Lecture 16
Data Hazards I

School of Computer Science and Engineering
Soongsil University

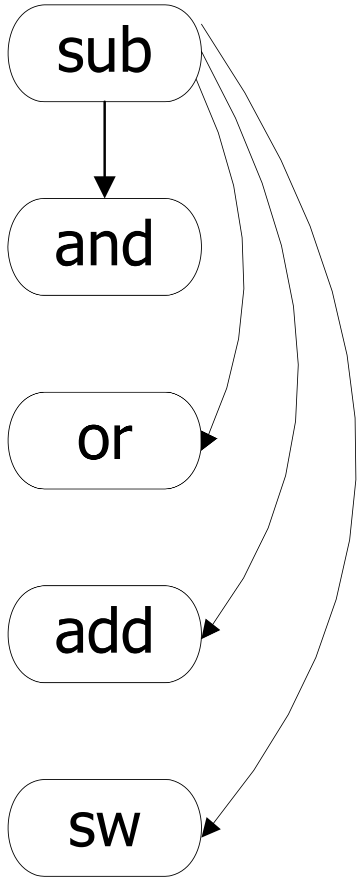
4. The Processor

4.7 Data Hazards: Forwarding vs. Stalling

- **Data dependence**

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

- **Data dependence graph**



Pipelined Dependence

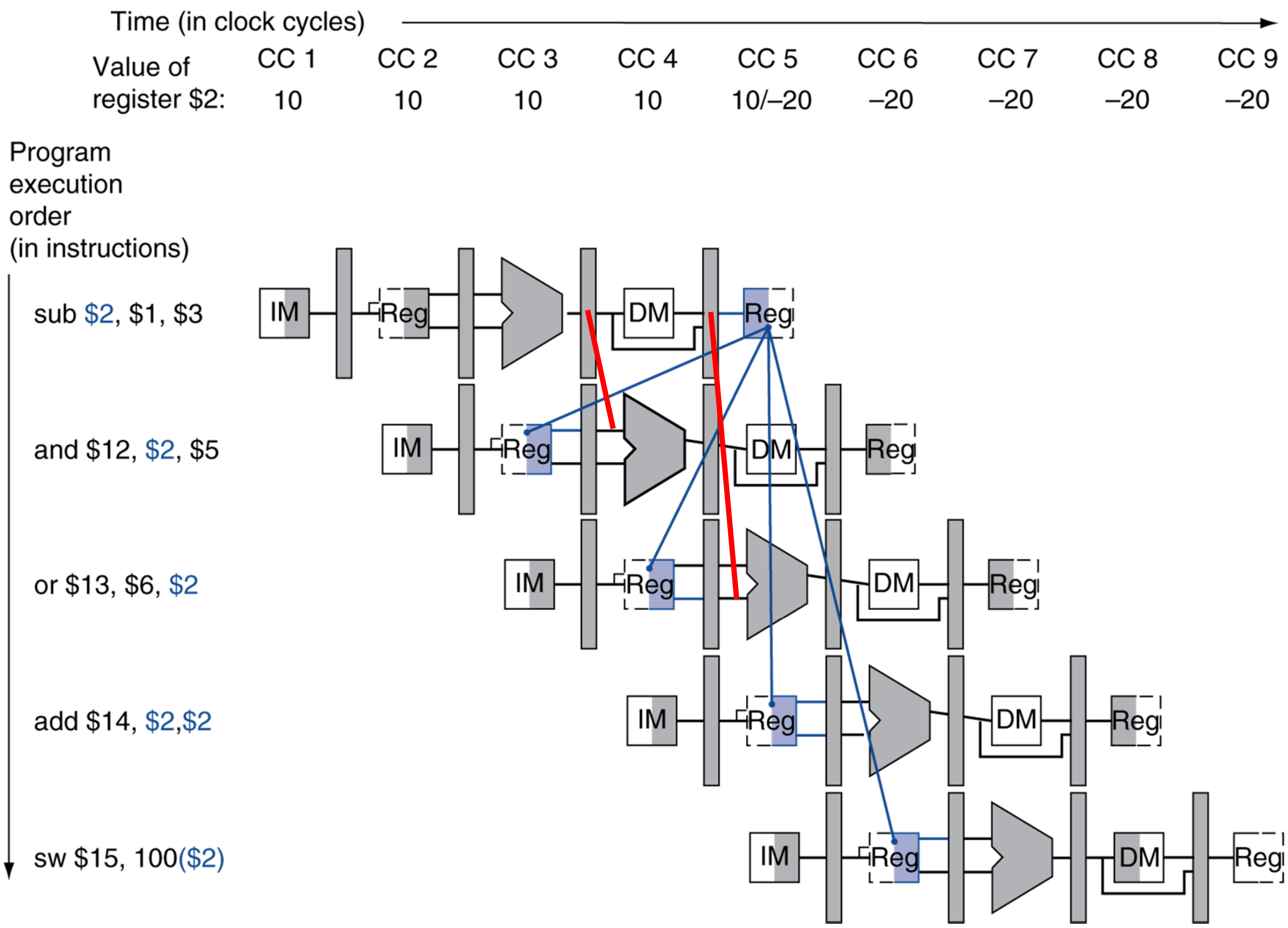
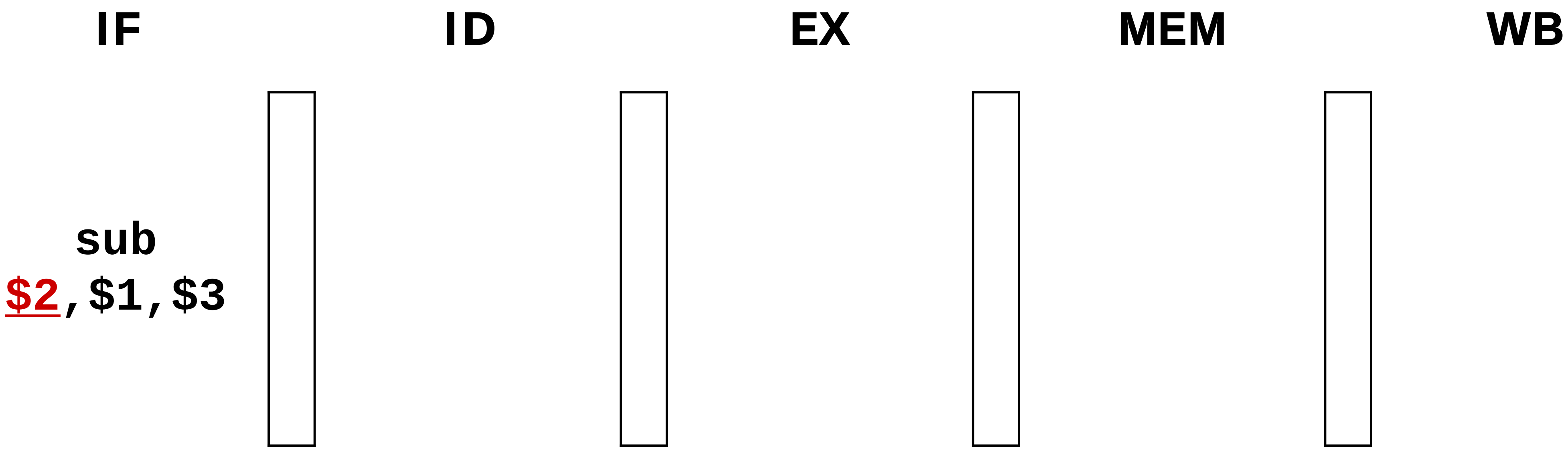
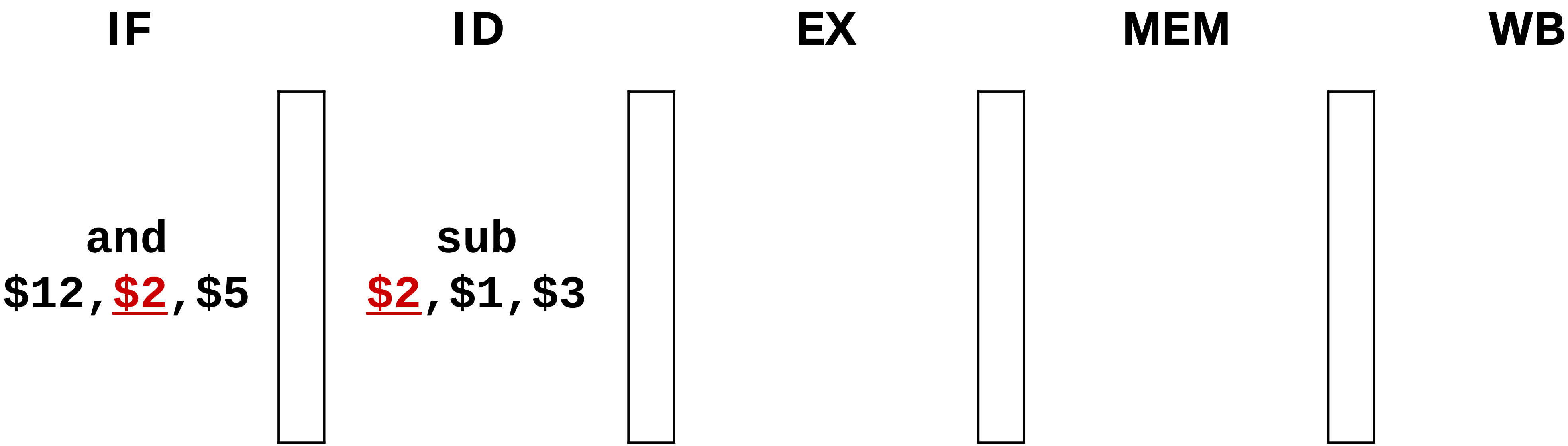


Figure 4.52

Clock Cycle 1



Clock Cycle 2



Clock Cycle 3

IF	ID	EX	MEM	WB
or $13,$6,<u>$2</u>	and $12,<u>$2</u>,$5	sub <u>$2</u>,$1,$3		

Clock Cycle 4 ?

IF	ID	EX	MEM	WB
add $14,$2,$2	or $13,$6,$2	and $12,$2,$5	sub $2,$1,$3	

Data Hazard

- **Occur when the pipeline must be stalled because one step must wait for another to complete**

1. **Data dependence**
   - An instruction depends on completion of data access by a previous instruction
   - ```
     add    $s0, $t0, $t1
     sub    $t2, $s0, $t3
     ```
2. **Two dependent instructions are close enough**
   - The distance is dependent on the pipeline structure.
     ```

Solution 1 – Stalling

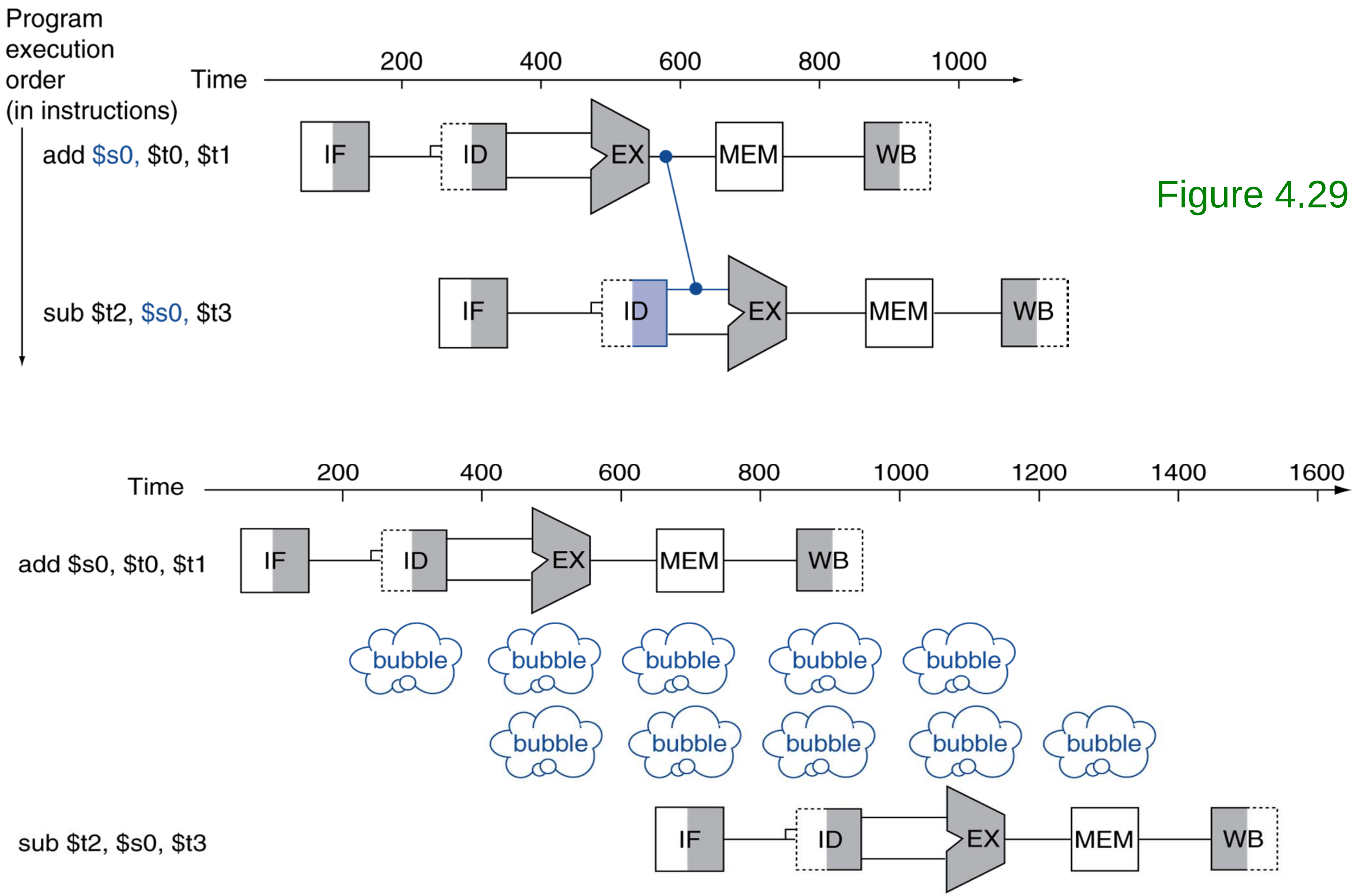


Figure 4.29

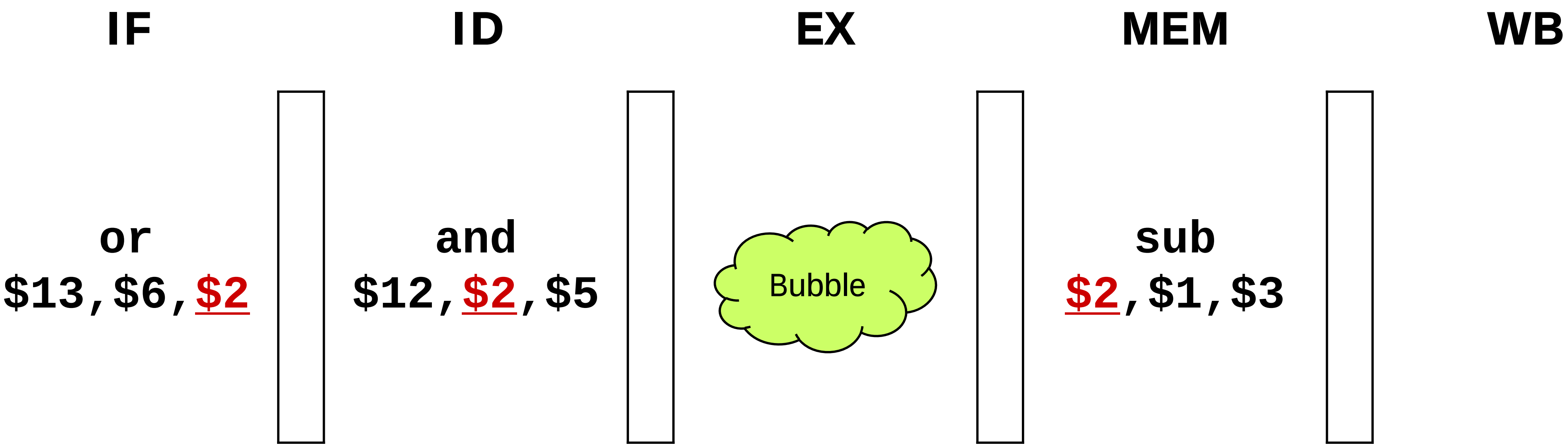
Pipeline Stall

	IF	ID	EX	MEM	WB
CC1	sub				
CC2	and	sub			
CC3	or	and	sub		
CC4	or	and	(bubble)	sub	
CC5	or	and	(bubble)	(bubble)	sub
CC6	add	or	and	(bubble)	(bubble)
CC7	sw	add	or	and	(bubble)
CC8		sw	add	or	and

Clock Cycle 3

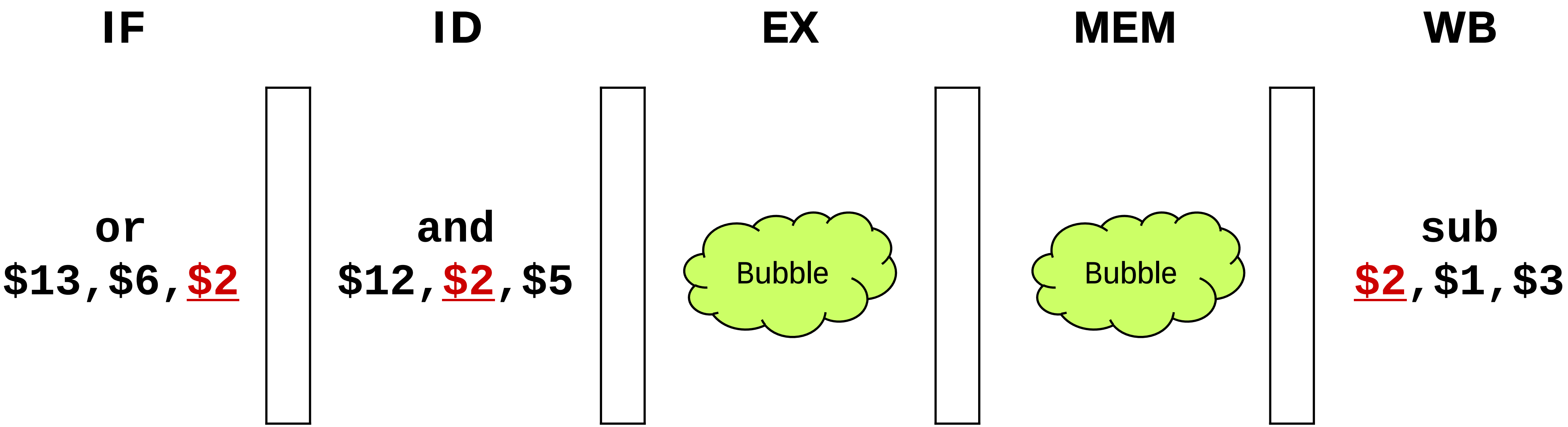
IF	ID	EX	MEM	WB
or $13,$6,$2	and $12,$2,$5	sub $2,$1,$3		

Clock Cycle 4

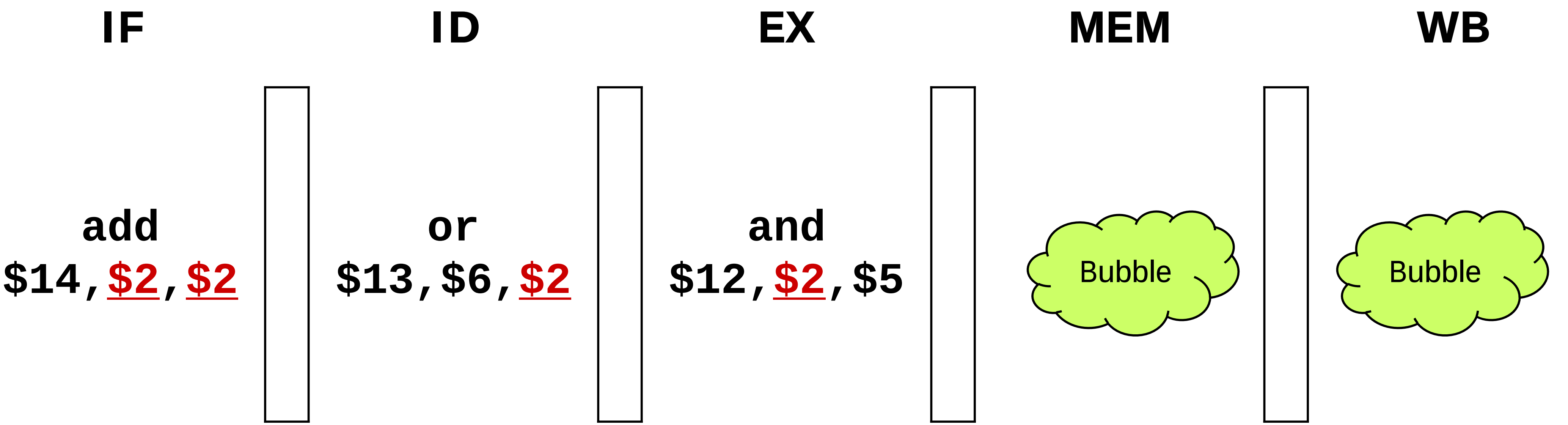


- **Pipeline interlock**
  - Hardware mechanisms to detect a data hazard and stall the pipeline until the hazard is cleared

Clock Cycle 5



Clock Cycle 6



Solution 2 – Insert nop Instructions

```
sub    $2, $1, $3
nop
nop
and    $12, $2, $5
or     $13, $6, $7
add    $14, $4, $8
sw     $15, 100($9)
```

Solution 3 – Code Scheduling

```
Sub   $2,  $1, $3
and   $12, $2, $5
or    $13, $6, $7
add   $14, $4, $8
sw    $15, 100($9)
```

```
sub   $2,  $1, $3
nop
nop
and   $12, $2, $5
or    $13, $6, $7
add   $14, $4, $8
sw    $15, 100($9)
```

```
sub   $2,  $1, $3
or    $13, $6, $7
add   $14, $4, $8
and   $12, $2, $5
sw    $15, 100($9)
```

Solution 4 – Forwarding (aka Bypassing)

- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath

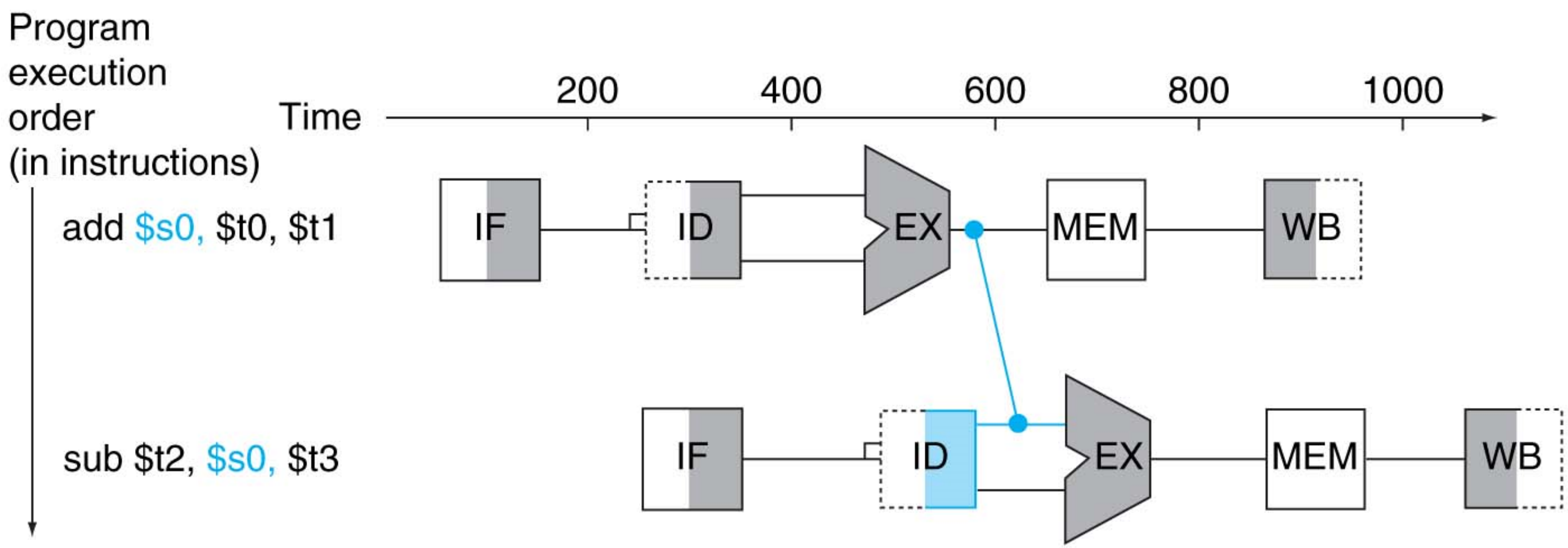


Figure 4.29

Hazard Conditions

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

- **Example: Dependence Detection**
  - **Classify the hazards in this sequence.**

```
sub   $2,  $1, $3   # Register $2 written by sub
and   $12, $2, $5   # 1st operand($2) set by sub
or    $13, $6, $2   # 2nd operand($2) set by sub
add   $14, $2, $2   # 1st($2) & 2nd($2) set by sub
sw    $15, 100($2)  # Index($2) set by sub
```

[Answer]

- **sub-and** hazard is type 1a

  EX/MEM.RegisterRd = ID/EX.RegisterRs = $2

- **sub-or** hazard is type 2b

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

- No data hazard

  between **sub** and **add**

  between **sub** and **sw**

Avoiding Unnecessary Forwarding

- **Instructions without register write**

  Check if RegWrite = 1.

- **Instructions having $zero as destination register**

  Check if EX/MEM.RegisterRd != 0 for 1a and 1b

  and MEM/WB.RegisterRd != 0 for 2a and 2b.

New Hazard Conditions

1a. EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)

1b. EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)

2a. MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)

2b. MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)