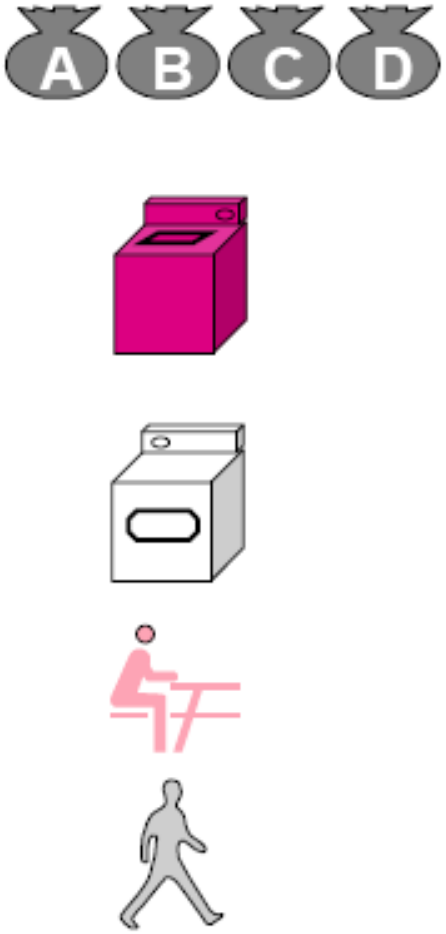
Enhancing Performance: Pipeline

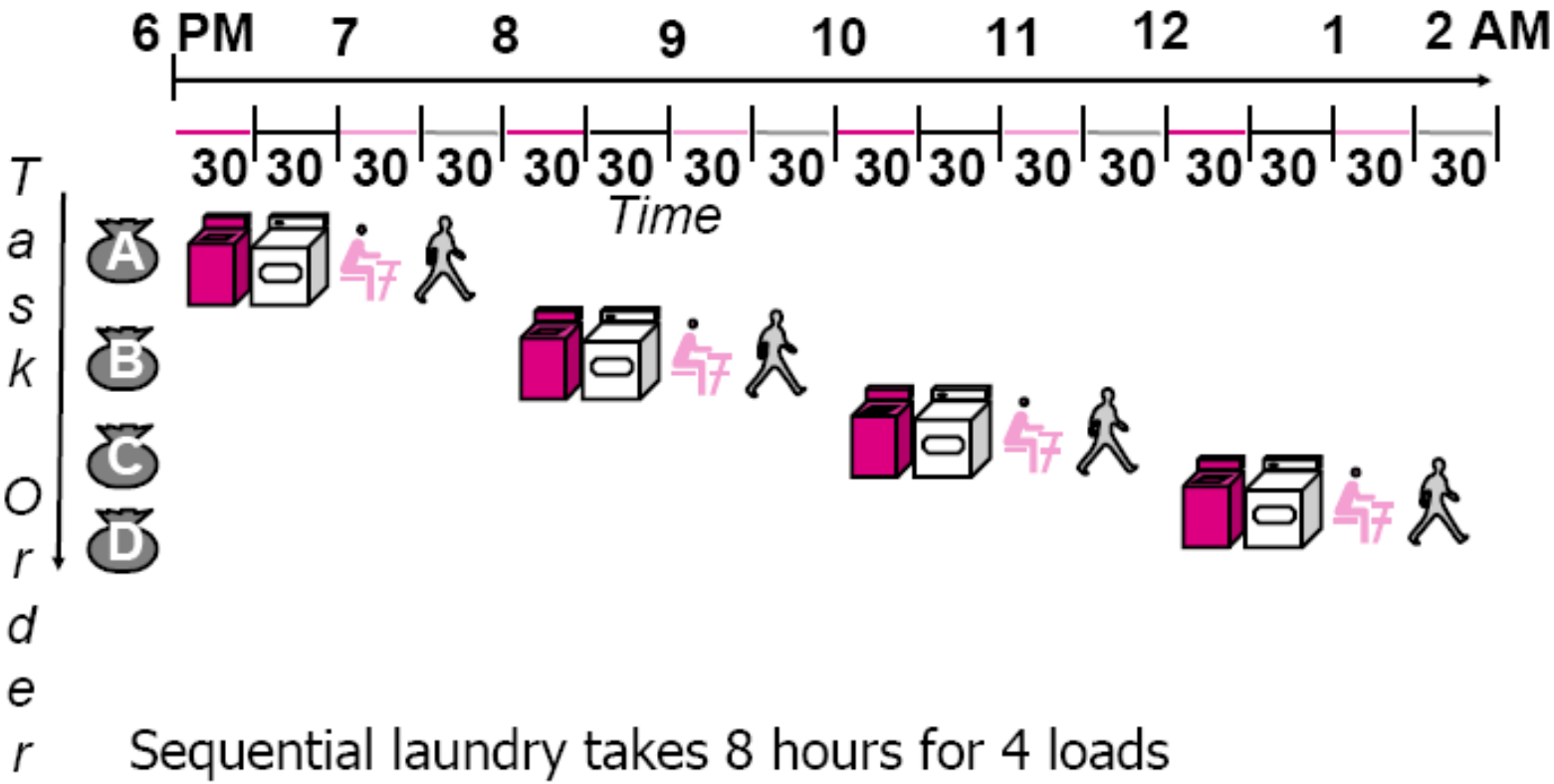
Pipelining in real life

Laundry Example

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 30 minutes
- "Folder" takes 30 minutes
- "Stasher" takes 30 minutes to put clothes into drawers



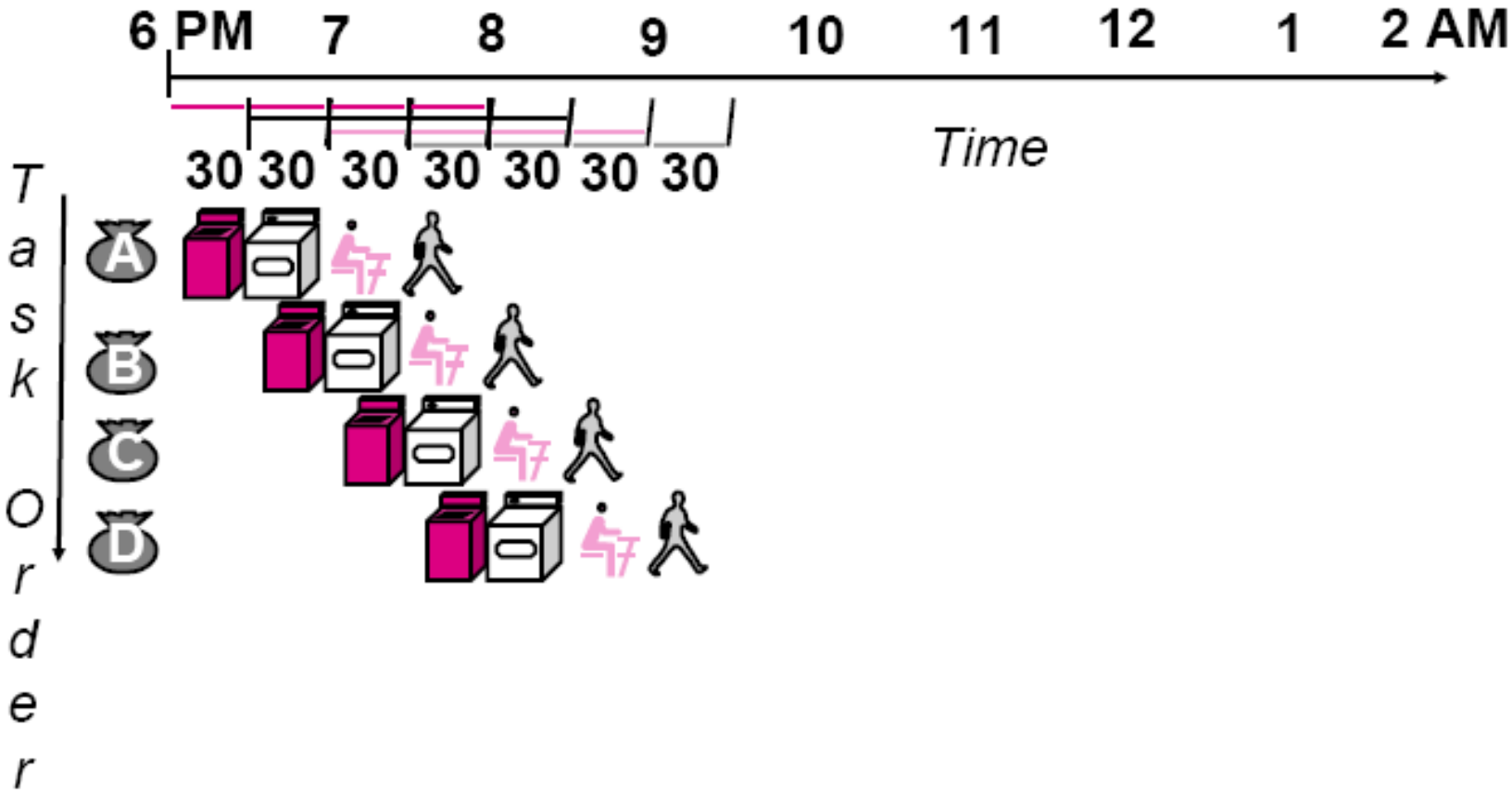
Sequential processing



Sequential laundry takes 8 hours for 4 loads

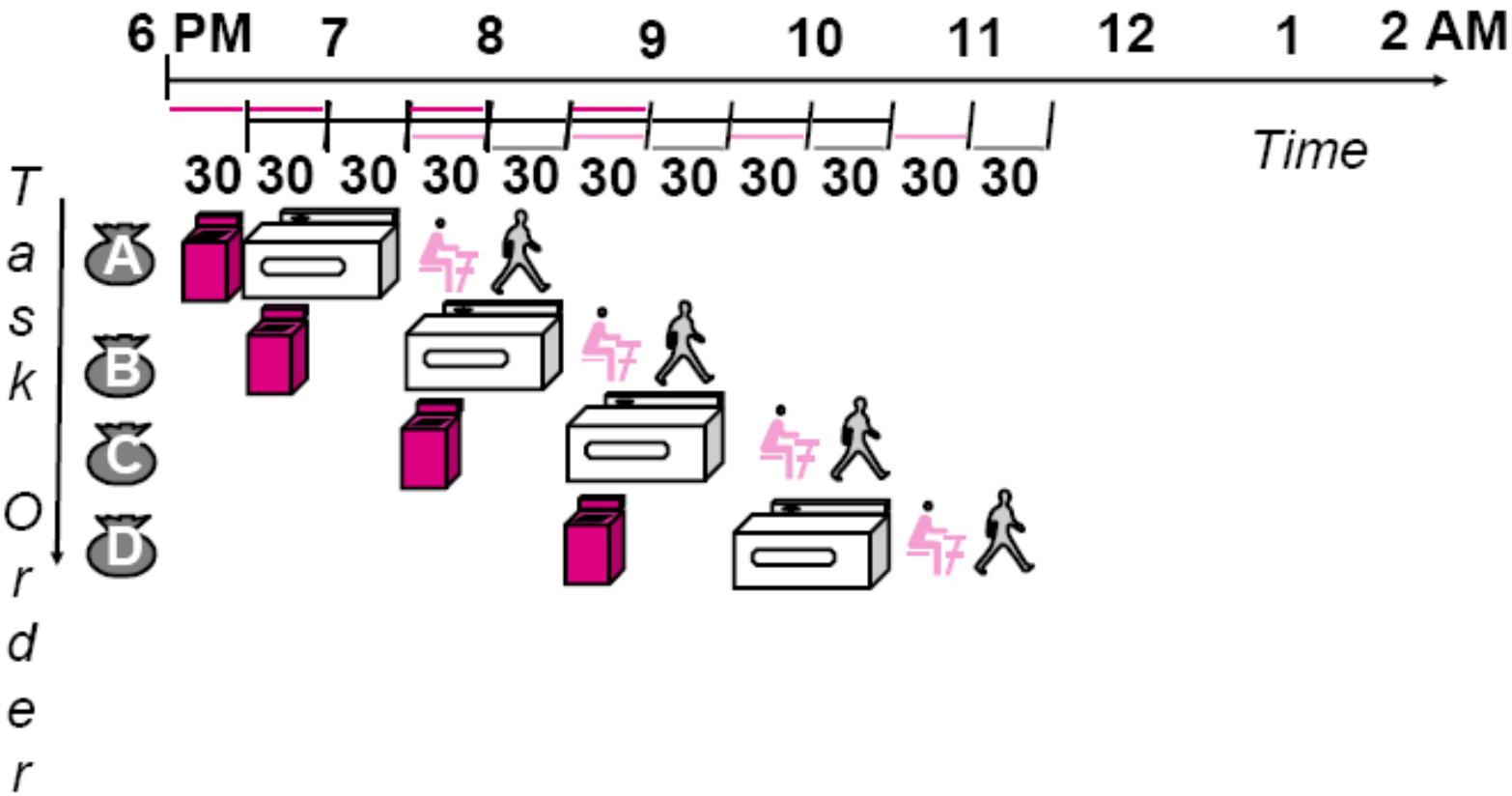
If they learned pipelining, how long would laundry take?

Pipelined processing



- Pipelined laundry takes 3.5 hours for 4 loads!

Unbalanced Pipeline



5.5 Hours. What is going on here?

Pipelining Principles

- **Pipelining does not help the *latency* of a single task, it helps the *throughput* of the entire workload.**
- **The pipeline rate is limited by the *slowest* pipeline stage.**
- ***Multiple* tasks operating simultaneously.**
- **Potential speedup = *number of pipe stages***
- **Unbalanced lengths of pipe stages reduces speedup.**
- **Time to "fill" pipeline and time to "drain" it reduces speedup.**

Hardware Pipelining

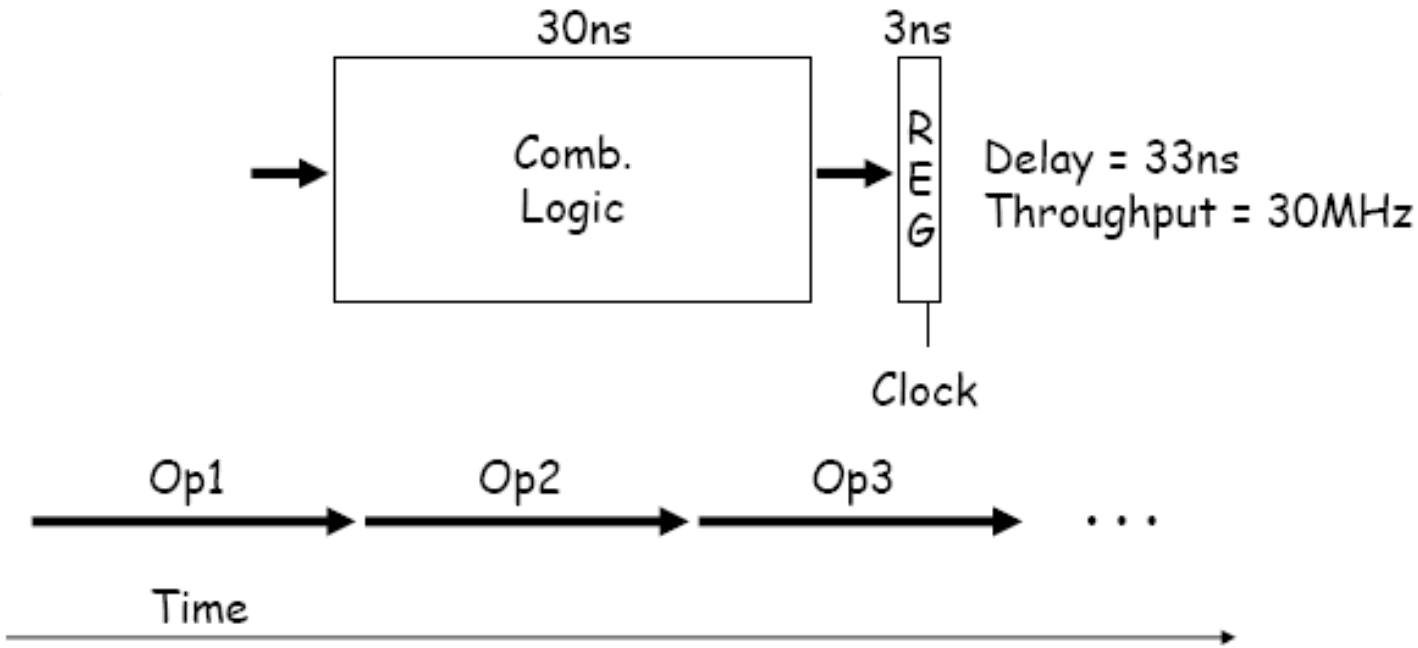
- **Pipelining is an implementation technique that exploits parallelism among instructions in a sequential instruction stream.**
- **A major advantage of pipelining over "parallel processing" is that it is not visible to the programmer.**
- **In a computer system, each pipeline stage completes a part of the instruction being executed.**
- **The time required between moving an instruction one step down the pipeline is a *machine (clock) cycle*.**
- **The length of a clock cycle is determined by the time required for the slowest stage to proceed.**

Hardware Pipelining

- **The hardware architect should try to *balance* the length of each pipeline stage.**
- **In a perfect pipeline, the time per instruction on the pipeline computer is:**

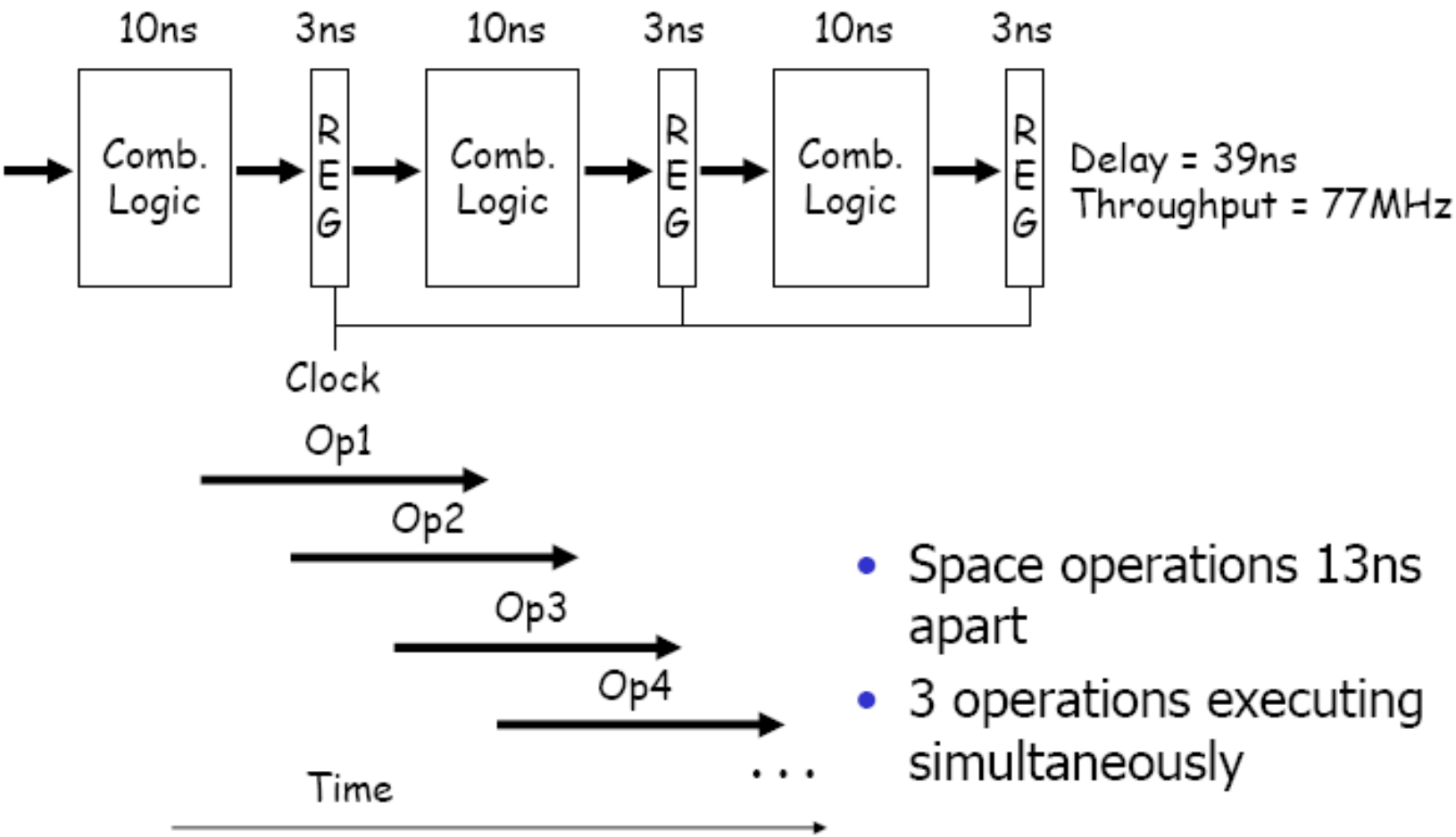
  Time per instruction on unpipelined computer
  Number of pipe stages
- **If we have n stages, we could speedup the execution n times — speedup = n.**
- **In practice, the pipeline stages will not be perfectly balanced - and there are additional over-heads. But we can get *close* to the ideal case.**

Unpipelined System



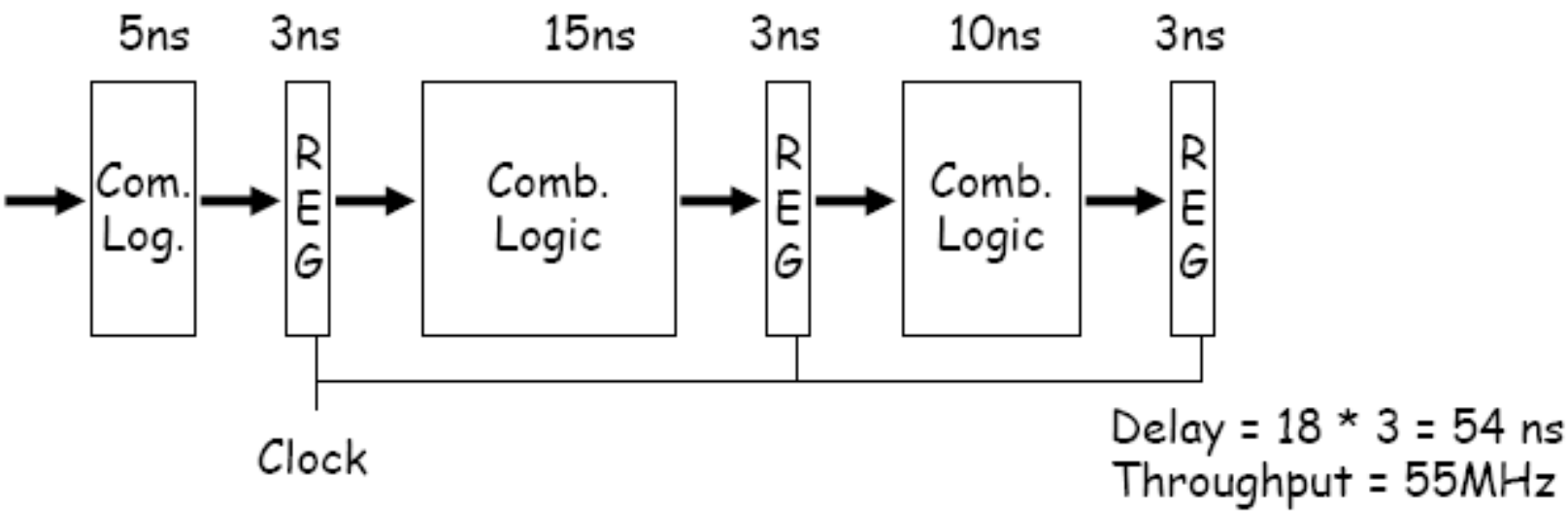
- One operation must complete before next can begin
- Operations spaced 33ns apart

3 Stage Pipelined System



- Space operations 13ns apart
- 3 operations executing simultaneously

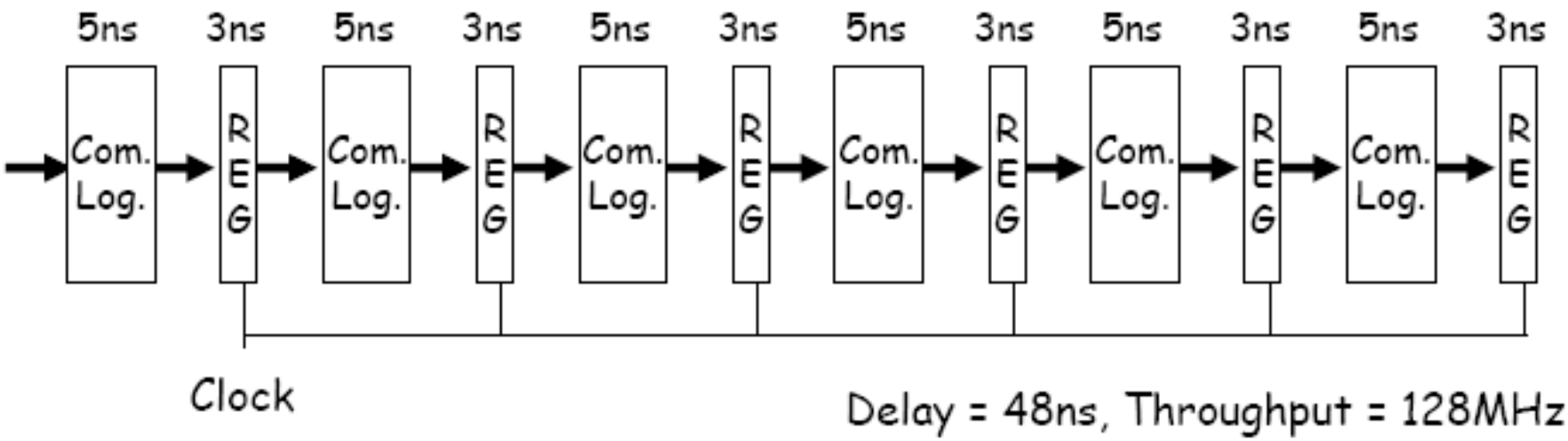
3 Stage Unbalanced Pipelined System



- Throughput limited by slowest stage
  Delay determined by clock period * number of stages

- Must attempt to balance stages

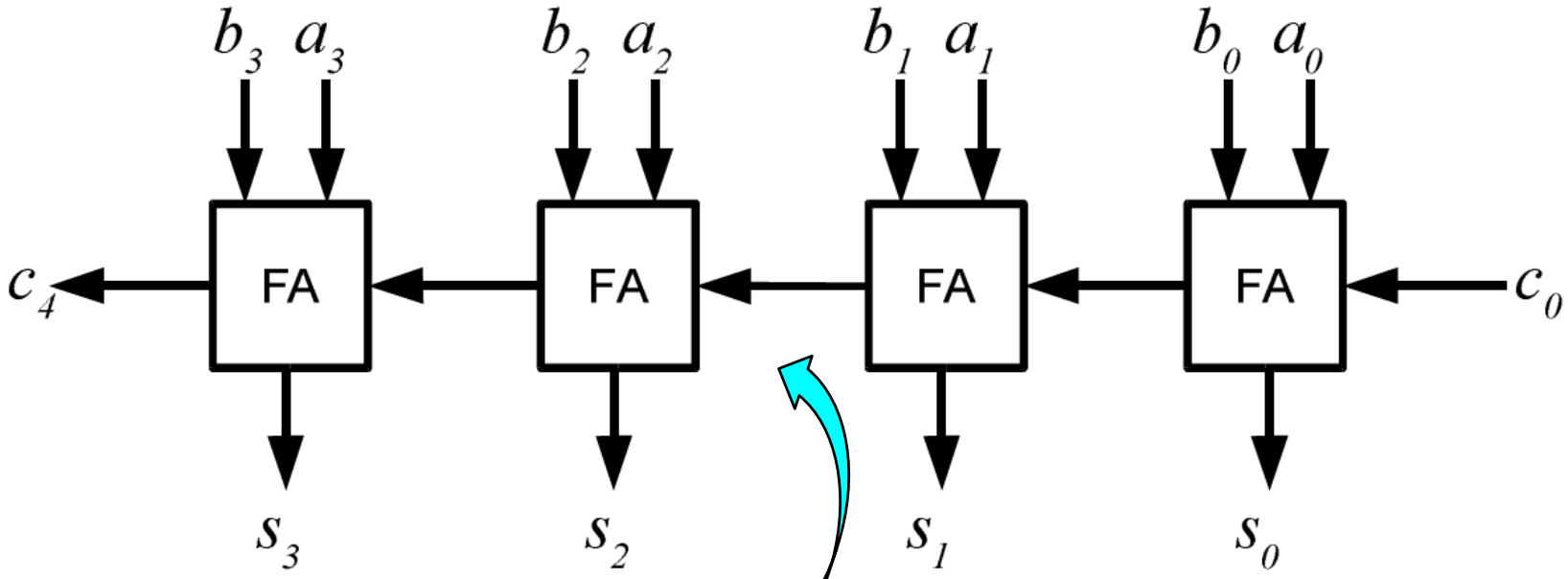
Deep Pipelined System



- Diminishing returns as we add more pipeline stages
- Register delays become limiting factor
  - Increased latency
  - Small throughput gains

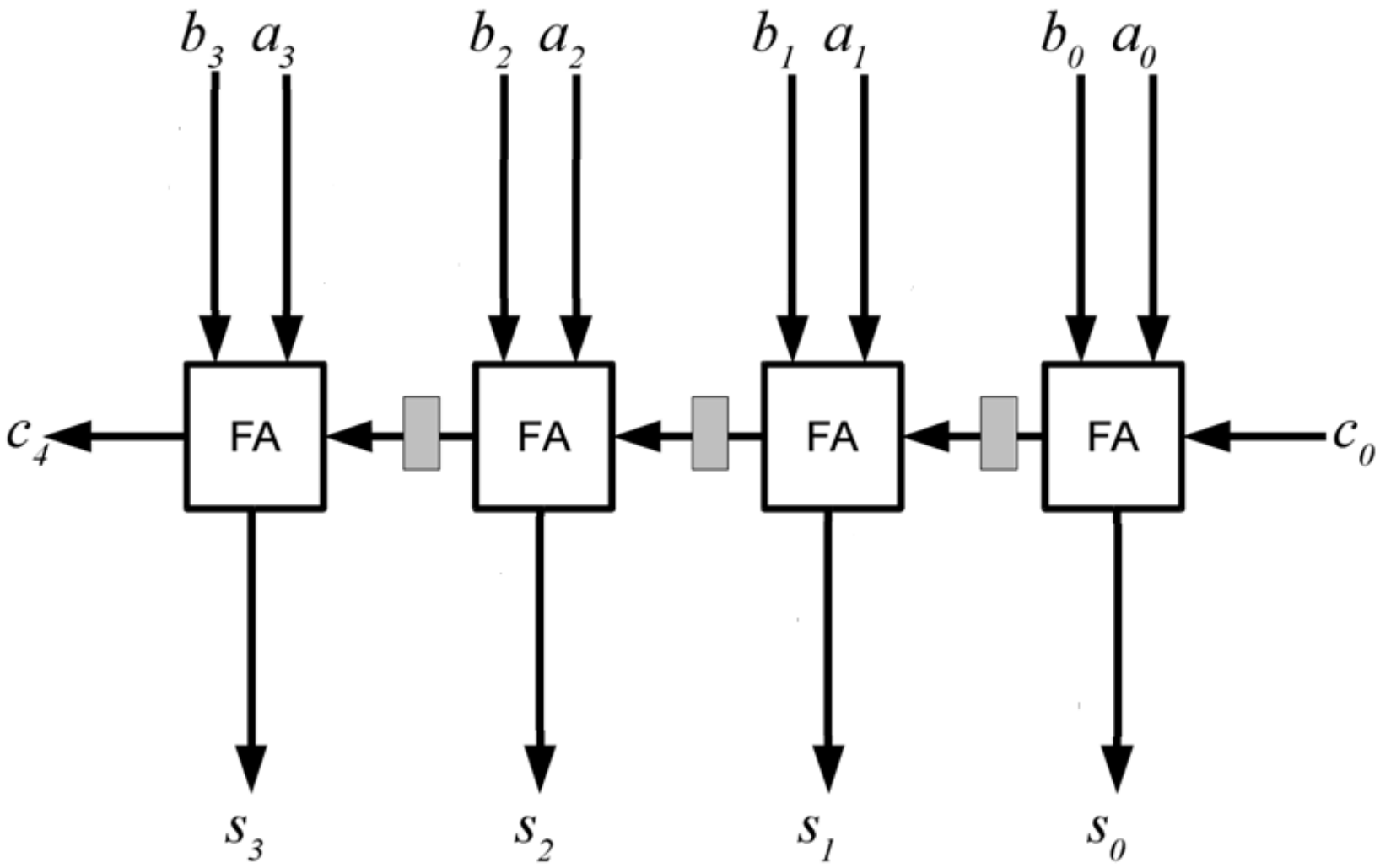
More complicated pipeline example

- **Ripple-Carry adder (n=4)**

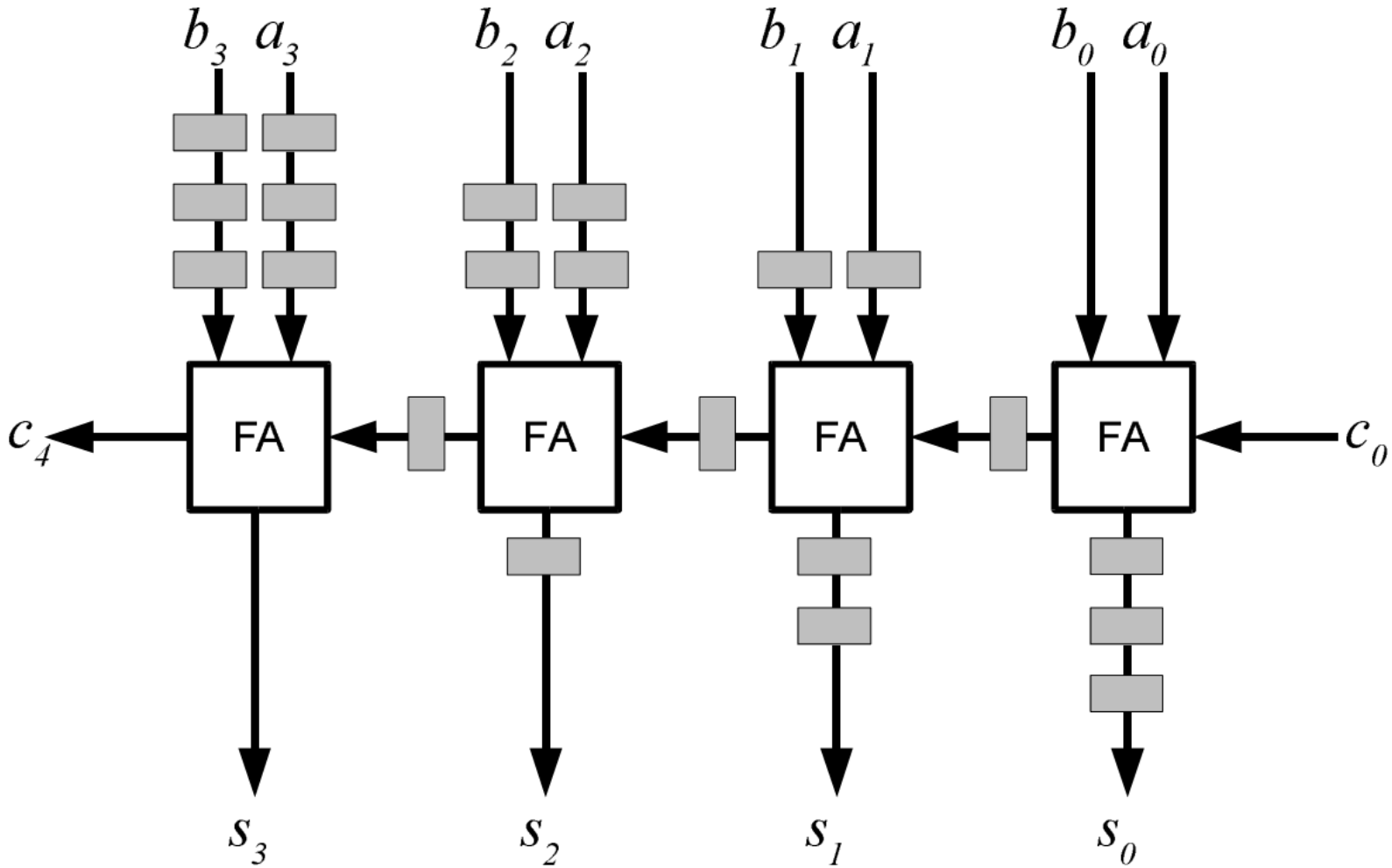


The bottleneck of ripple-carry adder is carry chain!

What's wrong?

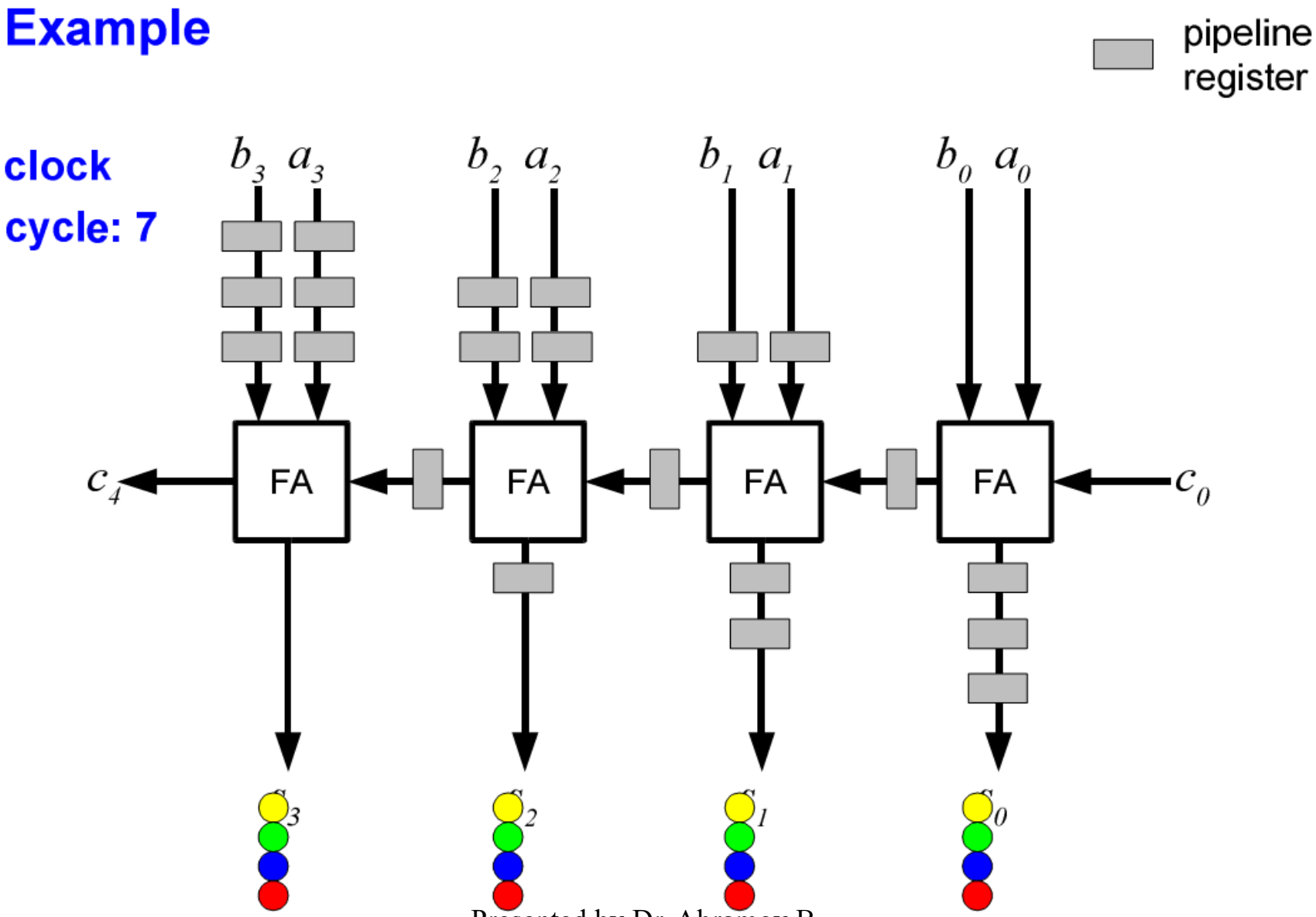


The balanced solution



Pipeline in action

Example



Pipeline in real system- CPU

CPU Life Cycle:

- At most one of the following can occur within one clock cycle:
  - » One ALU operation
  - » One register file access (1 write and 2 reads)
  - » One memory access (Actually will take multiple clock cycles before we get a cache)
- Our execution stages will be separated into 5 cycles
  - » Instruction fetch
    - Fetch new instruction from memory, compute next PC value
    - Performed for all instructions
  - » Decode
    - Fetch register values from register file, compute branch address
    - Performed for all instructions

Pipeline in real system- CPU

- **Execute**
  - **Perform A/L/S operation for A/L/S R and I-type instructions**
  - **Compute address for load and store instructions**
  - **Determine if branch is taken for branch instructions**
  - **Jump for jump instructions**
  - **Link for branch-and-link and jump-and-link instructions**
- **Memory**
  - **Access memory for load and store instructions (skip for all others)**
- **Write back**
  - **Write register result back to register file for A/L/S/load instructions (skip for all others)**

Pipeline in real system – MIPS CPU

- **Instructions in a MIPS processor are executed in at most five clock cycles as follows:**

1. ***Instruction fetch cycle* (IF)**

```
IR = Memory[PC]
PC = PC + 4
```

2. ***Instruction decode & register fetch cycle* (ID)**

```
A = Reg[IR[25-21]]
B = Reg[IR[20-16]]
Target = PC + (sign-extend(IR[15-0]) << 2)
```

Pipeline in real system – MIPS CPU

3. *Execution, memory address computation, or branch completion* (EX)

- **Memory reference**

```
ALUoutput = A + sign-extend(IR[15-0]
```

- **Arithmetic-logical instruction (R-type)**

```
ALUoutput = A op B
Branch If (A == B) PC = Target
```

4. *Memory access or R-type instruction completion cycle* (MEM)

- **Memory reference**

```
memory-data = Memory[ALUoutput]
```

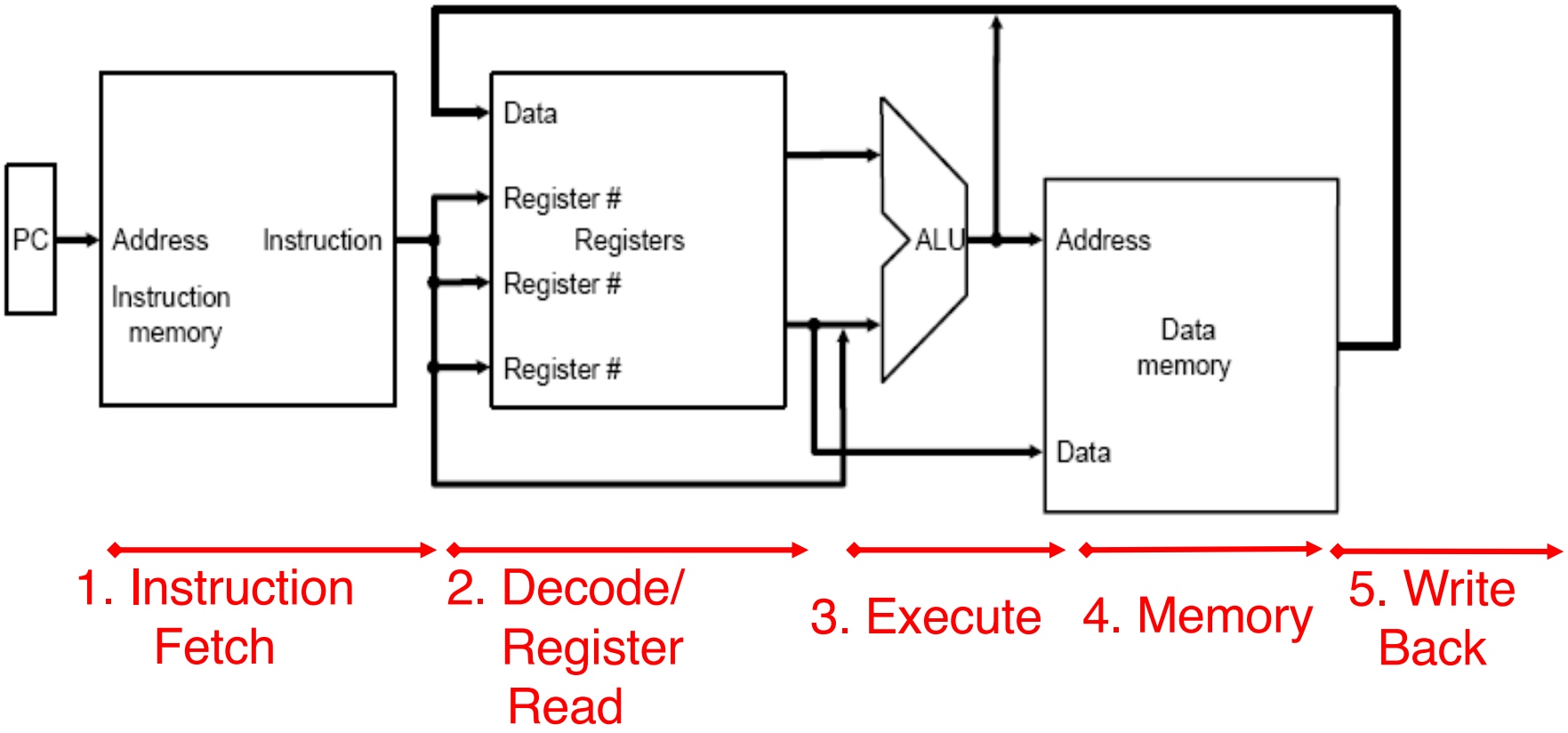
or

```
Memory[ALUoutput] = B
```

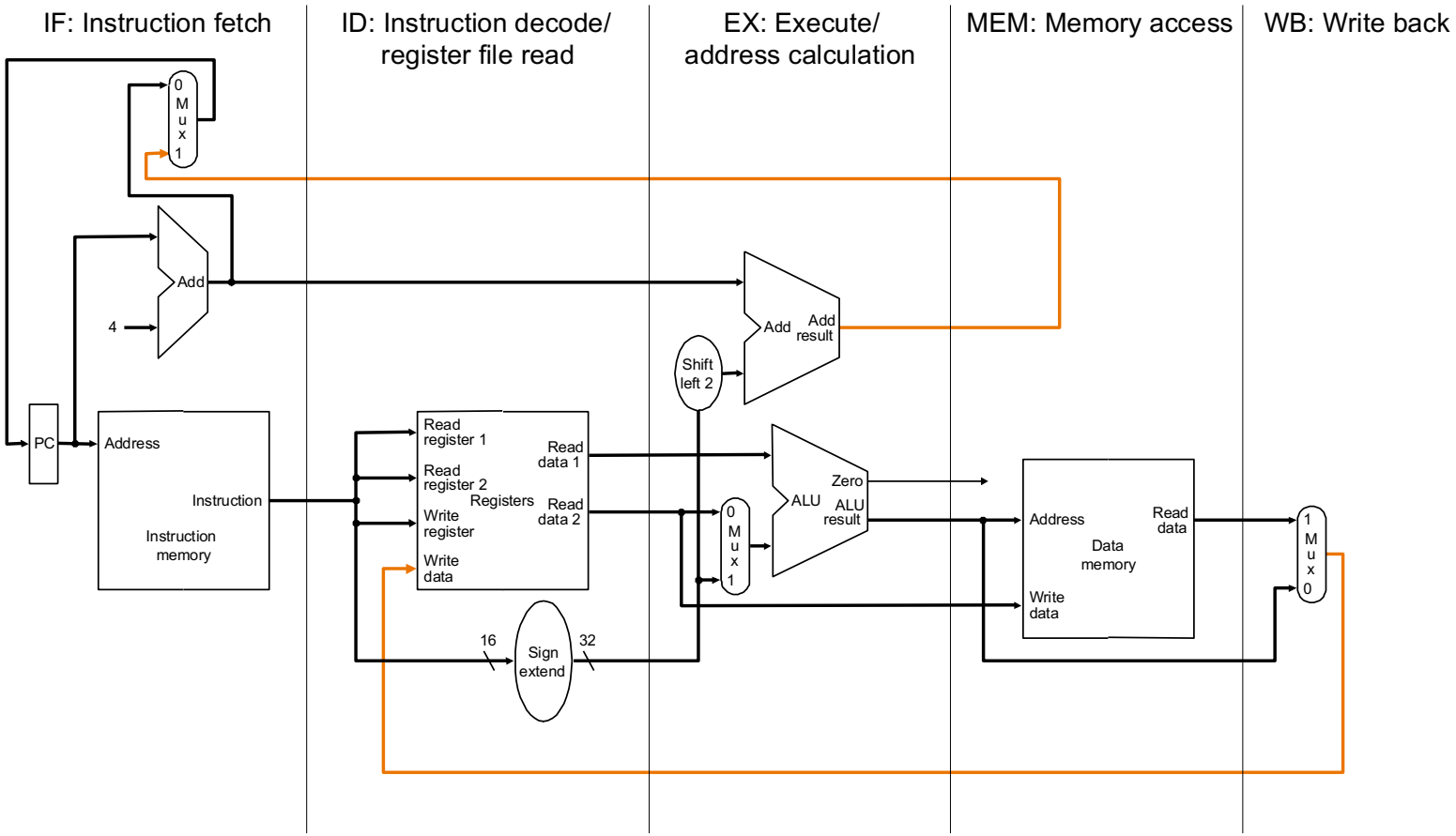
- **Arithmetic-logical instruction (R-type)**

```
Reg[IR[15-11]] = ALUoutput
```

מעבד חד מחזורי:
5 שלבים בזרימת הנתונים



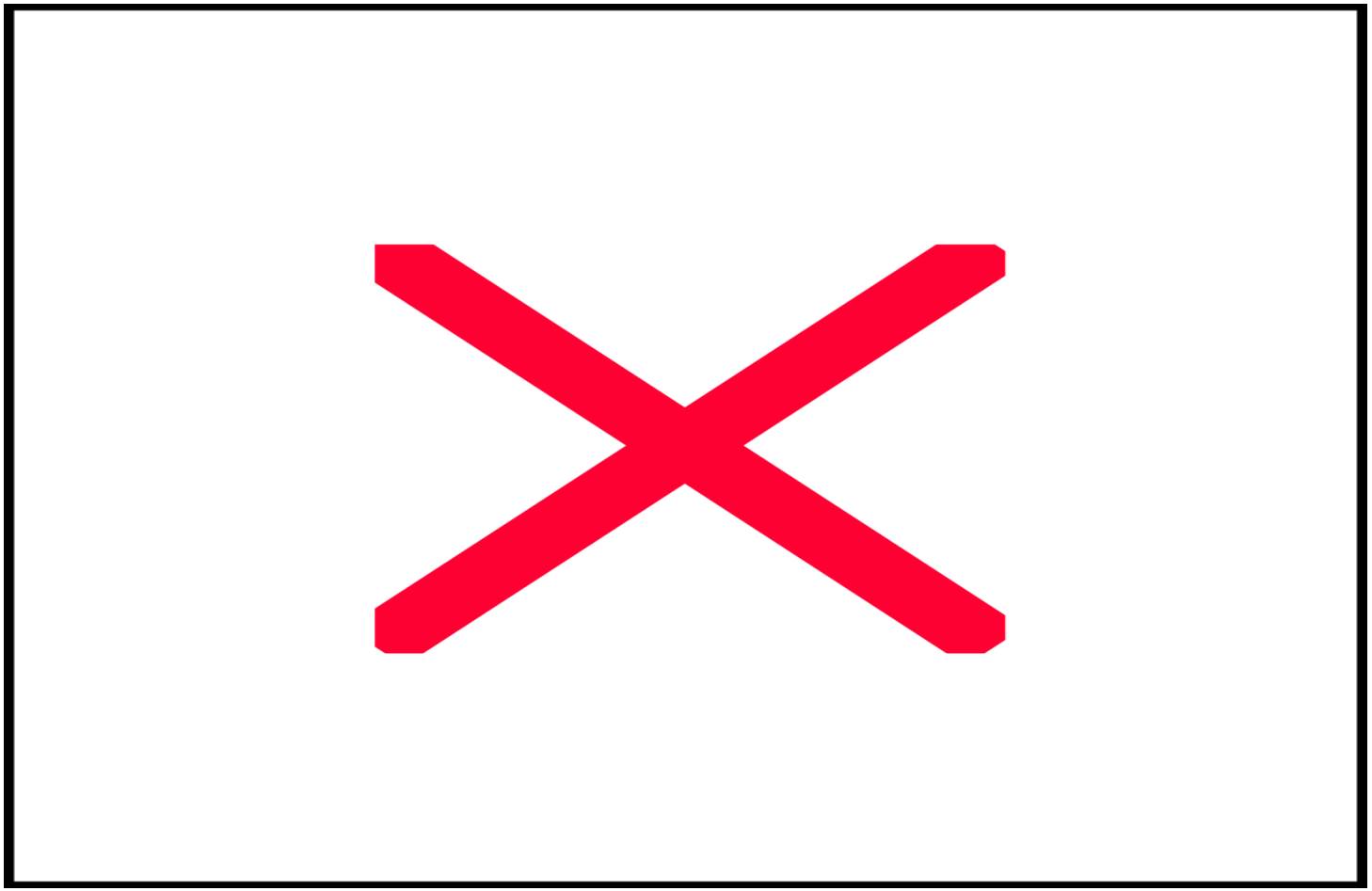
Single Cycle Processor with stages



Pipelining

- חלוקת מסלול הנתונים לחמישה שלבים הופכת אותו למחשב מרובה מחזורים
  - CPI=5
  - רק פקודה אחת מתבצעת בכל רגע
  - רק שלב אחד מתוך חמישה פעיל — בזבוז !!
- האם ניתן להפעיל את כל חמשת השלבים בו זמנית?
  - כן! Pipeline של חמש פקודות שונות המתבצעות בו זמנית

ביצוע פקודות ב"טור"(חד מחזורי) בהשוואה להצנרה



הביצועים

- ביצוע פקודה בודדת:
  - זמן המחזור נקבע ע"פ הפקודה האיטית ביותר (800ps בפקודה lw)
  - השהיה= מחזור אחד= 800ps
  - תדר שעון מקסימלי = 1.25Ghz=1/800ps

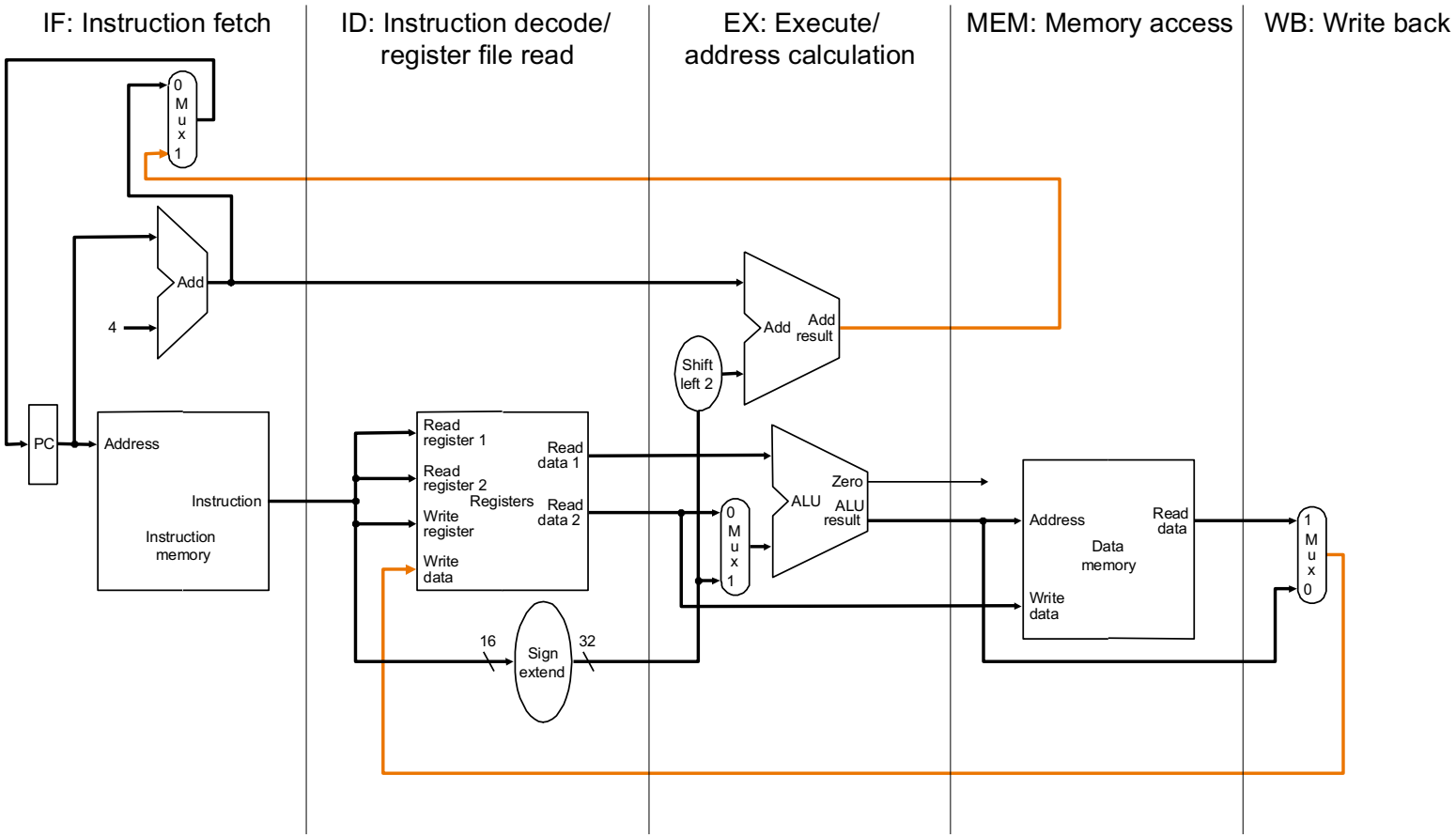
- ביצוע Pipeline
  - זמן המחזור נקבע ע"פ השלב (תחנה) האיטי ביותר (200ps)
  - השהיה= עומק ה pipeline 5 מחזורים = 1000ps
  - תדר שעון מקסימלי = 5Ghz =1/200ps

עבור מספר רב של פקודות נקבל speedup=4

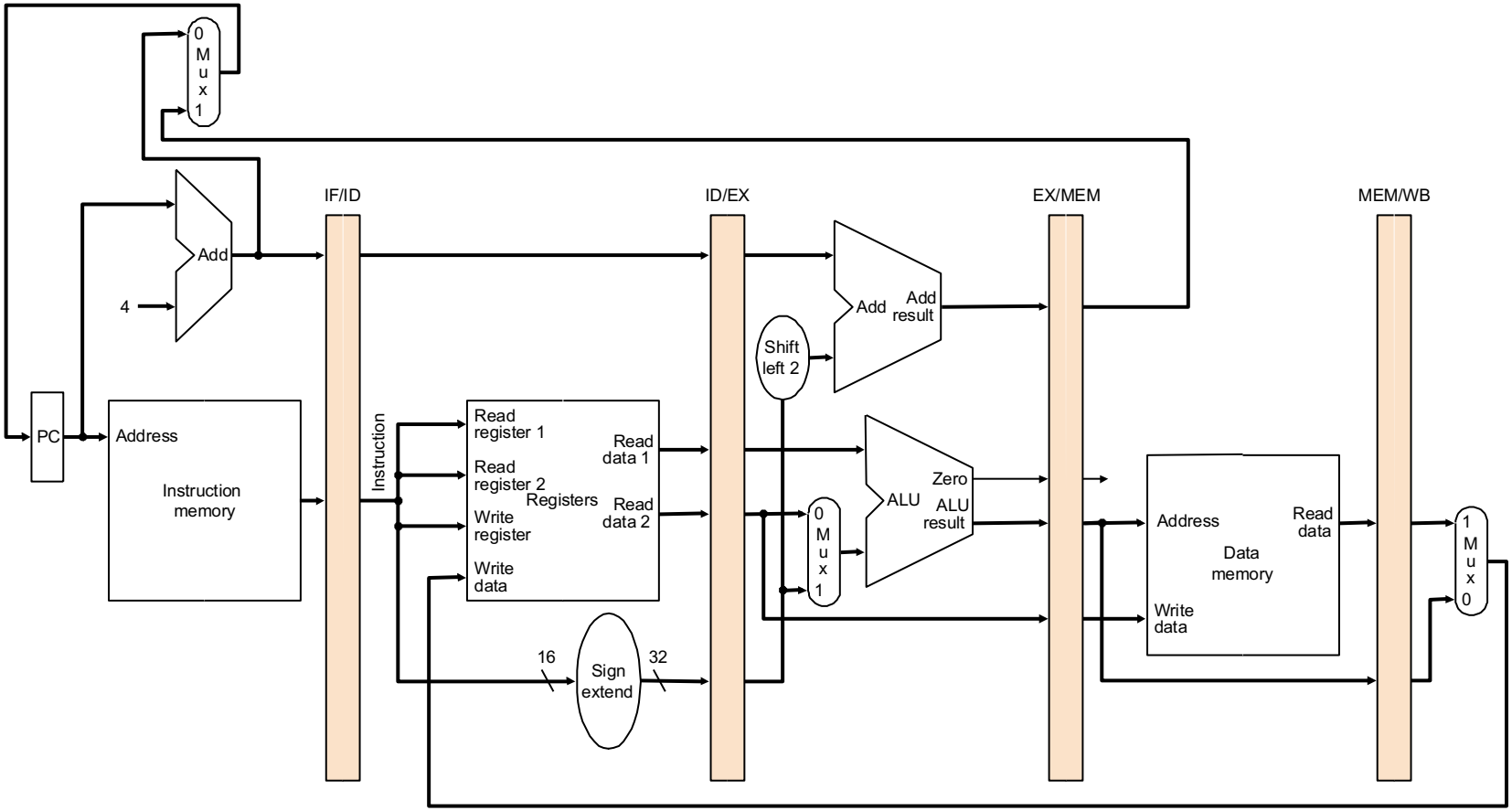
MIPS ISA & Pipeline

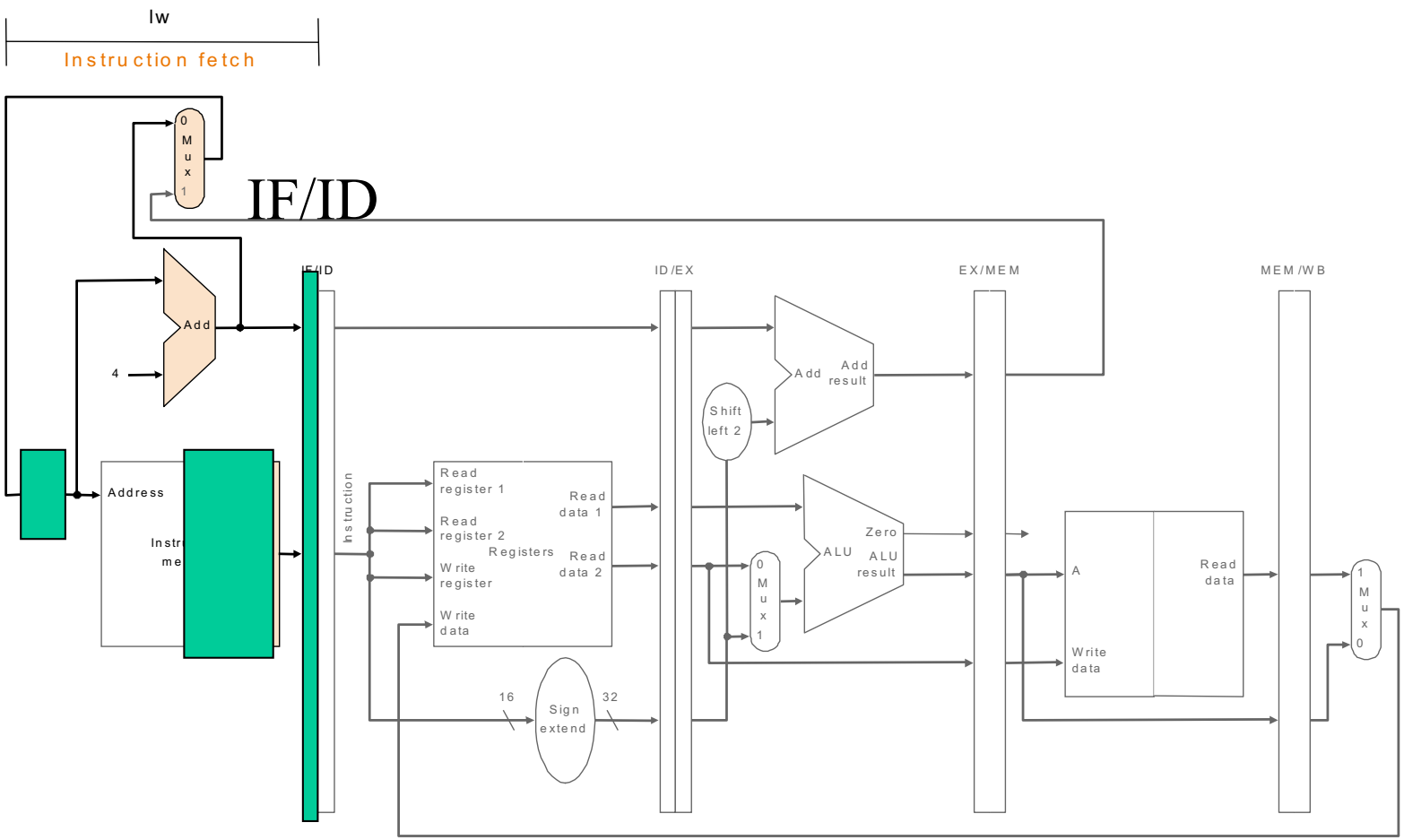
- What makes it easy
  - all instructions are the same length (32 bits)
    - can fetch in the 1st stage and decode in the 2nd stage
  - few instruction formats (three) with symmetry across formats
    - can begin reading register file in 2nd stage
  - memory operations can occur only in loads and stores
    - can use the execute stage to calculate memory addresses
  - each MIPS instruction writes at most one result (i.e., changes the machine state) and does so near the end of the pipeline (MEM and WB)

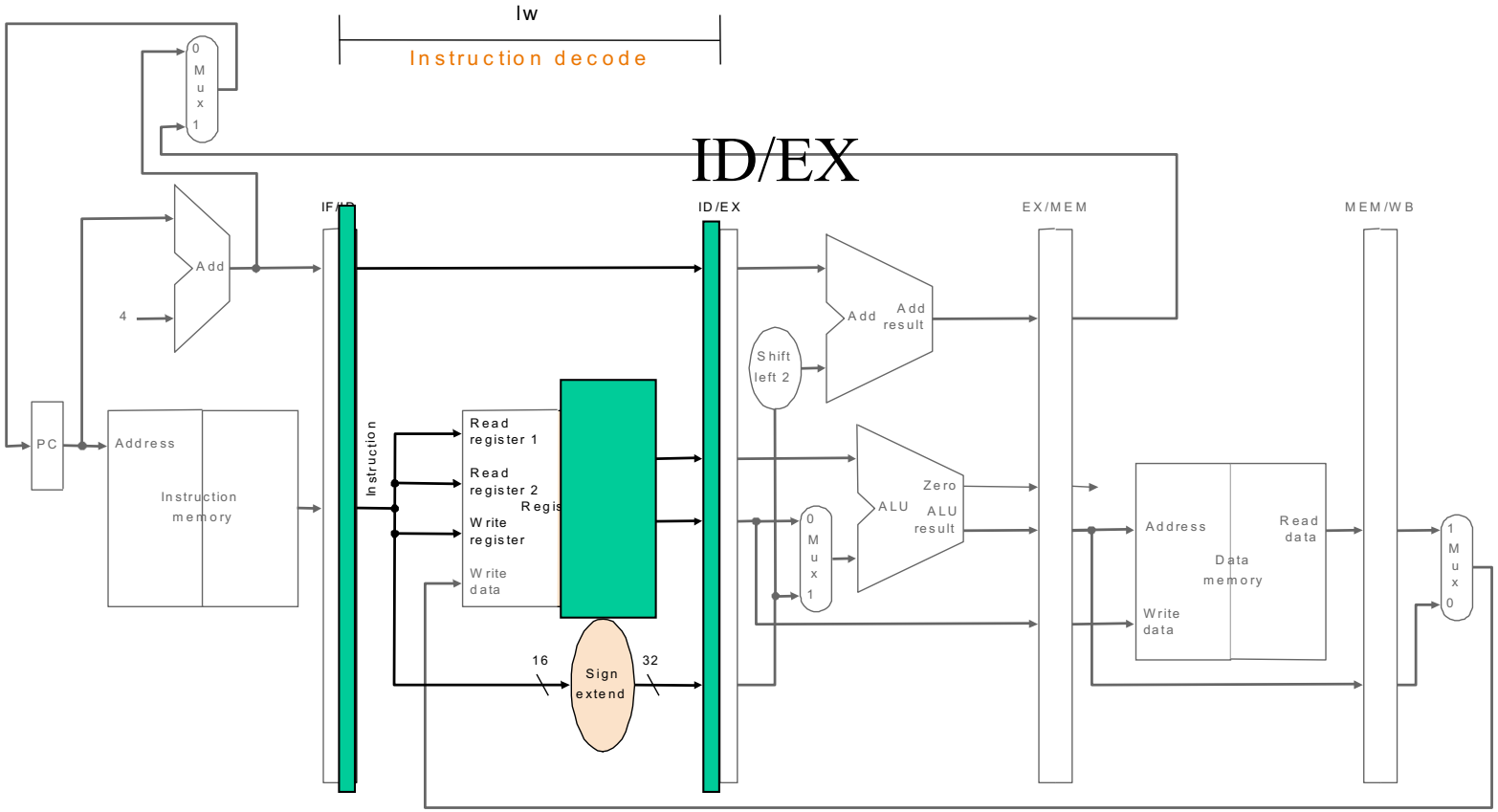
חלוקה לשלבים

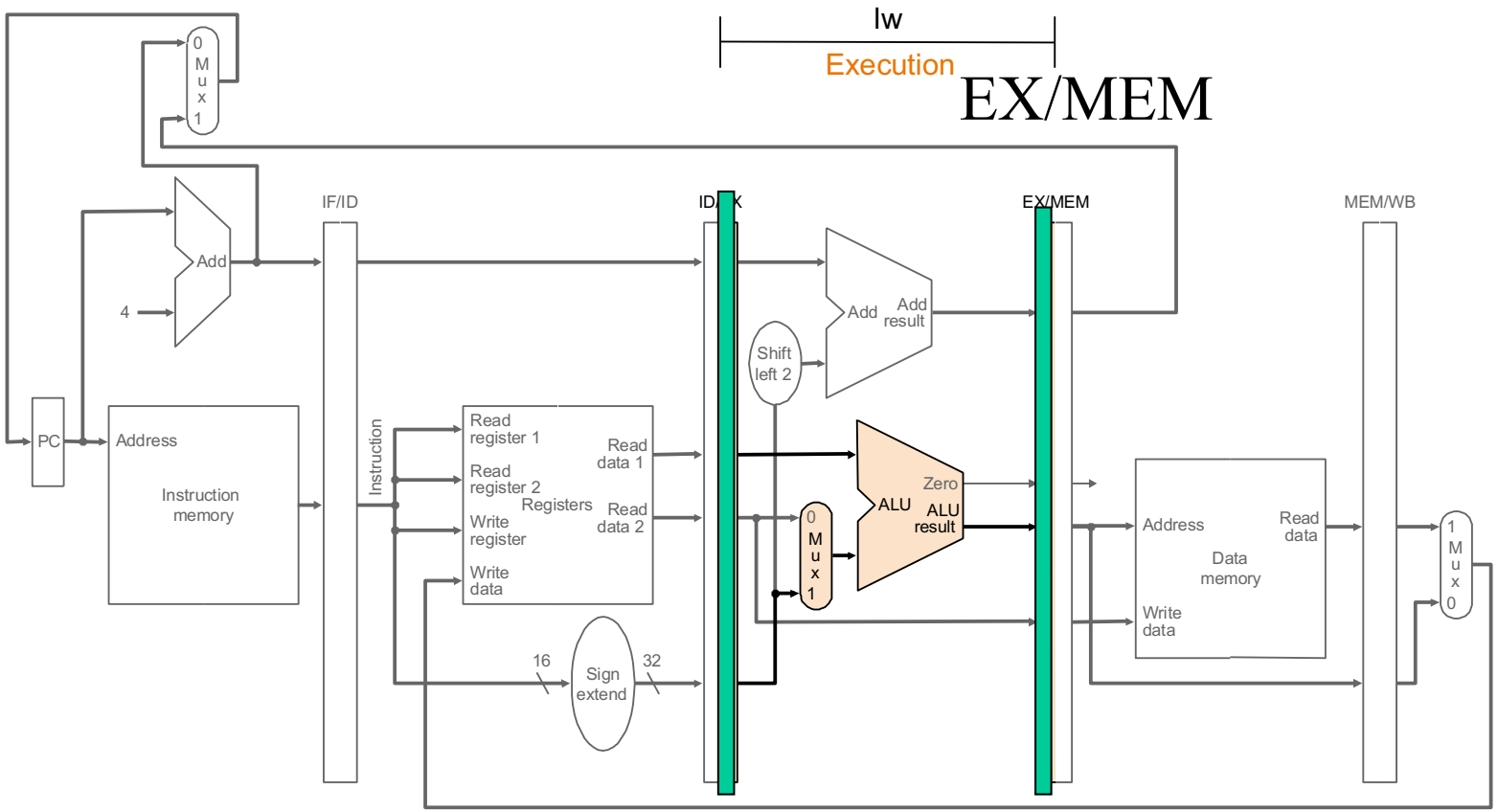


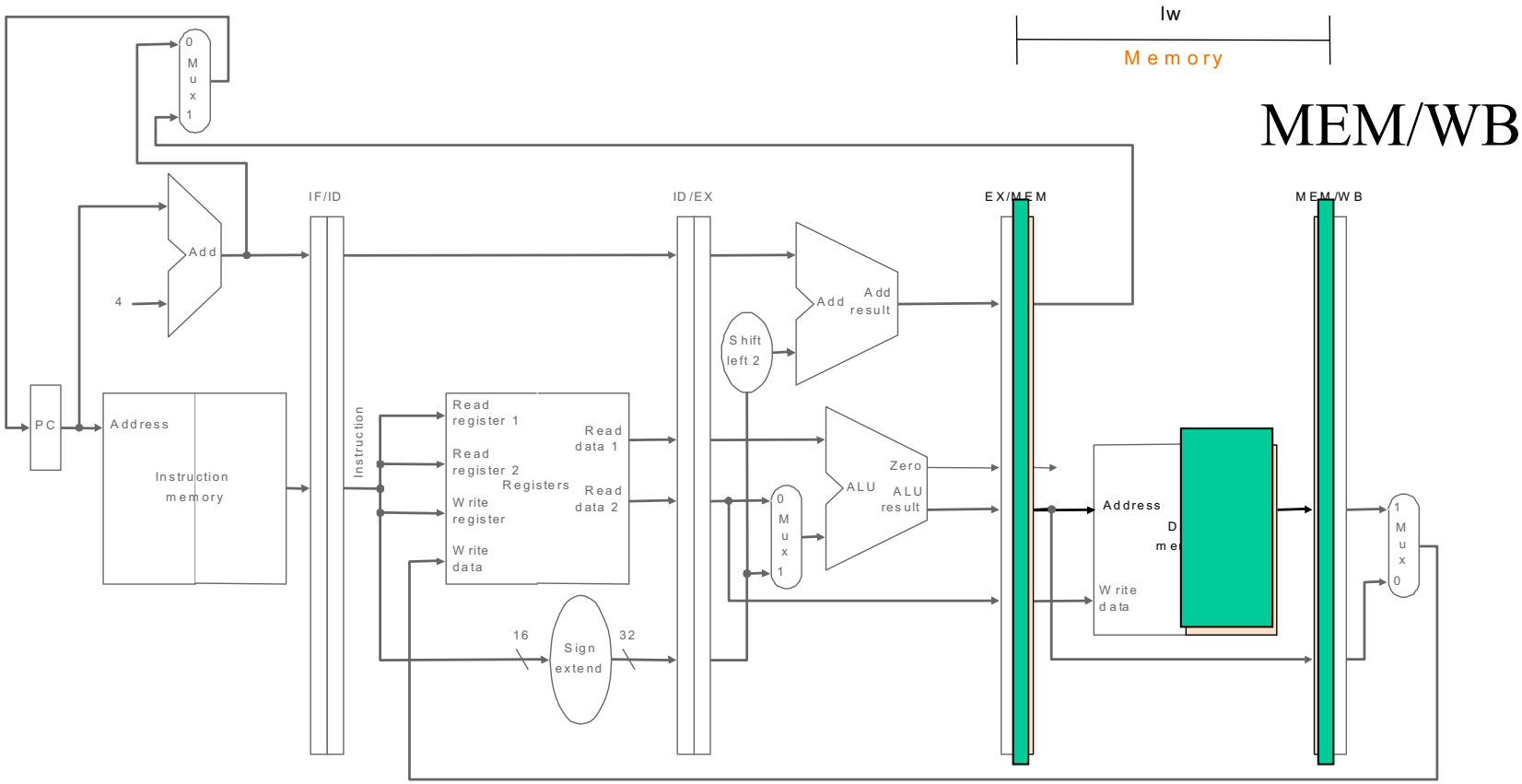
הוספת אוגרי הצנרת

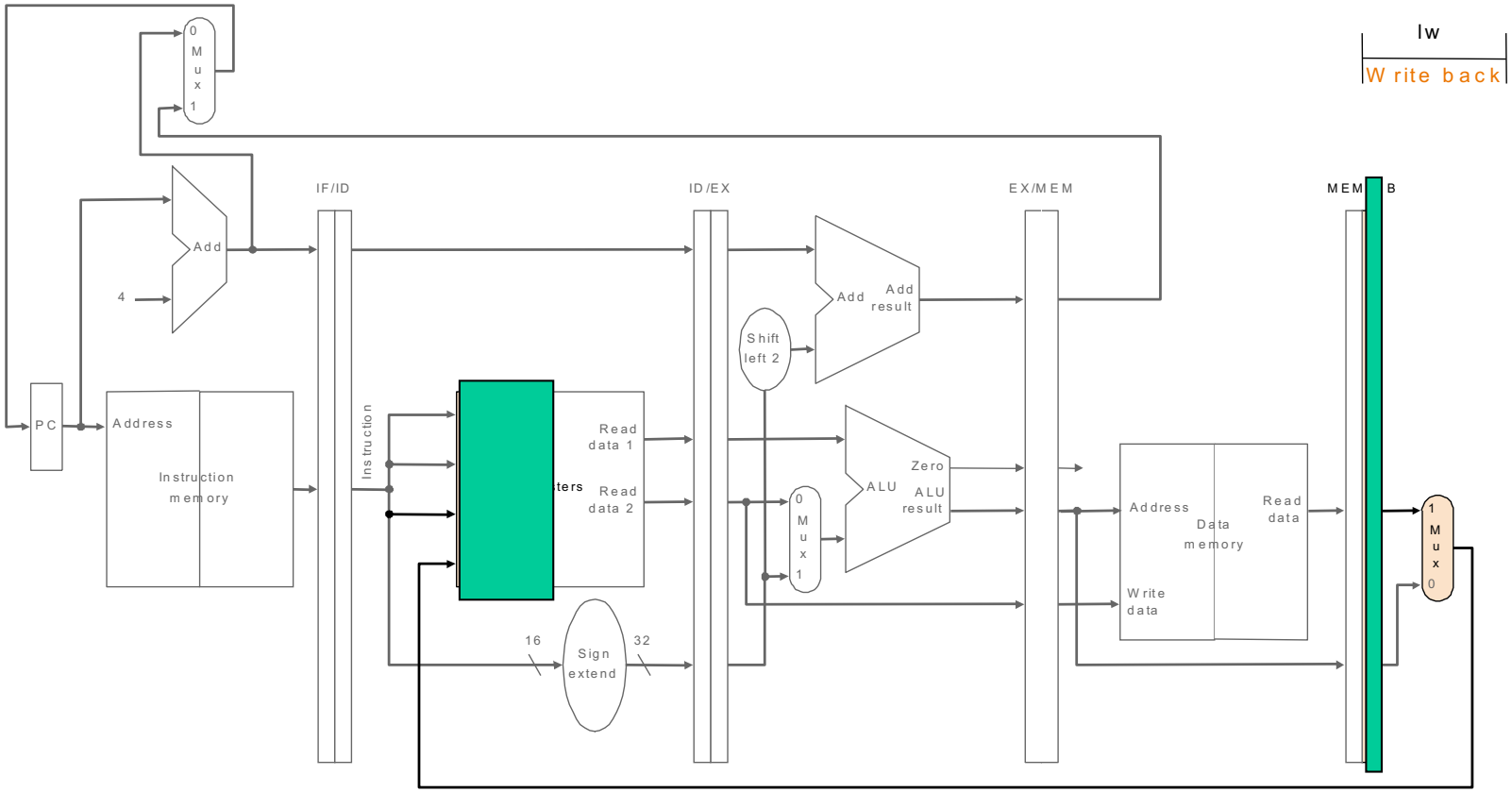



lw
Instruction fetch
IF/ID
0
M
u
x
1
Add
4
Address
Instr
me
IF/ID
Instruction
ID/EX
EX/MEM
MEM/WB
Add
Add
result
Shift
left 2
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
ALU
Zero
ALU
result
0
M
u
x
1
16
Sign
extend
32
A
Write
data
Read
data
1
M
u
x
0


lw
Instruction decode
ID/EX
0
M
u
x
1
Add
4
PC
Address
Instruction
memory
IF/ID
Instruction
Read
register 1
Read
register 2
Regis
Write
register
Write
data
16
Sign
extend
32
ID/EX
Add
Add
result
Shift
left 2
ALU
Zero
ALU
result
0
M
u
x
1
EX/MEM
Address
Read
data
Data
memory
Write
data
MEM/WB
1
M
u
x
0


lw
Execution
EX/MEM
0
M
u
x
1
Add
4
PC
Address
Instruction
memory
IF/ID
Instruction
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
16
Sign
extend
32
Shift
left 2
Add
Add
result
0
M
u
x
1
ALU
Zero
ALU
result
EX/MEM
Address
Data
memory
Read
data
Write
data
MEM/WB
1
M
u
x
0

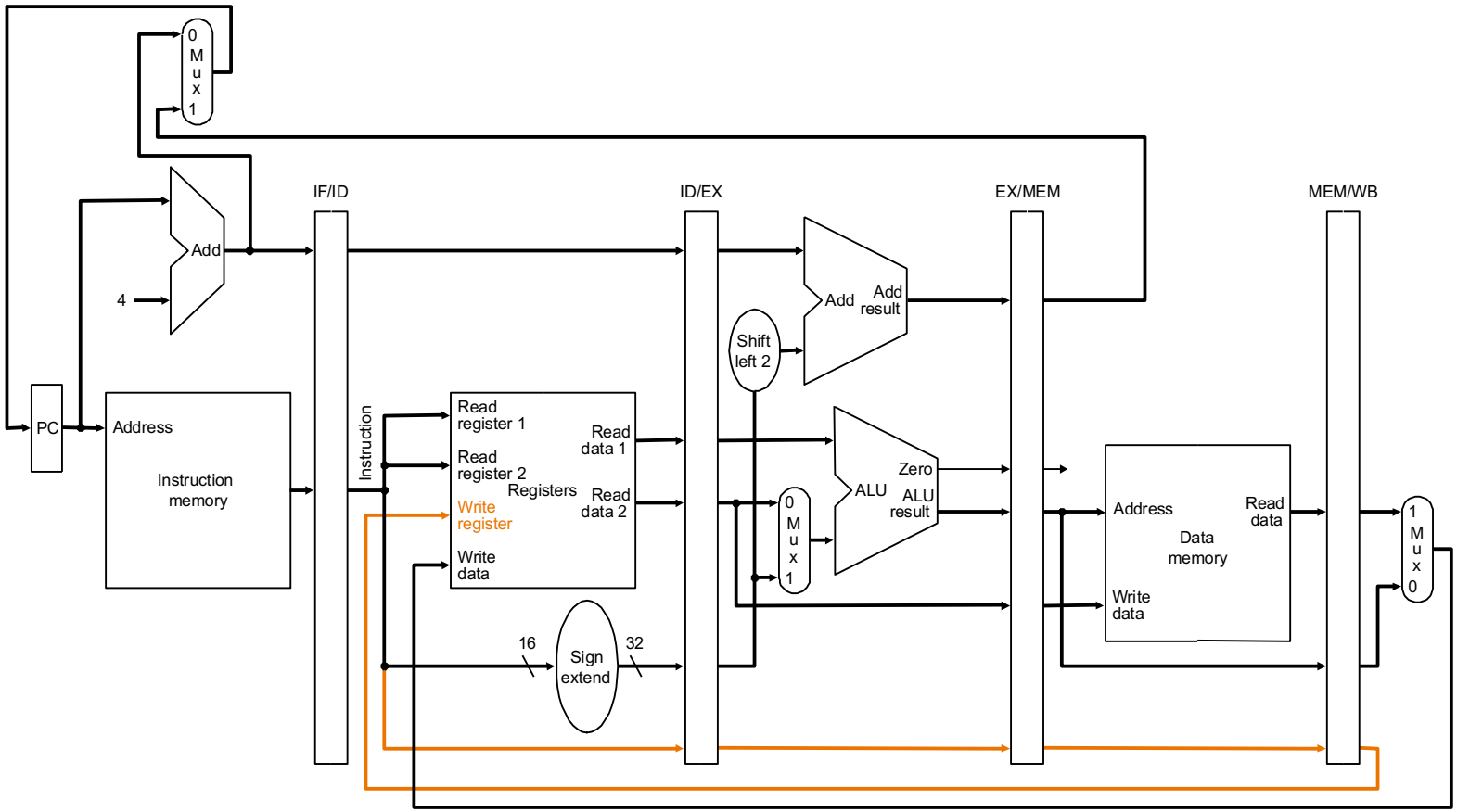

lw
Memory
MEM/WB
IF/ID
ID/EX
EX/MEM
MEM/WB
0
M
u
x
1
Add
4
PC
Address
Instruction
memory
Instruction
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
16
Sign
extend
32
Shift
left 2
Add
Add
result
0
M
u
x
1
ALU
Zero
ALU
result
Address
Write
data
1
M
u
x
0


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
0
M
u
x
1
Add
4
PC
Address
Instruction
memory
Instruction
Read
data 1
Read
data 2
16
Sign
extend
32
Shift
left 2
Add
Add
result
ALU
Zero
ALU
result
Address
Data
memory
Read
data
Write
data
1
M
u
x
0

הצצה על בקרת ה-Pipeline

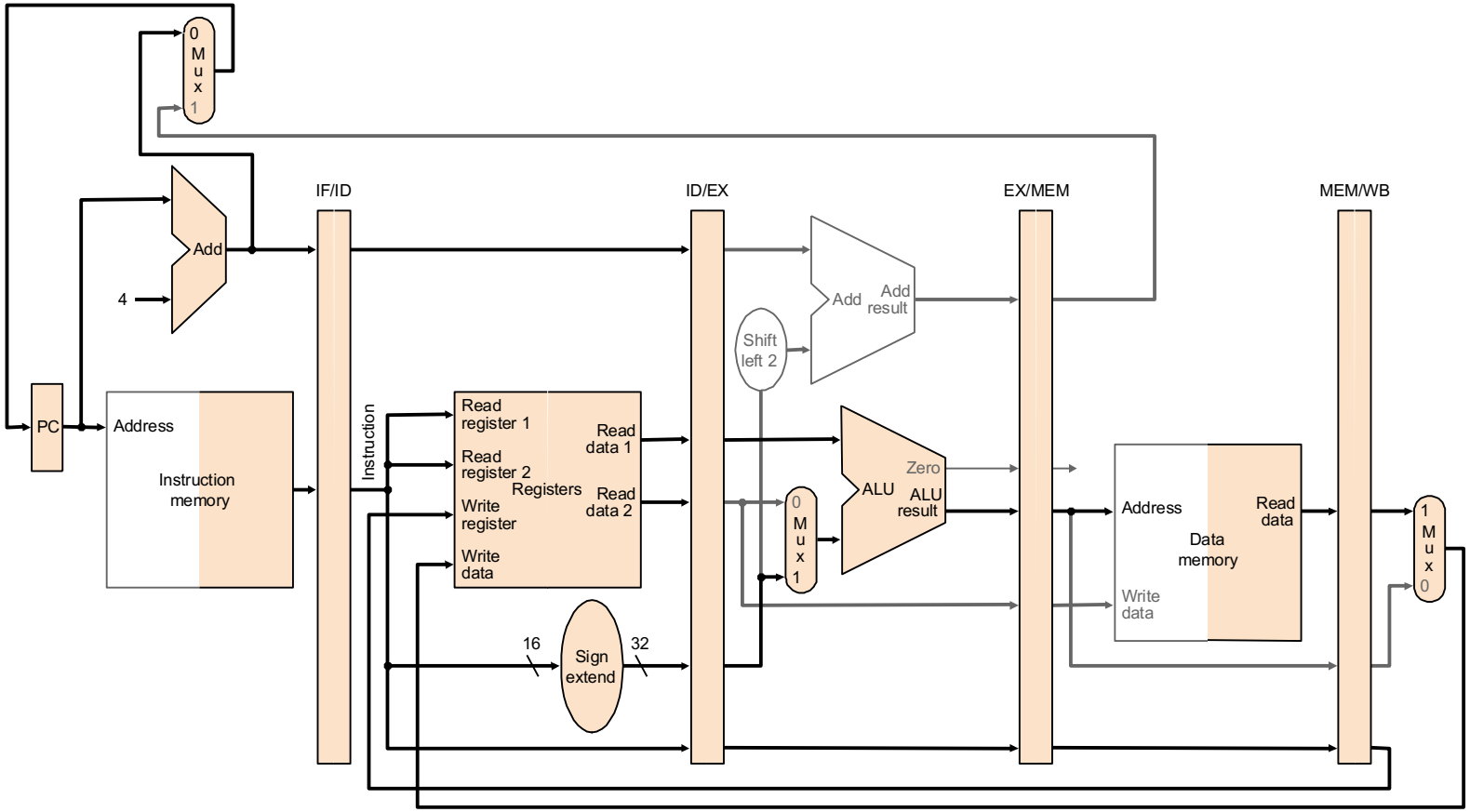
- בעיה: עד שמגיעים לשלב write-back של LW (שלושה מחזורים לאחר פענוח הפקודה), כבר נעלם מספר הרגיסטר (rt/rd) ולא נזכור לאן לכתוב את התוצאה...
- פתרון:
  - נשלח את מספר הרגיסטר (rt/rd) במורד ה-pipeline יחד עם הנתונים!
  - המילה לטעינה לרגיסטר ומספר הרגיסטר יגיעו יחד ל-Register File
- בעצם זו רק דוגמה — הבעיה והפתרון מתאימים לכל אותות הבקרה

Pipeline with correction

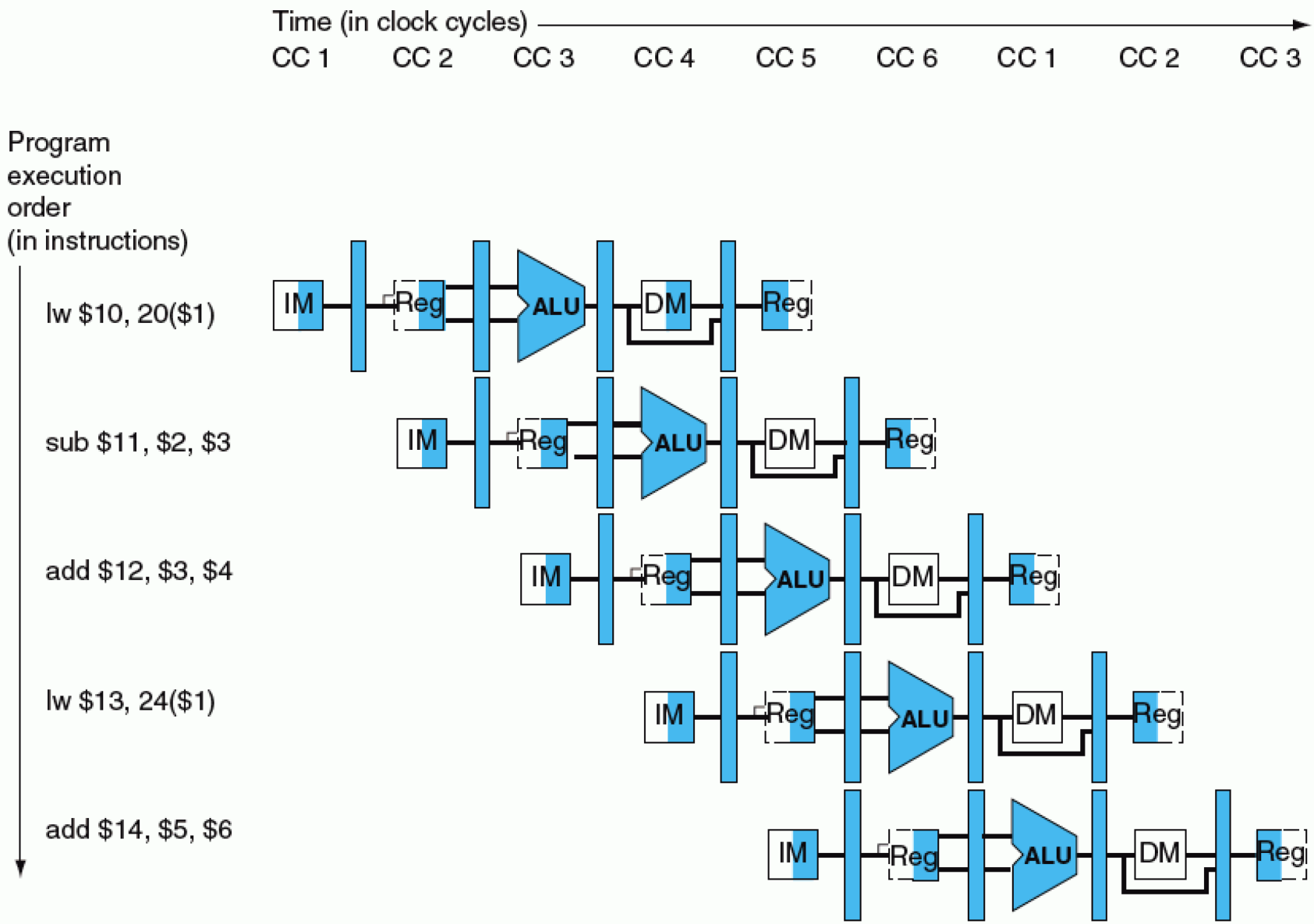


Keep the right Rd all the way!

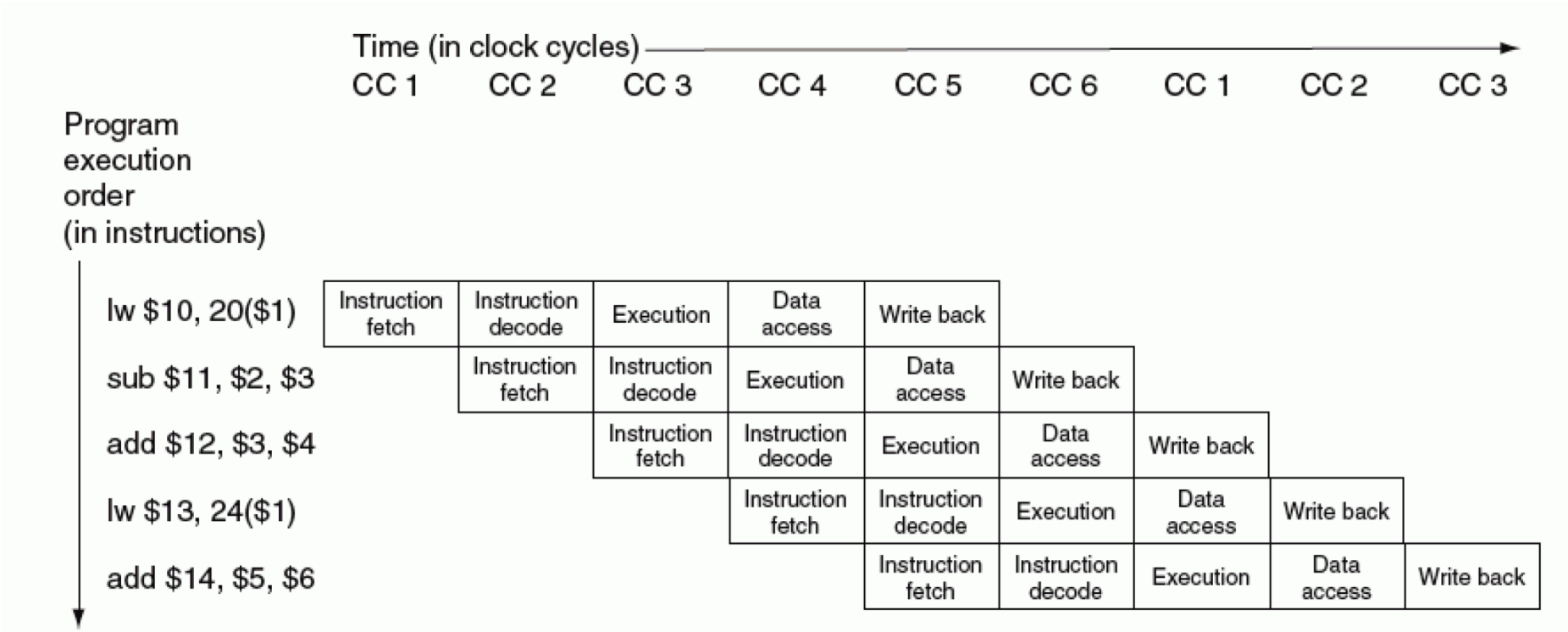
Updated CPU


0
M
u
x
1
Add
4
PC
Address
Instruction
memory
IF/ID
Instruction
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
16
Sign
extend
32
ID/EX
Add
Add
result
Shift
left 2
0
M
u
x
1
ALU
Zero
ALU
result
EX/MEM
Address
Data
memory
Read
data
Write
data
MEM/WB
1
M
u
x
0

התקדמות הפקודות בזמן בתוך הצינרת לפי המבני חומרה



התקדמות הפקודה בזמן לפי השלבים (timeline)



חתכים לפי השינוי במשאב\רכיב בזמן

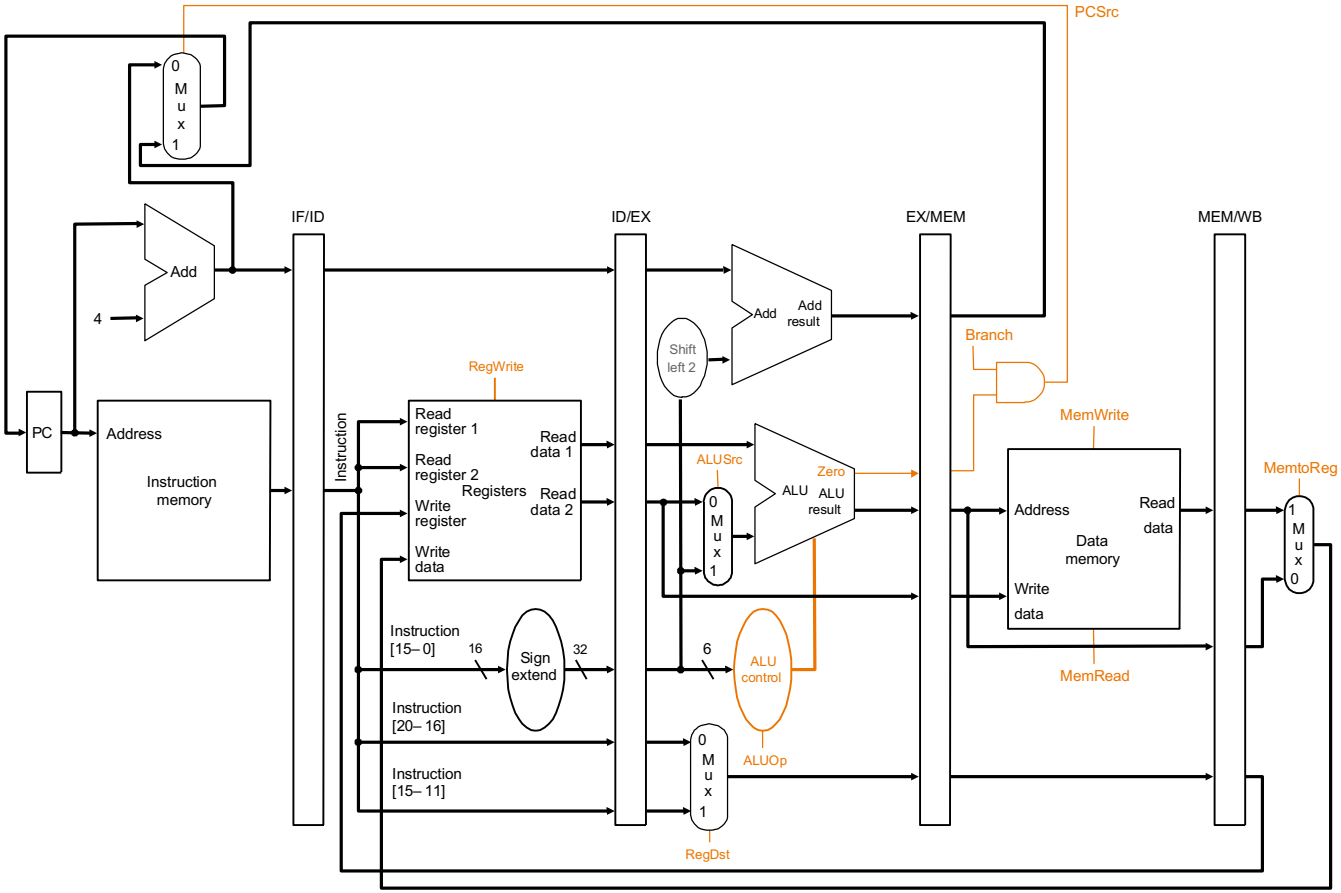
Sample Program

```
I1: ADD R4,R3,R2
I2: AND R6,R5,R4
I3: SUB R1,R9,R8
I4: XOR R3,R2,R1
I5: OR  R7,R6,R5
```

Time:	t1	t2	t3	t4	t5	t6	t7	t8
Stage								
IF:	I1	I2	I3	I4	I5	I6	I7	I8
ID:		I1	I2	I3	I4	I5	I6	I7
EX:			I1	I2	I3	I4	I5	I6
MEM:				I1	I2	I3	I4	I5
WB:					I1	I2	I3	I4

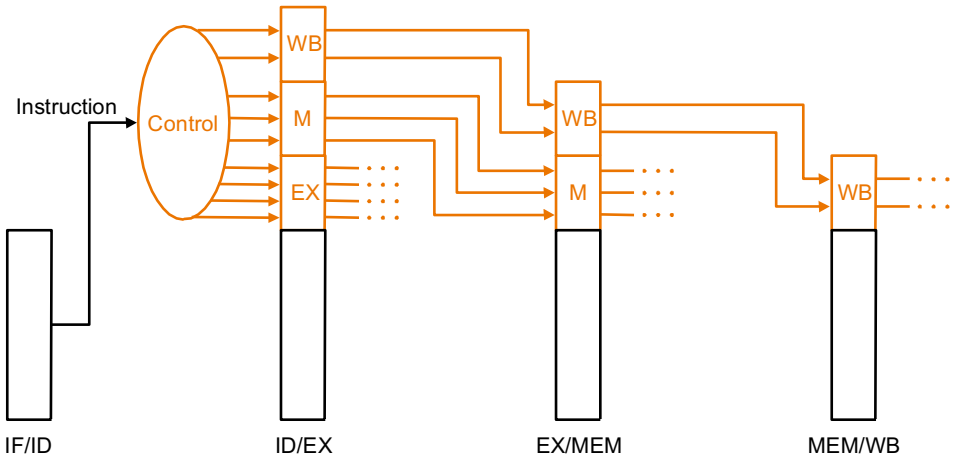
Pipeline is "full"

Pipeline Control

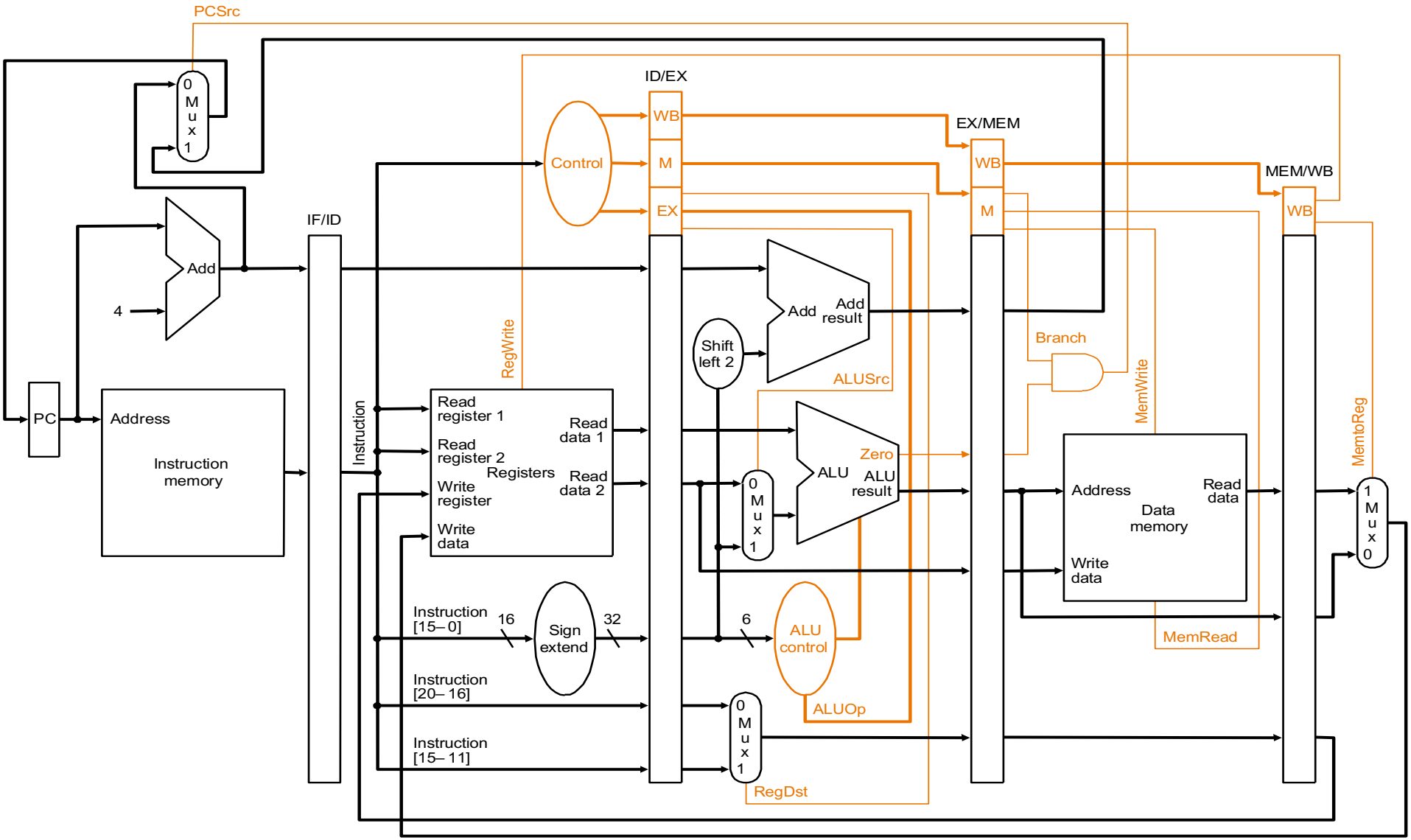


קווי הבקרה

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X



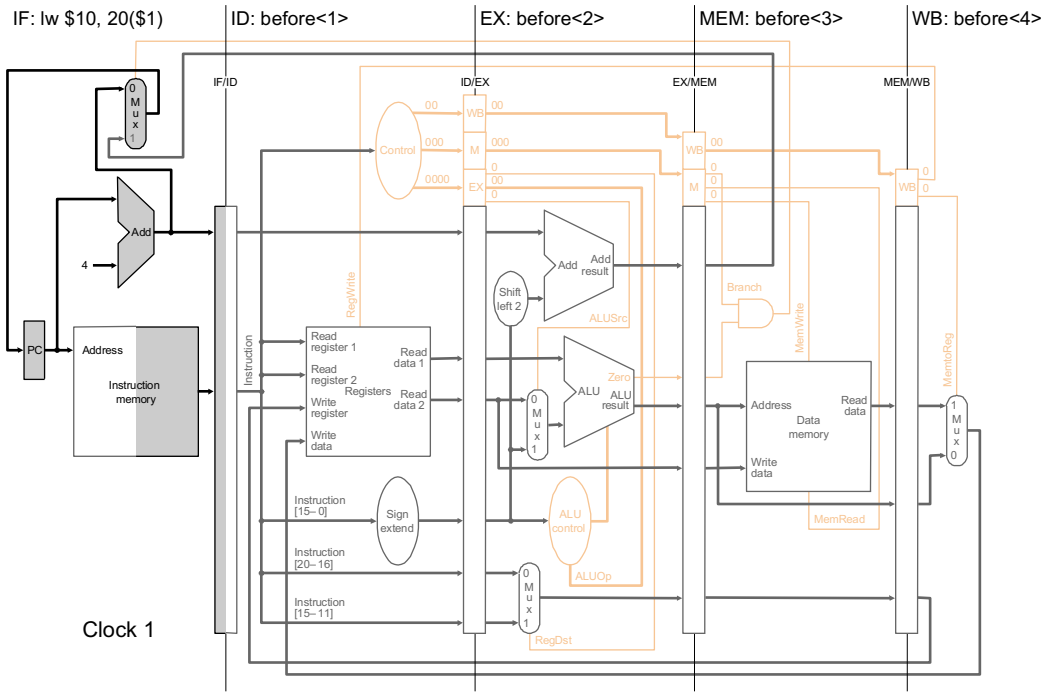
Datapath with Control

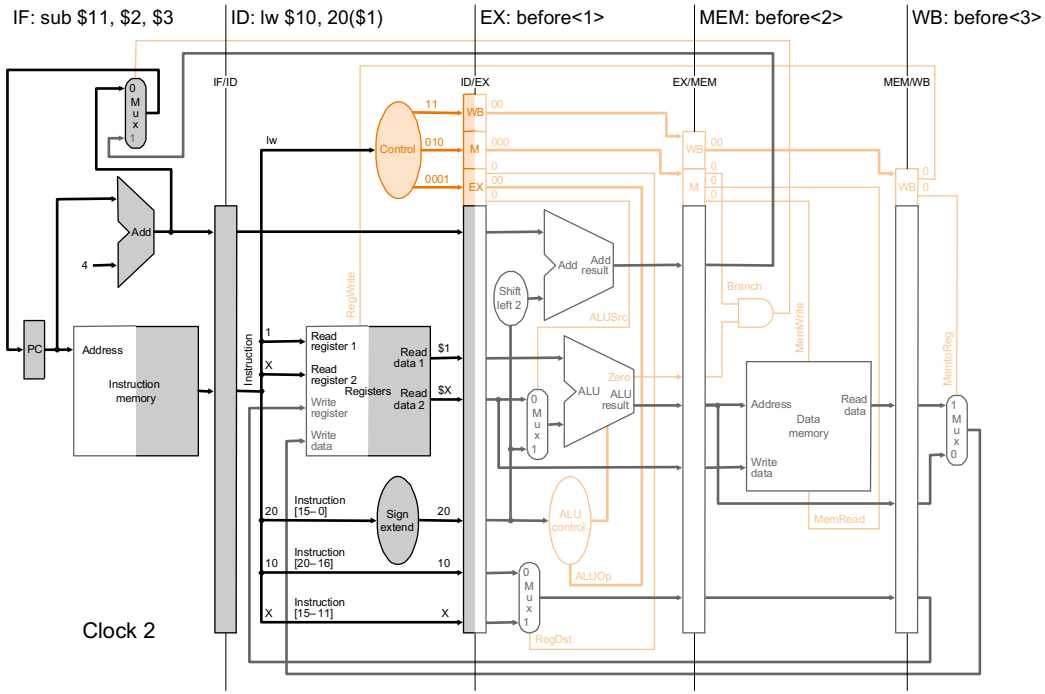


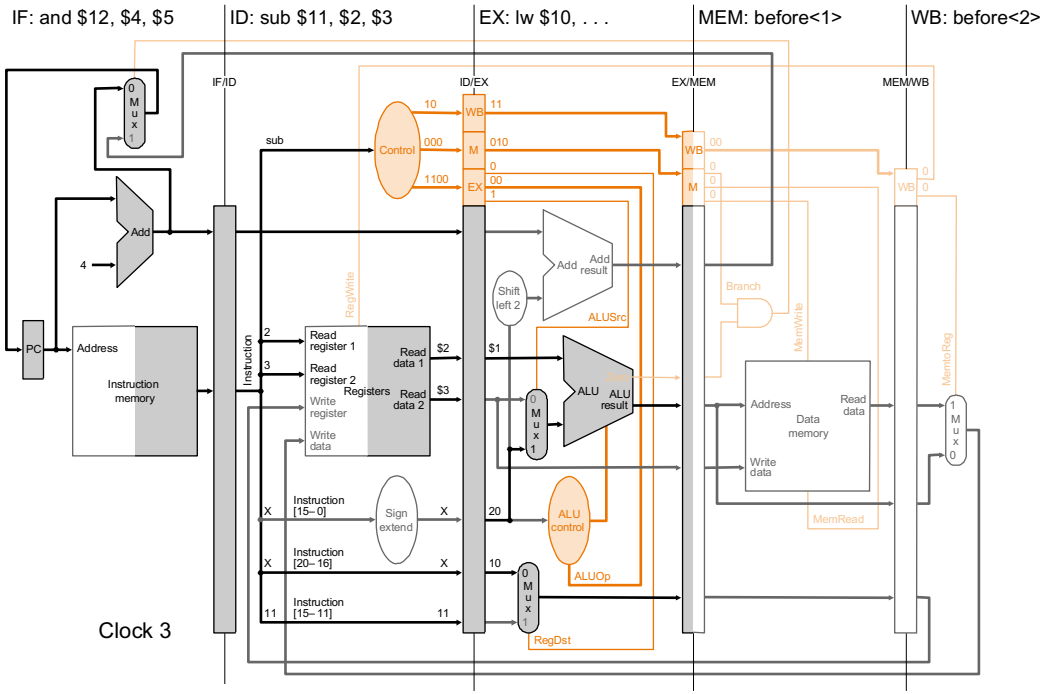
דוגמא

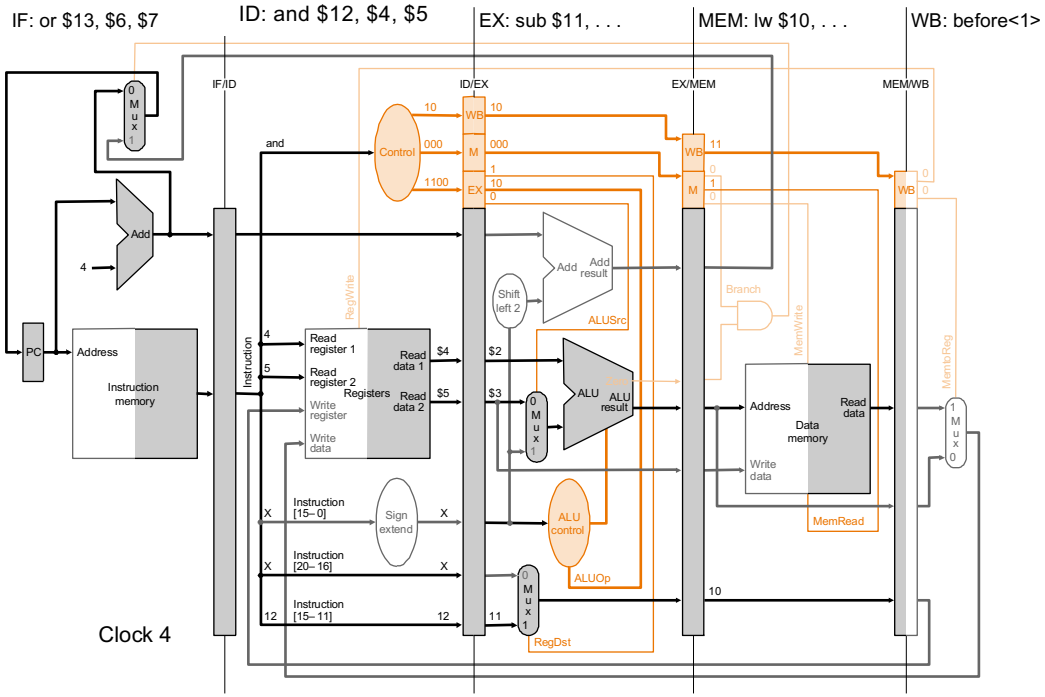
A demonstration of a sequence of instructions:

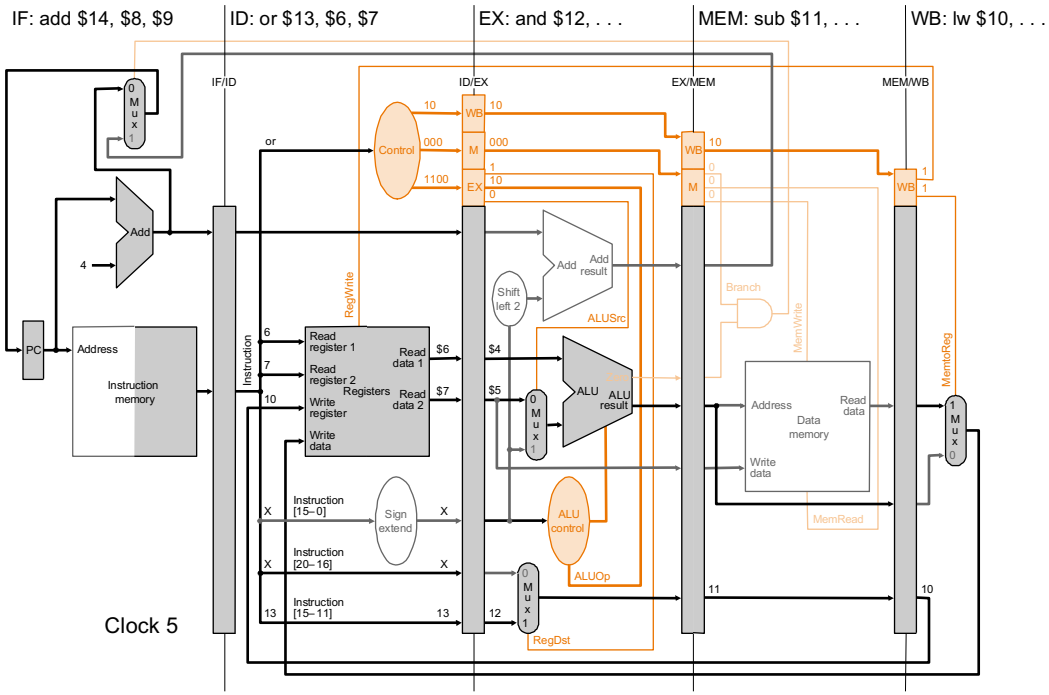
```
lw      $10,20($1)
sub     $11,$2,$3
and     $12,$4,$5
or      $13,$6,$7
add     $14,$8,$9
```

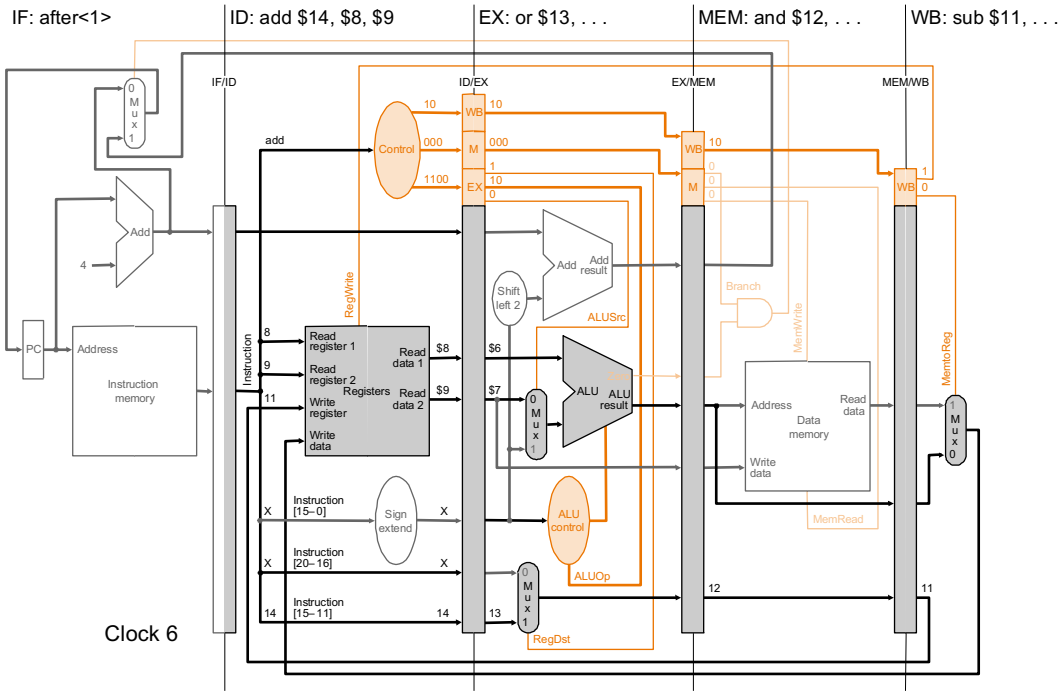

IF: lw $10, 20($1)
ID: before<1>
EX: before<2>
MEM: before<3>
WB: before<4>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 1

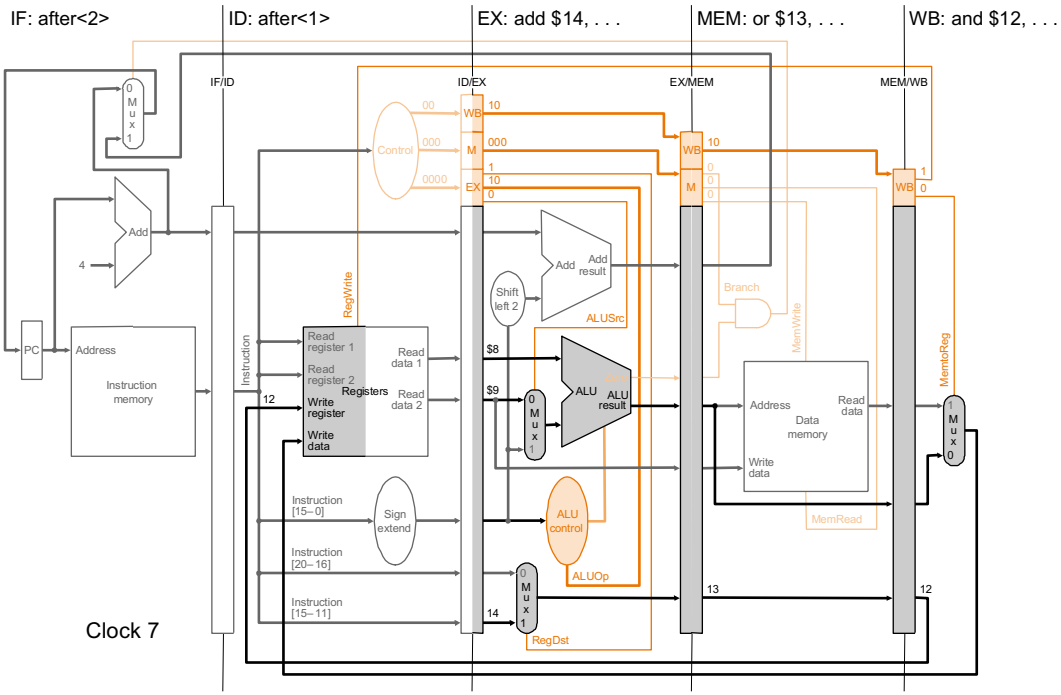

IF: sub $11, $2, $3
ID: lw $10, 20($1)
EX: before<1>
MEM: before<2>
WB: before<3>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 2

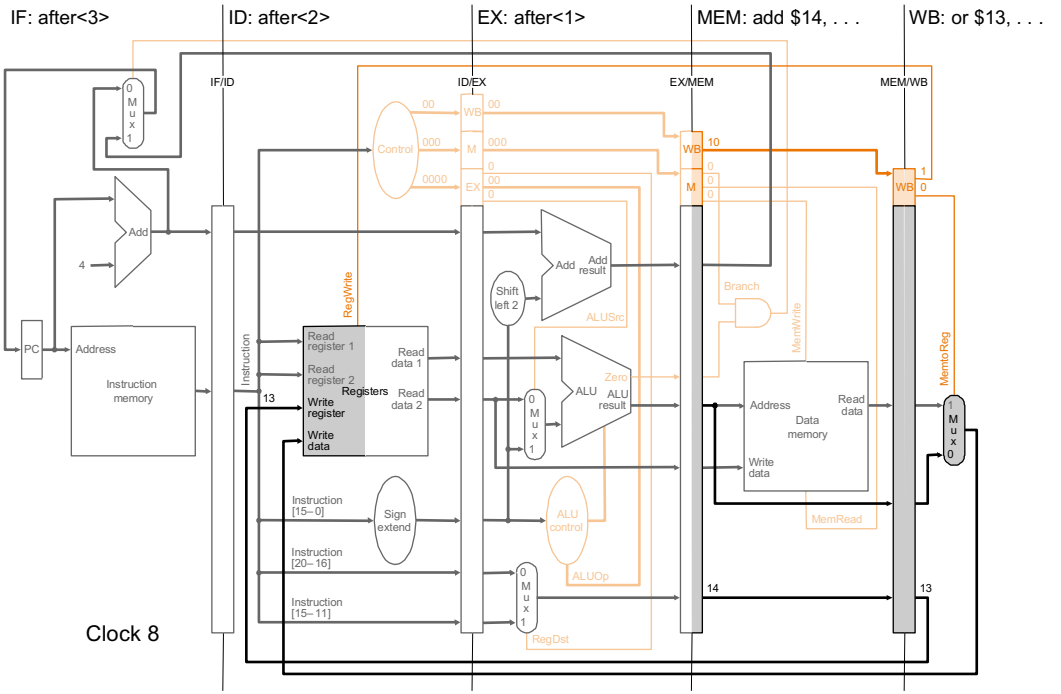

IF: and $12, $4, $5
ID: sub $11, $2, $3
EX: lw $10, . . .
MEM: before<1>
WB: before<2>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 3


IF: or $13, $6, $7
ID: and $12, $4, $5
EX: sub $11, . . .
MEM: lw $10, . . .
WB: before<1>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 4


IF: add $14, $8, $9
ID: or $13, $6, $7
EX: and $12, . . .
MEM: sub $11, . . .
WB: lw $10, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
Add
Add result
Shift left 2
ALUSrc
Branch
MemWrite
MemtoReg
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
ALU
ALU result
Address
Data memory
Read data
Write data
MemRead
Sign extend
ALU control
ALUOp
RegDst
RegWrite
Clock 5


IF: after<1>
ID: add $14, $8, $9
EX: or $13, . . .
MEM: and $12, . . .
WB: sub $11, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
Add
Add result
Shift left 2
ALUSrc
Branch
MemWrite
MemtoReg
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
ALU
ALU result
Address
Data memory
Read data
Write data
MemRead
Sign extend
ALU control
ALUOp
RegDst
RegWrite
Clock 6


IF: after<2>
ID: after<1>
EX: add $14, . . .
MEM: or $13, . . .
WB: and $12, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 7


IF: after<3>
ID: after<2>
EX: after<1>
MEM: add $14, . . .
WB: or $13, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 8

Can Pipelining Get Us Into Trouble?

- Yes: Pipeline Hazards
  - structural hazards: attempt to use the same resource by two different instructions at the same time
  - data hazards: attempt to use data before it is ready
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline
  - control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - branch instructions

- Can always resolve hazards by waiting
  - pipeline control must detect the hazard
  - and take action to resolve hazards

Dealing With Structural Hazards

- Structural hazards result from the CPU data path not having resources to service all the required overlapping resources.
- Suppose a processor can only read and write from the registers in one clock cycle. This would cause a problem during the ID and WB stages.
- Assume that there are not separate instruction and data caches, and only one memory access can occur during one clock cycle. A hazard would be caused during the IF and MEM cycles.

דוגמא לסיכון נתונים

```
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)
```

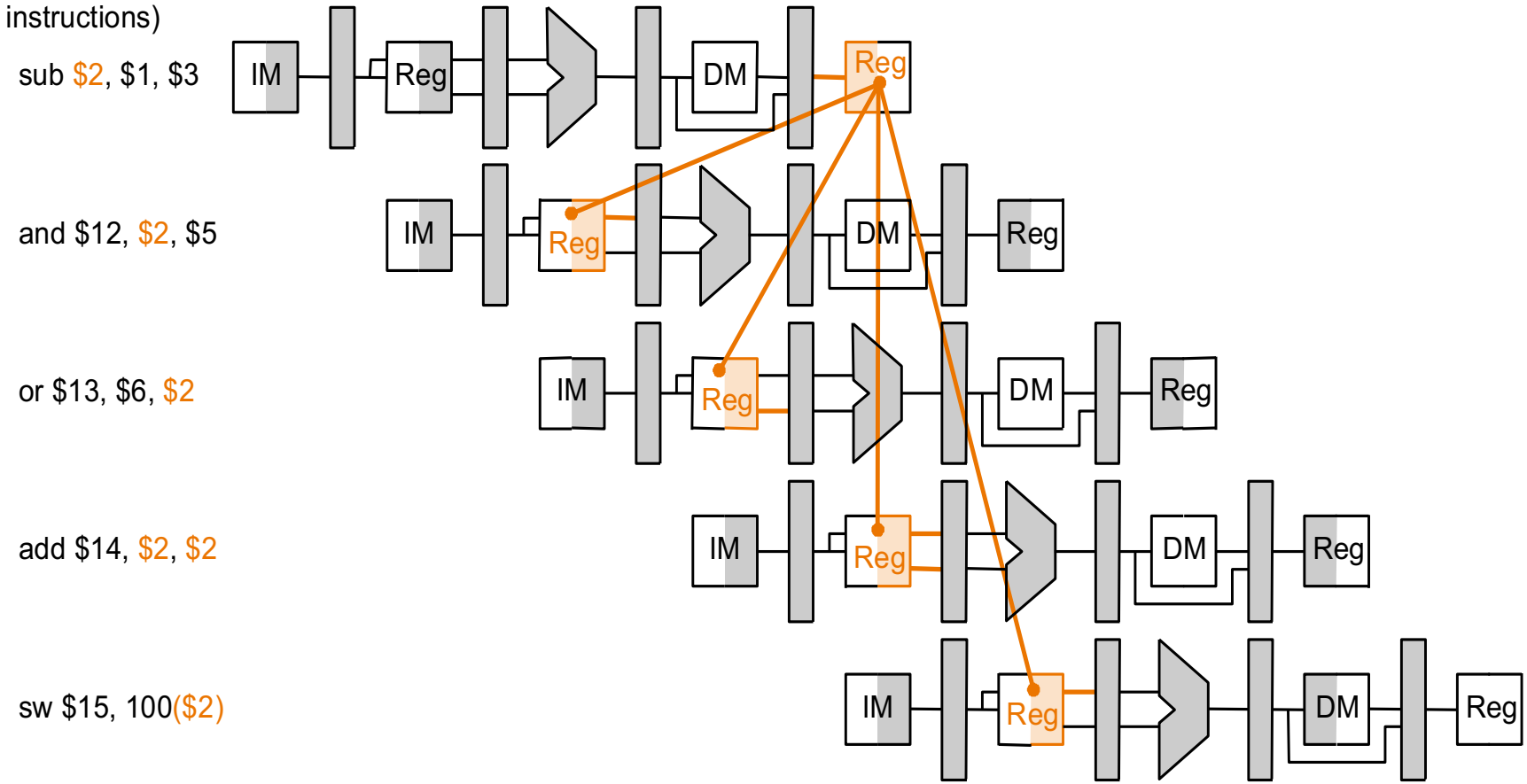
- *Register $2 is updated only at the WB phase, i.e., the 5th clock cycle (actually at the end of the 5th clock cycle).*
- *However, we try to use it at the 3rd clock cycle when we read $2 at the decode phase of the and instruction*

Graphic representation of data hazards:

Time (in clock cycles)

Value of register $2:	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20

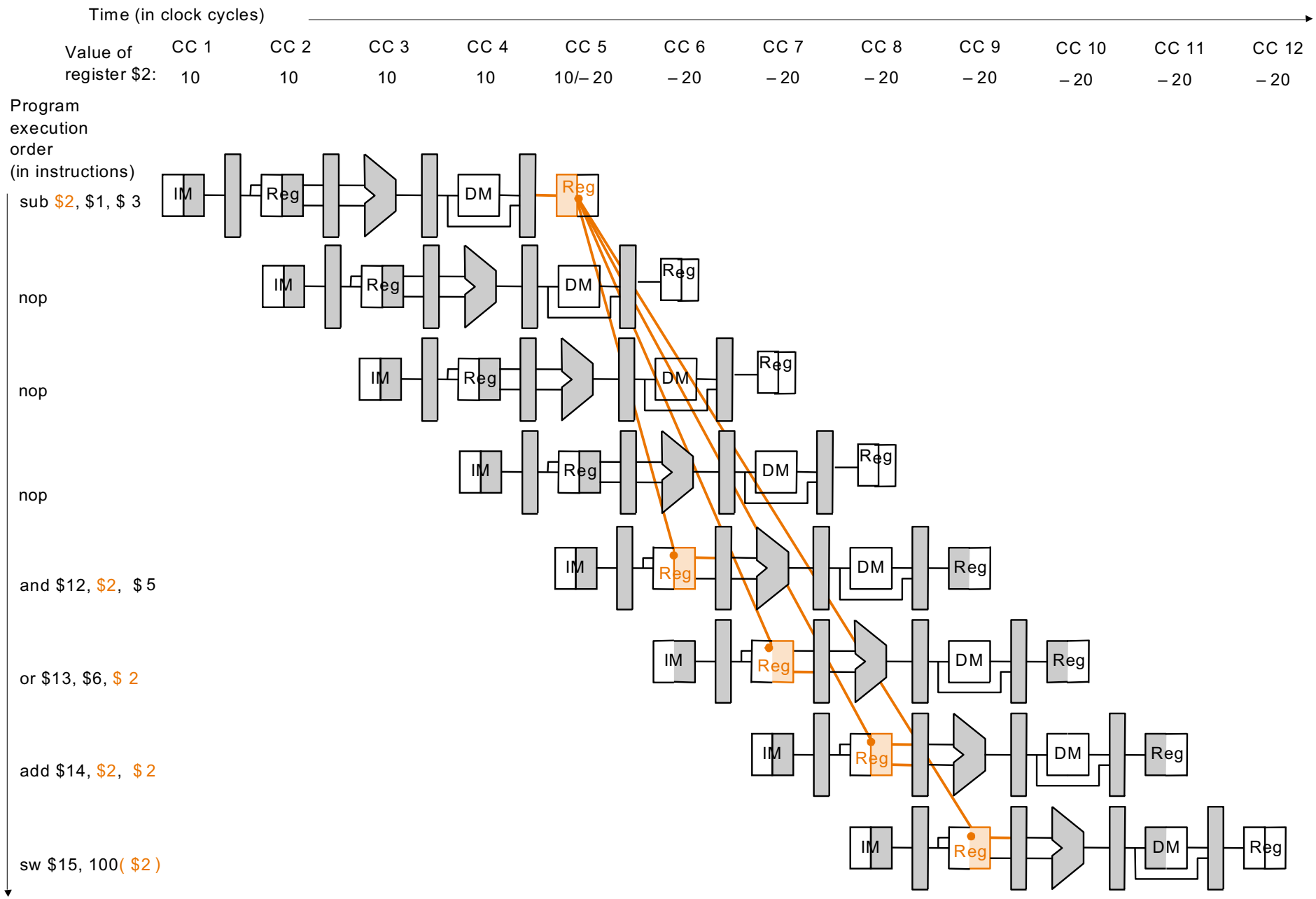
Program execution order (in instructions)



Solving data hazards by adding nops

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

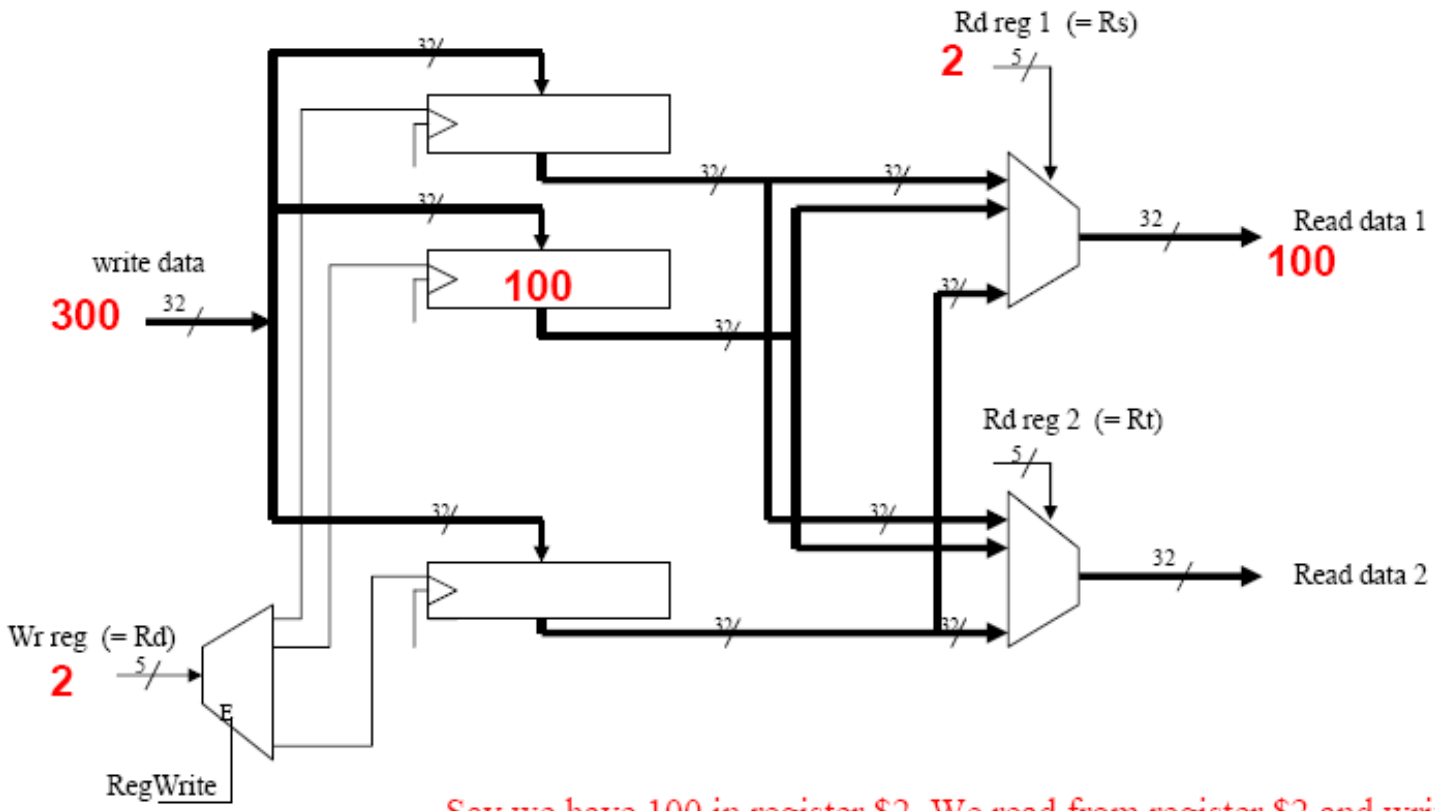
Solving data hazards by adding nops



וקריאה לאותו אוגר באותה פעימת שעון

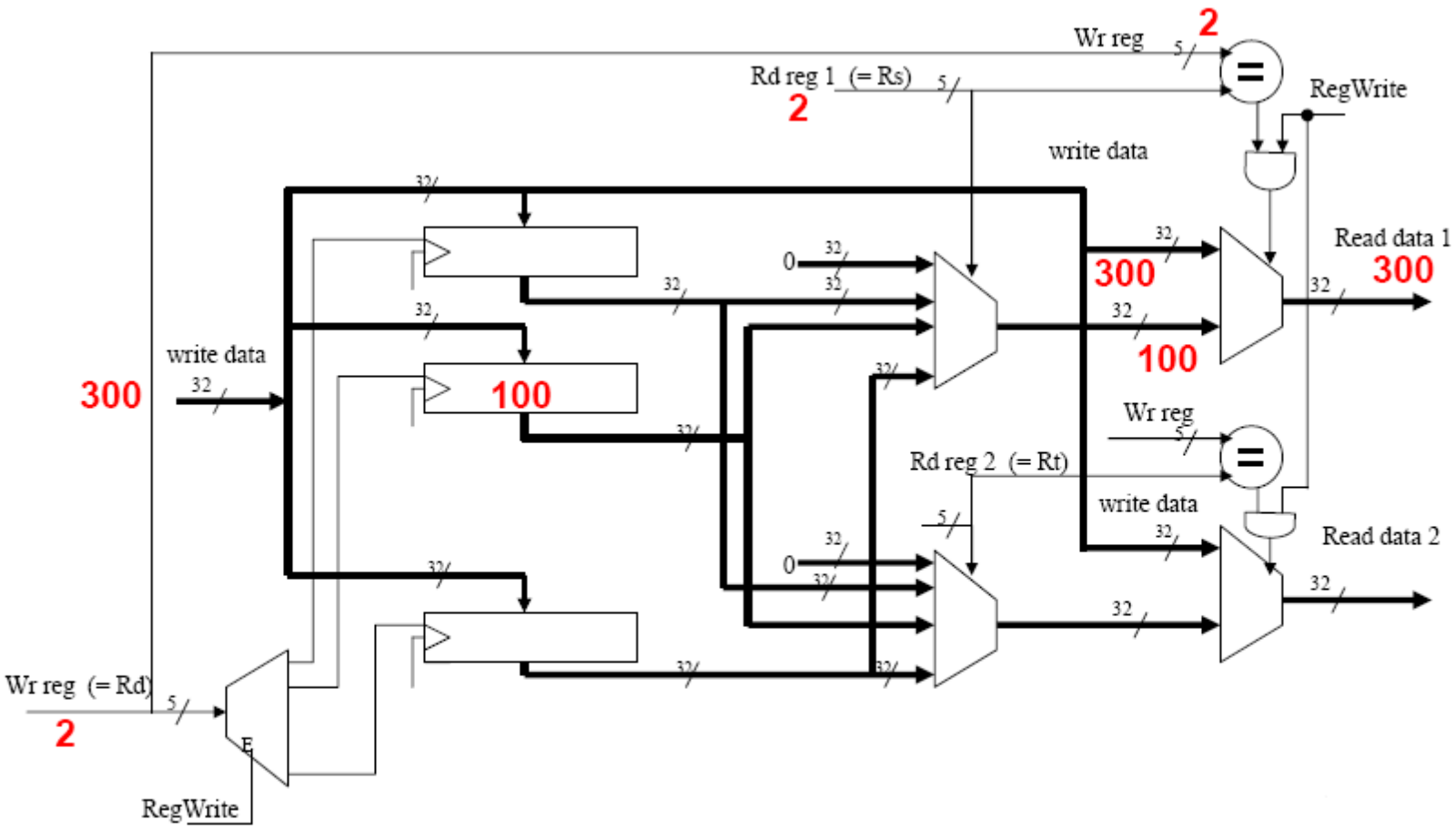
- כתיבות למיקבץ האוגרים מתבצעות בחלקה הראשון של פעימת השעון והקריאות ממיקבץ האוגרים בחלקה השני של פעימת השעון. כך שלמעשה ניתן לחסוך nop אחד.
- השינוי החומרתי הינו די פשוט.

Data Hazards- Hardware Bottleneck



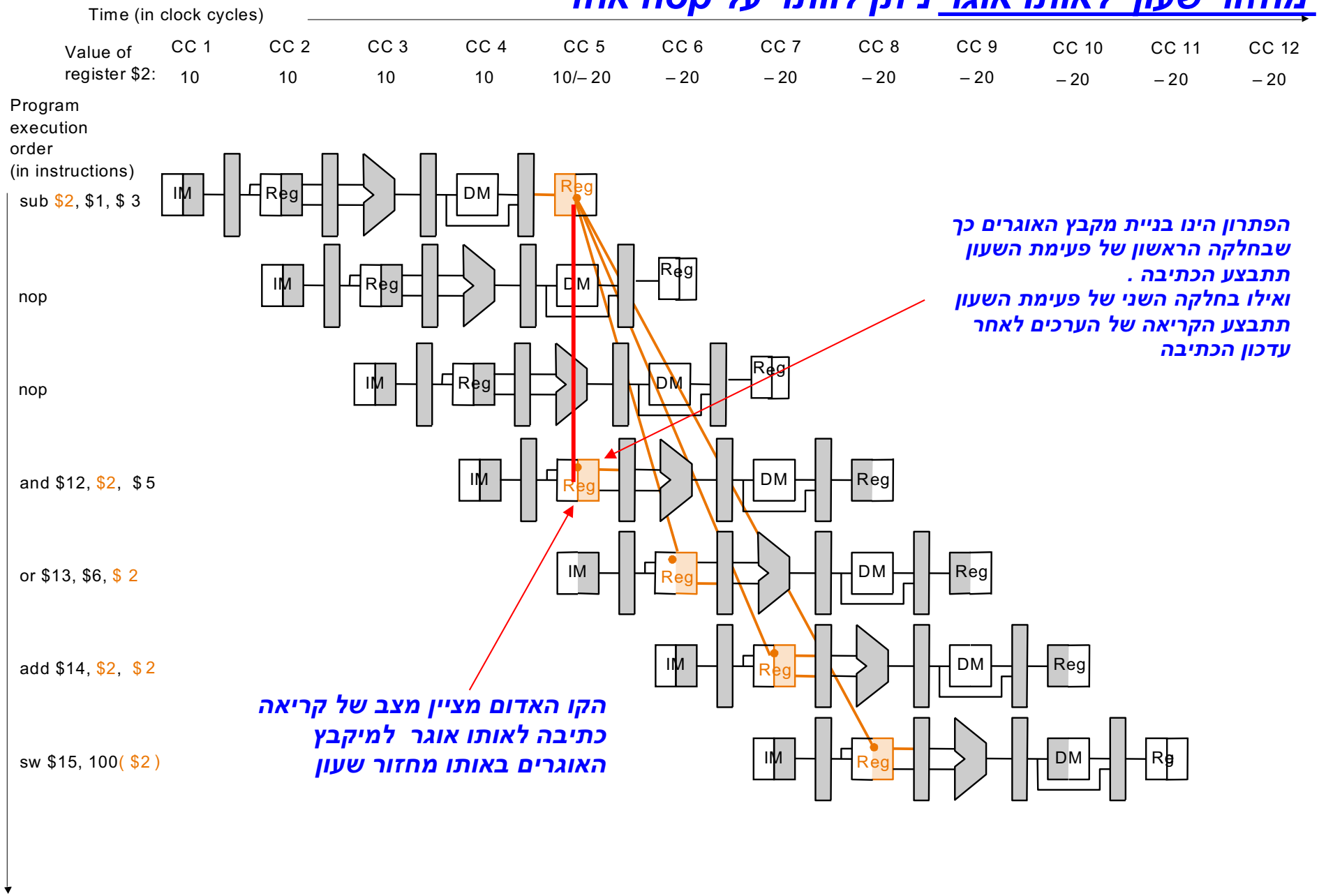
Say we have 100 in register $2. We read from register $2 and write 300 into register2. We still get 100 at the output!!! (until the next CK rising edge).

Data Hazards- Hardware Adaptation



When we read and write from/to the same register (e.g., $2) simultaneously, we bypass the register, which is updated only at the next rising edge of the CK).

בבניה מתאימה על מיקבץ האוגרים ניתן לפתור מצב בו יש קריאה וכתיבה באותו מחזור שעון לאותו אוגר ניתן לוותר על nop אחד



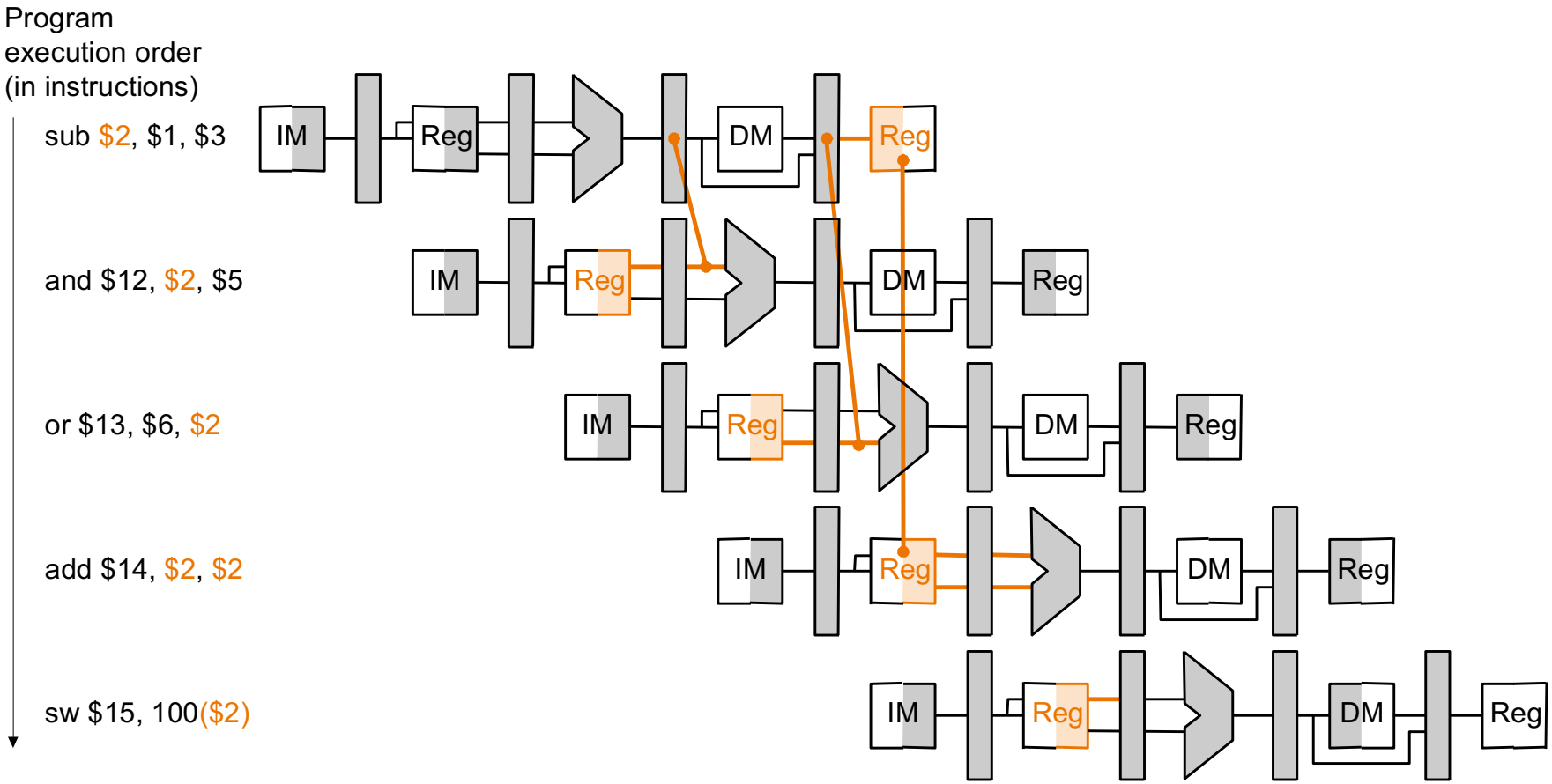
Solving data hazards by adding nops

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

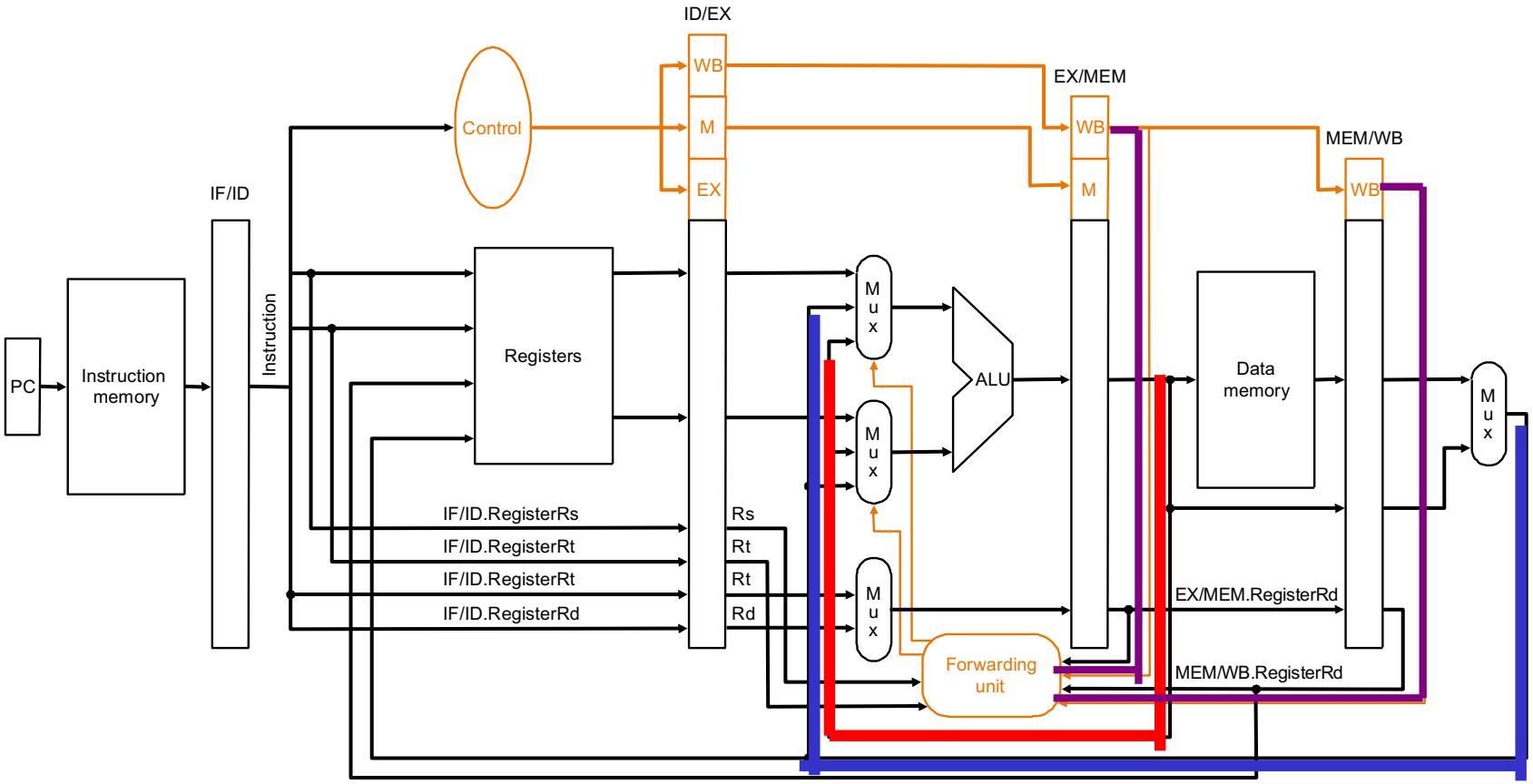
Forwarding – גניבת הערכים

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
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X



Forwarding (done at the execute phase)



If ID/EX.Rs=EX/MEM.Rd, i.e., the Rd of the previous instruction equals the Rs of the current instruction (which is in the "decode" phase), then we use the "ALUout" of the previous instruction instead of the output of the GPR.

If ID/EX.Rs=MEM/WB.Rd, i.e., the Rd of the previous instruction equals the Rs of the current instruction (which is in the "decode" phase), then we use the "ALUout" of the previous instruction instead of the output of the GPR.

Similarly, compare also ID/EX.**Rt** to EX/MEM.Rd and to MEM/WB.Rd

Data Forwarding Control Conditions

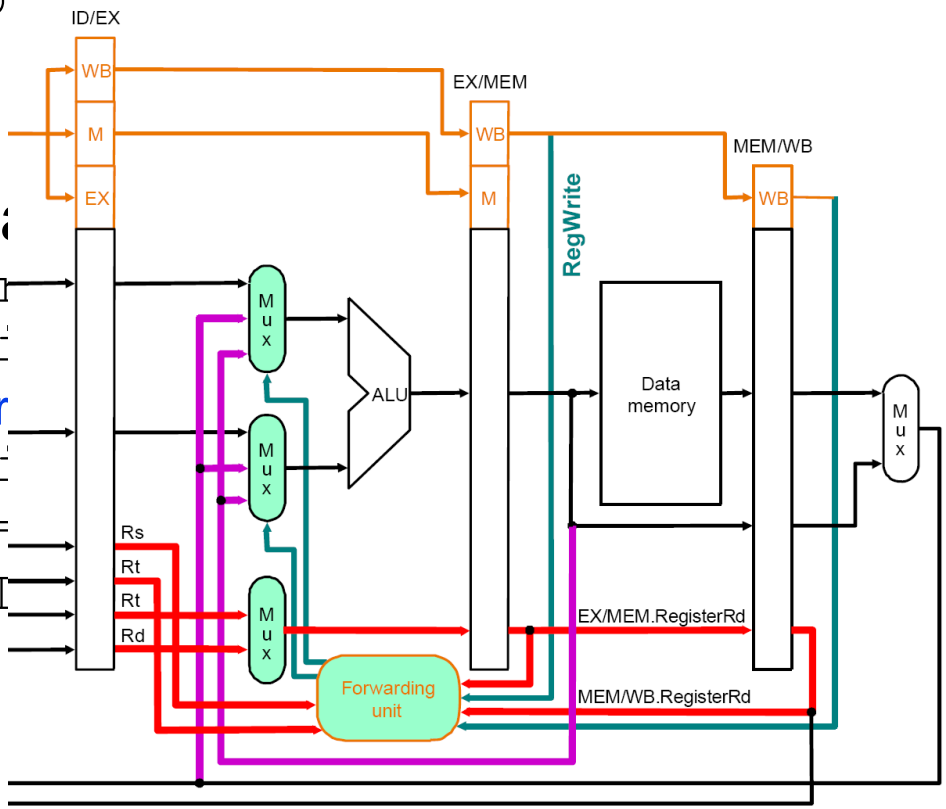
1. EX/MEM hazard:

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 10
The same for ID
```

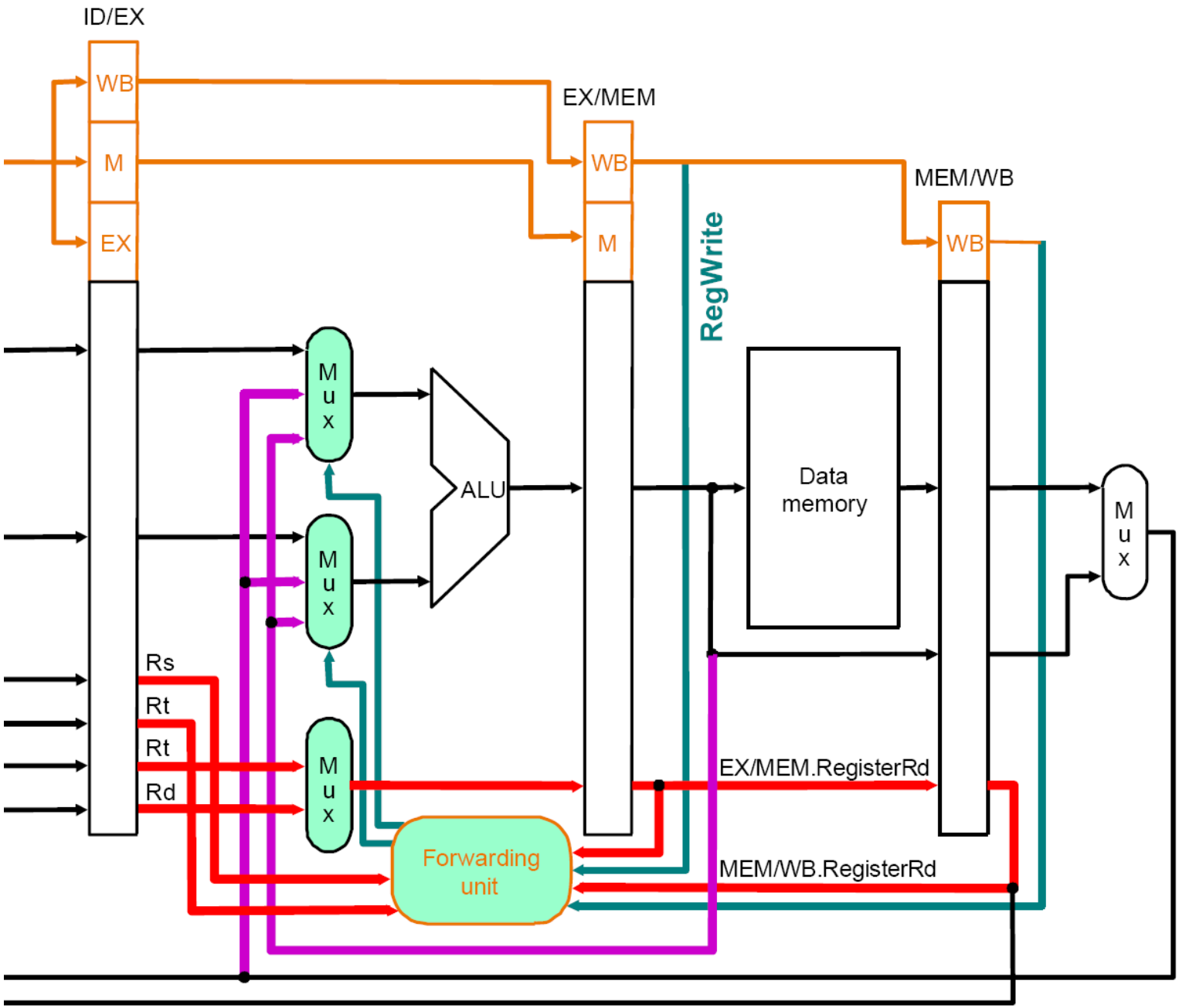
Forwards from previous instruction

2. MEM/WB haz

```
if (MEM/WB.RegWr
and (MEM/WB.Regi
and (EX/MEM.Register
and (MEM/WB.Regi
      ForwardA =
(The same for ID
```



from 2 instructions back


ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
RegWrite
Mux
Mux
Mux
ALU
Data memory
Mux
Rs
Rt
Rt
Rd
EX/MEM.RegisterRd
MEM/WB.RegisterRd
Forwarding unit

An example for forwarding

Sub	$2, $1, $3	
And	$4, $2, $5	needs forwarding from the previous instruction
Or	$4, $4, $2	needs forwarding from two instructions back
Add	$9, $4, $2	needs forwarding from 3 instructions back (thru the “transparent” GPR)

Here we discuss the $2 register only

(The first two cases are handled in the execute phase, the last one, in the decode phase).

An example for forwarding

Sub	$2, $1, $3	
And	$4, $2, $5	
Or	$4, $4, $2	needs forwarding from the previous instruction
Add	$9, $4, $2	needs forwarding from the previous instruction

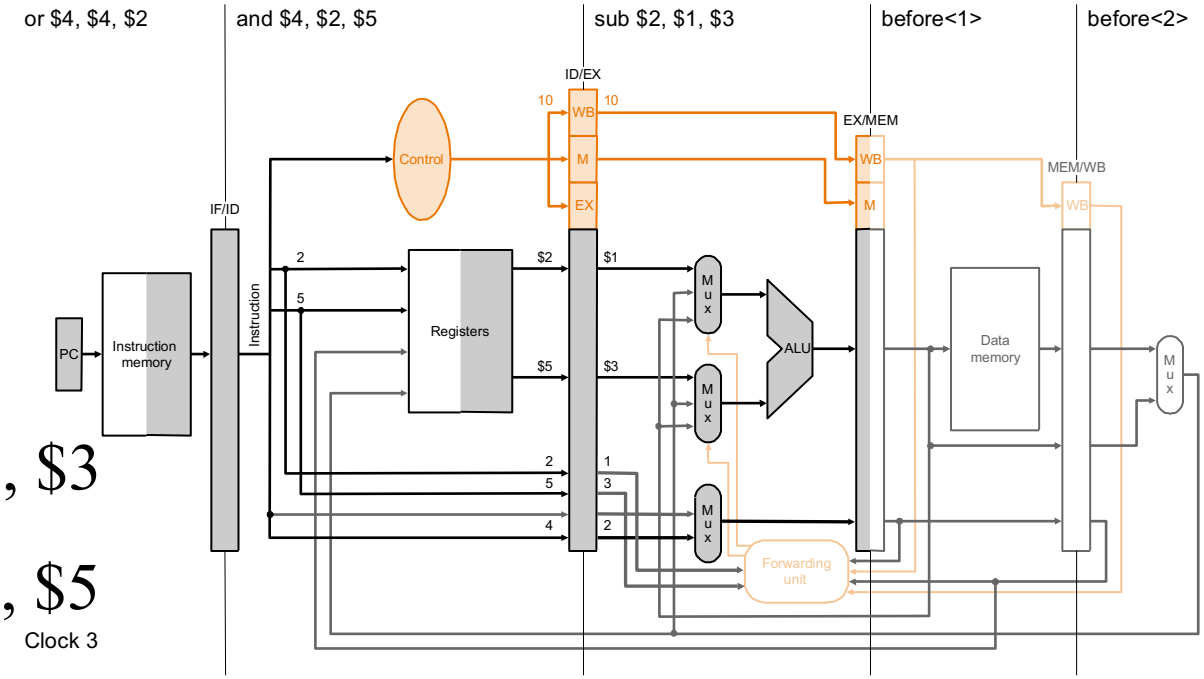
Here we discuss the $4 register and there are two case (the 2nd one in purple)

Sub $2, $1, $3

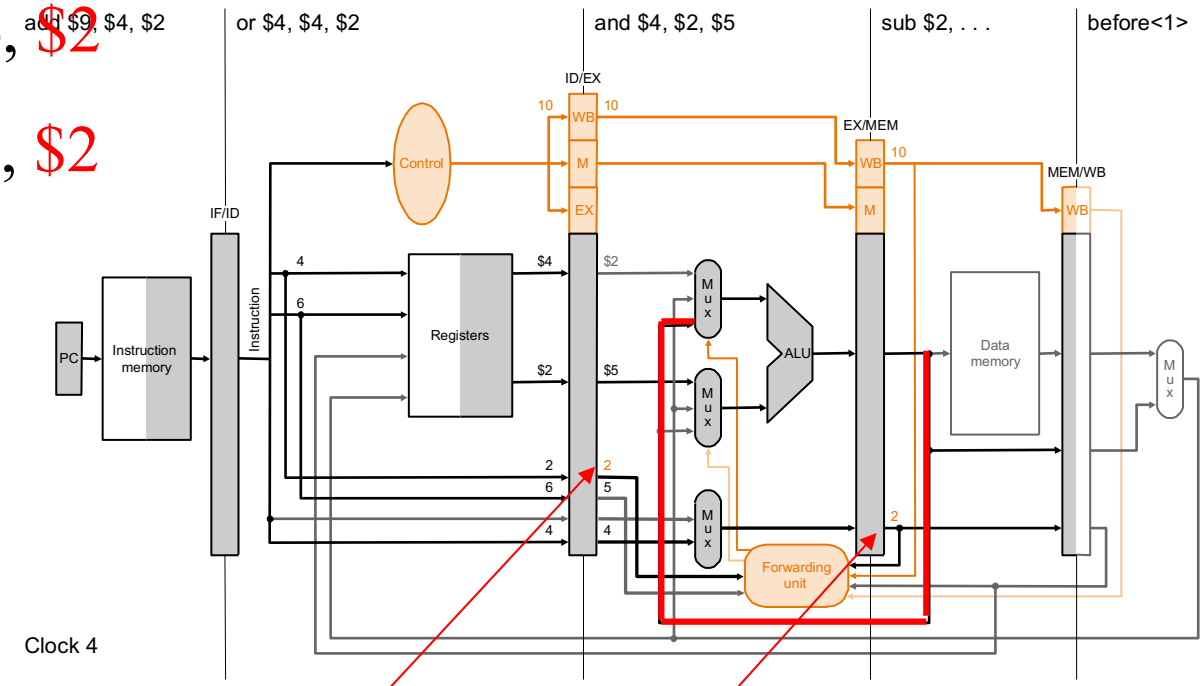
And $4, $2, $5

Or $4, $4, $2

Add $9, $4, $2

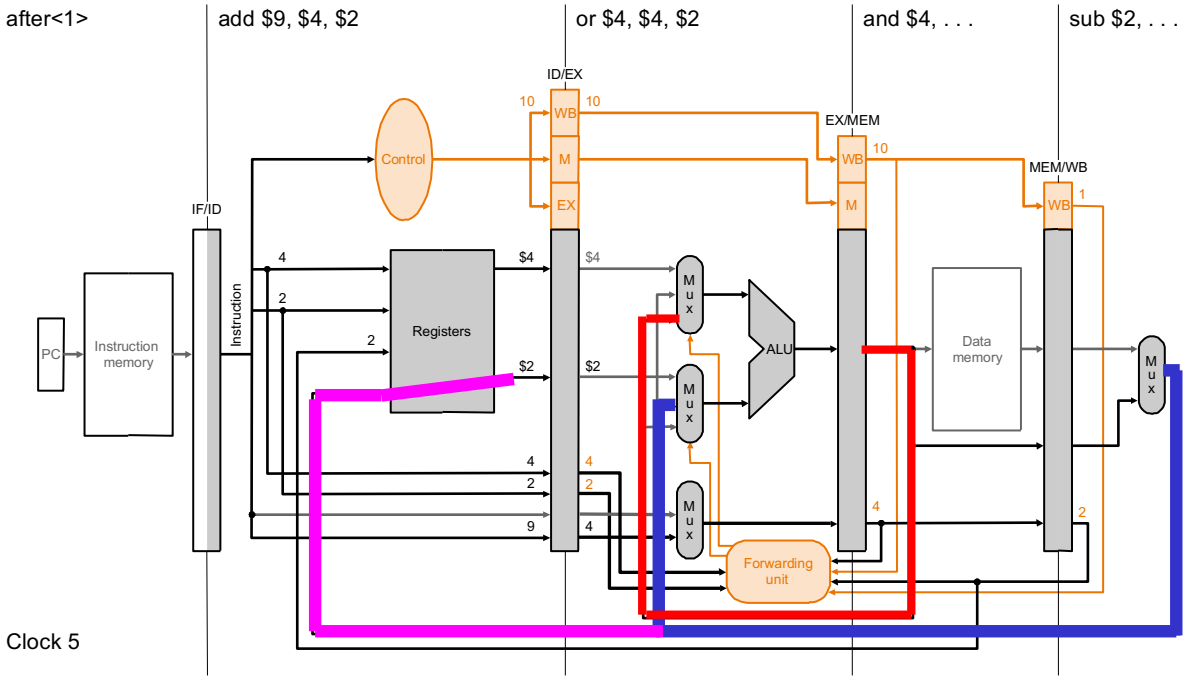


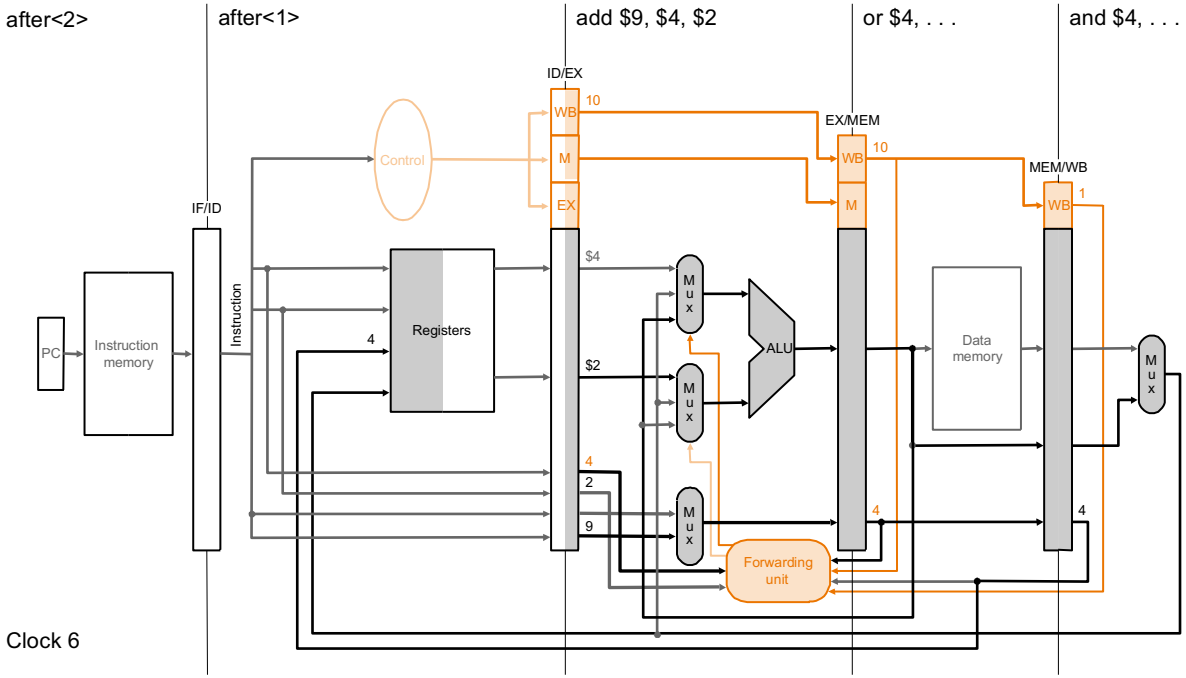
Clock 3



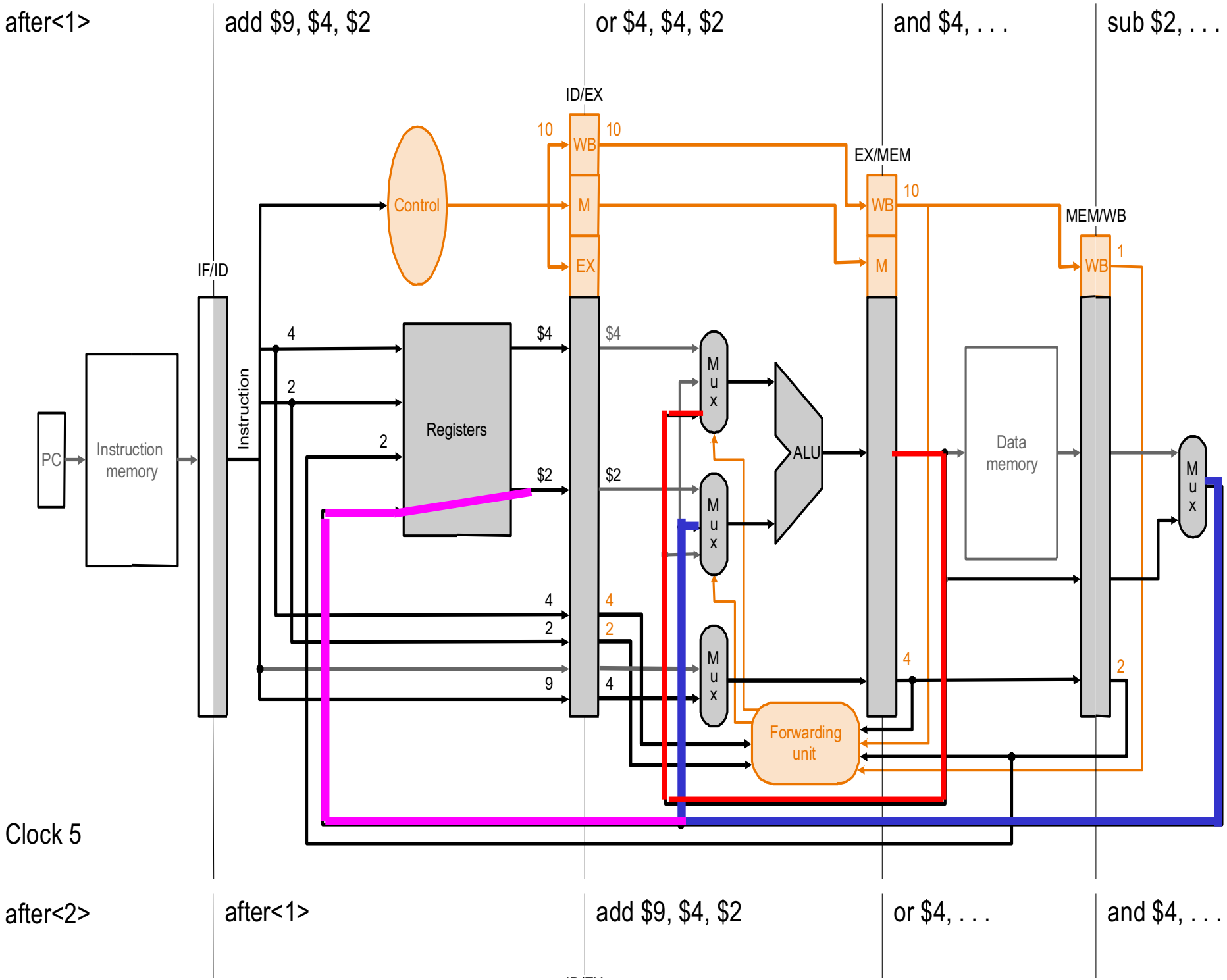
Clock 4

Since Rs=2 and Rd of previous inst. was 2, we use ALUout instead of Rs



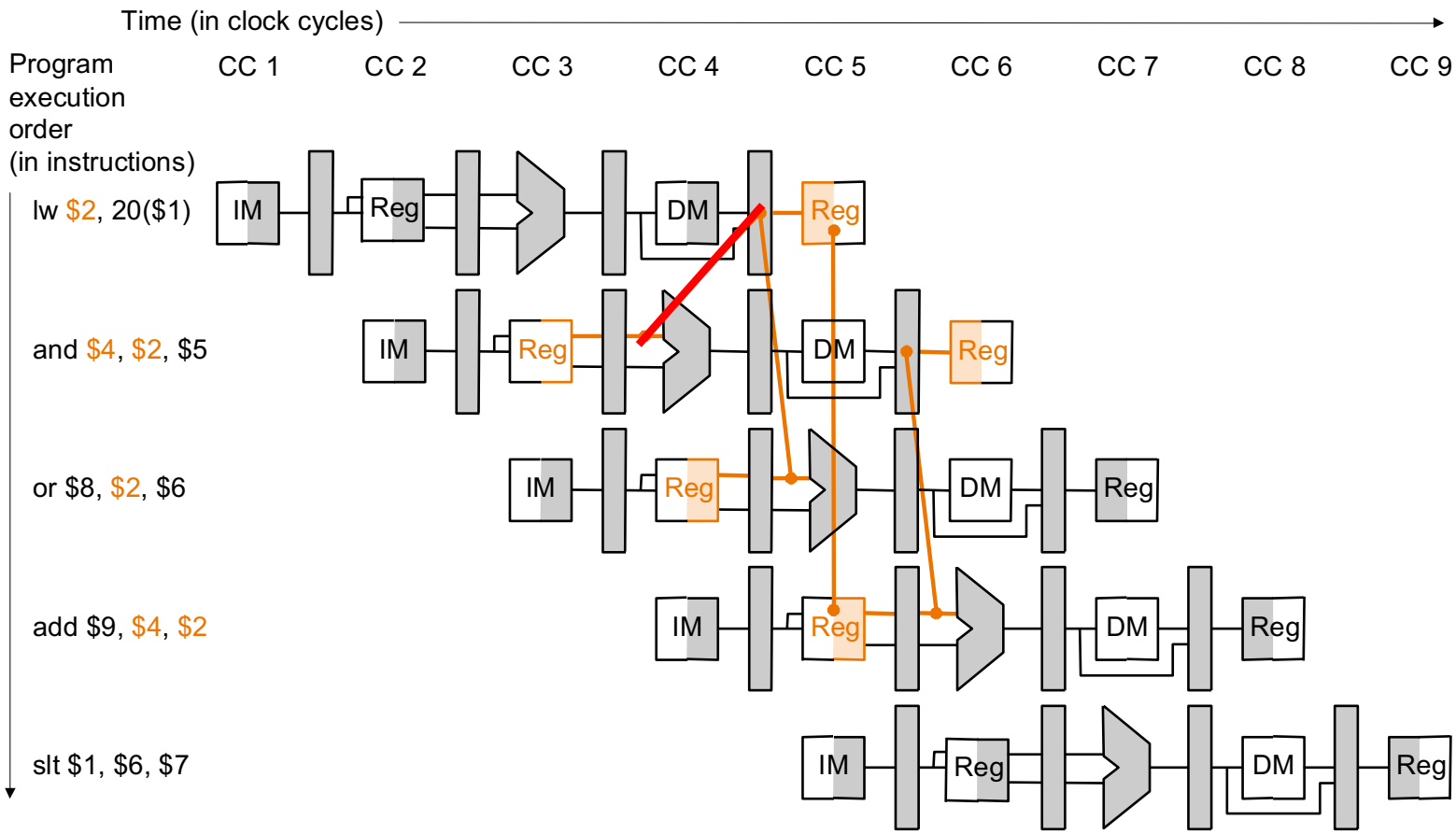


In blue we see forwarding from two instructions back (Mem->Exec.), in red, from previous instruction (WB->Exec.), in purple, from 3 instructions back (WB->Decode).


after<1>
add $9, $4, $2
or $4, $4, $2
and $4, . . .
sub $2, . . .
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
PC
Instruction memory
Instruction
Registers
ALU
Data memory
Mux
Forwarding unit
$4
$2
Clock 5
after<2>
after<1>
add $9, $4, $2
or $4, . . .
and $4, . . .

The solution does not work for lw - לא תמיד הפתרון עובד

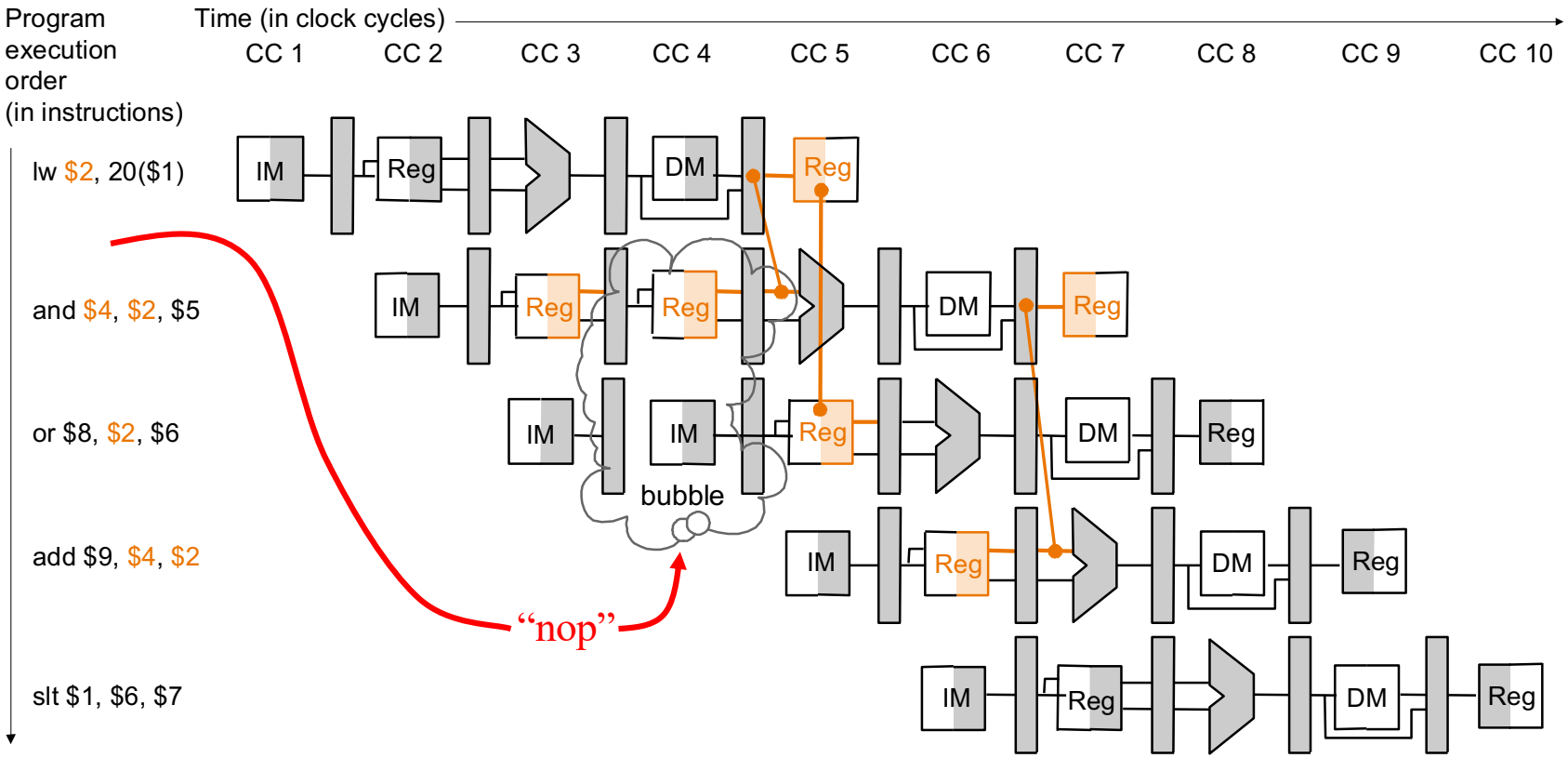
(in lw we do not have the data in the pipe!, it comes from the data memory!)



If the previous instruction was **lw** to a register and we try to use the register in the current instruction, we have a problem, since we cannot go back in time!

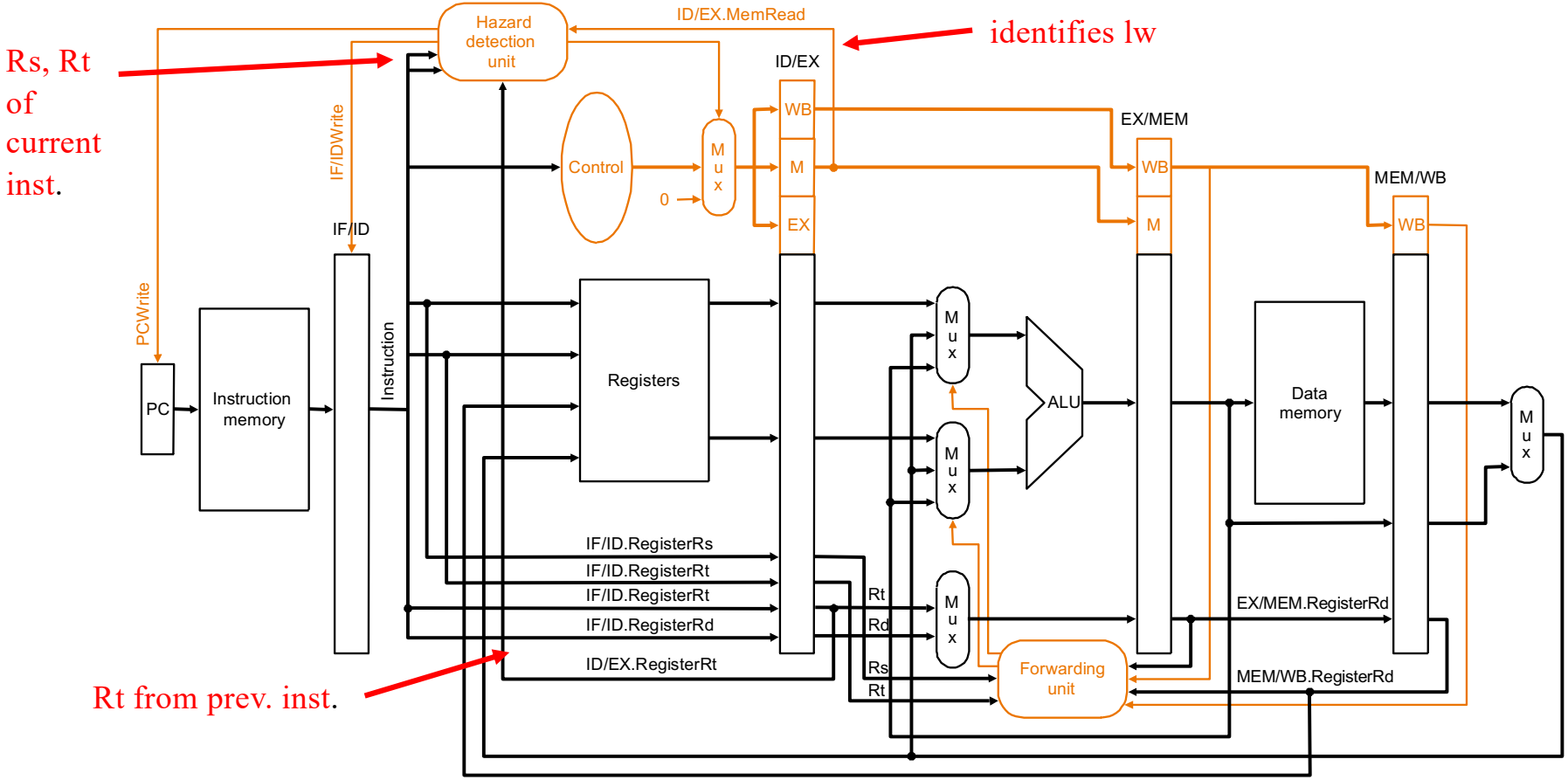
One solution is to avoid such cases by adding a **nop** (by the Assembler) whenever Rt of the **lw** is equal to Rs or Rt of the following instruction.

Another h/w solution is to add Bubbles, i.e., add nop by hardware



We need to hold IF/ID for one ck cycle and insert a "nop: into ID/EX. This is equal to adding a nop instruction by the Assembler.

Hazard detection unit



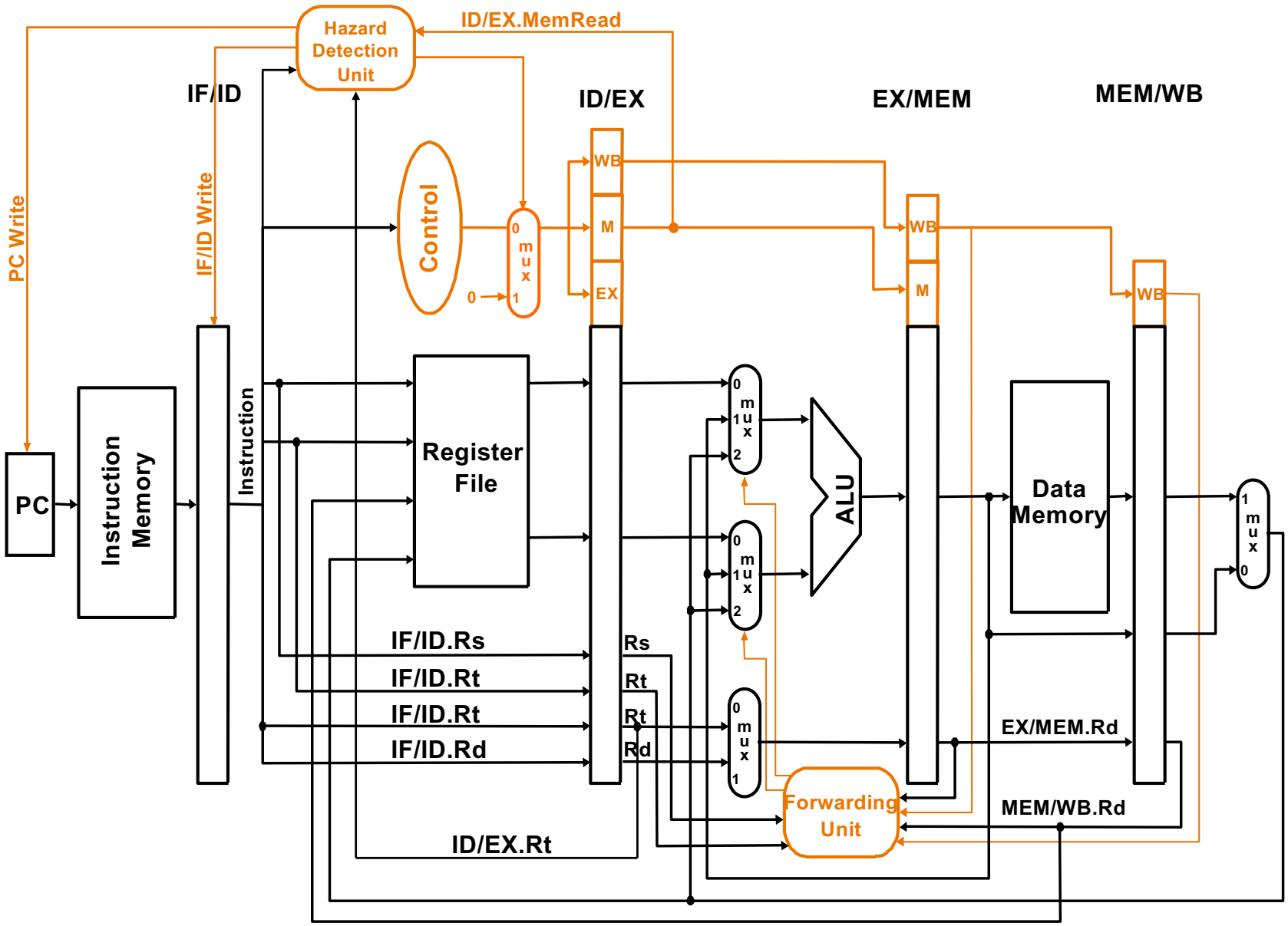
We need to hold the IF/ID and PC for one ck cycle and insert a “nop: into ID/EX. This is equal to adding a nop instruction by the Assembler.

If (ID/EX.MemRd)&& ((ID/EX.Rt= =IF/ID.Rs) || (ID/EX.Rt= =IF/ID.Rt)) we must “stall” the pipeline! This means that prev. inst was lw and it was to the current Rs or Rt. (of course if one of them is not used, don’t stall)

Holding means”freeze” the IF/ID and the PC for 1 clock cycle

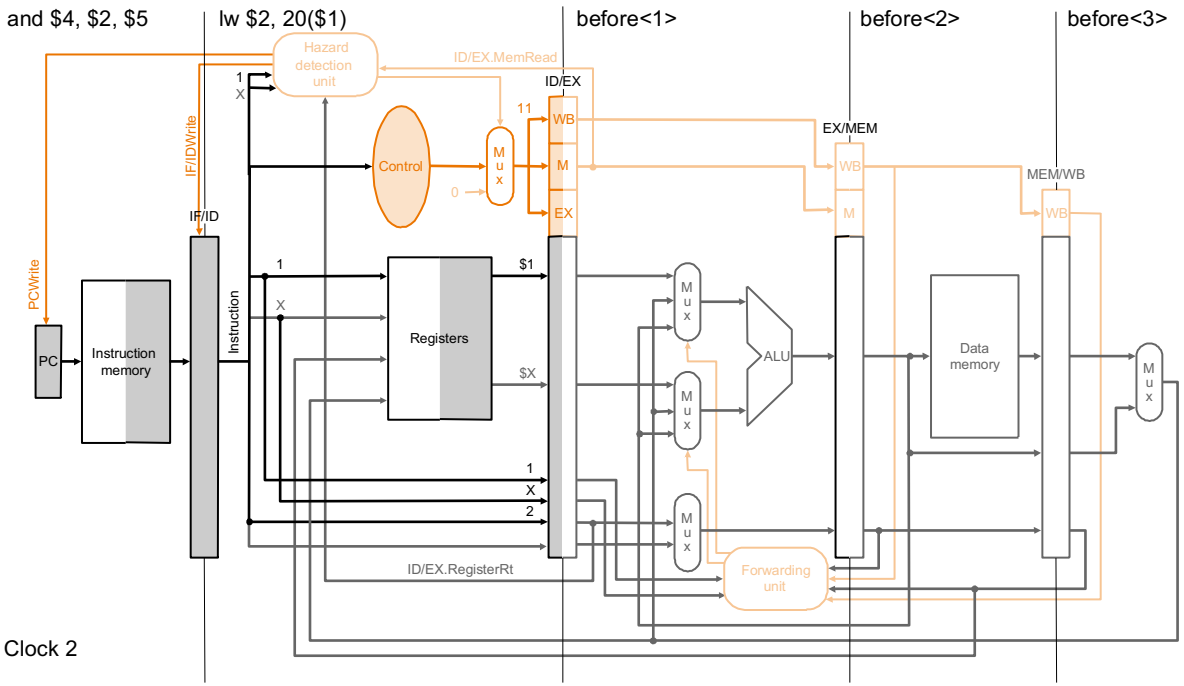
Hold the IF/ID by not giving a IF/IDWrire signal and do not increment the PC (which already points at the nex instruction) by not giving the PCWrite signal. Inserting a nop is by clearing all control signals.

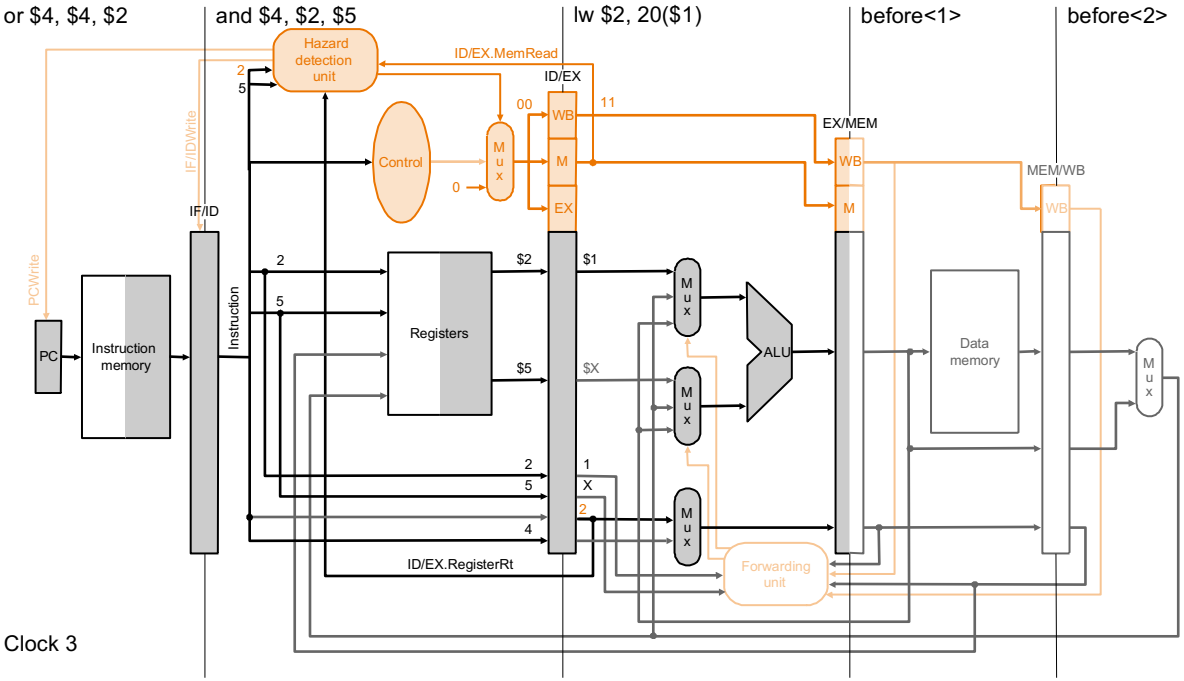
Forwarding + Hazard Detection Unit

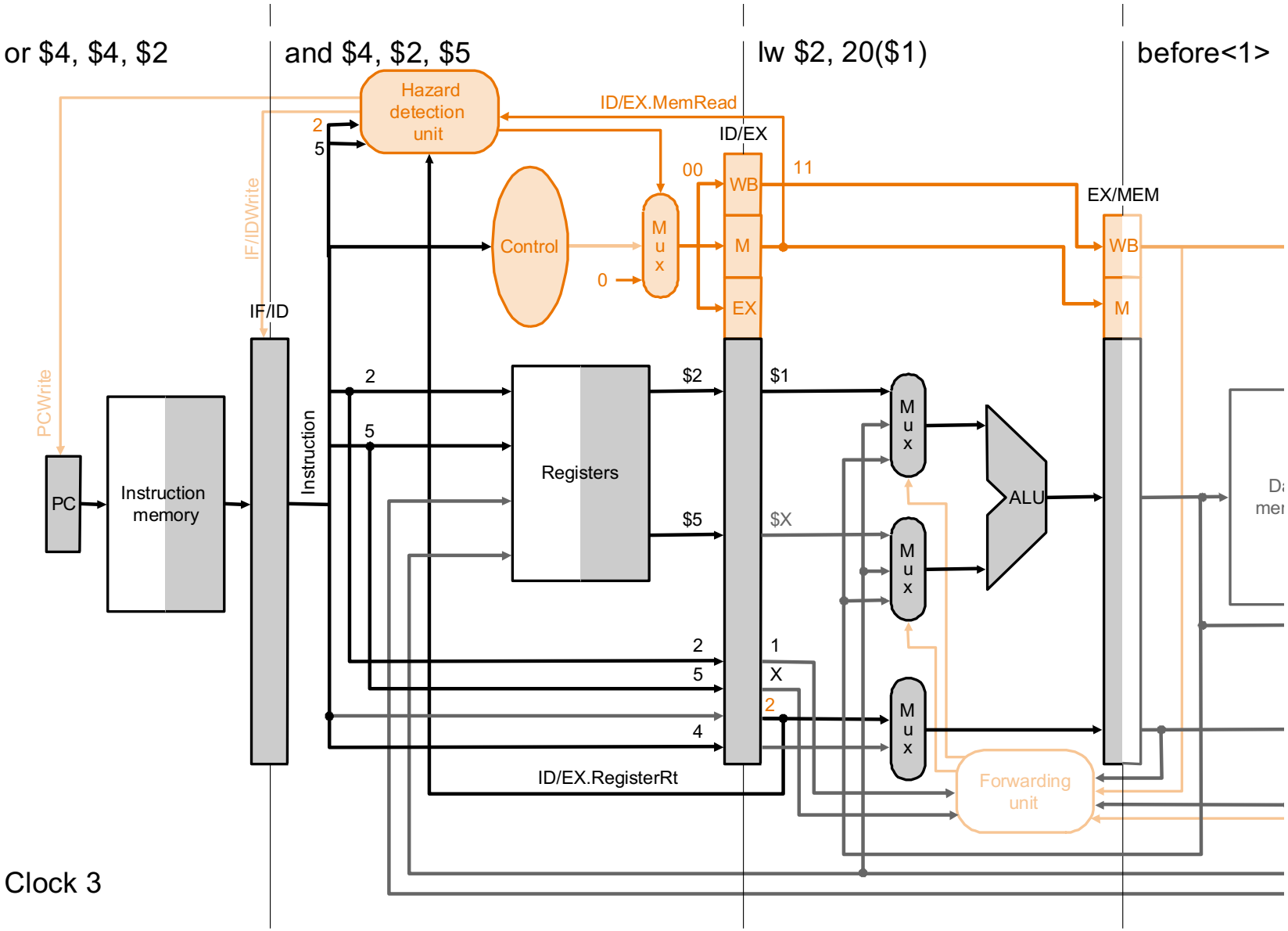


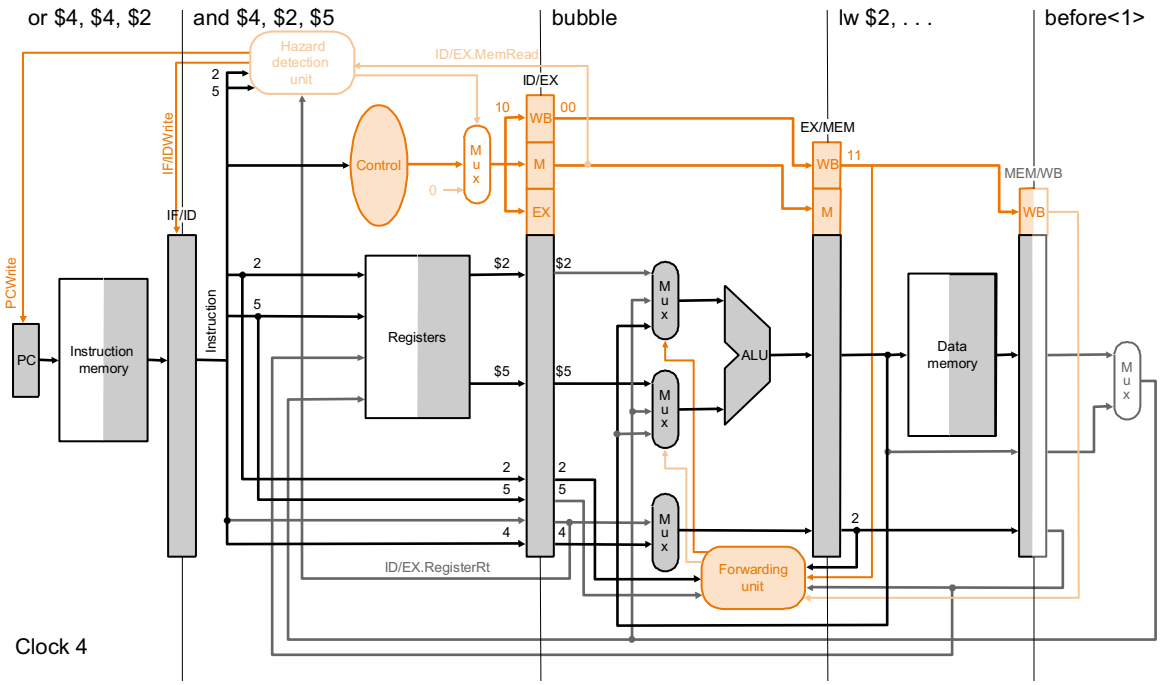
An example for lw hazard detection

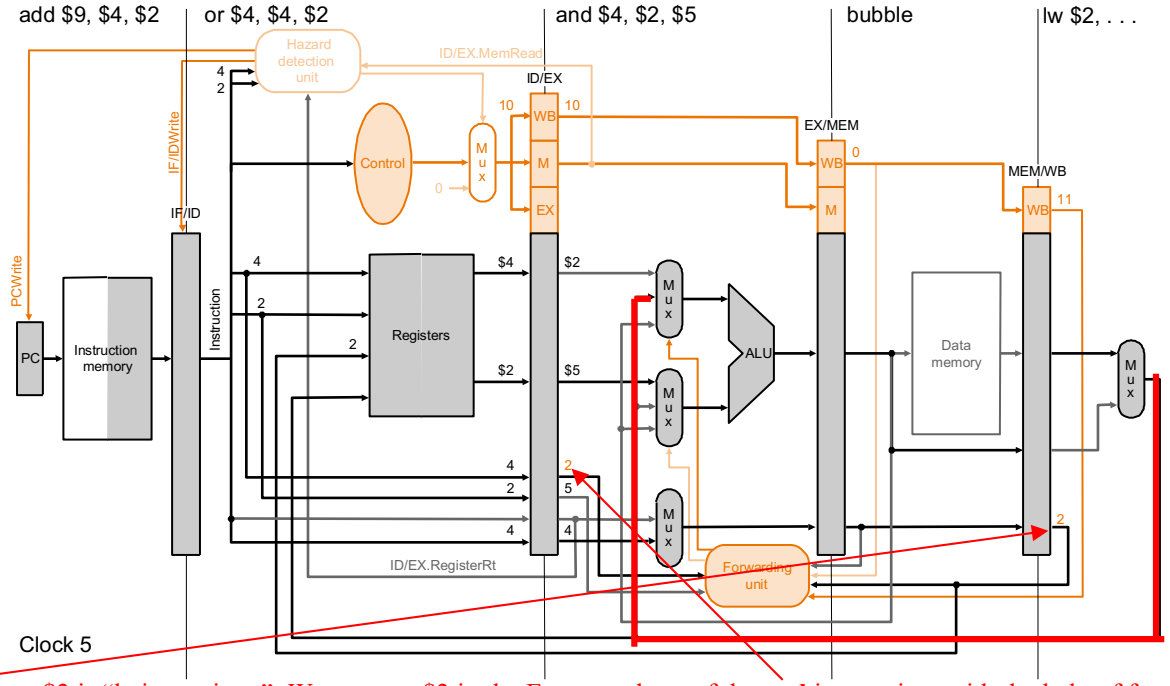
```
lw      $2, 20($1)
And     $4, $2, $5
Or      $4, $4, $2
Add     $9, $4, $2
```


and $4, $2, $5
lw $2, 20($1)
before<1>
before<2>
before<3>
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
Control
Mux
WB
M
EX
IF/IDWrite
IF/ID
PCWrite
PC
Instruction memory
Instruction
Registers
$1
$X
ALU
Data memory
ID/EX.RegisterRt
Forwarding unit
Clock 2

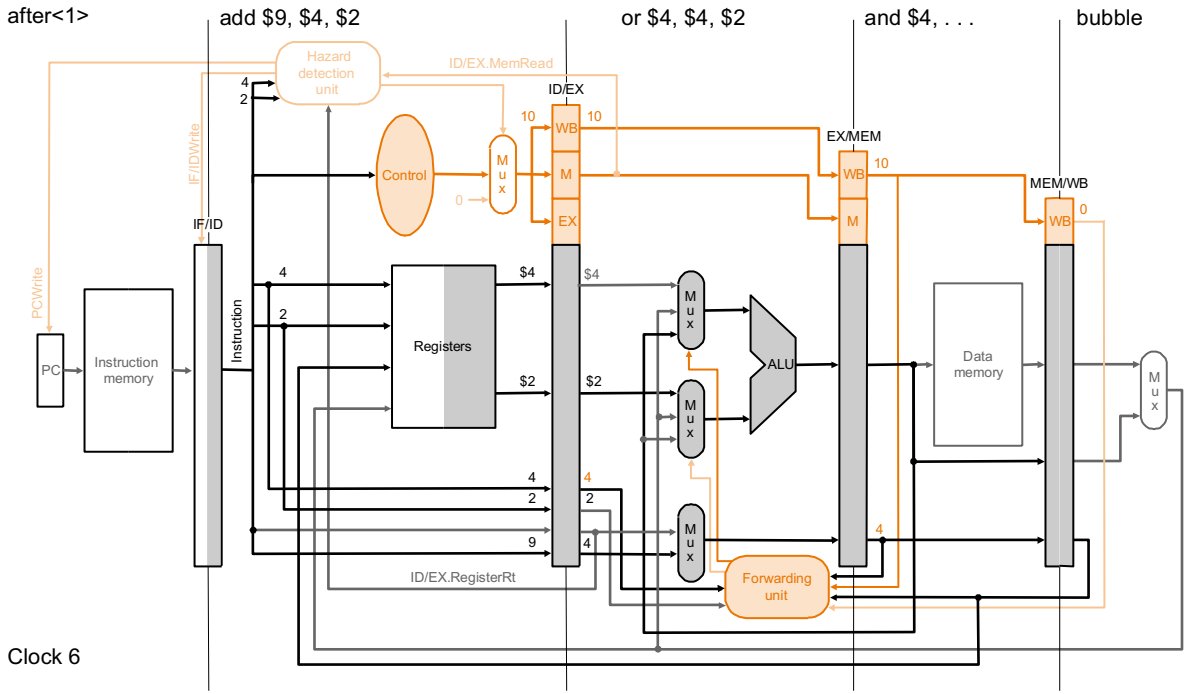

or $4, $4, $2
and $4, $2, $5
lw $2, 20($1)
before<1>
before<2>
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
Control
Mux
WB
M
EX
IF/IDWrite
IF/ID
PCWrite
PC
Instruction memory
Instruction
Registers
$2
$1
$5
$X
ALU
Data memory
ID/EX.RegisterRt
Forwarding unit
Clock 3

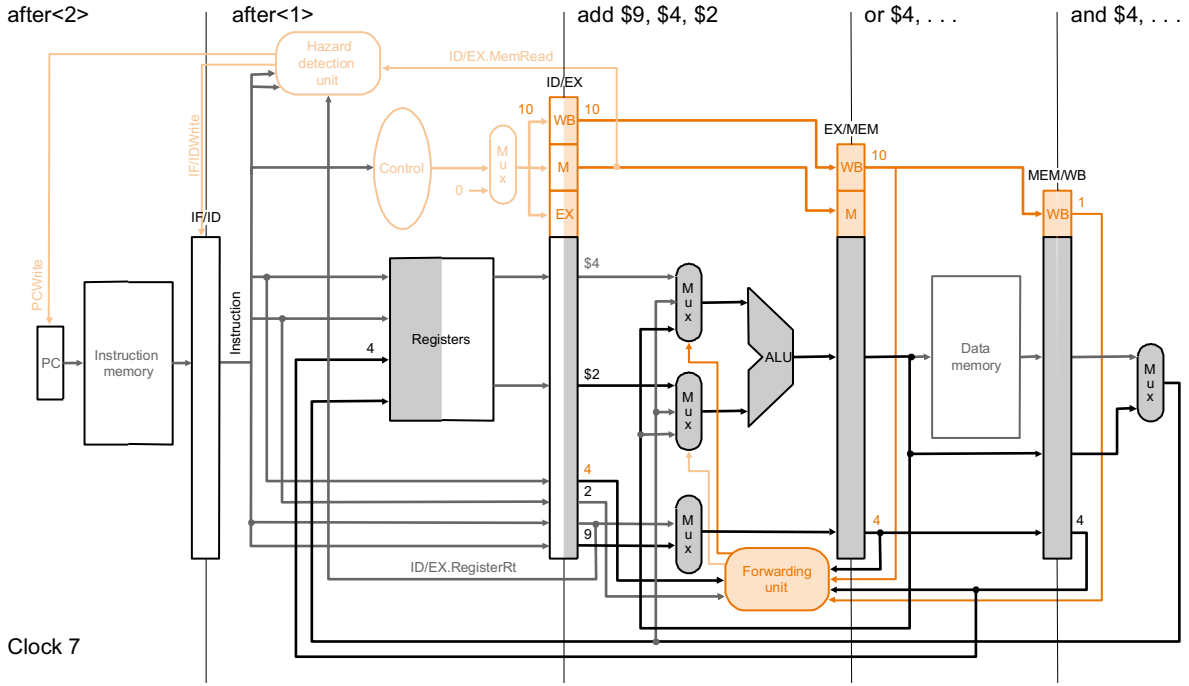

or $4, $4, $2
and $4, $2, $5
lw $2, 20($1)
before<1>
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
IF/IDWrite
PCWrite
IF/ID
Instruction
PC
Instruction memory
Control
Mux
WB
M
EX
Registers
$2
$5
$1
$X
ALU
Forwarding unit
ID/EX.RegisterRt
Clock 3





The **lw** instruction is in the WB phase. $2 is "being written". We can use $2 in the Execute phase of the **and** instruction, with the help of forwarding.


after<1>
add $9, $4, $2
or $4, $4, $2
and $4, . . .
bubble
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/IDWrite
PCWrite
IF/ID
Instruction
Control
Mux
WB
M
EX
PC
Instruction memory
Registers
$4
$2
ALU
Data memory
ID/EX.RegisterRt
Forwarding unit
Clock 6


after<2>
after<1>
add $9, $4, $2
or $4, . . .
and $4, . . .
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/IDWrite
PCWrite
IF/ID
Instruction
Control
Mux
WB
M
EX
PC
Instruction memory
Registers
$4
$2
ALU
Data memory
ID/EX.RegisterRt
Forwarding unit
Clock 7

Peer Instruction

A. Thanks to pipelining, I have reduced the time it took me to wash my shirt.

B. Longer pipelines are always a win (since less work per stage & a faster clock).

C. We can rely on compilers to help us avoid data hazards by reordering instrs.

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT

Peer Instruction Answer

A. Throughput better, not execution time

B. "…longer pipelines do usually mean faster clock, but branches cause problems!

C. "they happen too often & delay too long." Forwarding! (e.g, Mem ⇒ ALU)

A. Thanks to pipelining, I have reduced the time it took me to wash my shirt

FALSE

B. Longer pipelines are always a win (since less work per stage & a faster clock).

FALSE

C. We can rely on compilers to help us avoid data hazards by reordering instrs.

FALSE

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT

Summary of hazards

Data hazards:

* Forward from previous instruction

* Forward from two instructions ago

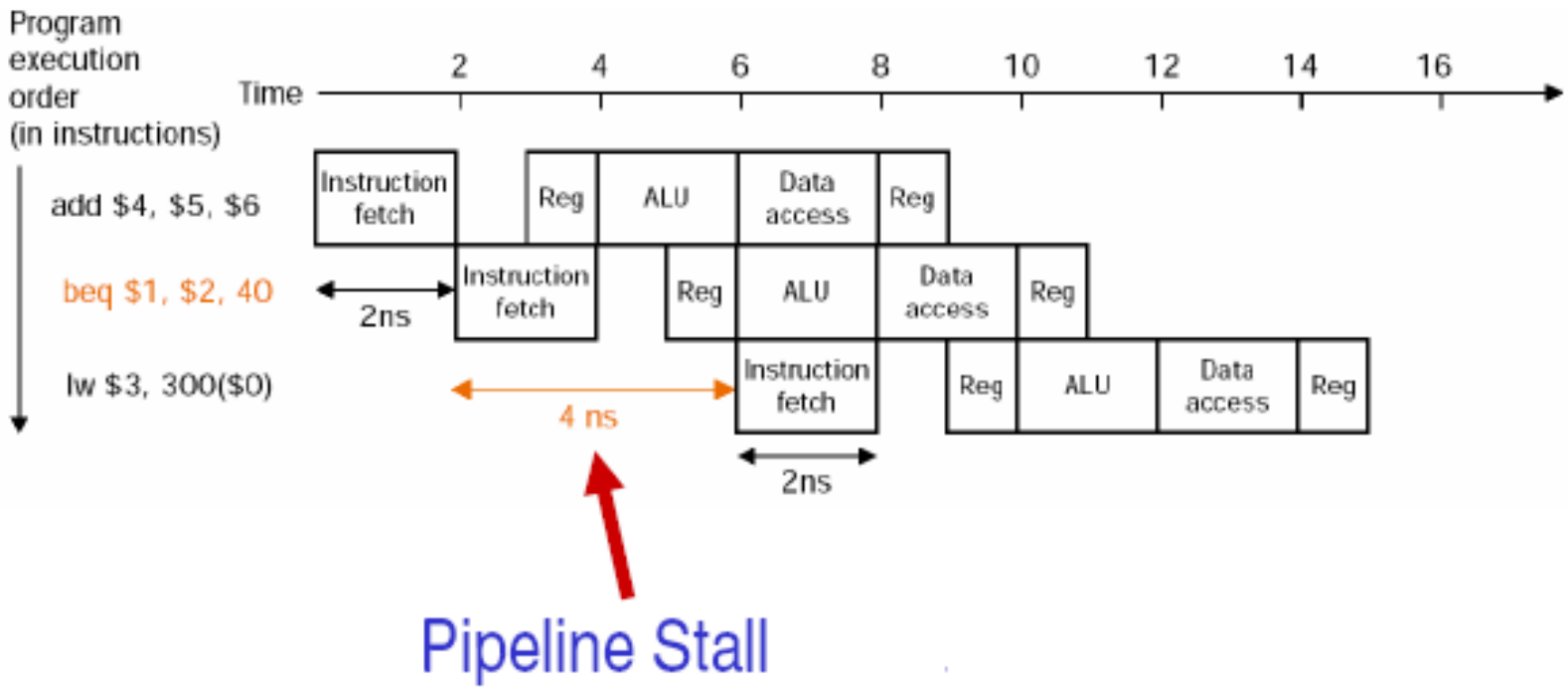
* (Forward "transparent"GPR = from 3 instructions ago)

* If we cannot forward, (after lw) we stall the pipe by inserting a nop and freezing IF/ID and PC for 1 ck cycle

Dealing With Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

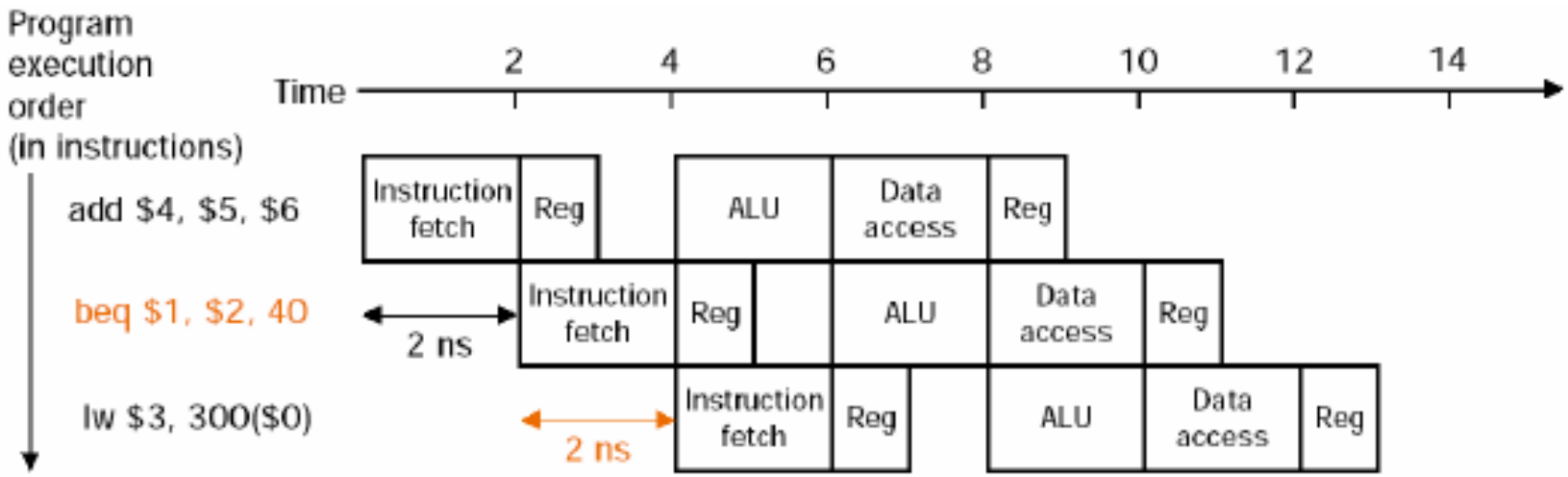
Solution 1: Stall



Dealing With Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Solution 2: Predict the Branch Target



Dealing With Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Solution 2: (Mis)Predict the Branch Target