동안 wait 하는게 필요

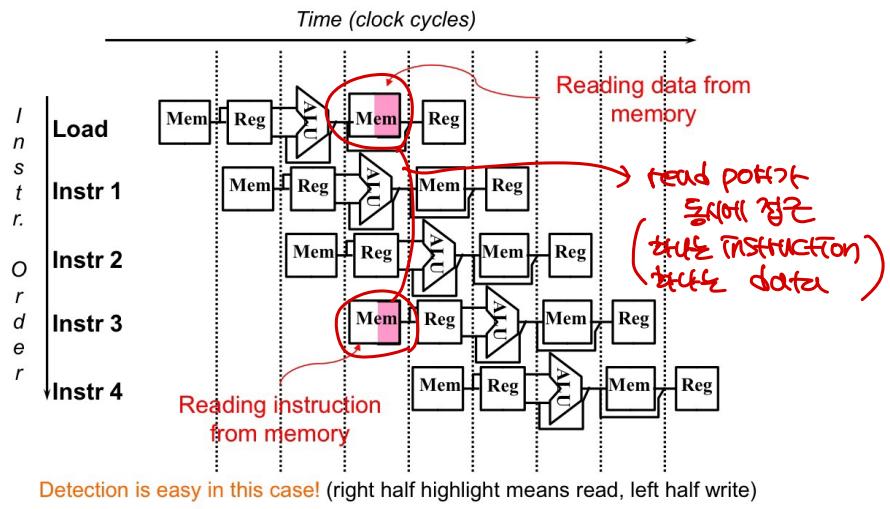
- 해결책 ① : stall → bubble

- 해결책 ② : pipelined datapath는 분리된 (inst/data memory) 필요

- 해결책 ③ : 한 cycle마다 memory가 1개 word보다 더 많이 read 혹은 write

↳ read port는 2개 이상 가지면 memory 더 많이 사용 가능

(↔ register file : 2 read / 1 write)



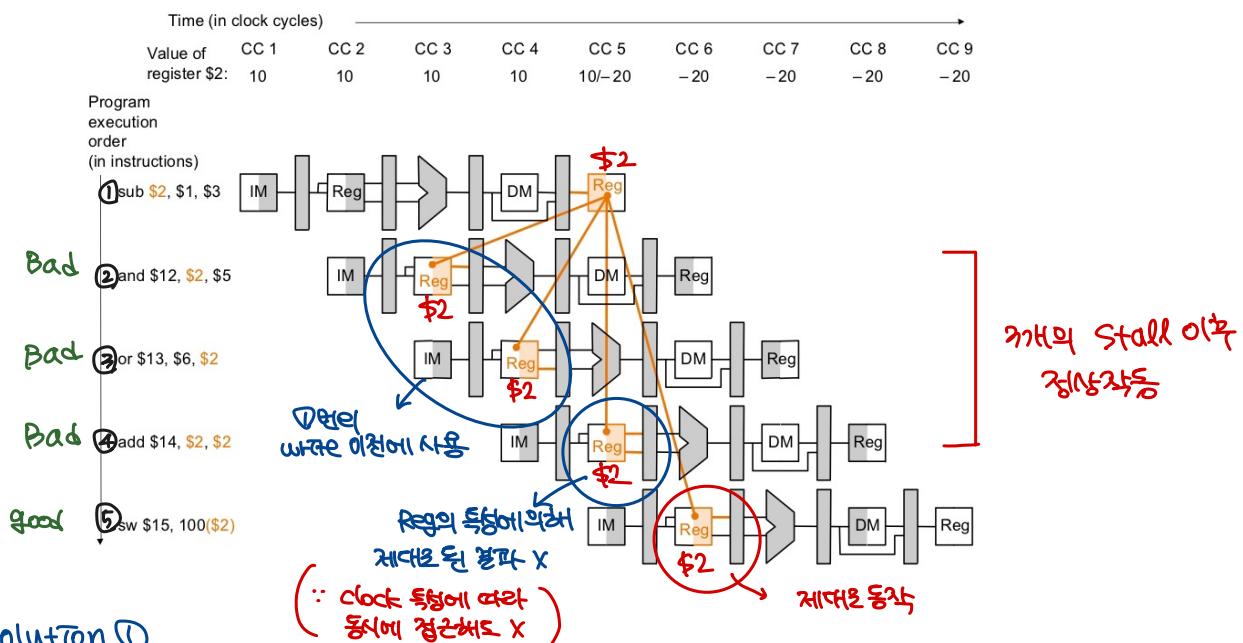
→ 결론: duplicate memory 사용, 즉, inst Mem과 Data Mem 둘다 ✗.

② Data Hazards

- ① sub \$2, \$1, \$3
- ② and \$12, \$2, \$5
- ③ or \$13, \$6, \$2
- ④ add \$14, \$2, \$2
- ⑤ SW \$15, 100(\$2)

①번의 결과값이 written되기 전에
②-⑤에서 \$2에 대한 read 발생
⇒ 명령어 간 dependency 문제

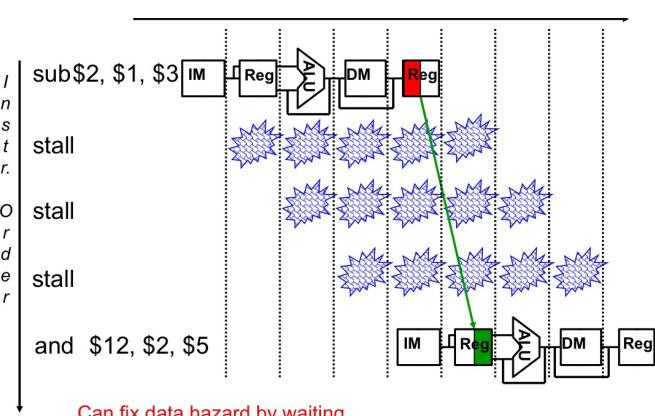
* dependency



Solution ①

* simple solution (SW 대체)
필요한 곳에 stall (nop 삽입)

But. 성능이 정말 나빠짐
→ throughput Bad



Solution ②

* Internal Bypassing path (= Reg File Internal Forwarding)

Reg File Access 허브에서의 전류를 줄여

$\rightarrow RA1 = WA$ 같은 때 register에

마지 data를 받아온다

\Rightarrow stall(1) 2stall(2) 0tall(3) 필요

Solution ③

* Forwarding (= data Forward Path)

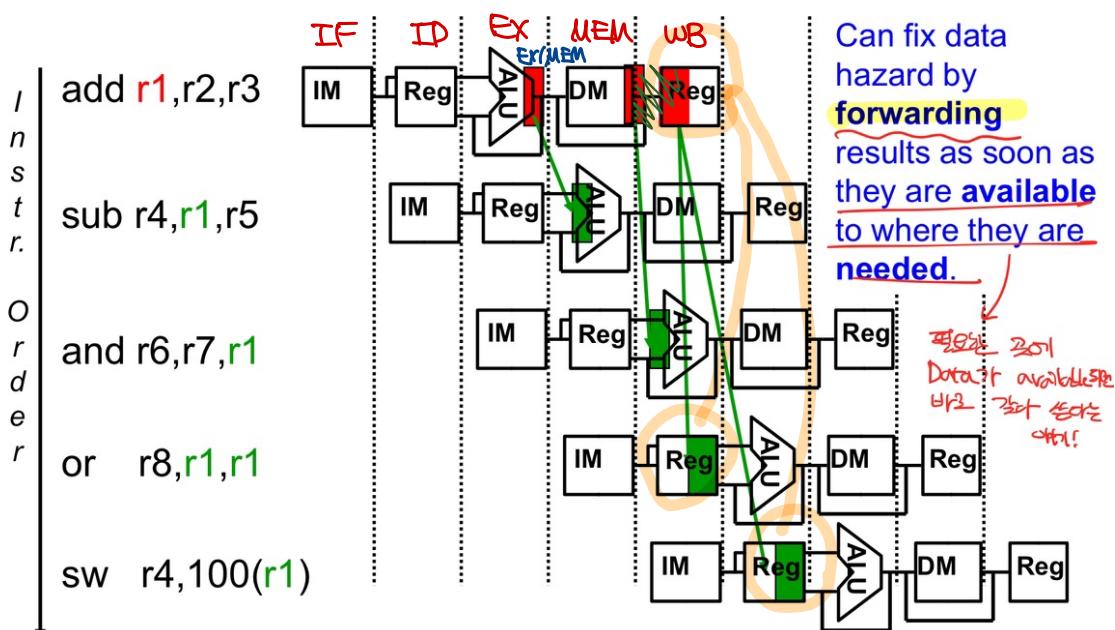
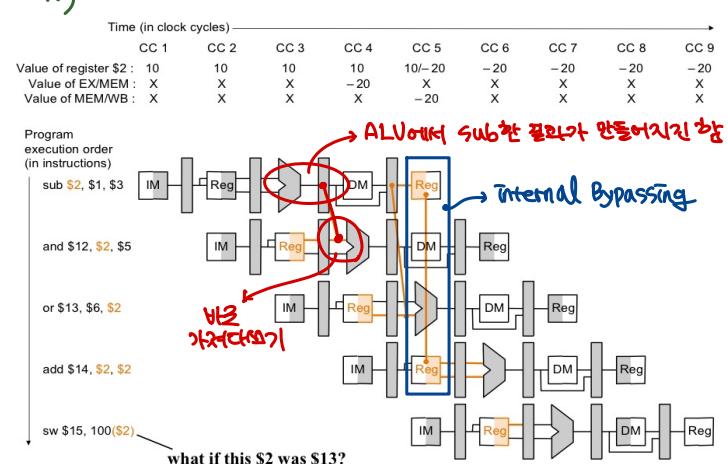
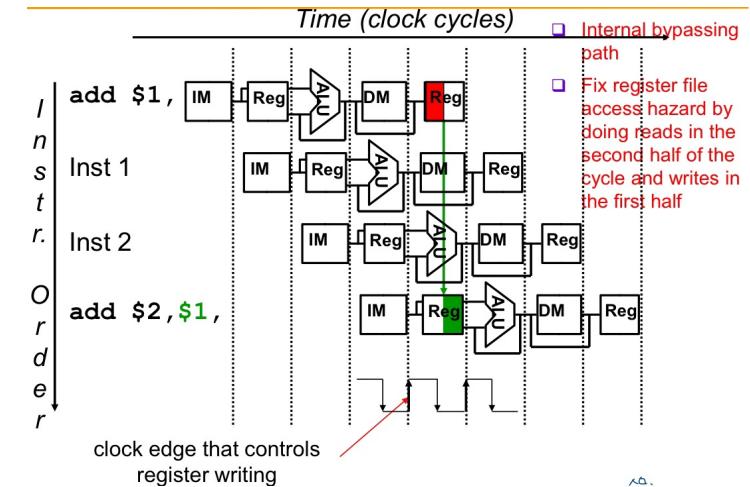
register file에 값을 다른 대기자

기다리지 않고 계산이 원활되면서 전달

\rightarrow pipeline Register에 있는 값을

forwarding

\rightarrow stall 없이



\Rightarrow Data Forwarding (즉 BY PASSING) 차트-1 알아보자

* Data Forwarding (Bypassing) \Rightarrow R-format 명령어만 생각해보자

data dependency 해결에 유용

\rightarrow R-type 명령어는 EX 지나야 결과 나옴

[EX Stage : R-type ALU Result 혹은 effective address calculation]

[MEM Stage : lw 결과 생성]

\Rightarrow 다른 Pipeline Register로부터 Data Forwarding \rightarrow ALU Input으로 사용해보자

\Rightarrow ALU Input 부분에 MUX 추가

\Rightarrow Rd data [EX/EL RS] \rightarrow passing 가능
 (ALU Input) [RT ALU Input]

00	: normal (ID/EX)
10	: 이전 (EX/MEM)
01	: 현재 (MEM/WB)

\Rightarrow Data dependency 해결 + stall 없이 정상 실행 \Rightarrow 성능 향상

i) Forward Detecting \rightarrow forwarding 필요 여부 어떻게 판단?

• Pipeline을 거친다ег number 알고 있어야 함

ex) IP/EX의 RS = ID/EX에 저장되어 있는 T4에 관한 Reg Num

• EX stage에 있는 ALU의 operand: ID/EX RS, ID/EX RT를 Input으로 받음

① Data Hazard 발생 예측 확인

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

} Fwd from EX/MEM / pipeline reg

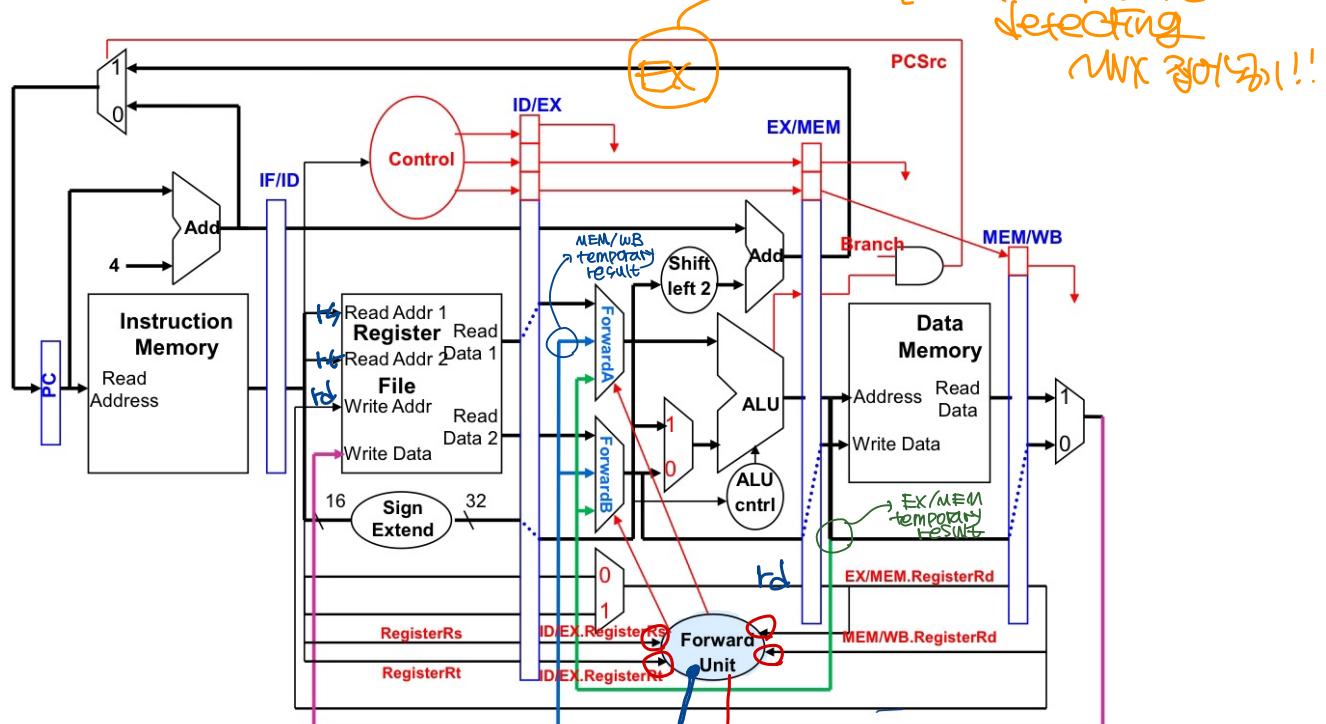
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

} Fwd from MEM/WB / pipeline reg

② 해당되는 Control Signal이 인지 확인 \rightarrow 그대로 Reg Write

③ Rd 값이 \$zero가 아님지 확인



Control line inputs to Forward Unit EX/MEM.RegWrite and MEM/WB.RegWrite not shown on diagram

Control signal
제공 안함.

\Rightarrow data Forwarding
여기 걸림차이로 Control Input 차이

ii) Data Forwarding Control Condition

① simple

- EX/MEM hazard: 이전 instruction forwarding

```
C
//RegisterRd는 written될 register의 number
//RegisterRs, RegisterRt는 Rs, Rt register의 number
if (EX/MEM.RegisterRd == ID/EX.RegisterRs)
    ForwardA = 10 //Forwarding과 관련한 MUX
if (EX/MEM.RegisterRd == ID/Ex.RegisterRt)
    ForwardB = 10 //Forwarding과 관련한 MUX
```

해당 경우에는 이전 instruction에서 결과를 Forwarding해야 한다.

- MEM/WB hazard: 두번 이전 instruction forwarding

```
C
if (MEM/WB.RegisterRd == ID/EX.RegisterRs)
    ForwardA = 01
if (MEM/WB.RegisterRd == ID/Ex.RegisterRt)
    ForwardB = 01
```

해당 경우에는 2번 이전의 instruction에서 결과를 Forwarding해야 한다.

EX/MEM 이전 10
MEM/WB 이이전 01

② Control Signal 추가 ⇒ 1인지 확인 (RegWrite)

- EX/MEM hazard:

```
C
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd == ID/EX.RegisterRs))
    ForwardA = 10 //Forwarding과 관련한 MUX
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd == ID/Ex.RegisterRt))
    ForwardB = 10 //Forwarding과 관련한 MUX
```

EX/MEM.RegWrite 값이 1일 때(RegWrite를 수행할 때에만) Data

- MEM/WB hazard:

```
C
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd == ID/EX.RegisterRs))
    ForwardA = 01
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd == ID/Ex.RegisterRt))
    ForwardB = 01
```

MEM/WB.RegWrite 값이 1일 때(RegWrite를 수행할 때에만) Data

RegWrite는 1인지?
RegWrite는 1인지?

③ \$zero register가 아닌 조건 추가

- EX/MEM hazard:

```
C
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRs))
    ForwardA = 10 //Forwarding과 관련한 MUX
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
    and (EX/MEM.RegisterRd == ID/Ex.RegisterRt))
    ForwardB = 10 //Forwarding과 관련한 MUX
```

EX/MEM.RegisterRd가 \$zero가 아닐 때에만, Data Forwarding 하도록 설
계.

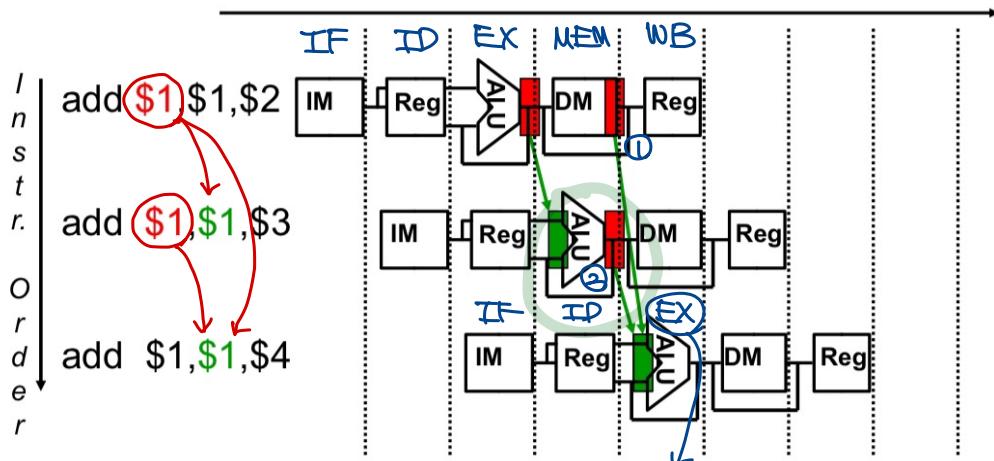
- MEM/WB hazard:

```
C
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0)
    and (MEM/WB.RegisterRd == ID/EX.RegisterRs))
    ForwardA = 01
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0)
    and (MEM/WB.RegisterRd == ID/Ex.RegisterRt))
    ForwardB = 01
```

MEM/WB.RegisterRd가 \$zero가 아닐 때에만, Data Forwarding 하도록 설

RegisterRd는 \$zero 아님?
RegisterRd는 \$zero 아님?

④ 특수한 경우를 생각해보자



①, ② 두 곳에서 모두 forward 가능

But, 가장 최근인 ③에서 해야 함 (④은 X)

⇒ 추가적인 조건 필요

- MEM/WB hazard: (해당 경우에만 발생하는 문제이다. 해당 경우에만 조건을 추가한다)

```

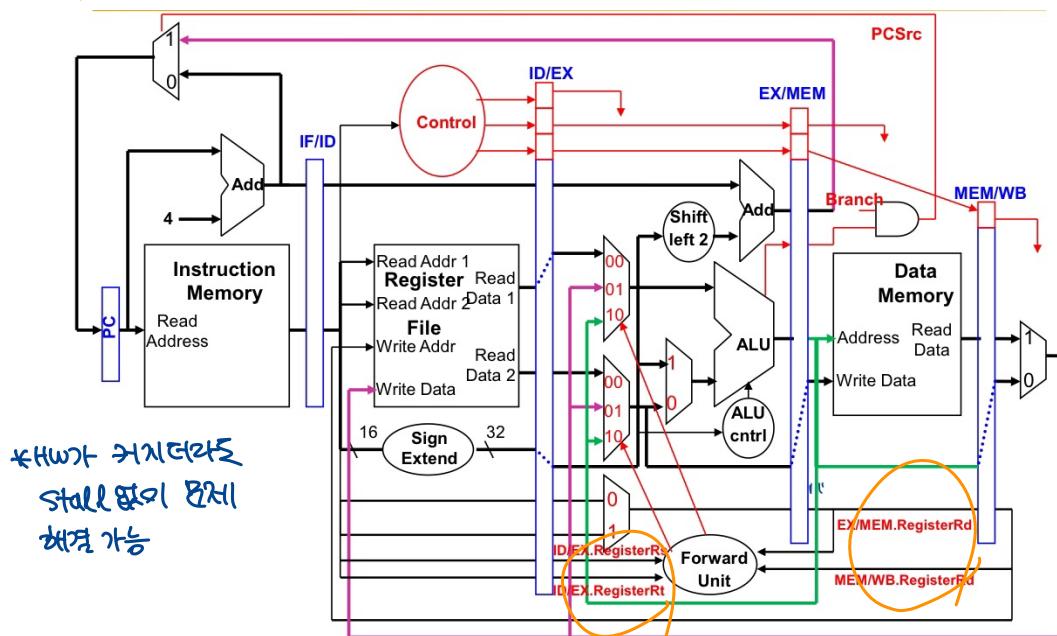
C
if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd != 0)
    and (MEM/WB.RegisterRd == ID/EX.RegisterRs)
    and (not (EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
              and (EX/MEM.RegisterRd == ID/EX.RegisterRs)))
)
ForwardA = 01

if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd != 0)
    and (MEM/WB.RegisterRd == ID/EX.RegisterRt)
    and (not (EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
              and (EX/MEM.RegisterRd == ID/EX.RegisterRs)))
)
ForwardB = 01

```

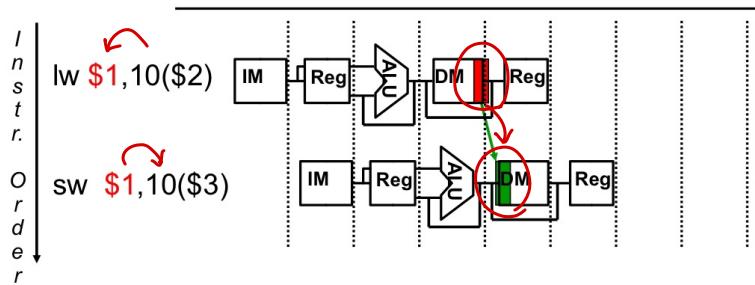
I 번이전 INSTRUCTION forward가
이전 stage에서 이루어진 전작이
있으지 check
NOT effect EX(MEM)

* 3/2 Data Path



Control line inputs to Forward Unit EX/MEM.RegWrite and MEM/WB.RegWrite not shown on diagram

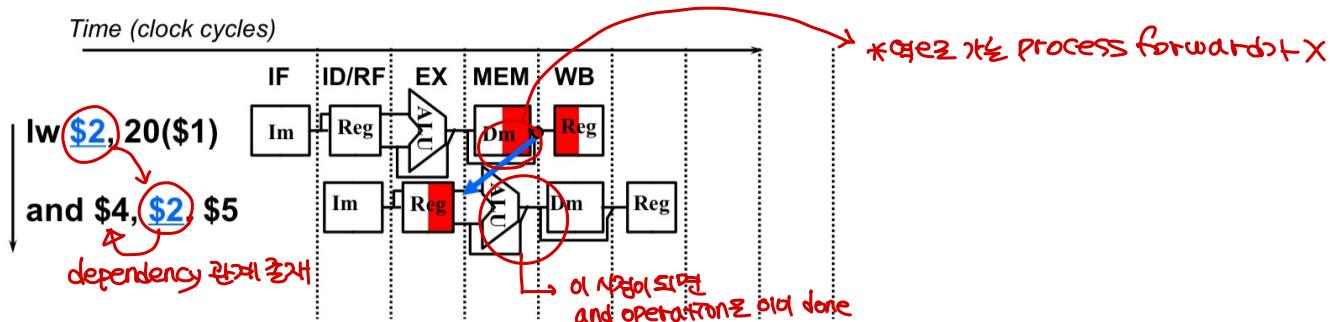
* Memory-to-Memory Copies → R format 이면 Data dependency 발생하는 경우



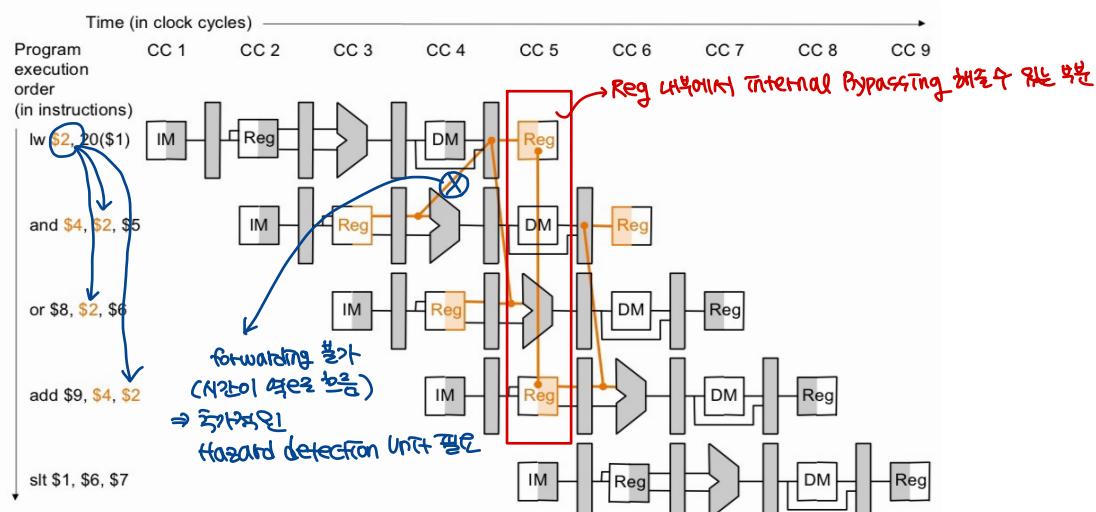
Solution

- ① MEM 단계를 Forward Unit으로
 - ② Load 시에 Stalling 방지

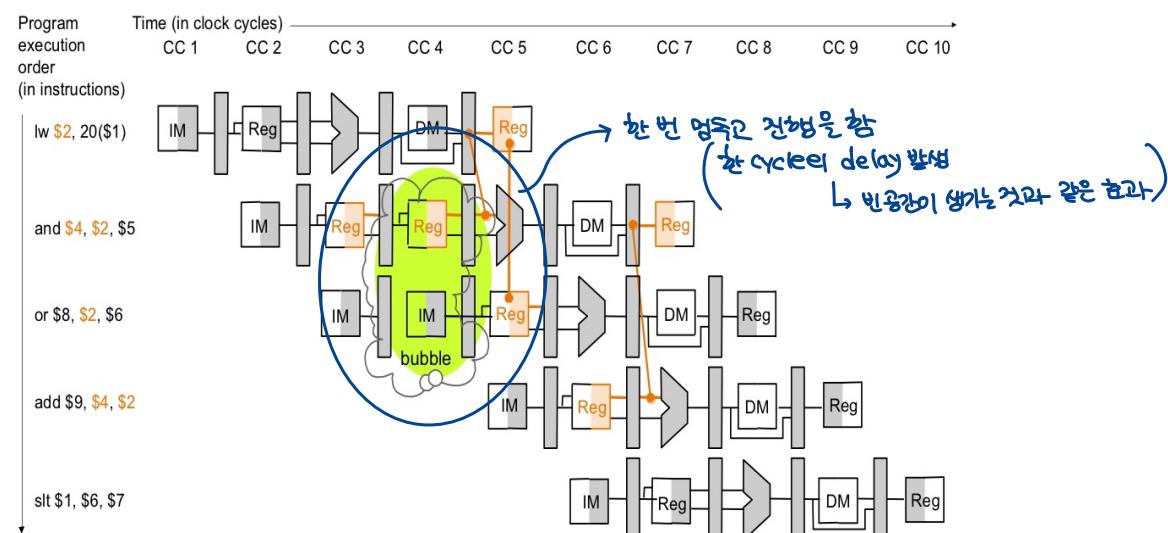
* Data Forwarding → Load!

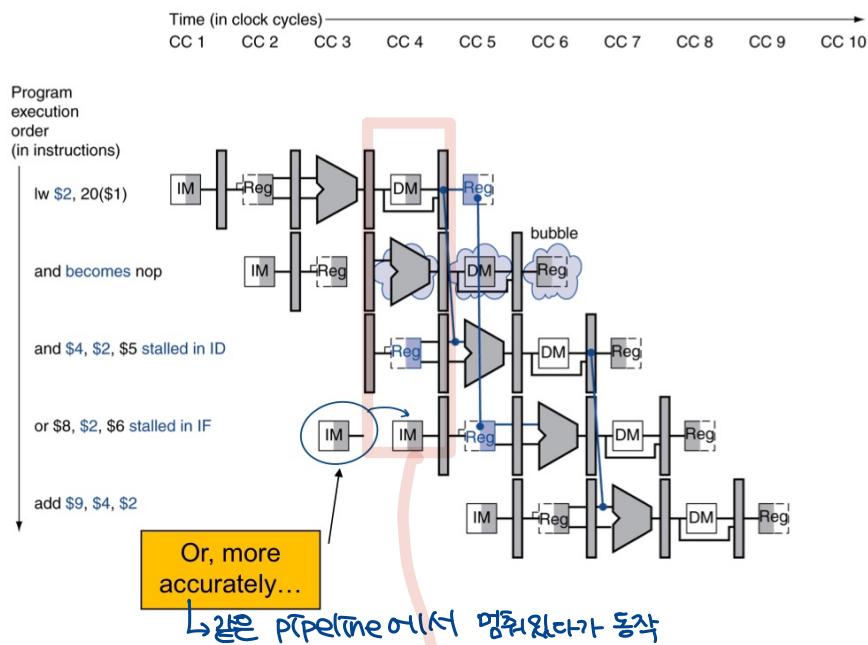
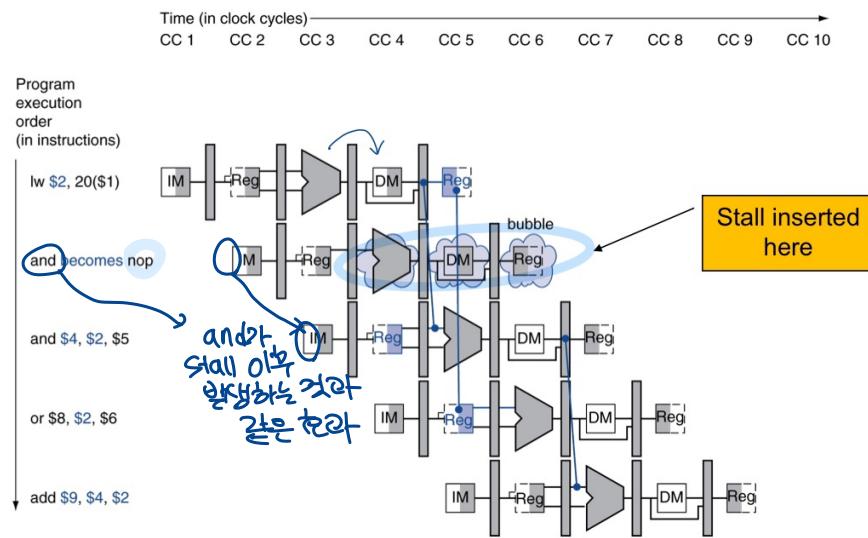


⇒ 무언가에 delay / stall 수행해야함.



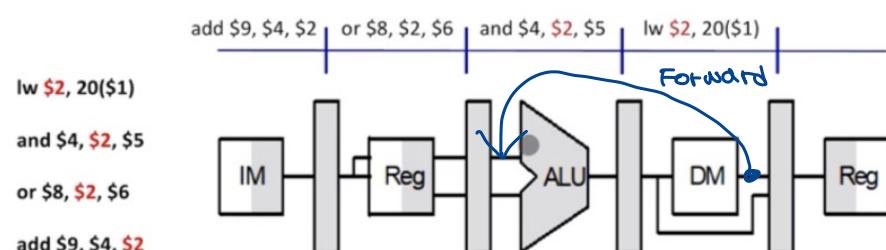
→ Stalling (HW쪽으로 멈추는 방법)





⇒ (W에 대한 stall을 피하기) 위(위)
이렇게 적용하면 안될까?

CC4



→ 2개의 stage를 한 번에 처리

→ Longest Path Delay가 2개가 되어 1st cycle 안에 계산이 불가능

ii) Load-use Hazard Detection Unit

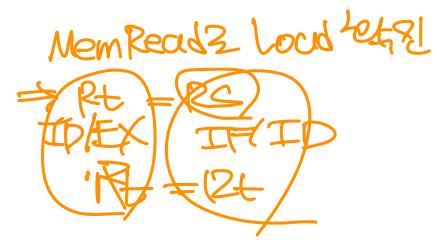
ID에서 hazard detection 단계에 필요함

→ Load instruction ↔ Load 한 data 사용
→ Stall을 걸어 넣음

1. ID Hazard Detection (MemRead) → control signal 터미널

```
if (ID/EX.MemRead) ⚡
and ((ID/EX.RegisterRt = IF/ID.RegisterRs)
or (ID/EX.RegisterRt = IF/ID.RegisterRt))
stall the pipeline
```

→ 1 cycle stall ↳ 개별 forwarding logic 이용 가능



iii) Stall Hardware

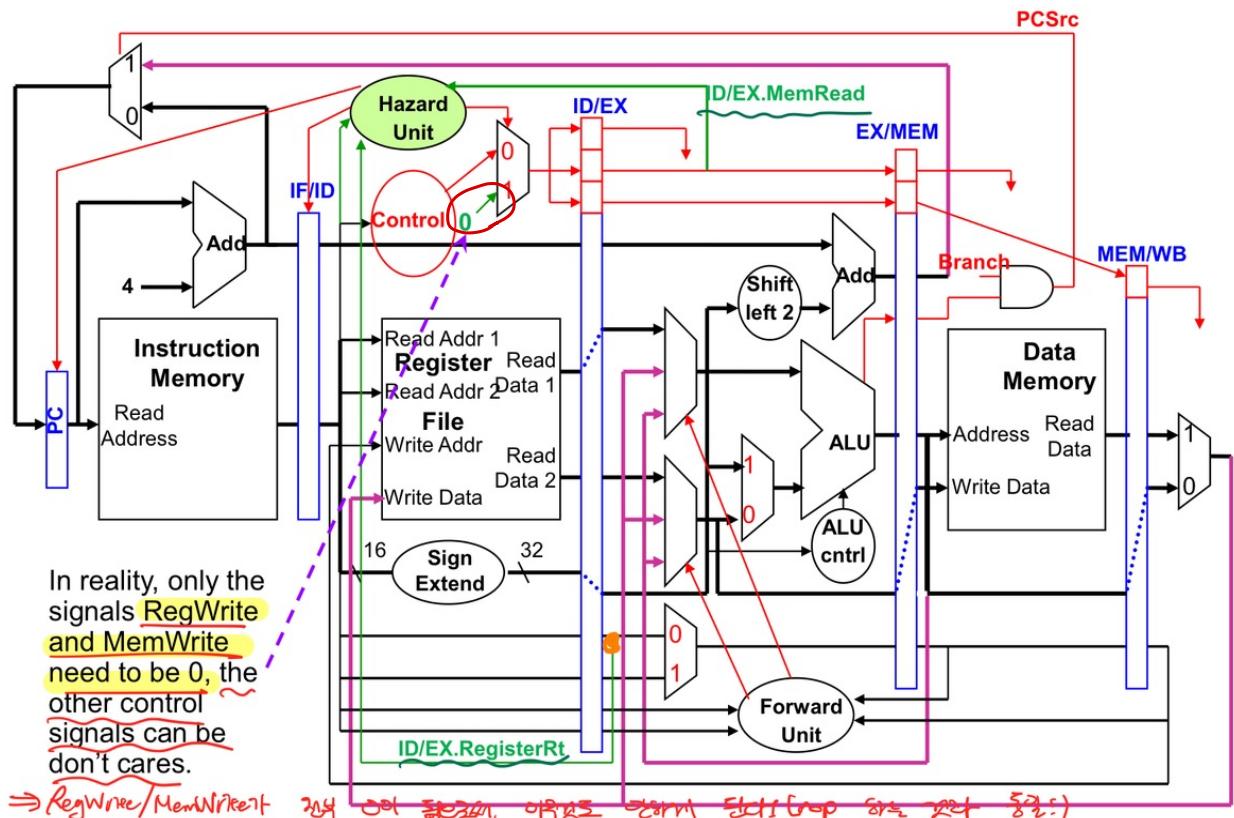
① IF/ID stage에 있는 INSTRUCTION 계속 progress 되는 것 방지

PC register / IF/ID register가 계속 progress 되는 것 방지

Hazard detection unit : PC의 IF/ID register들의 writing job control

② EX stage 뒤부터는 INSTRUCTION flush 필요 (nop execute)

→ Hazard Detection Unit이 0을 선택 → datapath 일부 변환 X





Code Scheduling to Avoid Stalls

conflict 35(2) decide!!

- ❑ Reorder code to avoid use of load result in the next instruction
- ❑ C code for $A = B + E; C = B + F;$

