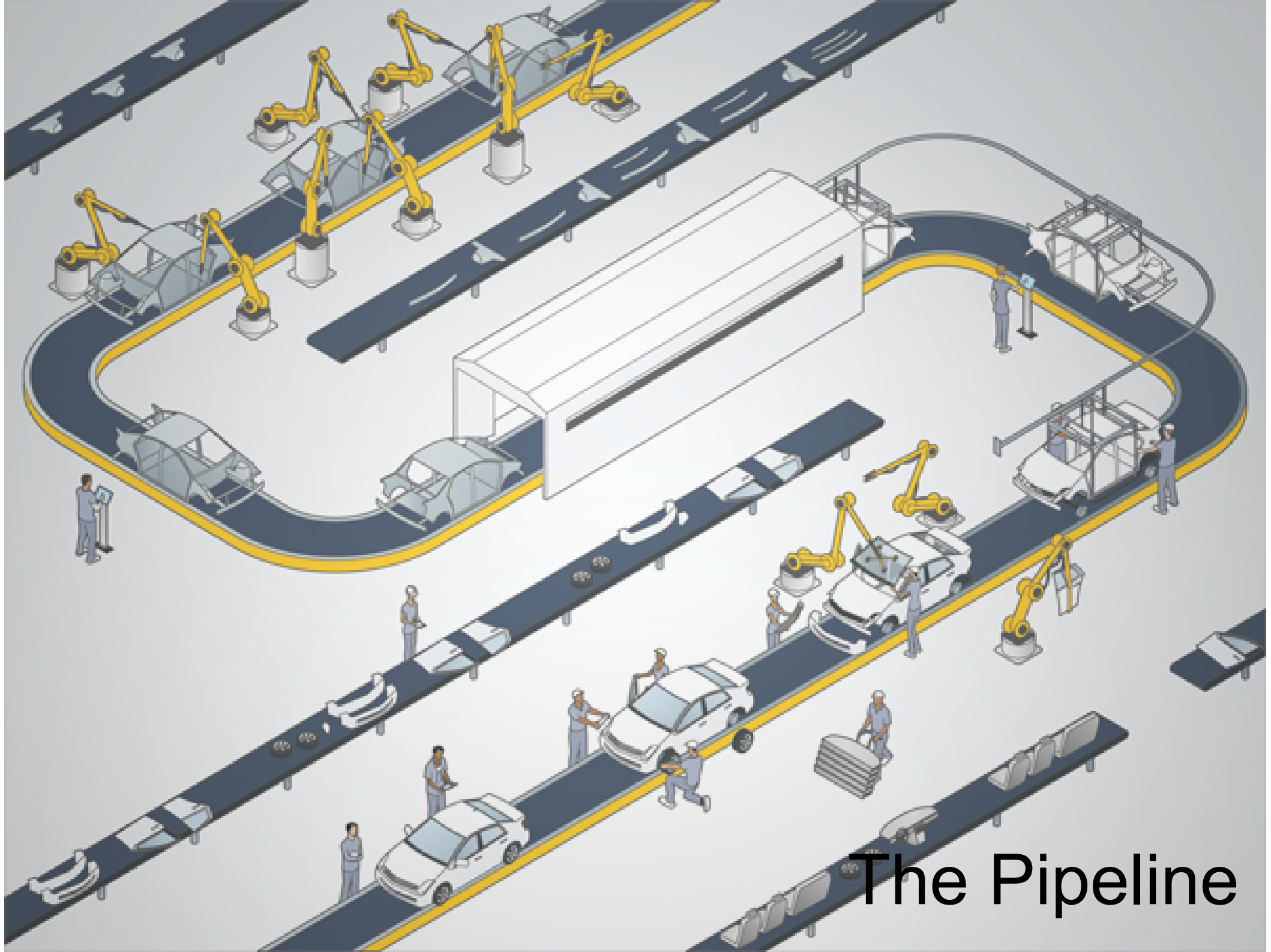
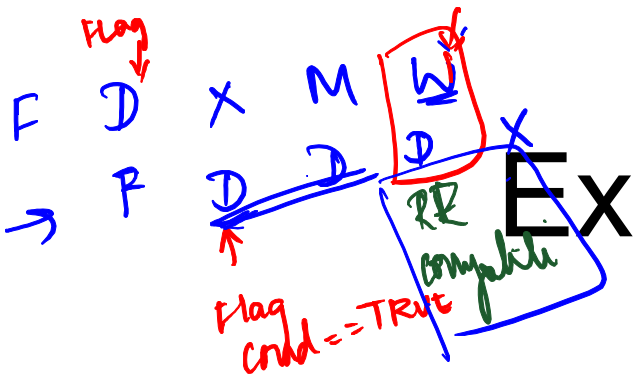

The Pipeline

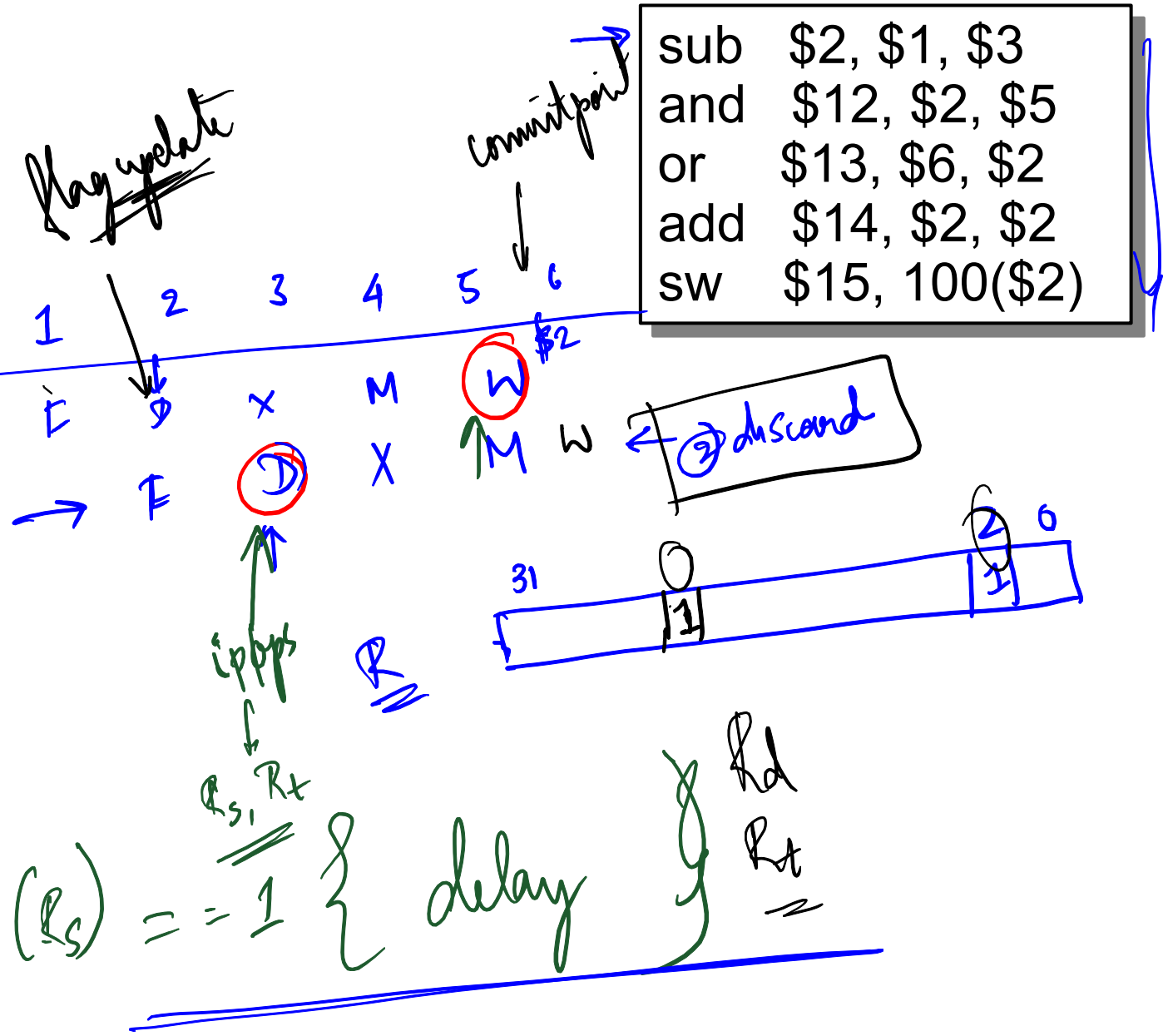
Outline

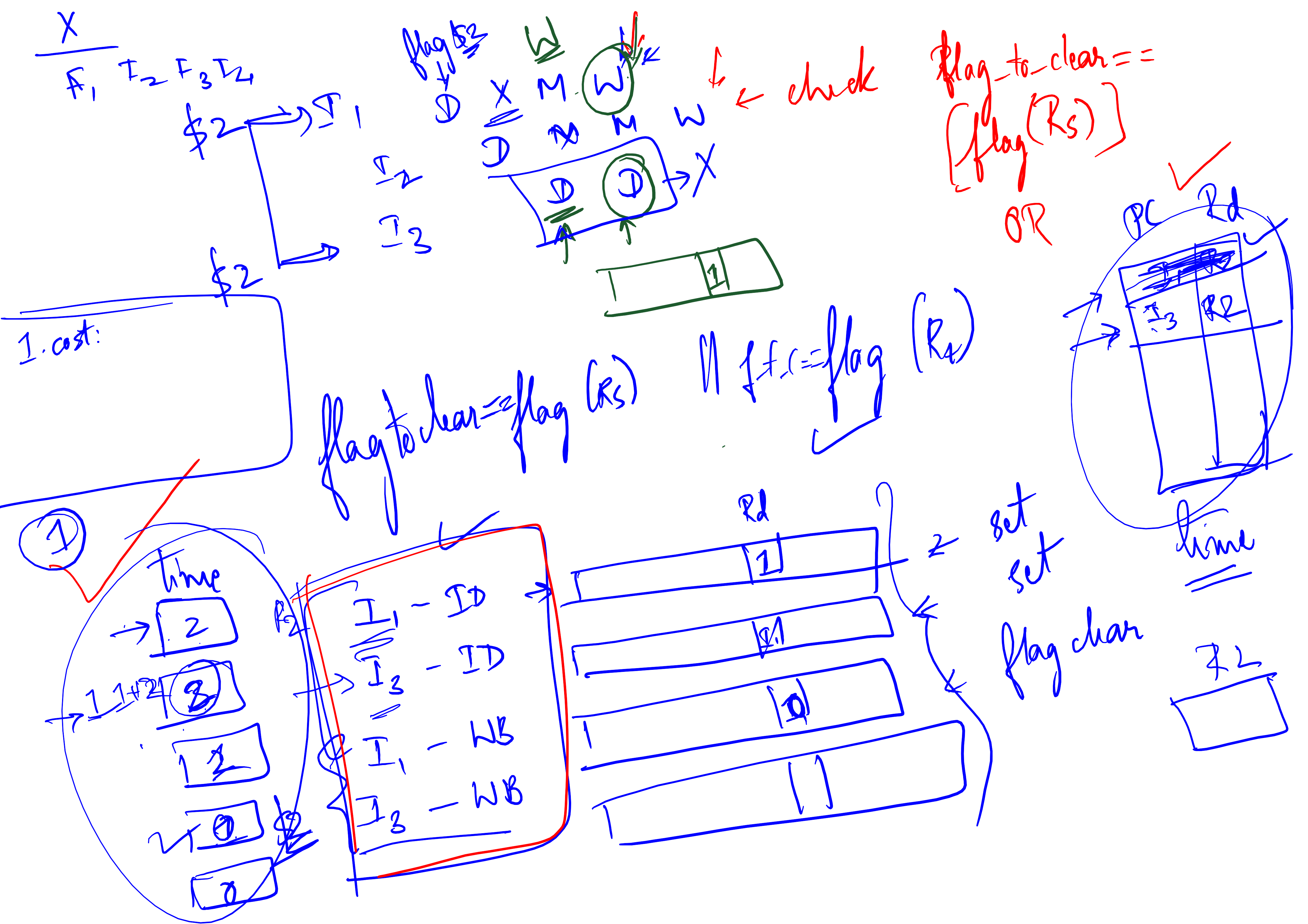
- Why Pipeline?
  - How to pipeline?
- Speedup of the pipeline
- Pipelined datapath
  - Execution of instructions
  - Pipeline Timing diagram
- Dependences, Hazards
  - Structural, Data - Stalling, Forwarding
- Control Hazards, Branch prediction

Execution Sequence




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

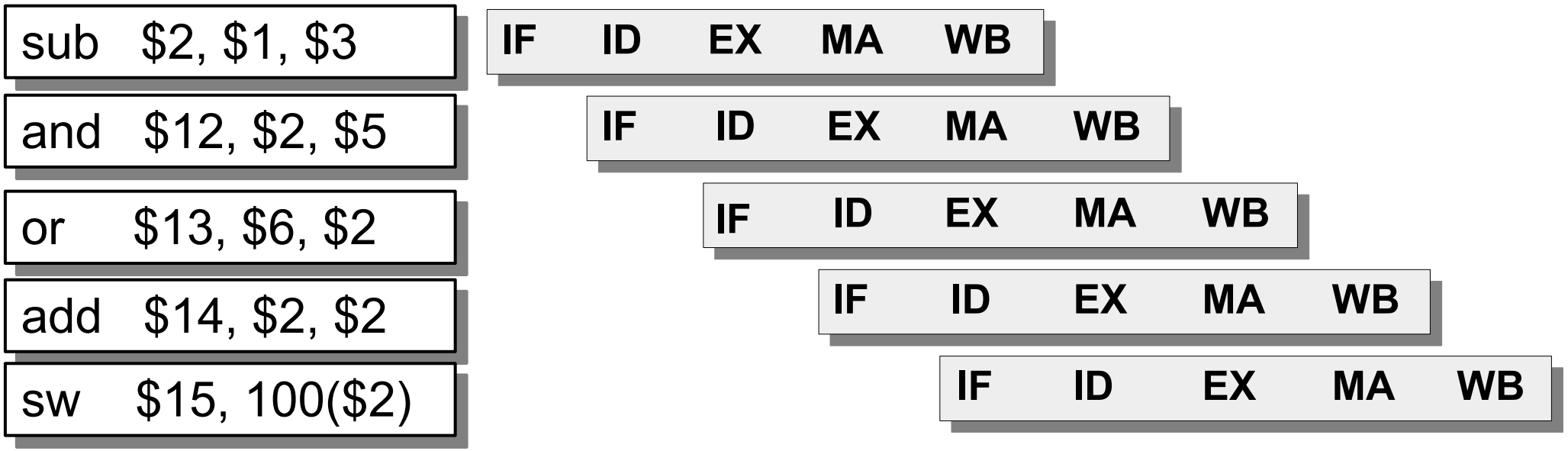




check
flag_to_clear == [flag(Rs)]
OR
1. cost:
flag_to_clear==flag (Rs) || f.t.c==flag (Rt)
time
I1 - ID
I3 - ID
I1 - WB
I3 - WB
Rd
set
set
flag clear
PC
Rd
I3
R2
time

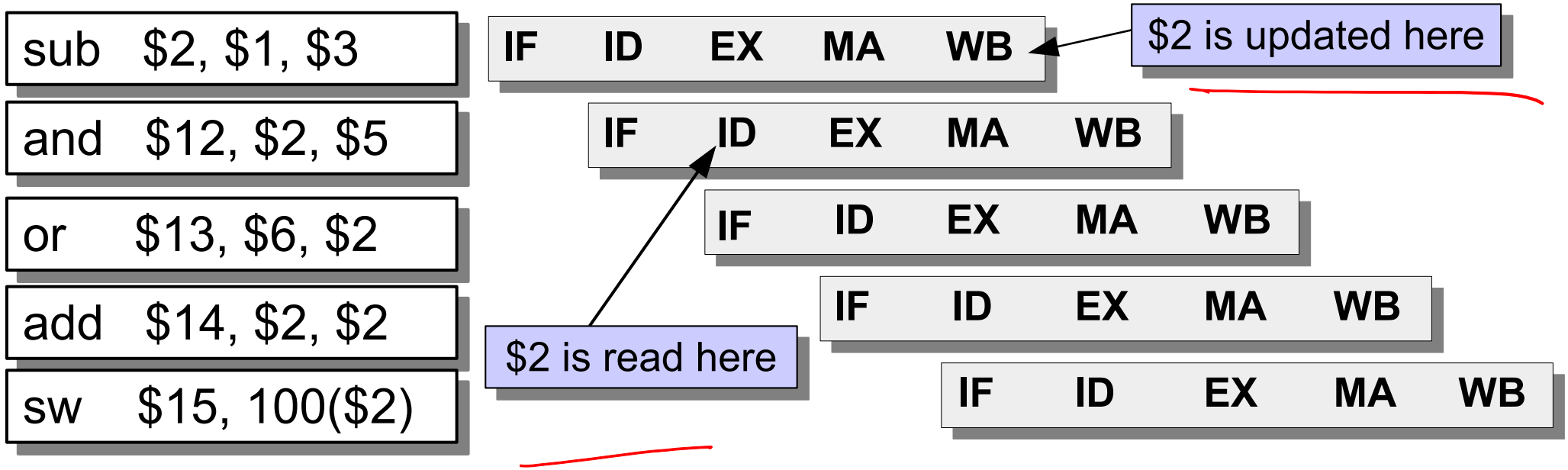
Execution Sequence

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



Execution Sequence – Dependence

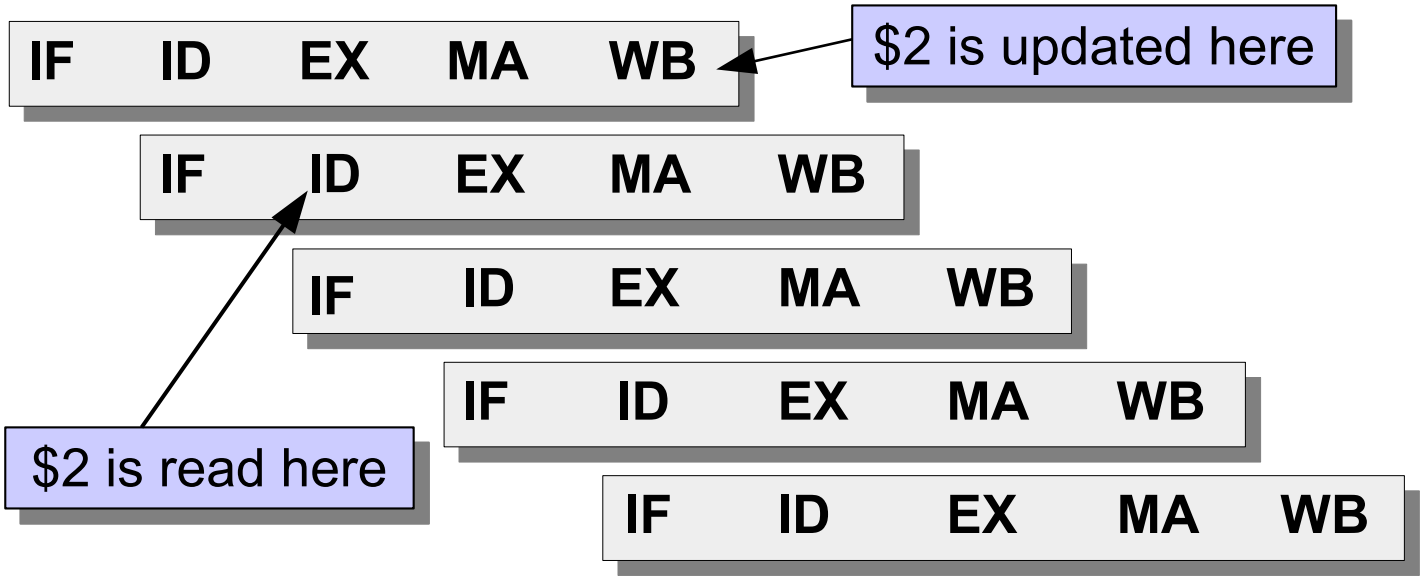
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```



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)
```



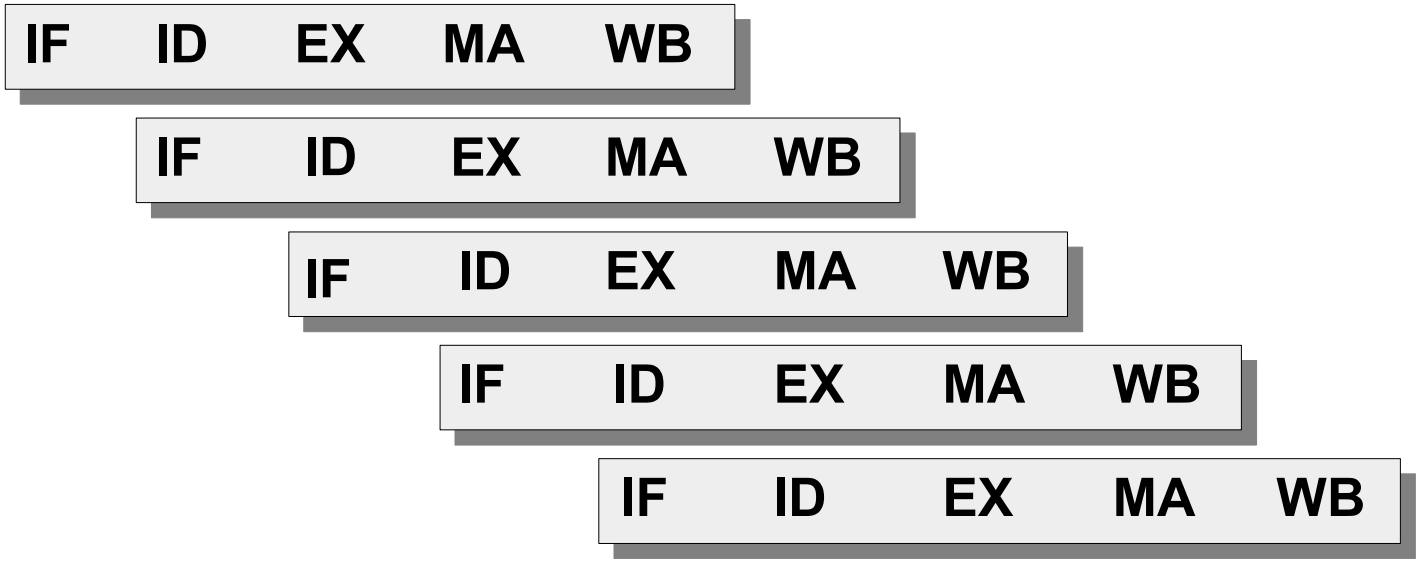
Execution Sequence – Dependence

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)
```

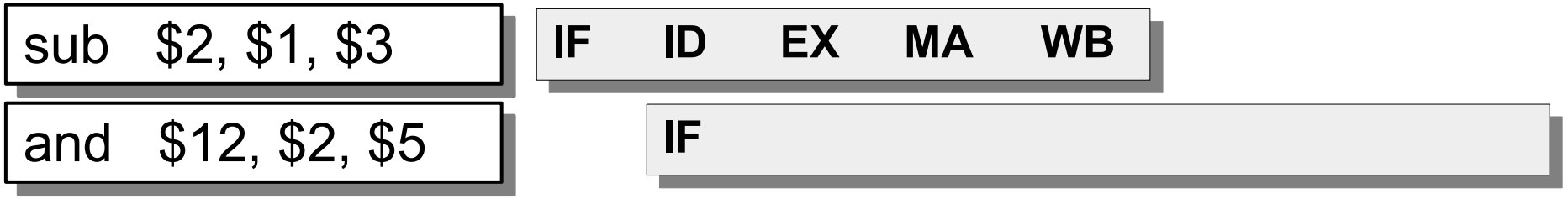
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```



Execution Sequence – Dependence

and instruction should
read $2 **after** sub **writes**

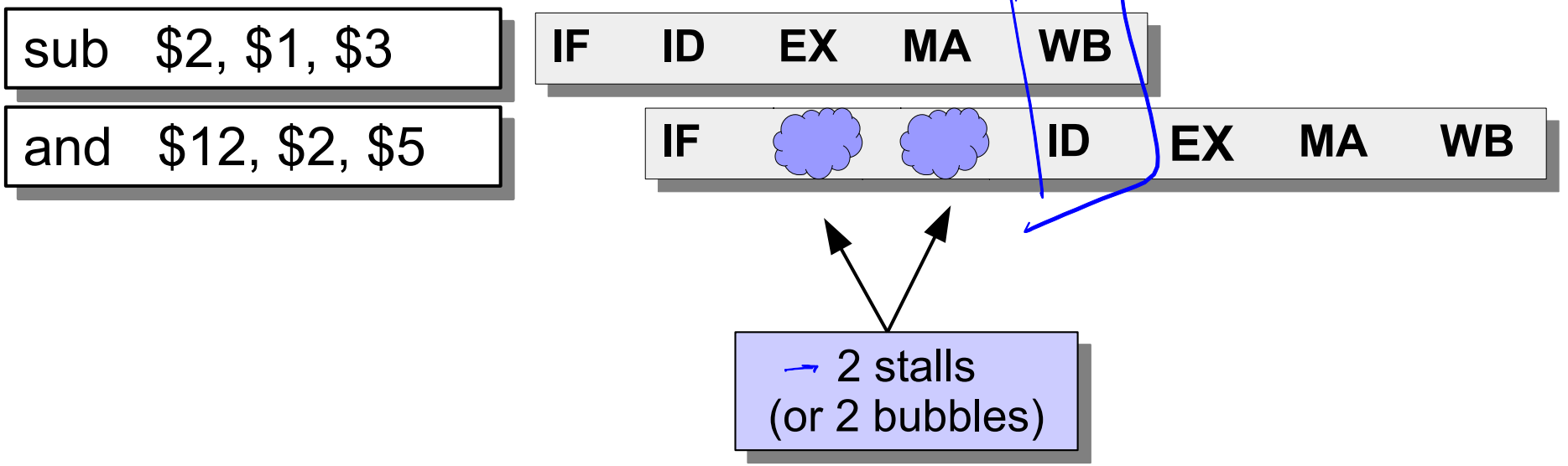
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```



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**

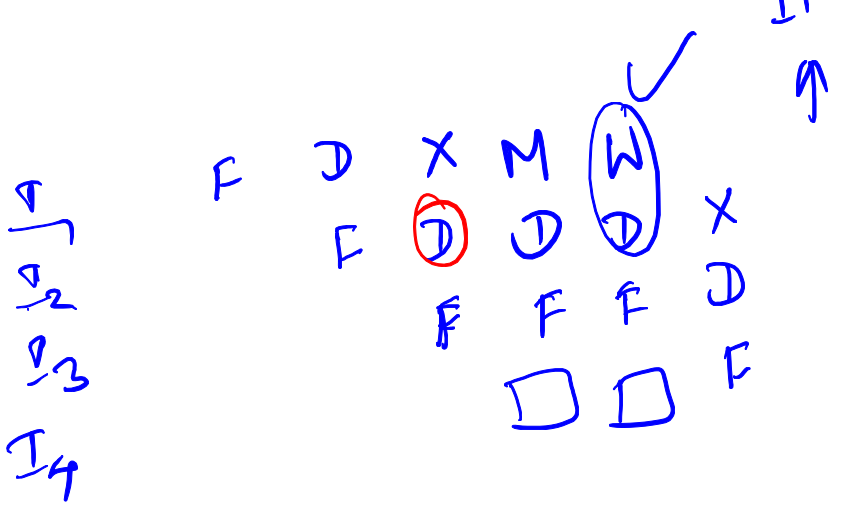
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```

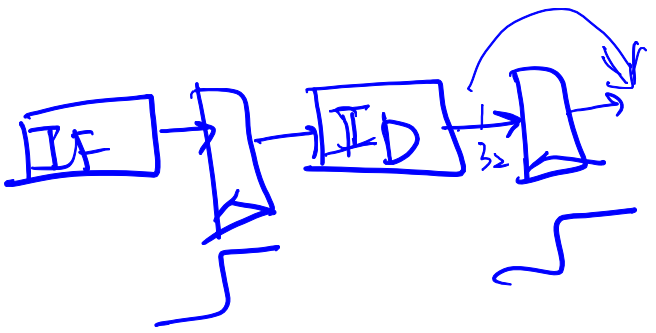
sub $2, $1, $3	IF	ID	EX	MA	WB
and $12, $2, $5	IF				ID

1 2 3 4 5

OK

IF IF

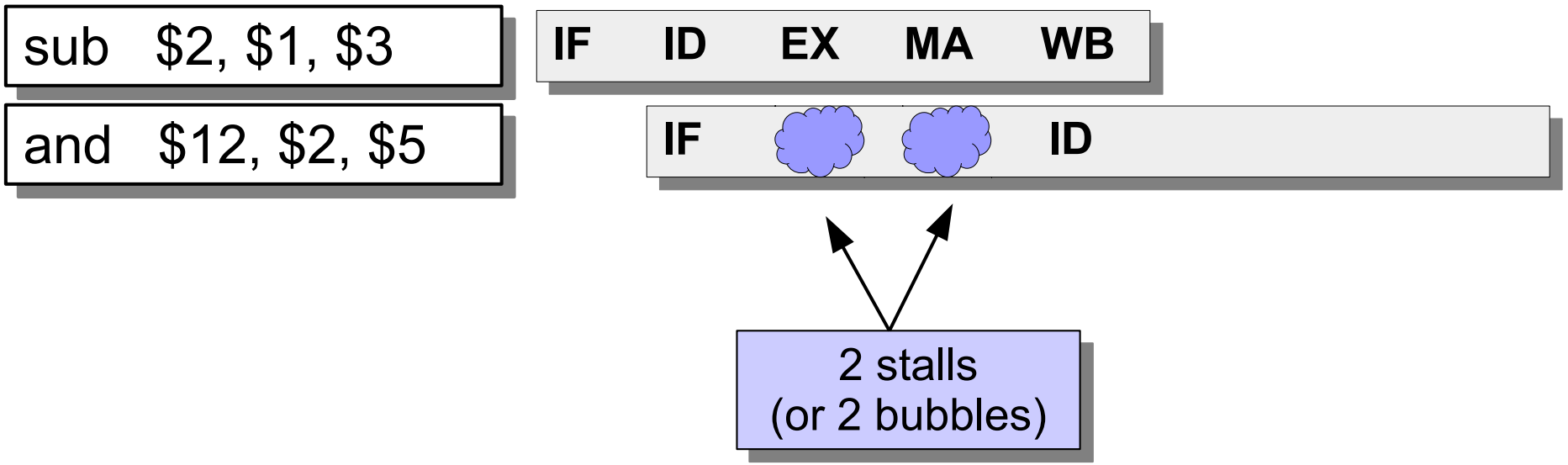




Execution Sequence – Dependence

and instruction should
read $2 **after** sub **writes**

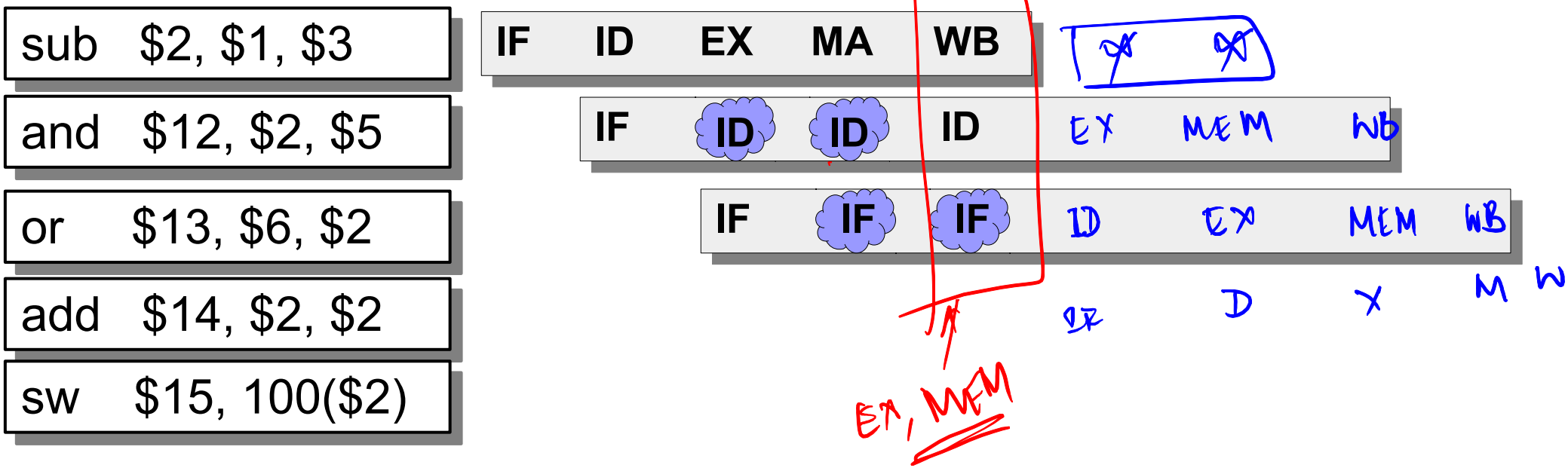
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```

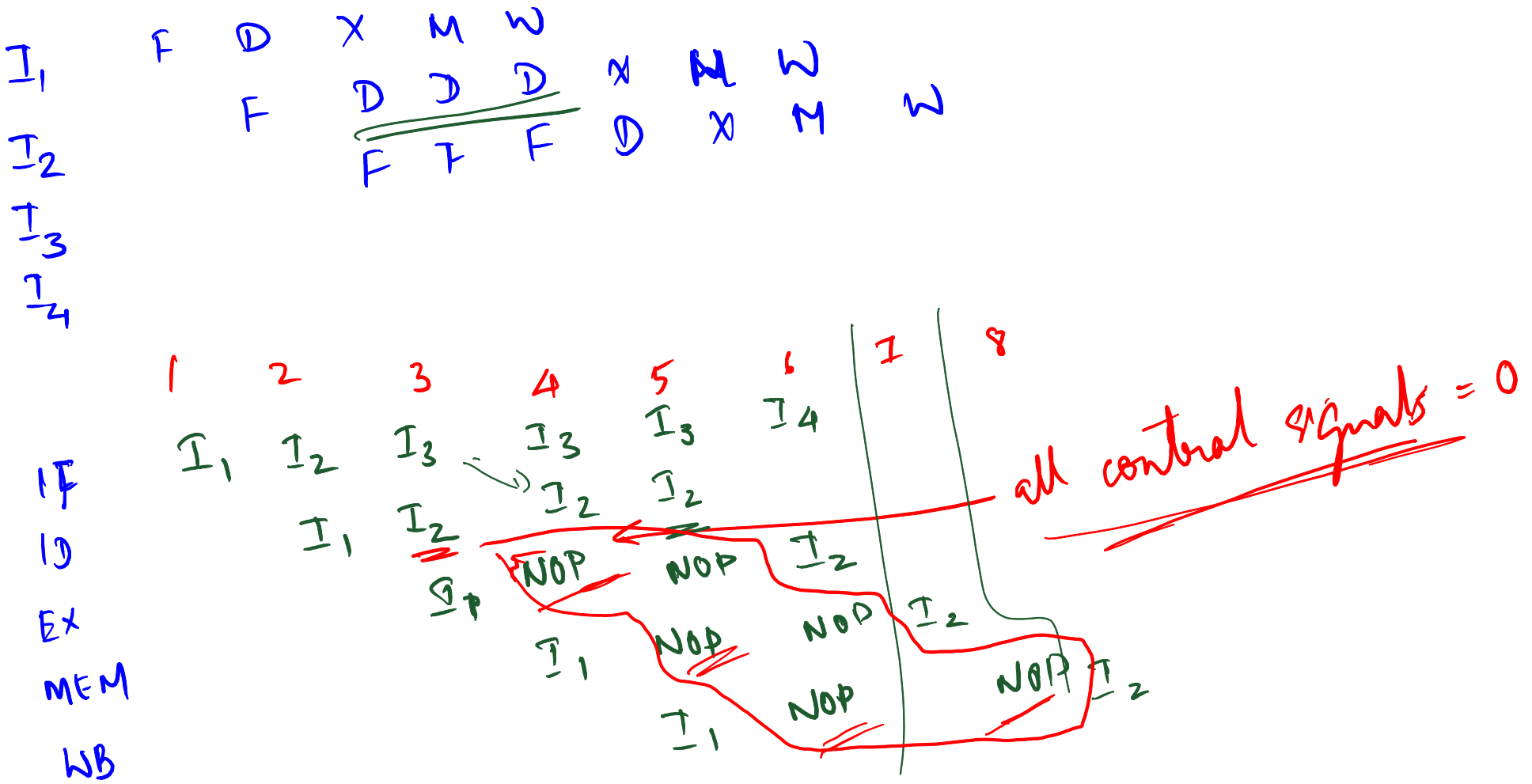


Execution Sequence – Dependence

and instruction should **read** $2 **after** sub **writes**

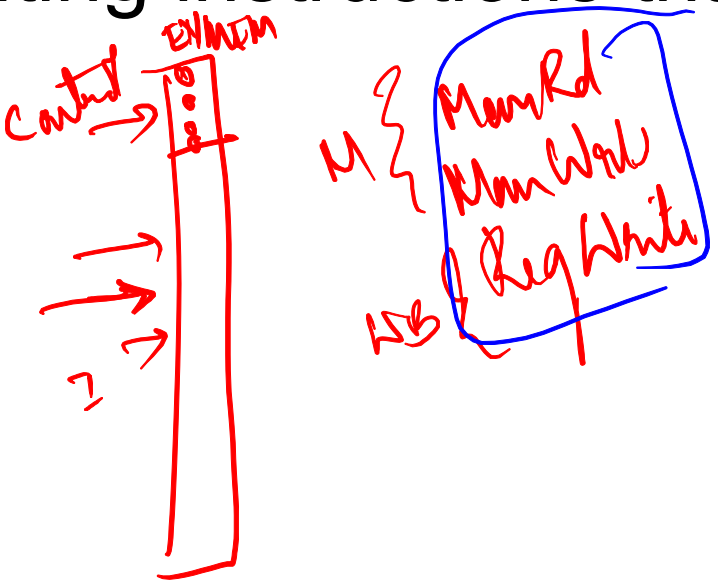
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```




I1
I2
I3
I4
F D X M W
F D D D X M W
F F F D X M W
1 2 3 4 5 6 7 8
IF
ID
EX
MEM
WB
I1 I2 I3
I1 I2
I3 I2
I3 I2
NOP
NOP
NOP
NOP
NOP
NOP
I2
I2
I2
I1
I1
all control signals = 0

Pipeline Stall

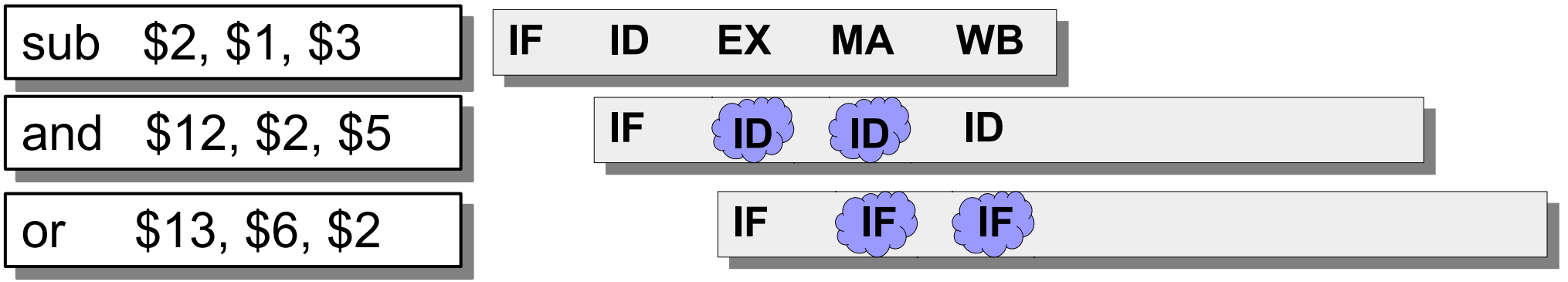
- Stall propagates
- Stall: Prevent the PC and IF/ID pipeline register from changing
- EX stage is executing instructions that have no effect
  - NOP



Execution Sequence – Dependence

and instruction should
read $2 **after** sub **writes**

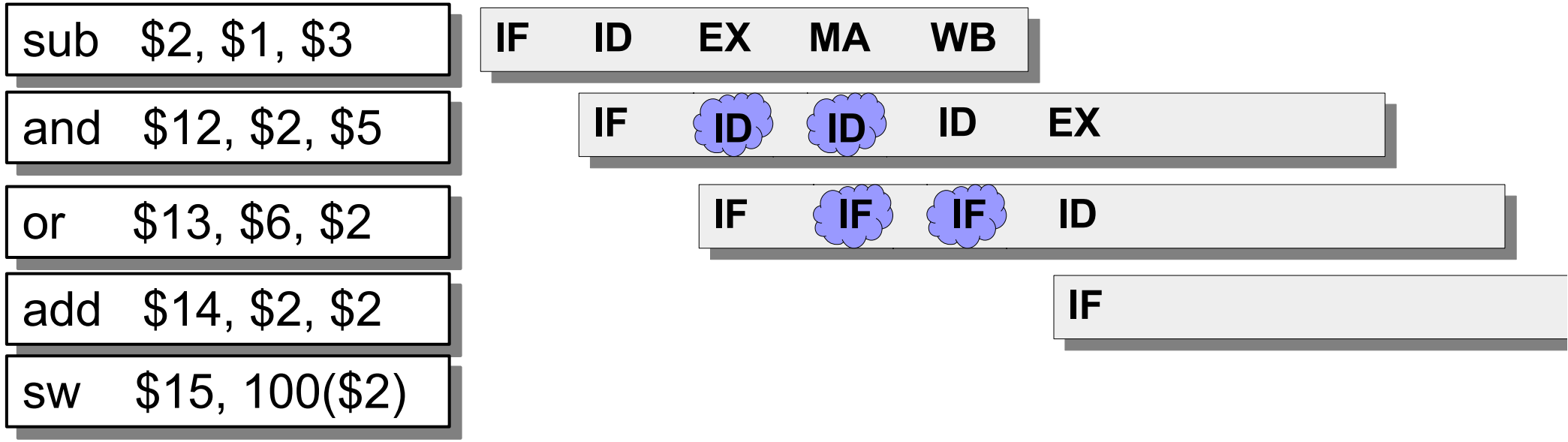
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```



Execution Sequence – Dependence

and instruction should
read $2 **after** sub **writes**

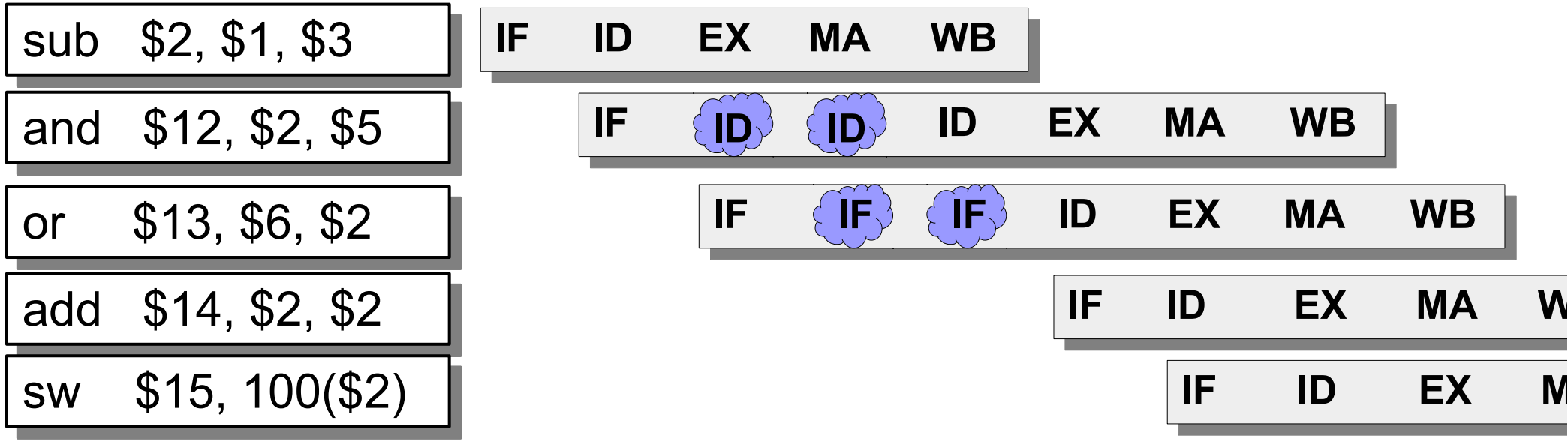
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```



Execution Sequence – Dependence

and instruction should
read $2 **after** sub **writes**

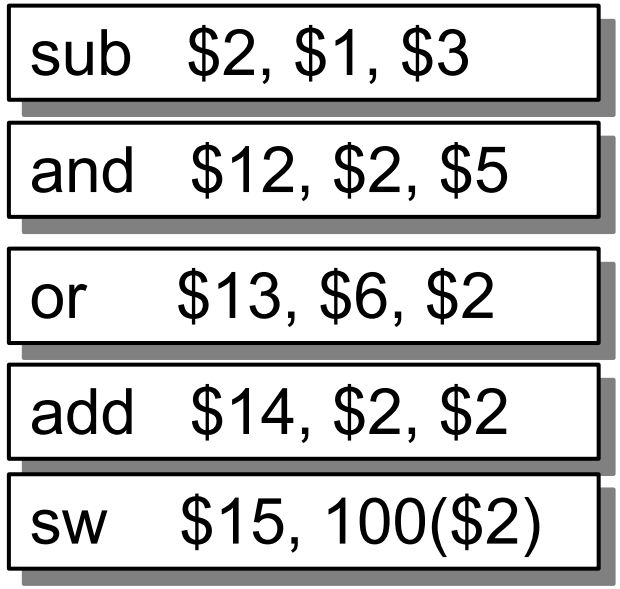
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```

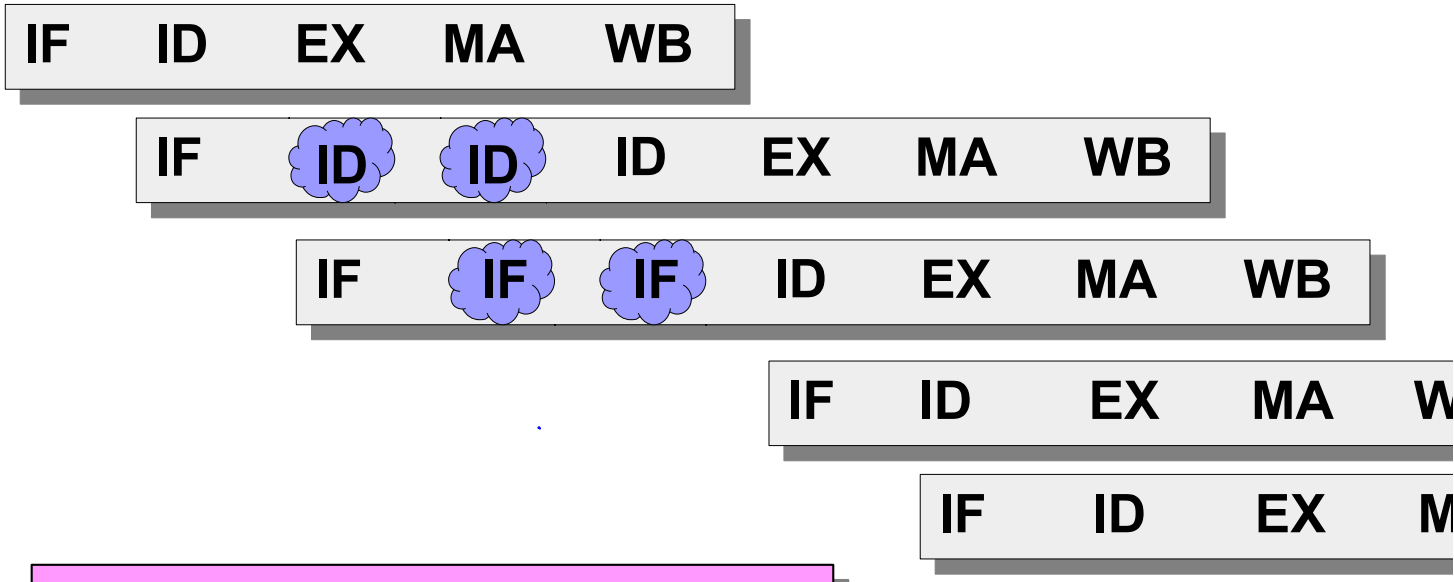


Execution Sequence – Dependence

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)
```

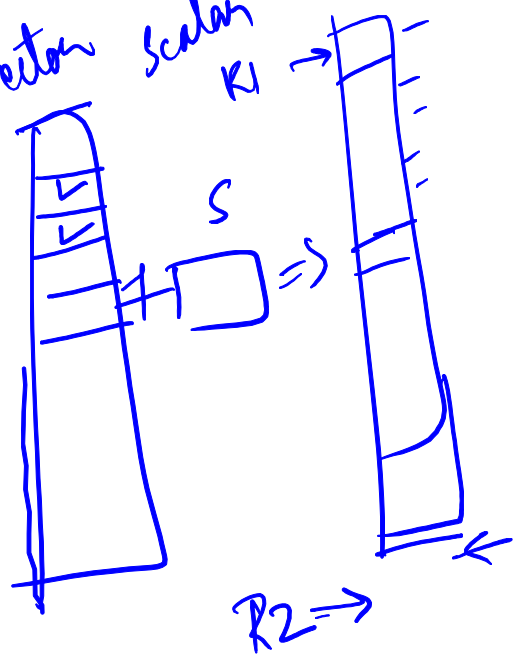




What is the Speedup?

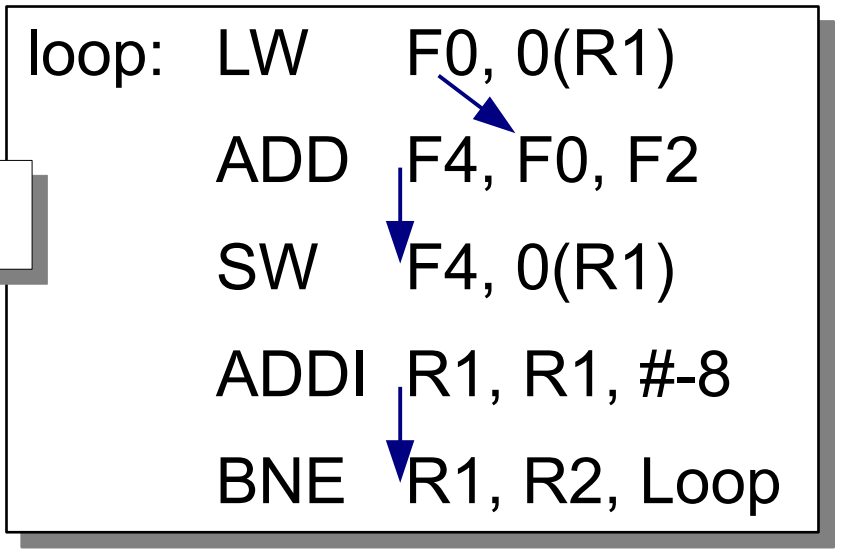
Dependence

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

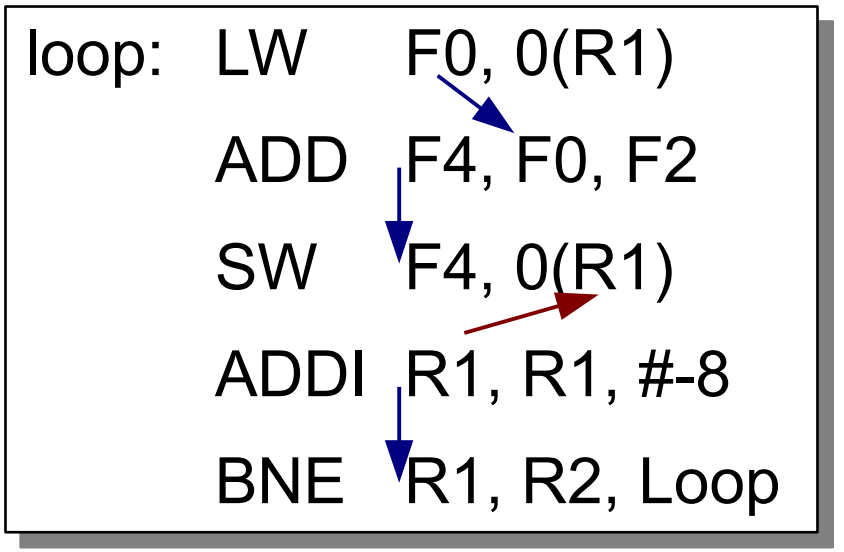


Dependence

Data Dependence



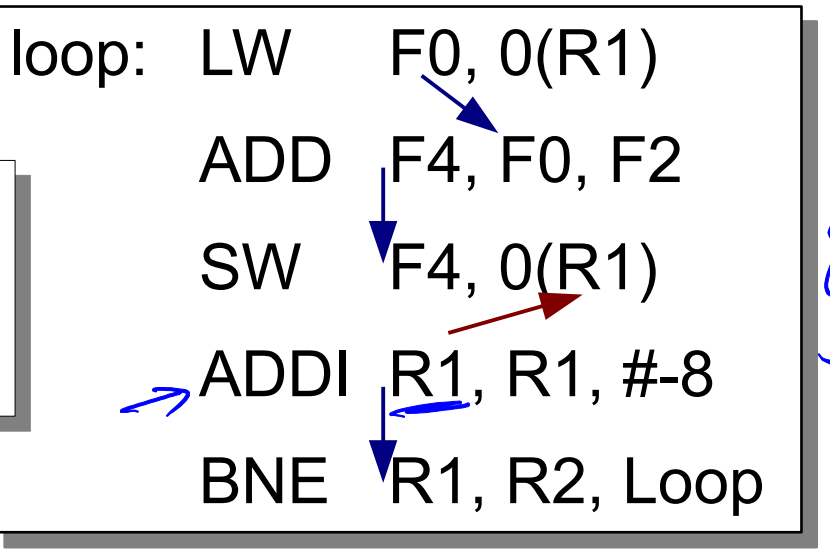
Dependence



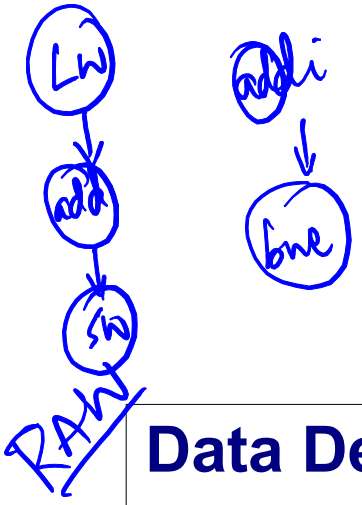
Dependence

Data Dependence

Name Dependence



Dependence

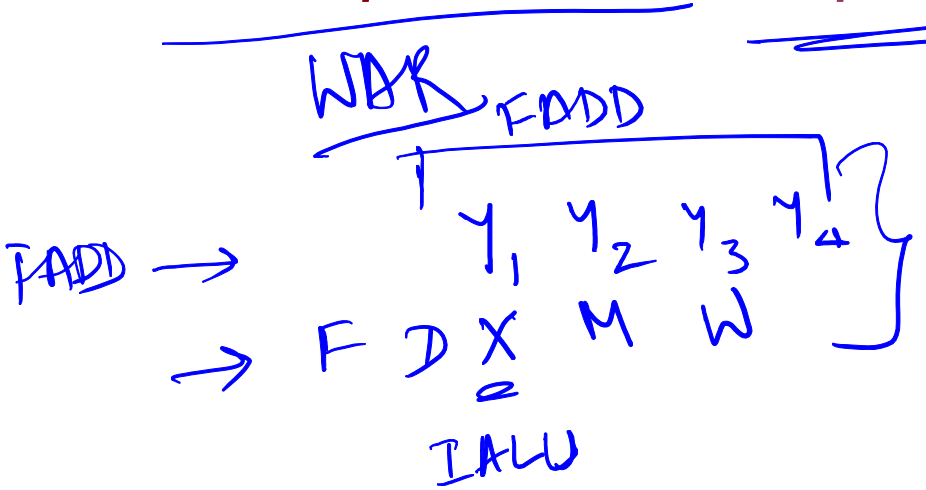


Data Dependence

Name Dependence

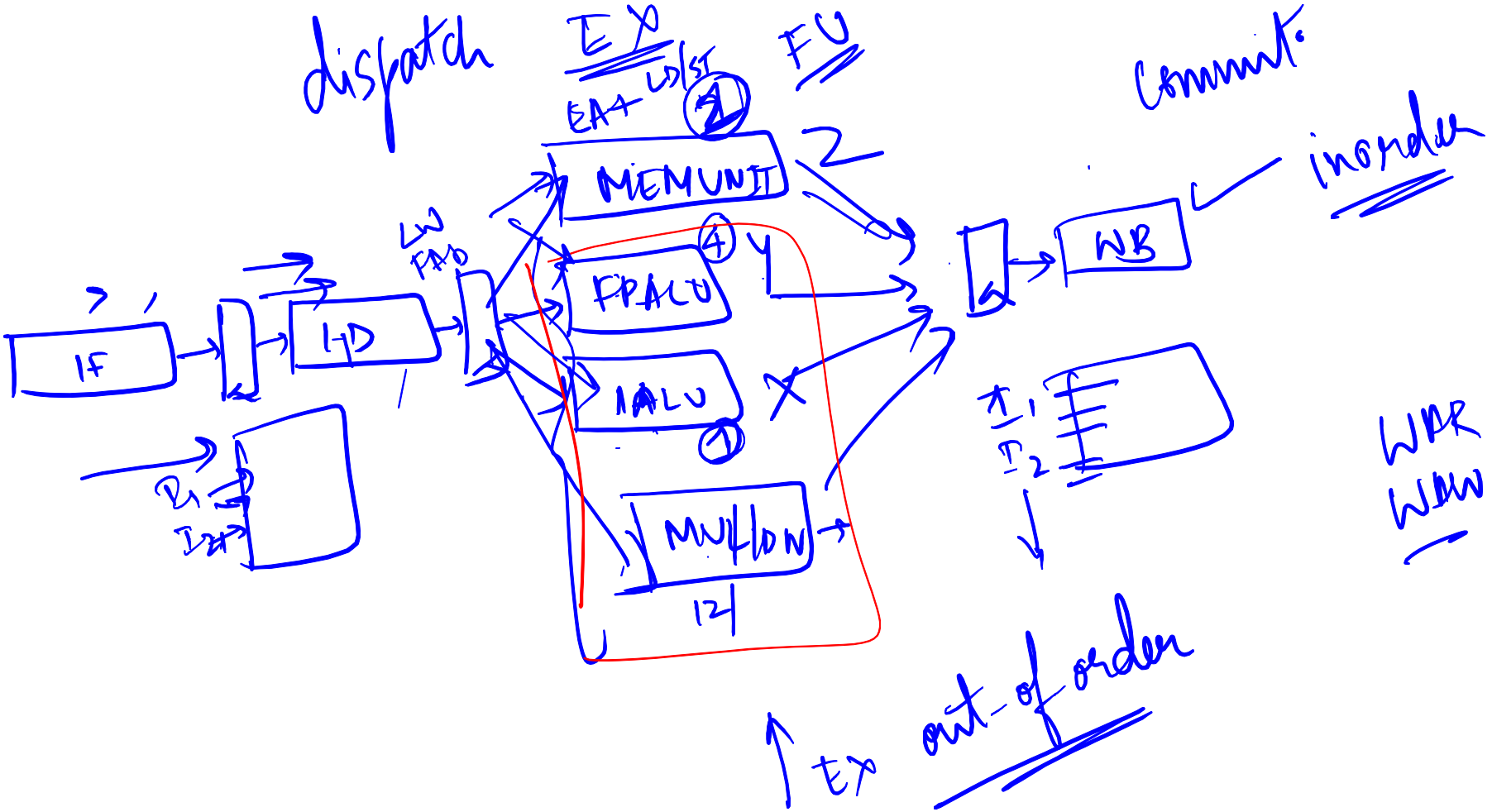
```
loop:  LW    F0, 0(R1)
       ADD   F4, F0, F2
       SW    F4, 0(R1)
       ADDI  R1, R1, #-8
       BNE   R1, R2, Loop
```

- Name dependence
  - *antidependence*, *output dependence*




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

WAW


dispatch
EX
FU
EA
LD/ST
MEMUNIT
FPALU
IALU
MULTDIV
LW
FADD
IF
ID
WB
commit
in order
WAR
WAW
out of order
EX

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
```

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

```
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
```

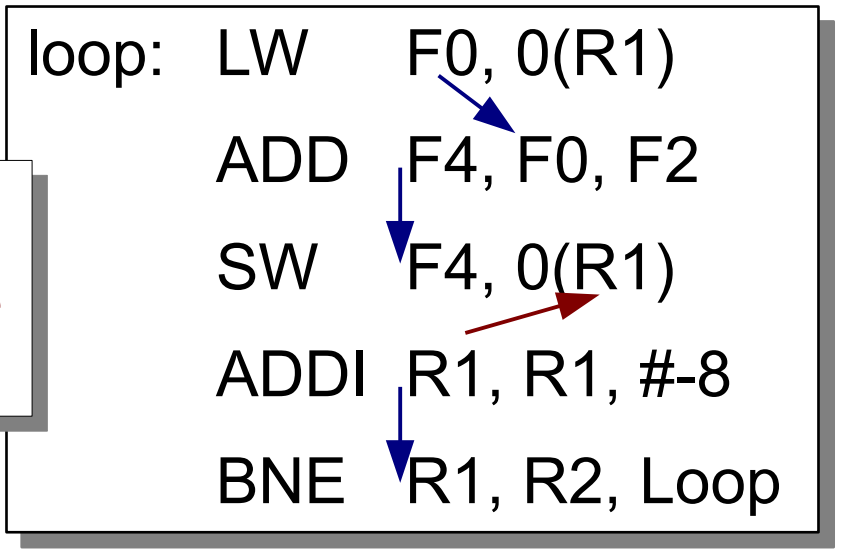
- RAW, WAR, WAW dependences

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

Dependence

Data Dependence

Name Dependence



- 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

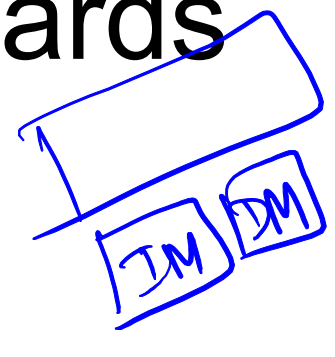
- Hazard: *n.* An unavoidable danger or risk, even though often foreseeable.
- Situations that prevent the next instruction in the instruction stream from being executing during its designated clock cycle

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

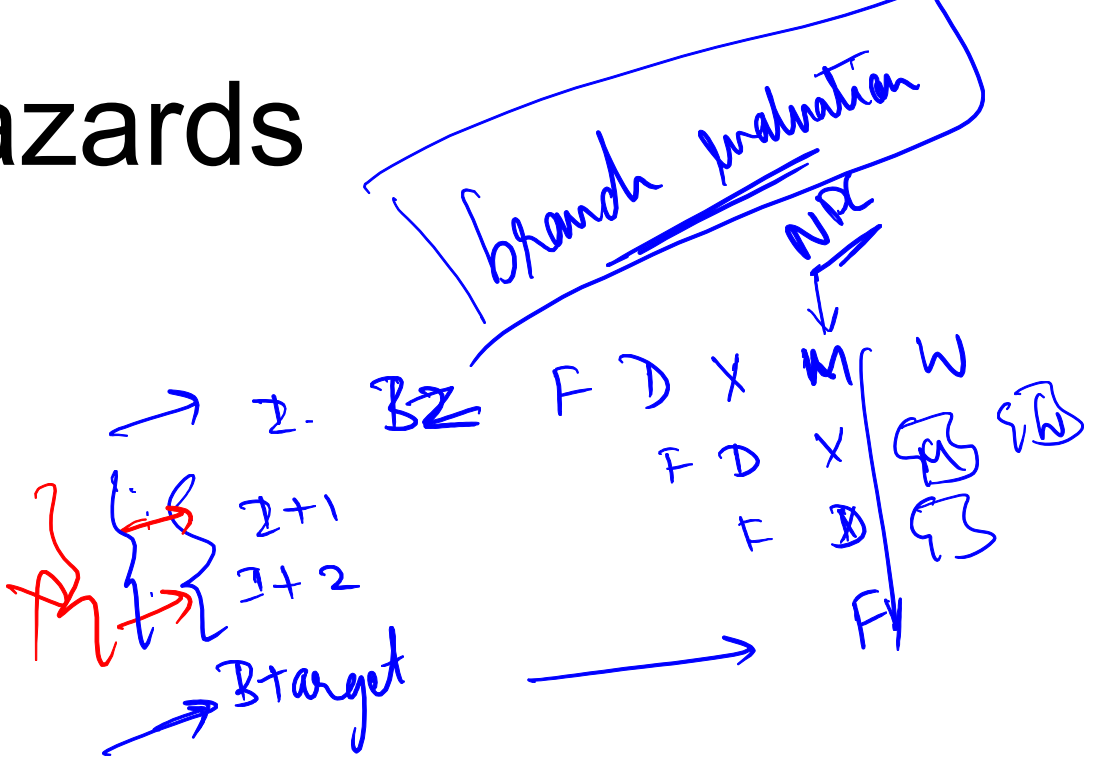


Hazards

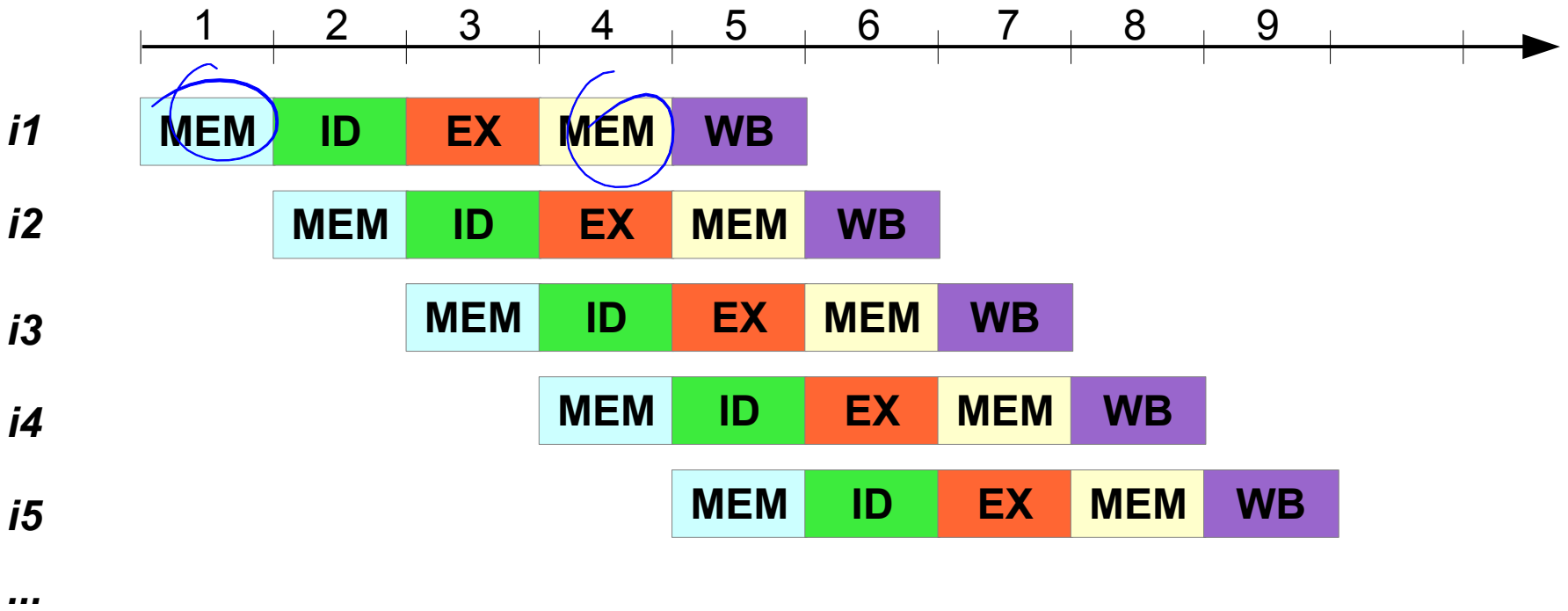
- Structural Hazards
  - Resource conflicts
- Data Hazards
  - RAW, WAW, WAR
-

Hazards

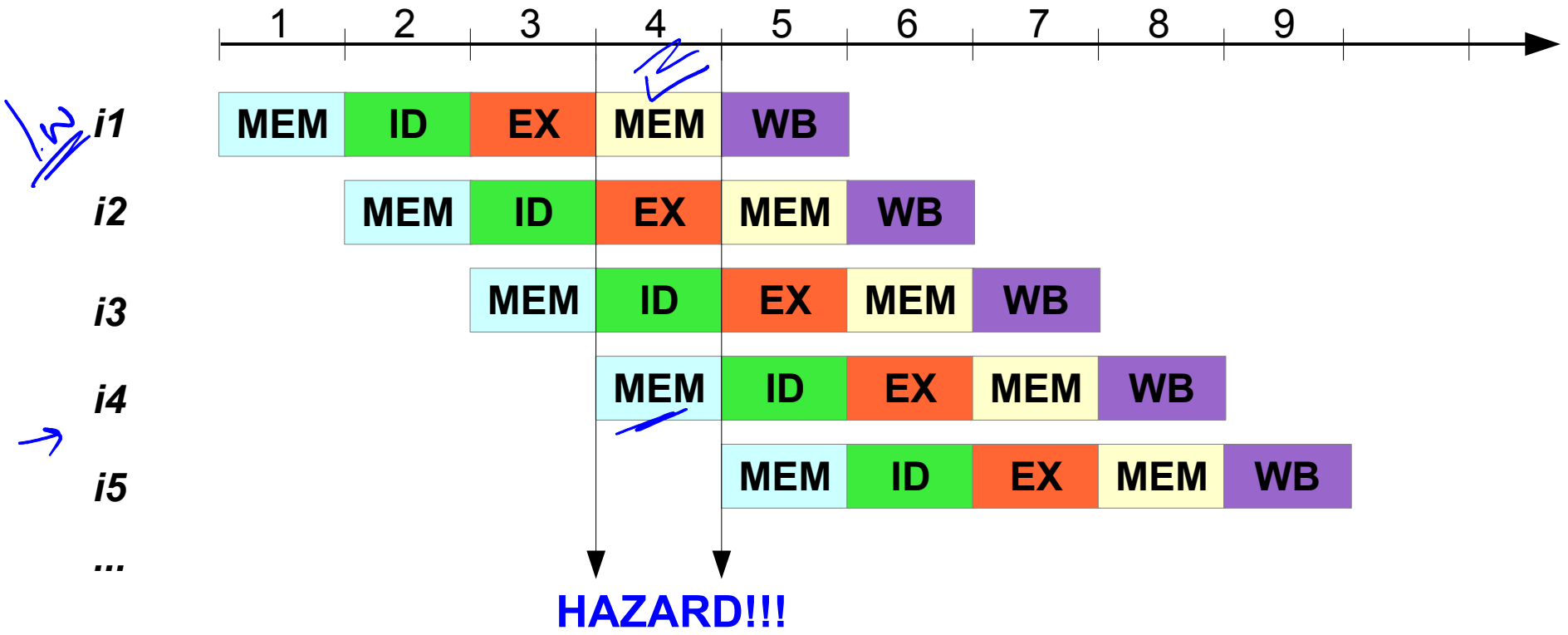
- Structural Hazards
  - Resource conflicts
- Data Hazards
  - RAW, WAW, WAR
- Control Hazard
  - Whether or not an instruction should be executed depends on a control decision made by an earlier instruction



Structural Hazard

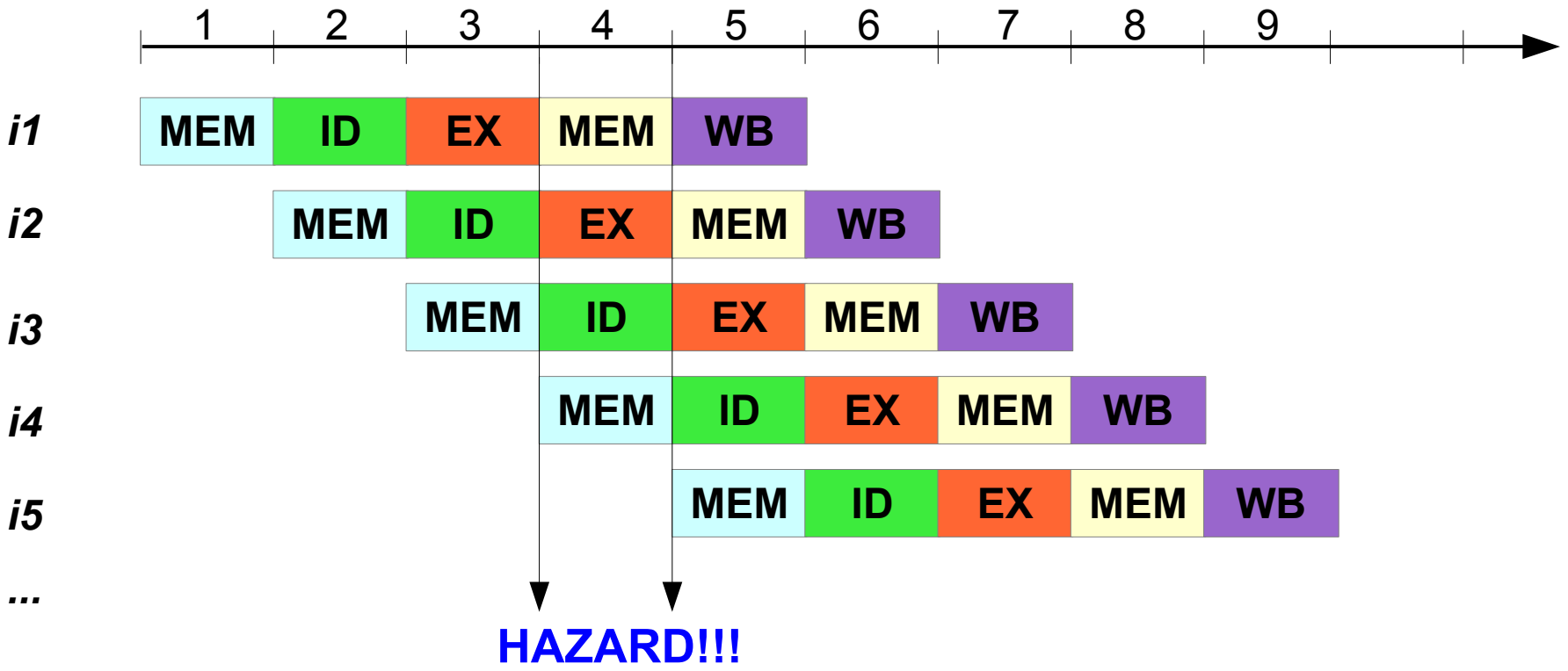


Structural Hazard



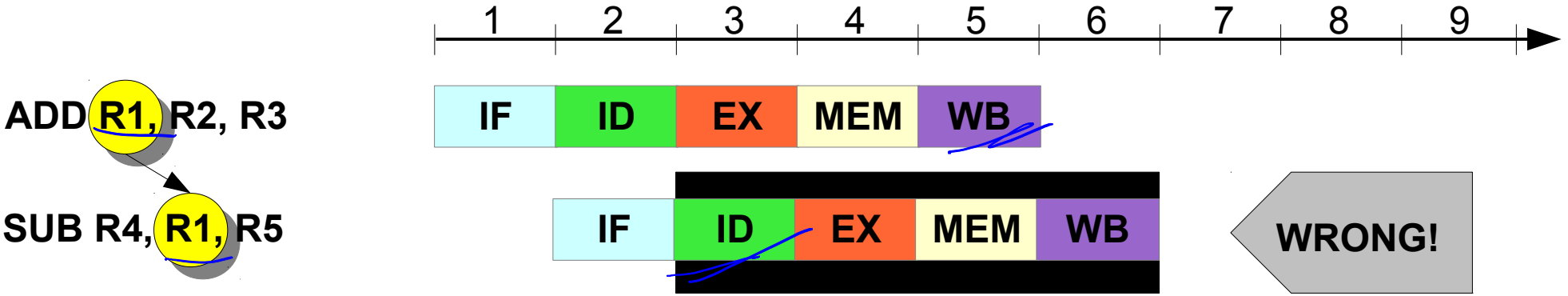
- Lack of resources

Structural Hazard

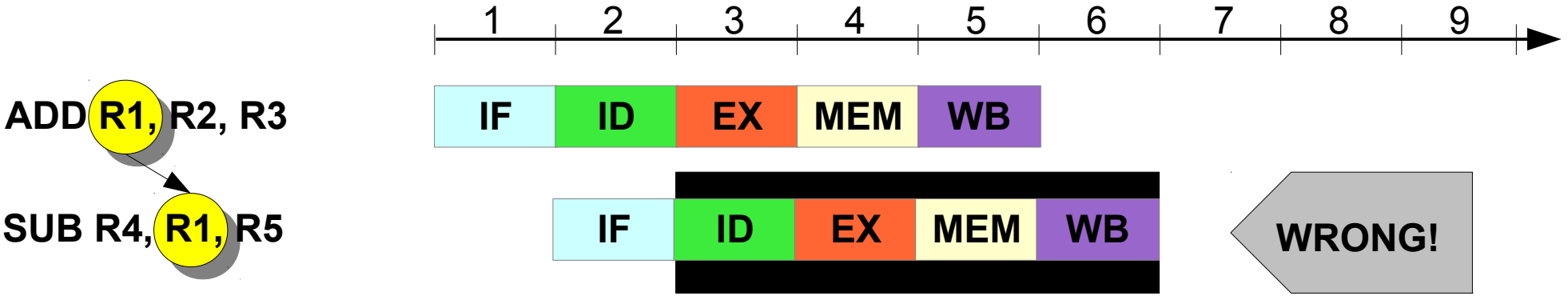


- Lack of resources
- Solution: Increase resources
  - Use of separate Data and Instruction memories in the MIPS pipeline

Data Hazard

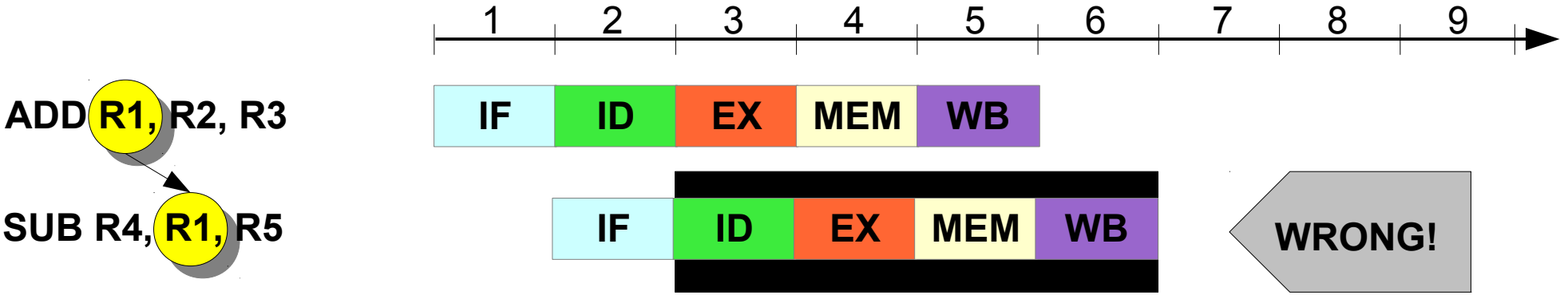

1
2
3
4
5
6
7
8
9
ADD R1, R2, R3
SUB R4, R1, R5
IF
ID
EX
MEM
WB
IF
ID
EX
MEM
WB
WRONG!

Data Hazard



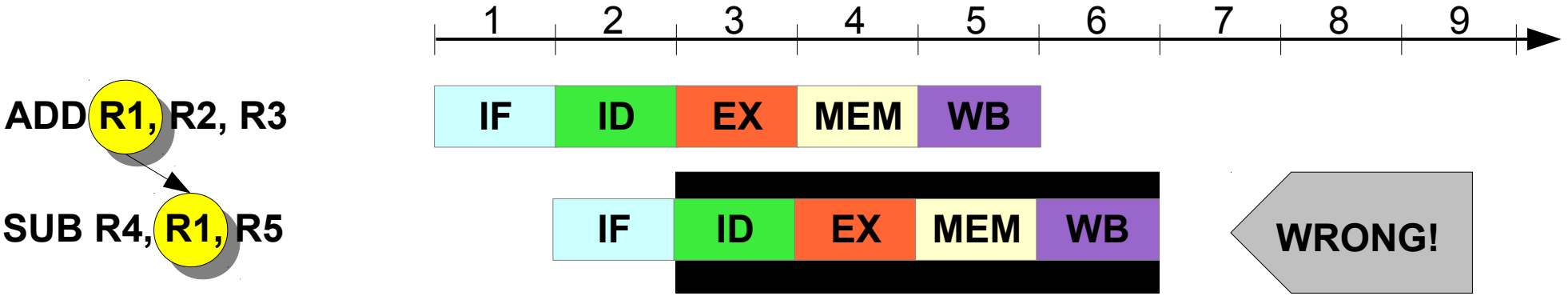
- Data (input operands) required by the instruction are not ready

Data Hazard

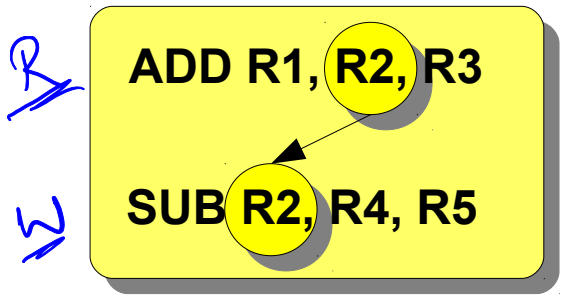


- Data (input operands) required by the instruction are not ready
- Data dependence
- RAW, WAR, WAW dependences

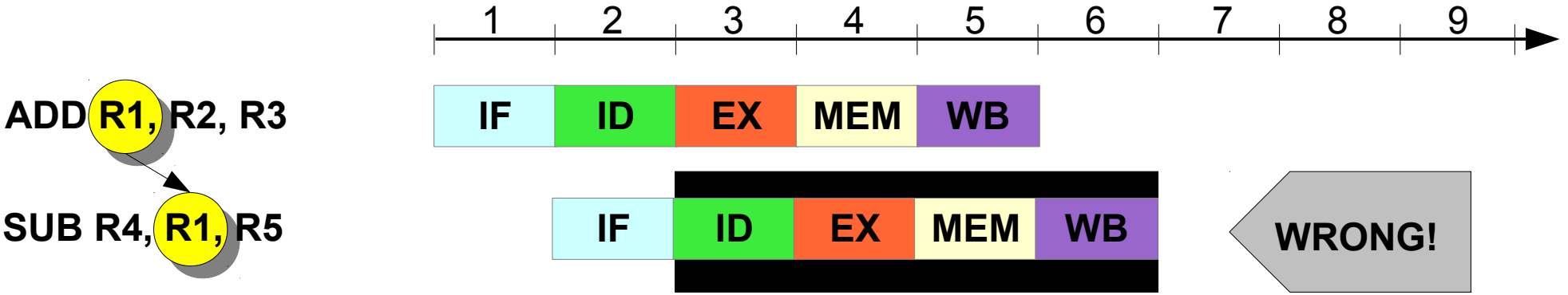
Data Hazard



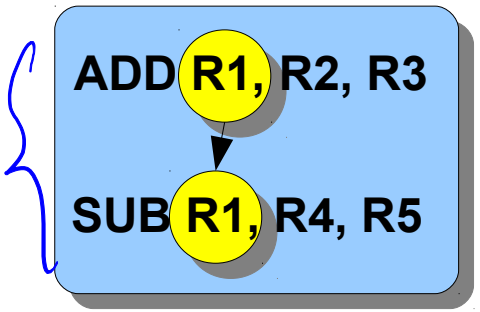
- Data (input operands) required by the instruction are not ready
- Data dependence
- RAW, WAR, WAW dependences



Data Hazard

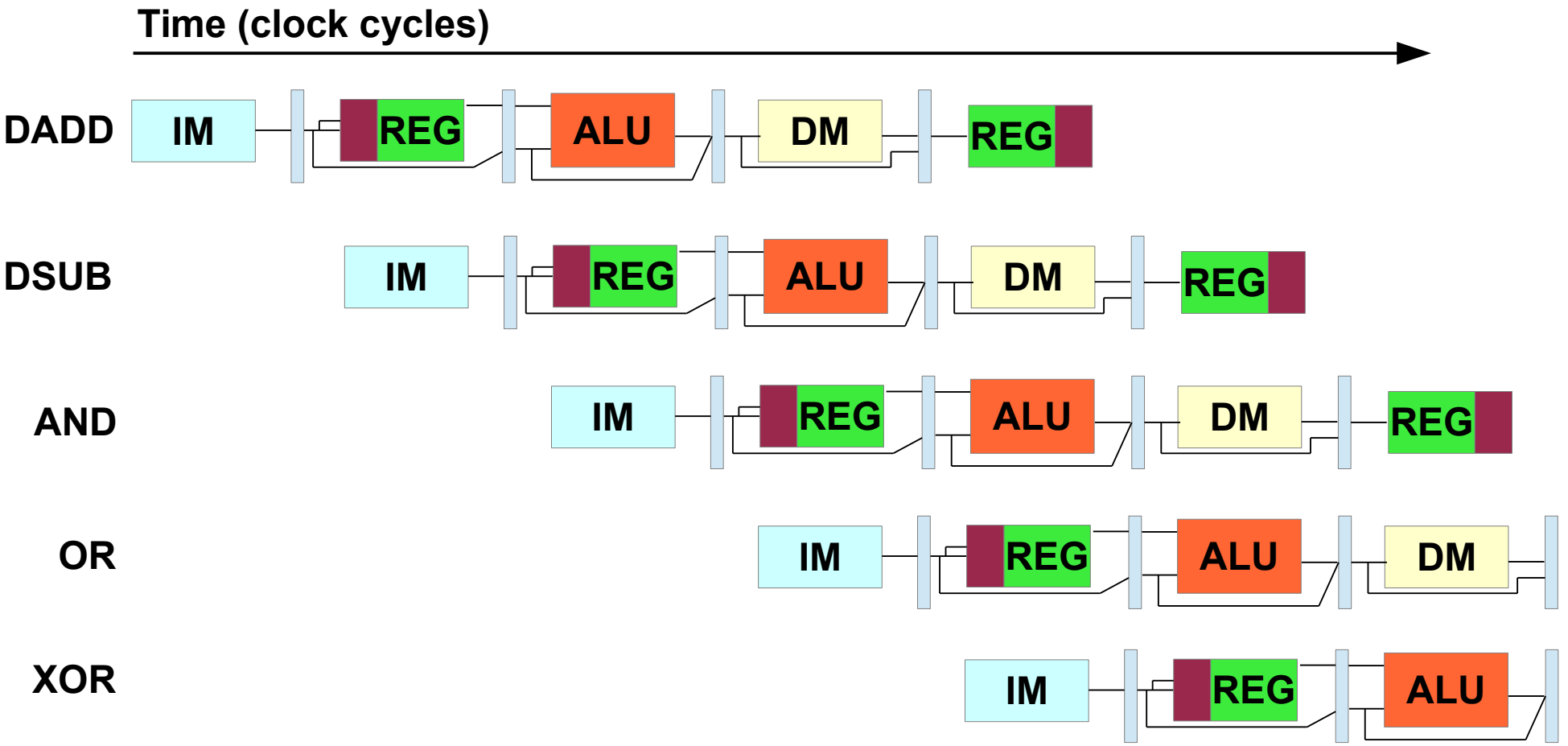


- Data (input operands) required by the instruction are not ready
- Data dependence
- RAW, WAR, WAW dependences



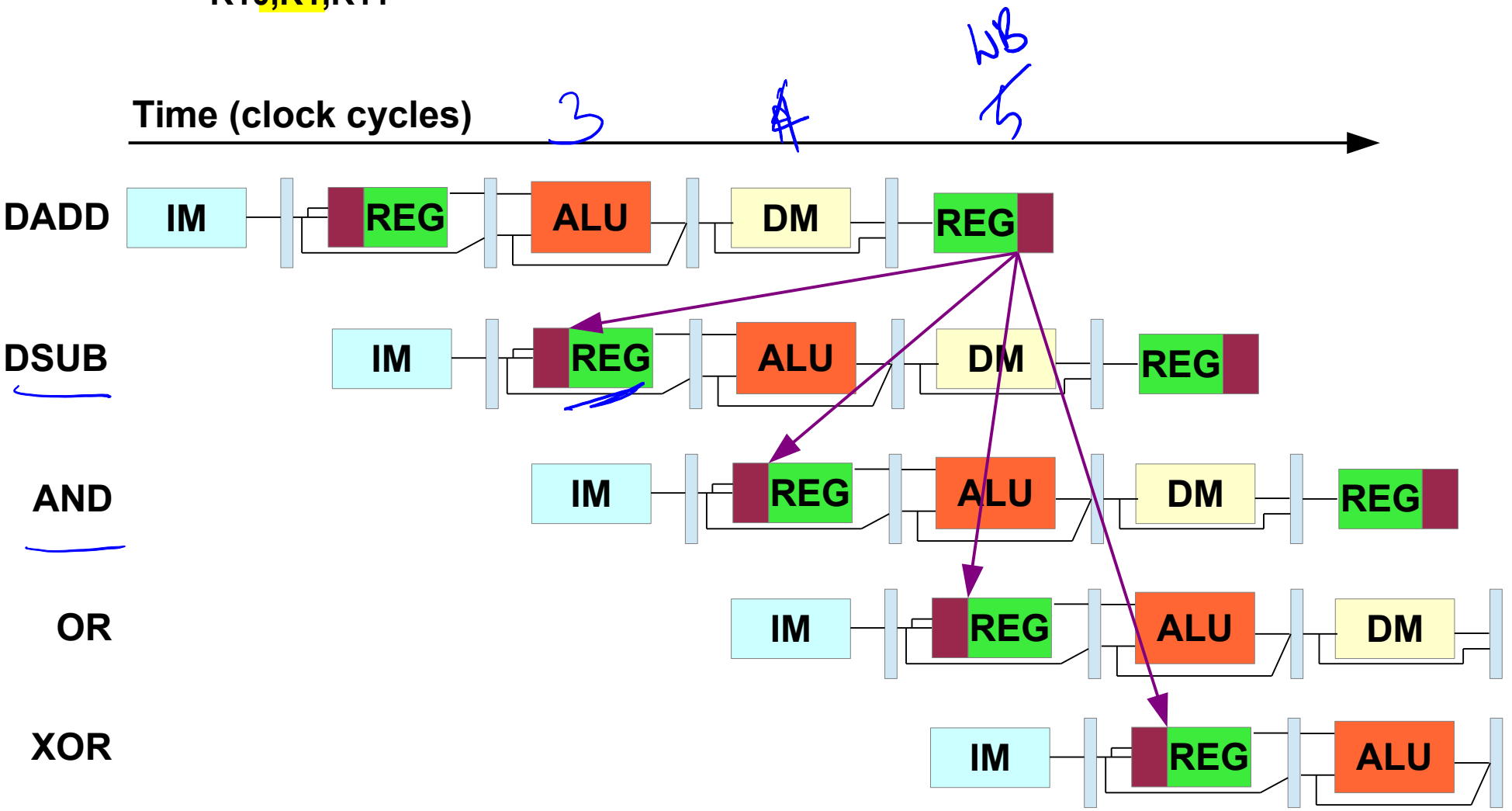
Data Hazard

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



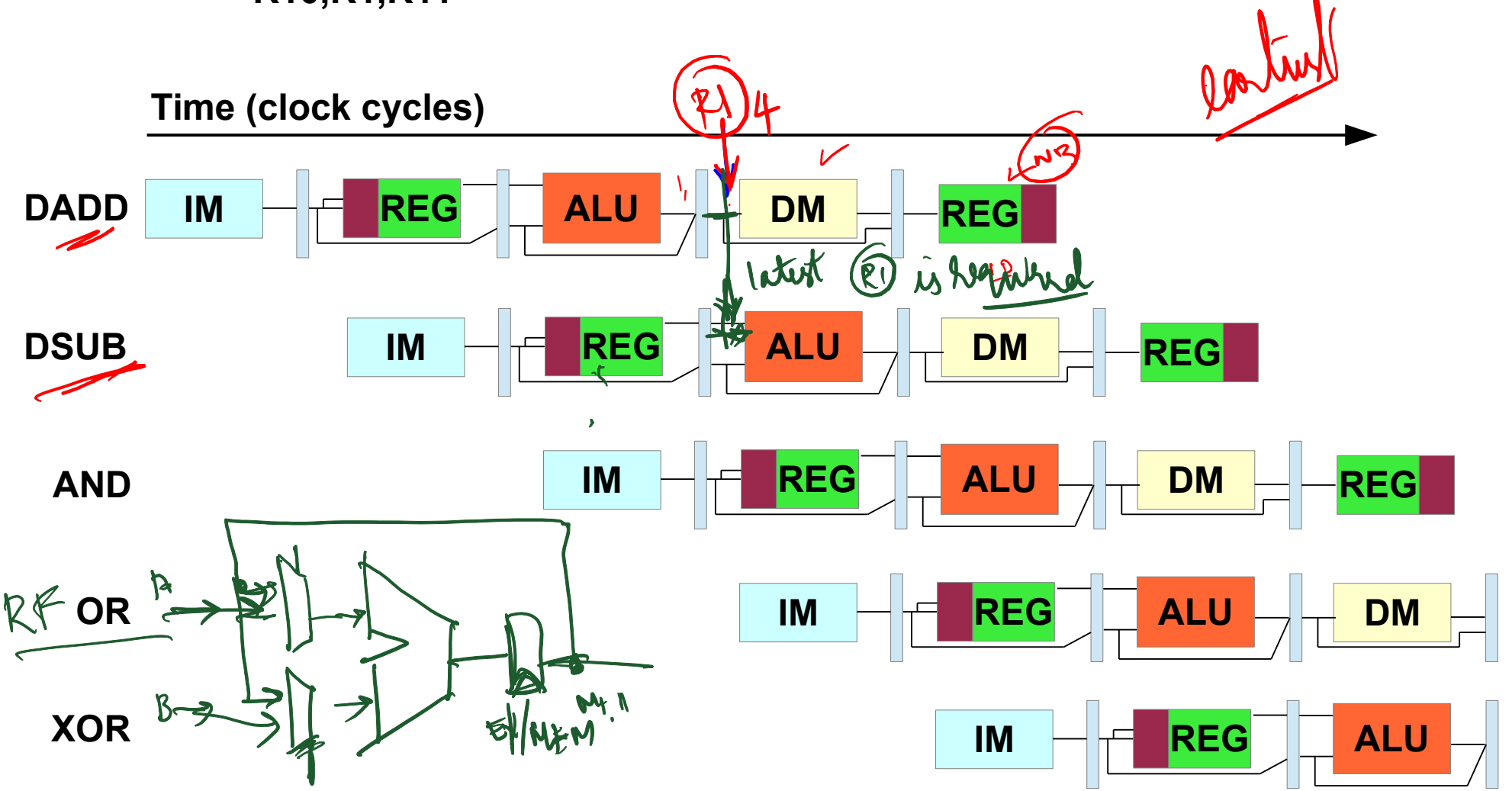
Data Hazard

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



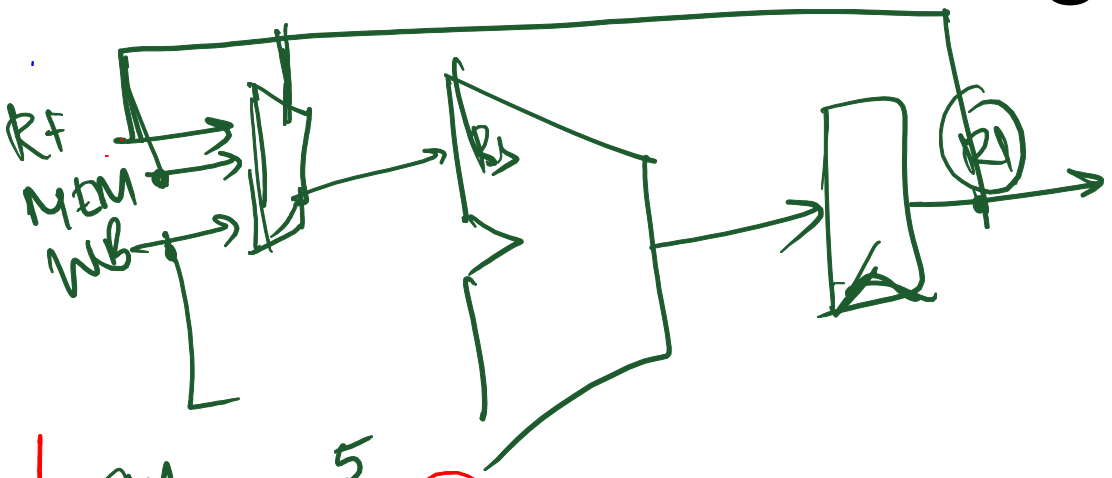
Avoiding Data Hazards – Forwarding

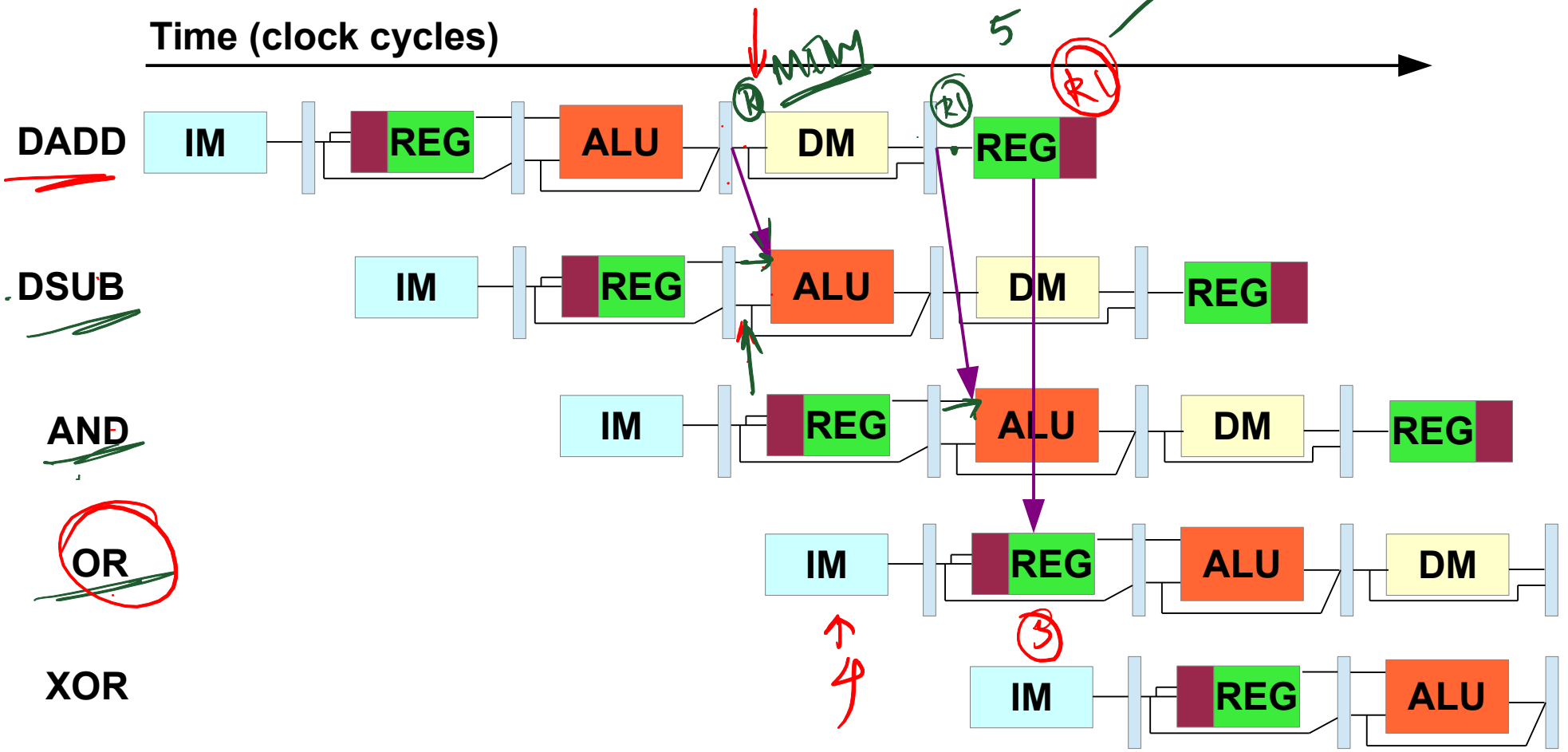
```
DADD   R1,R2,R3
DSUB   R4,R1,R5
AND    R6,R1,R7
OR     R8,R1,R9
XOR    R10,R1,R11
```



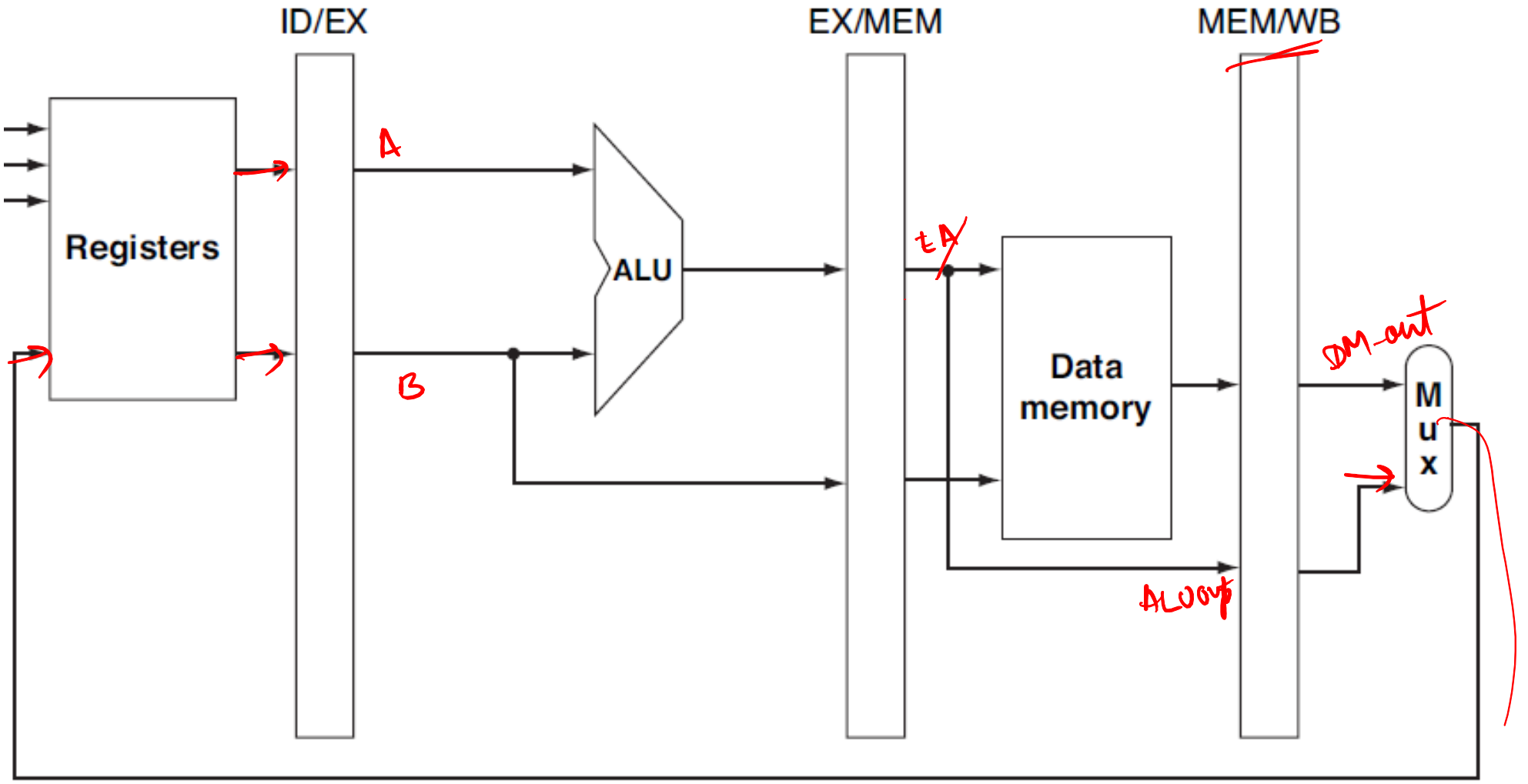
Avoiding Data Hazards – Forwarding

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

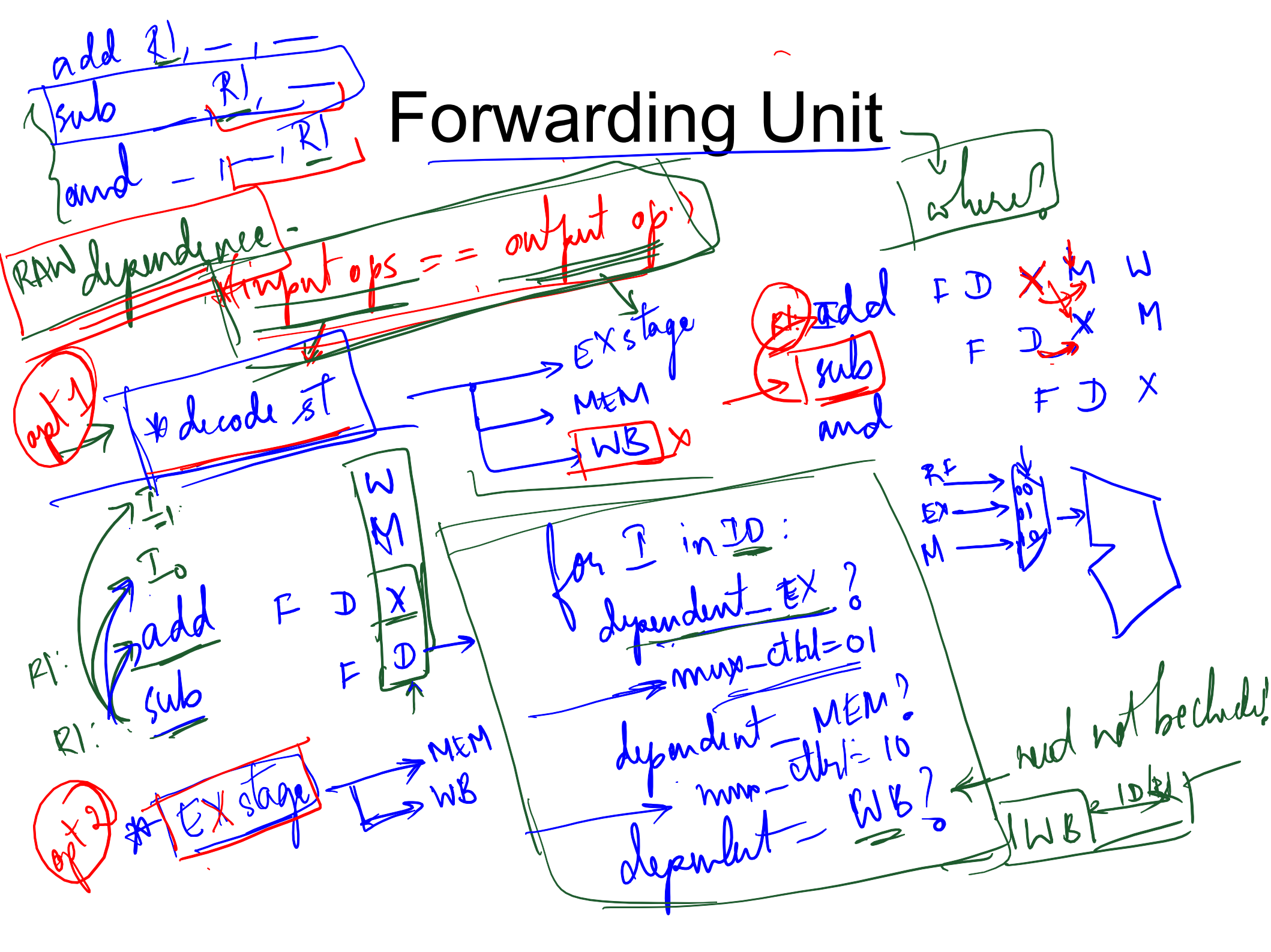




Pipeline without Forwarding



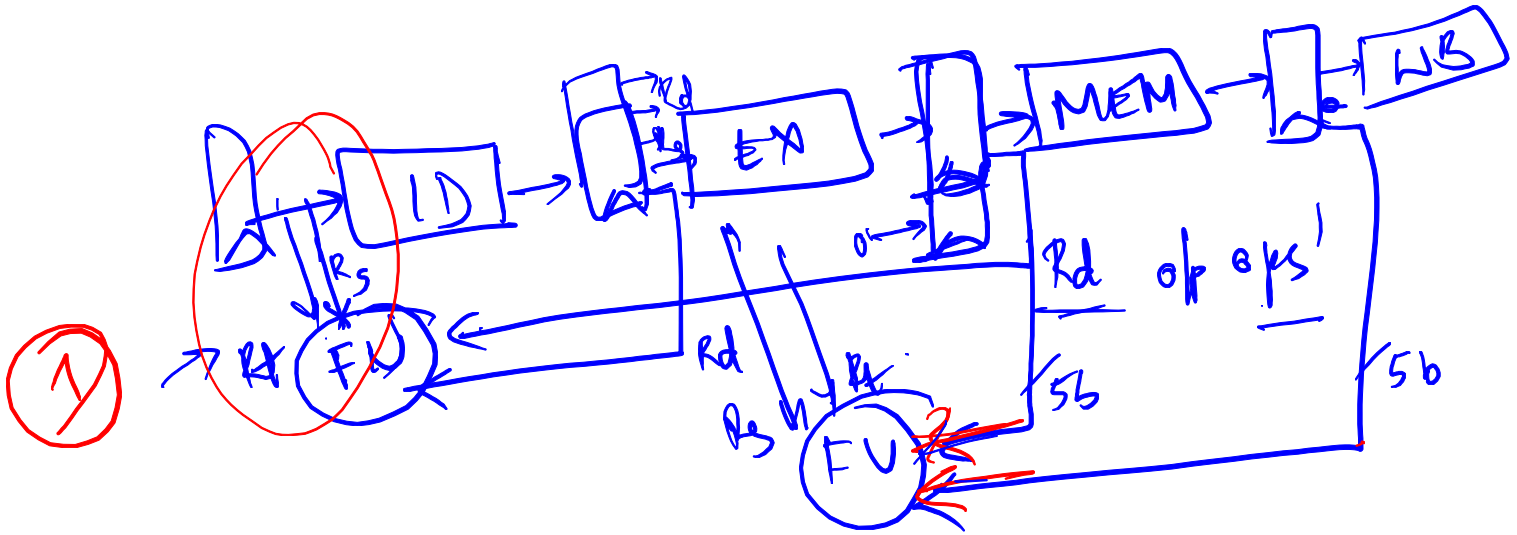
Forwarding Unit



check takes large time

↳ dependent_MEM ⇒ $(Rs_{EX} == Rd_{M}) \;||\; (Rt_{EX} == Rd_{M})$

- ~~lw~~
- ~~sub~~
- ~~(XOR) OR~~



1

2

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
DSUB   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?
  - ($DSUB_{Rs}$ == $DADD_{Rd}$) OR ($DSUB_{Rt}$ == $DADD_{Rd}$)
  - Before passing input ops to the ALU
-

Forwarding Unit Logic

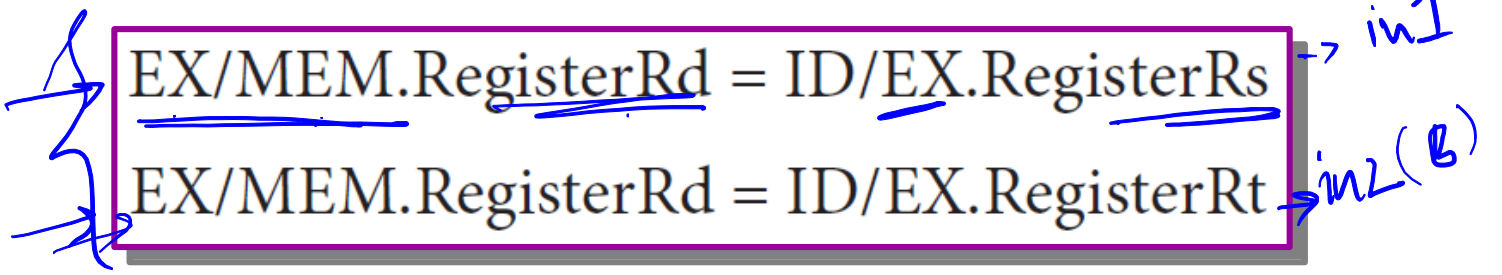
```
DADD   R1,R2,R3
DSUB   R4,R1,R5
AND    R6,R1,R7
```

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

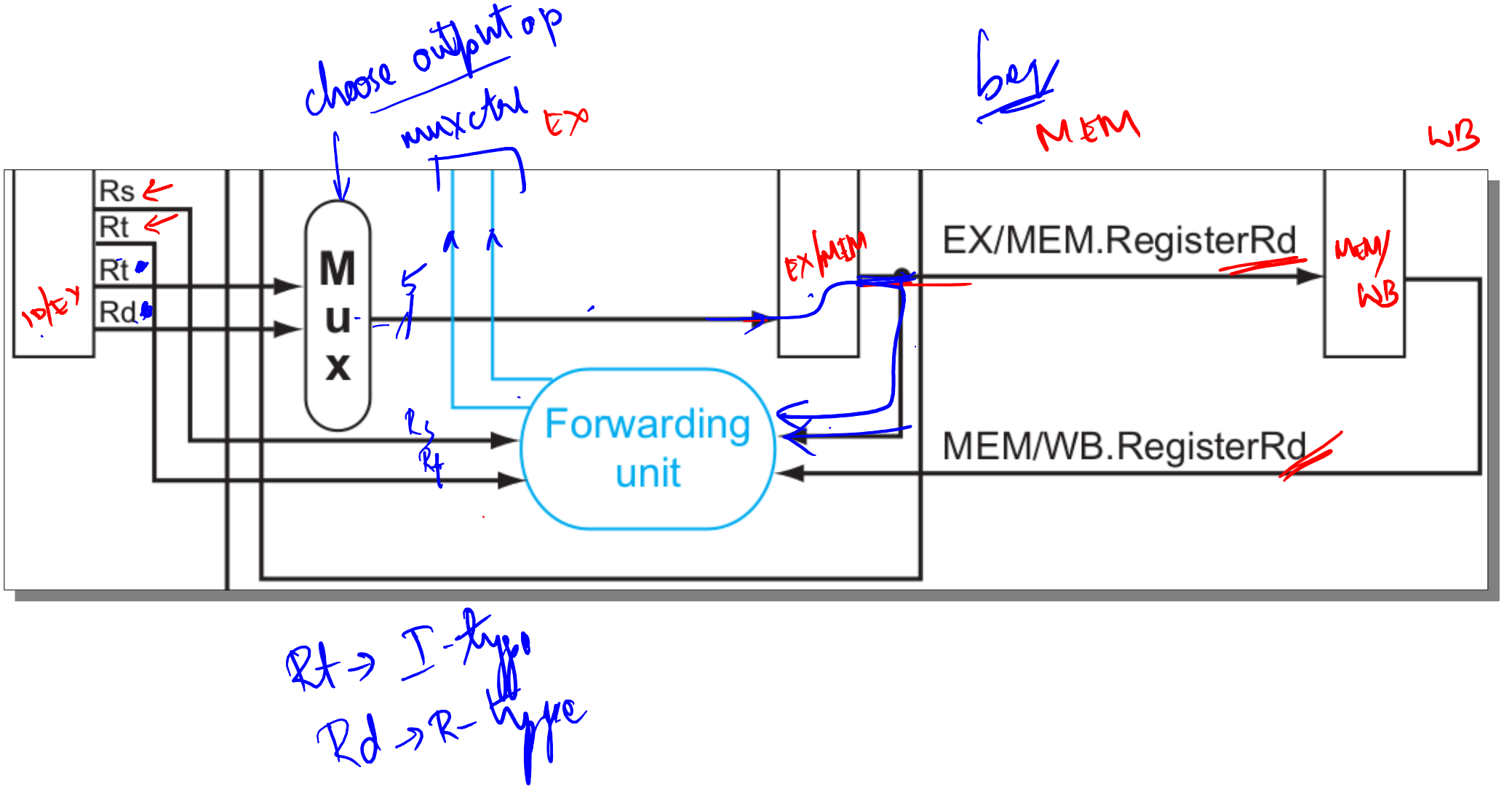
Forwarding Unit Logic

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

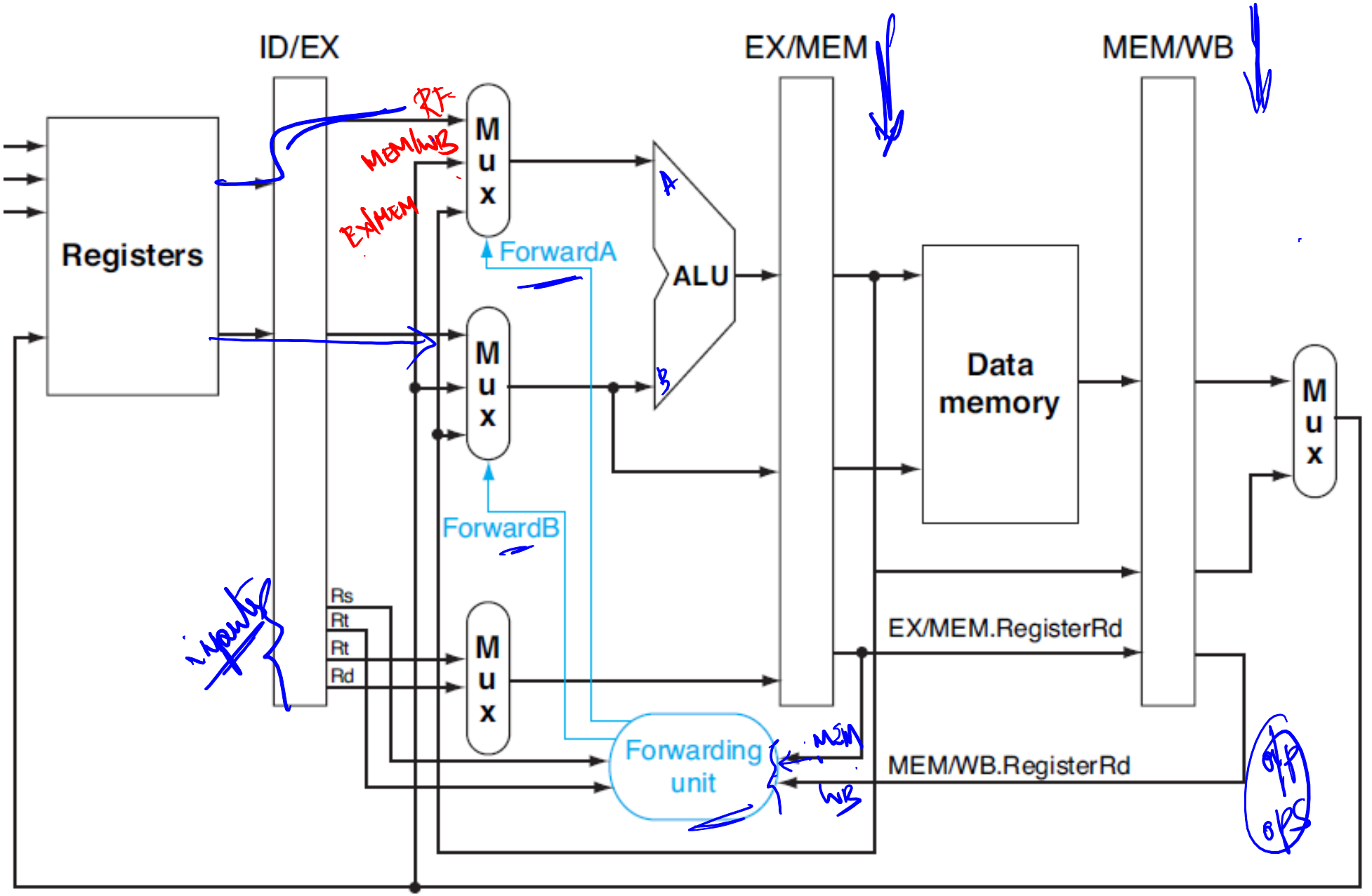
- DADD is in MEM; DSUB is in EX.
- Input Ops of DSUB = Output Op of DADD?
  - ($DSUB_{Rs}$ == $DADD_{Rd}$) OR ($DSUB_{Rt}$ == $DADD_{Rd}$)
  - Before passing input ops to the ALU
- If true, feed $DADD_{Rd}$ as corresponding input to the ALU



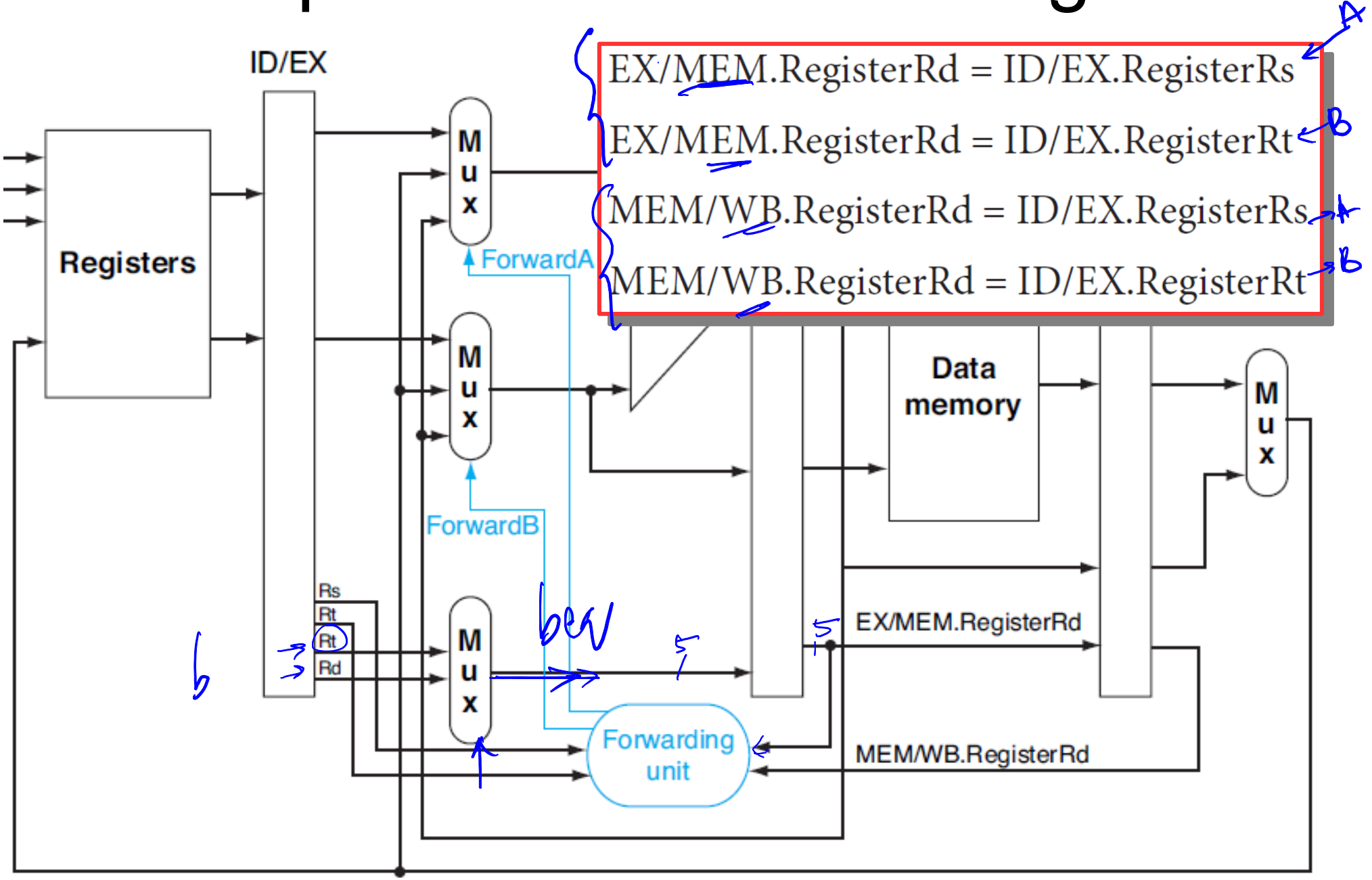
Forwarding Unit – Inputs



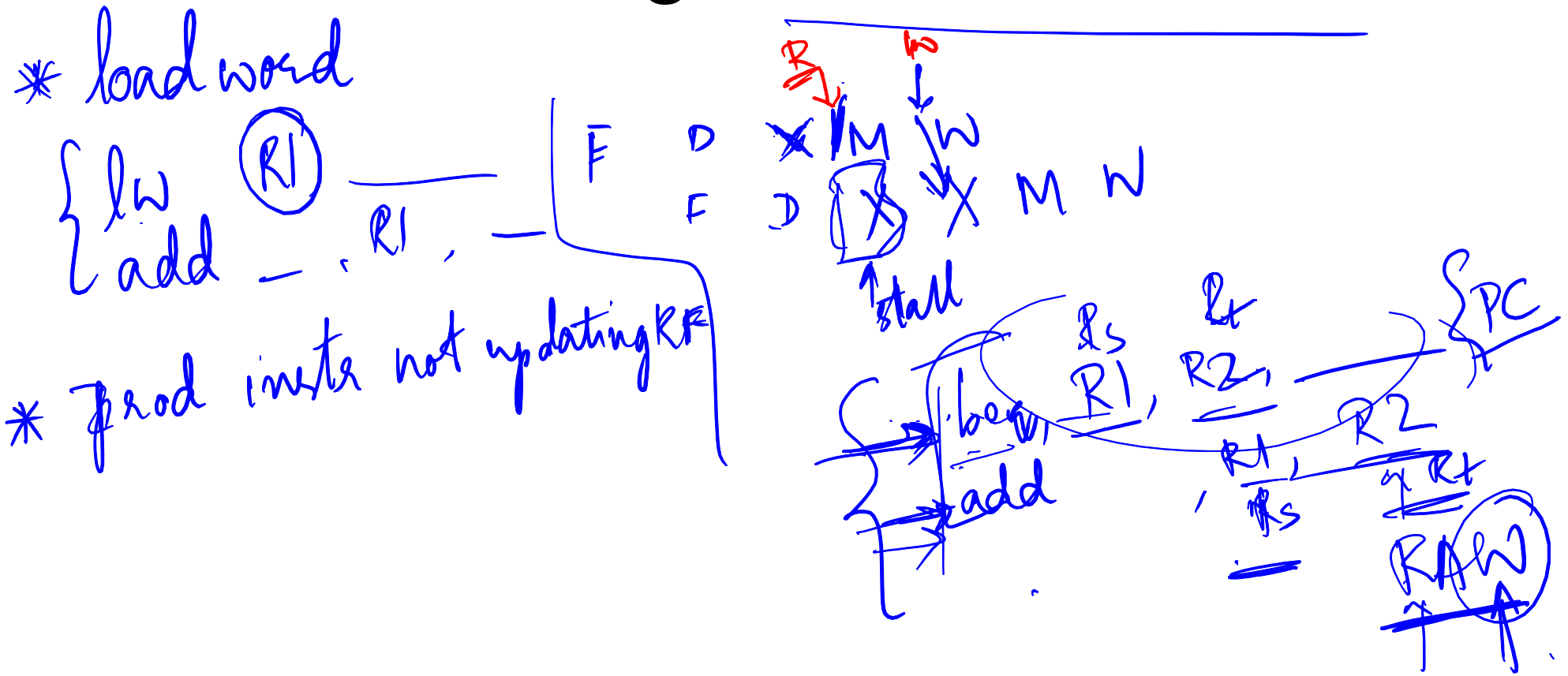
Pipeline with Forwarding



Pipeline with Forwarding



Forwarding – Corner Cases



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
```

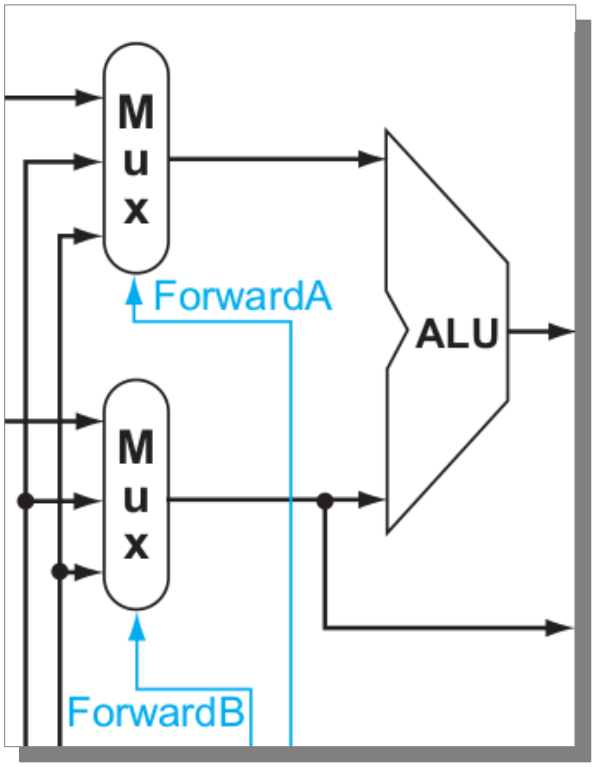
- Use of $0 as an operand must yield a value of 0
- Forward only if:

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

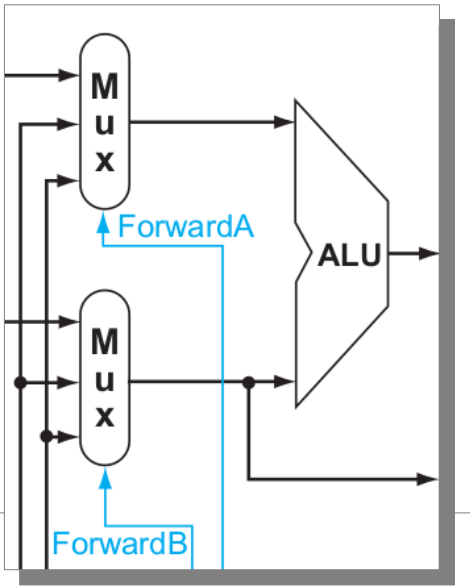
Forwarding Unit Logic

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

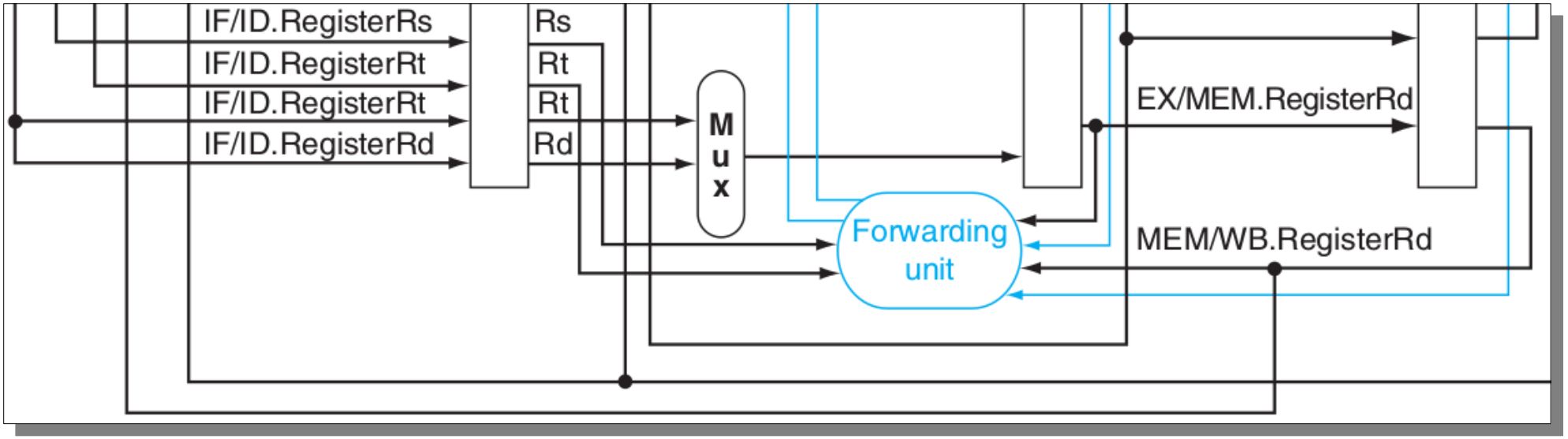
Input A to ALU

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

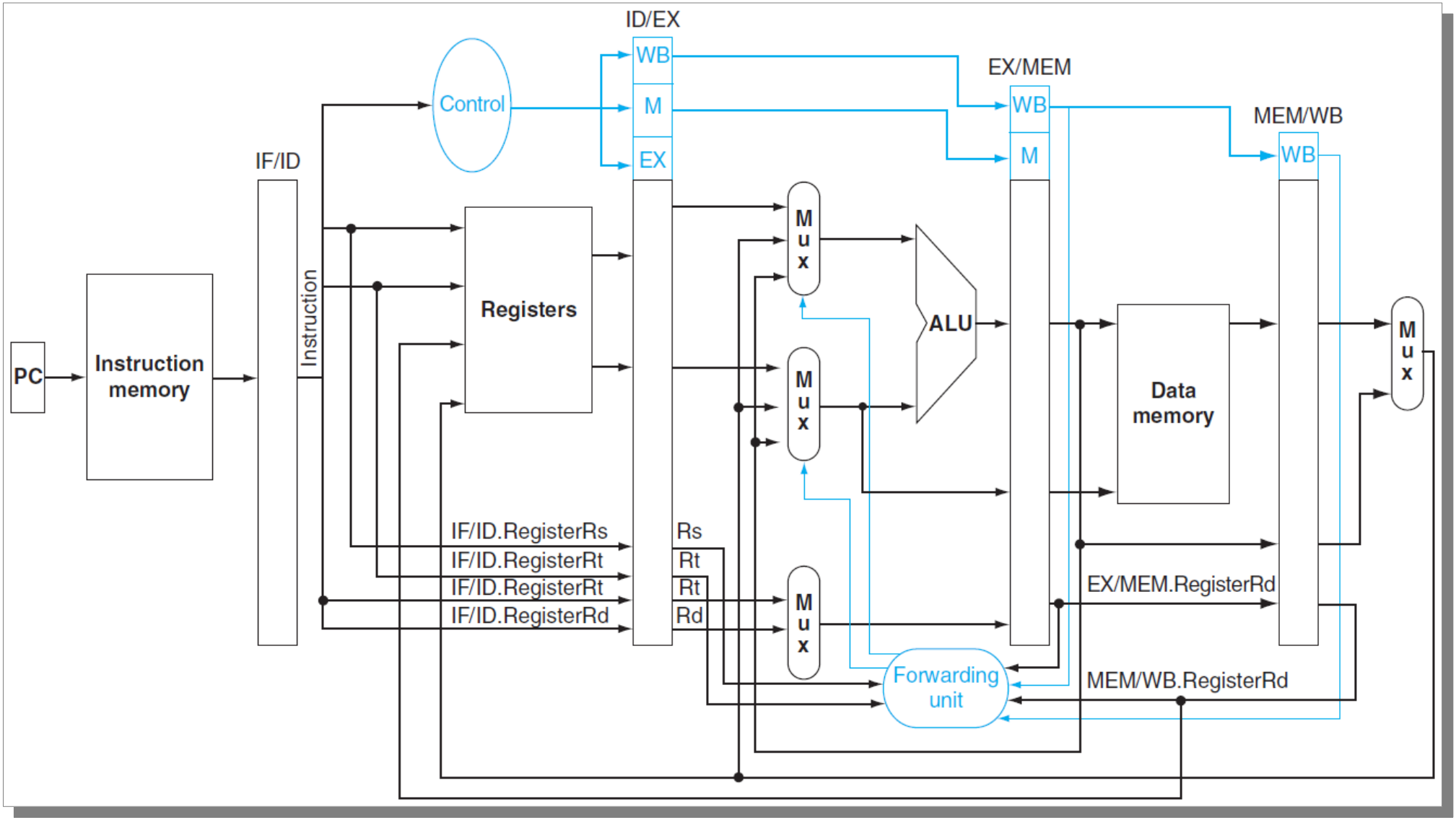
Input B to ALU

```
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

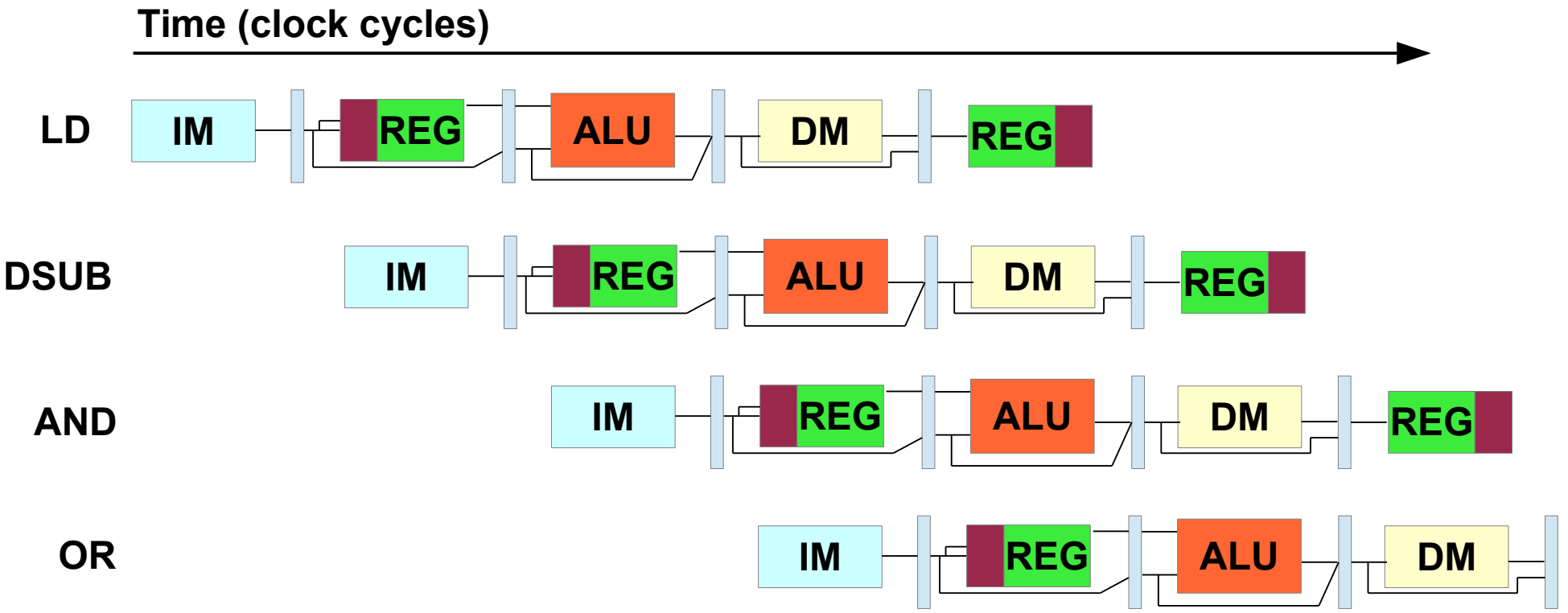


Data Hazard – Load Instruction

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

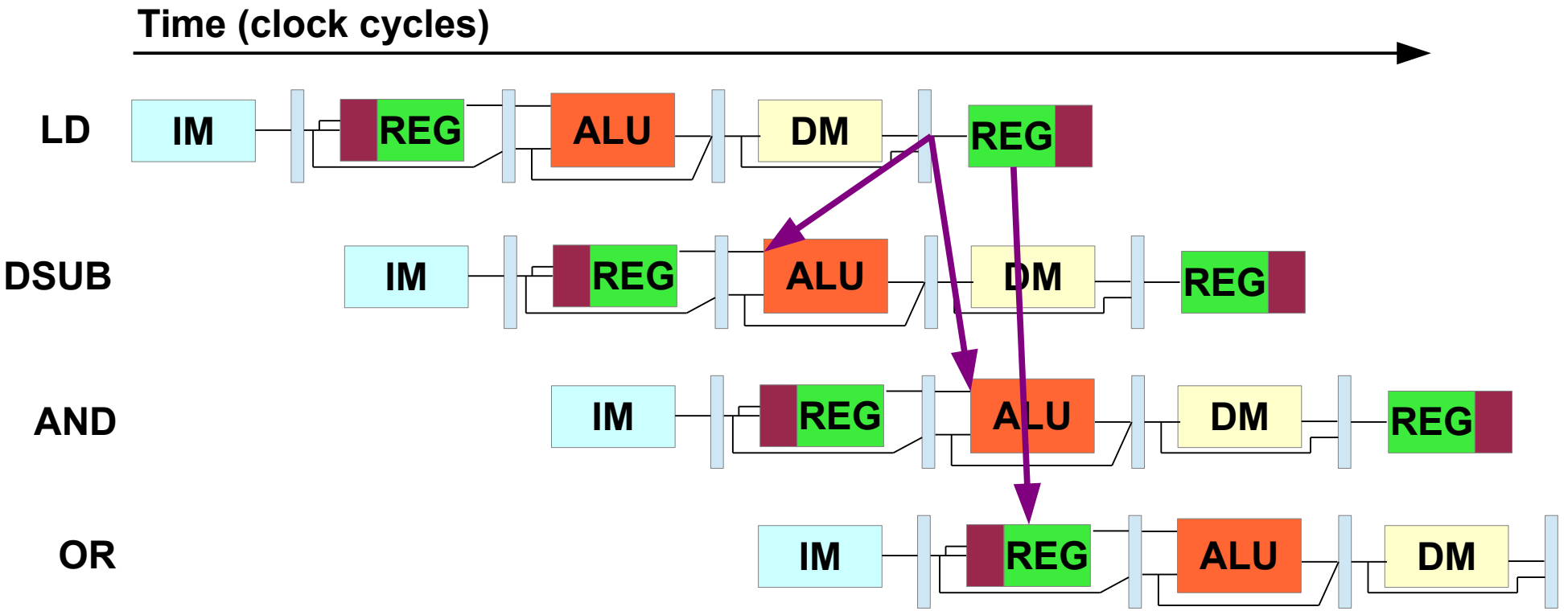
Data Hazard – Load Instruction

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



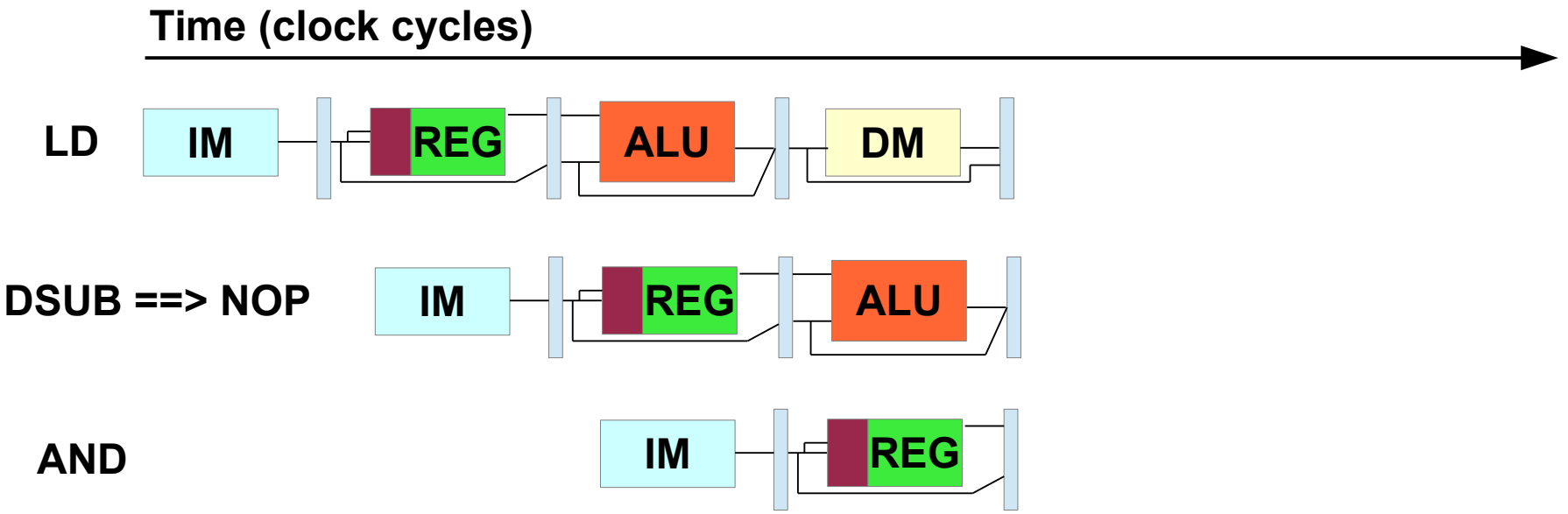
Data Hazard – Load Instruction

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



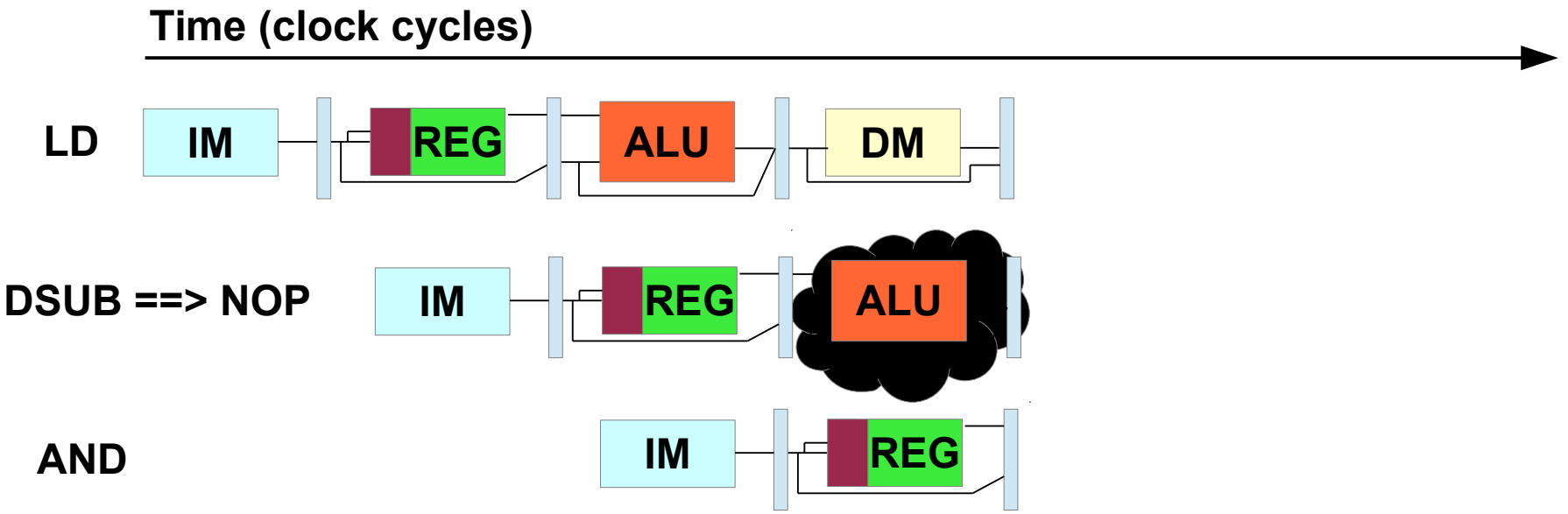
Data Hazard – Load Instruction

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



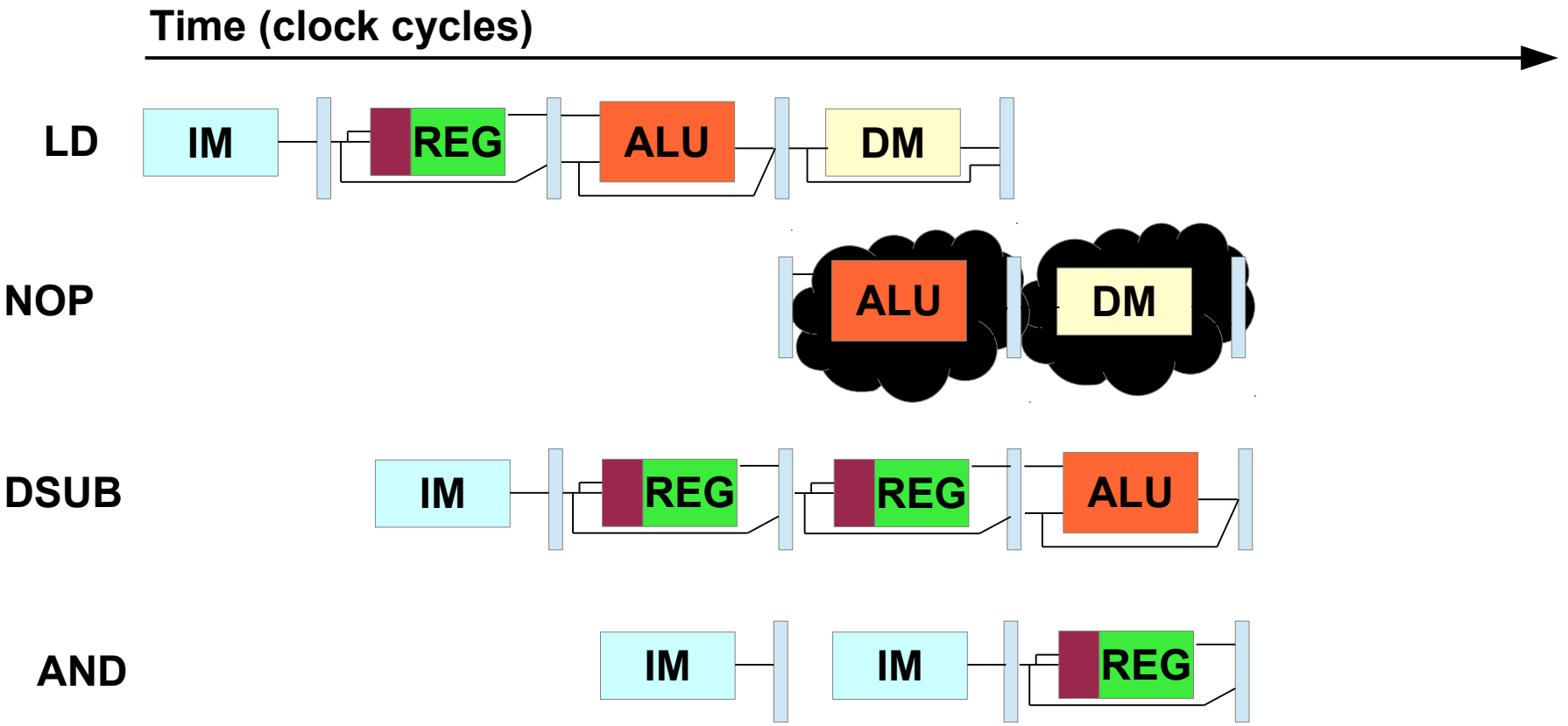
Data Hazard – Load Instruction

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



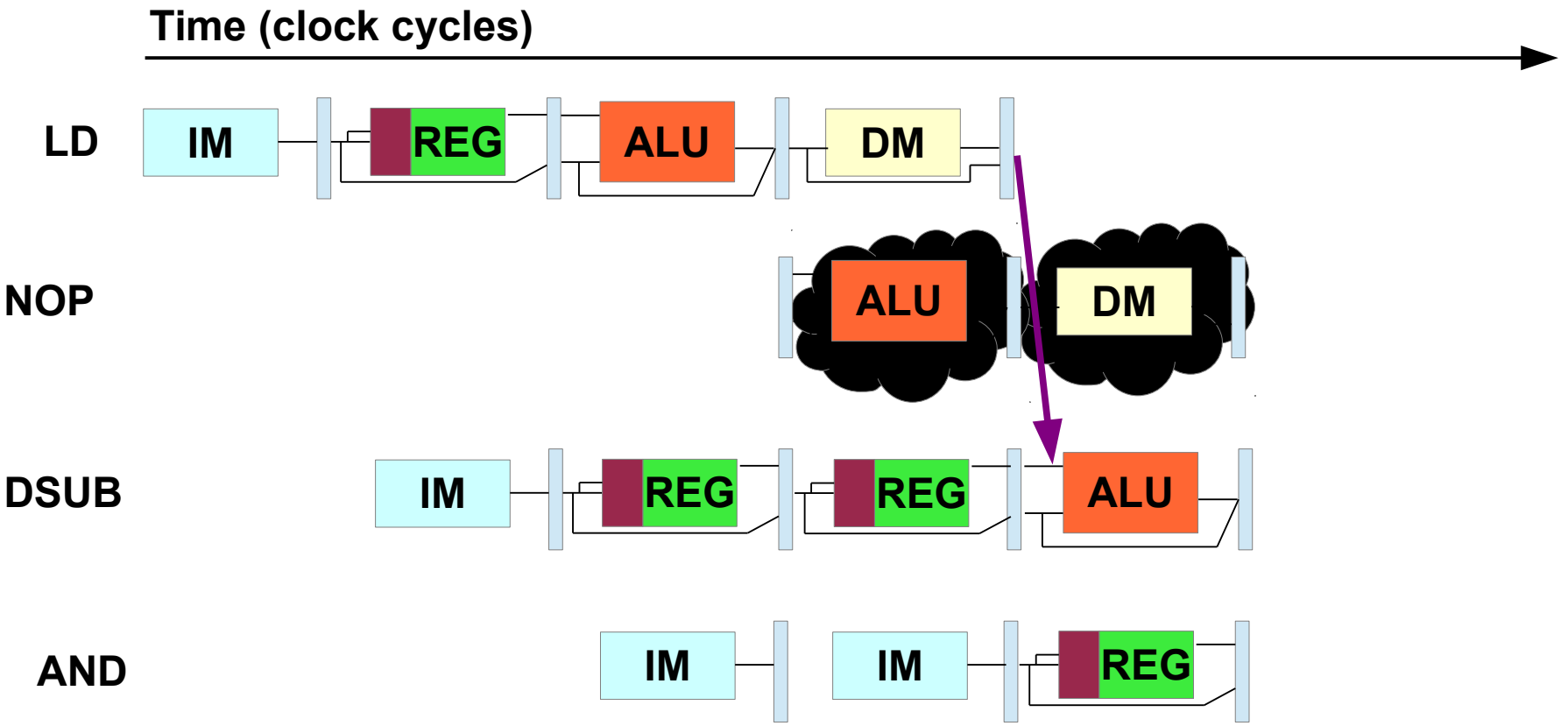
Data Hazard – Load Instruction

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



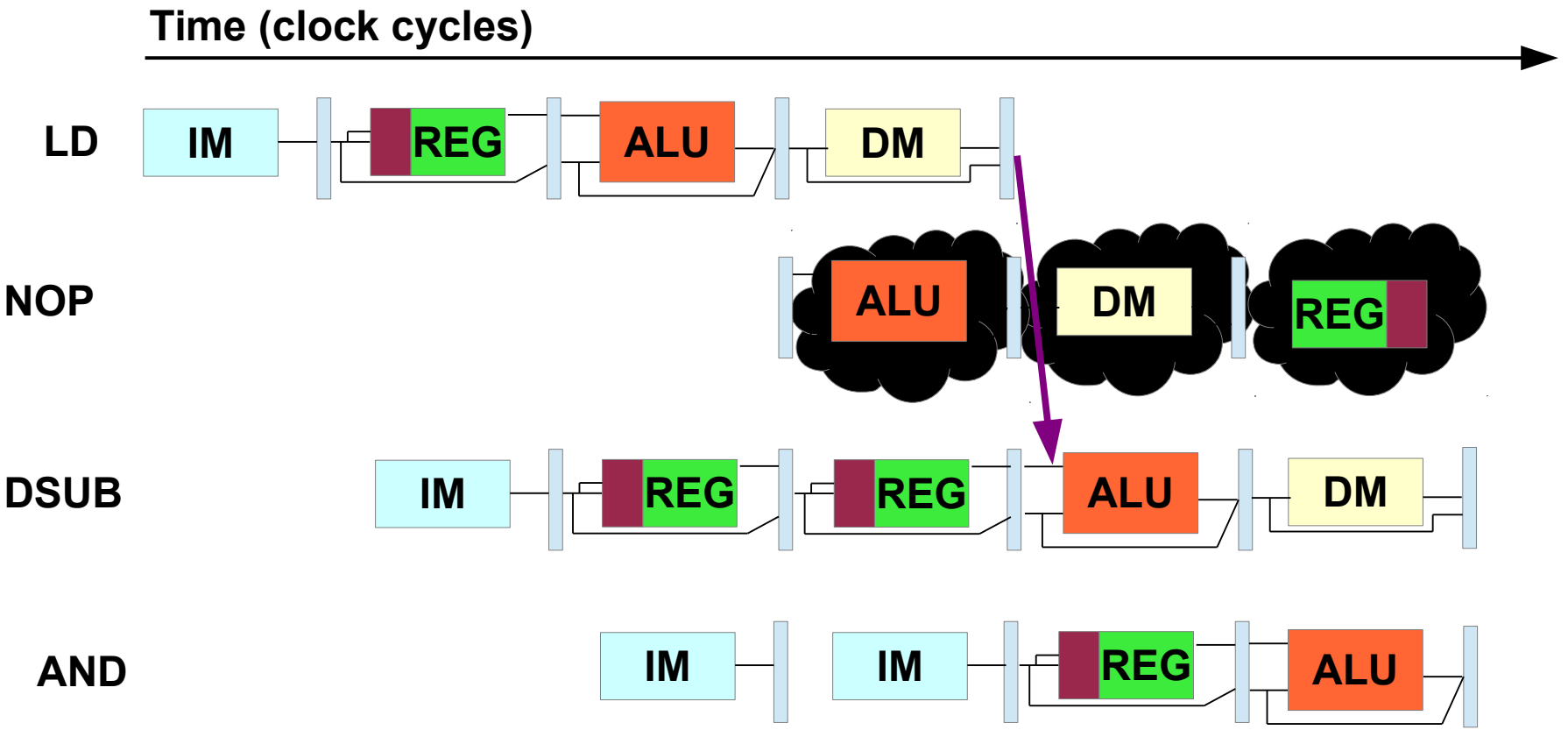
Data Hazard – Load Instruction

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



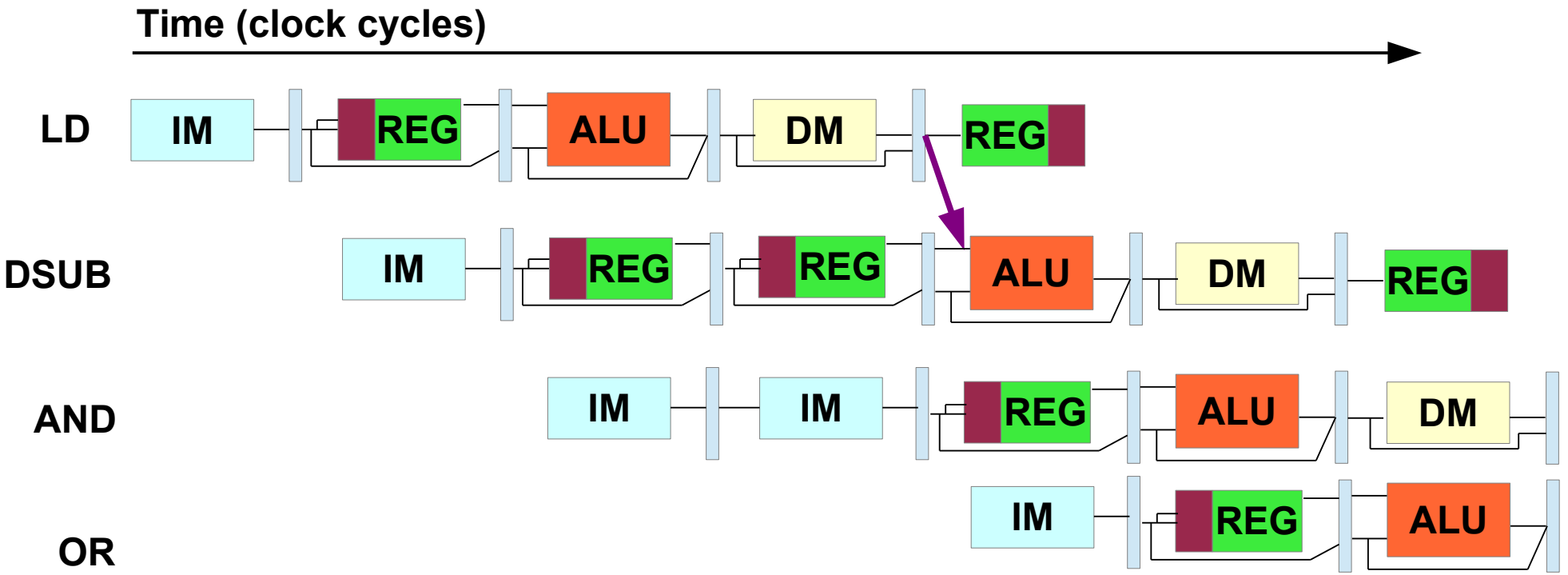
Data Hazard – Load Instruction

```
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

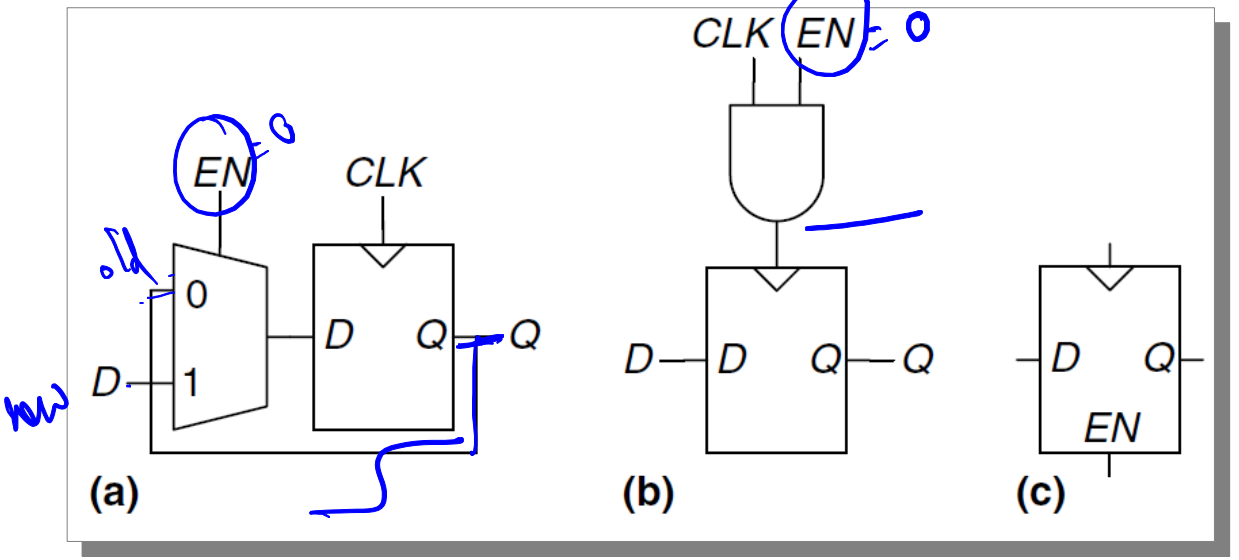
- Input operand required latest in EX. If input op not ready, stall at EX.
  - EX can’t progress until input is available
- EX stalls ==> ID and IF can’t progress

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

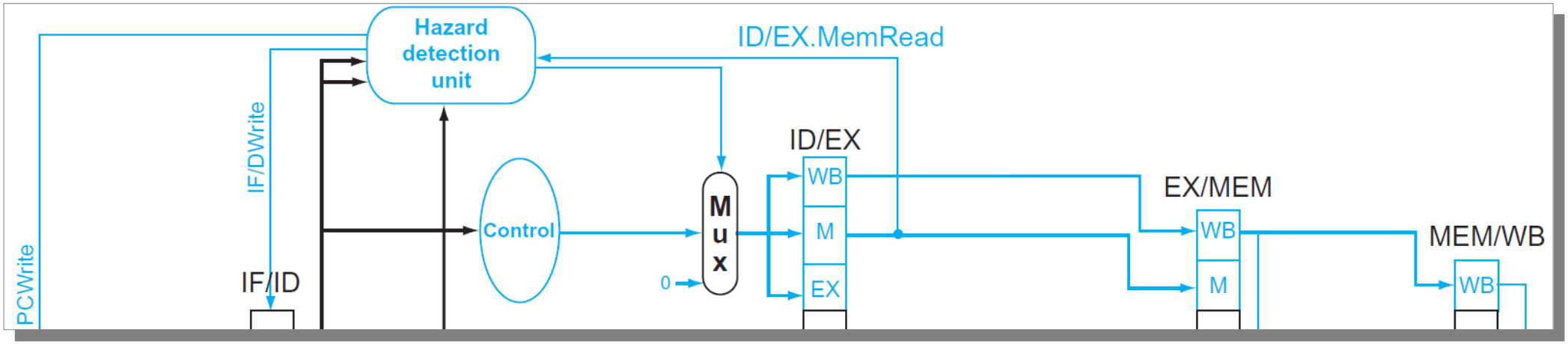
- Input operand required latest in EX. If input op not ready, stall at EX.
  - EX can't progress until input is available
- EX stalls ==> ID and IF can't progress
- To stall IF and ID: prevent PC update
  - IF/ID pipeline register does not change
-



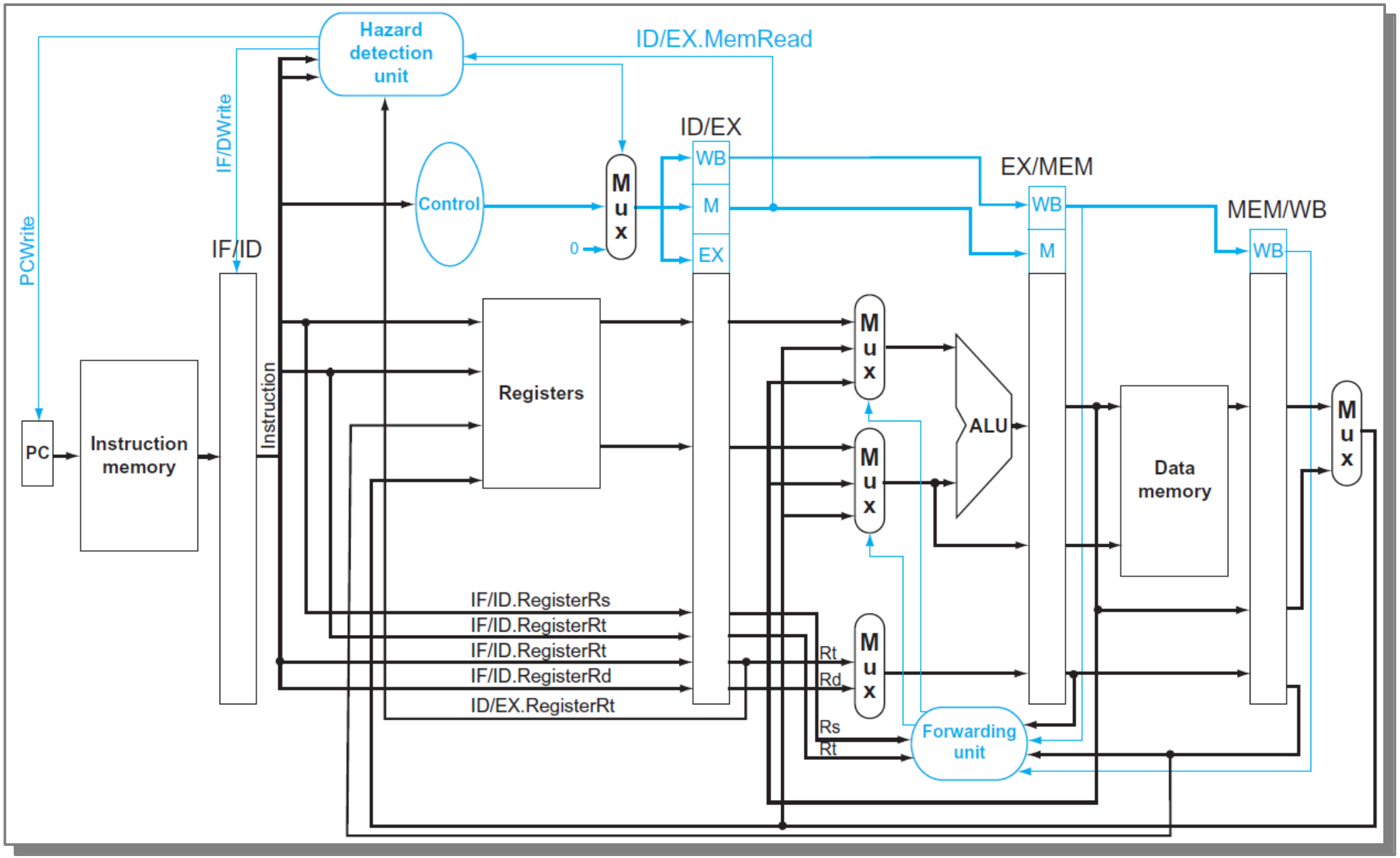
Pipeline Stall

- 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 that 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

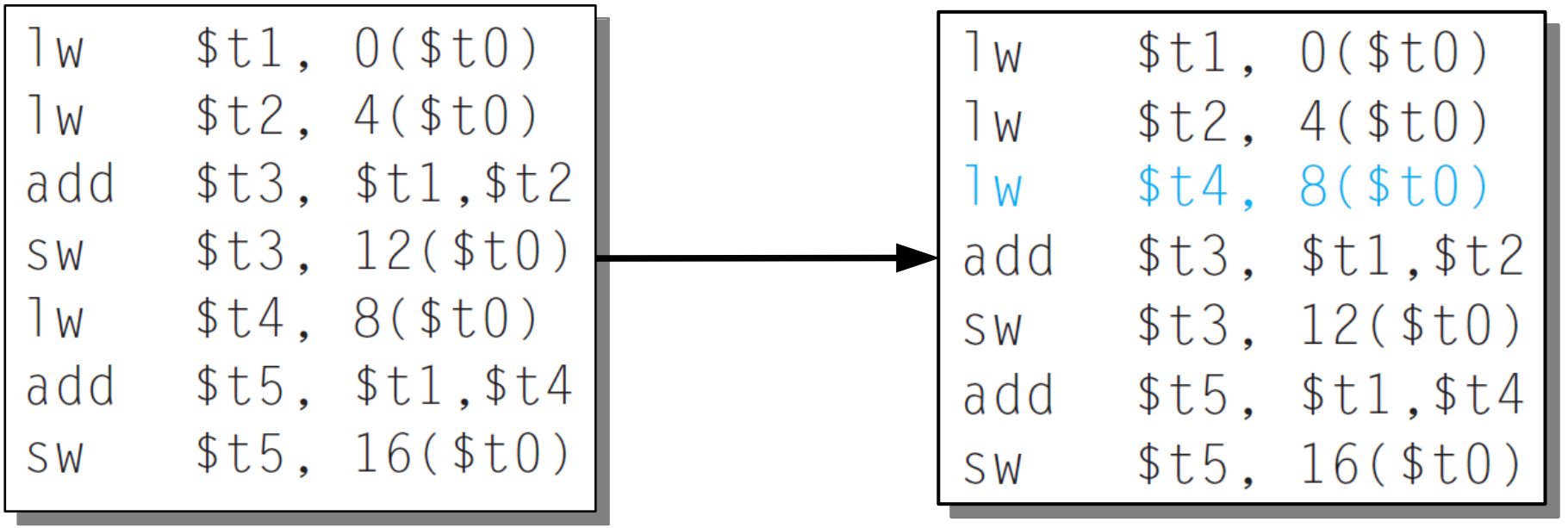
Data Hazard – Solutions

- 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



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