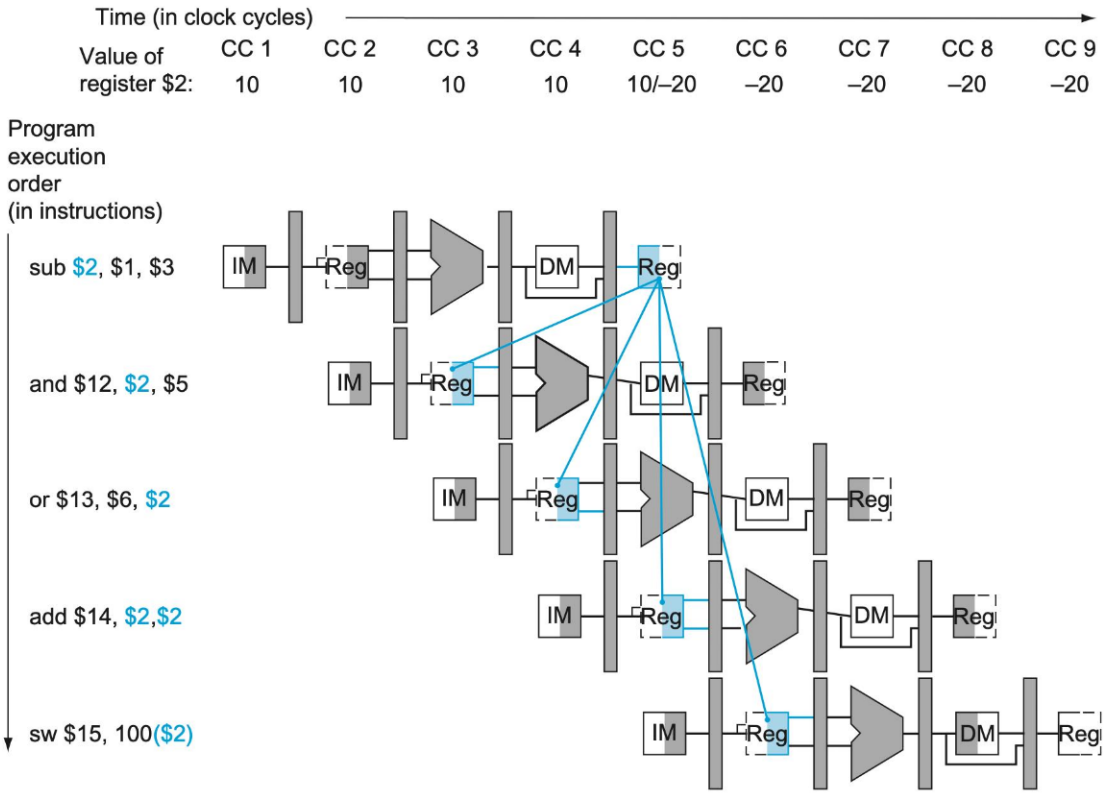
Computer Architecture (ENE1004)

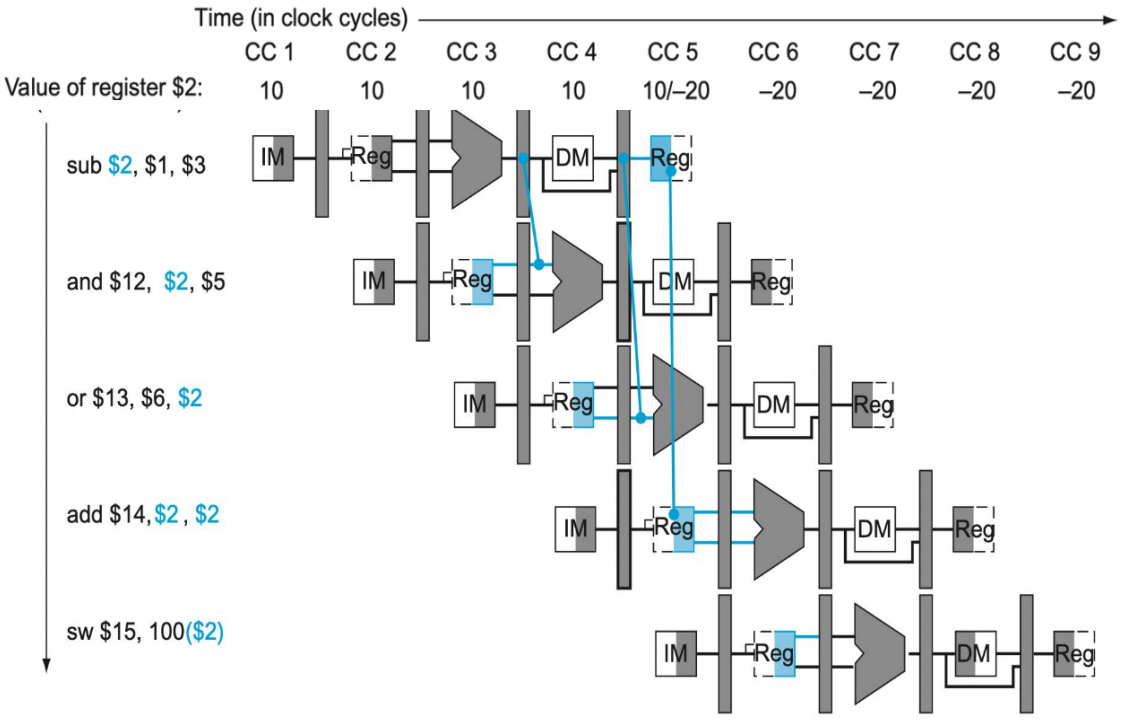
▼ Lec 16

Lec 16: The Processor 9

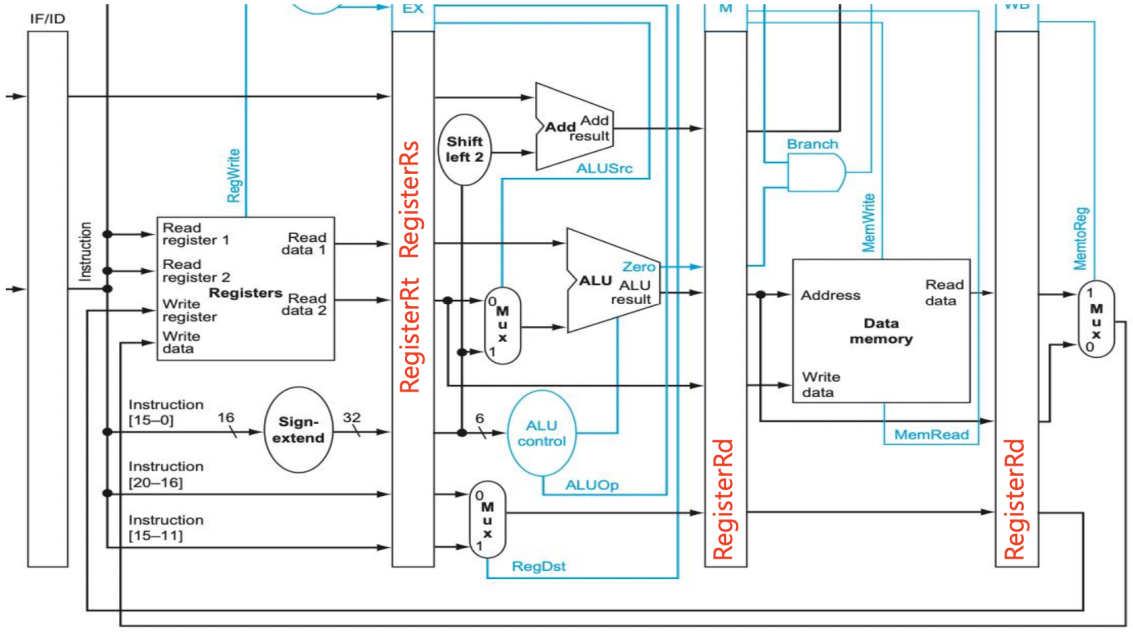
Data Hazard - RegWrite Scenarios



Data Forwarding (Solution) - RegWrite Scenarios



Data Hazard Detection - RegWrite Scenarios



- Four different cases where the data hazard occurs
  - EX / MEM.RegisterRd == ID / EX.RegisterRs ? (btw dest of 1st instruction & 1st source of 2nd instruction : 첫 번째 명령의 목적지 & 두 번째 명령의 첫 번째 소스)

- EX/MEM.RegisterRd == ID/EX.RegisterRt ? (btw dest of 1st instruction & 2nd source of 2nd instruction : 첫 번째 명령의 목적지 & 두 번째 명령의 두 번째 소스)
- MEM/WB.RegisterRd == ID/EX.RegisterRs ? (btw dest of 1st instruction & 1st source of 3rd instruction : 첫 번째 명령의 목적지 & 세 번째 명령의 첫 번째 소스)
- MEM/WB.RegisterRd == ID/EX.RegisterRt ? (btw dest of 1st instruction & 2nd source of 3rd instruction : 첫 번째 명령의 목적지 & 세 번째 명령의 두 번째 소스)

Data Hazard Detection - RegWrite Scenarios

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
```

[For the 2nd instruction]

```
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
```

[For the 3rd instruction]

(1) We need to check whether the first instruction writes register or not (첫 번째 명령어가 레지스터를 쓰는지 여부를 확인해야 한다.)

- If it does not write a register, we do not have to worry about the data hazard (레지스터를 쓰지 않는다면 데이터 위험은 걱정할 필요가 없다.)
- Check EX/MEM.RegWrite and MEM/WB.RegWrite for the 2nd and 3rd instruction (두 번째와 세 번째 명령어에 대해 EX/MEM.RegWrite와 MEM/WB.RegWrite를 확인한다.)

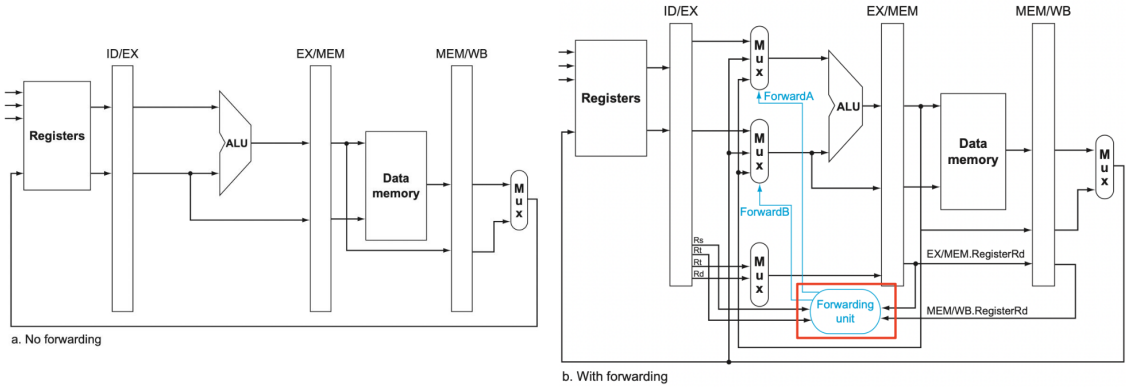
(2) We need to check whether the write register number is “0” or not (쓰기 레지스터 번호가 “0”인지 여부를 확인해야 한다.)

- sll $0, $1, 2; this does not make any sense; but, programmers/compiler may make a mistake (sll $0, $1, 2; 이것은 의미가 없다; 그러나 프로그래머/컴파일러가 실수할 수 있다.)
- In such a case, we do not have to worry about the data hazard (이 경우 데이터 위험에 대해 걱정할 필요가 없다.)
- Check whether EX/MEM.RegisterRd and MEM/WB.RegisterRd for the 2nd and 3rd instructions (2번째와 3번째 명령어에 대해 EX/MEM.RegisterRd와 MEM/WB.RegisterRd가 맞는디 확인한다.)

(3) We need to check whether the write register number is used as a source register (쓰기 레지스터 번호가 소스 레지스터로 사용되는지 확인해야 한다.)

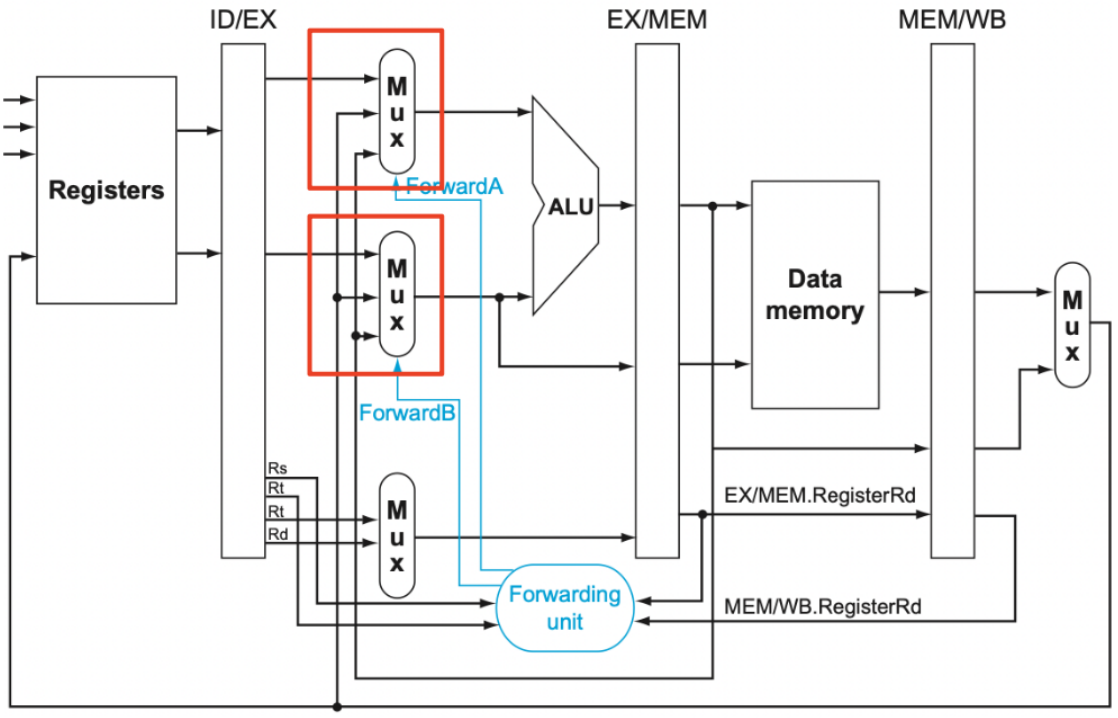
- Check the four conditions in the previous slide (이전 슬라이드의 네 가지 조건을 확인한다.)

Data Forwarding - Implementation (1)



- We place the “forwarding unit”
- (i) it detects the data hazard based on the following values (inputs) (다음 값(입력)을 기반으로 데이터 위험을 감지한다.)
  - EX/MEM.RegisterRd, MEM/WB.RegisterRd, ID/EX.RegisterRs, ID/EX.RegisterRt
  - RegWrite signal value (this figure does not show it) (RegWrite 신호 값 (이 그림에는 표시되지 않음))

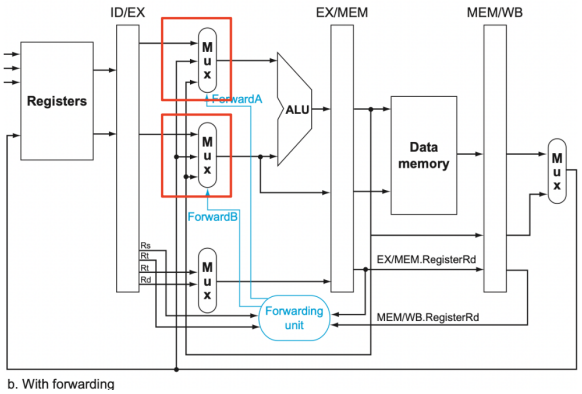
Data Forwarding - Implementation (2)



b. With forwarding

- We place "multiplexers" for two sources of ALU; the inputs of the multiplexers are (두 가지 ALU 소스에 대해 “멀티플렉서”를 배치한다. 멀티플렉서의 입력은 다음과 같다.)
  - (1) from register file (or ID/EX register) - no data hazard (레지스터 파일 (또는 ID/EX 레지스터)에서 - 데이터 위험 없음)
  - (2) from MEM/WB register – forwarding to the 3rd instruction (MEM/WB 레지스터에서 - 세 번째 명령으로 전달함)
  - (3) from EX/MEM register – forwarding to the 2nd instruction (EX/MEM 레지스터에서 - 두 번째 명령으로 전달함)

Data Forwarding - Implementation (3)

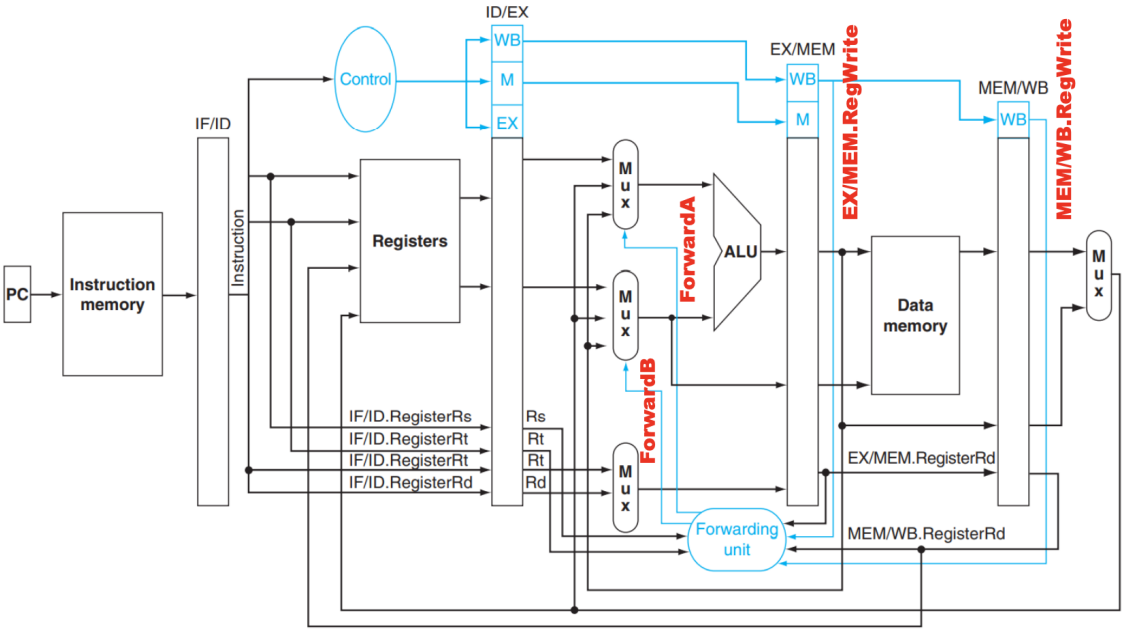


b. With forwarding

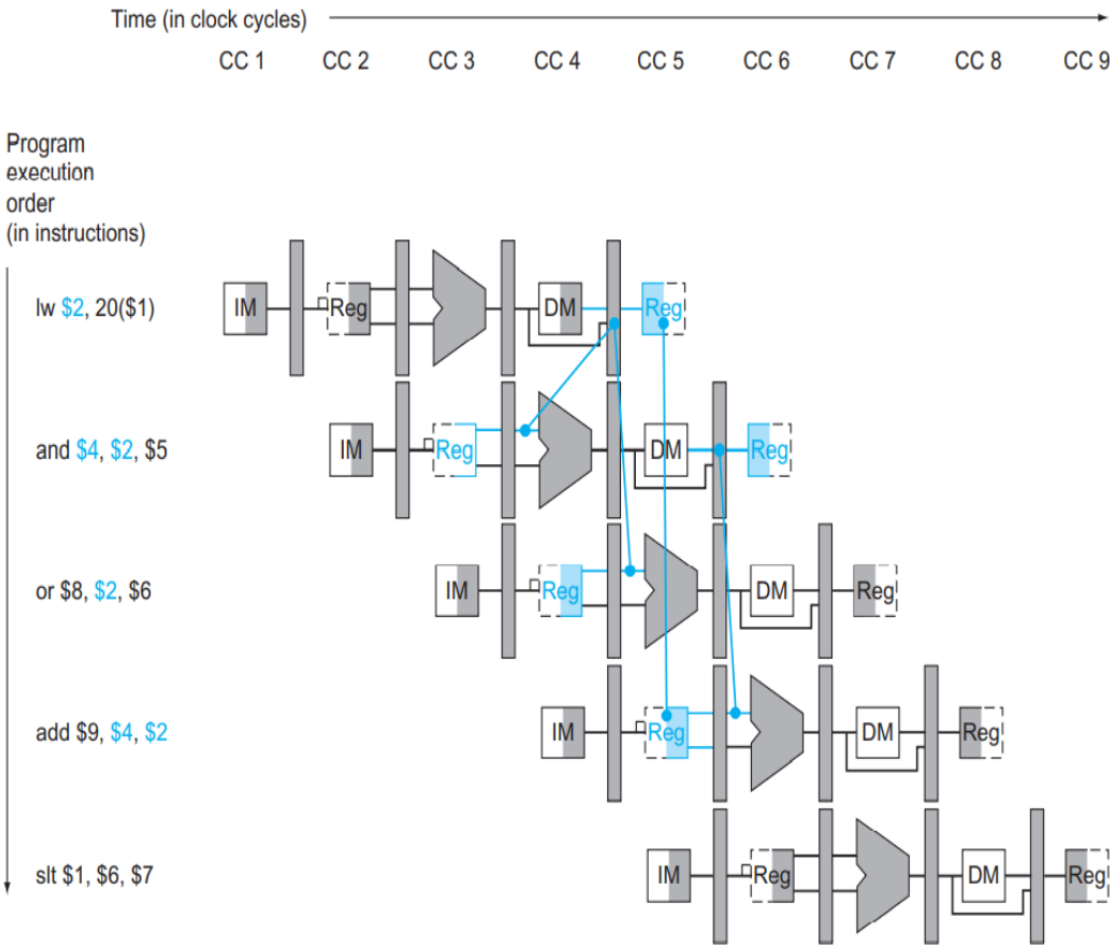
Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

- (ii) The forwarding unit determines one of the three inputs using 2-bit control signal (forwarding 유닛은 2비트 제어 신호를 사용하여 3개의 입력 중 하나를 결정한다.)
    - ForwardA & ForwardB determine the 1st and the 2nd source of the ALU (ForwardA 및 ForwardB는 ALU의 첫 번째 및 두 번째 소스를 결정한다.)
    - “00” selects the 1st input (no data hazard) (”00”은 첫 번째 입력을 선택한다. (데이터 위험 없음))
    - “01” selects the 2nd input (forwarding from MEM/WB for the 3rd instruction) (”01”은 두 번째 입력을 선택한다. (세 번째 명령에 대해 MEM/WB에서 포워딩))
    - “10” selects the 3rd input (forwarding from EX/MEM for the 2nd instruction) (”10”은 세 번째 입력을 선택한다. (두 번째 명령에 대해 EX/MEM에서 포워딩))

Data Forwarding - Implementation (4)



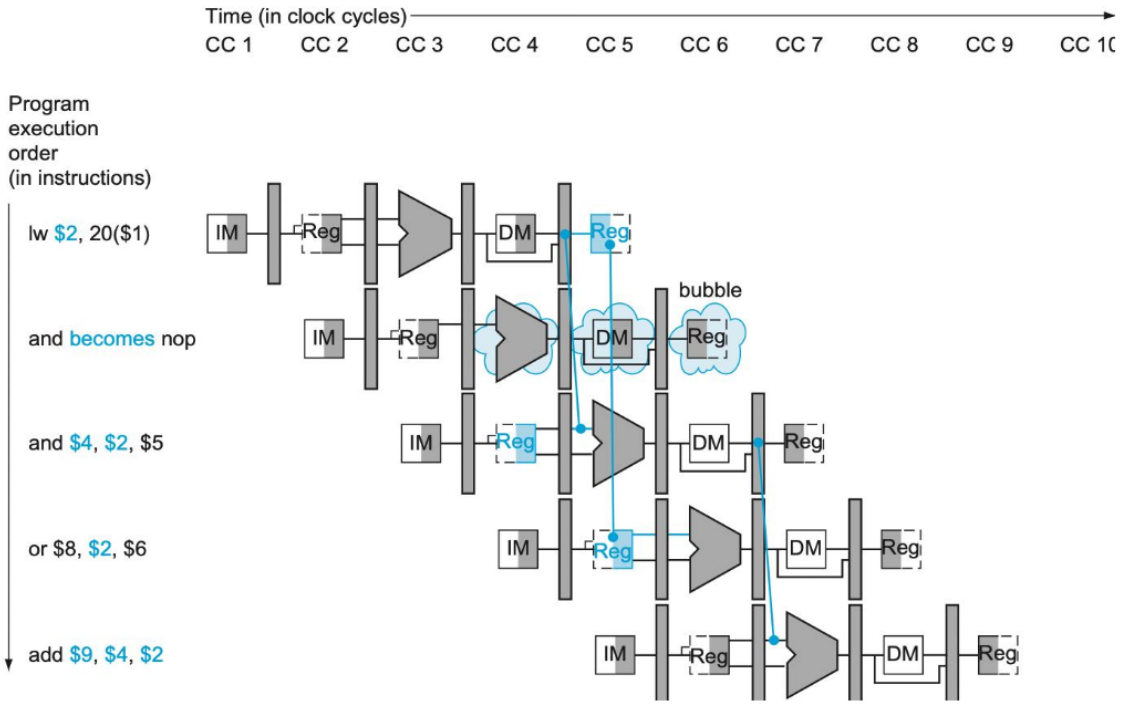
Data Hazard - Stall Scenarios



- There is a case where data forwarding cannot resolve the data hazard (데이터 전달로 데이터 위험을 해결할 수 없는 경우가 있다.)
  - Destination register of lw instruction (lw 명령의 대상 레지스터)
  - Next instruction attempts read the register (다음 명령어 시도는 레지스터를 읽는다)
  - $2 register in the example (예제에서 $2 레지스터)
- lw $2, 20($1)
  - The data read from the data memory in MEM stage is stored in MEM/WB register (MEM 단계에서 데이터 메모리에서 읽은 데이터는 MEM/WB 레지스터에 저장된다.)
- and $4, $2, $5
  - The data read from the data memory is needed at least in EX stage (데이터 메모리에서 읽은 데이터는 적어도 EX 단계에서 필요하다.)

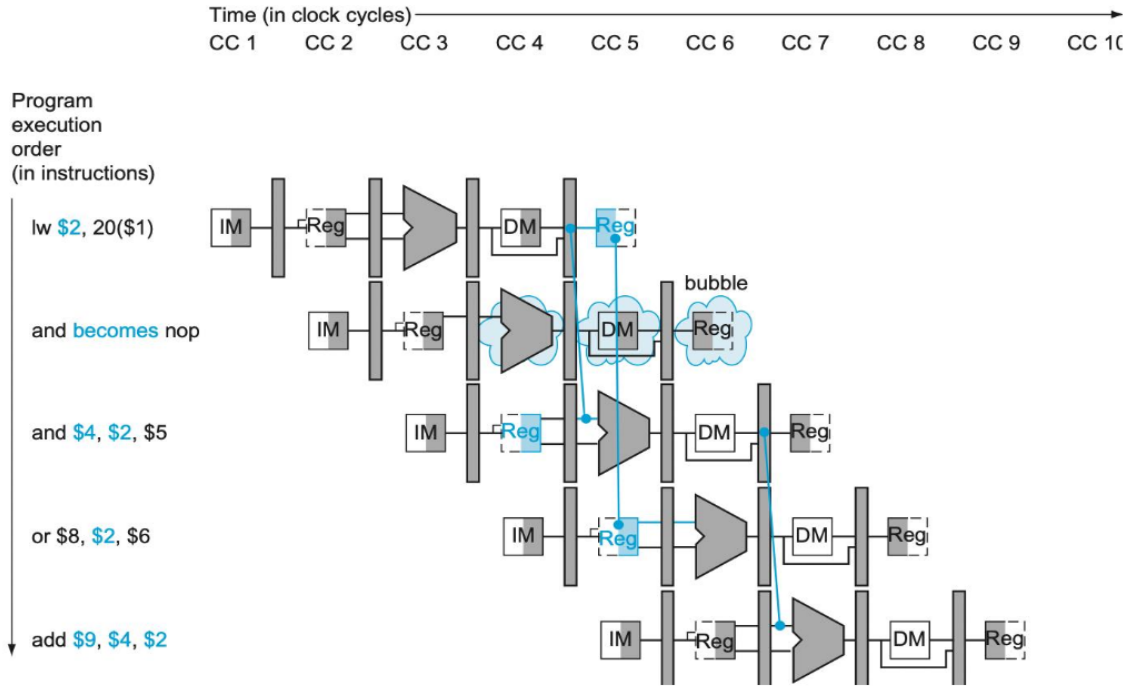
- The data can be forwarded from MEM/WB register to Rs port of the ALU; but, it is too late (데이터는 MEM/WB 레지스터에서 ALU의 Rs 포트로 전달될 수 있디만, 너무 늦는다.)
    - How can wd handle this problem? (wd는 이 문제를 어떻게 처리할 수 있을까?)

Stall (solution) - Stall Scenarios (1)



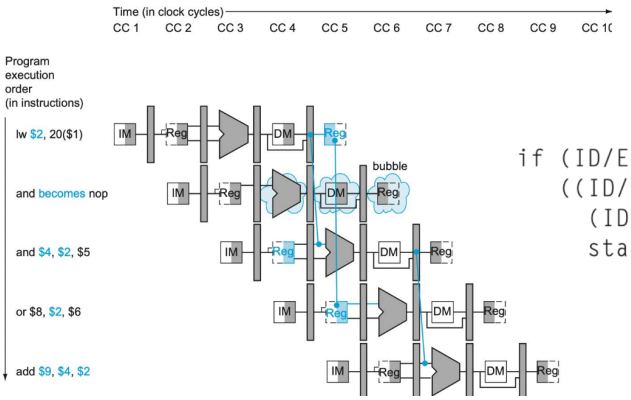
- The only way to resolve this is to stall the pipeline (wasting time by doing nothing) (이 문제를 해결할 수 있는 유인할 방법은 파이프라인을 멈추는 것이다. (아무것도 하지 않음으로써 시간 낭비))
    - (1) We prevent and (in ID stage) & or (in IF stage) from making progress for one CC (and (ID 단계에서) 및 or (IF 단계에서)가 하나의 CC에 대해 진행되지 않도록 한다.)
    - (2) We insert nop (an instruction that does nothing) into the pipeline and let it make progress (파이프라인에 nop(아무것도 하지 않는 명령어)를 삽입하고 진행을 허용한다.)
- After waiting for a CC, data forwarding (from MEM/WB to Rs of ALU) can resolve this (CC를 기다린 후, 데이터 forwarding(MEM/WB에서 ALU의 Rs로)을 통해 이 문제를 해결할 수 있다.)

Stall (solution) - Stall Scenarios (2)



- (1) We prevent and (in ID stage) & or (in IF stage) from making progress for one CC (하나의 CC에 대해 and (ID 단계에서) 및 or (IF 단계에서)가 진행되지 않도록 방지한다.)
    - If we prevent IF/ID pipeline register from changing, and will enter the ID stage again (IF/ID 파이프라인 레지스터가 변경되는 것을 방지하고 다시 ID 단계로 진입하는 경우)
    - If we prevent PC from changing, or will enter the IF stage again (PC가 변경되는 것을 방지하거나 IF 단계로 다시 진입하는 경우)
- (2) We insert nop into the pipeline (see bubble) and let it make progress (파이프라인에 nop를 삽입하고 (버블 참조) 진행하도록 한다.)
    - If we set the nine control signals to “0”, registers or memory are not written (9개의 제어 신호를 “0”으로 설정하면 레지스터나 메모리가 기록되지 않는다.)

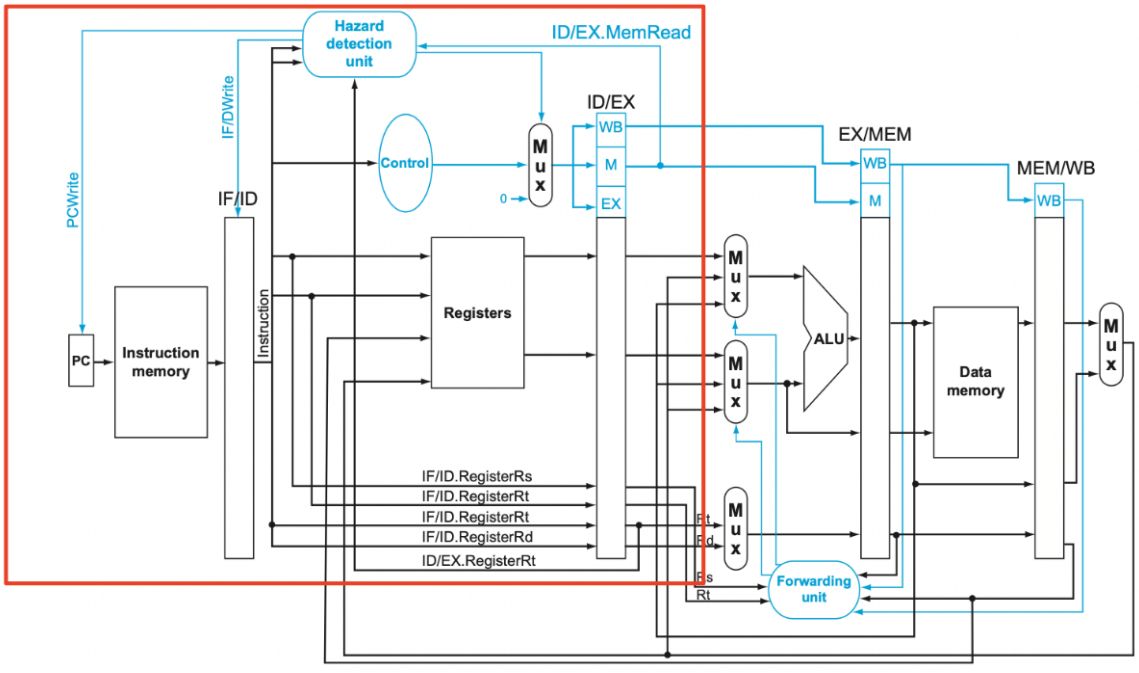
Data Hazard Detection for Stall




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

- (1) Is it a load instruction? (lw $2, 20($1))
  - ID/EX.MemRead == 1?
- (2) Is destination of the load instruction equal to a source of the next instruction? (load 명령의 목적지가 다음 명령의 소스와 같은가?) ($2)
  - ID/EX.RegisterRt == IF/ID.RegisterRs ?
  - ID/EX.RegisterRt == IF/ID.RegsiterRt ?

Stall - Implementation



- To emphasize hazard-related components, branch/sign-extension units are missing (위험 관련 구성 요소를 강조하기 위해 분기/부호 확장 장치가 누락되었다.)

- Hazard detection unit (위험 감지 장치)
  - Evaluates the conditions in the previous slide (이전 슬라이드의 조건을 평가한다.) (ID/EX.MemRead, ID/EX.RegisterRt, IF/ID.RegisterRs/Rt)
  - (If it needs a stall) prevents PC & IF/ID register from changing ((stall이 필요한 경우) PC 및 IF/ID 레지스터의 변경을 방지한다.) (PCWrite=0 , IF/IDWrite=0)
  - (If it needs a stall) selects “0” to all 9 control signals ((stall이 필요한 경우) 9개의 제어 신호 모두에 “0”을 선택한다.)