
Computer Architecture

Chapter 4C. Dealing with pipeline issues in the Processor

Hyuk-Jun Lee, PhD

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea

Email: hyukjunl@sogang.ac.kr



Sogang University

Data Hazards in ALU Instructions

- Consider this sequence:

sub \$2, \$1, \$3

and \$12, \$2, \$5

or \$13, \$6, \$2

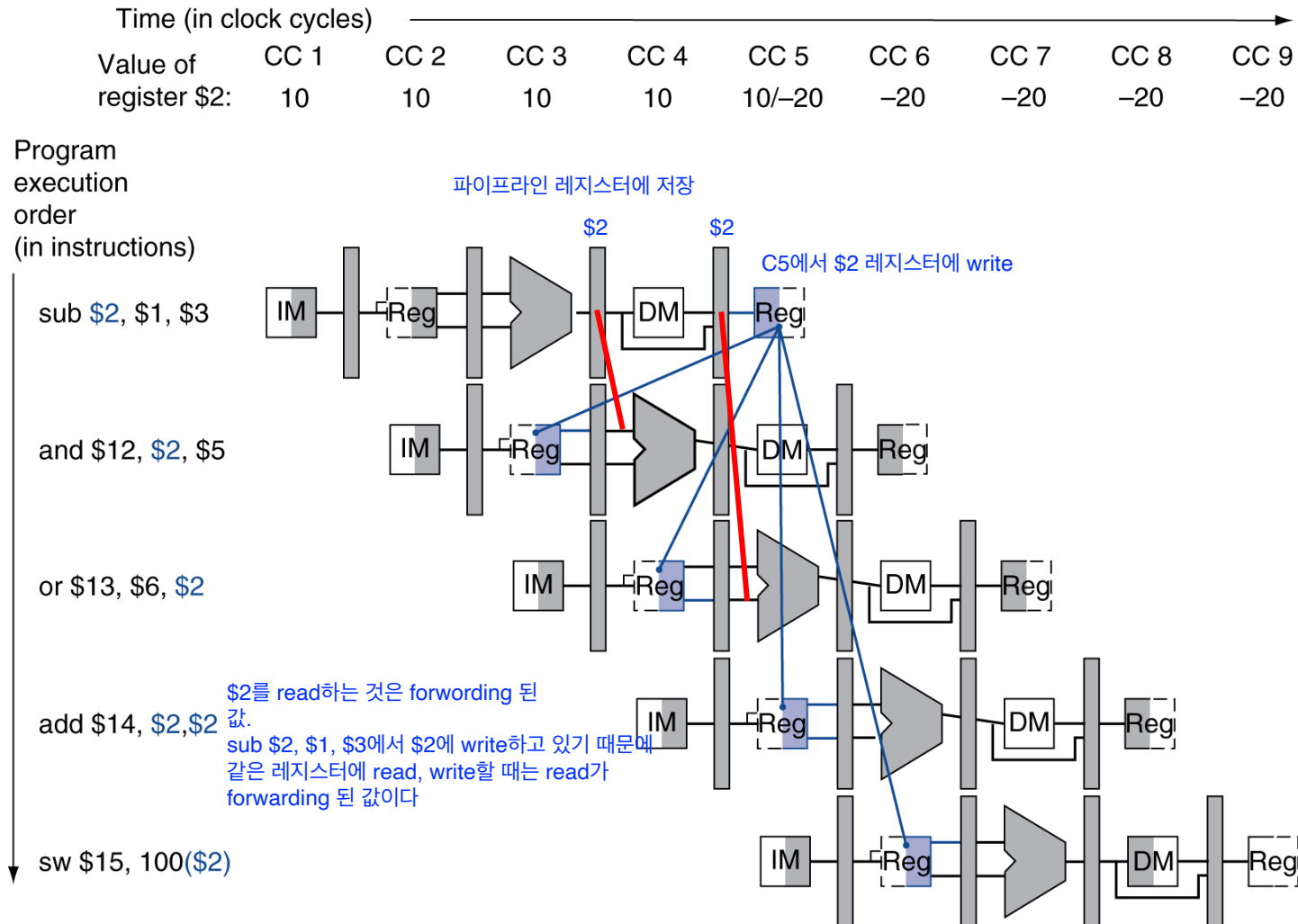
add \$14, \$2, \$2

sw \$15, 100(\$2)

- We can resolve hazards with forwarding
 - How do we detect when to forward?



Dependencies & Forwarding



Detecting the Need to Forward

- Pass register numbers along pipeline
 - e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
 - ID/EX.RegisterRs, ID/EX.RegisterRt
- Data hazards when
 - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
 - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
 - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
 - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Fwd from
EX/MEM
pipeline reg

Fwd from
MEM/WB
pipeline reg

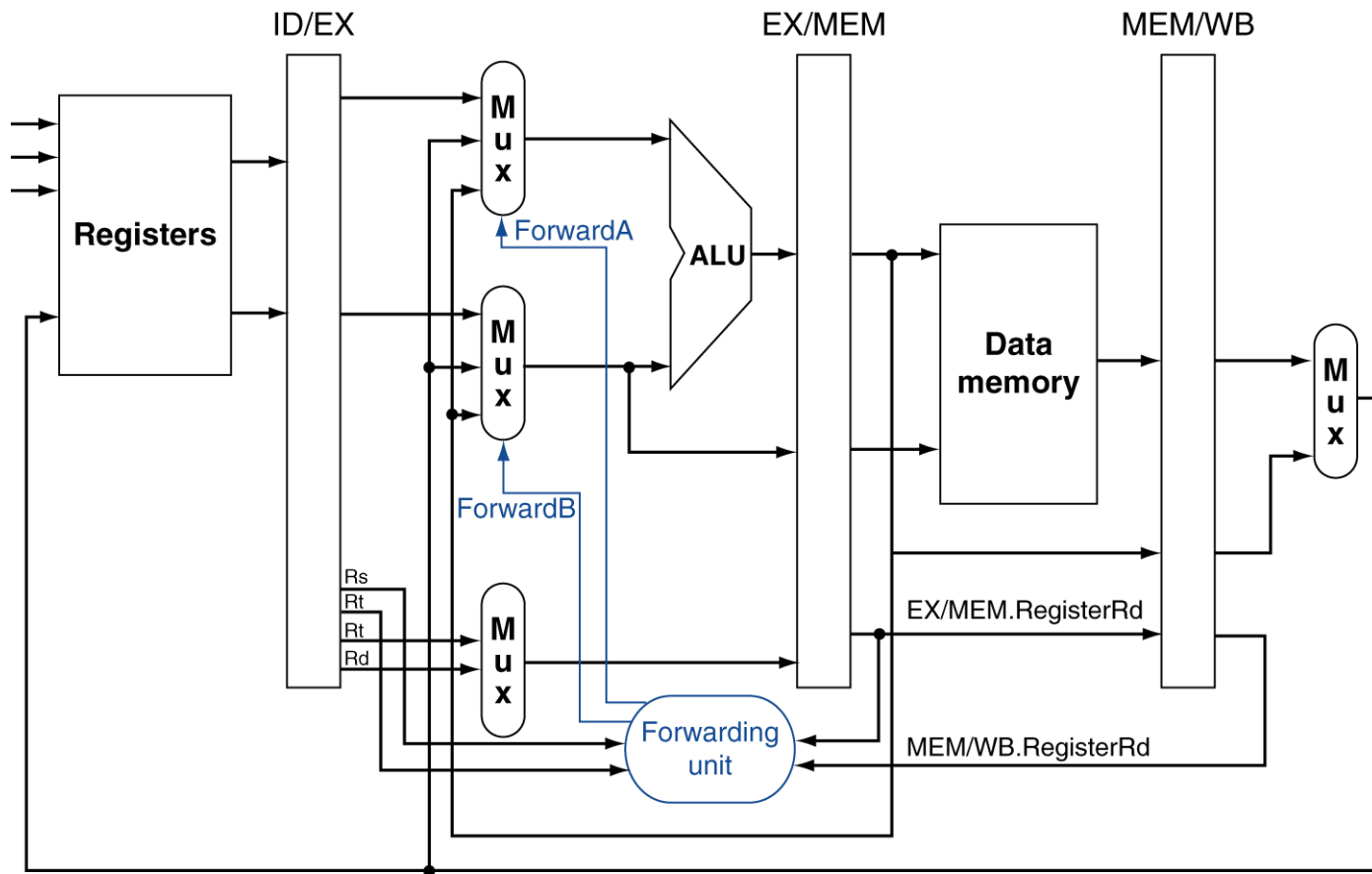


Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
 - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not \$zero
 - EX/MEM.RegisterRd \neq 0,
MEM/WB.RegisterRd \neq 0



Forwarding Paths



b. With forwarding



Forwarding Conditions

- EX hazard
 - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
ForwardA = 10
 - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
ForwardB = 10
- MEM hazard
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
ForwardA = 01
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
ForwardB = 01



Double Data Hazard

- Consider the sequence:
 add \$1, \$1, \$2
 add \$1, \$1, \$3
 add \$1, \$1, \$4
 Diagram: Arrows point from the first '\$1' of the second instruction to the second '\$1' of the first instruction, and from the first '\$1' of the third instruction to the second '\$1' of the second instruction.
- Both hazards occur
 - Want to use the most recent
- Revise MEM hazard condition
 - Only fwd if EX hazard condition isn't true



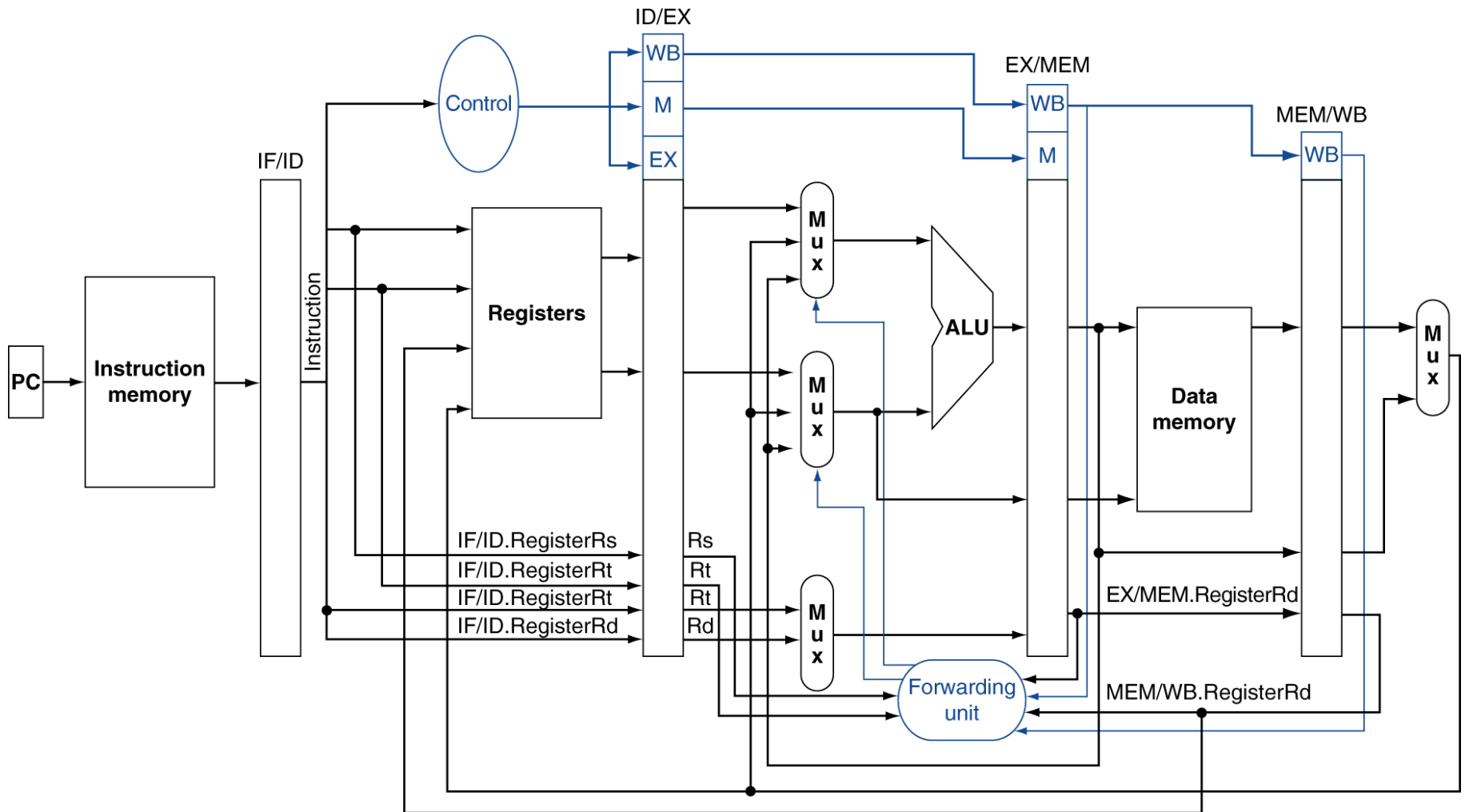
Revised Forwarding Condition

ex hazard가 더 최근의 결과를 만들기 때문에 우선권을 줘야함

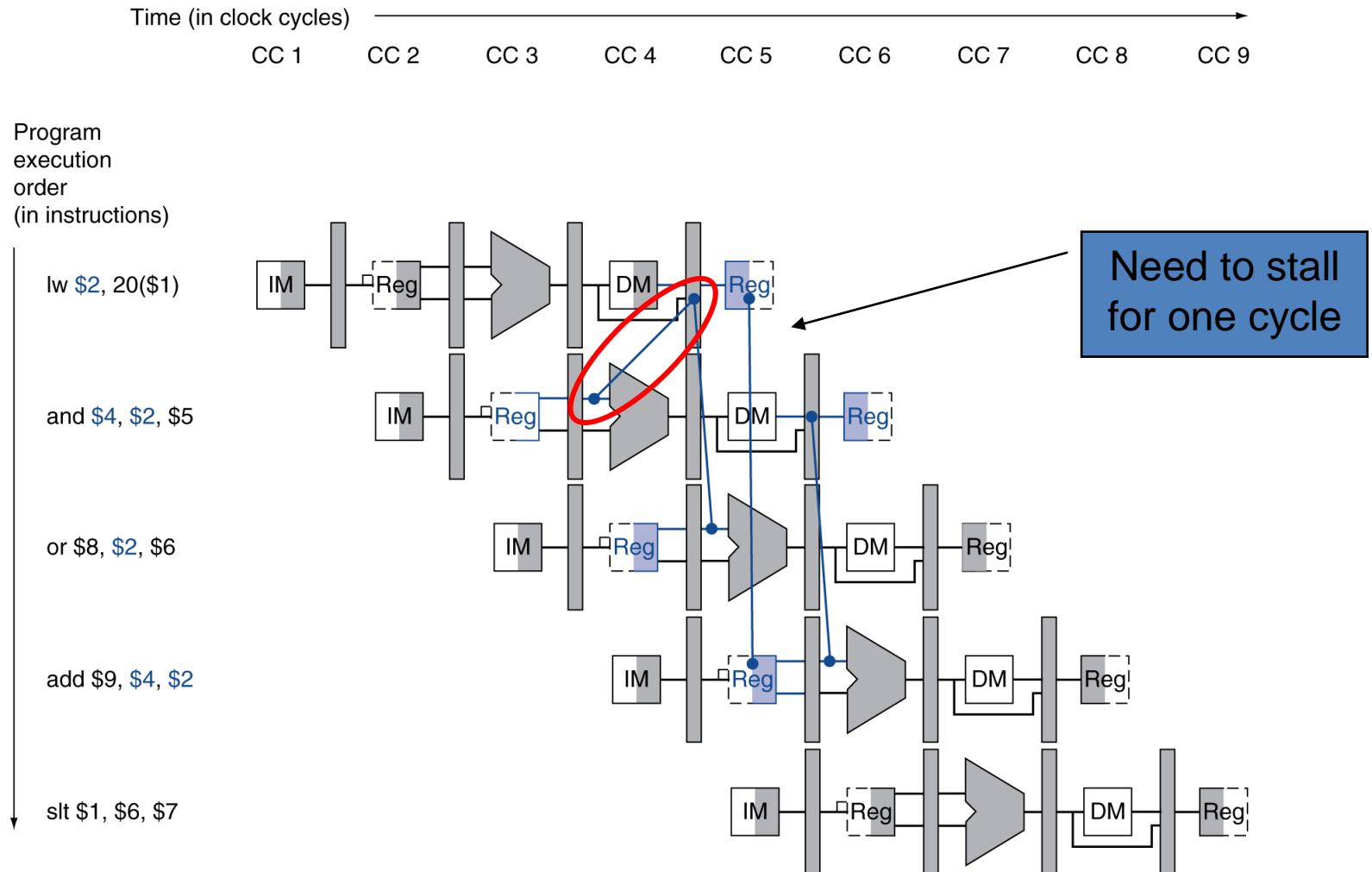
- MEM hazard
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
ForwardA = 01
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
ForwardB = 01



Datapath with Forwarding



Load-Use Data Hazard



Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
 - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
 - ID/EX.MemRead and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

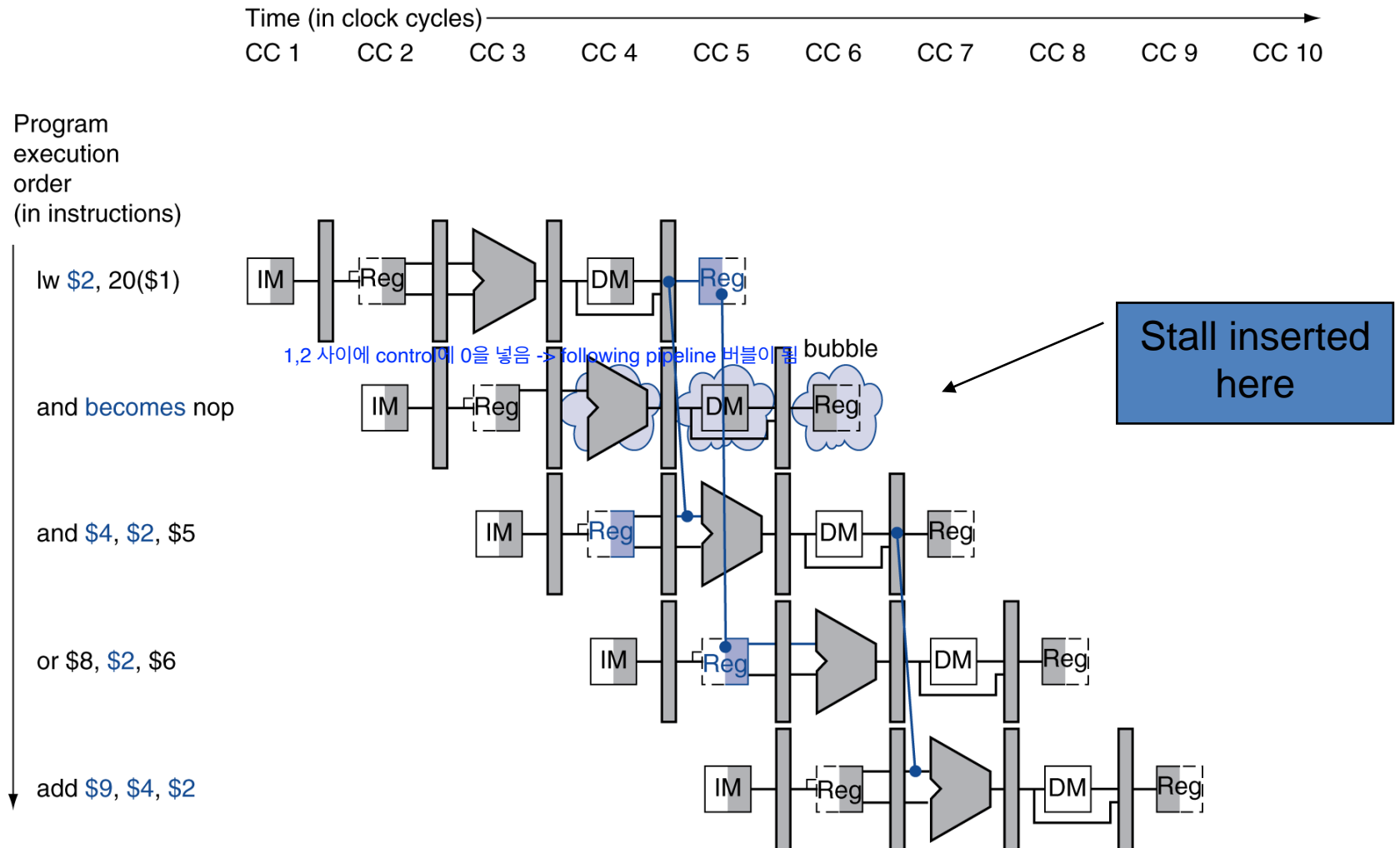


How to Stall the Pipeline

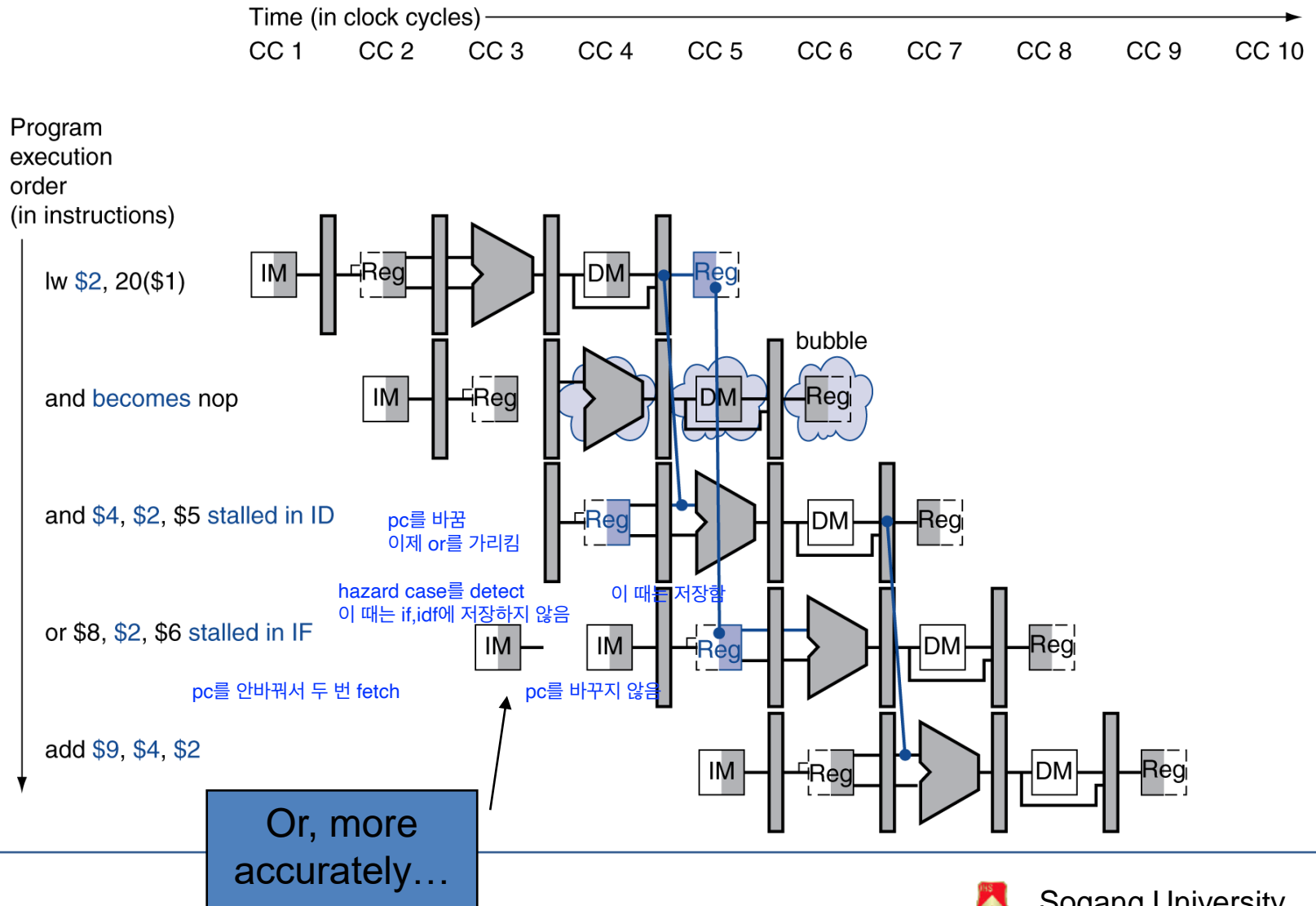
- Force control values in ID/EX register to 0
 - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
 - Using instruction is decoded again
 - Following instruction is fetched again
 - 1-cycle stall allows MEM to read data for $1w$
 - Can subsequently forward to EX stage



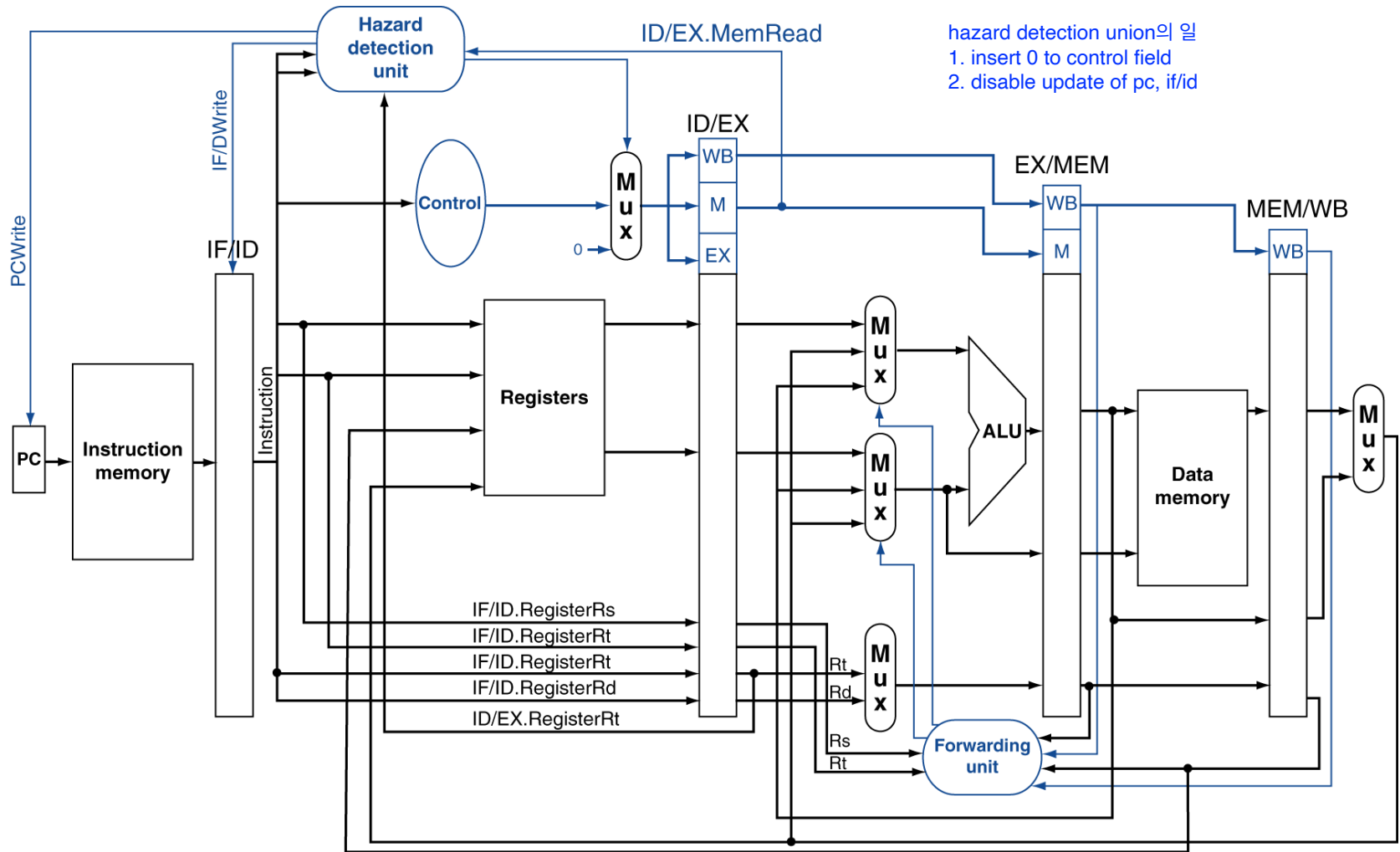
Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline



Datapath with Hazard Detection



Stalls and Performance

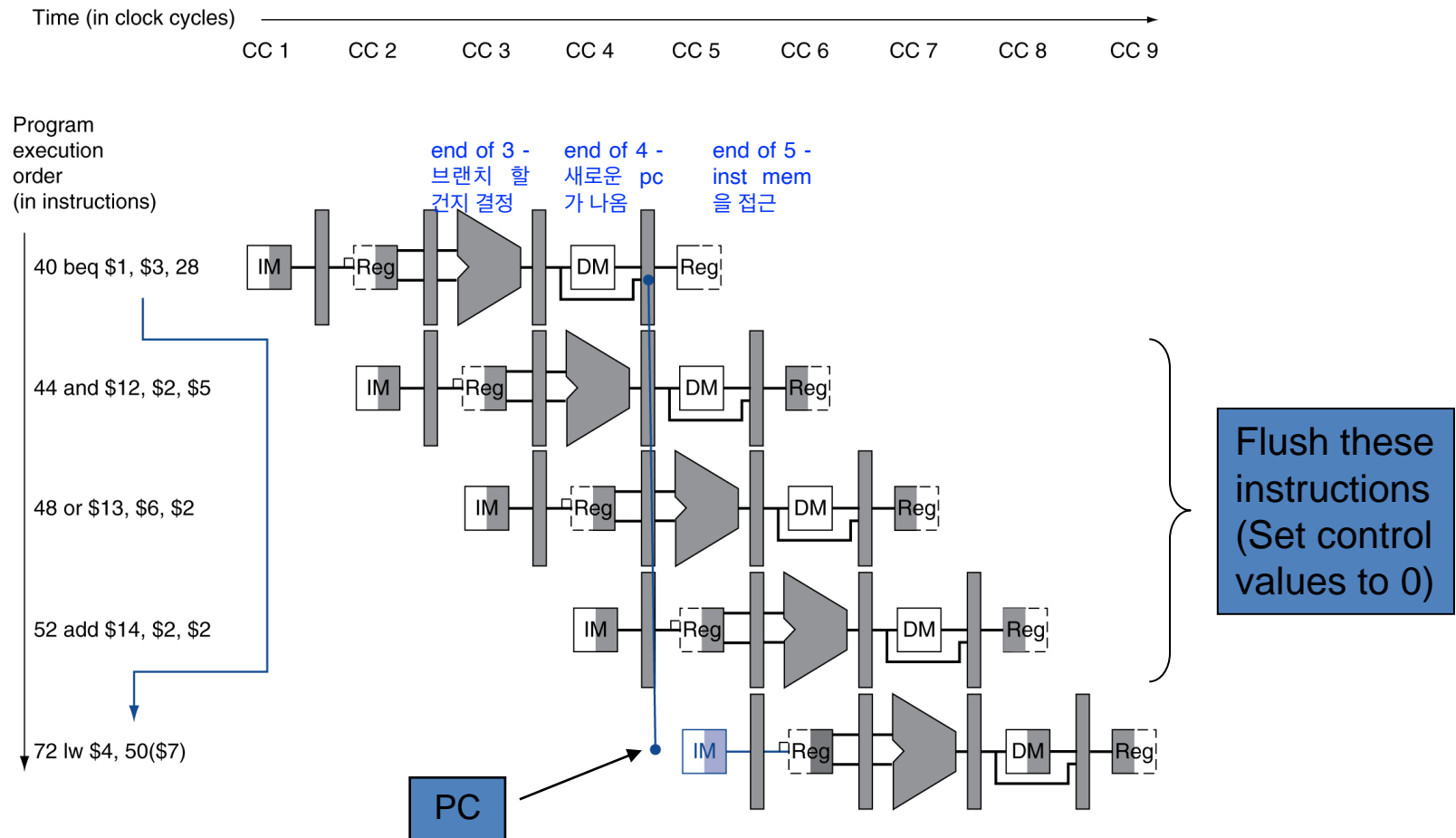
The BIG Picture

- Stalls reduce performance
 - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
 - Requires knowledge of the pipeline structure



Branch Hazards

- If branch outcome determined in MEM



Reducing Branch Delay

- Move hardware to determine outcome to ID stage
 - Target address adder
 - Register comparator
- Example: branch taken

브랜치 pc 계산을
ID stage 에 하면
1 사이클 더 빨리 할 수 있음

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

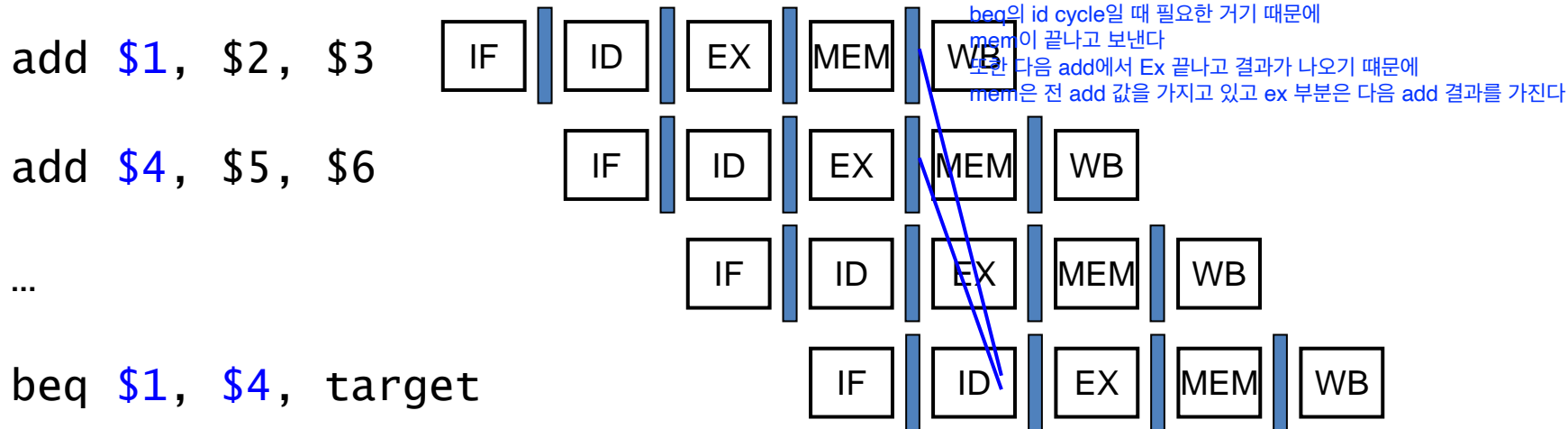


Data Hazards for Branches

- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

branch pc를 id stage에서 계산할 경우

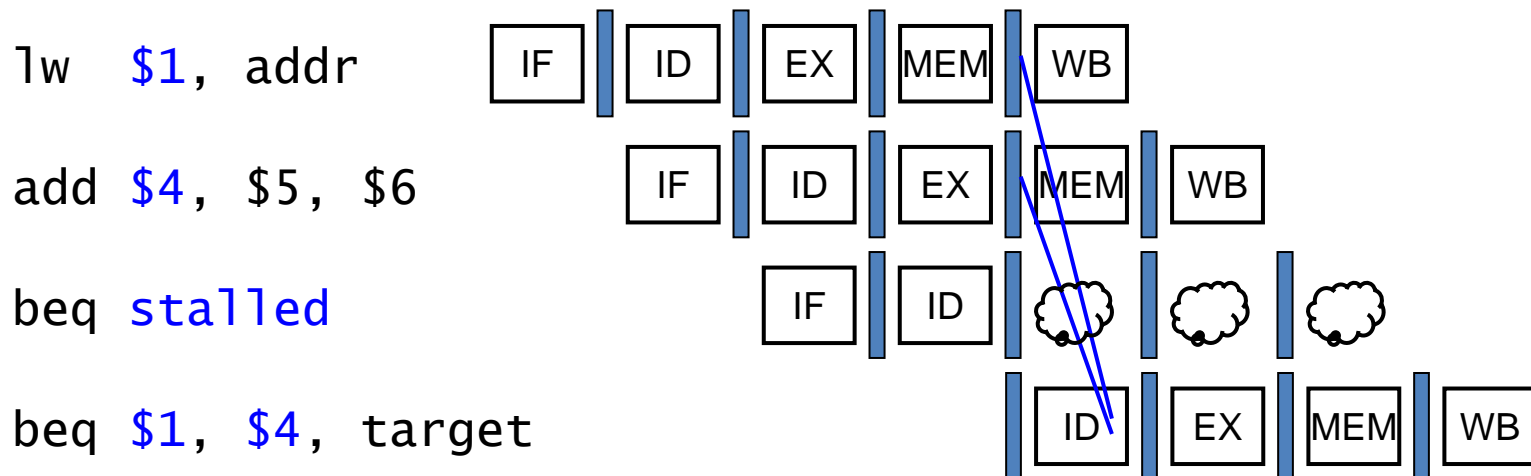
id stage 끝날 때 branch pc가 필요하기 때문에 ID stage로 forwarding하고 있음



- Can resolve using forwarding

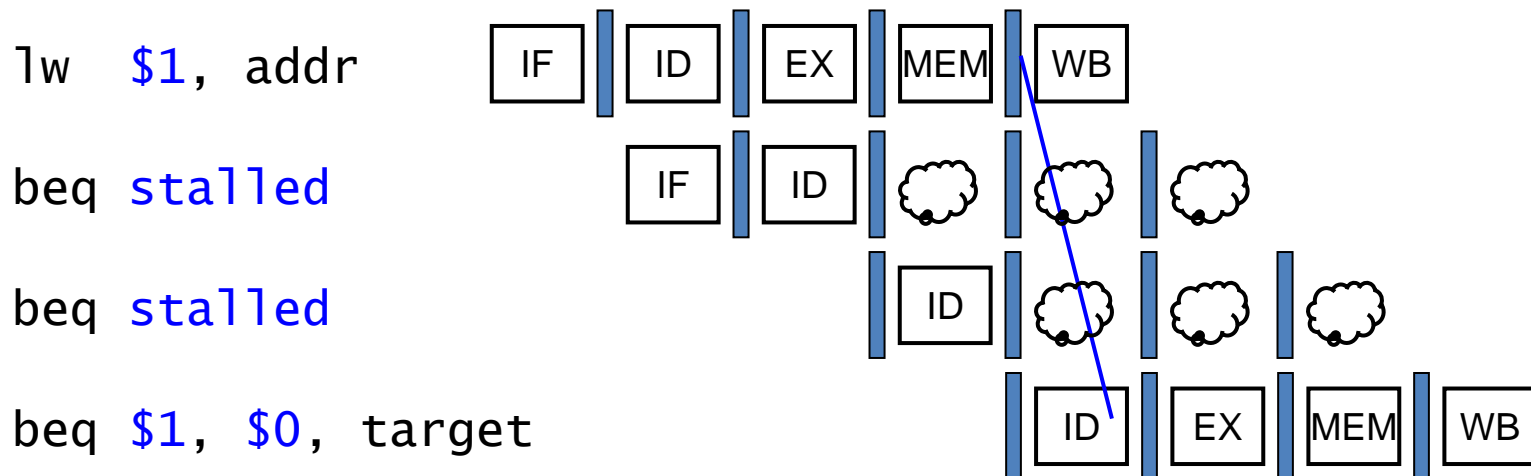
Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
 - Need 1 stall cycle



Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
 - Need 2 stall cycles



Fallacies

- Pipelining is easy (!)
 - The basic idea is easy
 - The devil is in the details
 - e.g., detecting data hazards
- Pipelining is independent of technology
 - So why haven't we always done pipelining?
 - More transistors make more advanced techniques feasible
 - Pipeline-related ISA design needs to take account of technology trends
 - e.g., predicated instructions

beq 20%
 $\text{cpi}(\text{control hazard}) = 1 + 0.2 \cdot 1$
 $\text{cpi}(\text{control} + \text{data branch hazard}) = 1.2 + 0.2 \cdot 0.3 \cdot 1$

out of total branch instruction
30% of preceeding inst insert 1 cycle bubble



Pitfalls

- Poor ISA design can make pipelining harder
 - e.g., complex instruction sets (VAX, IA-32)
 - Significant overhead to make pipelining work
 - IA-32 micro-op approach
 - e.g., complex addressing modes
 - Register update side effects, memory indirection
 - e.g., delayed branches
 - Advanced pipelines have long delay slots



Concluding Remarks

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism
 - More instructions completed per second
 - Latency for each instruction not reduced
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling (ILP)
 - Dependencies limit achievable parallelism
 - Complexity leads to the power wall

