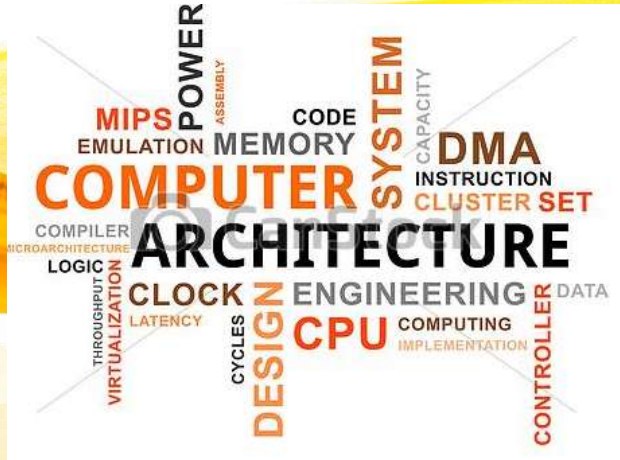
UNIT III
PROCESSOR AND PIPELINING

Fundamental concepts – Execution of a complete instruction – Multiple bus organization – Hardwired control – Micro programmed control – Pipelining: Basic concepts – **Data hazards – Instruction hazards** – Influence on Instruction sets – Data path and control consideration.



Recap the previous Class

HAZARDS

DATA HAZARDS

Need to wait for previous instruction

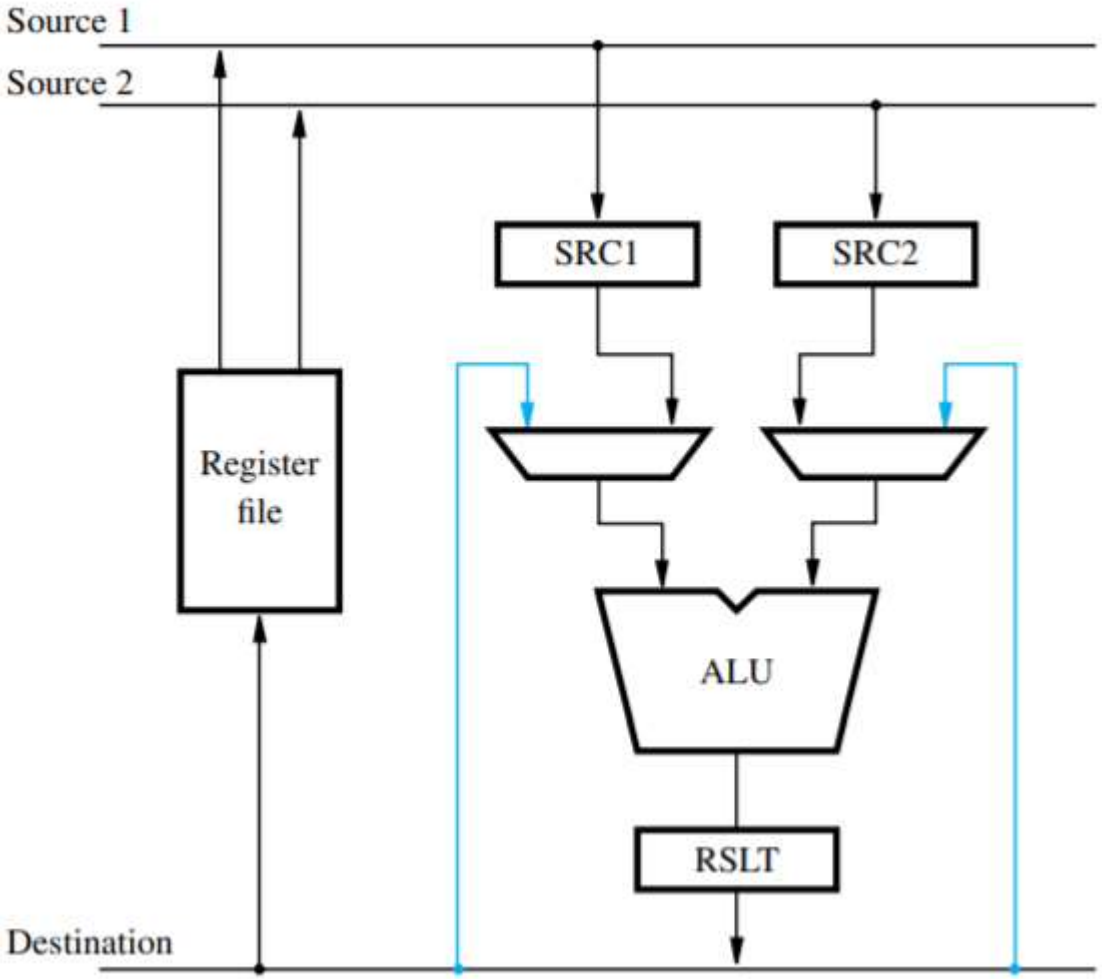
STRUCTURAL HAZARDS

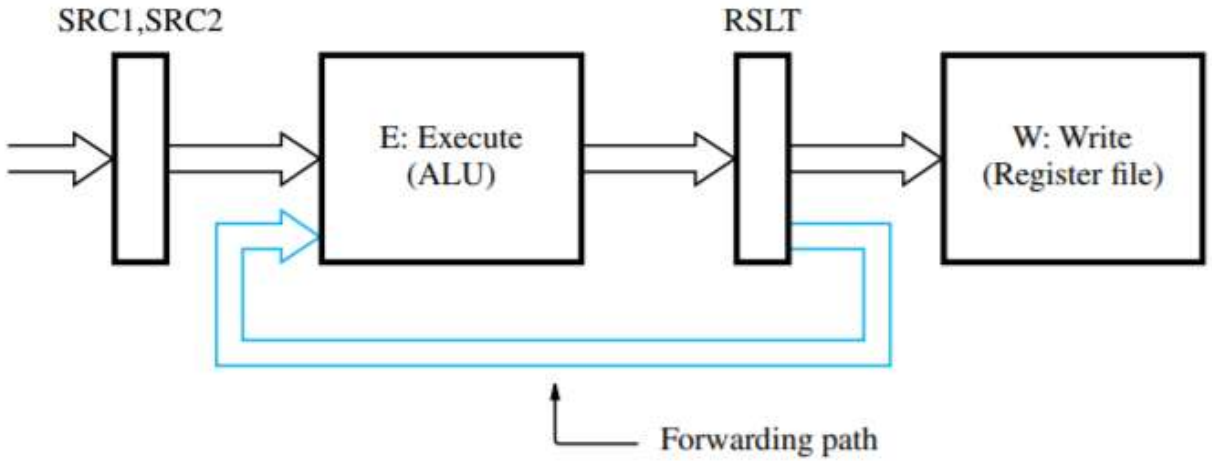
required resource is busy

CONTROL HAZARDS

Deciding on control action

Data Hazards





Data Hazards in ALU Instructions

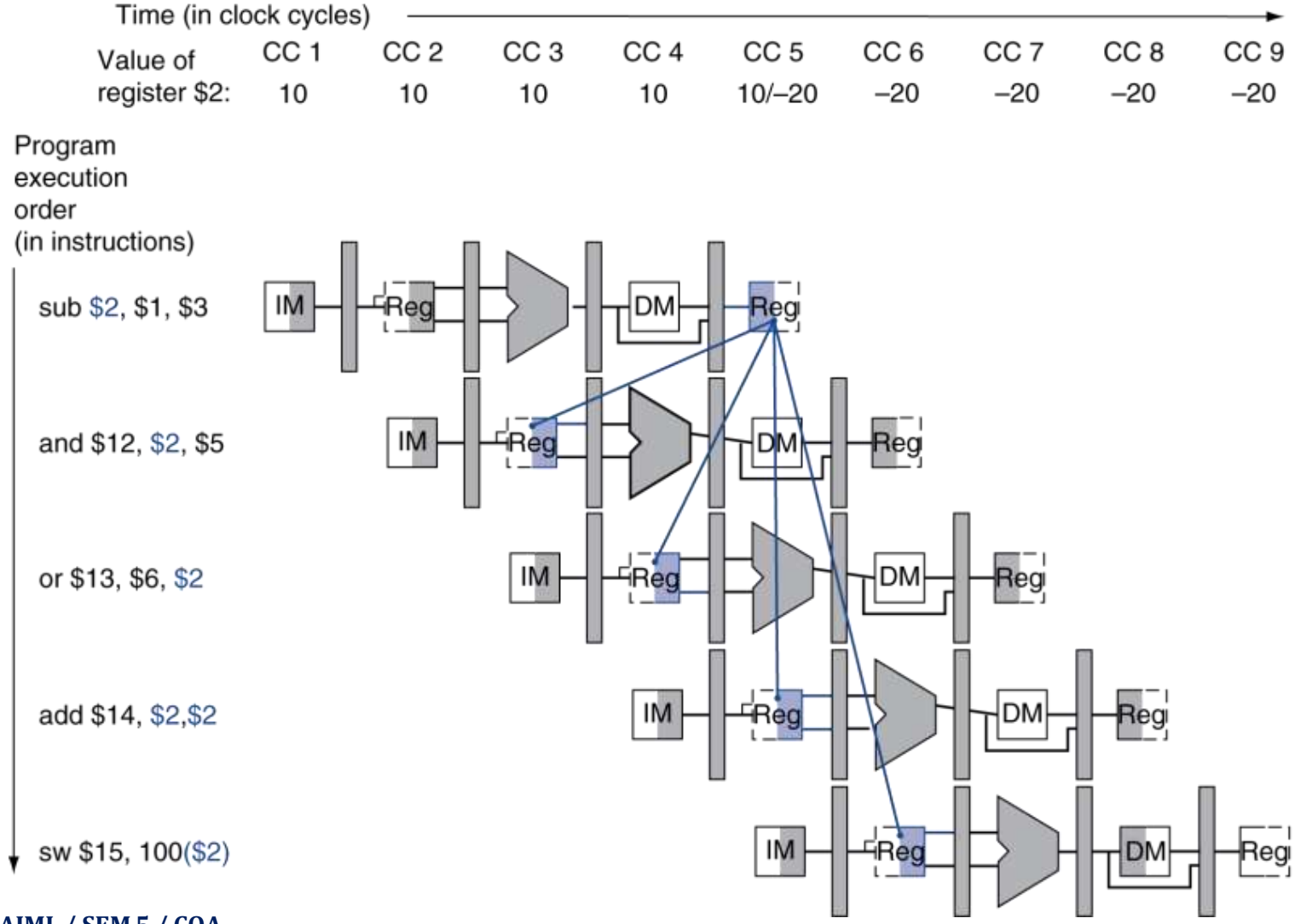
- Consider this sequence:

```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

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

} Fwd. from EX/MEM pipeline reg

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

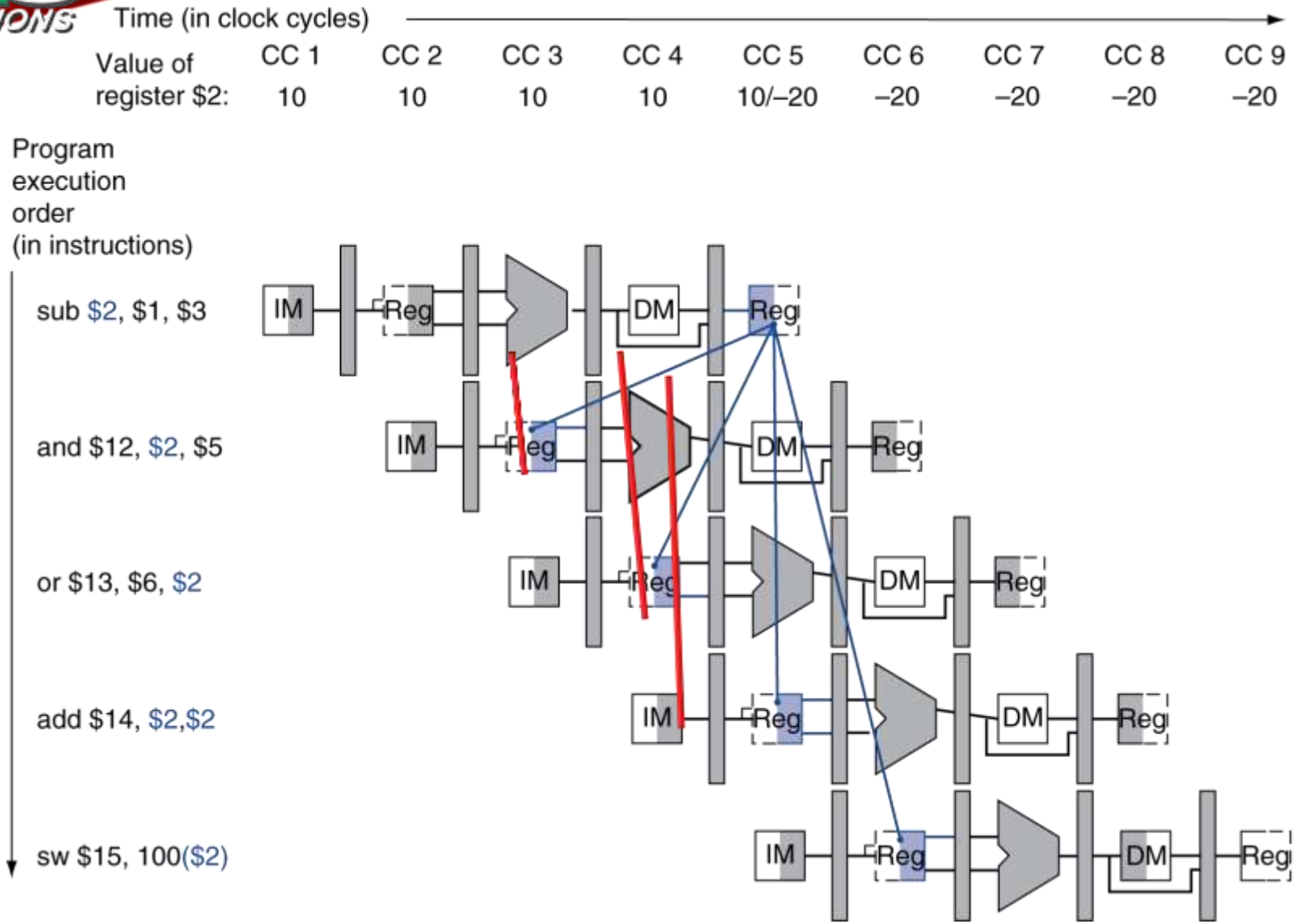
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

} Fwd. from MEM/WB pipeline reg

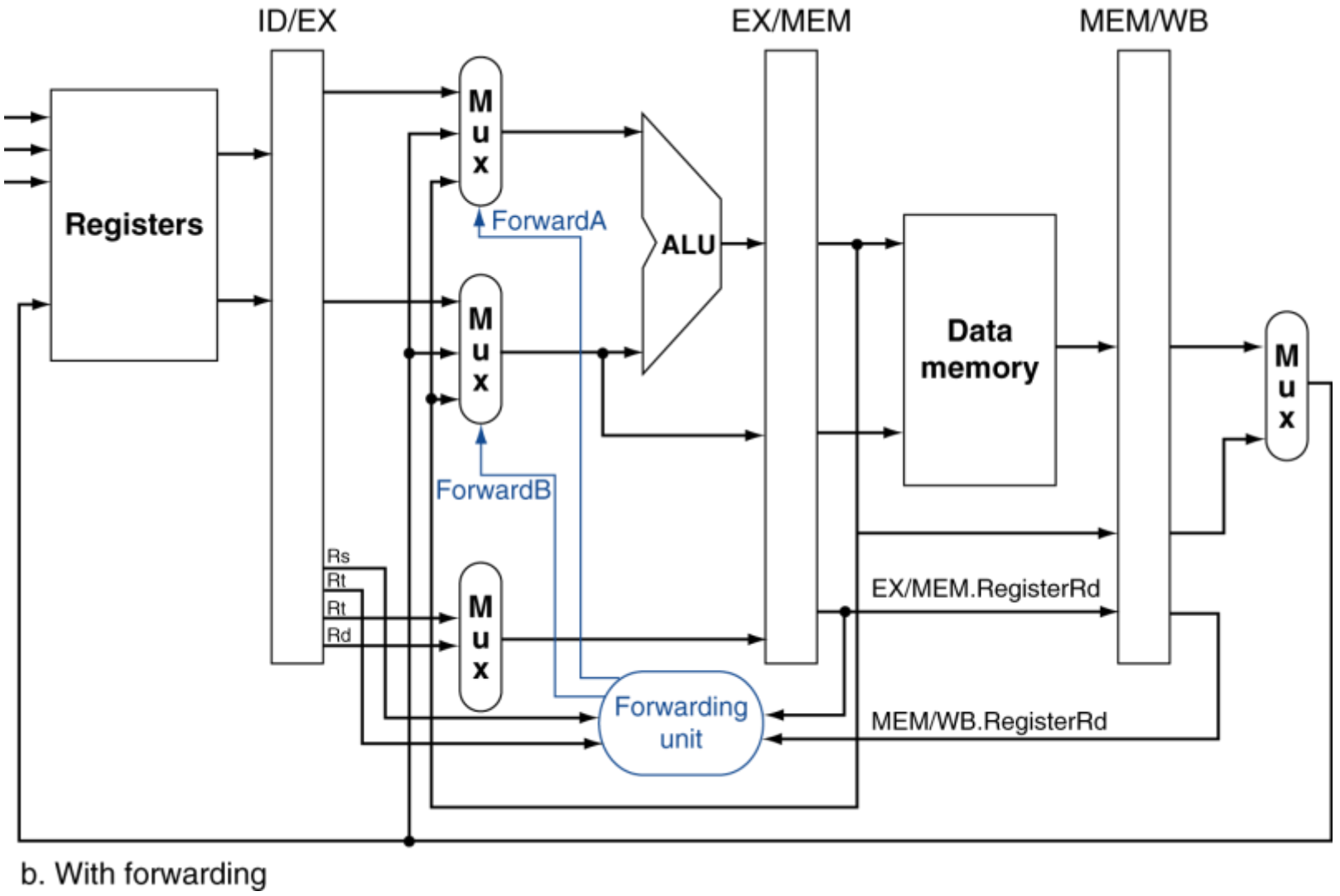
Detecting the Need to Forward

- 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 ≠ 0,

    MEM/WB.RegisterRd ≠ 0


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Value of register $2:
10
10
10
10
10/–20
–20
–20
–20
–20
Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2,$2
sw $15, 100($2)
IM
Reg
DM

Forwarding Paths



b. With forwarding

Control values

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.

Forwarding Conditions

EX hazard

- 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

MEM hazard

- 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

Double Data Hazard

Consider the sequence:

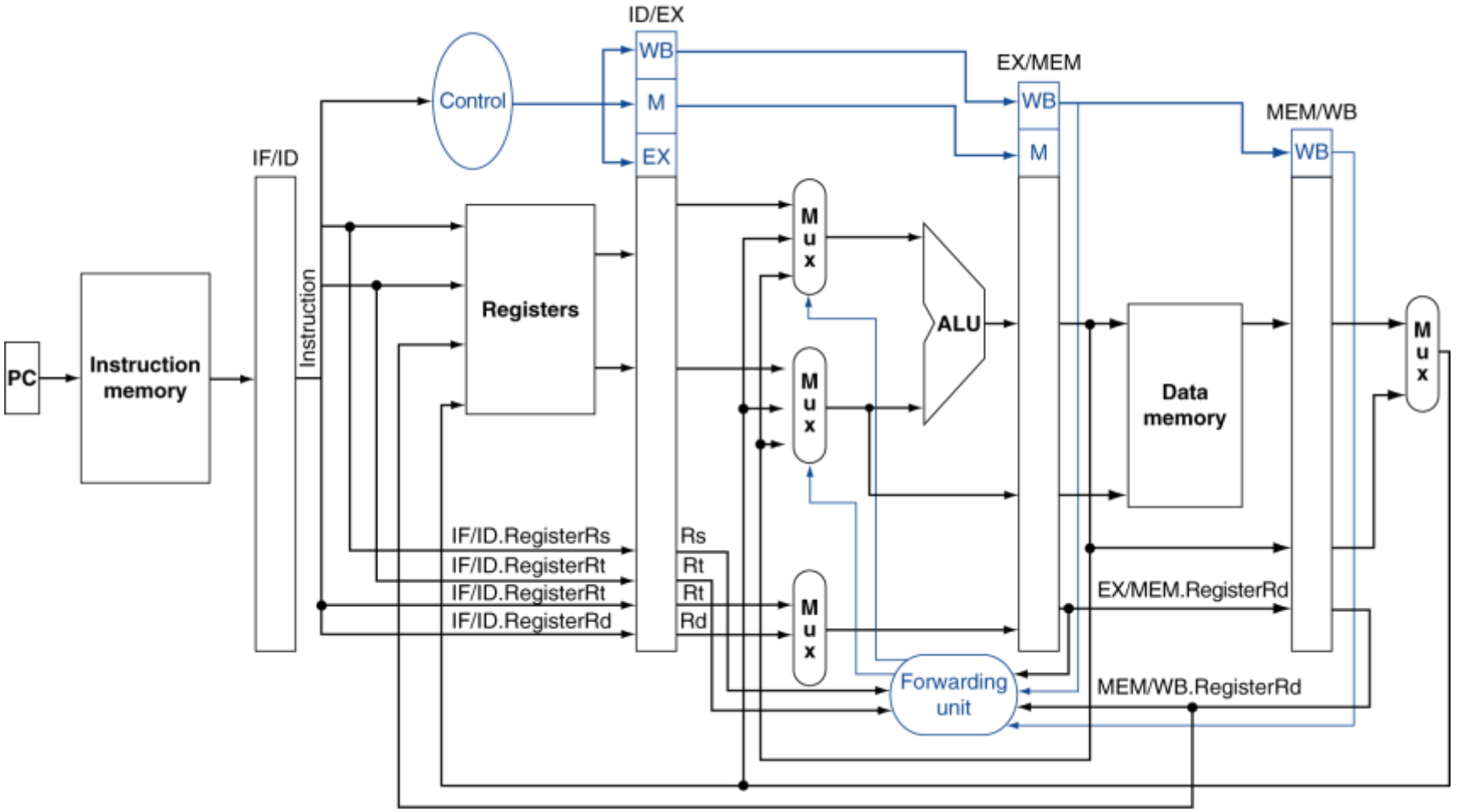
```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

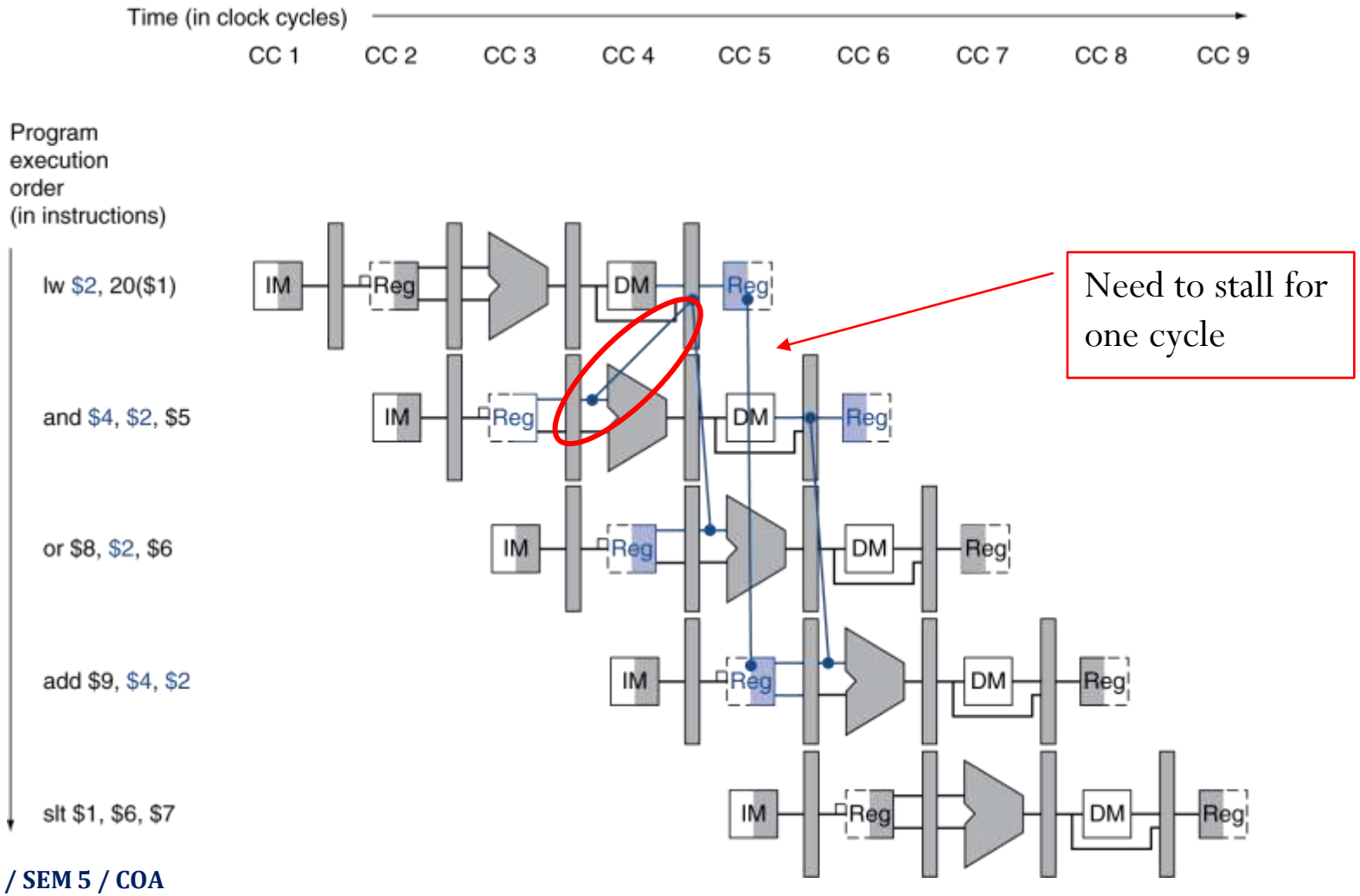
Revised Forwarding Condition

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

Datapath with Forwarding



Load-Use Data Hazard



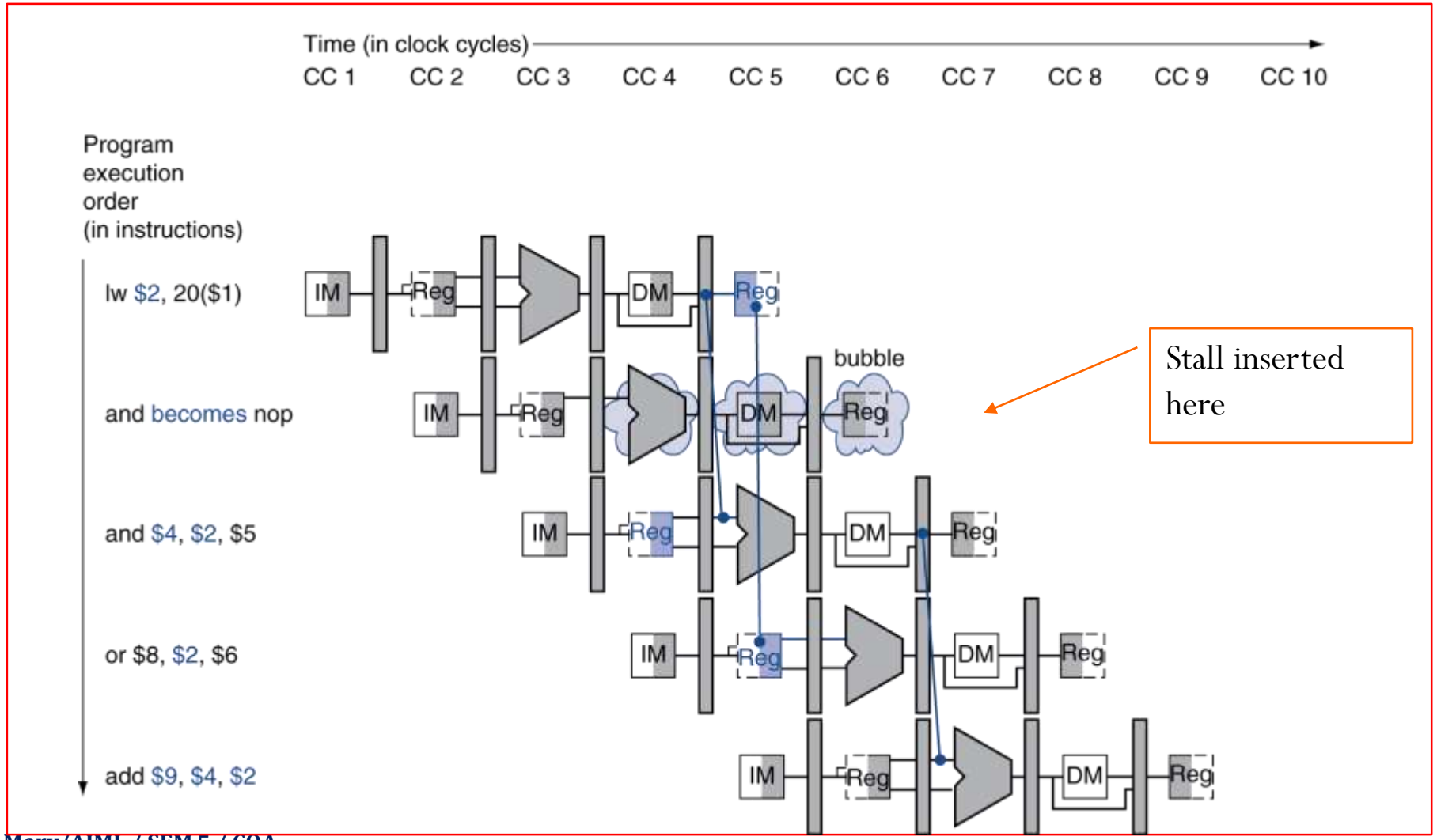
Load-Use Hazard Detection

- 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

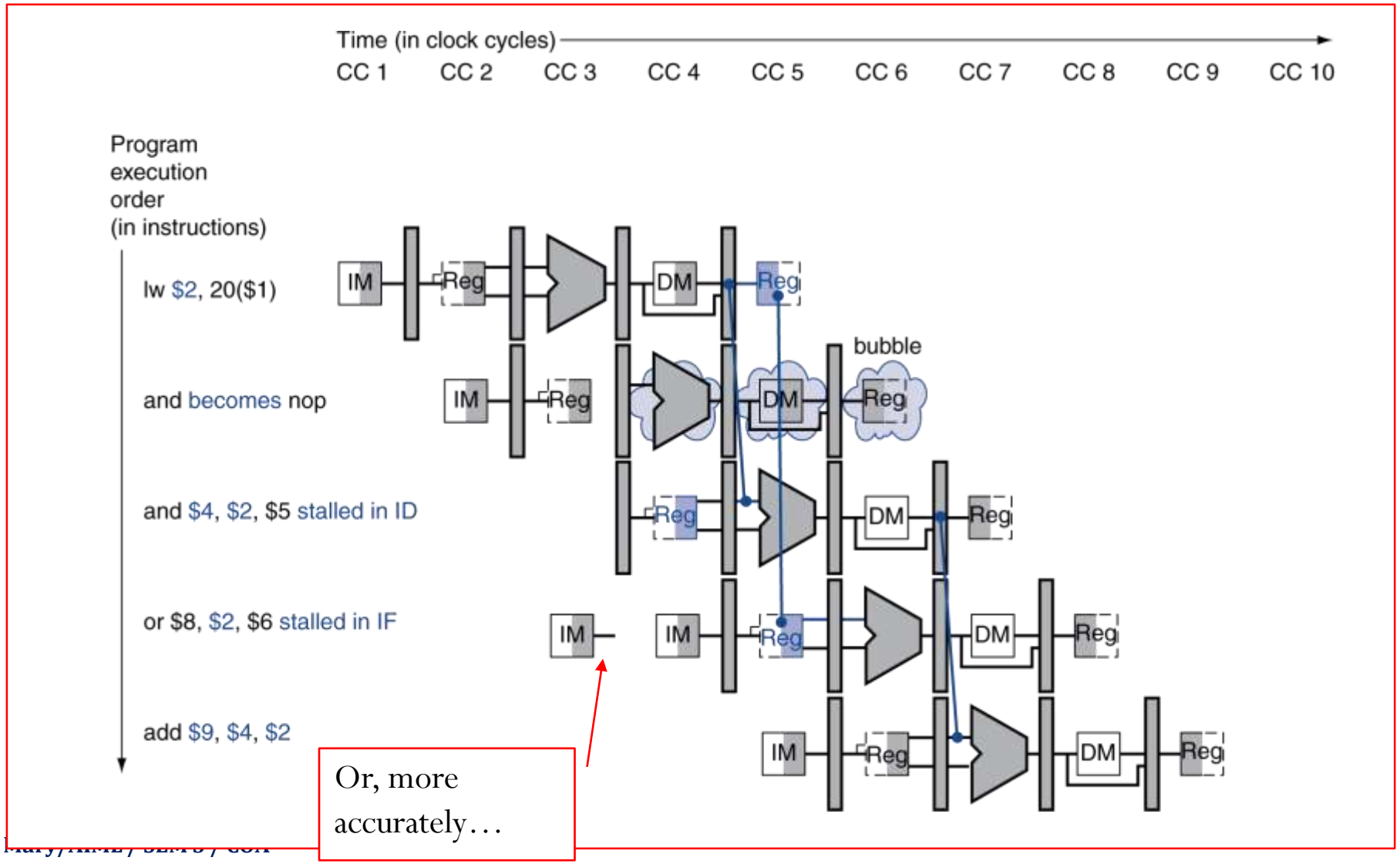
How to Stall the Pipeline

- 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 lw
    - Can subsequently forward to EX stage

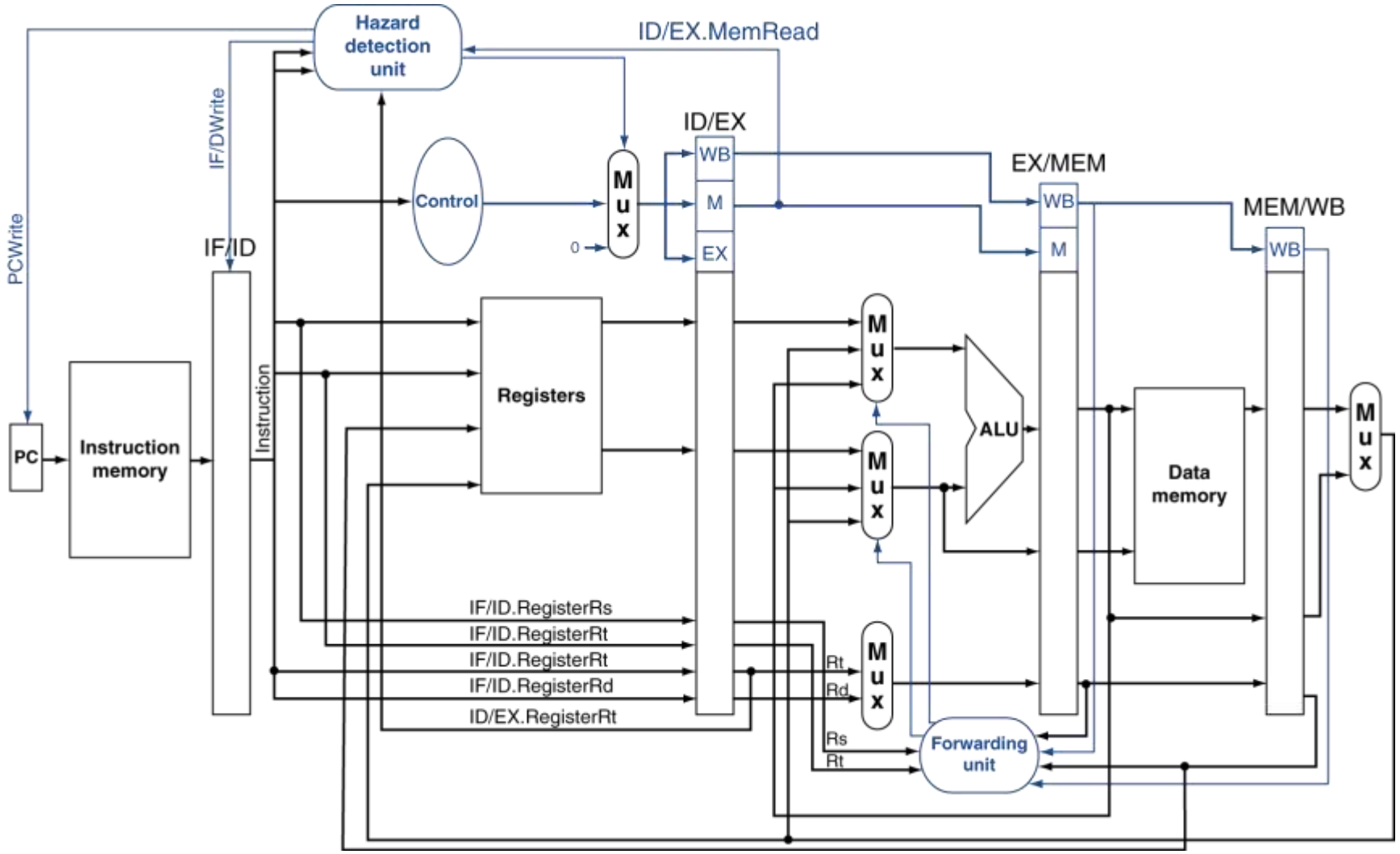
Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline

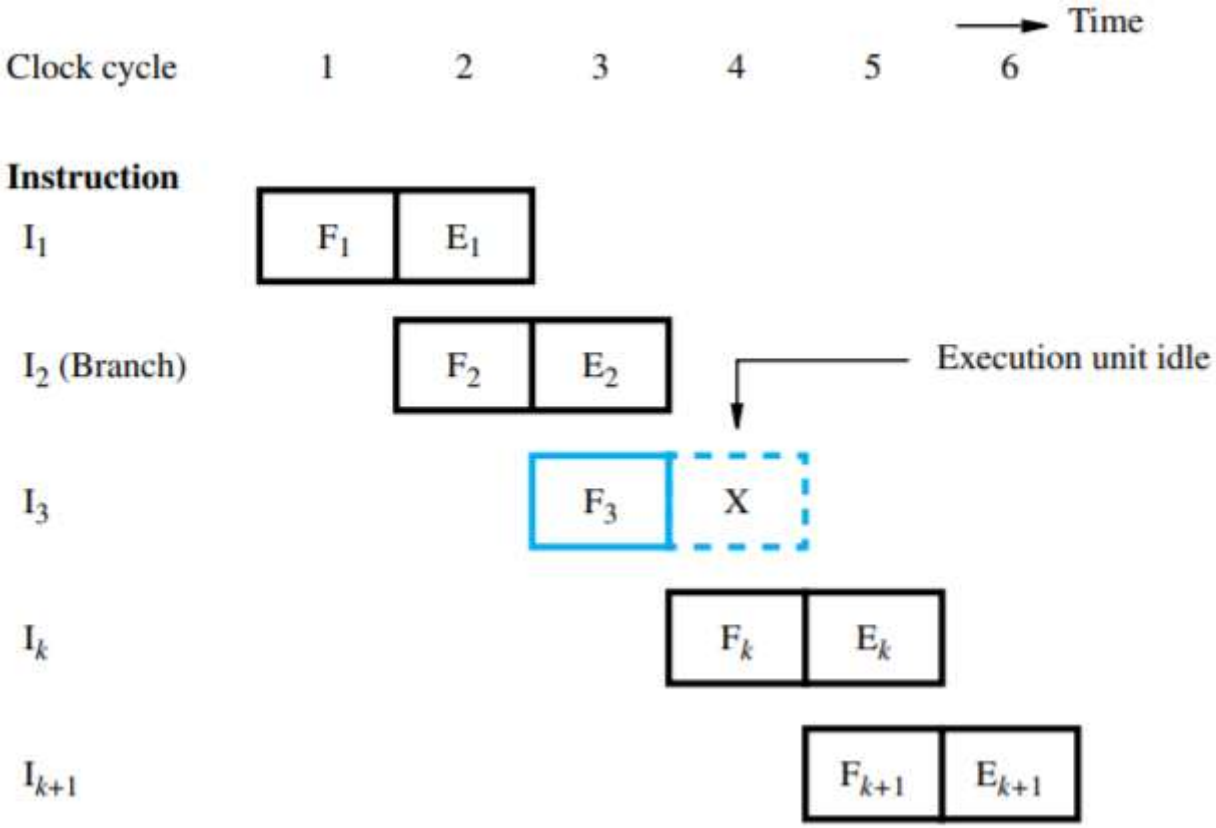


Datapath with Hazard Detection



Instructional Hazards

- Whenever the stream of instructions supplied by the instruction fetch unit is interrupted, the pipeline stalls.
- Cache miss
- Branch



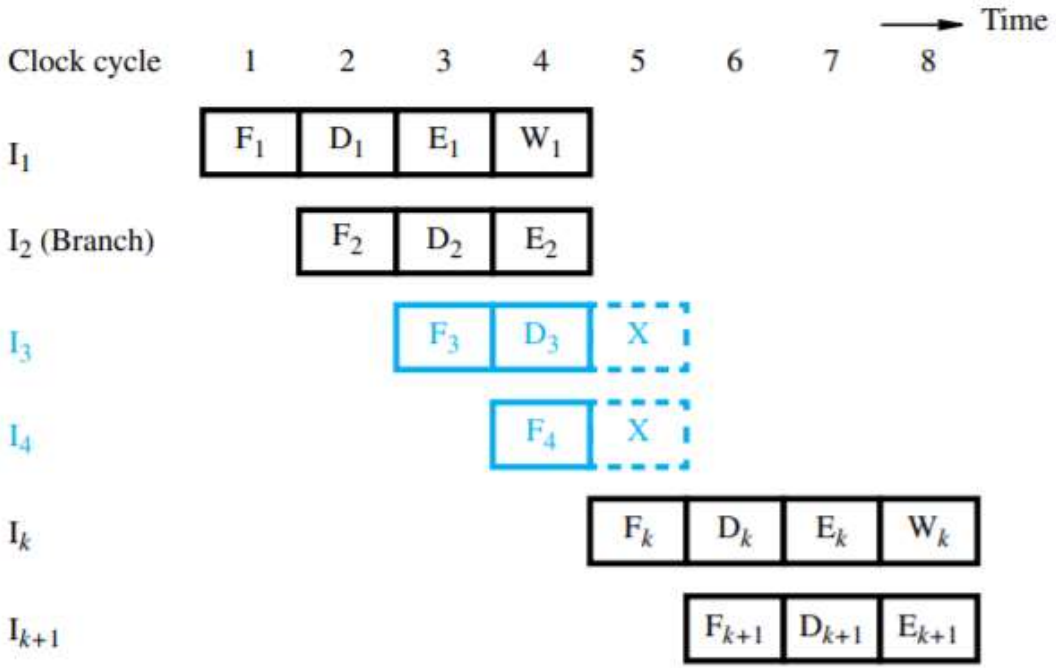
Unconditional Branch

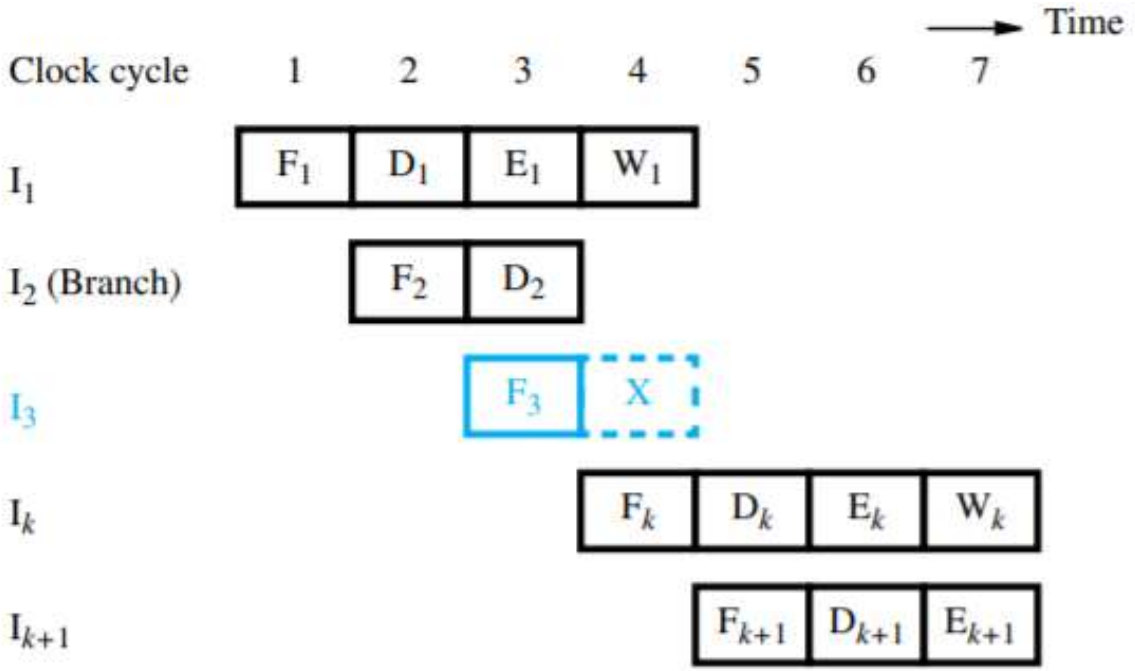
Branch instructions can alter the sequence of execution, but they must first be executed to determine whether and where to branch.

Branch instructions occur frequently. In fact, they represent about 20 percent of the dynamic instruction count of most programs.

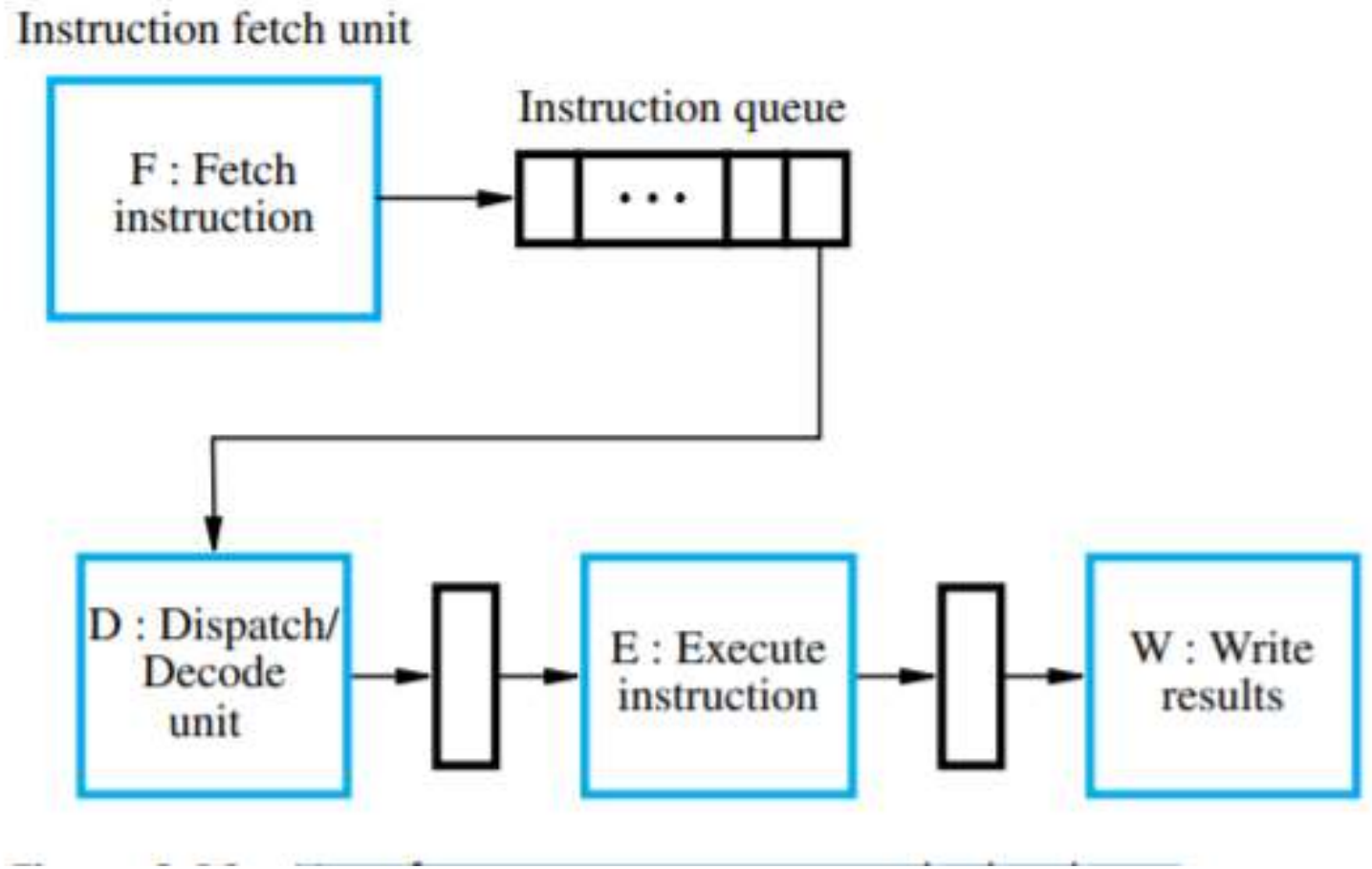
Reducing the branch penalty requires the branch target address to be computed earlier in the pipeline

Reducing the branch penalty requires the branch target address to be computed earlier in the pipeline





Hardware Organization



Conditional Branches And Branch Prediction

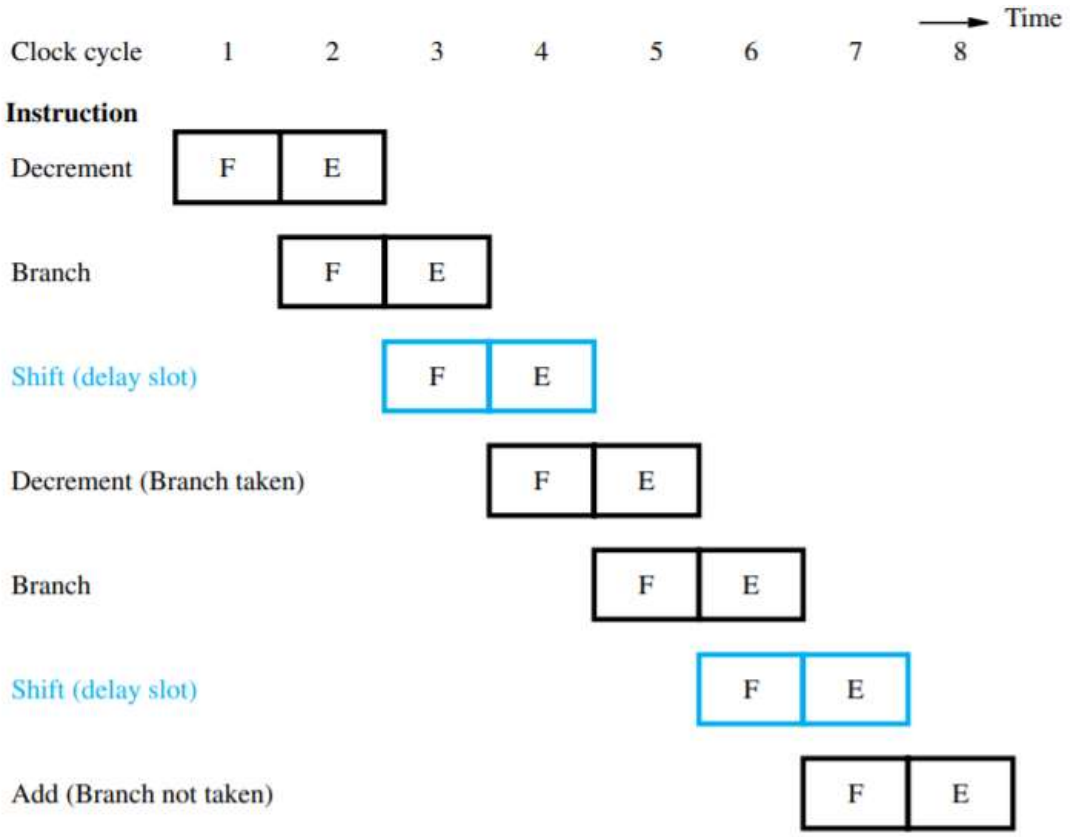
Reordering of instructions for a delayed branch

LOOP	Shift_left	R1
	Decrement	R2
	Branch=0	LOOP
NEXT	Add	R1,R3

(a) Original program loop

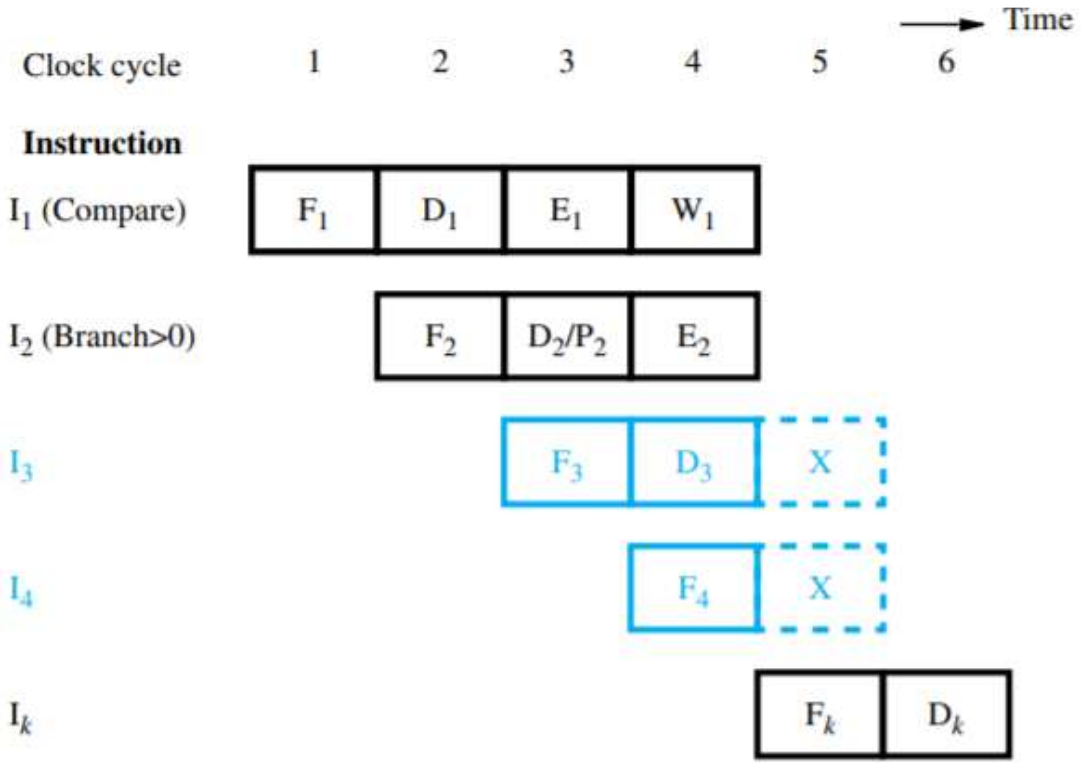
LOOP	Decrement	R2
	Branch=0	LOOP
	Shift_left	R1
NEXT	Add	R1,R3

(b) Reordered instructions



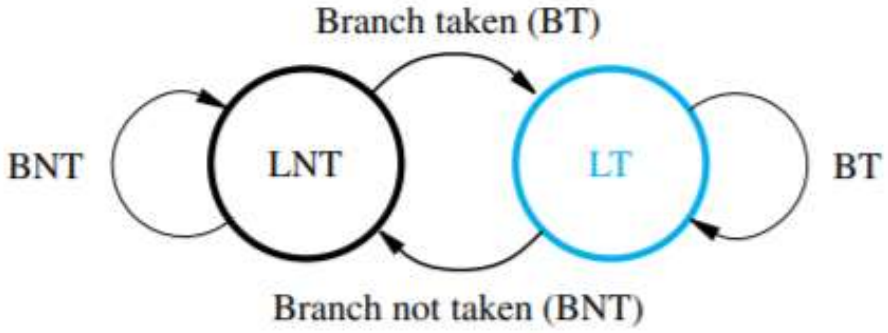
Conditional Branches And Branch Prediction

Timing when a branch decision has been incorrectly predicted as not taken

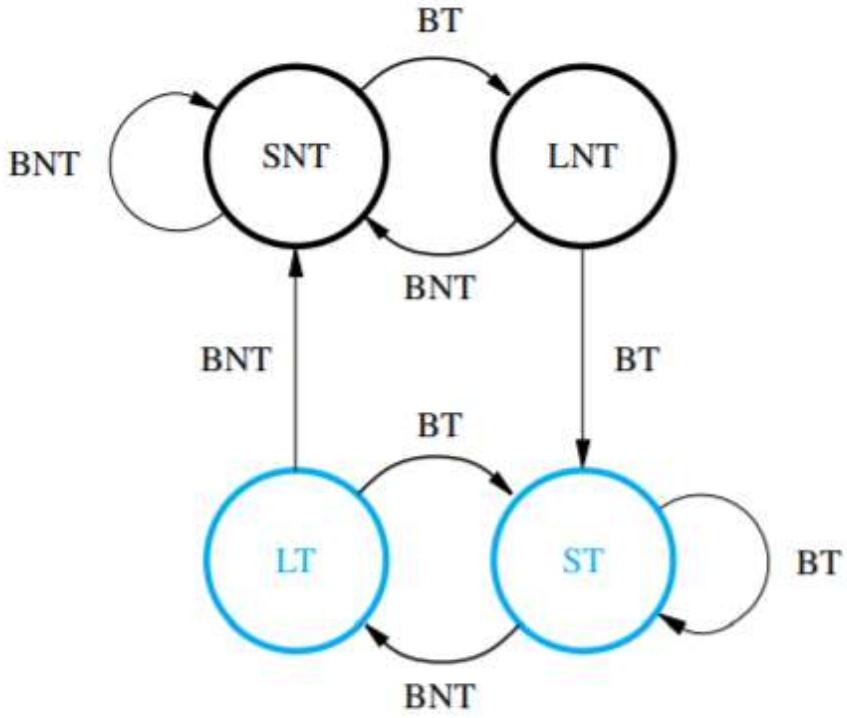


Conditional Branches And Branch Prediction

LT: Branch is likely to be taken
LNT: Branch is likely not to be taken



(a) A 2-state algorithm



(b) A 4-state algorithm

ST: Strongly likely to be taken
SNT: Strongly likely not to be taken


Thank You