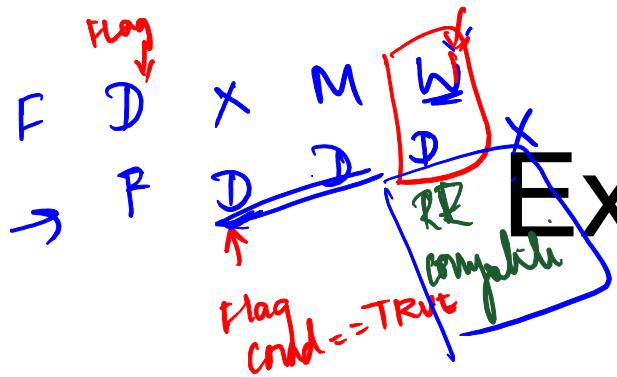


The Pipeline

Outline

- Why Pipeline?
 - How to pipeline?
- Speedup of the pipeline
- Pipelined datapath
 - Execution of instructions
 - Pipeline Timing diagram
- Dependences, Hazards
 - Structural, Data - Stalling, Forwarding
- Control Hazards, Branch prediction



Execution Sequence

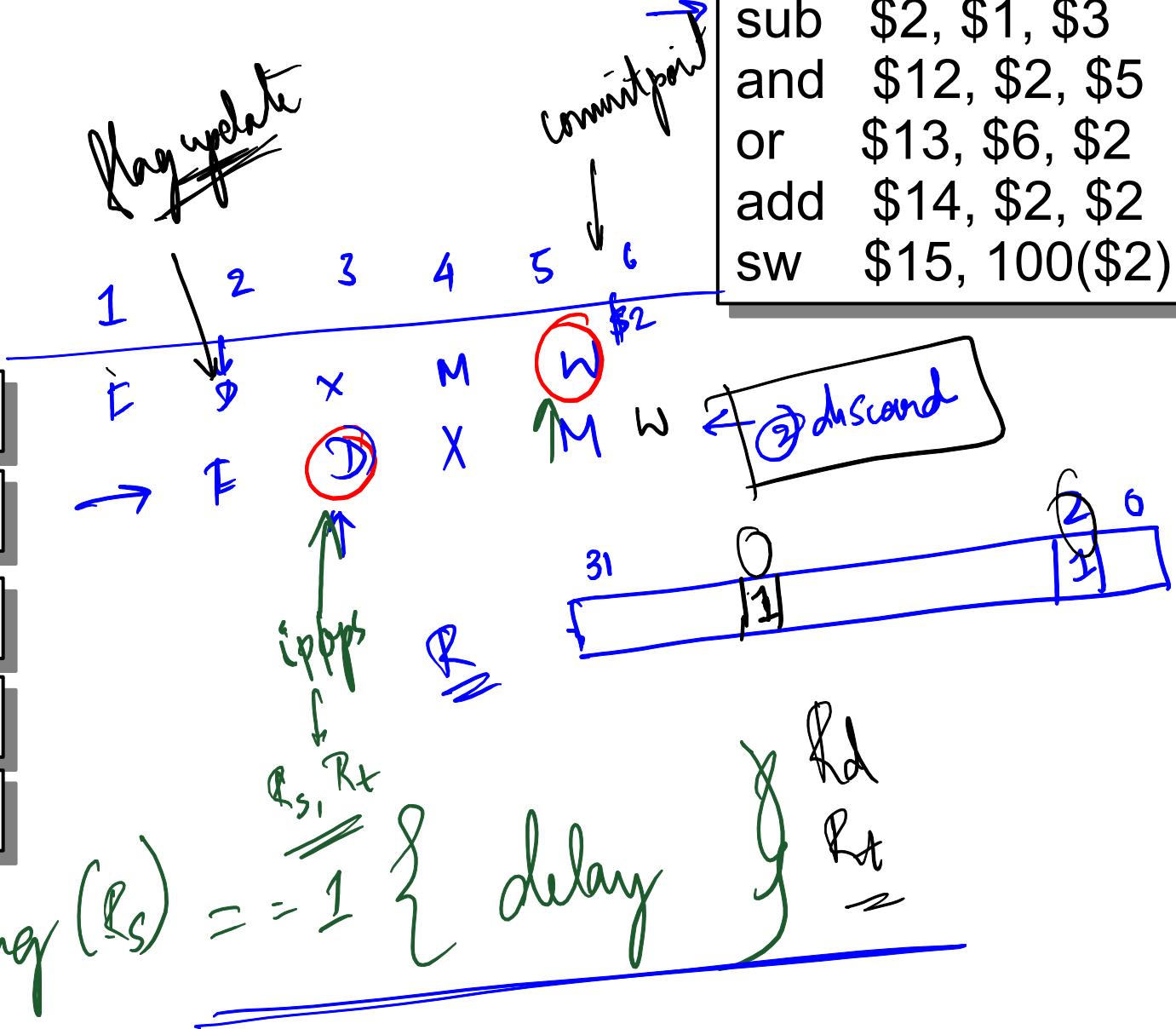
sub \$2, \$1, \$3

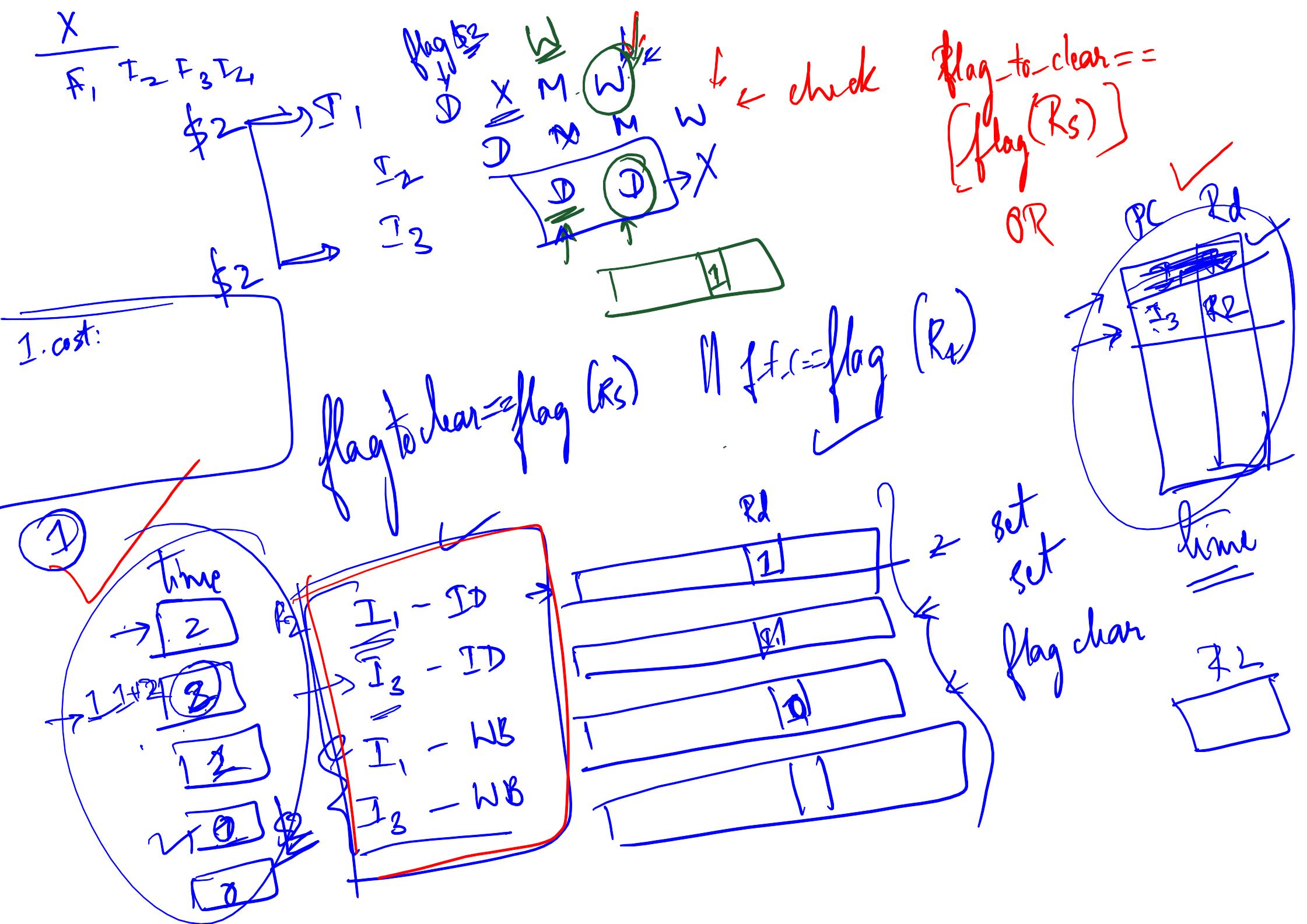
and \$12, \$2, \$5

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

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

sw \$15, 100(\$2)





Execution Sequence

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

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

IF ID EX MA WB

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

IF ID EX MA WB

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

IF ID EX MA WB

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

IF ID EX MA WB

sw \$15, 100(\$2)

IF ID EX MA WB

Execution Sequence – Dependence

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

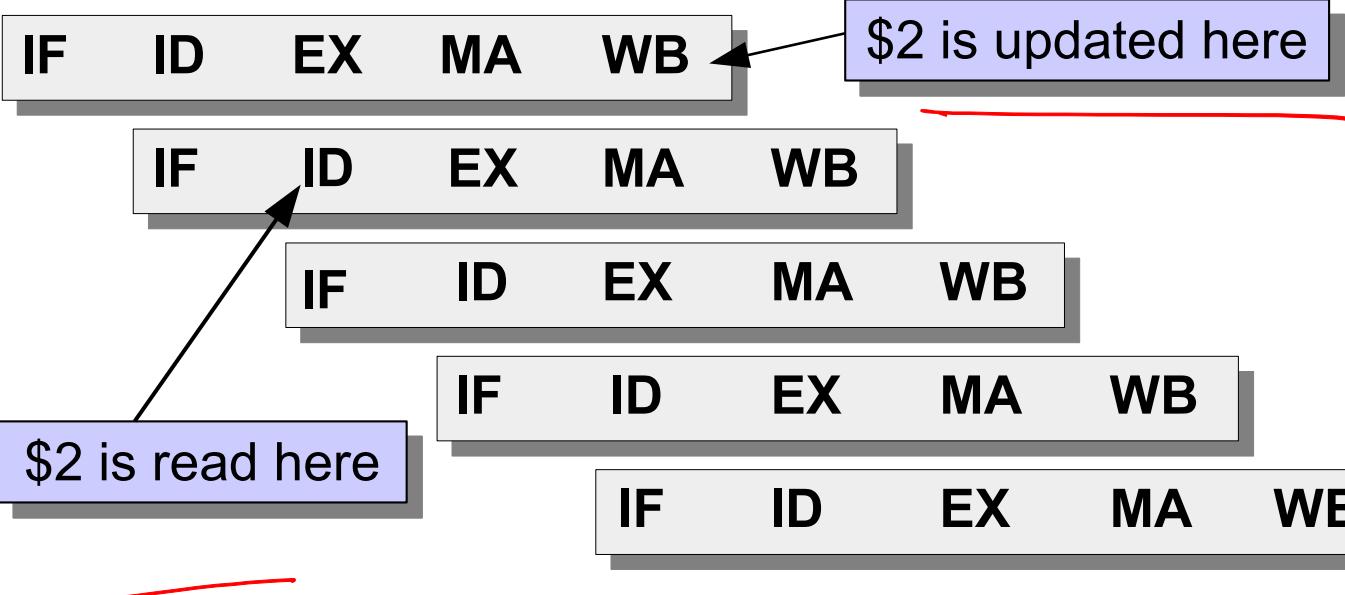
sub \$2, \$1, \$3

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

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

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

sw \$15, 100(\$2)



Execution Sequence – Dependence

Problem: \$2 is read before it is updated (written)

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

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

IF ID EX MA WB

\$2 is updated here

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

IF ID EX MA WB

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

IF ID EX MA WB

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

IF ID EX MA WB

sw \$15, 100(\$2)

\$2 is read here

IF ID EX MA WB

Execution Sequence – Dependence

RAW

and instruction should
read \$2 after sub writes

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

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

IF ID EX MA WB

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

IF ID EX MA WB

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

IF ID EX MA WB

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

IF ID EX MA WB

sw \$15, 100(\$2)

IF ID EX MA WB

Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

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

IF ID EX MA WB

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

IF

Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

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

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

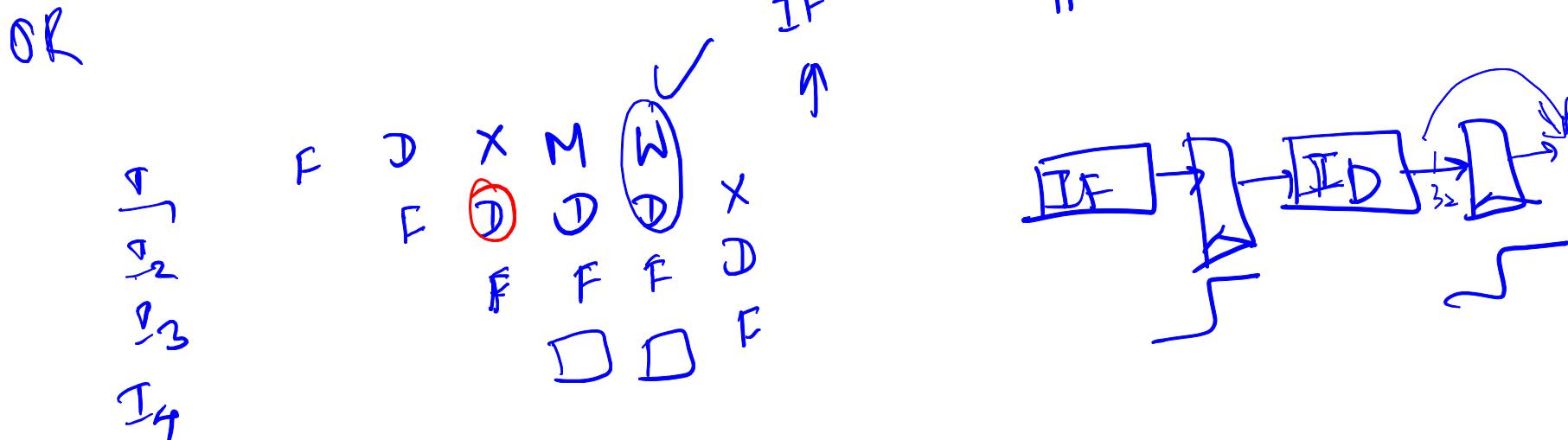
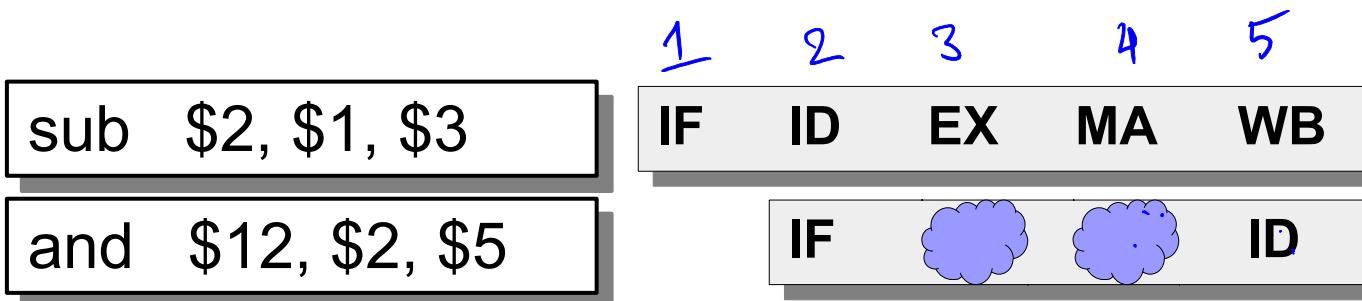


→ 2 stalls
(or 2 bubbles)

Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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



Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

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

IF ID EX MA WB

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

IF ID

2 stalls
(or 2 bubbles)

CR > 1.0

Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

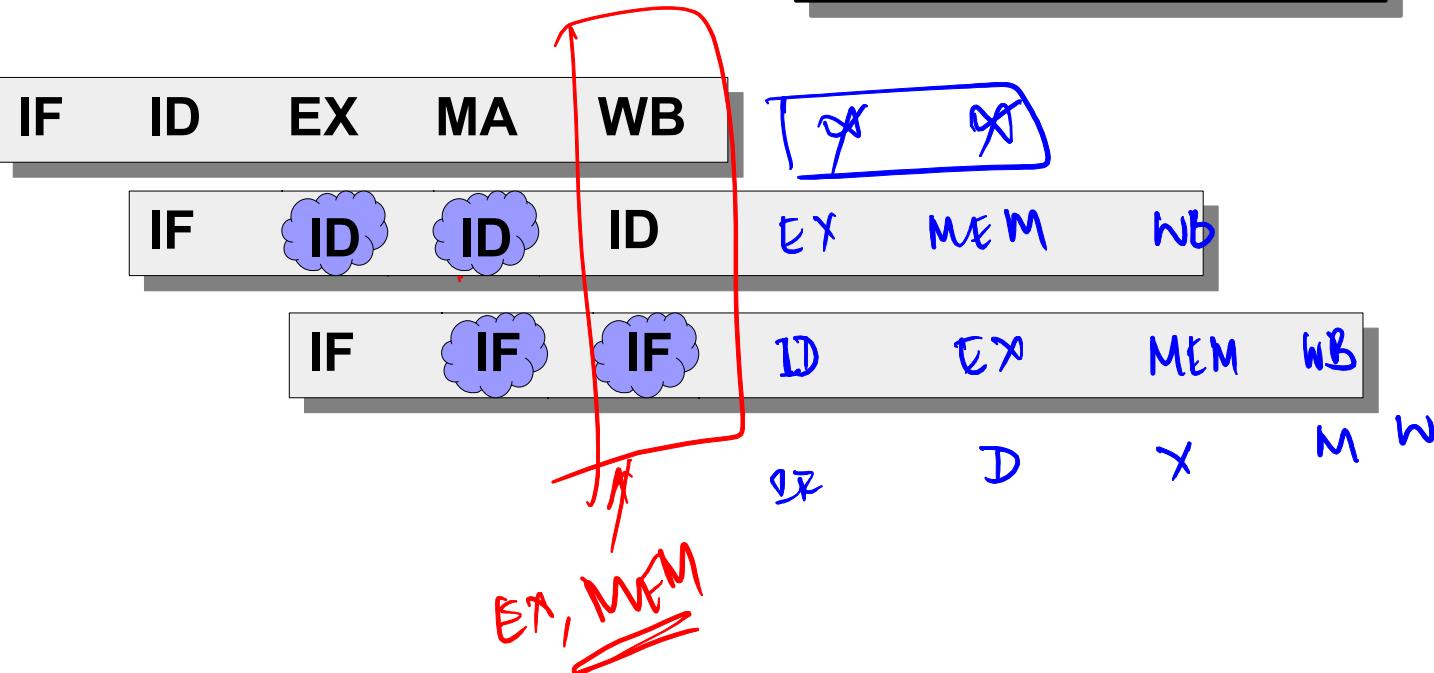
sub \$2, \$1, \$3

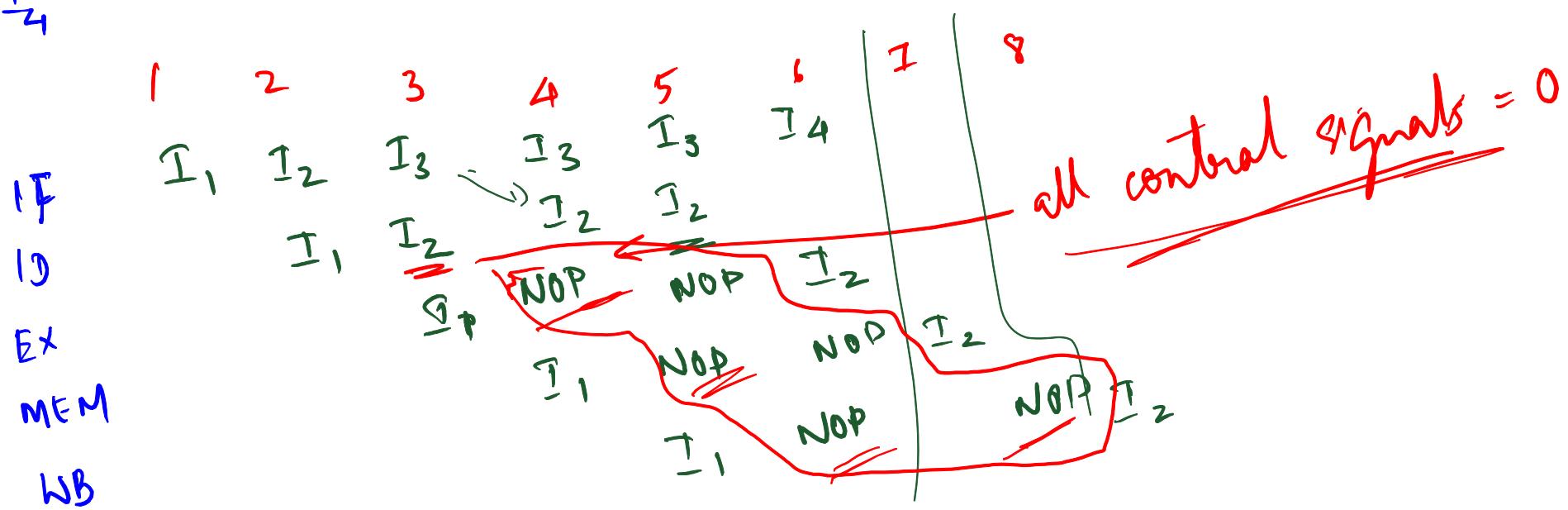
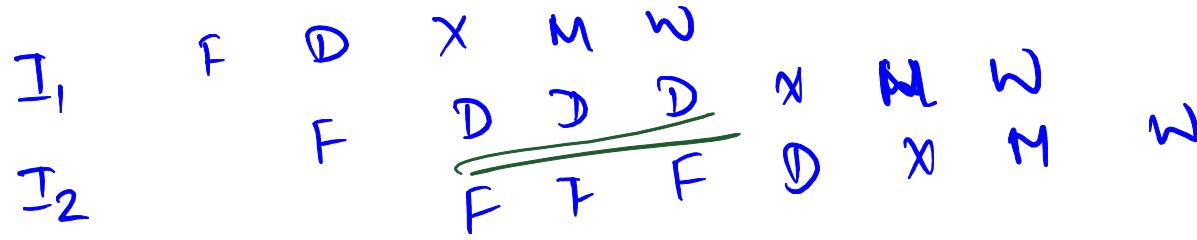
and \$12, \$2, \$5

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

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

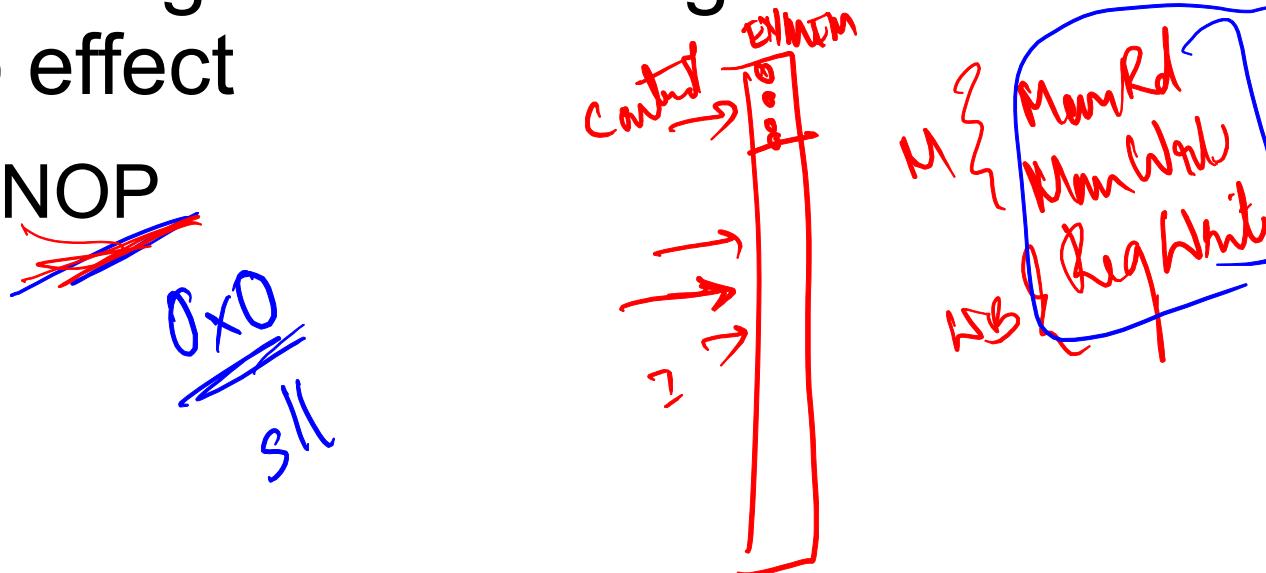
sw \$15, 100(\$2)





Pipeline Stall

- Stall propagates
- Stall: Prevent the PC and IF/ID pipeline register from changing
- EX stage is executing instructions that have no effect
 - NOP



Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

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

IF ID EX MA WB

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

IF ID ID ID

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

IF IF IF

Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

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

IF ID EX MA WB

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

IF ID ID EX

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

IF IF IF ID

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

IF

sw \$15, 100(\$2)

Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

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

IF ID EX MA WB

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

IF ID ID ID EX MA WB

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

IF IF IF ID EX MA WB

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

IF ID EX MA W

sw \$15, 100(\$2)

IF ID EX M

Execution Sequence – Dependence

and instruction should
read \$2 after sub writes

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

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

IF ID EX MA WB

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

IF ID ID ID EX MA WB

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

IF IF IF ID EX MA WB

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

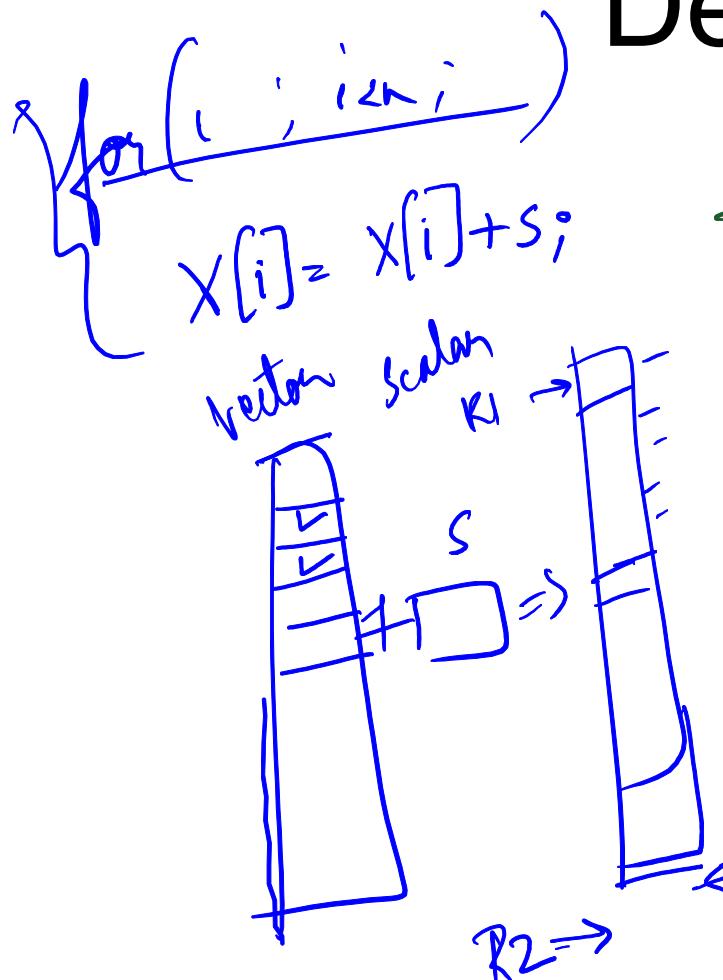
IF ID EX MA W

sw \$15, 100(\$2)

IF ID EX M

What is the Speedup?

Dependence

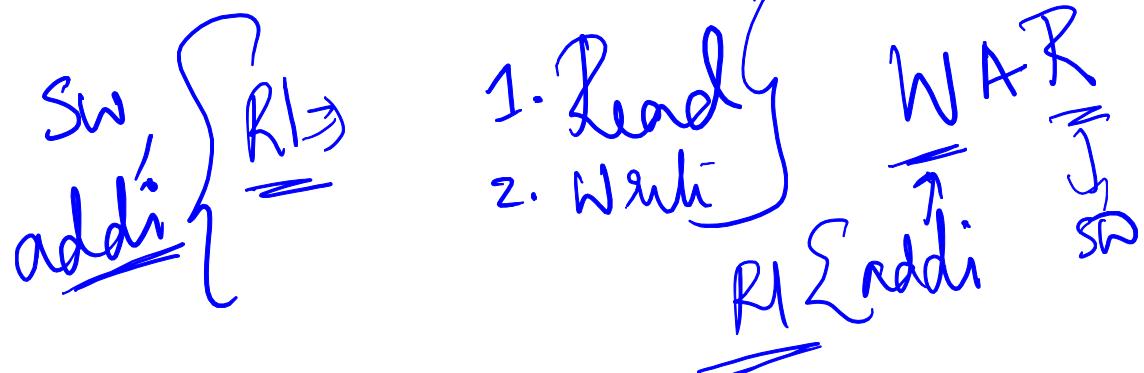


loop:

- LW F0, 0(R1) ①
- ADD F4, F0, F2 ②
- { SW F4, 0(R1) ③
- ADDI R1, R1, # -8
- BNE R1, R2, Loop

Basic
block

Data Dependence: RAW



Dependence

Data Dependence

```
loop: LW    F0, 0(R1)
      ADD   F4, F0, F2
      SW    F4, 0(R1)
      ADDI  R1, R1, #-8
      BNE   R1, R2, Loop
```

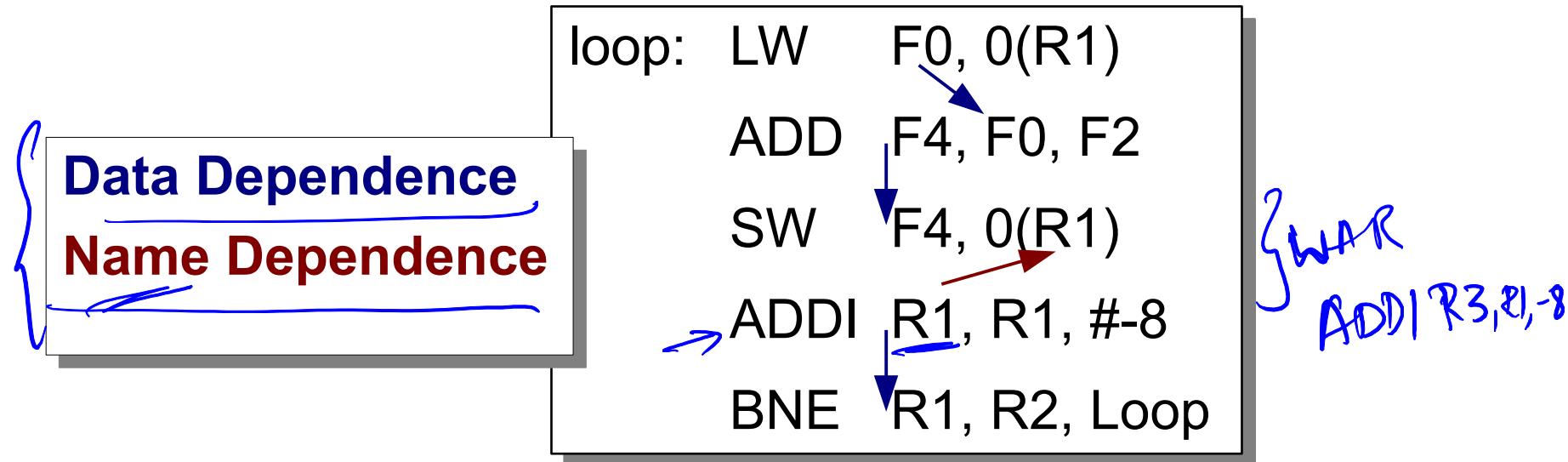
Dependence

```
loop: LW    F0, 0(R1)
      ADD   F4, F0, F2
      SW    F4, 0(R1)
      ADDI  R1, R1, #-8
      BNE   R1, R2, Loop
```

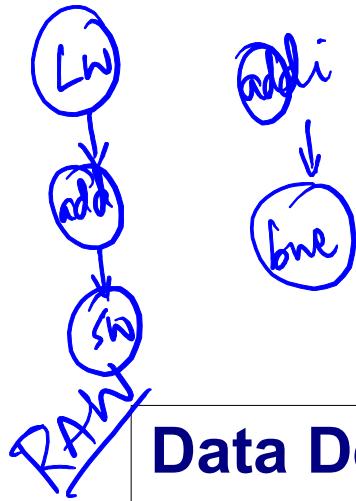
The diagram illustrates data dependencies between five instructions in a loop:

- LW**: $F0, 0(R1)$ → $F4$ (blue arrow)
- ADD**: $F4$ → $F4$ (blue arrow)
- SW**: $F4$ → $F4, 0(R1)$ (red arrow)
- ADDI**: $R1$ → $R1, R1, \#-8$ (blue arrow)
- BNE**: $R1$ → $R1, R2, \text{Loop}$

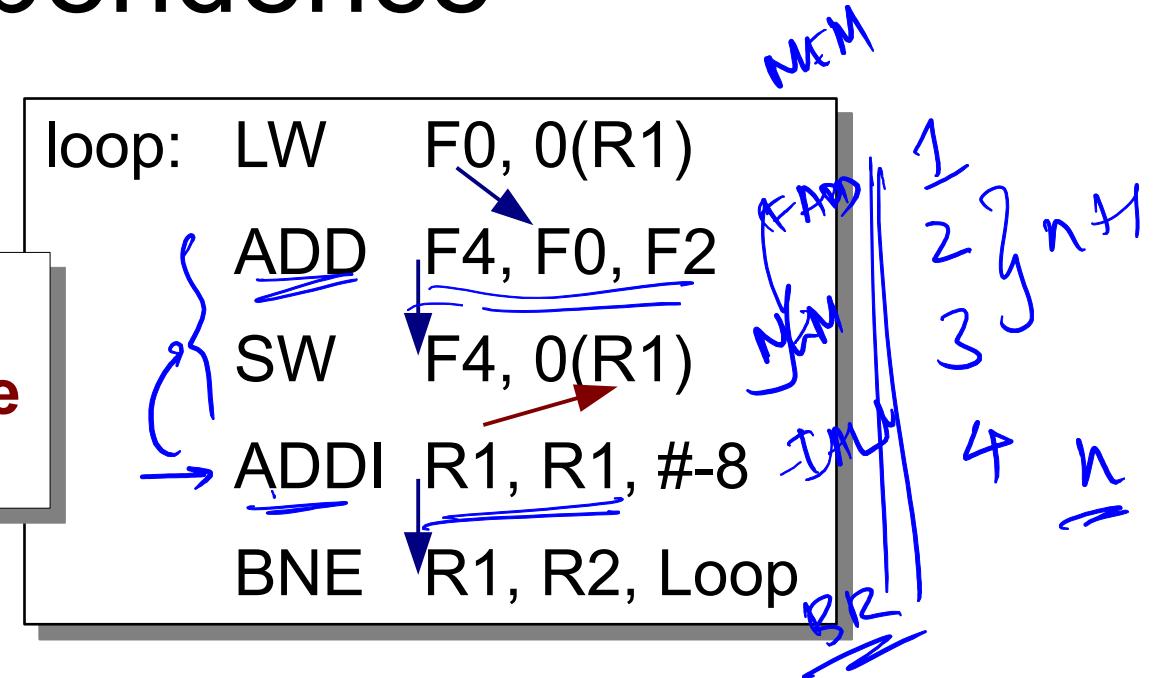
Dependence



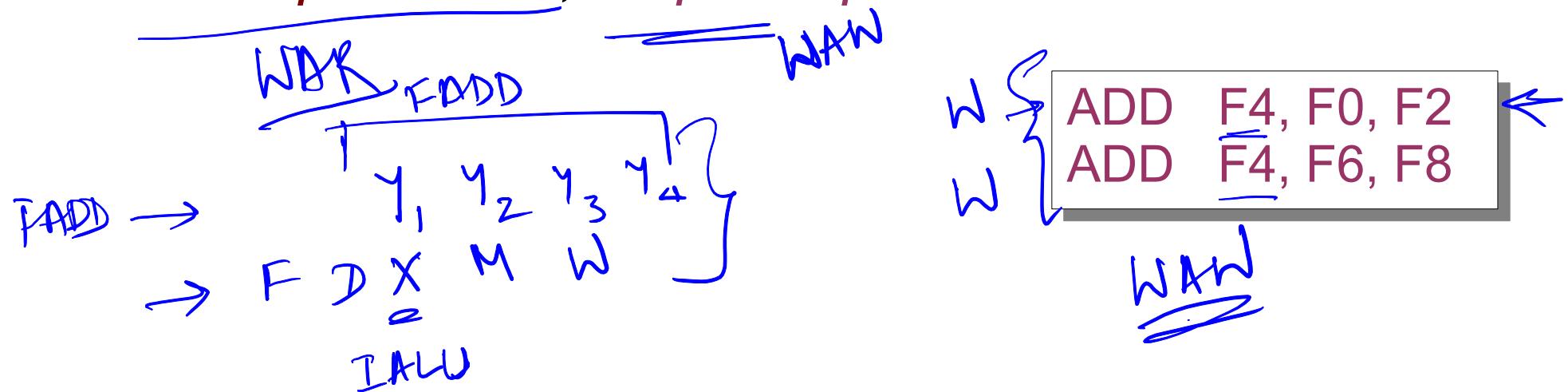
Dependence

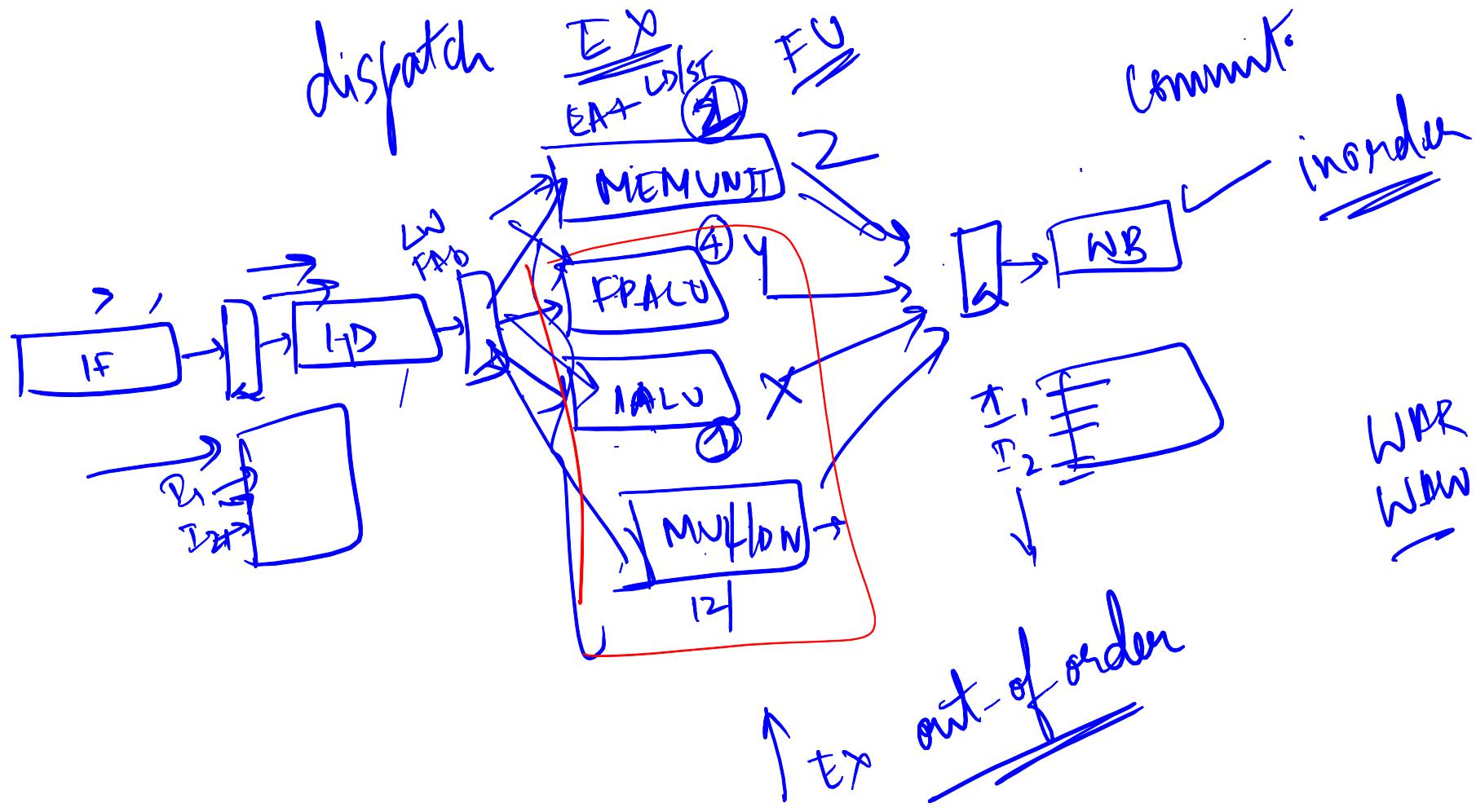


Data Dependence
Name Dependence



- Name dependence
 - antidependence, output dependence





Dependence

Data Dependence
Name Dependence

loop:
LW F0, 0(R1)
ADD F4, F0, F2
SW F4, 0(R1) *WAW*
ADDI R1, R1, # -8
BNE R1, R2, Loop

- Name dependence
 - *antidependence, output dependence*
- Register renaming

WAW
ADD F4, F0, F2
ADD F4, F6, F8 *WAW*

Dependence

Data Dependence
Name Dependence

```
loop: LW F0, 0(R1)
      ADD F4, F0, F2
      SW F4, 0(R1)
      ADDI R1, R1, #-8
      BNE R1, R2, Loop
```

- RAW, WAR, WAW dependences

stall

```
ADD F4, F0, F2
ADD F4, F6, F8
```

Dependence

Data Dependence
Name Dependence

```
loop: LW F0, 0(R1)
      ADD F4, F0, F2
      SW F4, 0(R1)
      ADDI R1, R1, #-8
      BNE R1, R2, Loop
```

- RAW, WAR, WAW dependences
- Hazard
 - Overlap during execution could change the order of access to the operand involved in the dependence.

```
ADD F4, F0, F2
ADD F4, F6, F8
```

Pipeline Hazards

- Hazard: *n.* An unavoidable danger or risk, even though often foreseeable

Pipeline Hazards

- Hazard: *n.* An unavoidable danger or risk, even though often foreseeable.
- Situations that prevent the next instruction in the instruction stream from being executing during its designated clock cycle

Pipeline Hazards

- Hazard: *n.* An unavoidable danger or risk, even though often foreseeable.
- Situations that prevent the next instruction in the instruction stream from being executing during its designated clock cycle
- Reduce the performance from the ideal speedup gained by pipelining

Hazards

- Structural Hazards
 - Resource conflicts

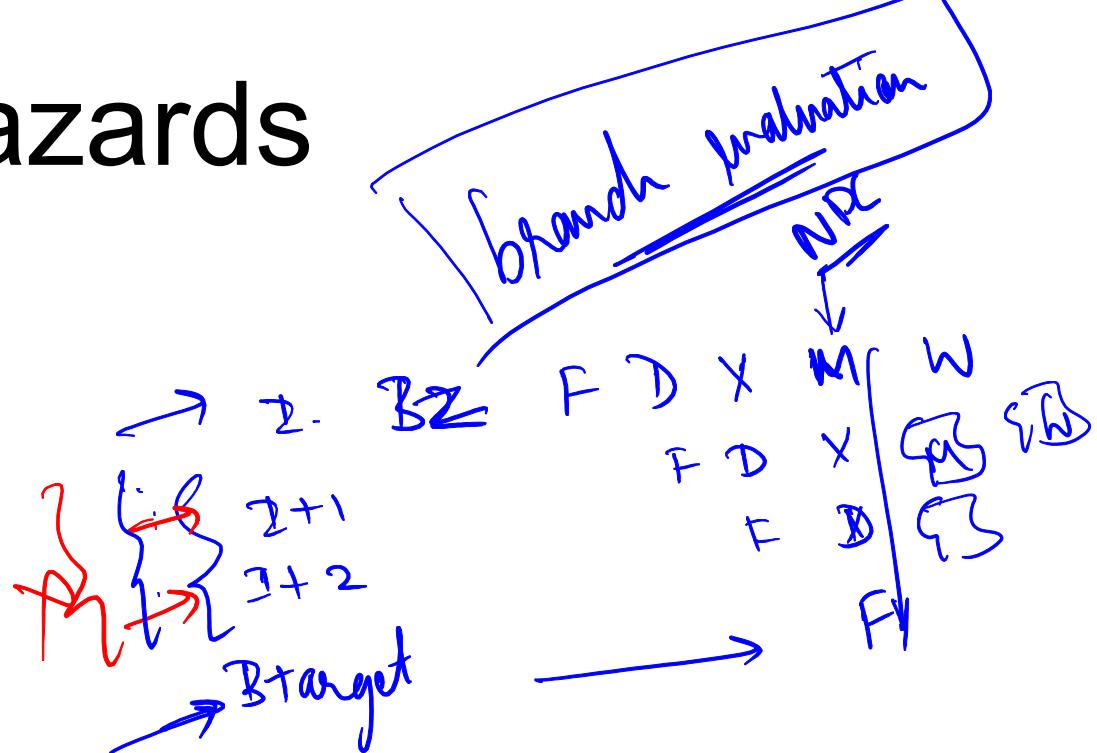


Hazards

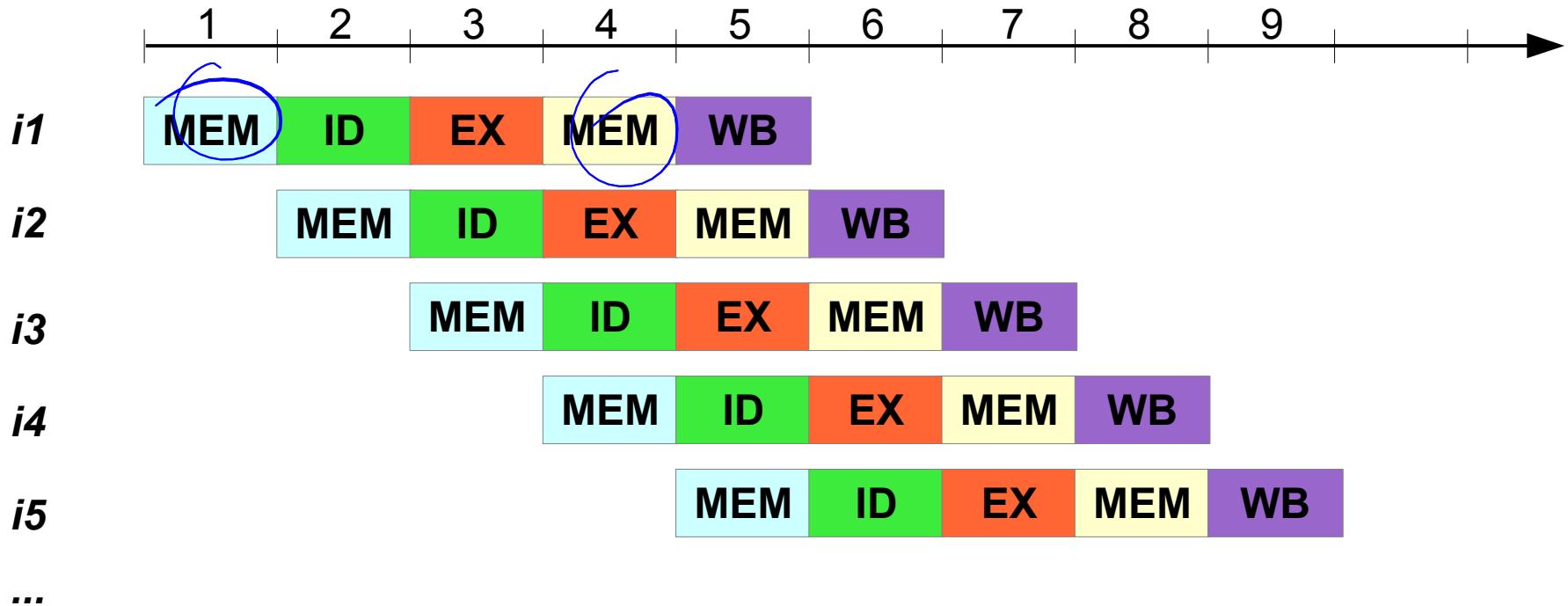
- Structural Hazards
 - Resource conflicts
- Data Hazards
 - RAW, WAW, WAR
-

Hazards

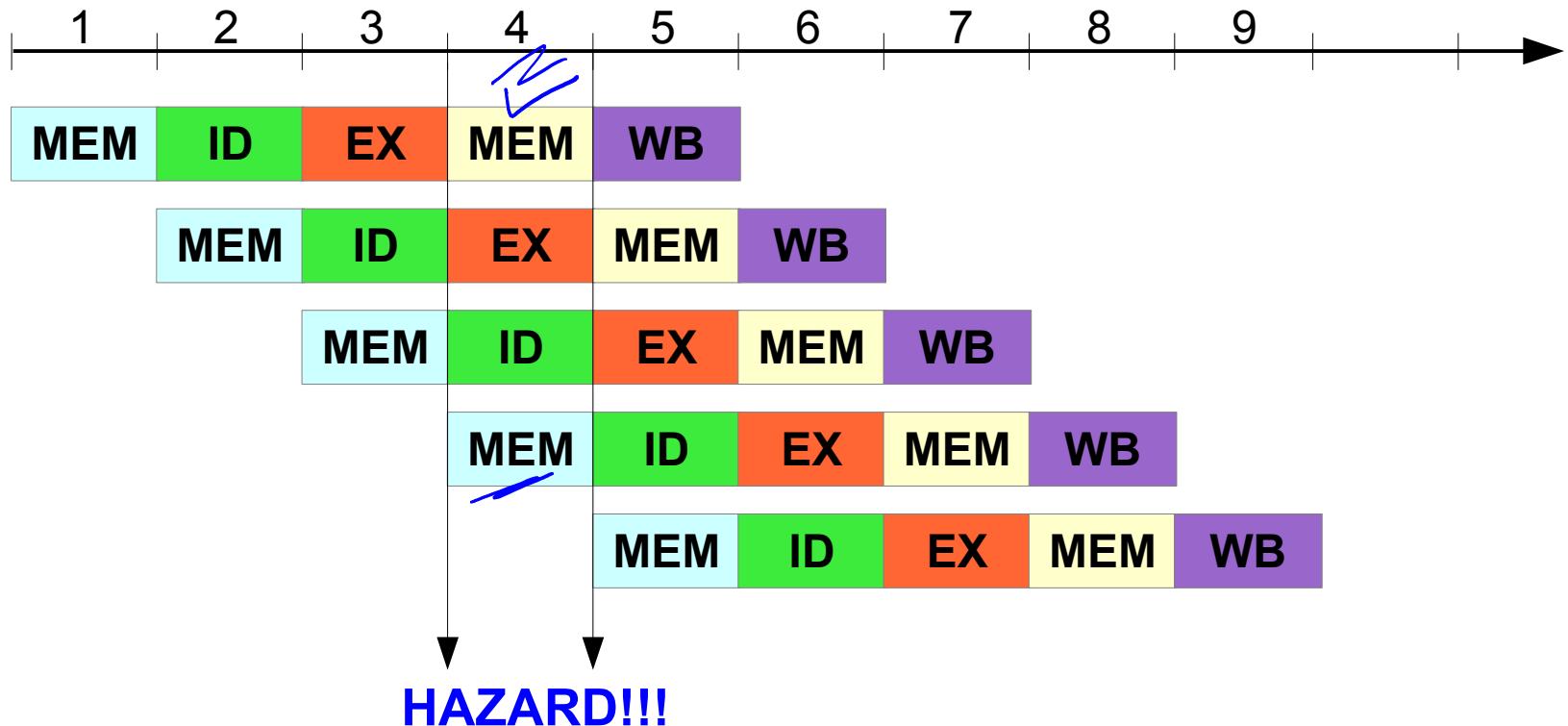
- Structural Hazards
 - Resource conflicts
- Data Hazards
 - RAW, WAW, WAR
- Control Hazard
 - Whether or not an instruction should be executed depends on a control decision made by an earlier instruction



Structural Hazard

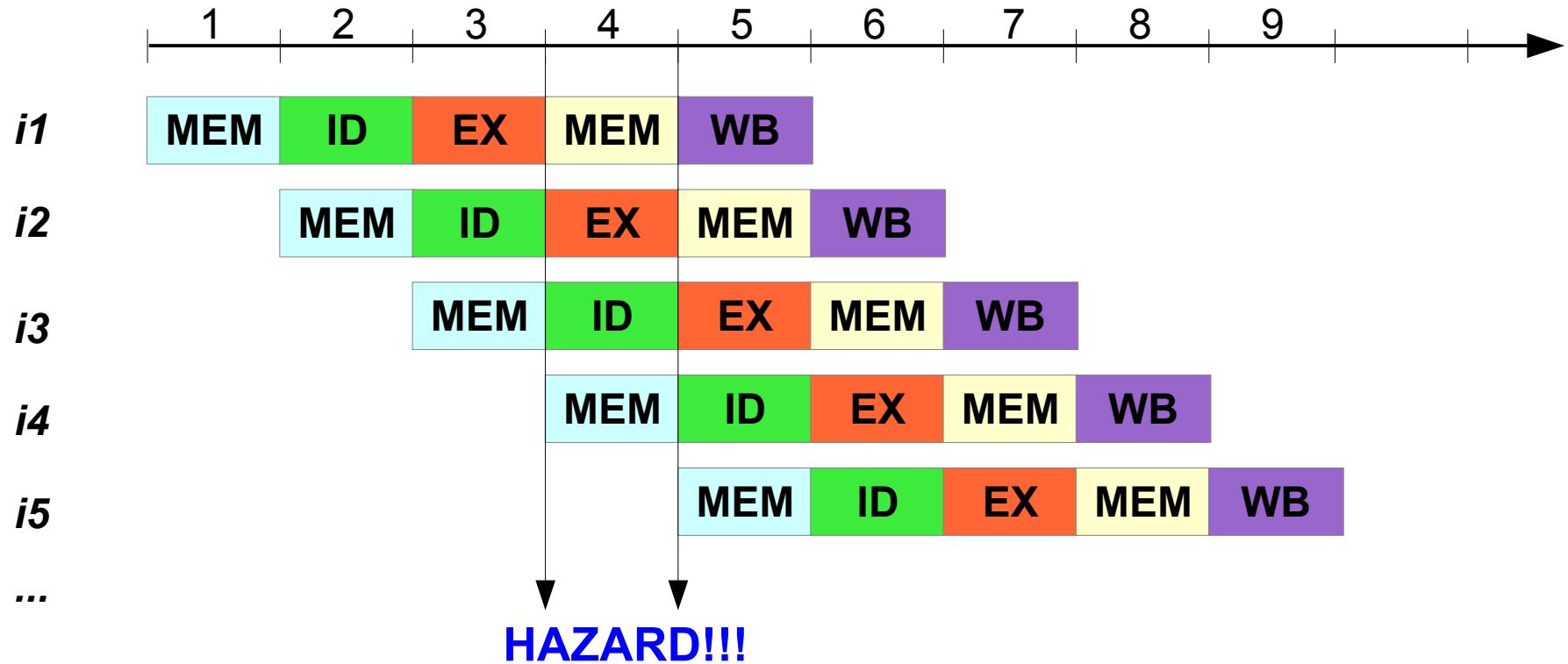


Structural Hazard



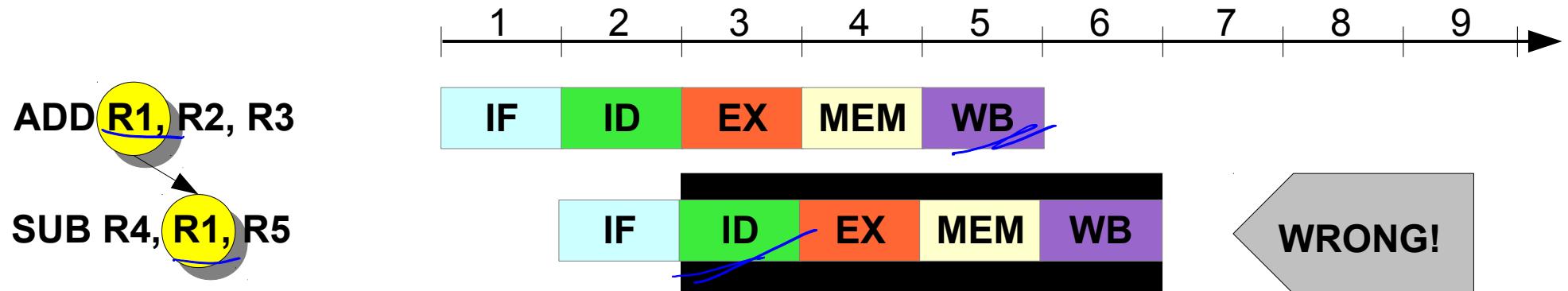
- Lack of resources

Structural Hazard

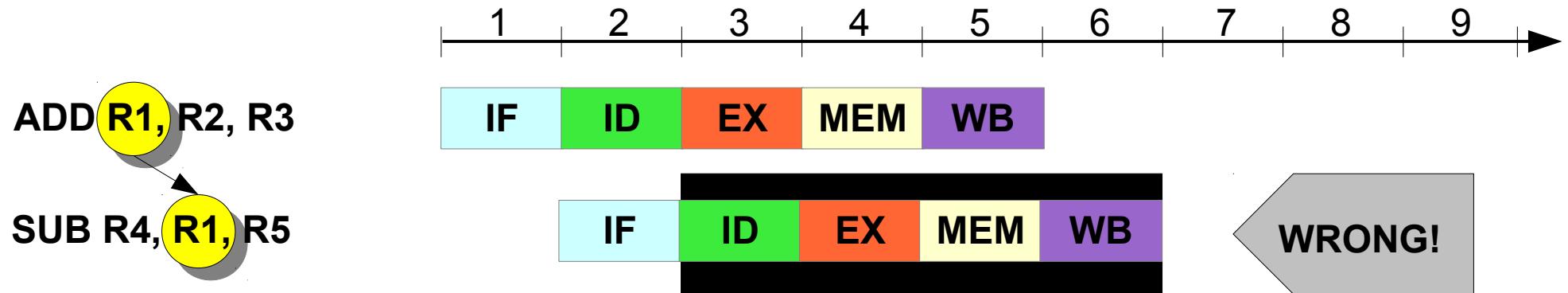


- Lack of resources
- Solution: Increase resources
 - Use of separate Data and Instruction memories in the MIPS pipeline

Data Hazard

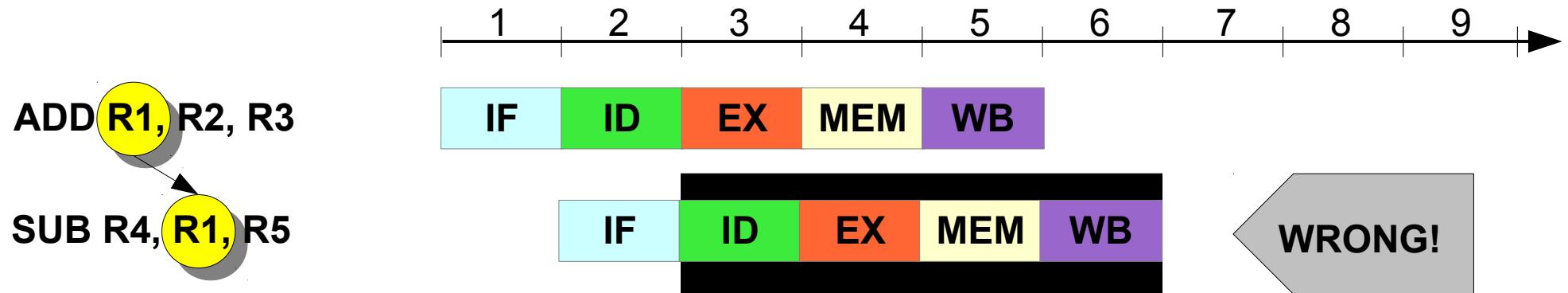


Data Hazard



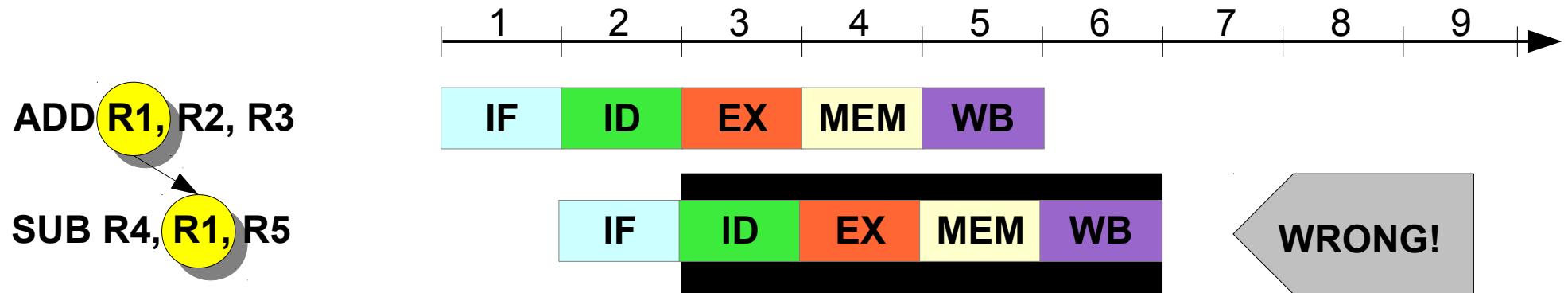
- Data (input operands) required by the instruction are not ready

Data Hazard

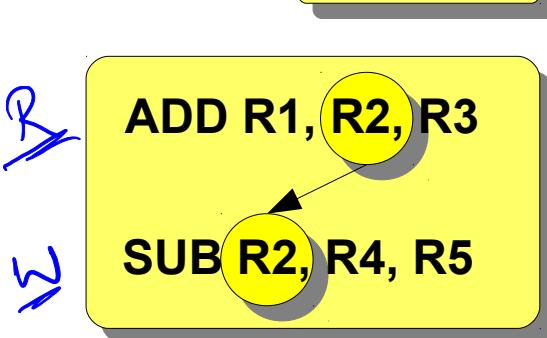


- Data (input operands) required by the instruction are not ready
- Data dependence
- RAW, WAR, WAW dependences

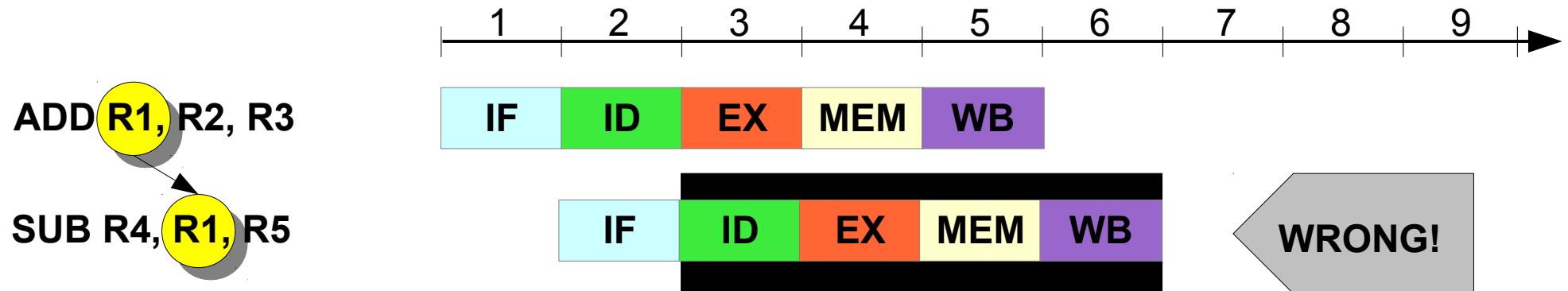
Data Hazard



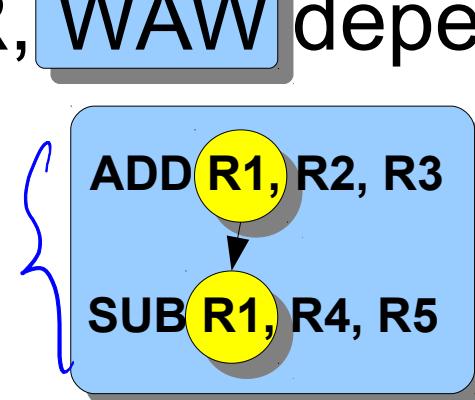
- Data (input operands) required by the instruction are not ready
- Data dependence
- RAW, WAR, WAW dependences



Data Hazard

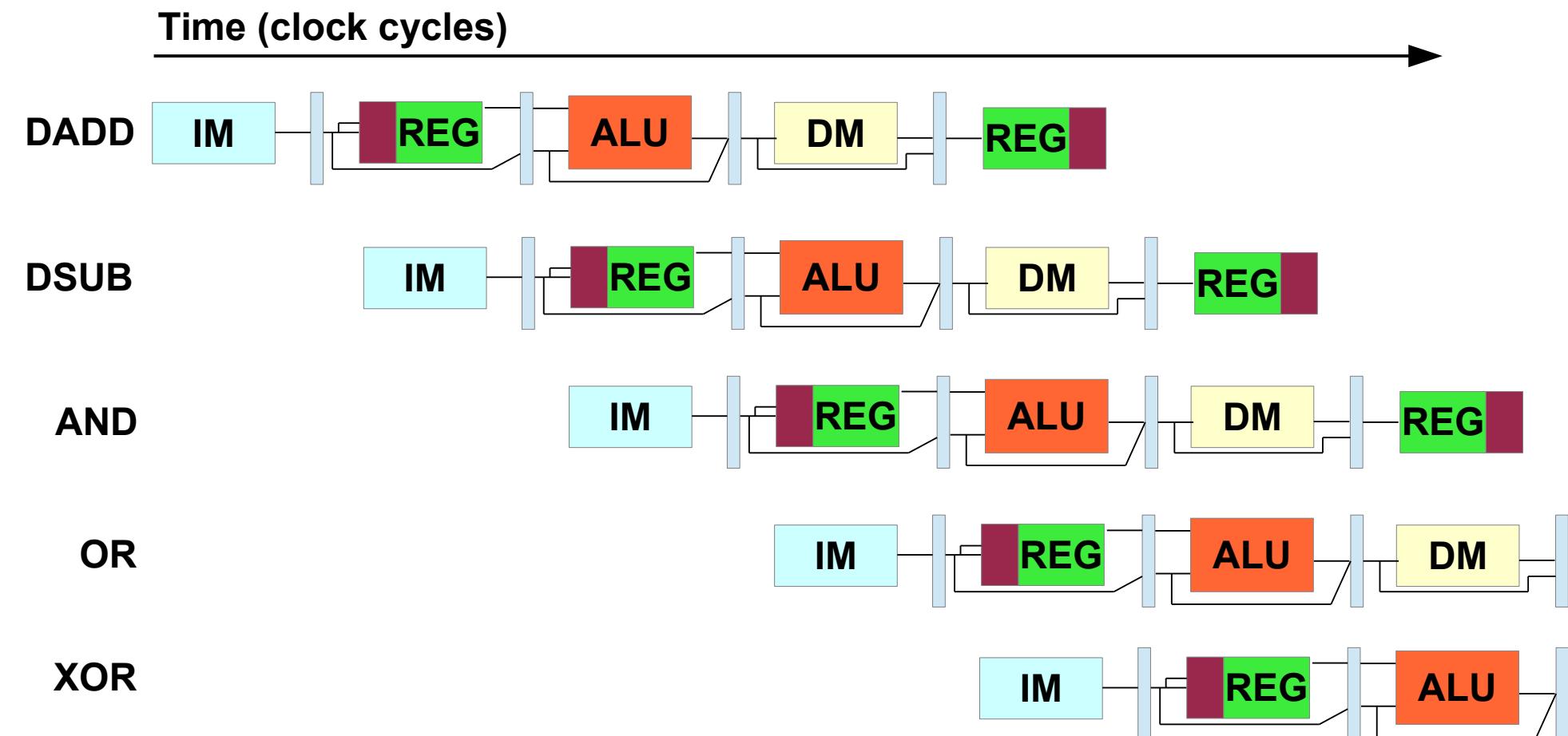


- Data (input operands) required by the instruction are not ready
- Data dependence
- RAW, WAR, WAW dependences



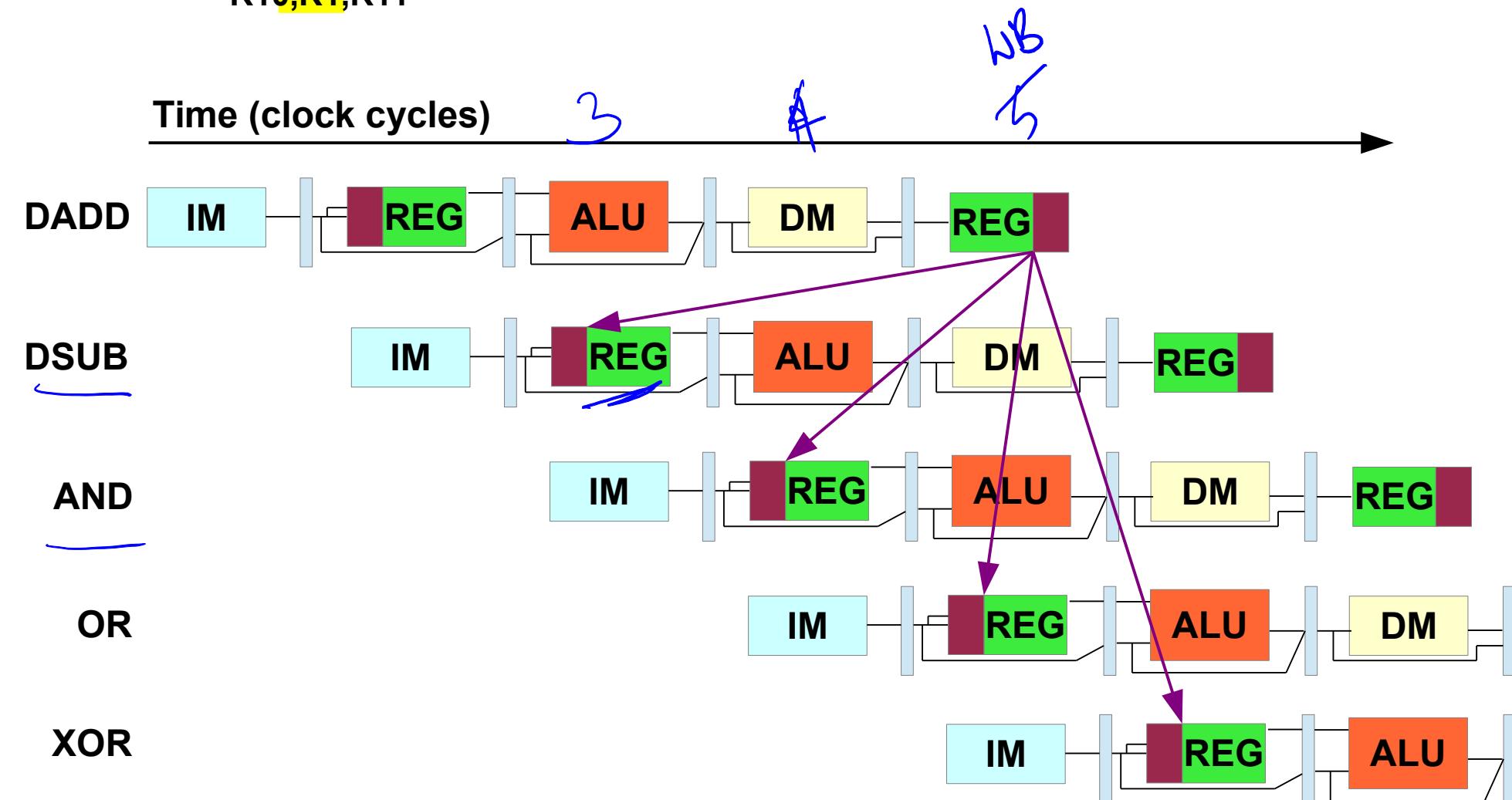
Data Hazard

DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9
XOR	R10,R1,R11



Data Hazard

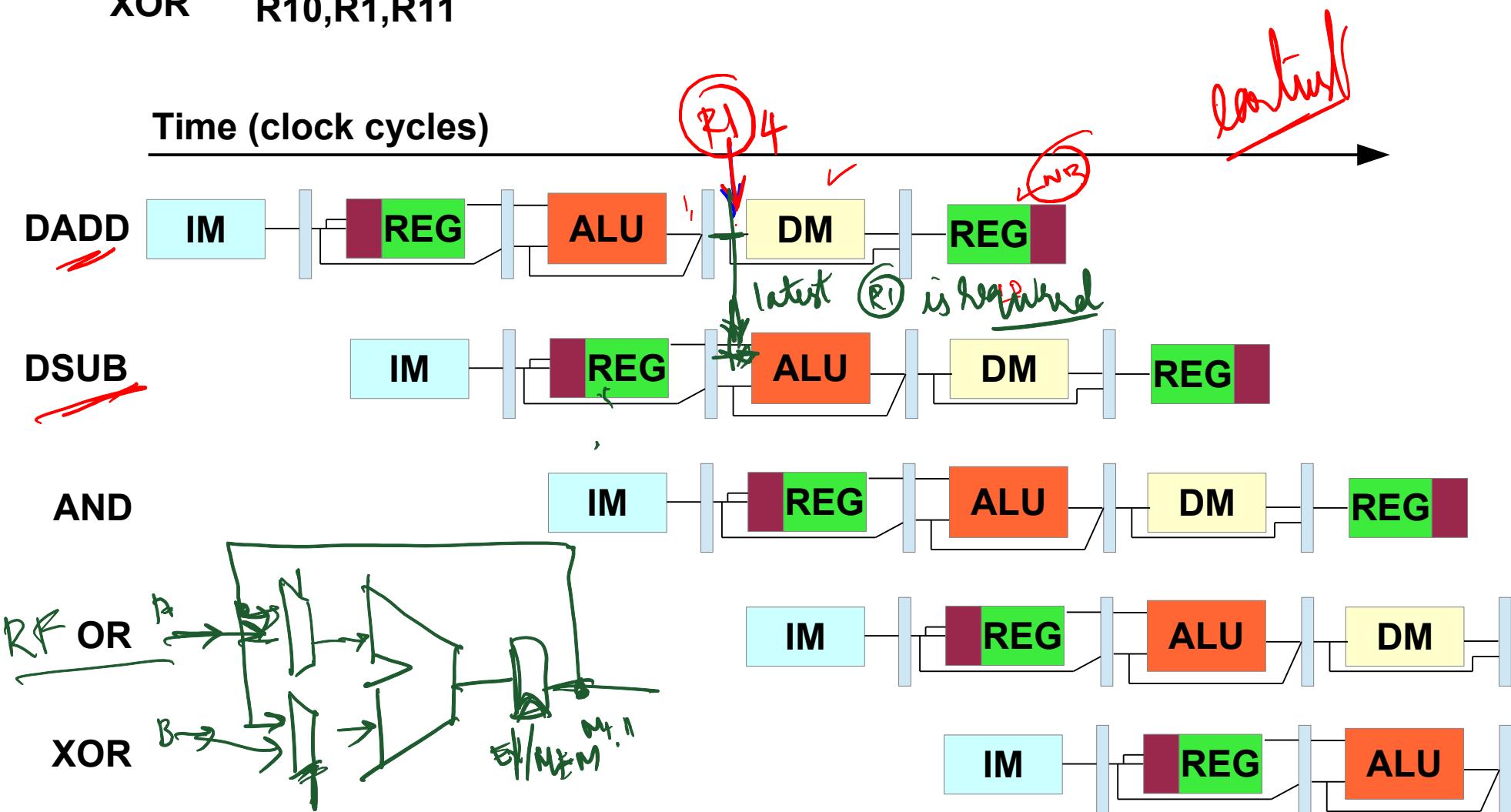
DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9
XOR	R10,R1,R11



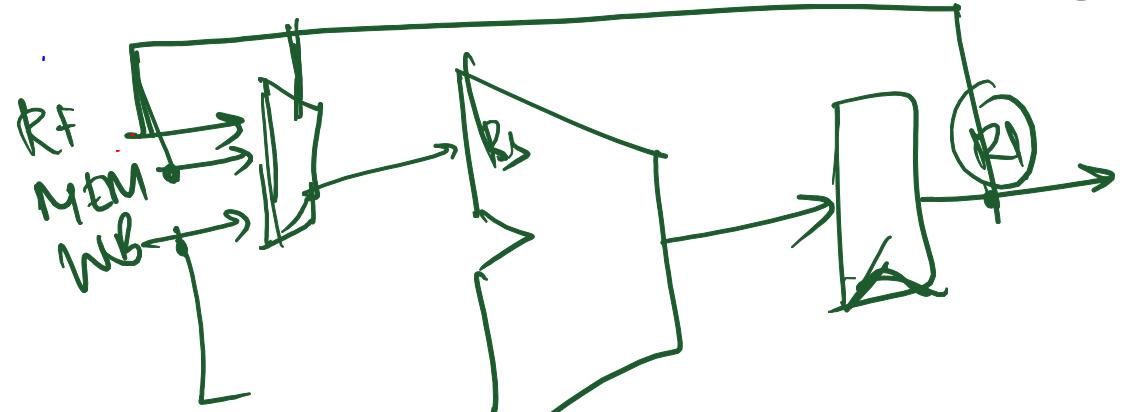
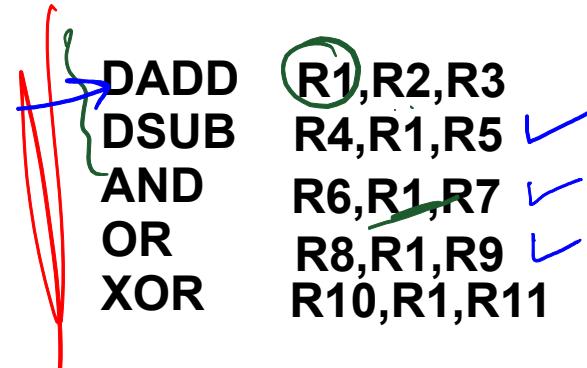
Avoiding Data Hazards – Forwarding

→ DADD	$R_d = R_1$
→ DSUB	$R_s = R_4$
AND	R_6, R_1, R_7
OR	R_8, R_1, R_9
XOR	R_{10}, R_1, R_{11}

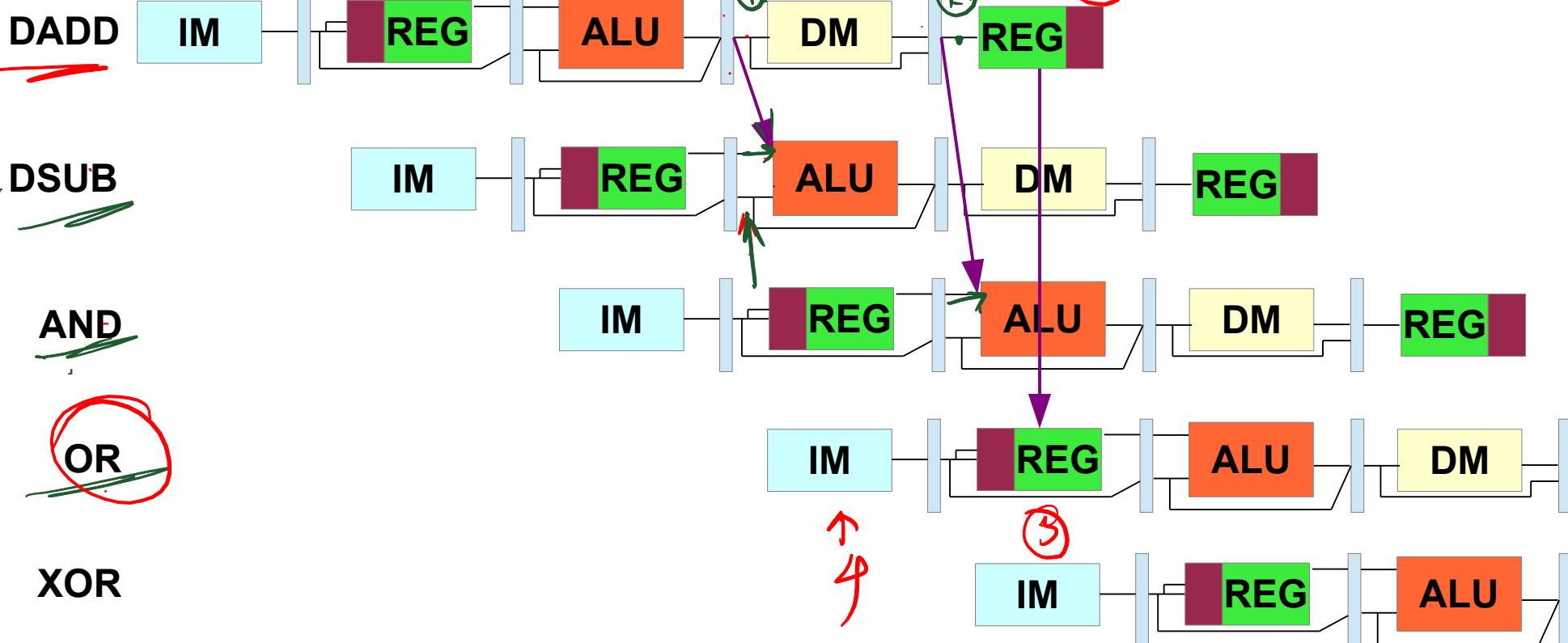
* stall the pipeline



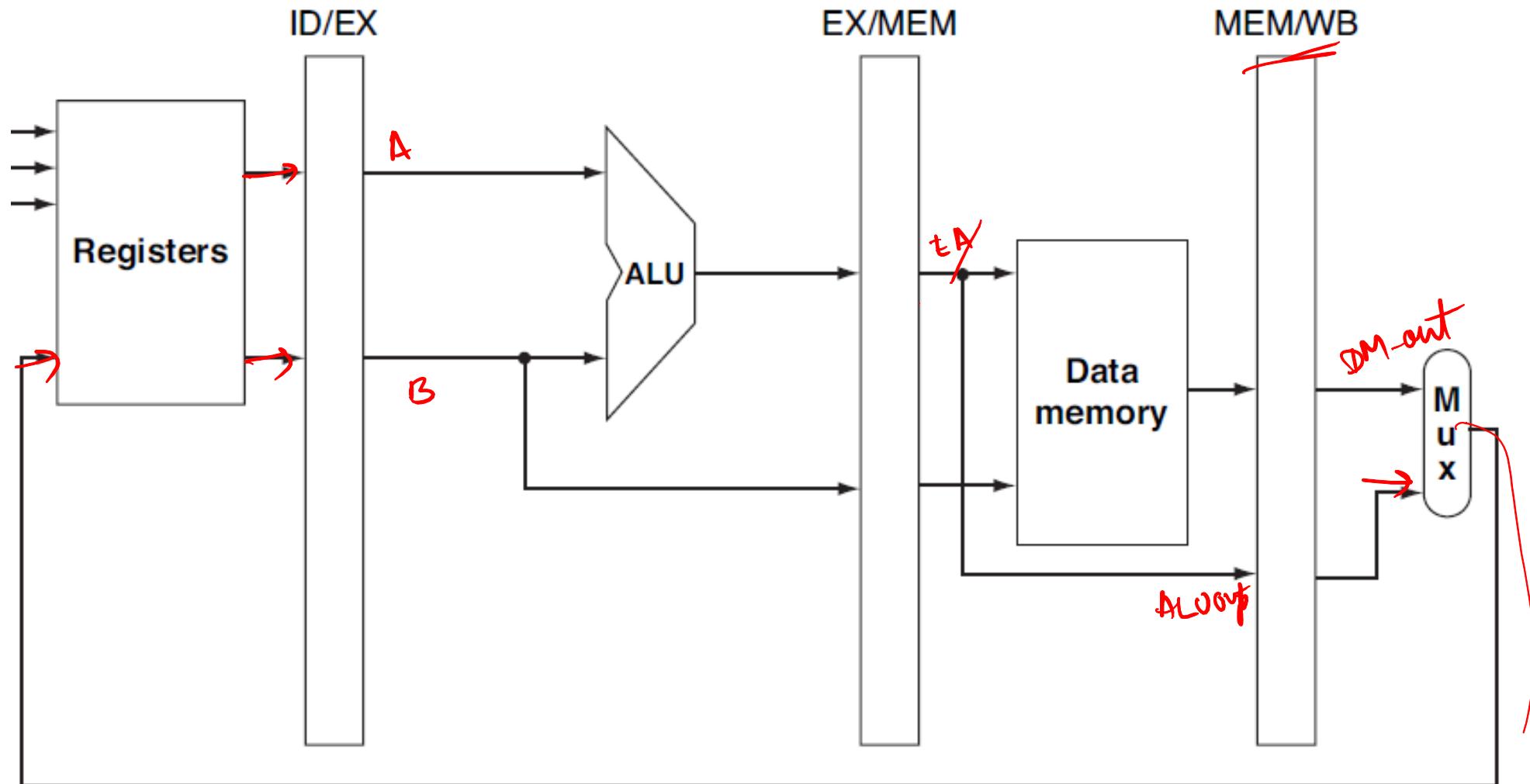
Avoiding Data Hazards – Forwarding



Time (clock cycles)



Pipeline without Forwarding



Forwarding Unit

Diagram illustrating the assembly of a stack frame:

- `add R1, -1(RI)`
- `sub RI, -1(RI)`
- `and -1(RI)`

The stack grows downwards.

~~RAW dependency~~ - ~~Input ops~~ == ~~output op.~~
stage

A hand-drawn diagram on a whiteboard. A red circle contains the text "opt". A green arrow points from this circle to a blue rectangular box. Inside the box, the text "# decode st" is written in blue ink.

Diagram illustrating memory access and data flow in a processor. The diagram shows a stack frame with local variables I_1 , I_0 , add , and sub . A pointer $R1$ points to the stack frame. The stack frame has entries F , D , X , and D . An arrow labeled "add" points to the stack frame. A register $R1$ also points to the stack frame. A memory location MEM is shown at the bottom.

A hand-drawn diagram illustrating the flow of data from the EX stage to the next stages. A large box labeled "EX stage" is shown with a red border. To its left is a red circle containing a smaller circle with an "X" and the number "2". Two blue arrows point away from the bottom right corner of the EX stage box. The top arrow points to the word "MEM" and the bottom arrow points to the letters "WB".

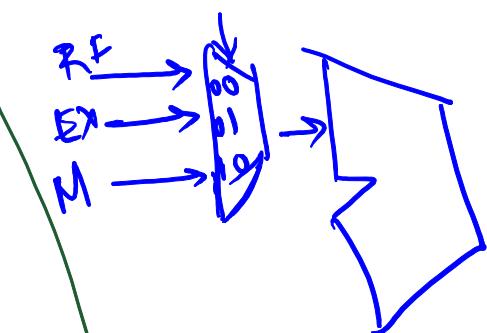
for \vec{r} in 3D:
dependent_Ex?
...-ctbl=0

~~mix~~ → dependent - MEM?
→ mix - Ctrl = 10
~~dependent~~ - WB

WB

need not be ch

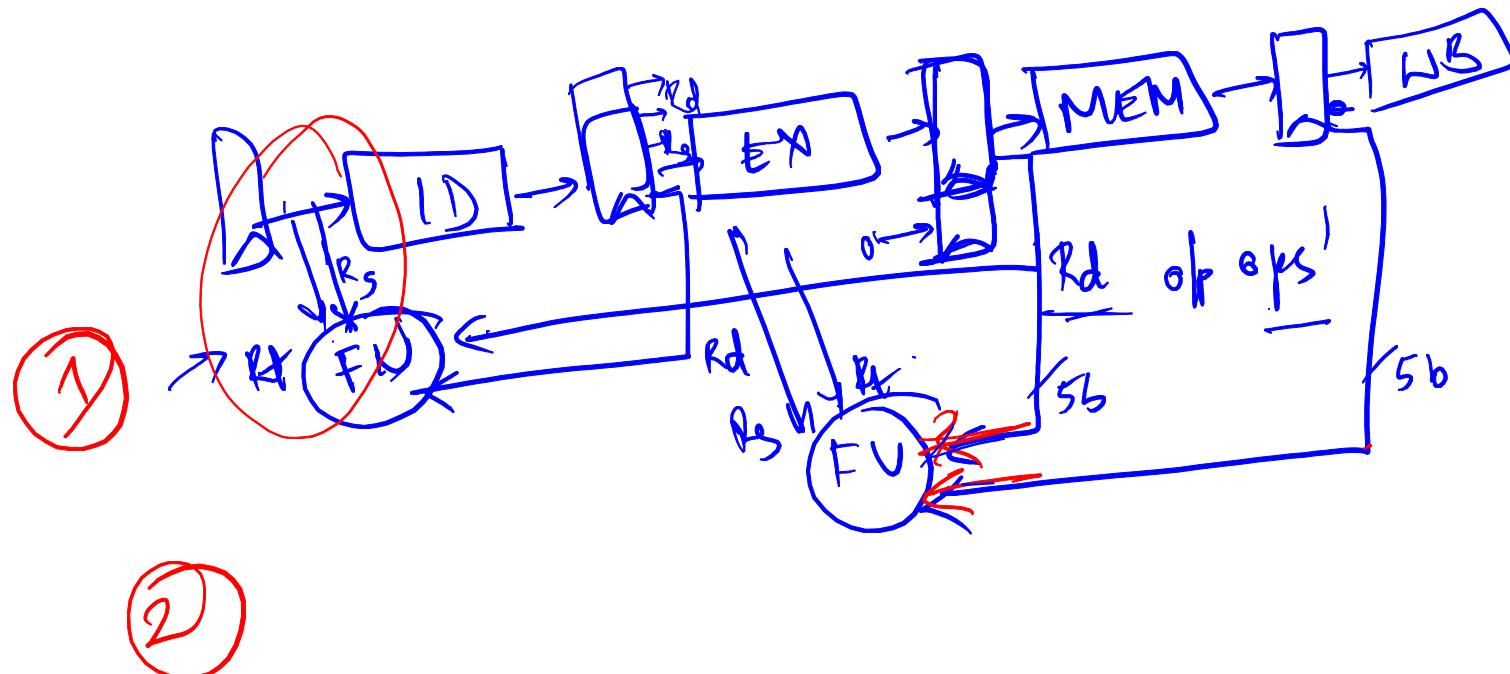
10



where

check takes large time
 \hookrightarrow dependent MEM \Rightarrow $(R_s = R_d)_M$ || $(R_{t_tx} = R_d)_M$
 ✗ $R_s = R_d$
 ✗ $R_{t_tx} = R_d$

✗ \oplus \ominus
 ✗ \oplus \ominus
 ✗ (\oplus) \ominus



Forwarding Unit

- Identify dependences between instructions executing in the pipeline

Forwarding Unit

- Identify dependences between instructions executing in the pipeline
- Supply generated output as input to a dependent instruction
 - In which stage is the newly generated output?

MEM
WB

Forwarding Unit

- Identify dependences between instructions executing in the pipeline
- Supply generated output as input to a dependent instruction
 - In which stage is the newly generated output?
- Which stage should the Forwarding Unit be in?
 - when is the latest the input operand is required?

Forwarding Unit Logic

DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7

- DADD is in MEM; DSUB is in EX.

Forwarding Unit Logic

DADD	R1,R2,R3
<u>DSUB</u>	R4,R1,R5
AND	R6,R1,R7

- DADD is in MEM; DSUB is in EX.
- Input Ops of DSUB = Output Op of DADD?


Forwarding Unit Logic

DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7

- DADD is in MEM; DSUB is in EX.
- Input Ops of DSUB = Output Op of DADD?
 - $(\underline{\text{DSUB}_{Rs}} == \text{DADD}_{Rd}) \text{ OR } (\underline{\text{DSUB}_{Rt}} == \text{DADD}_{Rd})$
 - Before passing input ops to the ALU → *wait after*
-

Forwarding Unit Logic

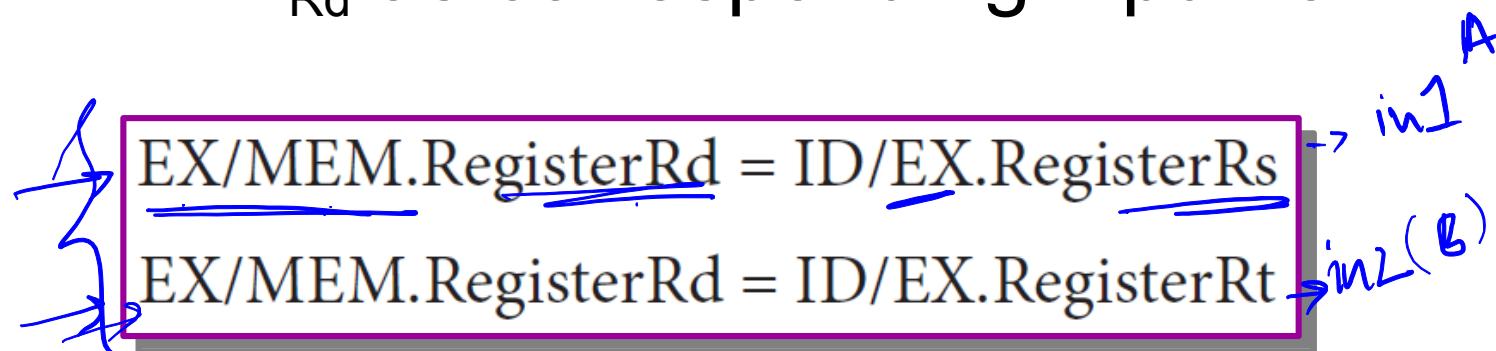
DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7

- DADD is in MEM; DSUB is in EX.
- Input Ops of DSUB = Output Op of DADD?
 - $(DSUB_{Rs} == DADD_{Rd}) \text{ OR } (DSUB_{Rt} == DADD_{Rd})$
 - Before passing input ops to the ALU
- If true, feed $\underline{\underline{DADD_{Rd}}}$ as corresponding input to the ALU

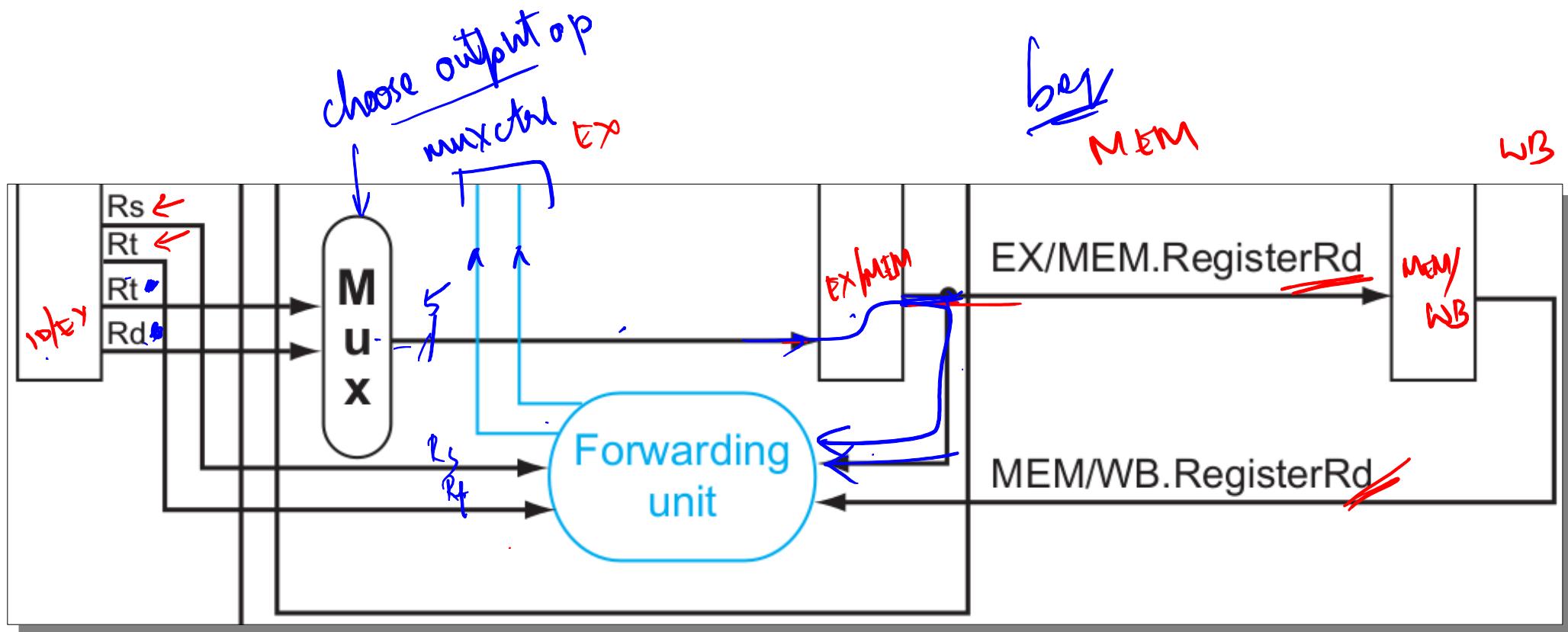
Forwarding Unit Logic

DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7

- DADD is in MEM; DSUB is in EX.
- Input Ops of DSUB = Output Op of DADD?
 - $(DSUB_{Rs} == DADD_{Rd}) \text{ OR } (DSUB_{Rt} == DADD_{Rd})$
 - Before passing input ops to the ALU
- If true, feed $DADD_{Rd}$ as corresponding input to the ALU

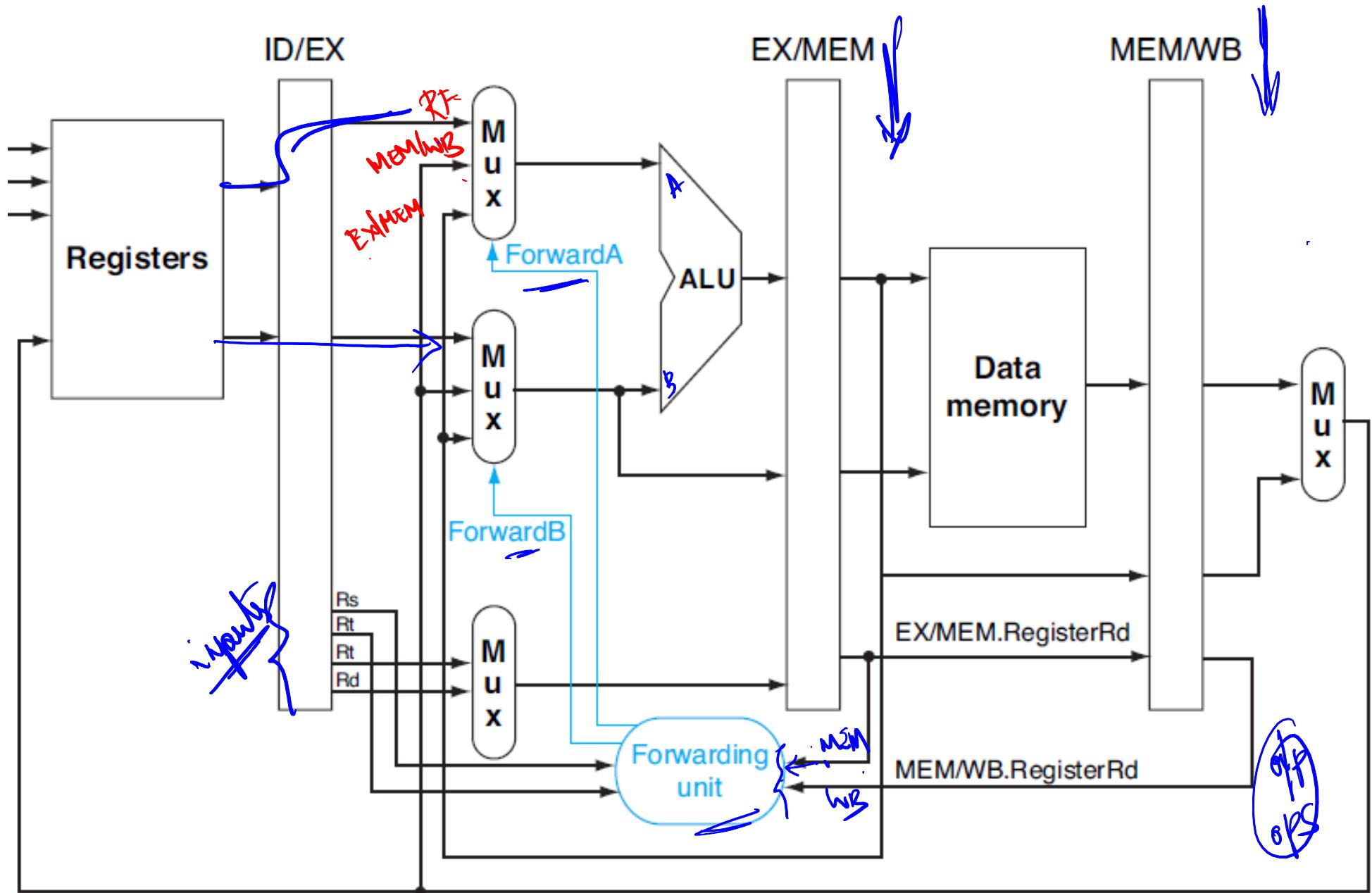


Forwarding Unit – Inputs

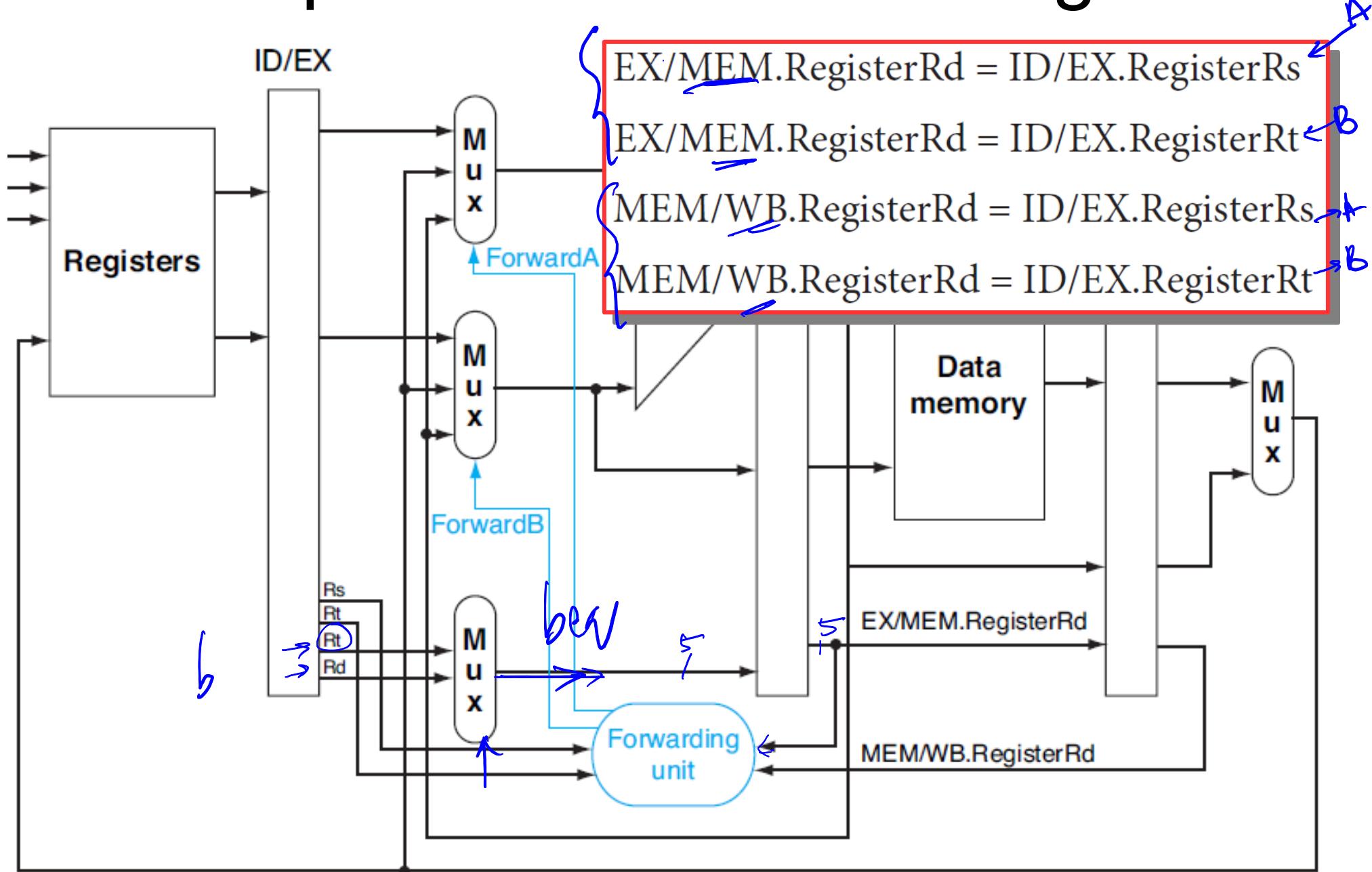


$Rt \rightarrow I\text{-type}$
 $Rd \rightarrow R\text{-type}$

Pipeline with Forwarding

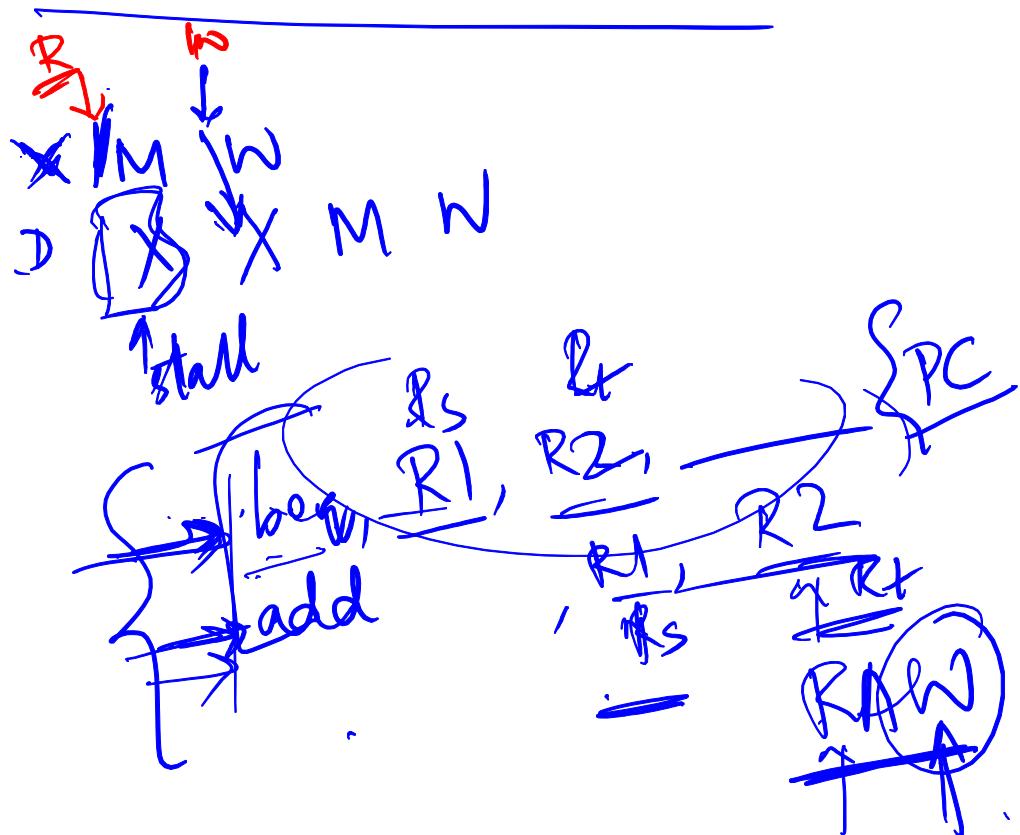
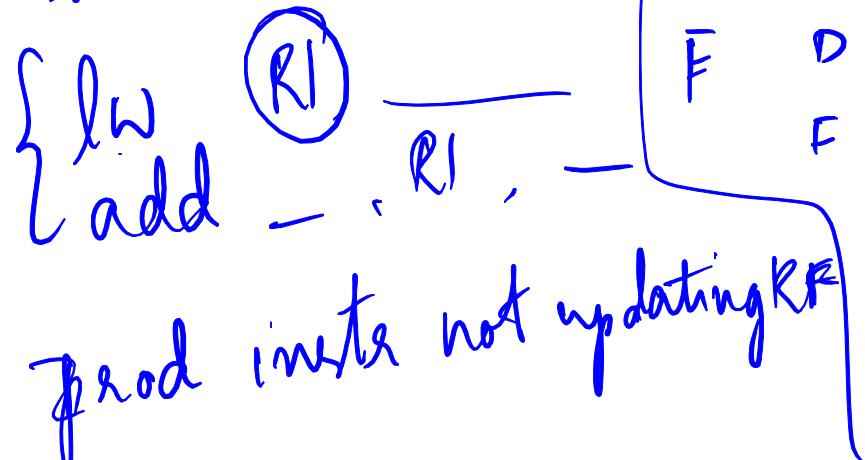


Pipeline with Forwarding



Forwarding – Corner Cases

* load word



Forwarding – Corner Cases

- Not all instructions write to the RF
 - Output operands are invalid

Forwarding – Corner Cases

- Not all instructions write to the RF
 - Output operands are invalid
- Check if instruction updates RF
 - RegWrite signal

Forwarding – Corner Cases

- Not all instructions write to the RF
 - Output operands are invalid
- Check if instruction updates RF
 - RegWrite signal
- Forward only if:

```
EX/MEM.RegisterWrite == True  
MEM/WB.RegisterWrite == True
```

Forwarding – Corner Cases

```
sll $0, $1, $2  
add $4, $4, $0
```

Forwarding – Corner Cases

```
sll $0, $1, $2  
add $4, $4, $0
```

- Use of \$0 as an operand must yield a value of 0

Forwarding – Corner Cases

```
sll $0, $1, $2  
add $4, $4, $0
```

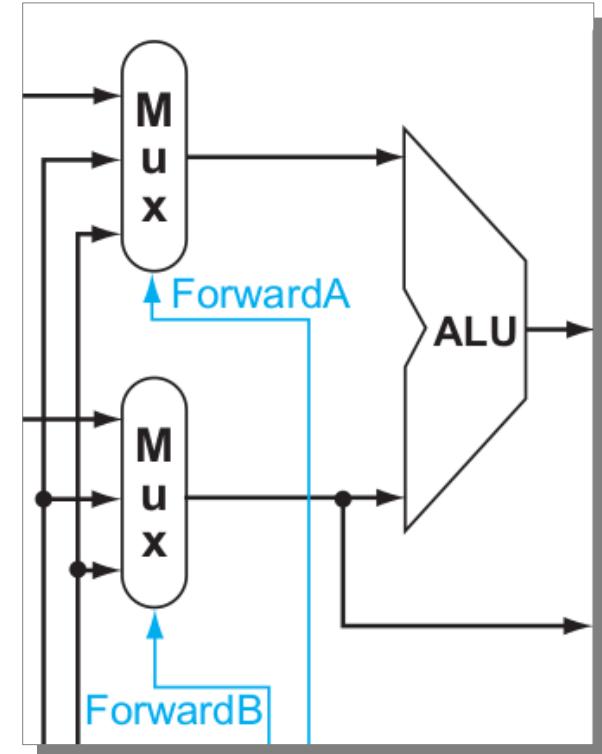
- Use of \$0 as an operand must yield a value of 0
- Forward only if:

```
EX/MEM.RegisterRd ≠ 0  
MEM/WB.RegisterRd ≠ 0
```

Forwarding Unit Logic

Input A to ALU

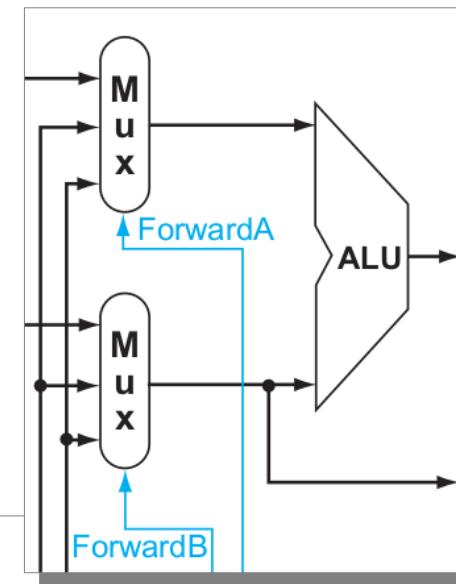
```
if (EX/MEM.RegWrite  
and (EX/MEM.RegisterRd ≠ 0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10
```



Forwarding Unit Logic

Input A to ALU

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



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.

Forwarding Unit Logic

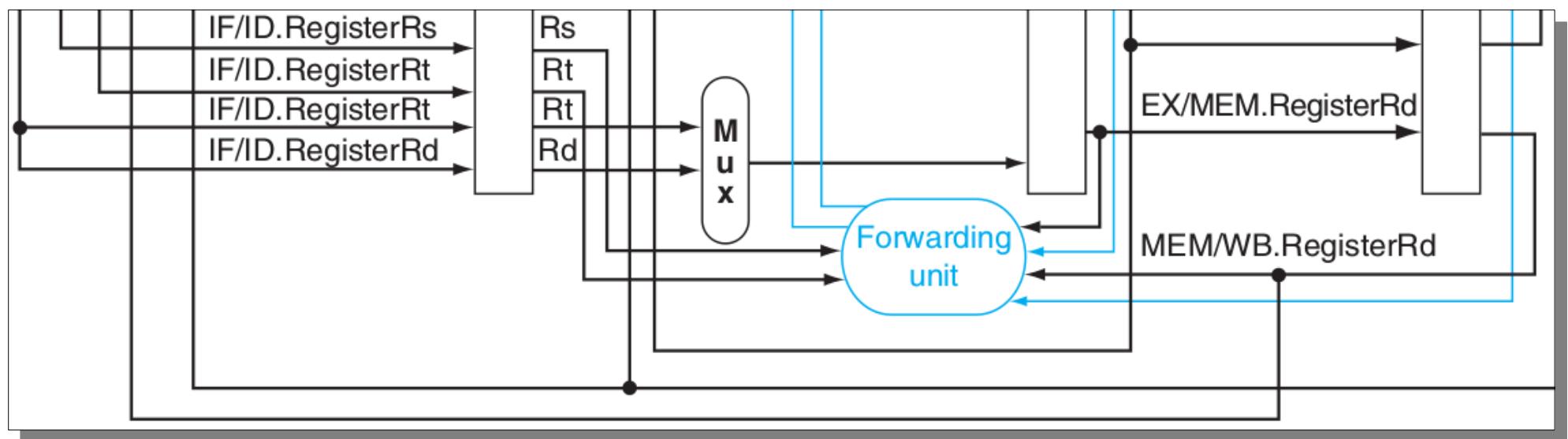
Input A to ALU

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

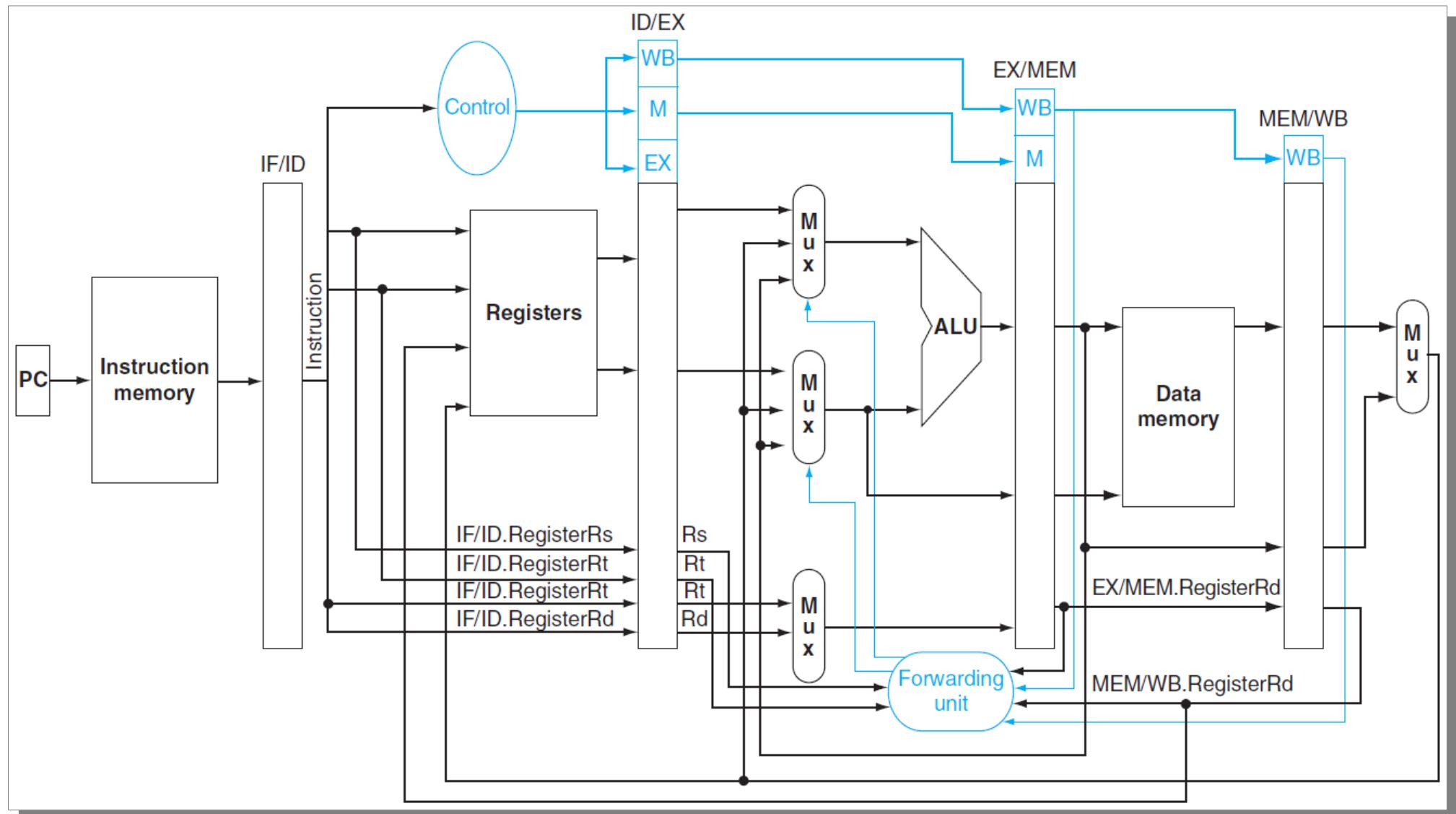
Input B to ALU

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

Forwarding Unit – Inputs



Pipeline with Forwarding Unit



Data Hazard – Load Instruction

LD R1,0(R2)

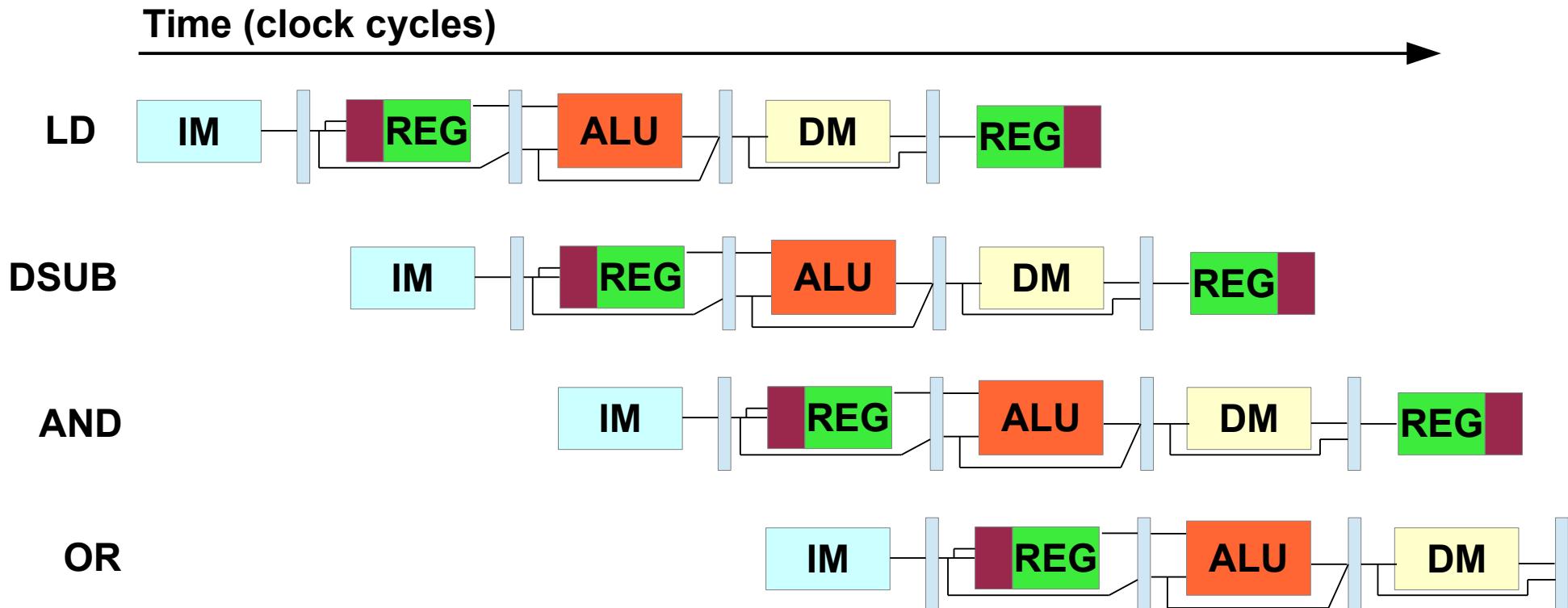
DSUB R4,R1,R5

AND R6,R1,R7

OR R8,R1,R9

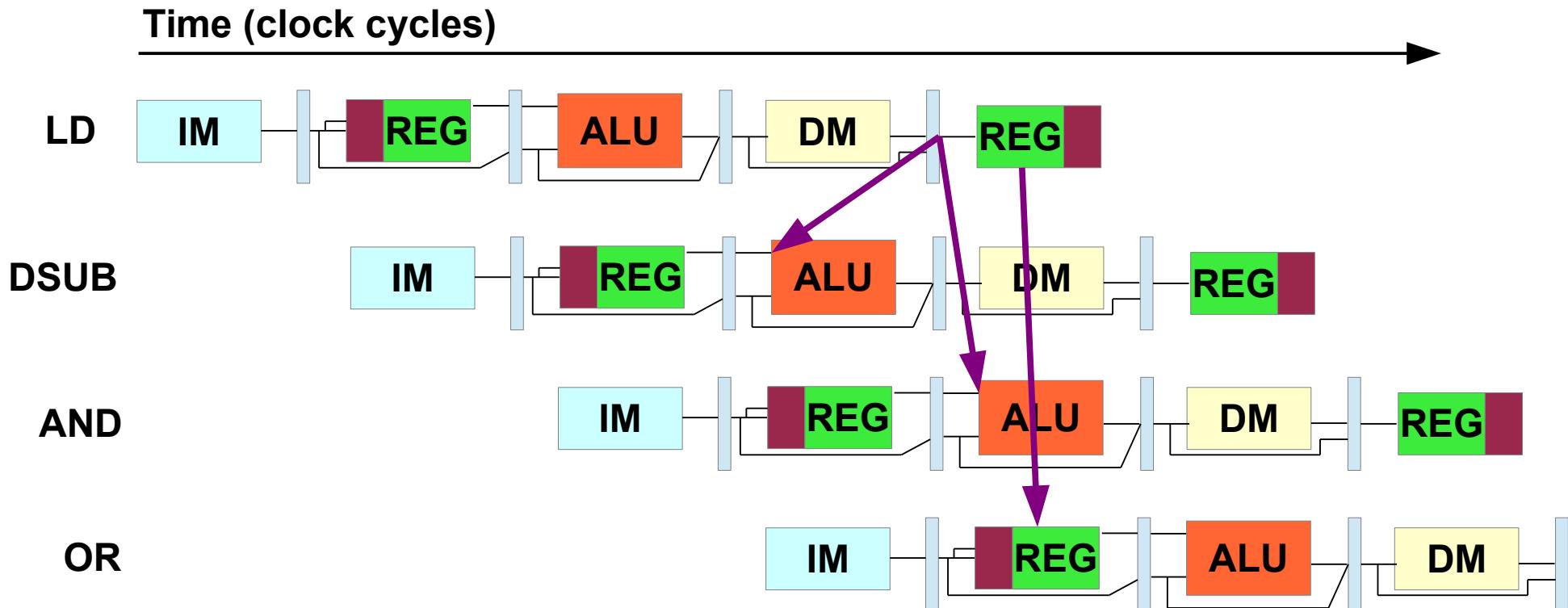
Data Hazard – Load Instruction

LD R1,0(R2)
DSUB R4,R1,R5
AND R6,R1,R7
OR R8,R1,R9



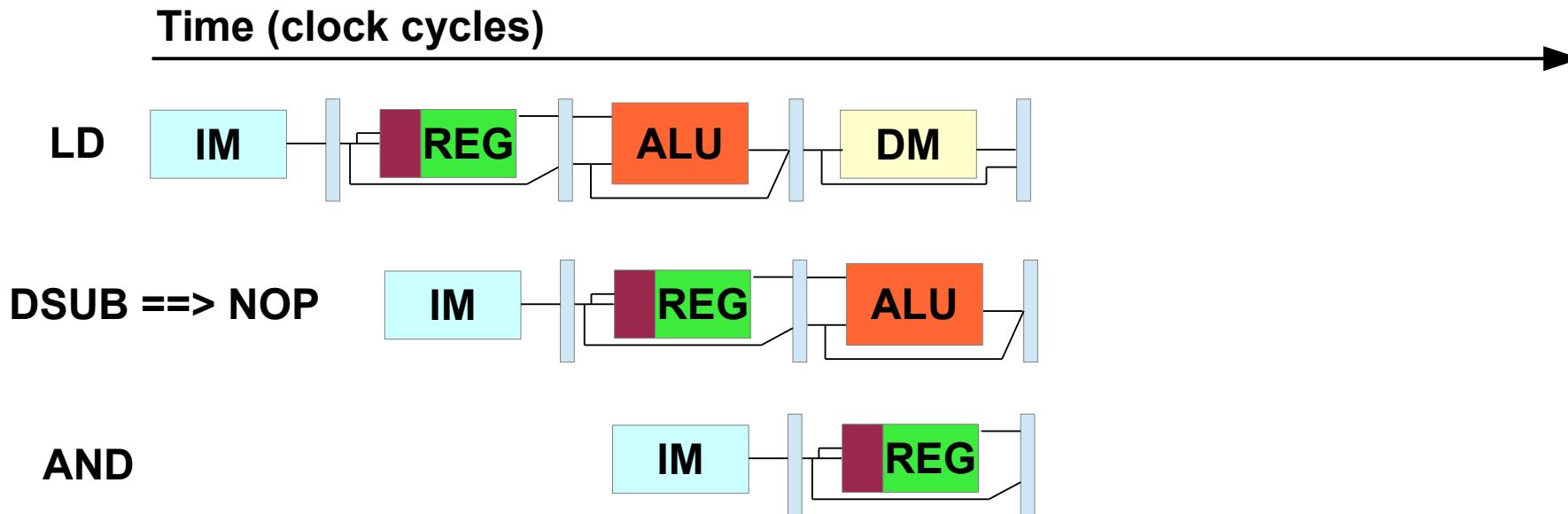
Data Hazard – Load Instruction

LD	R1,0(R2)
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9



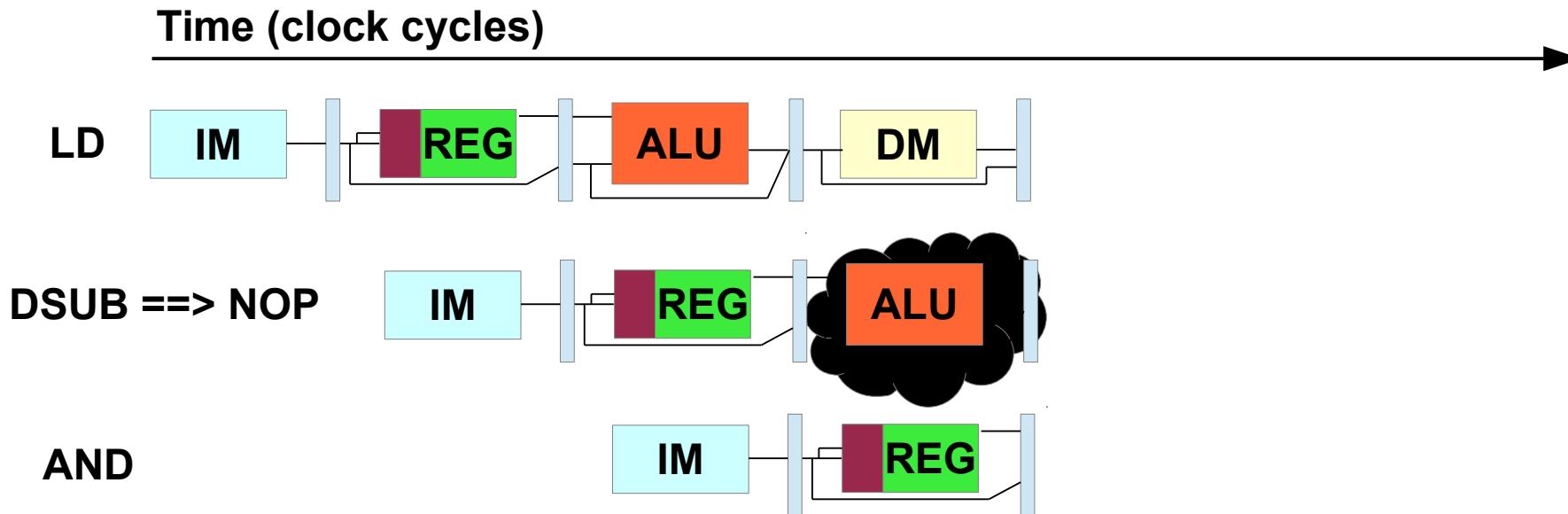
Data Hazard – Load Instruction

LD R1,0(R2)
DSUB R4,R1,R5
AND R6,R1,R7
OR R8,R1,R9



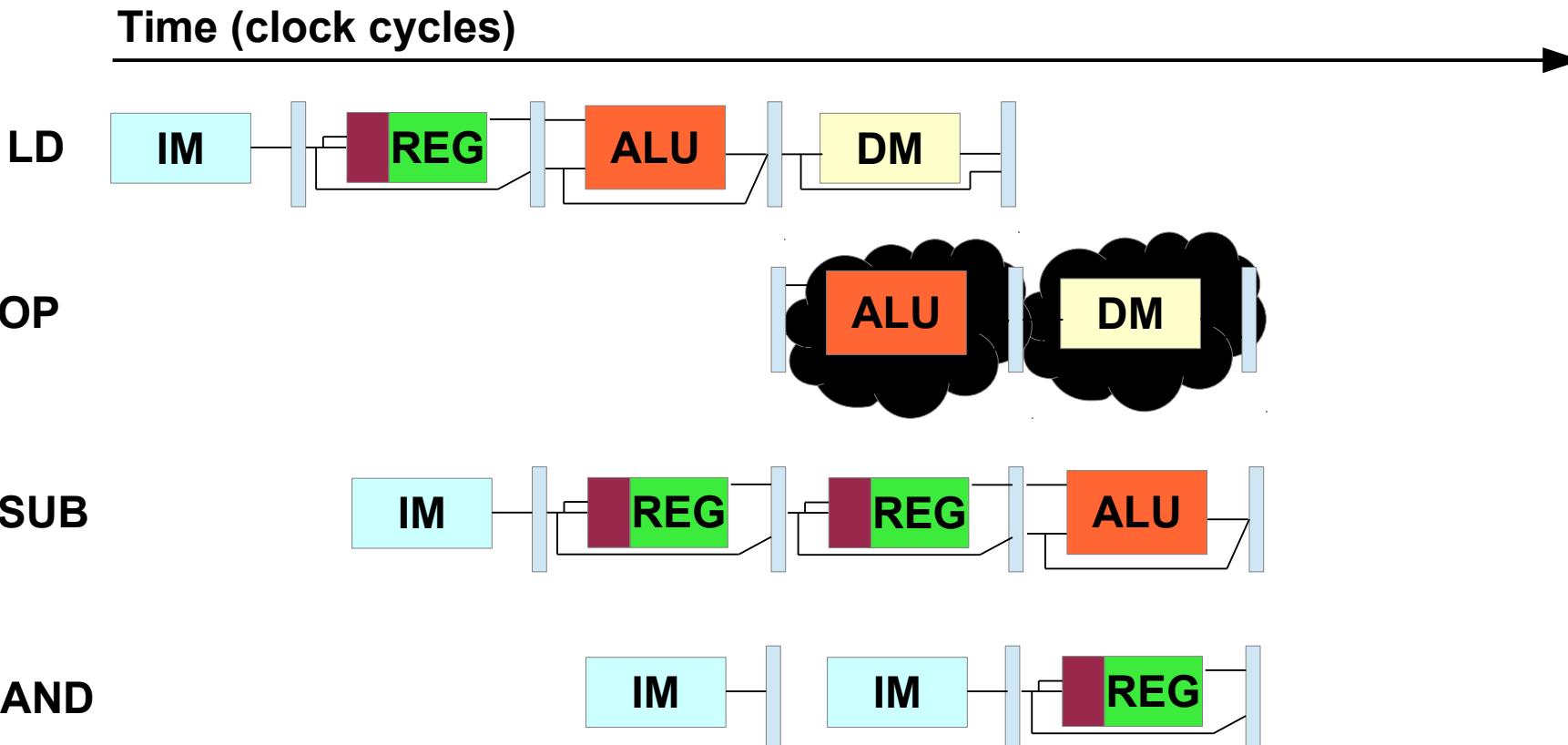
Data Hazard – Load Instruction

LD R1,0(R2)
DSUB R4,R1,R5
AND R6,R1,R7
OR R8,R1,R9



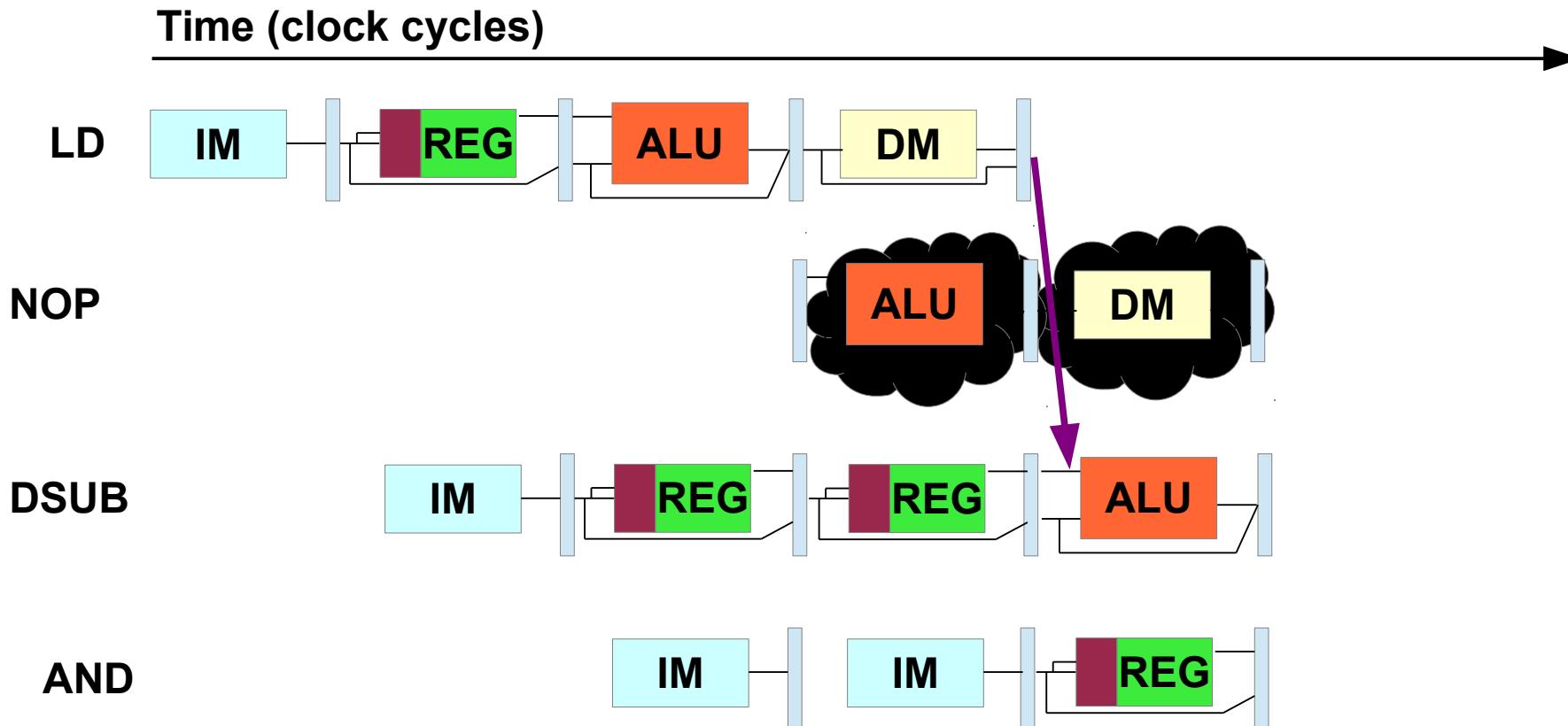
Data Hazard – Load Instruction

LD	R1,0(R2)
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9



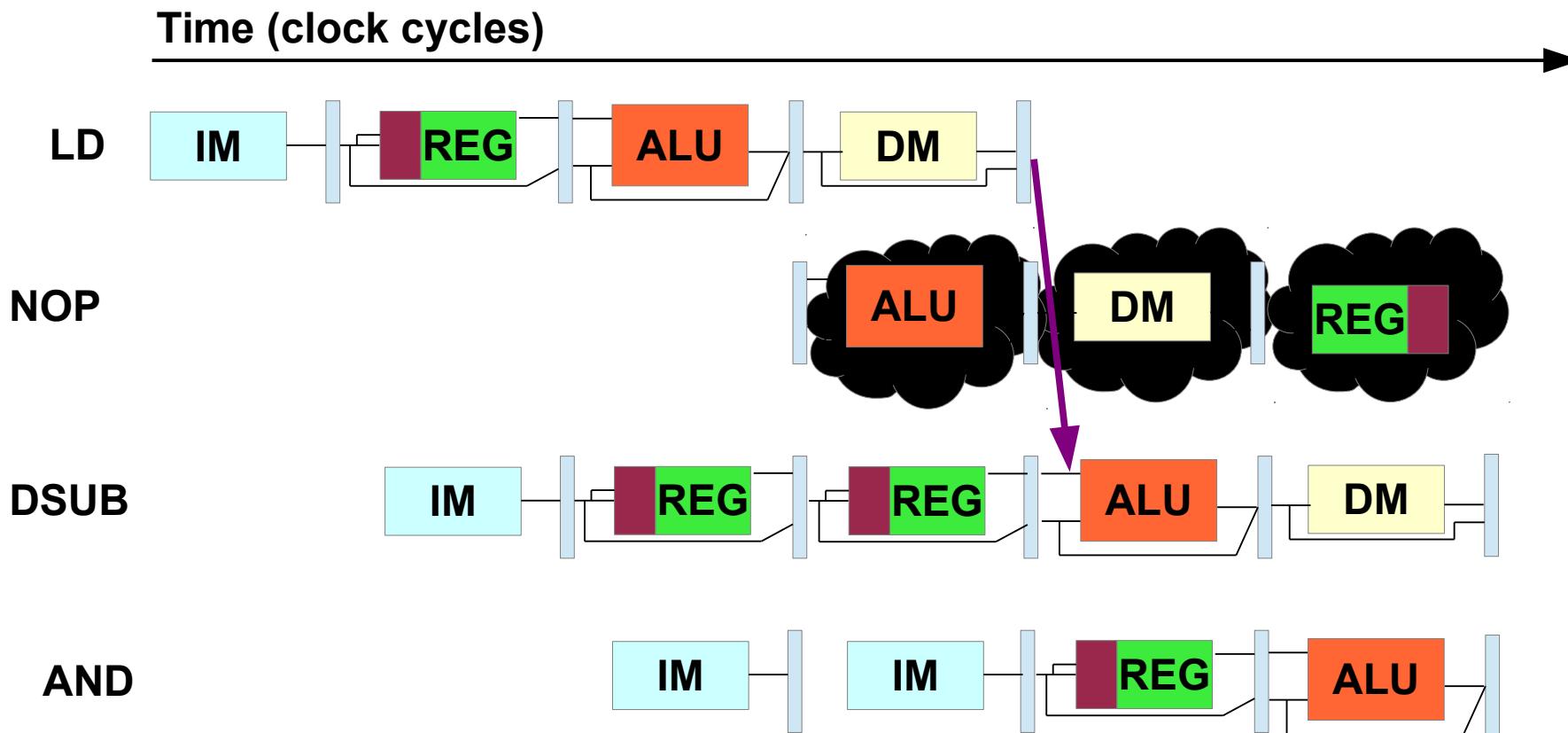
Data Hazard – Load Instruction

LD	R1,0(R2)
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9



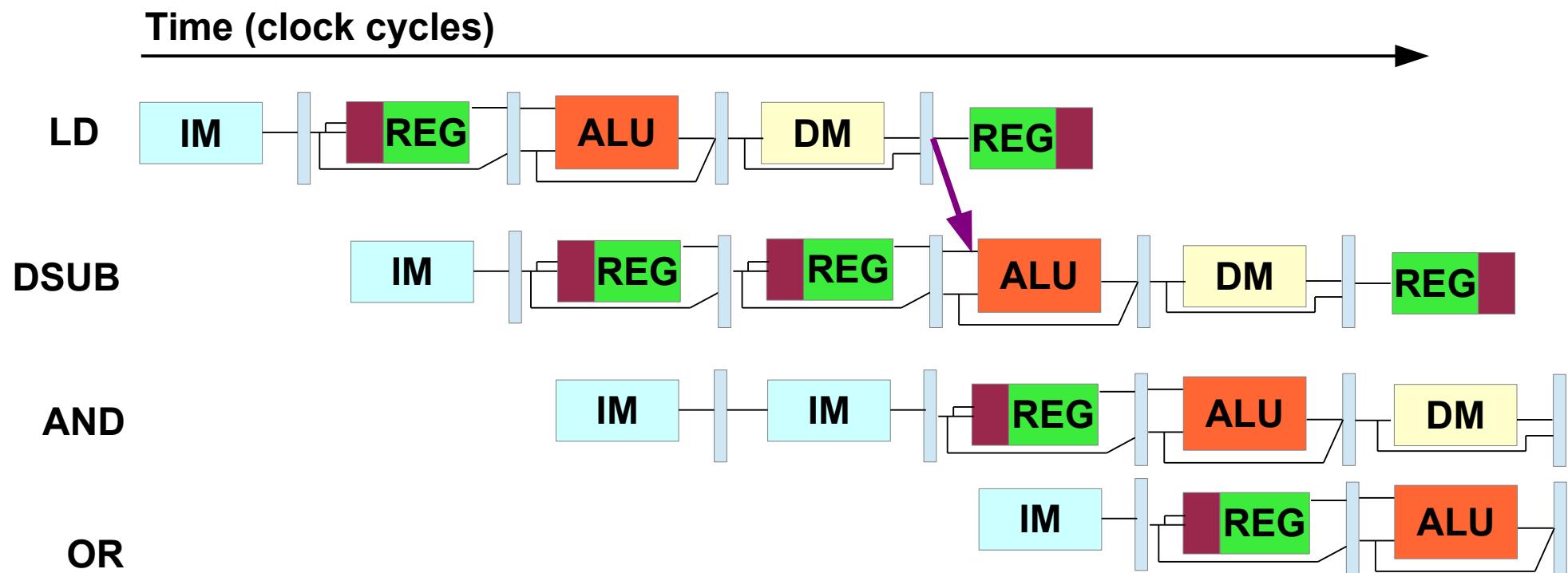
Data Hazard – Load Instruction

LD	R1,0(R2)
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9



Data Hazards – Stalls

LD	R1,0(R2)
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9



Forwarding Unit – Load Condition

Forwarding Unit – Load Condition

- If the producer instruction is a Load and the dependent (consumer) is in ALU, stall.

Forwarding Unit – Load Condition

- If the producer instruction is a Load and the dependent (consumer) is in ALU, stall.

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

Pipeline Stall

Pipeline Stall

- Input operand required latest in EX. If input op not ready, stall at EX.
 - EX can't progress until input is available

Pipeline Stall

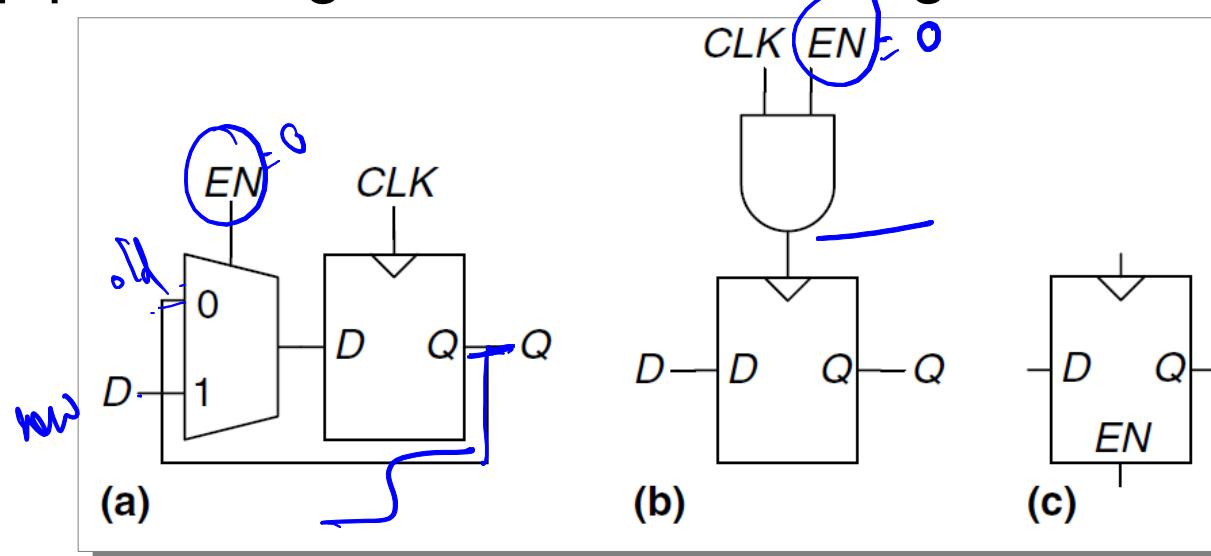
- Input operand required latest in EX. If input op not ready, stall at EX.
 - EX can't progress until input is available
- EX stalls ==> ID and IF can't progress

Pipeline Stall

- Input operand required latest in EX. If input op not ready, stall at EX.
 - EX can't progress until input is available
- EX stalls ==> ID and IF can't progress
- To stall IF and ID: prevent PC update
 - IF/ID pipeline register does not change
-

Pipeline Stall

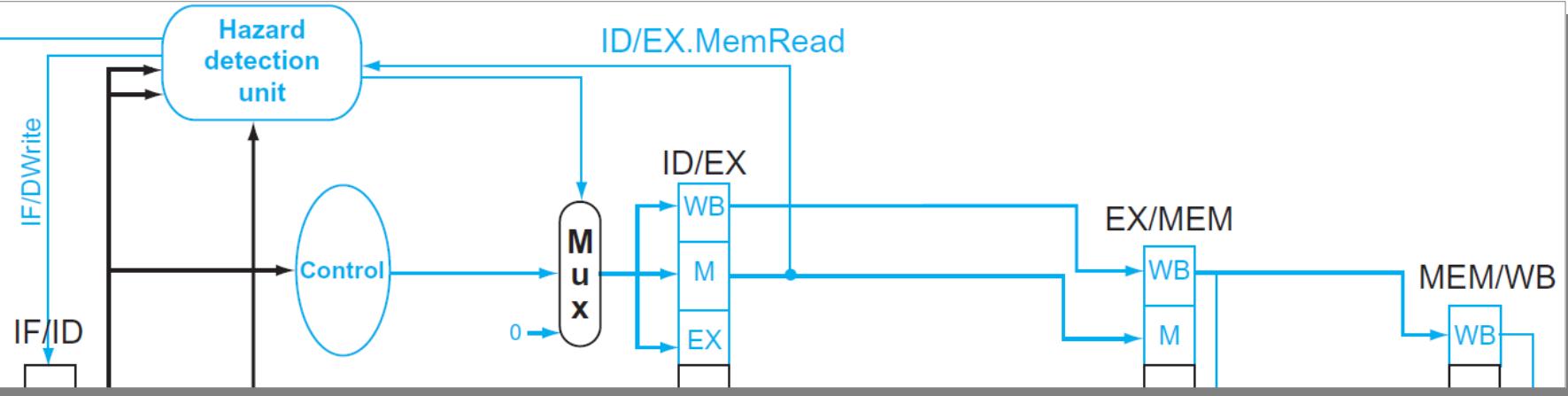
- Input operand required latest in EX. If input op not ready, stall at EX.
 - EX can't progress until input is available
- EX stalls ==> ID and IF can't progress
- To stall IF and ID: prevent PC update
 - IF/ID pipeline register does not change
-



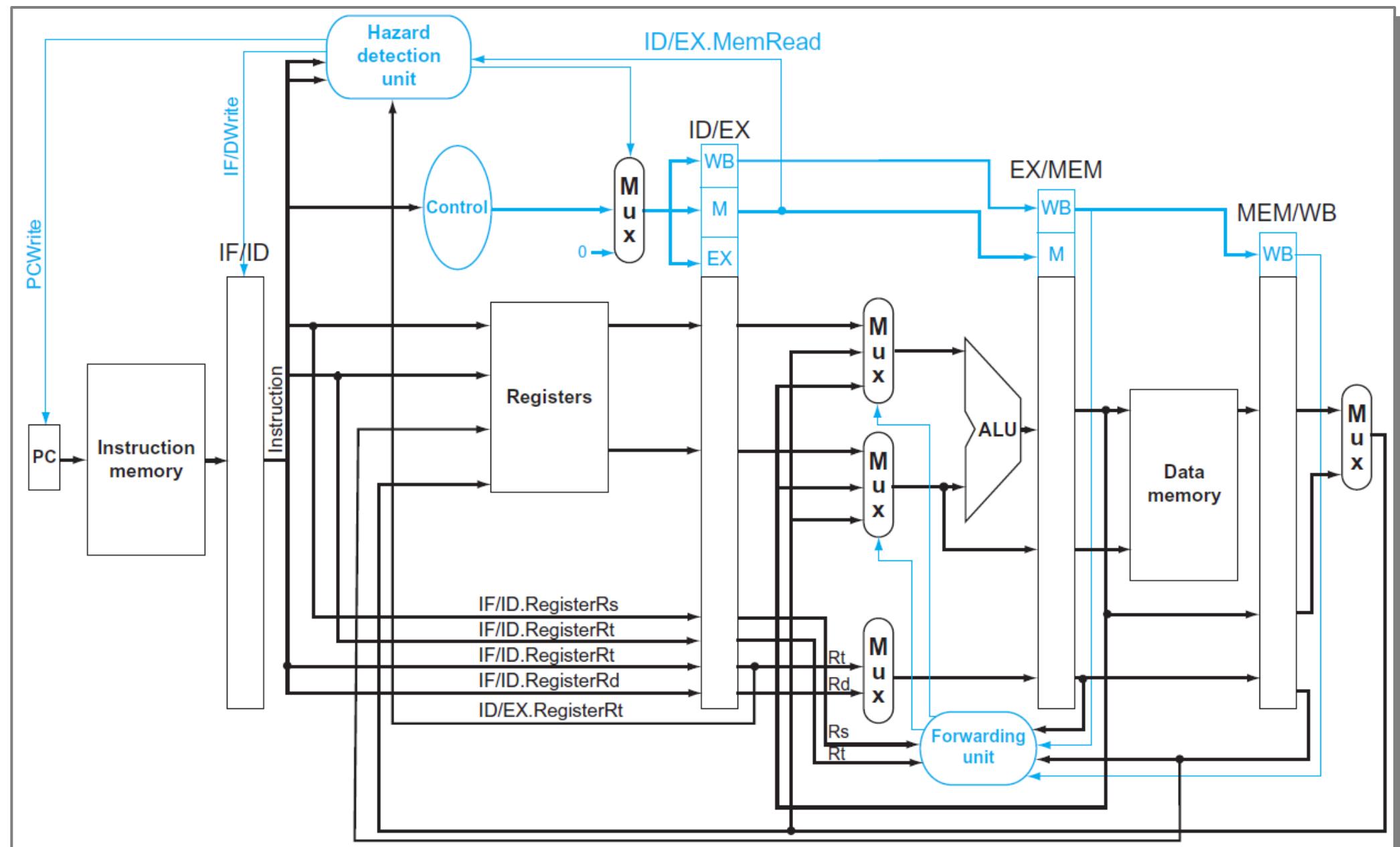
Pipeline Stall

- Input operand required latest in EX. If input op not ready, stall at EX.
 - EX can't progress until input is available
- EX stalls ==> ID and IF can't progress
- To stall IF and ID
 - prevent PC update
 - IF/ID pipeline register does not change
- EX stage instruction should have no effect
 - NOP
 - Deasserting all nine control signals in the EX, MEM, and WB stages

Hazard Detection Unit



The Pipeline



Data Hazard – Solutions

- Data Forwarding
- Instruction Reordering

Data Hazard – Solutions

- Data Forwarding
- Instruction Reordering

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add  $t3, $t1,$t2
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add  $t5, $t1,$t4
sw    $t5, 16($t0)
```

Data Hazard – Solutions

- Data Forwarding
- Instruction Reordering

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add   $t3, $t1,$t2
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add   $t5, $t1,$t4
sw    $t5, 16($t0)
```



```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
lw    $t4, 8($t0)
add   $t3, $t1,$t2
sw    $t3, 12($t0)
add   $t5, $t1,$t4
sw    $t5, 16($t0)
```

Outline

- Why Pipeline?
 - How to pipeline?
- Speedup of the pipeline
- Pipelined datapath
 - Execution of instructions
 - Pipeline Timing diagram
- Dependences, Hazards
 - Structural, Data - Stalling, Forwarding
- Control Hazards, Branch prediction