5118007-02 Computer Architecture

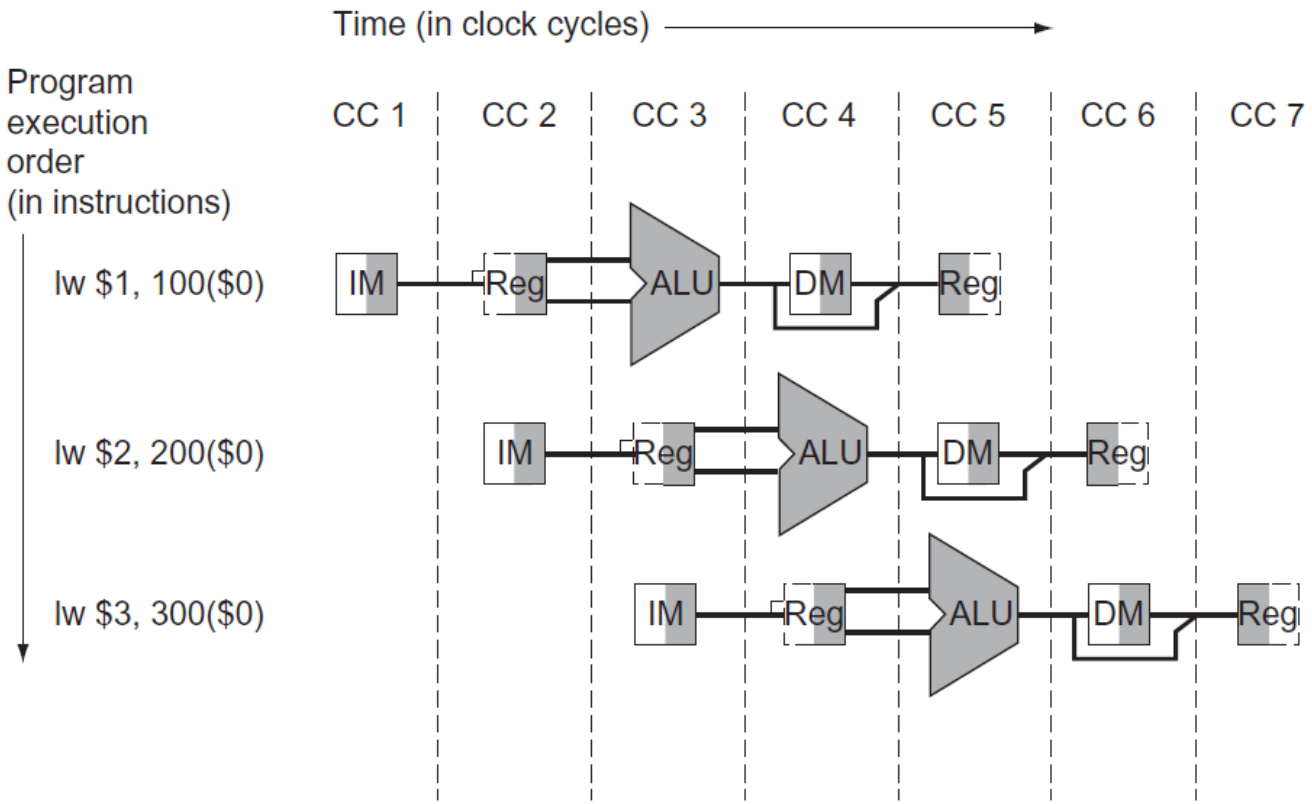
Ch. 4 The Processor

21 May 2024

Shin Hong

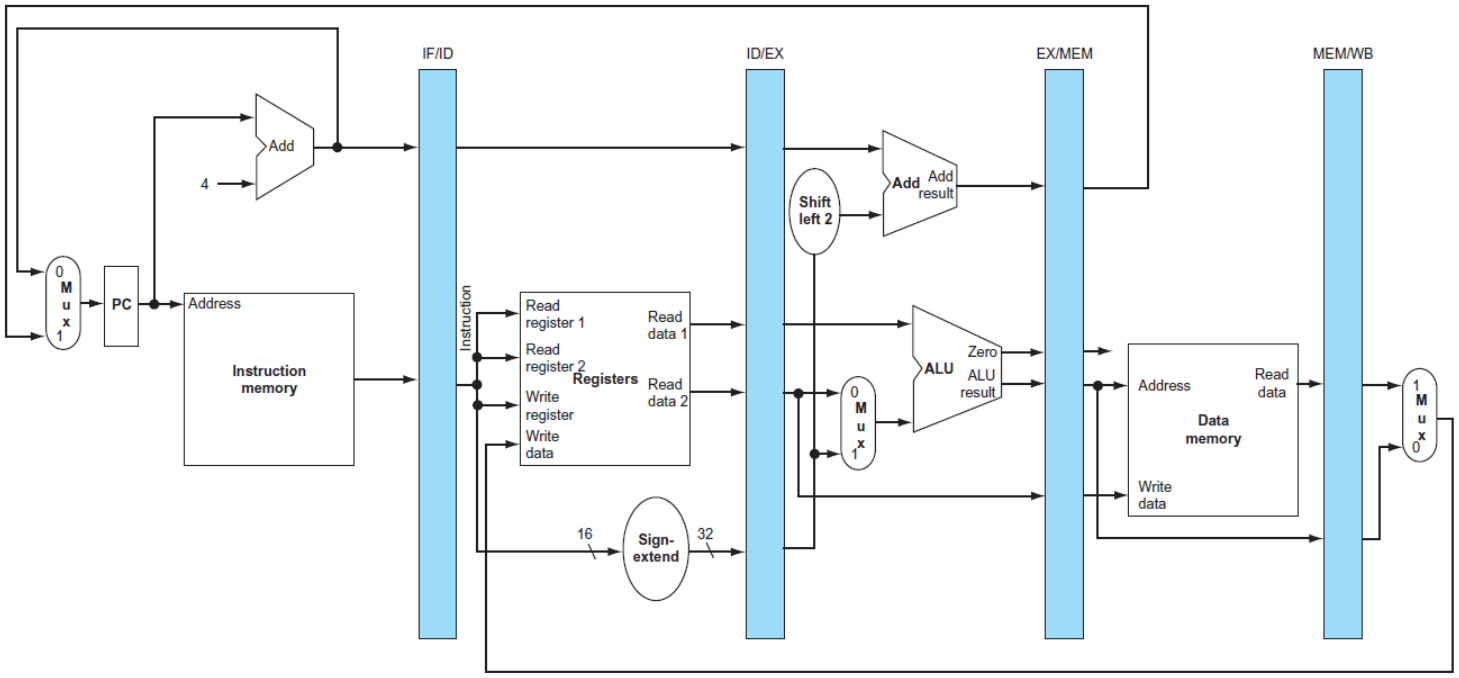
Pipelining of Instruction Executions

- K instruction executions can be overlapped over K steps

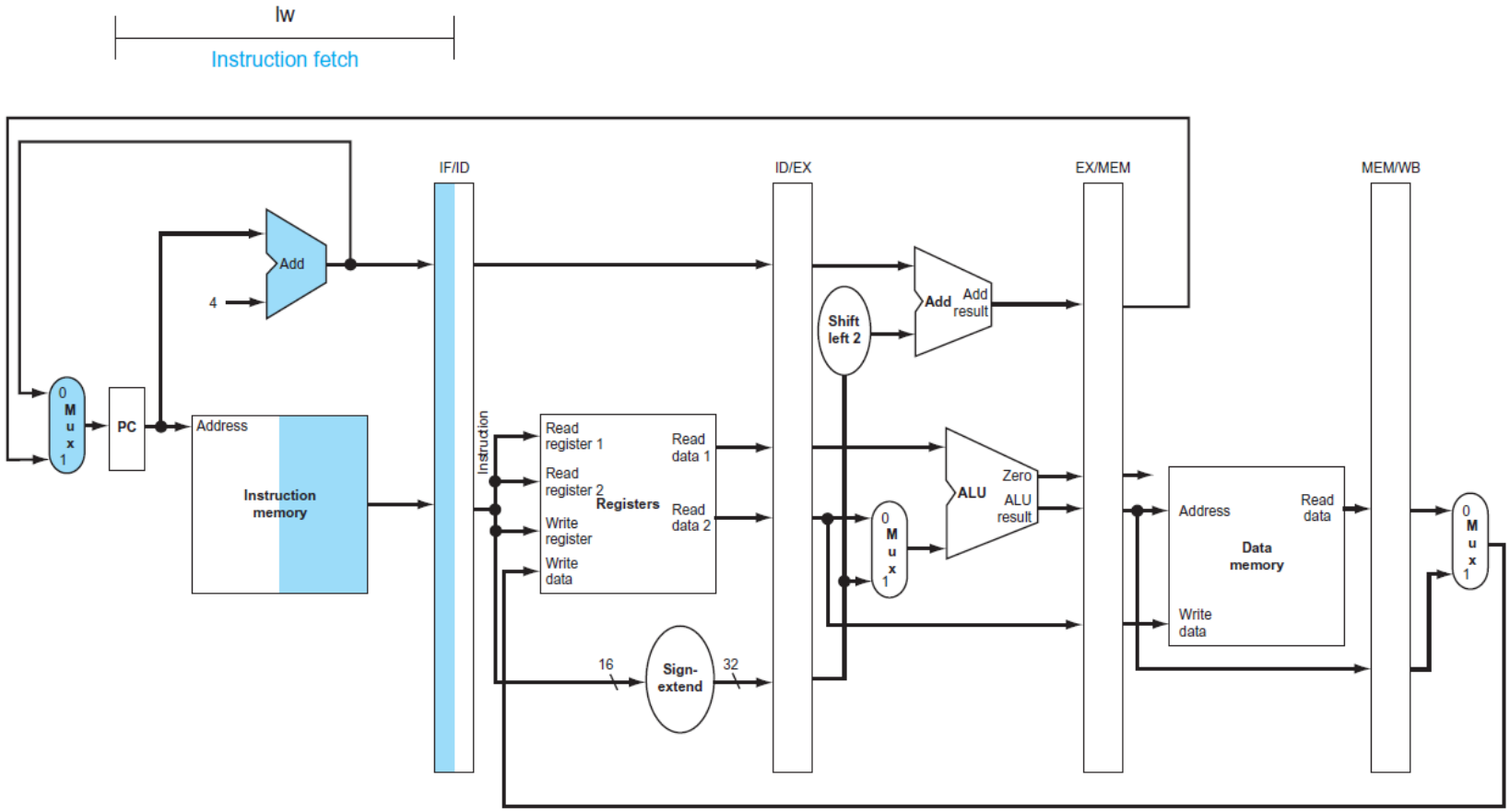


Pipeline Registers

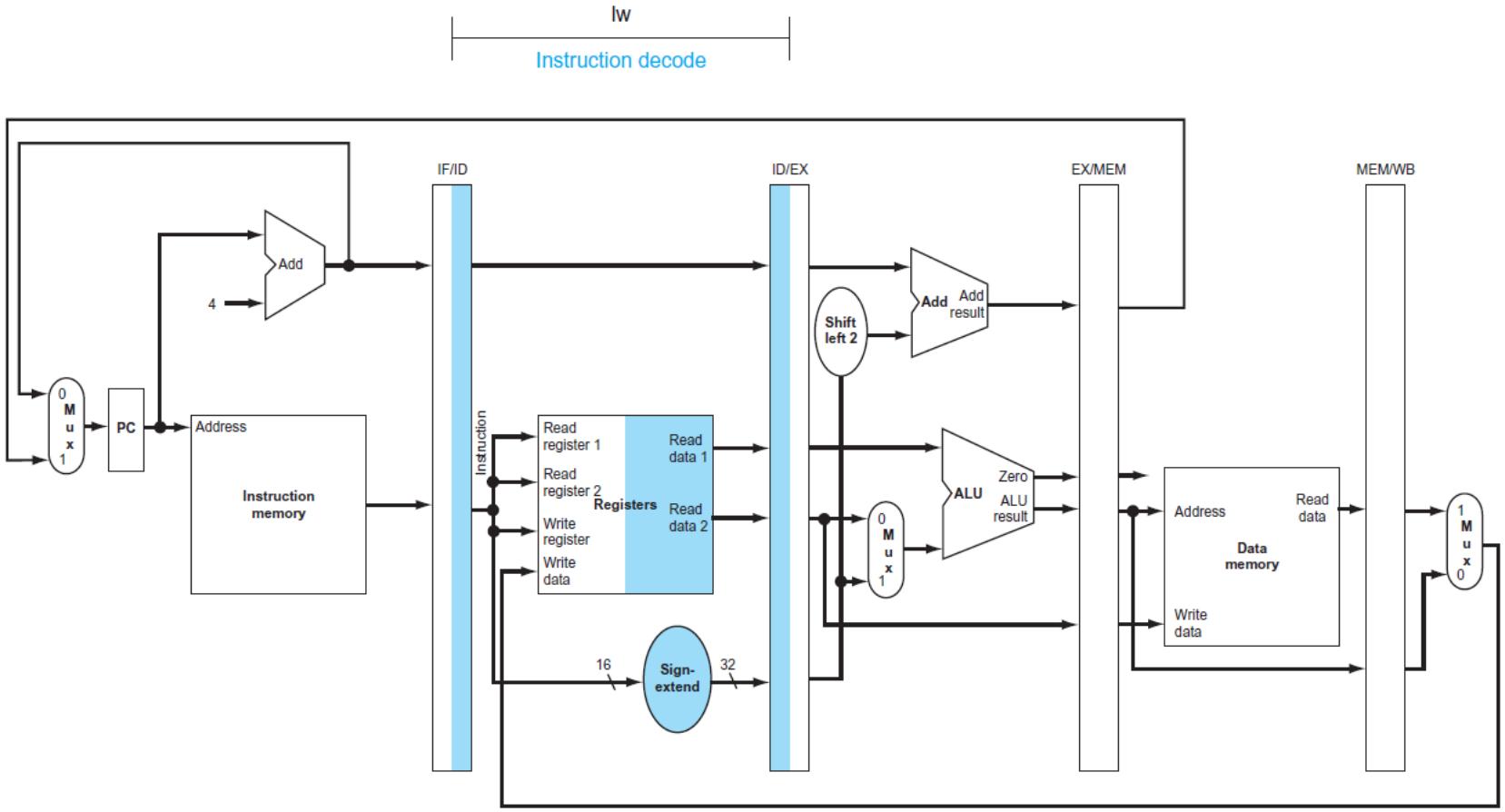
- Transfer data and control via pipeline registers: all instructions advance during each clock cycle from one pipeline register to the next
- Load data at each clock cycle



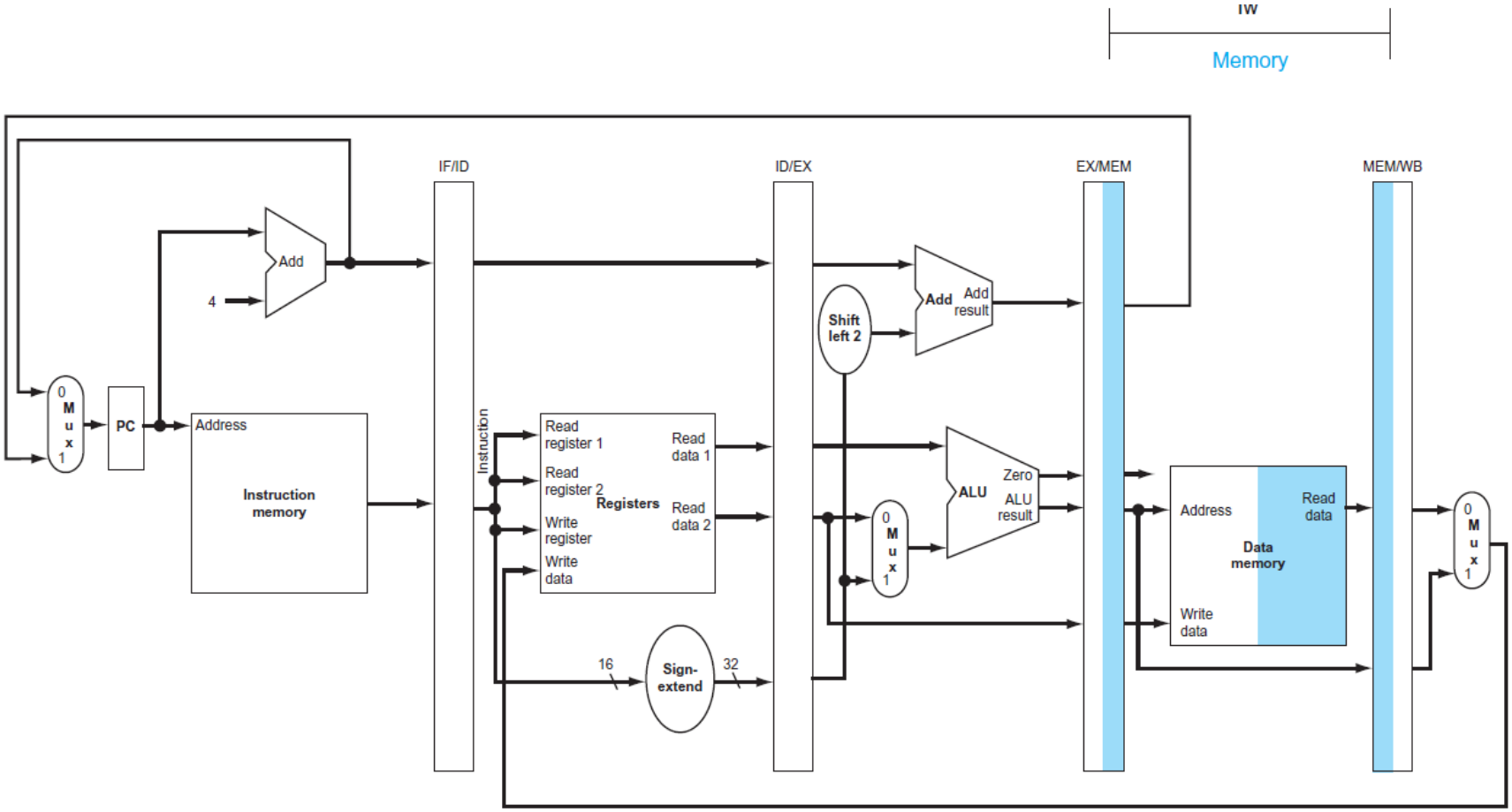
Ex. Load Instruction



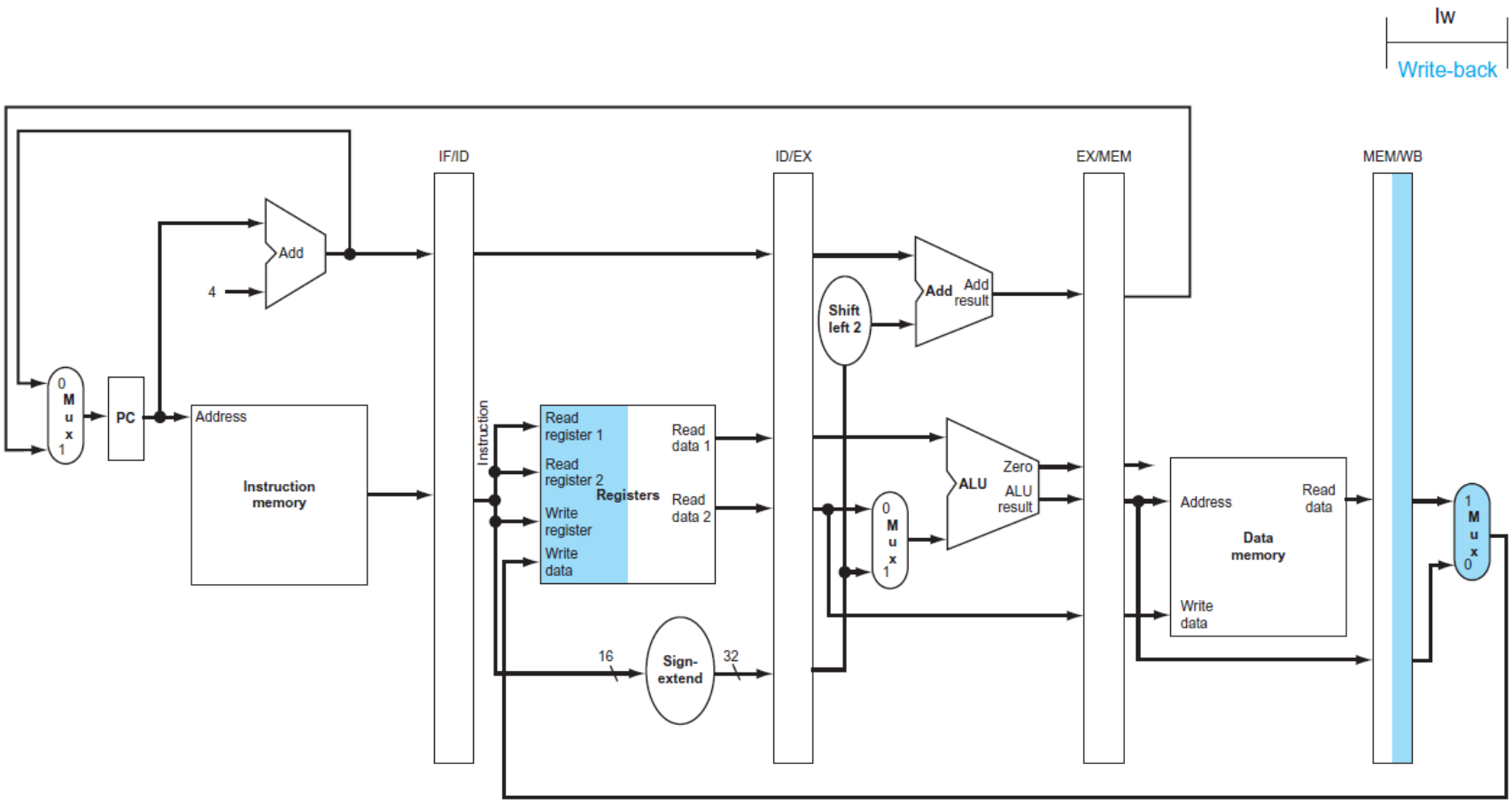
Ex. Load Instruction



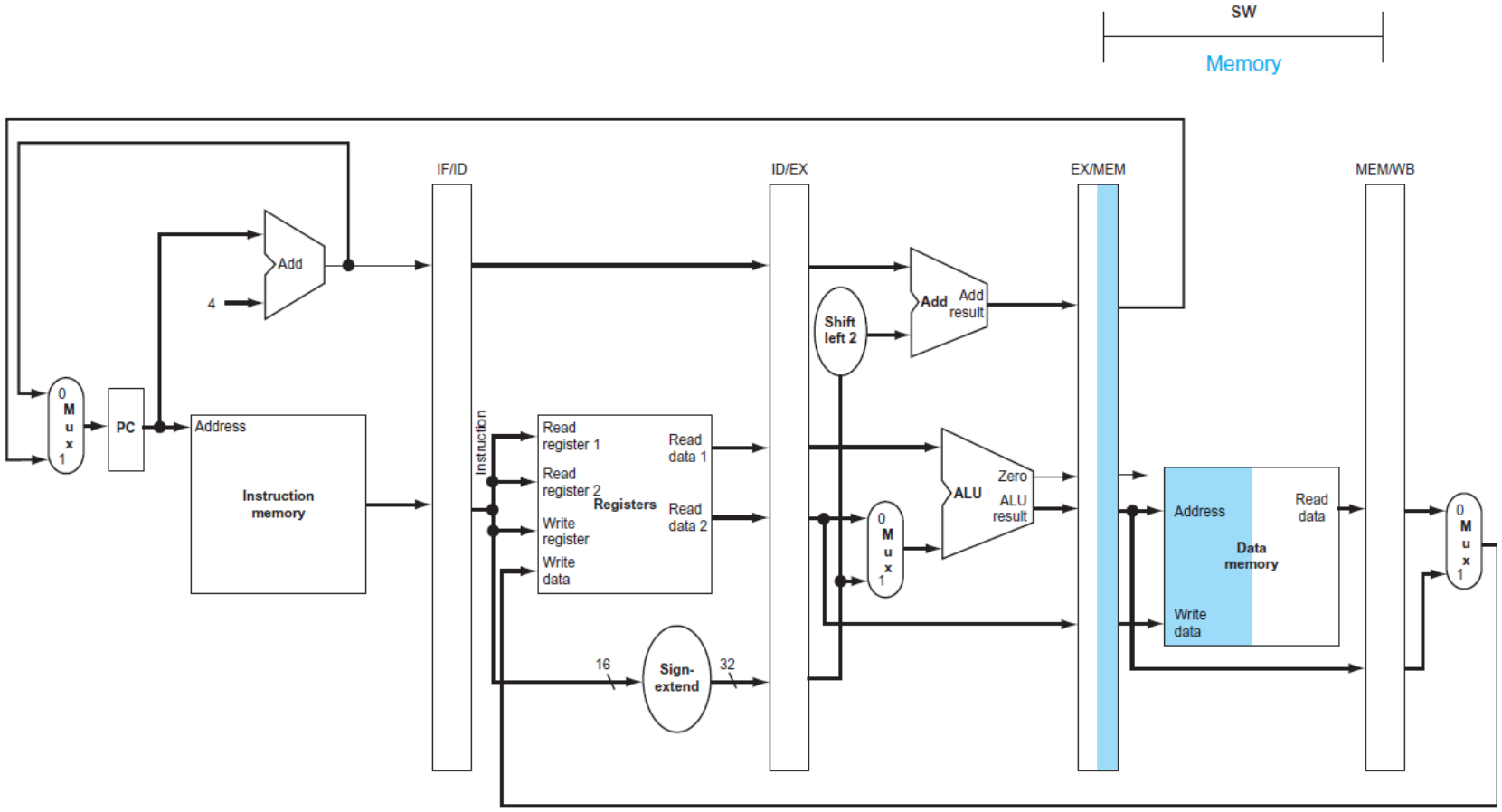
Ex. Load Instruction



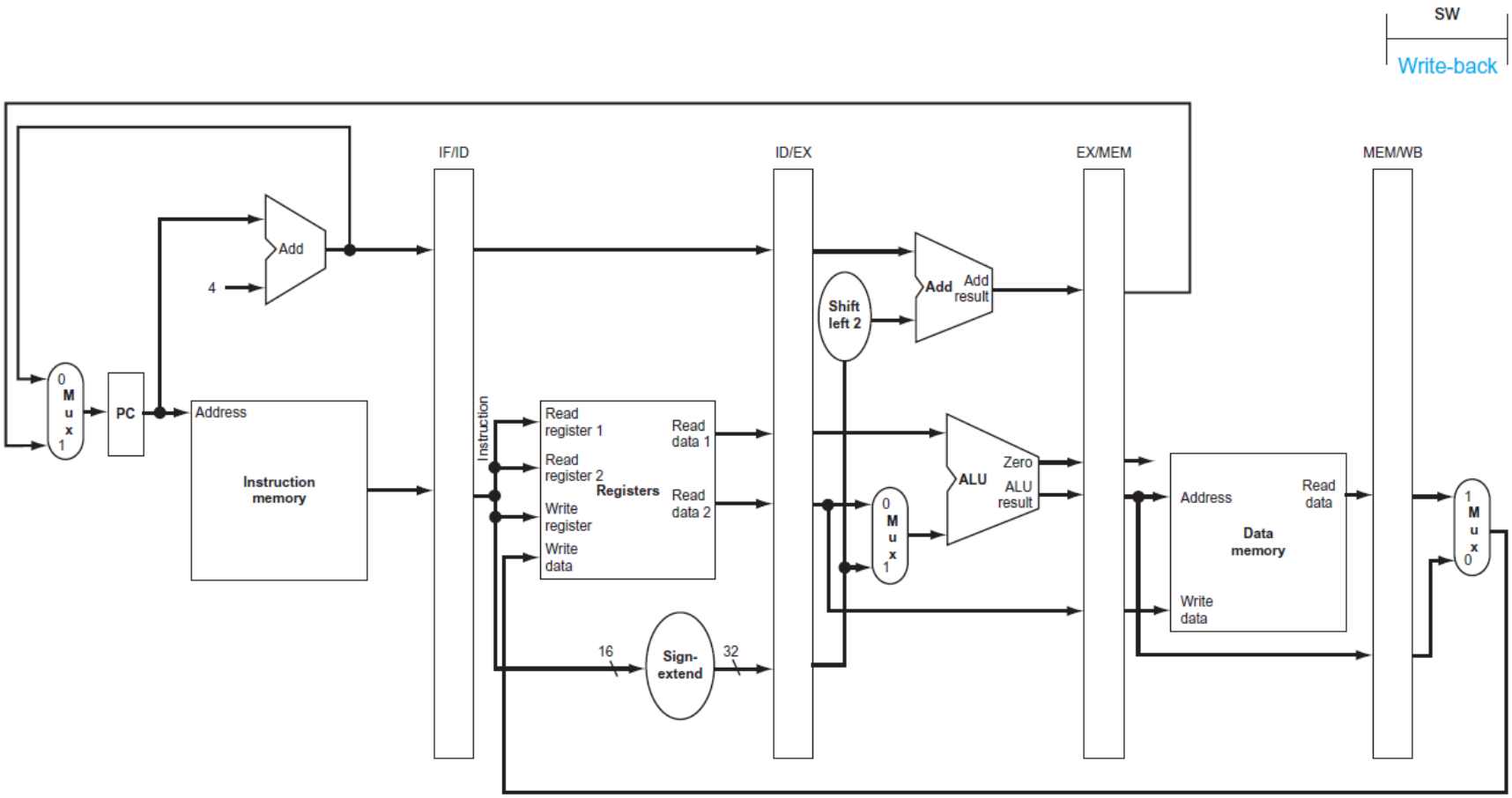
Ex. Load Instruction



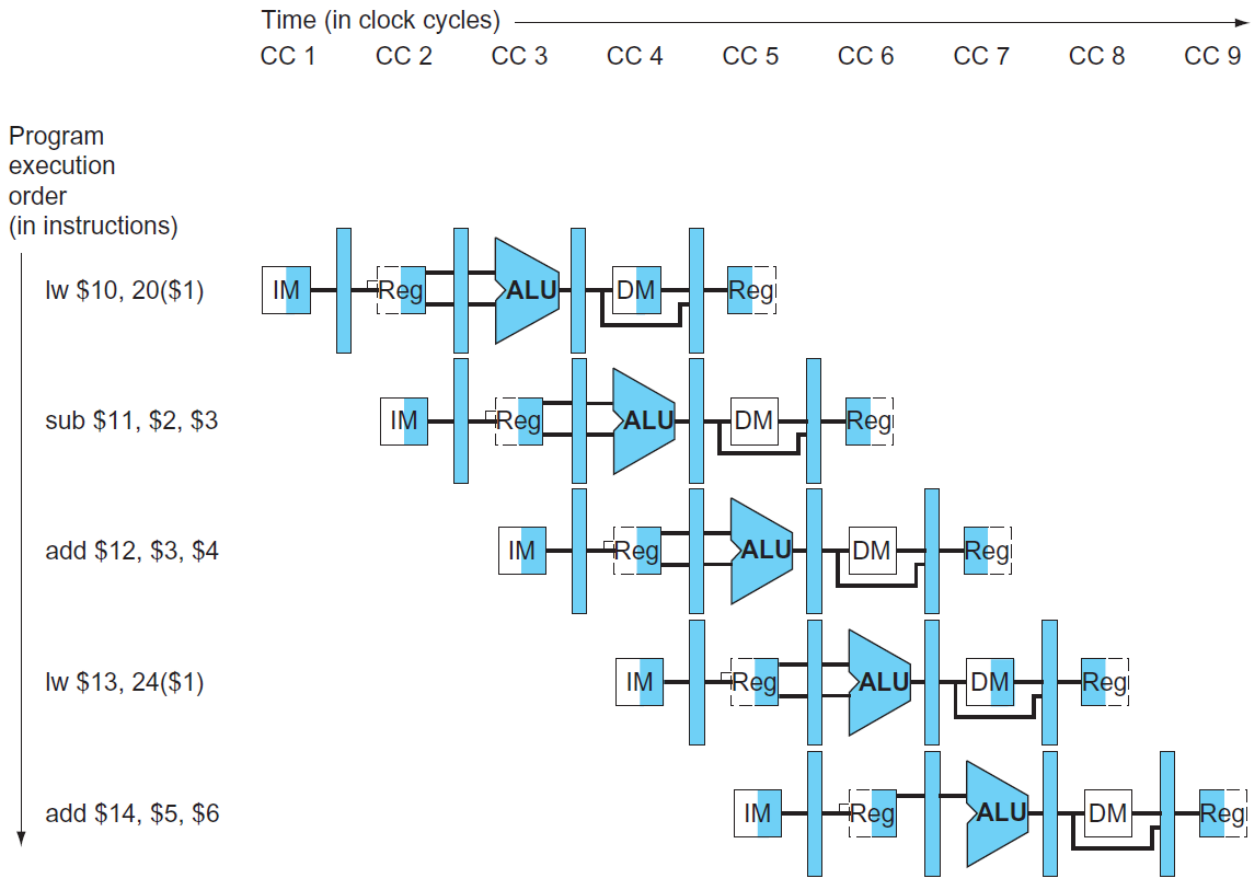
Ex. Store Instruction



Ex. Store Instruction



Multi-clock-cycle Pipeline Diagram



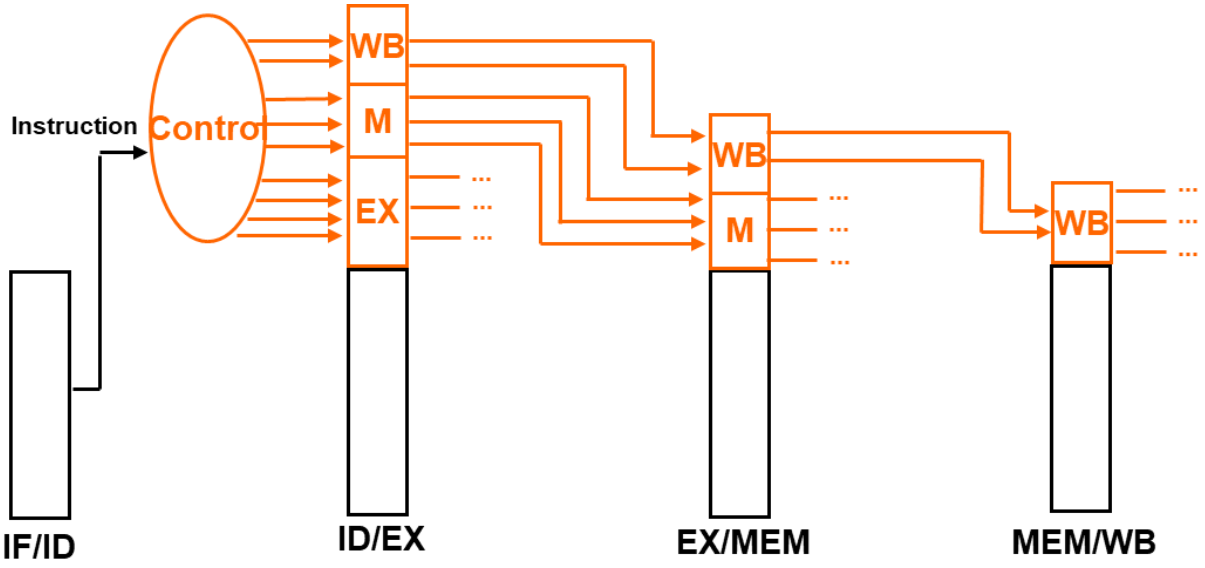
Correct Arguments?

1. Allowing jumps, branches, and ALU instructions to take fewer stages than the five required by the load instruction will increase pipeline performance under all circumstances.
2. Trying to allow some instructions to take fewer cycles does not help, since the throughput is determined by the clock cycle; the number of pipe stages per instruction affects latency, not throughput.
3. You cannot make ALU instructions take fewer cycles because of the write-back of the result, but branches and jumps can take fewer cycles, so there is some opportunity for improvement.
4. Instead of trying to make instructions take fewer cycles, we should explore making the pipeline longer, so that instructions take more cycles, but the cycles are shorter. This could improve performance.

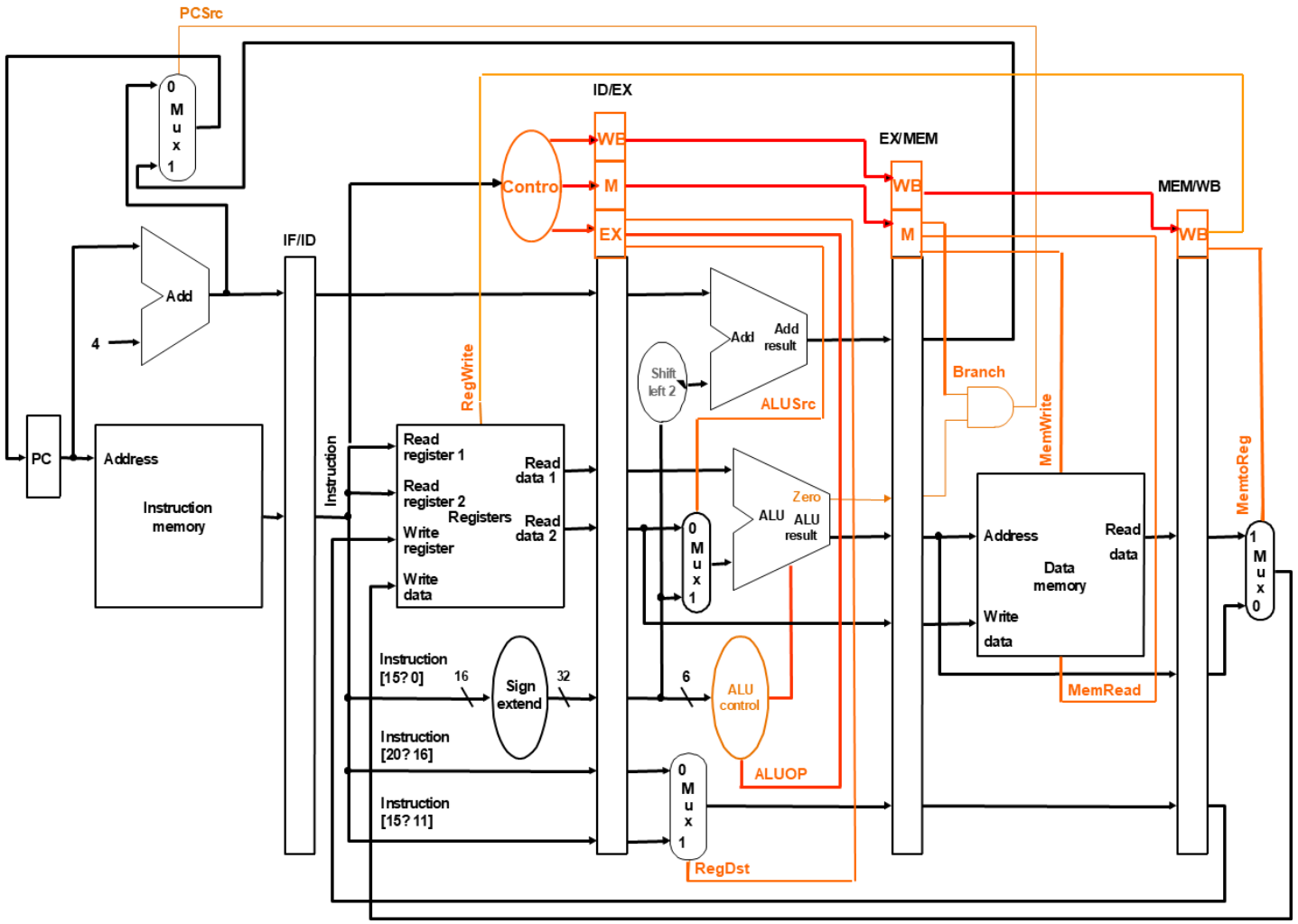
Pipelined Control

- Control signals can be classified by the affecting stages

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-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



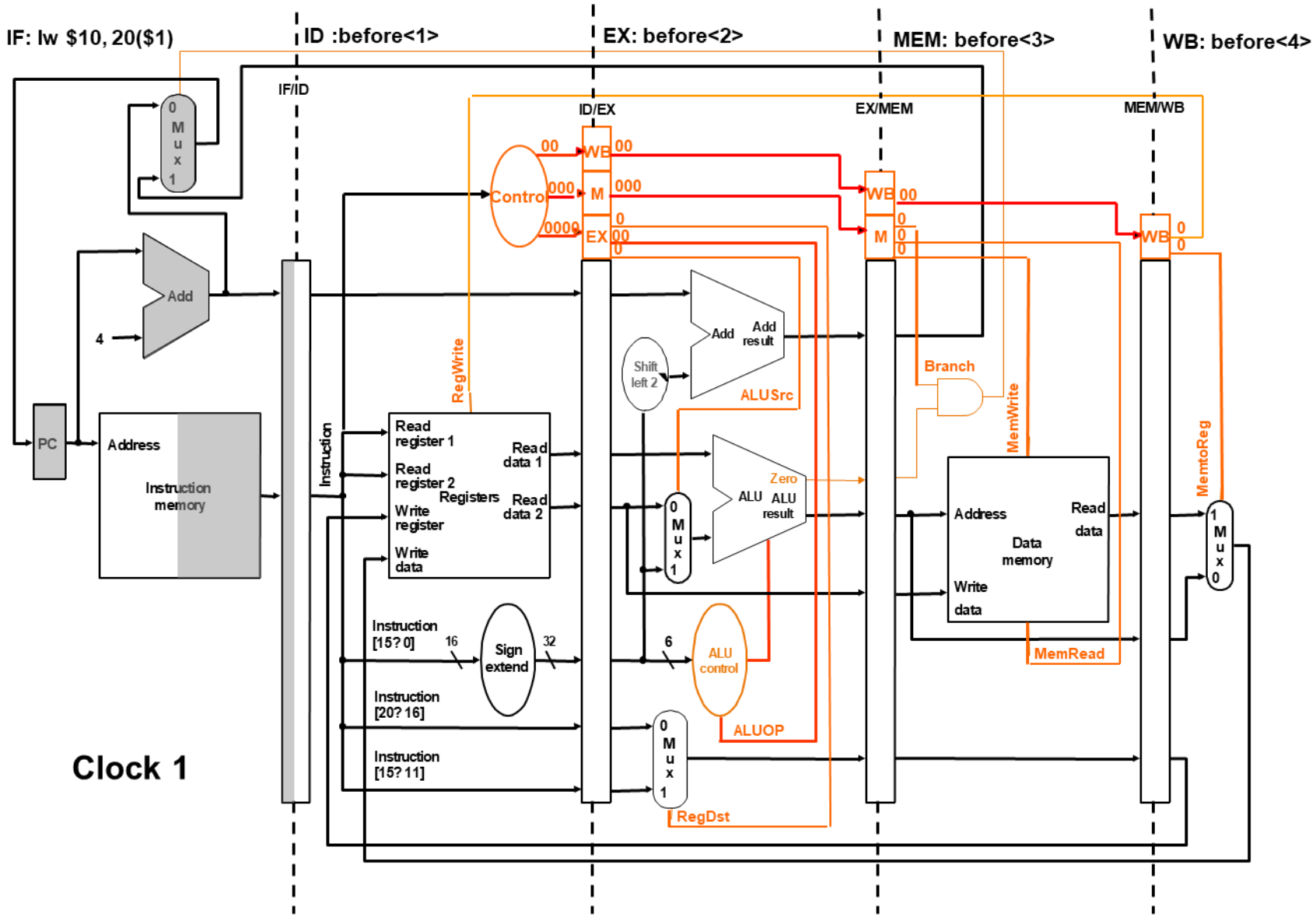
Pipelined Datapath with Control Signal



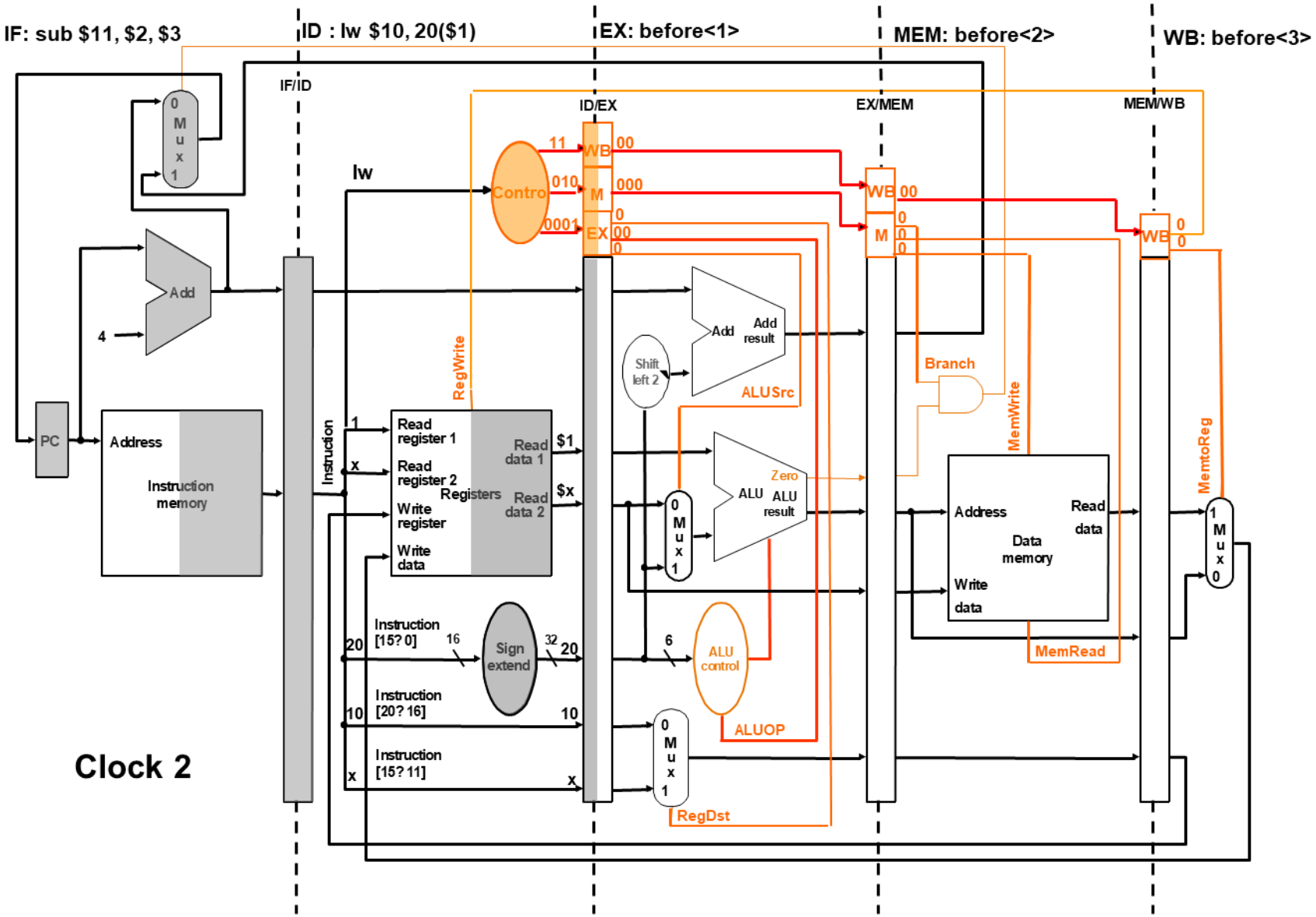
Example

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

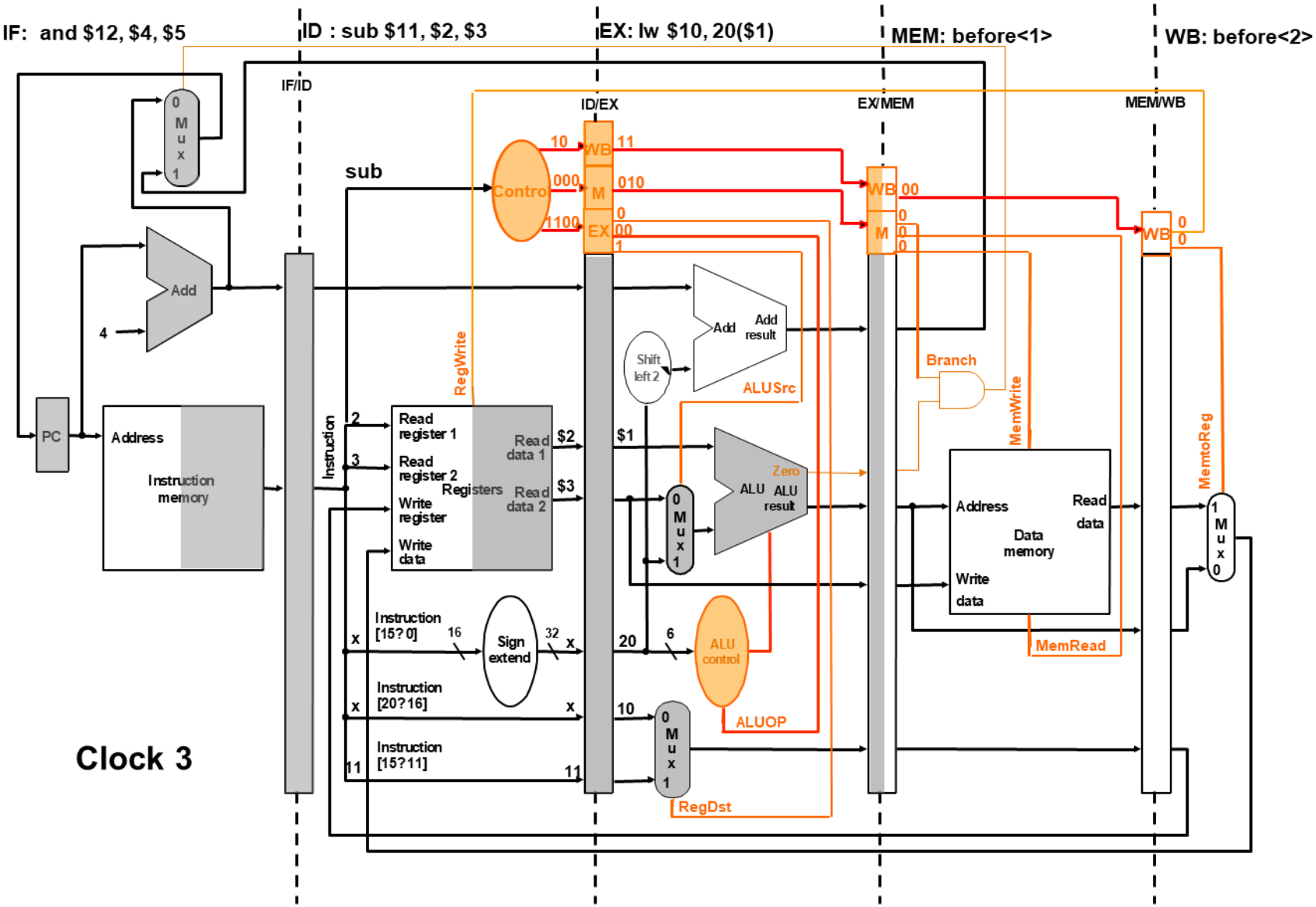
Example


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
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
RegWrite
Control
00
000
0000
WB
M
EX
Add
Add result
Shift left 2
ALUSrc
Zero
ALU
ALU result
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
1 M u x 0
Instruction [15? 0]
16
Sign extend
32
6
ALU control
ALUOP
Instruction [20? 16]
Instruction [15? 11]
RegDst
Clock 1

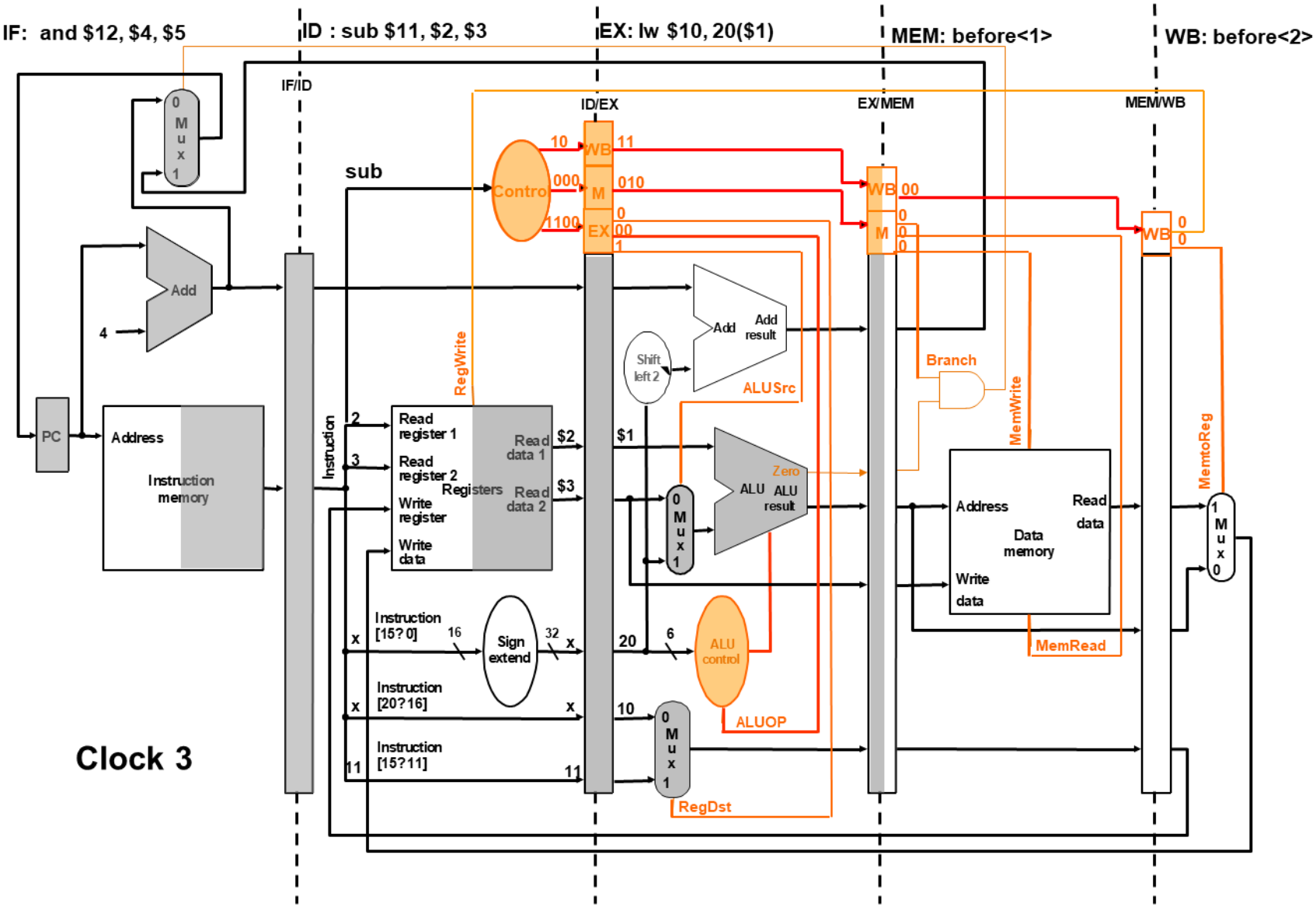
Example

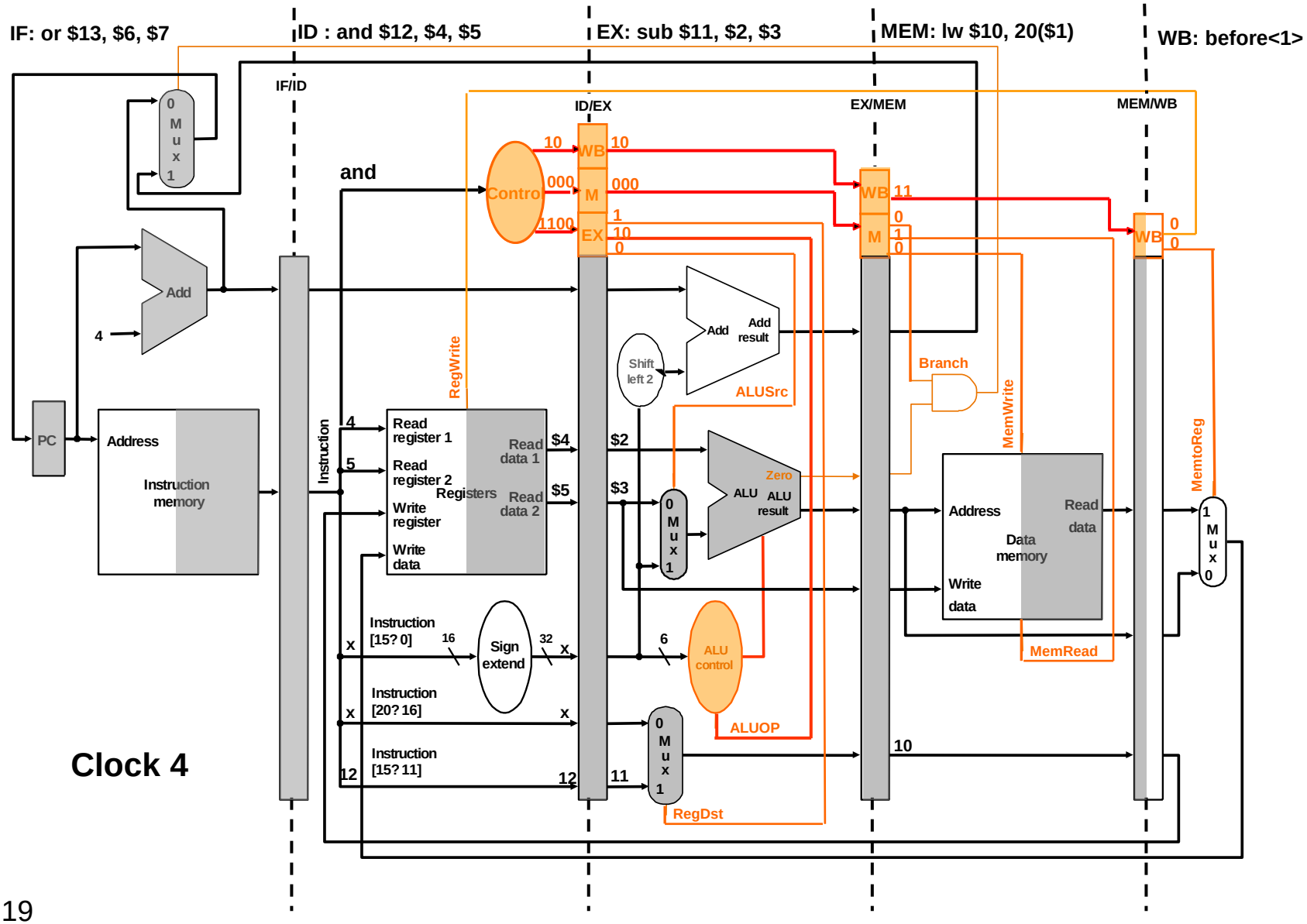

IF: sub $11, $2, $3
ID : lw $10, 20($1)
EX: before<1>
MEM: before<2>
WB: before<3>
IF/ID
ID/EX
EX/MEM
MEM/WB
lw
Control
11
010
0001
WB
M
EX
00
000
0
00
0
Add
4
PC
Address
Instruction memory
Instruction
1
x
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
$1
$x
RegWrite
Shift left 2
Add
Add result
ALUSrc
Branch
MemWrite
MemtoReg
Zero
ALU
ALU result
Address
Data memory
Read data
Write data
MemRead
Instruction [15? 0]
16
Sign extend
32
20
20
6
ALU control
ALUOP
Instruction [20? 16]
10
10
Instruction [15? 11]
x
x
RegDst
0 M u x 1
1 M u x 0
Clock 2

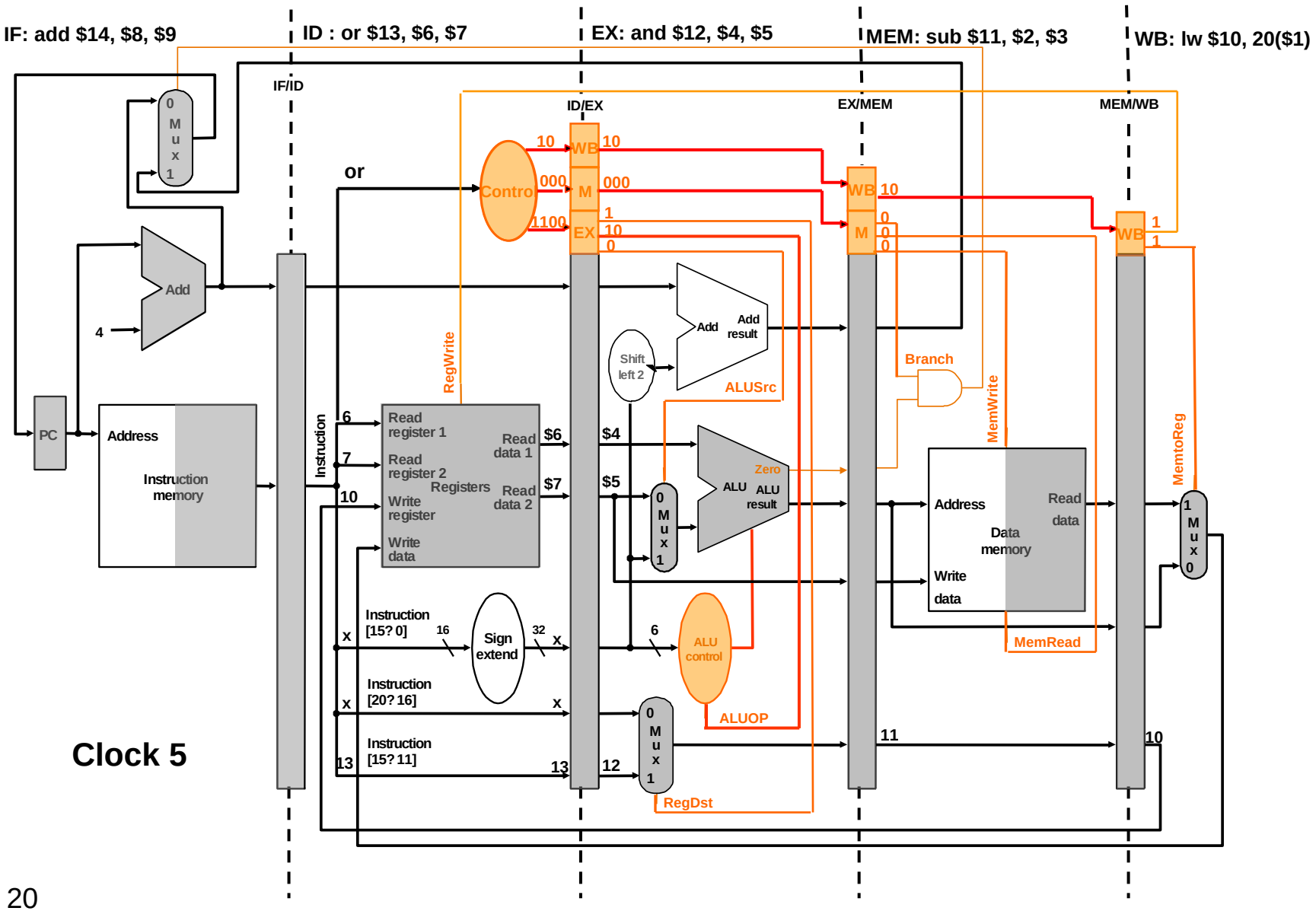
Example

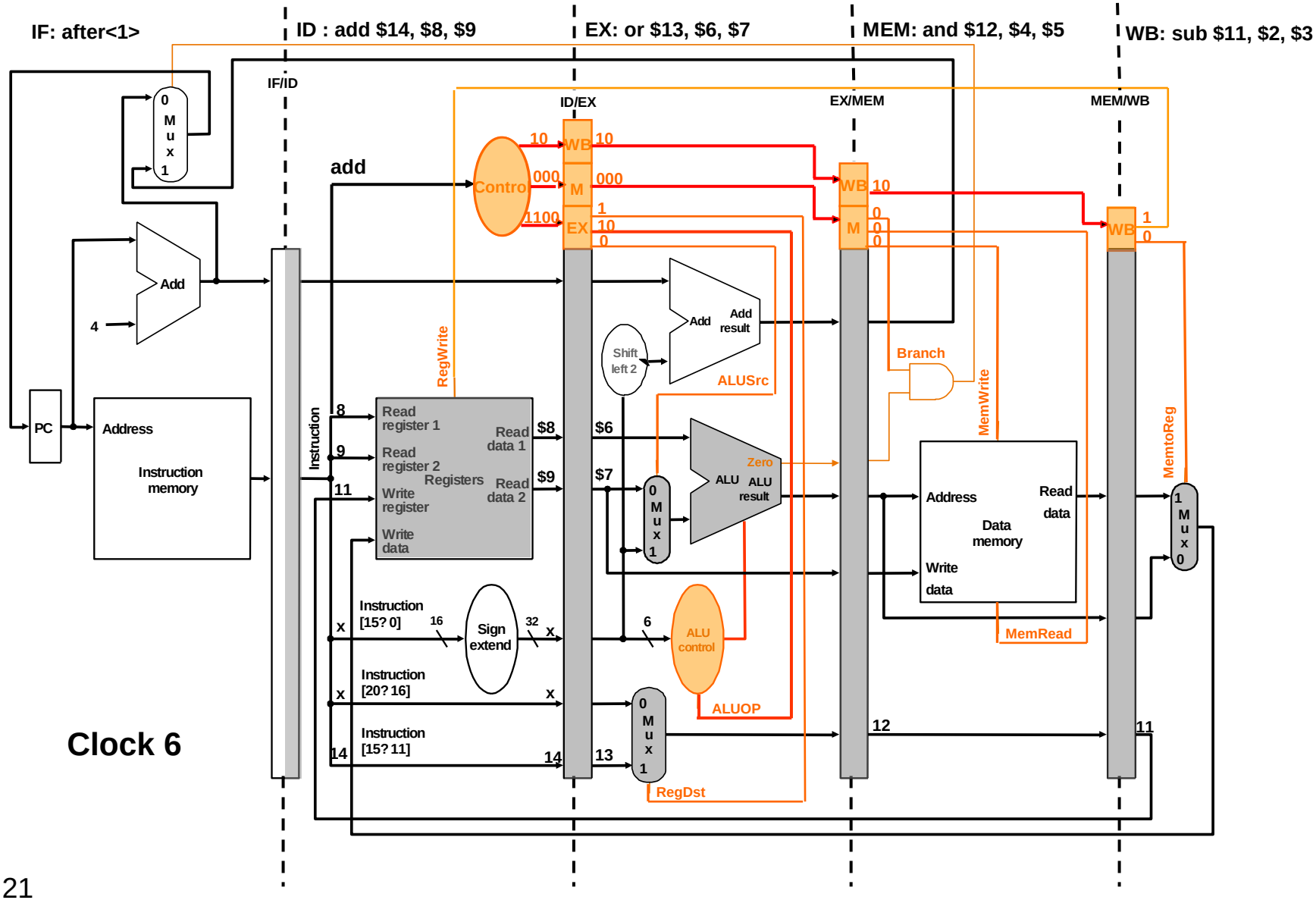

IF: and $12, $4, $5
ID : sub $11, $2, $3
EX: lw $10, 20($1)
MEM: before<1>
WB: before<2>
IF/ID
ID/EX
EX/MEM
MEM/WB
sub
Control
10
000
1100
WB
M
EX
11
010
00
0
1
00
0
0
0
0
0
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Shift left 2
Add
Add result
ALUSrc
Branch
MemWrite
MemtoReg
2
3
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
$2
$3
$1
Zero
ALU
ALU result
Address
Data memory
Read data
Write data
MemRead
Instruction [15?0]
16
Sign extend
32
x
20
6
ALU control
ALUOP
Instruction [20?16]
10
Instruction [15?11]
11
RegDst
Mux
0
1
Clock 3

Example

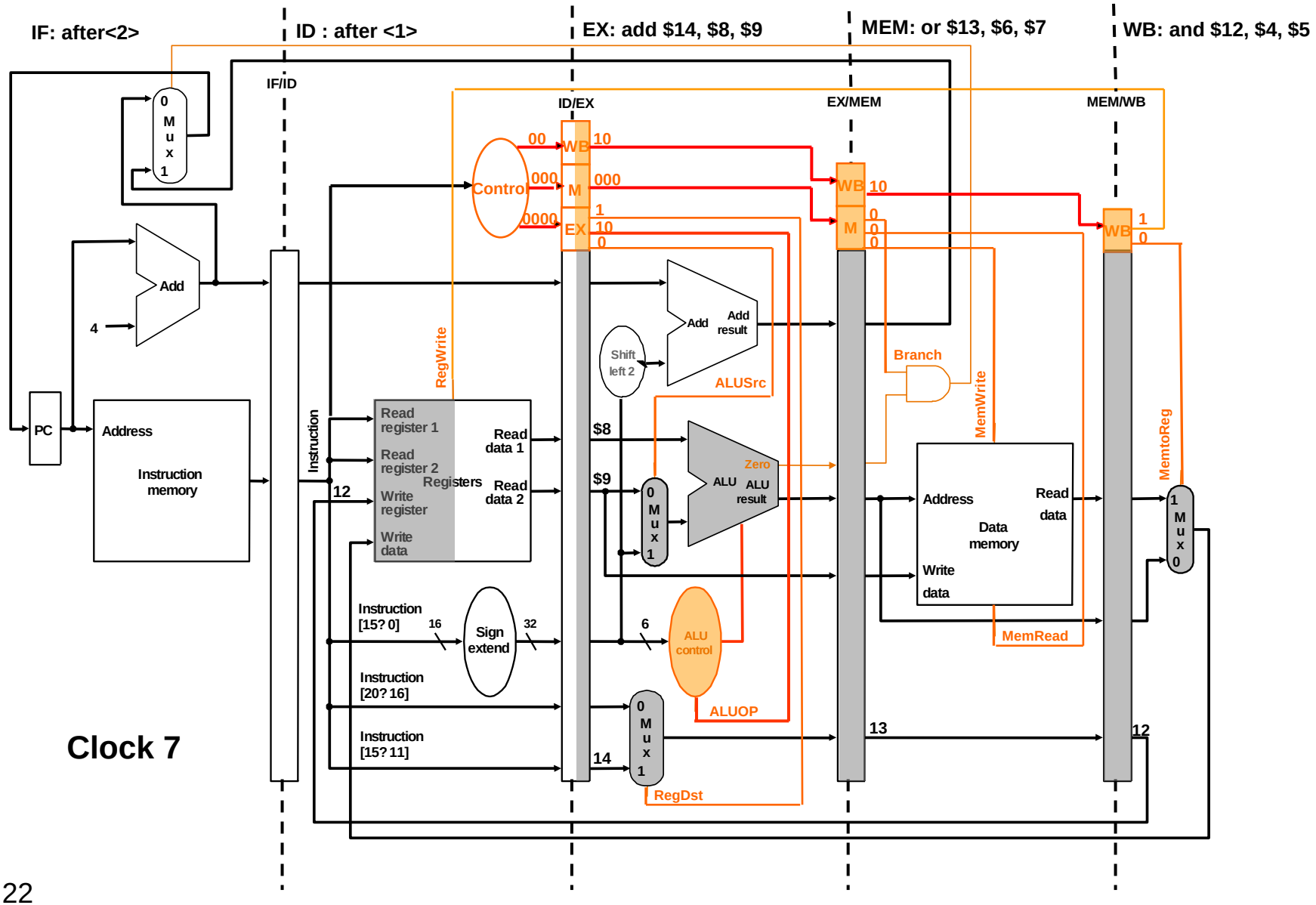

IF: and $12, $4, $5
ID : sub $11, $2, $3
EX: lw $10, 20($1)
MEM: before<1>
WB: before<2>
IF/ID
ID/EX
EX/MEM
MEM/WB
sub
Control
10
000
1100
WB
M
EX
11
010
00
0
1
00
0
0
0
0
0
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Shift left 2
Add
Add result
ALUSrc
Branch
MemWrite
MemtoReg
2
3
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
$2
$3
$1
Zero
ALU
ALU result
Address
Data memory
Read data
Write data
MemRead
Instruction [15?0]
16
Sign extend
32
x
20
6
ALU control
ALUOP
Instruction [20?16]
10
Instruction [15?11]
11
RegDst
Mux
0
1
Clock 3


IF: or $13, $6, $7
ID : and $12, $4, $5
EX: sub $11, $2, $3
MEM: lw $10, 20($1)
WB: before<1>
IF/ID
ID/EX
EX/MEM
MEM/WB
and
Clock 4
PC
Address
Instruction memory
Add
4
0 M u x 1
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
Control
10
000
1100
WB
M
EX
Shift left 2
Add
Add result
ALUSrc
Zero
ALU
ALU result
Branch
MemWrite
MemtoReg
MemRead
Address
Data memory
Read data
Write data
Instruction [15? 0]
Sign extend
16
32
ALU control
6
ALUOP
Instruction [20? 16]
Instruction [15? 11]
12
RegDst
$4
$5
$2
$3
11
1 M u x 0

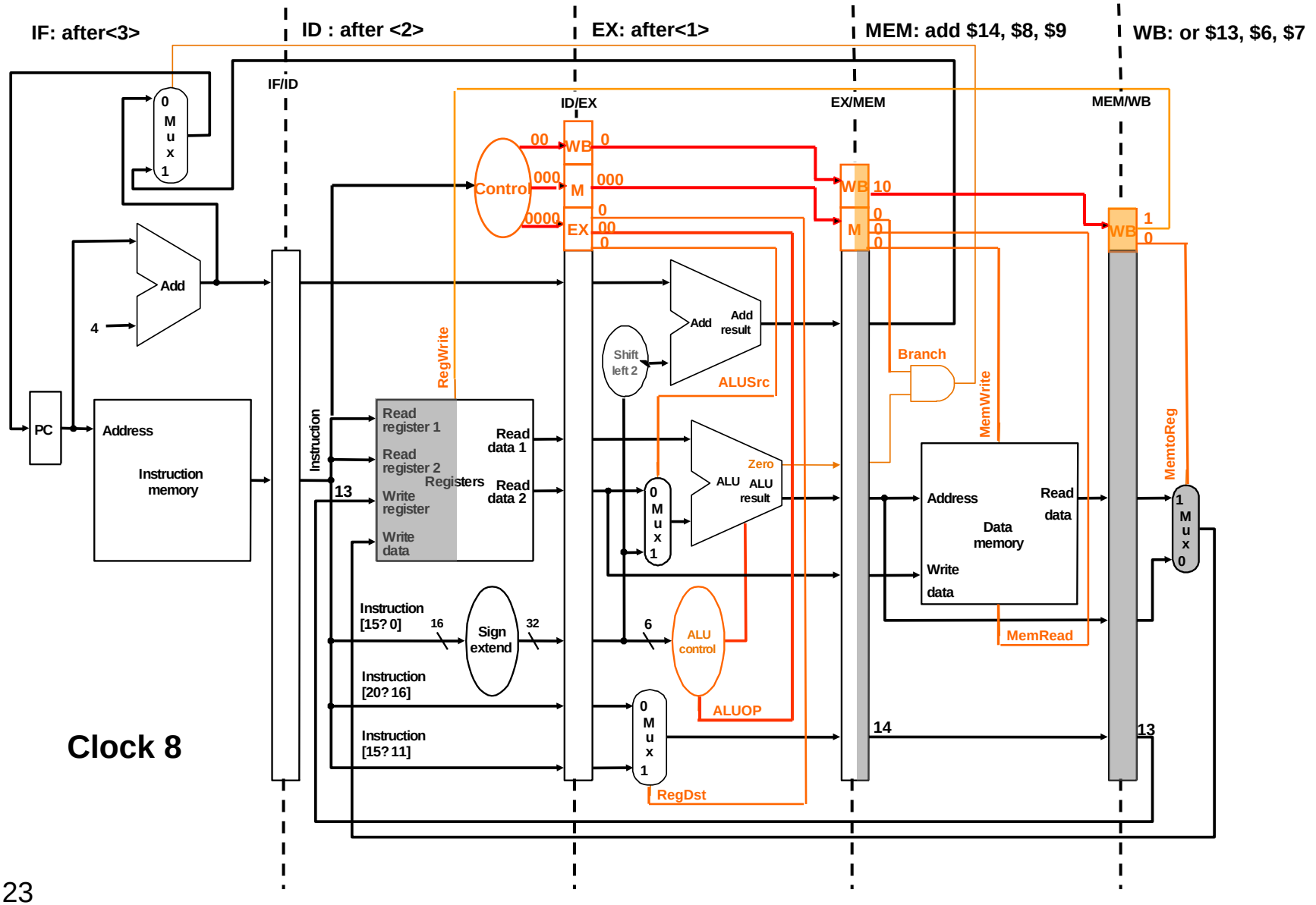

IF: add $14, $8, $9
ID : or $13, $6, $7
EX: and $12, $4, $5
MEM: sub $11, $2, $3
WB: lw $10, 20($1)
IF/ID
ID/EX
EX/MEM
MEM/WB
or
Clock 5
PC
Address
Instruction memory
Instruction
Add
4
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
RegWrite
Control
WB
M
EX
Shift left 2
Add result
ALUSrc
ALU
ALU result
Zero
Branch
MemWrite
MemtoReg
MemRead
Data memory
Read data
ALU control
ALUOP
RegDst
Sign extend
Instruction [15? 0]
Instruction [20? 16]
Instruction [15? 11]
Mux

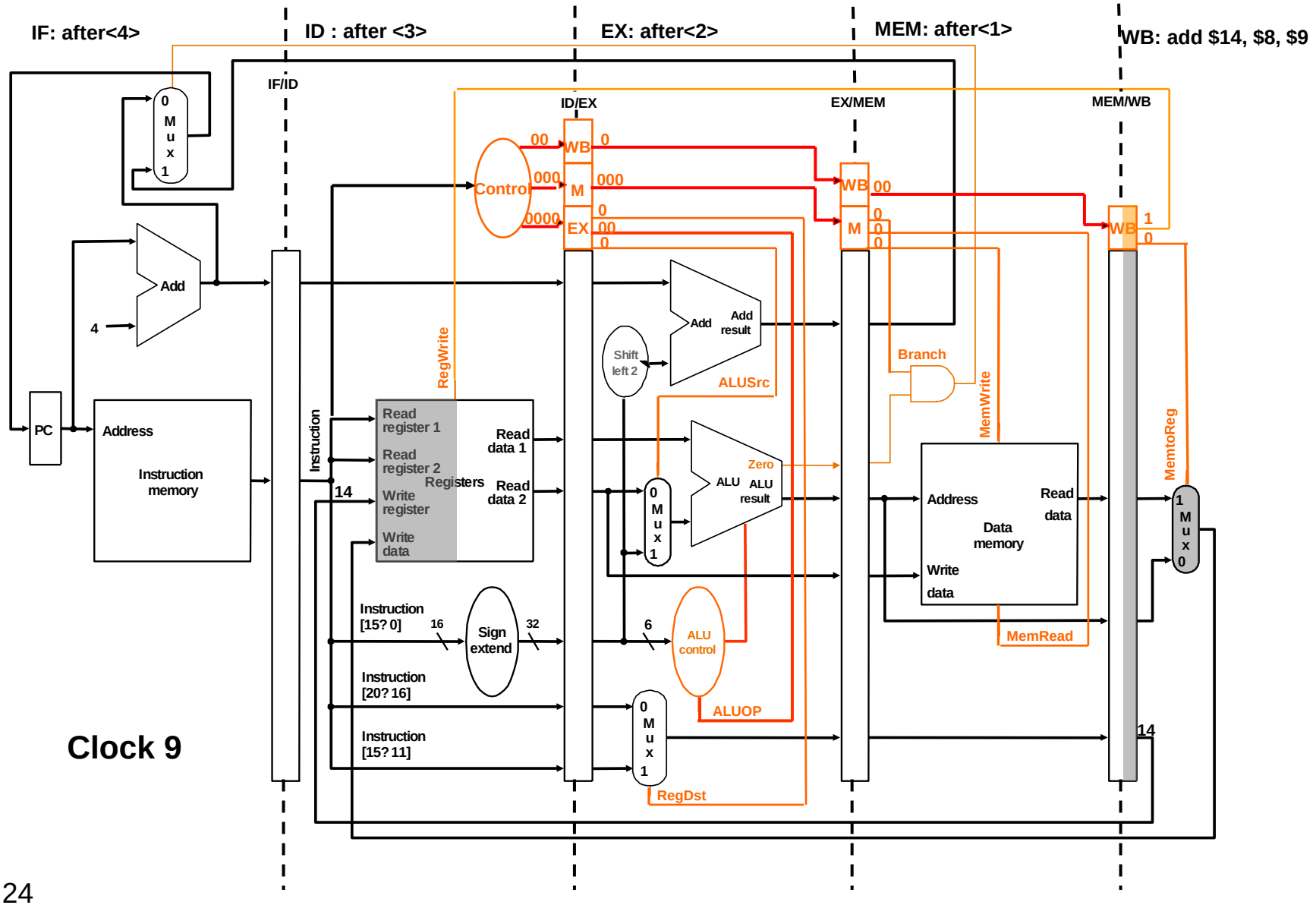

IF: after<1>
ID : add $14, $8, $9
EX: or $13, $6, $7
MEM: and $12, $4, $5
WB: sub $11, $2, $3
IF/ID
ID/EX
EX/MEM
MEM/WB
add
Control
WB
M
EX
10
000
1100
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
8
9
11
$8
$9
$6
$7
Shift left 2
Add result
ALUSrc
Branch
MemWrite
MemtoReg
Zero
ALU
ALU result
Address
Data memory
Read data
Write data
MemRead
Instruction [15? 0]
Sign extend
16
32
ALU control
6
ALUOP
Instruction [20? 16]
Instruction [15? 11]
14
13
12
11
RegDst
Mux
0
1

Clock 6


IF: after<2>
ID : after <1>
EX: add $14, $8, $9
MEM: or $13, $6, $7
WB: and $12, $4, $5
IF/ID
ID/EX
EX/MEM
MEM/WB
0 M u x 1
Add
4
PC
Address
Instruction memory
Instruction
12
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
Control
00
000
0000
WB
M
EX
10
000
1
10
0
$8
$9
Add
Add result
Shift left 2
ALUSrc
Zero
ALU
ALU result
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
Instruction [15? 0]
16
Sign extend
32
6
ALU control
ALUOP
Instruction [20? 16]
Instruction [15? 11]
14
RegDst
13
12
1 M u x 0

Clock 7


IF: after<3>
ID : after <2>
EX: after<1>
MEM: add $14, $8, $9
WB: or $13, $6, $7
IF/ID
ID/EX
EX/MEM
MEM/WB
0 M u x 1
Add
4
PC
Address
Instruction memory
Instruction
13
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Control
00
000
0000
WB
M
EX
0
00
0
Add
Add result
Shift left 2
ALUSrc
Zero
ALU
ALU result
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
Instruction [15? 0]
16
Sign extend
32
6
ALU control
ALUOP
Instruction [20? 16]
Instruction [15? 11]
RegDst
14
13
10
1
Clock 8


IF: after<4>
ID : after <3>
EX: after<2>
MEM: after<1>
WB: add $14, $8, $9
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 9
PC
Address
Instruction memory
Add
4
Instruction
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
RegWrite
Control
WB
M
EX
Shift left 2
Add result
ALUSrc
ALU
ALU result
Zero
Branch
MemWrite
MemtoReg
MemRead
Data memory
Read data
Write data
Instruction [15? 0]
Sign extend
Instruction [20? 16]
Instruction [15? 11]
ALU control
ALUOP
RegDst
Mux

Hazard

The situation when the next instruction cannot be executed in the following clock cycle.

1.Structural Hazard : Hardware cannot support combination of instruction.

ex) The instruction and data should be fetched from the memory at the same time.

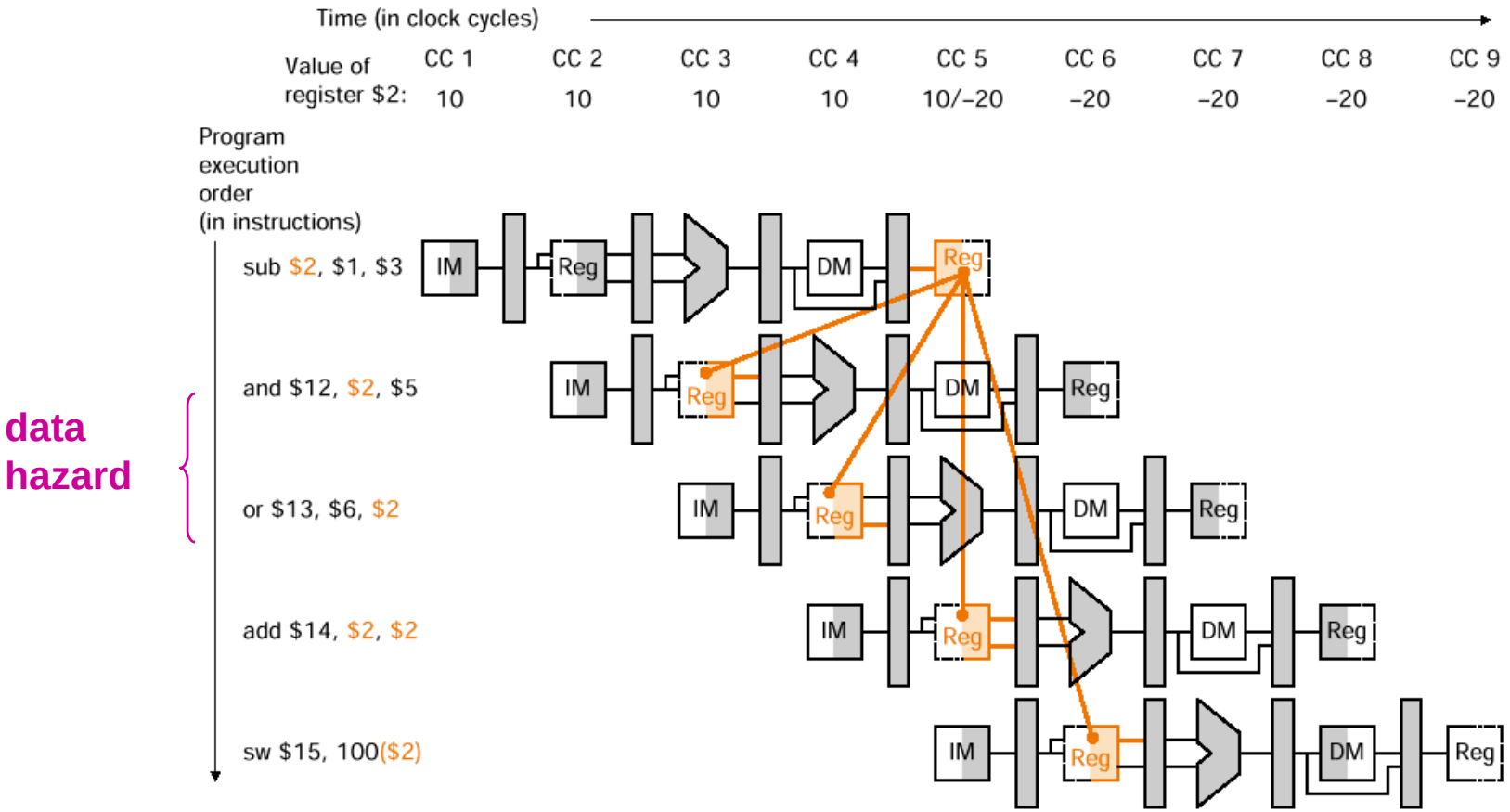
2. Data Hazard : An instruction depends on the results of a previous instruction still in pipeline.

ex) add $s0, $t0, $t1

sub $t2, $s0, $t3

Dependencies

- Problem with starting next instruction before first is finished
  - dependencies that go backward in time are data hazards



Hazard

3. Control Hazard : Control may be transferred to another instruction based on the results of an instruction

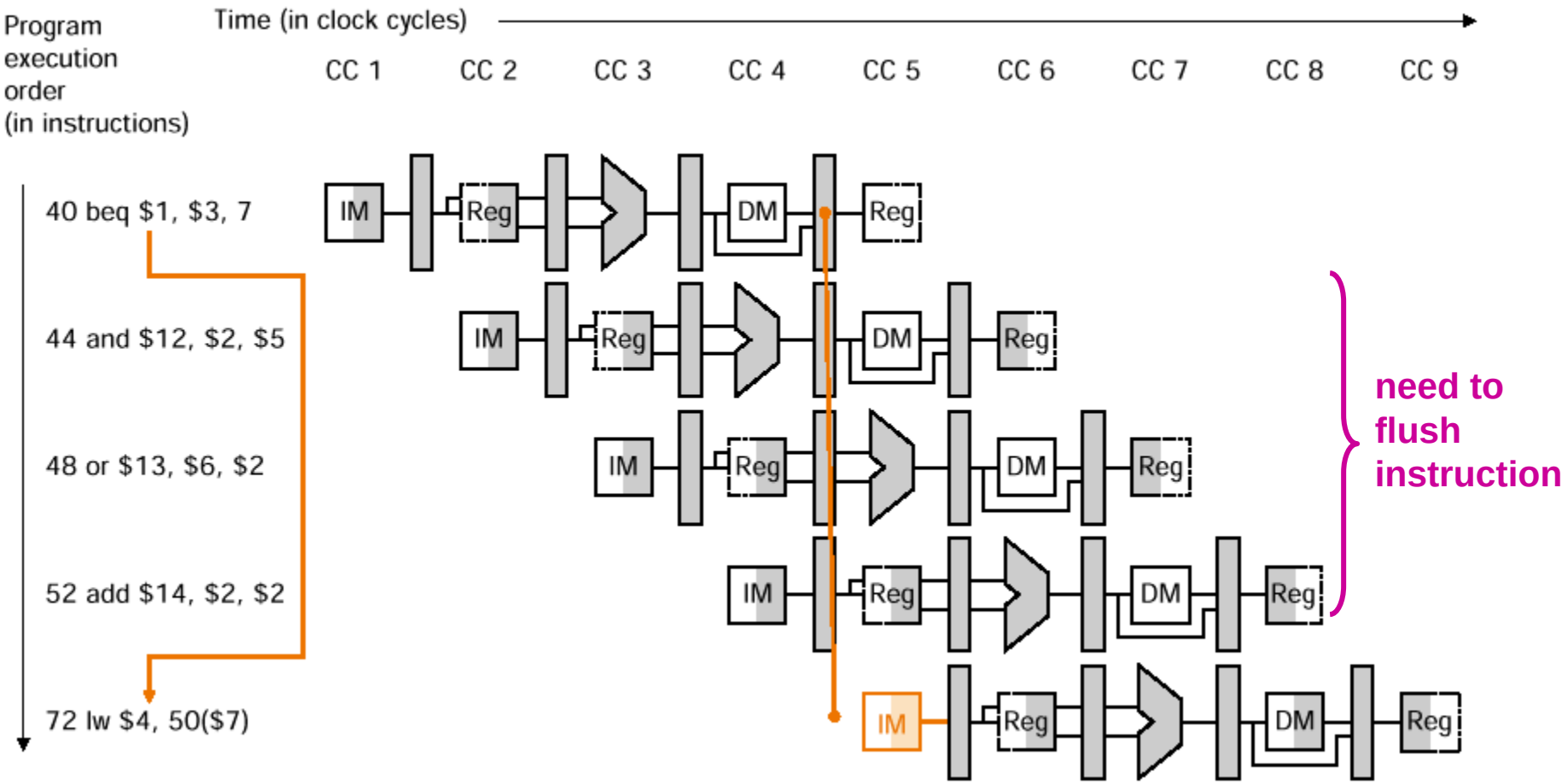
```
Ex)     beq  $t0, $t1, L1
        add  . . . . . .
        sub  . . . . . .
     L1 and  . . . . . .
```

→All the Hazards can be resolved by waiting

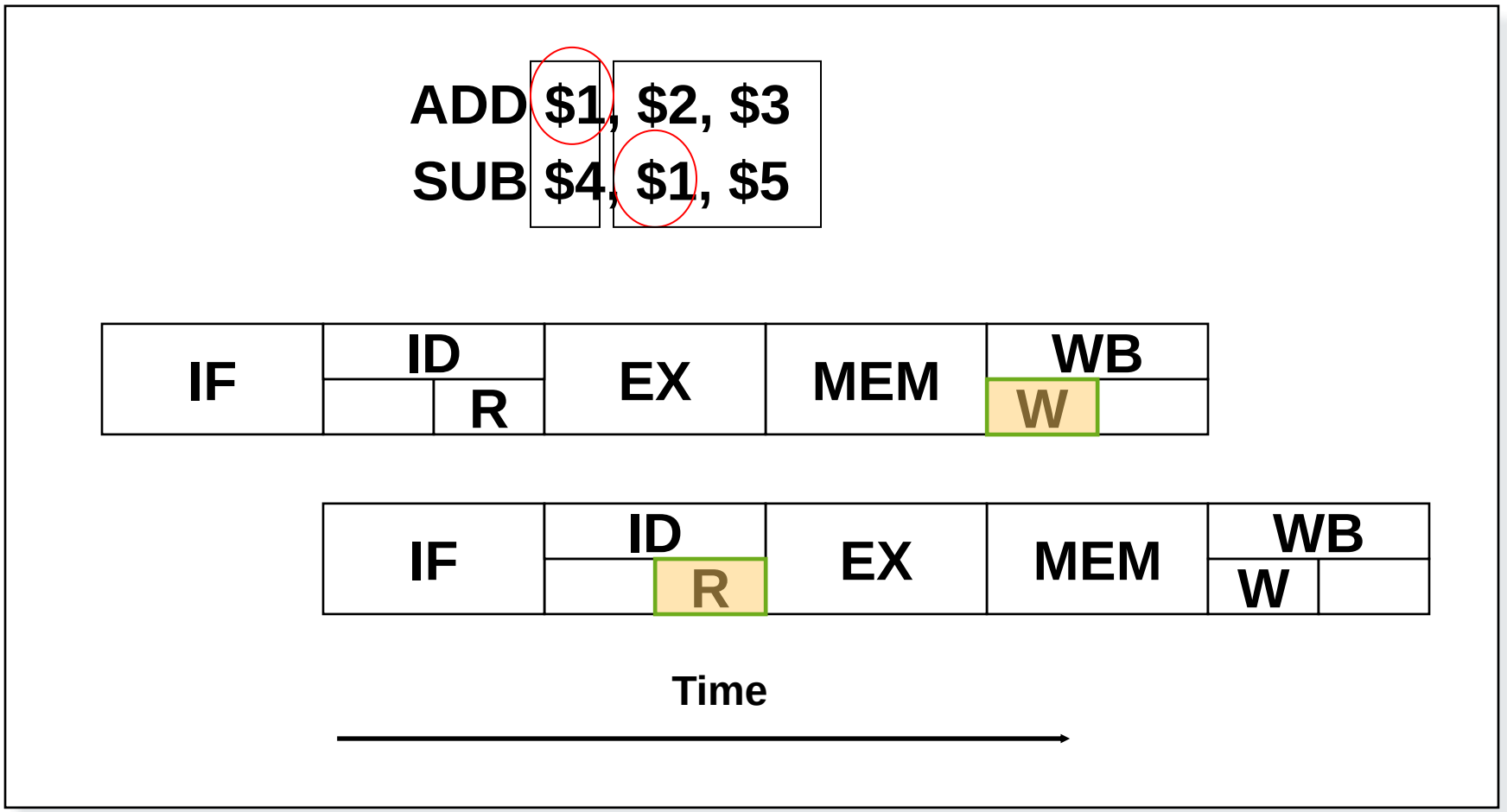
- pipeline control must detect the hazard
- take action (or delay action) to resolve hazards

Control (Branch) Hazard Example

- **When we decide to branch, other instructions are in the pipeline!**
- **We are predicting branch not taken**
  - **need to add hardware for flushing instructions if we are wrong**



Data hazard (RAW hazard)



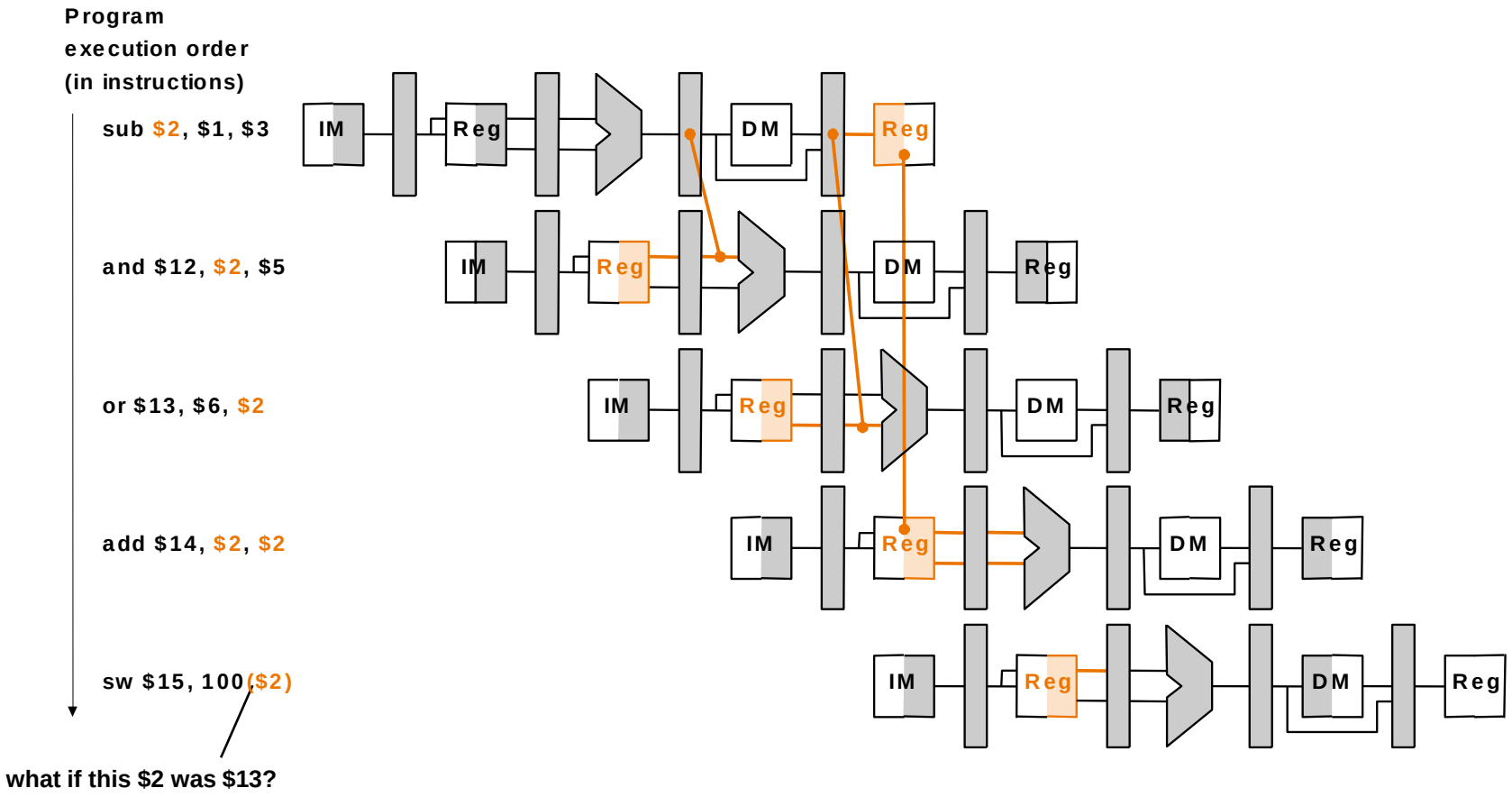
Solutions to Data Hazard

- Freezing the pipeline
- (Internal) Forwarding
- Compiler scheduling

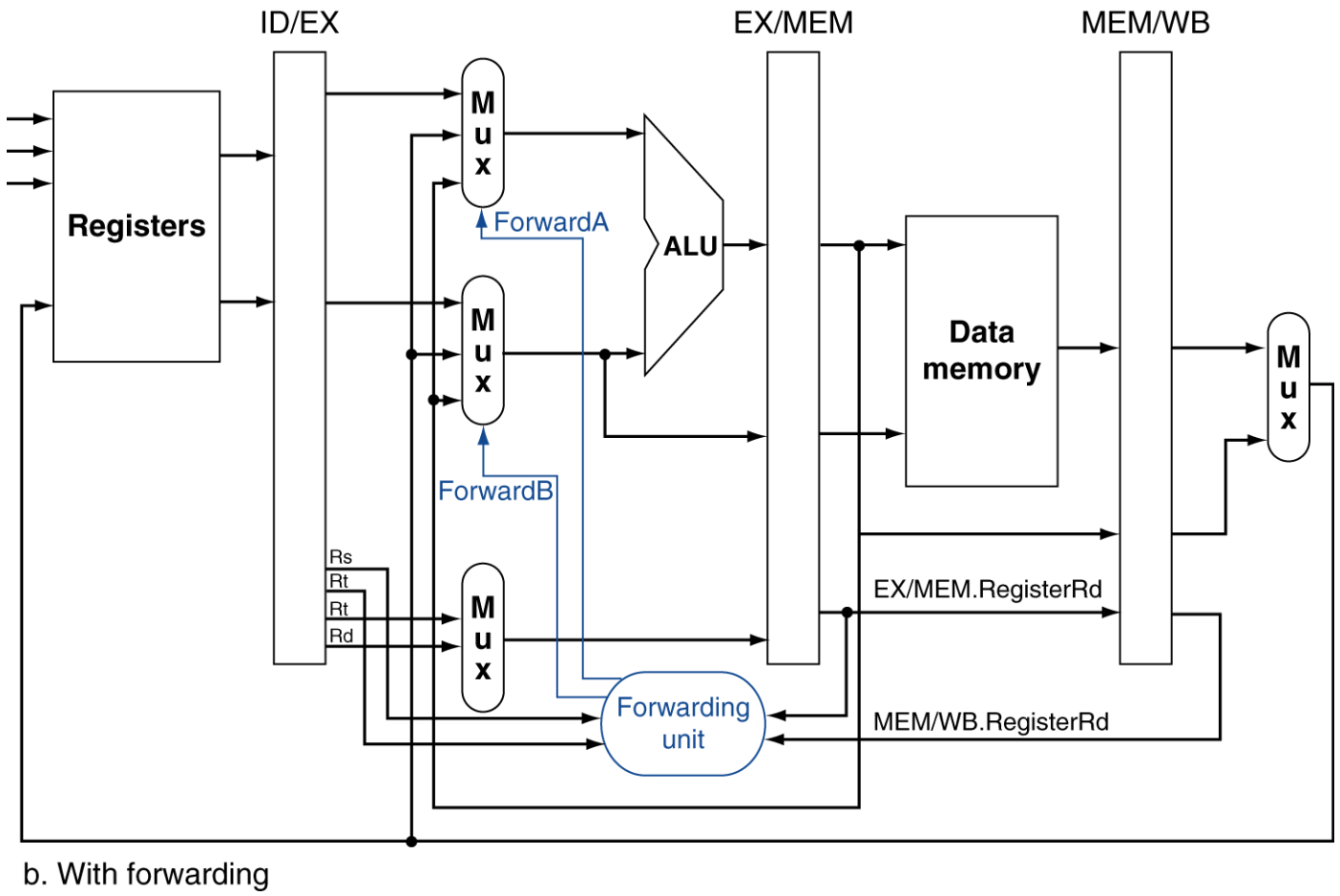
Forwarding

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 Paths



b. With forwarding

EX hazard

```
sub   $2, $1,$3      # Register $2 written by sub
and   $12,$2,$5      # 1st operand($2) depends on sub
```

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

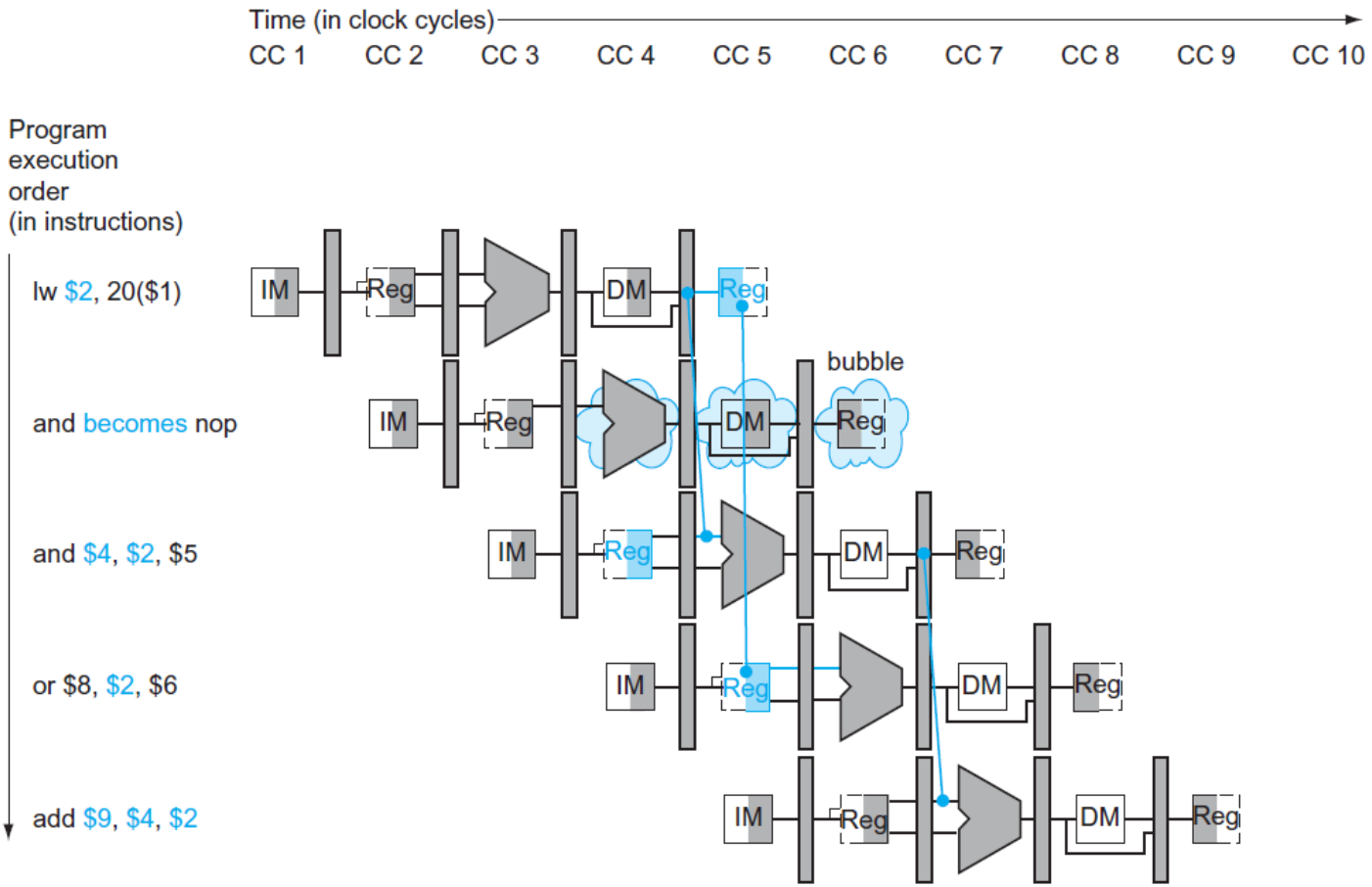
```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠  0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

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

MEM Hazard

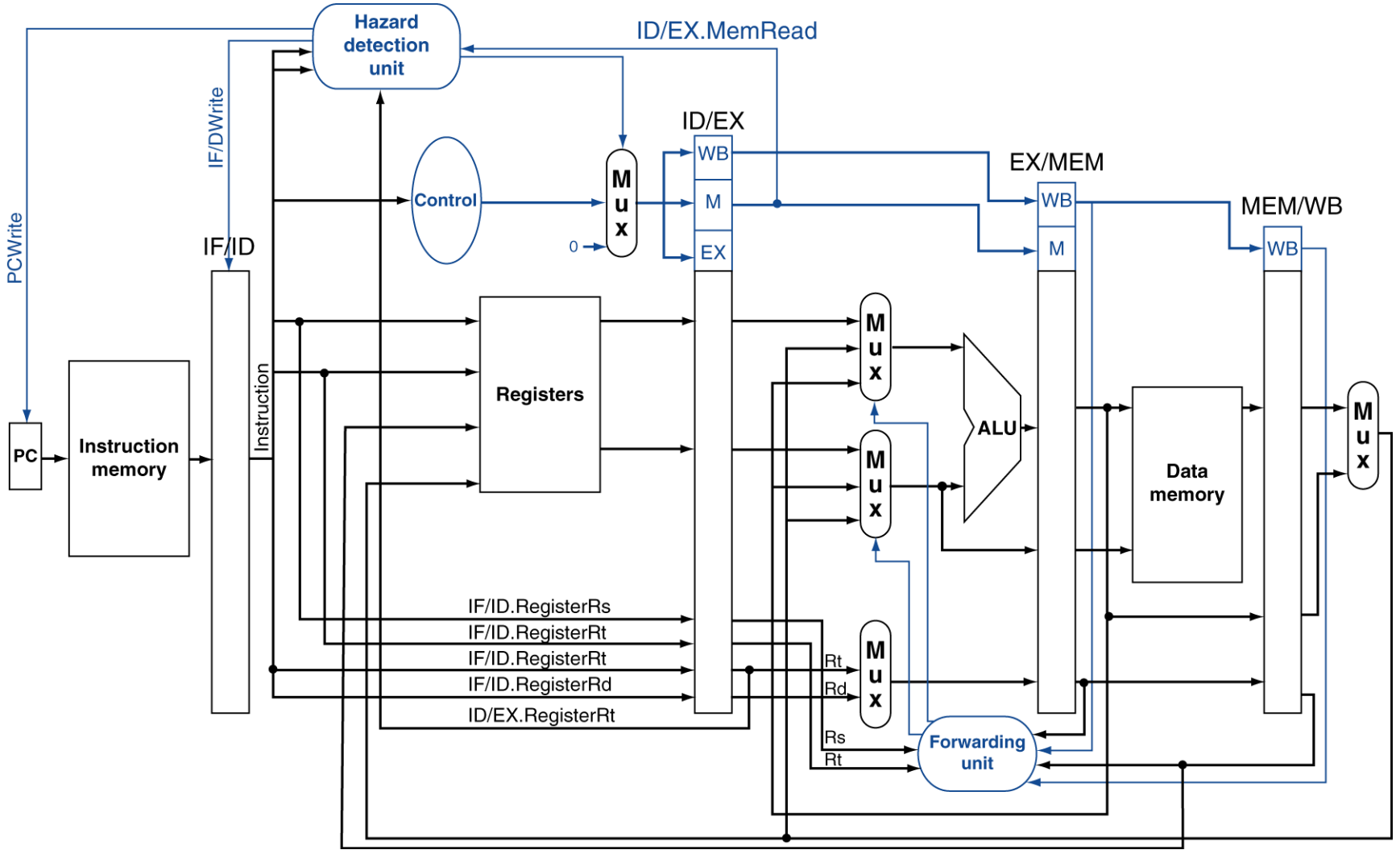
```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠  0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠  0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01


if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠  0)
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠  0)
       and (EX/MEM.RegisterRd ≠  ID/EX.RegisterRs))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

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

Stalls

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



Datapath with Hazard Detection



Data Hazard : Compiler scheduling

ADD $1, $2, $3
SUB $4, $1, $5
lw $6, 100($7)
AND $8, $8, $10

- Add ‘nop’ instruction

  ADD $1, $2, $3
  NOP
  NOP
  SUB $4, $1, $5

- Or rearrange the order of the instruction

  ADD $1, $2, $3
  lw $6, 100($7)
  AND $8, $8, $10
  SUB $4, $1, $5

Solutions to Control Hazard

- Stall
- Optimized branch processing
- Branch prediction
- Delayed branch

Control Hazard : Stall

- Caused by PC-changing instructions

  (Branch, Jump, Call/Return)

Branch instruction	**IF**	**ID**	**EX**	**MEM**	**WB**				
Branch successor		**IF**	**stall**	**stall**	**IF**	**ID**	**EX**	**MEM**	**WB**
Branch successor + 1						**IF**	**ID**	**EX**	**MEM**	**WB**
Branch successor + 2							**IF**	**ID**	**EX**	**MEM**
Branch successor + 3								**IF**	**ID**	**EX**
Branch successor + 4									**IF**	**ID**
Branch successor + 5										**IF**

For 5-stage pipeline, 3 cycle penalty
15% branch frequency. CPI = 1 + 0.15 * 3 = 1.45

Control Hazard : Optimized Branch Processing

1. Find out branch taken or not *early*

   → simplified branch condition

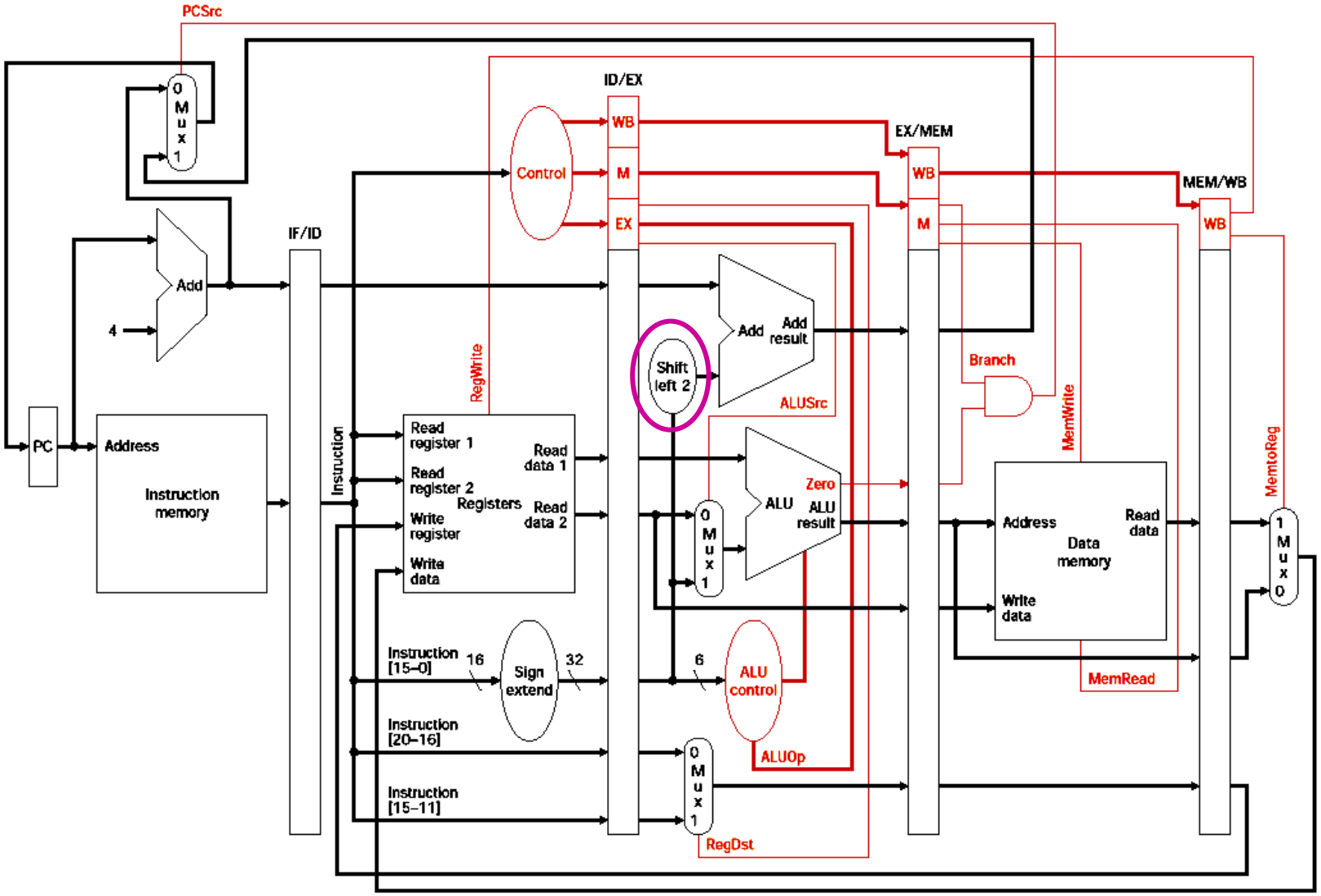
2. Compute branch target address *early*

   → extra hardware

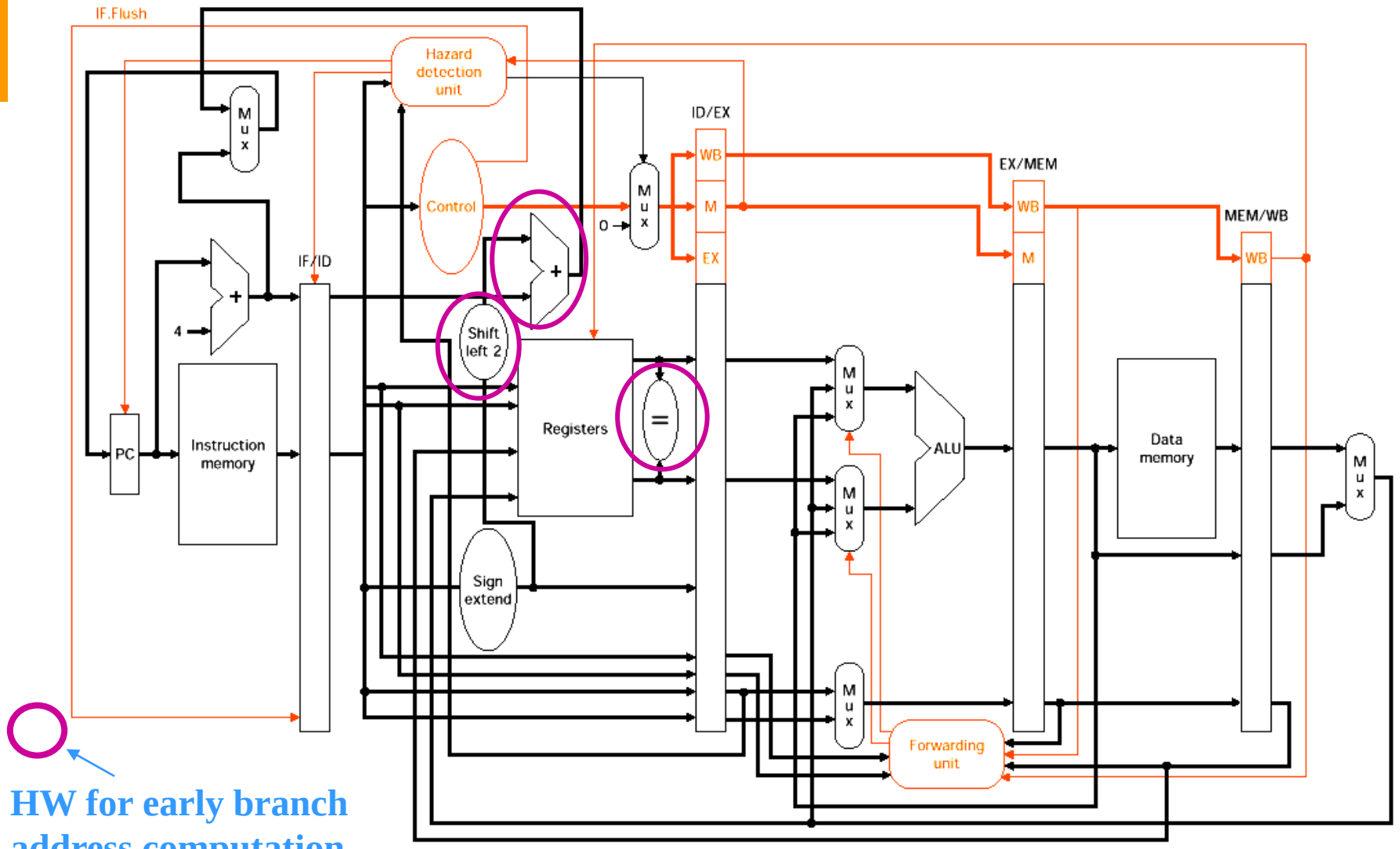
Control Hazard : Optimized Branch Processing

- Reducing Delay by moving Branch execution to ID
  - There is only one instruction to flush if the branch is taken (the currently being fetched instruction)
  - Move up the Branch Address Calculation
    - duplicate adder
  - Moving up the Branch Test itself (use XORs and OR for equality test)
  - IF.Flush control line : clearing IF/ID registers transforms the fetch instruction into nop (instruction that does no operation to change state)

Control Hazard : Optimized Branch Processing



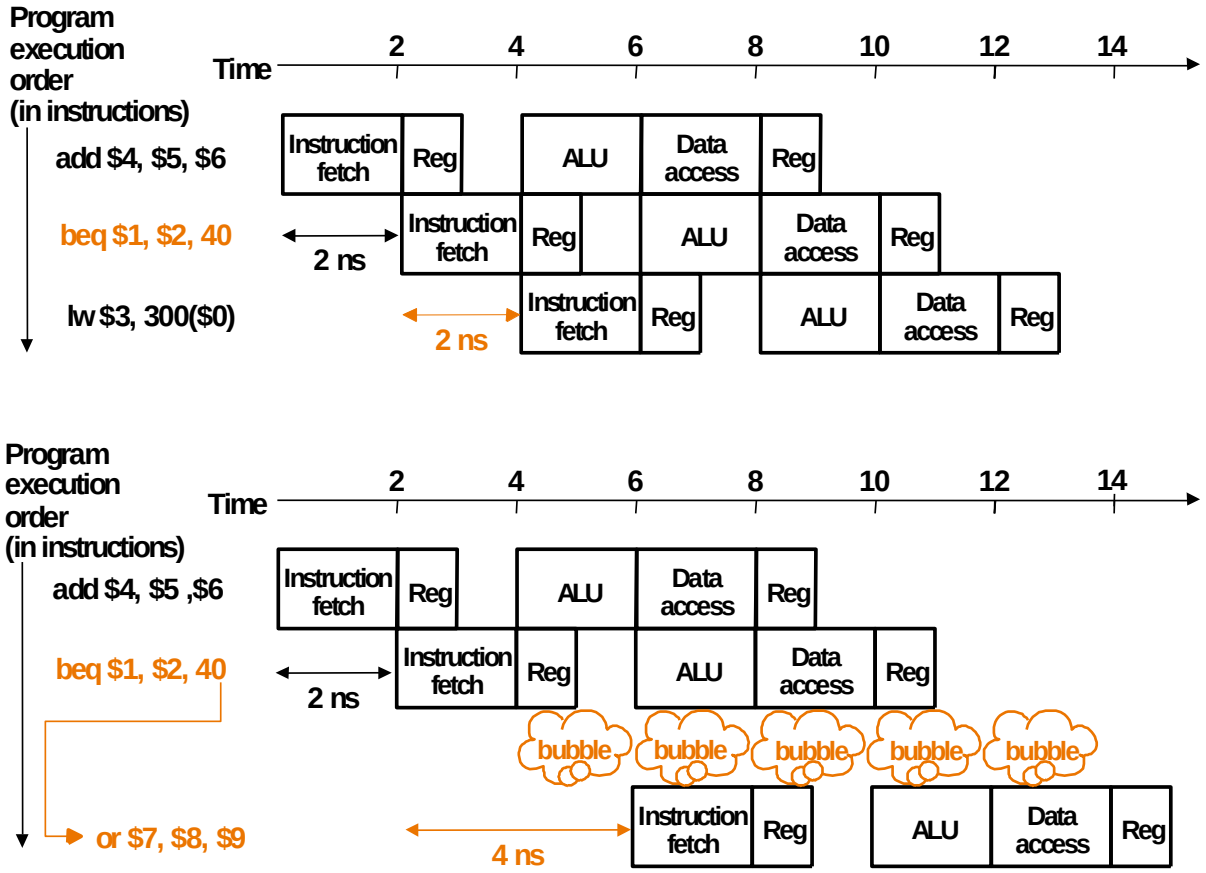
Control Hazard : Optimized Branch Processing



HW for early branch address computation

Control Hazard : Branch Prediction

- One simple approach is to always predict branch will fail.
- If the prediction is wrong insert a bubble. : flush the instruction in the pipe and fetch the instruction at branch target address.



Control Hazard : Branch Prediction

- Predict: guess one direction then back up if wrong
- Predict-not-taken

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction i + 1		IF	ID	EX	MEM	WB			
Instruction i + 2			IF	ID	EX	MEM	WB		
Instruction i + 3				IF	ID	EX	MEM	WB	
Instruction i + 4					IF	ID	EX	MEM	WB

Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction i + 1		IF	Idle	Idle	idle	Idle			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target +2					IF	ID	EX	MEM	WB

Static Branch Prediction

(1)Never branch – Assume the branch never take place.

(2)Always branch – Assume the branch always take place.

(3)Predict by op-code – Decision is made based on the opcode.

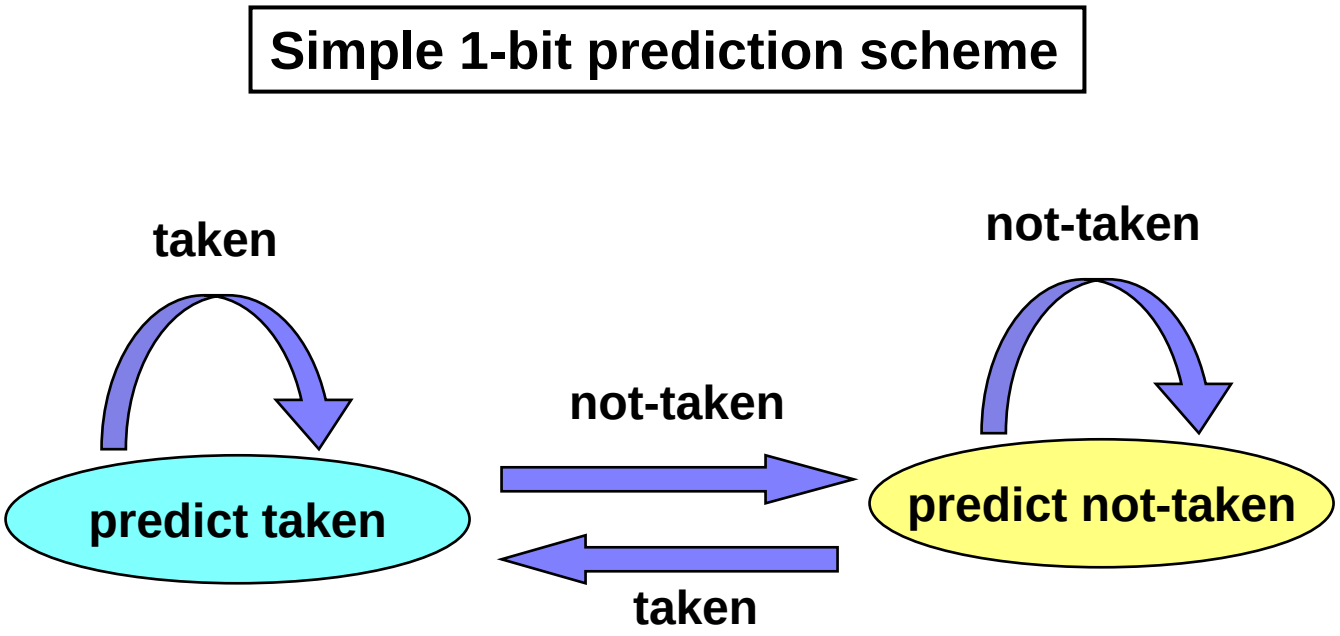
ex) beq ~ never branch

bne ~ always branch

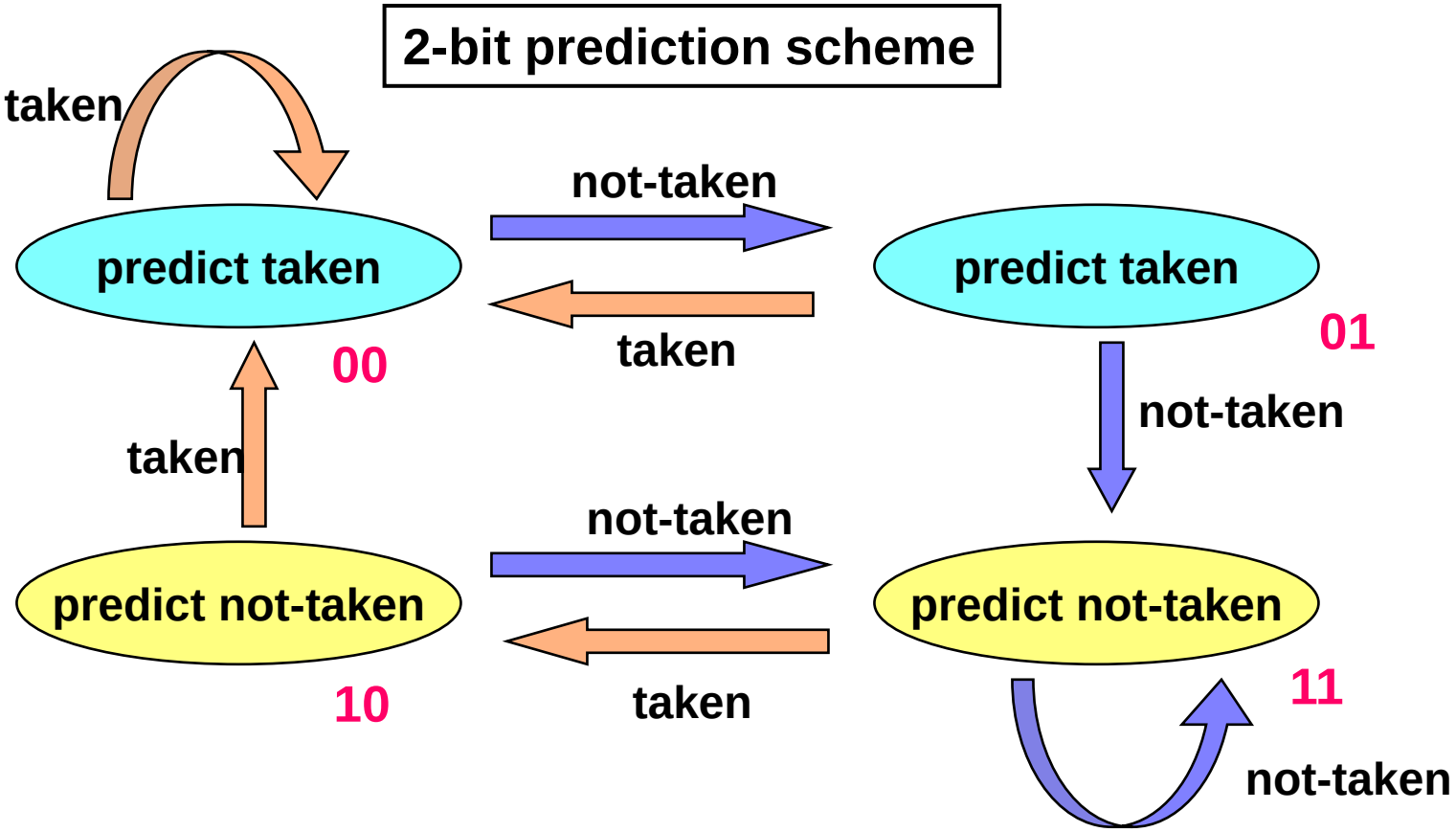
Dynamic Branch Prediction

- Use a Branch Prediction Buffer or Branch History Table
  - The memory contains a bit saying the branch was recently taken or not.
- Prediction is just a Hint assumed to be correct. Fetch begins in the predicted direction. If Hint turns out to be wrong, the prediction bits are changed.
- If an instruction is predicted to be taken, fetching begins from the target as soon as the PC is known. (It can be as early as ID stage)

Dynamic Branch Prediction



Dynamic Branch Prediction



A Prediction must be Wrong Twice before it is changed.

Exceptions and Interrupts)

- events other than branches or jumps that change the normal flow of instruction executions
  - to handle unexpected events from within the processor (exceptions), or external I/O devices (interrupt)

Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

Exceptions

- Two cases
  - Undefined instruction
  - Arithmetic overflow

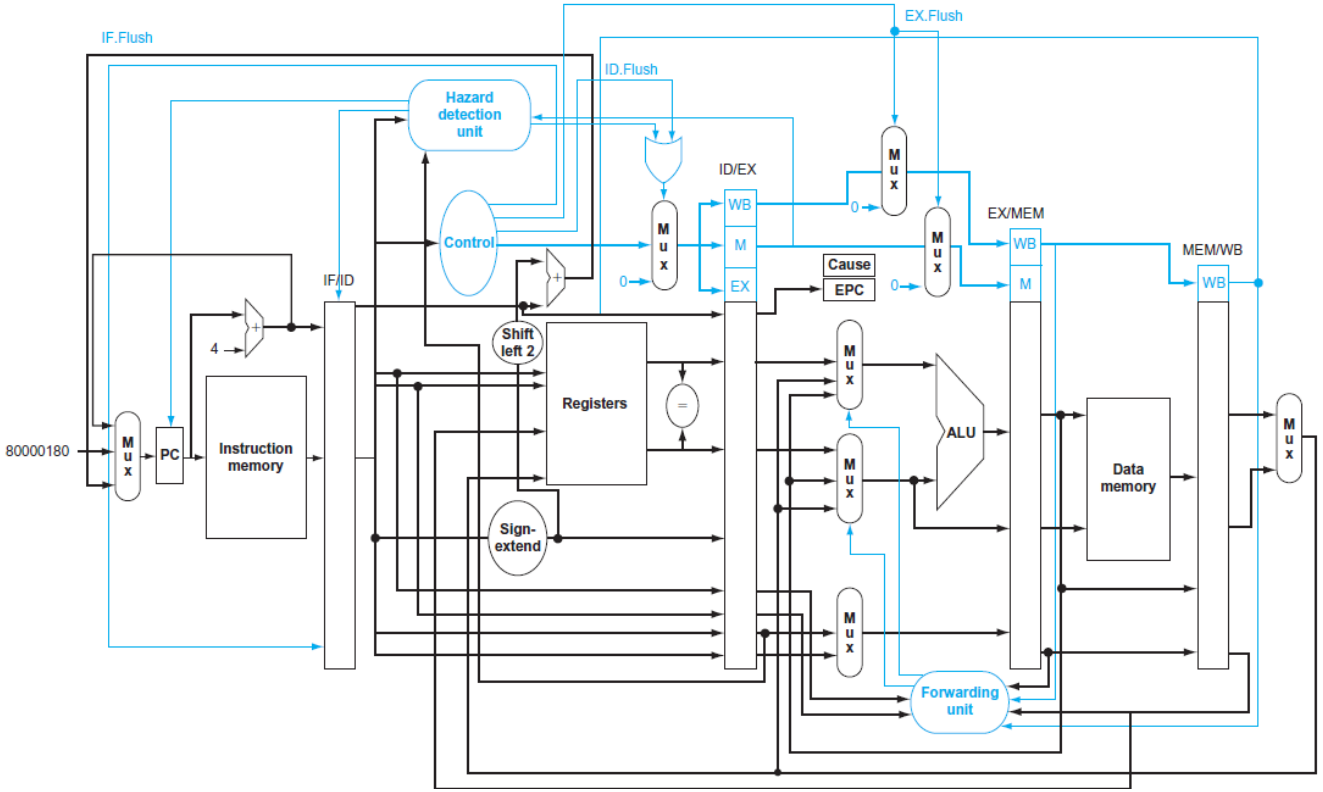
- Response
  - Save the address of the offending instruction in the Exception Program Counter (EPC)
  - Transfer control to the operating system at a specified address

Exception Handling

- How to identify the cause
  - via status register (Cause register)
    - MIPS32
    - Five-bit fields encodes the two possible source of exceptions
  - via vectored interrupt
    - Different interrupt handlers are executed depending on the cause

Exception in Pipeline Implementation

- An exception is treated as another form of control hazard
  - An overflow exception is detected in the EX stage
  - Prevent the instruction in EX from writing its result in WB, and flush the ID and IF stages.



RISC vs. CISC

- Instruction set design determines the way that machine language programs are constructed. It also determines structure of CPU.
- Early computer

  - small and simple instruction set

    (to minimize the hardware & lack of technology)

  ➔ Digital hardware became cheaper (VLSI technology)

  ➔ CISC (Complex Instruction Set Computer)
- In the early 80's, new concept has emerged.

  : Fewer instructions without having to use memory as often

  RISC (Reduced Instruction Set Computer)

CISC

- The essential goal of a CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high-level language.

  - to simplify the compilation & improve the overall computer performance

CISC

- Major characteristics of CISC

1. A large number of instructions (typically 100 to 250 instructions)
2. A large variety of addressing modes (5 to 20)

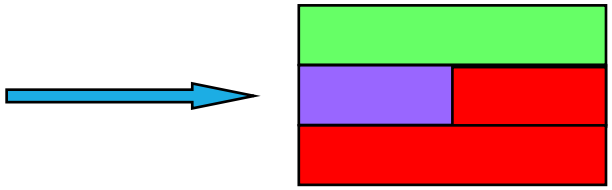
➔ 1 & 2 cause more complex hardware

➔ cause the computation to slow down

3. Variable-length instruction formats

ex) short register addressing mode inst.

& long direct addressing mode instruction.



➔ the instruction may not be aligned

➔ may need special decoding circuits

4. Instruction that manipulate operands in memory

RISC

- Reduce execution time by simplifying the instruction set
- Major characteristics of RISC
  1. Relatively fewer instructions
  2. Relatively fewer addressing mode
  3. Memory access limited to load and store instruction
     - Large number of registers in CPU
     - Overlapped register window to speed up procedure call & return

RISC

Overlapped Register Windows

- Subroutine call is a time-consuming job due to stack operation

(We need to save register values & return address and restore them when subroutine terminates.)

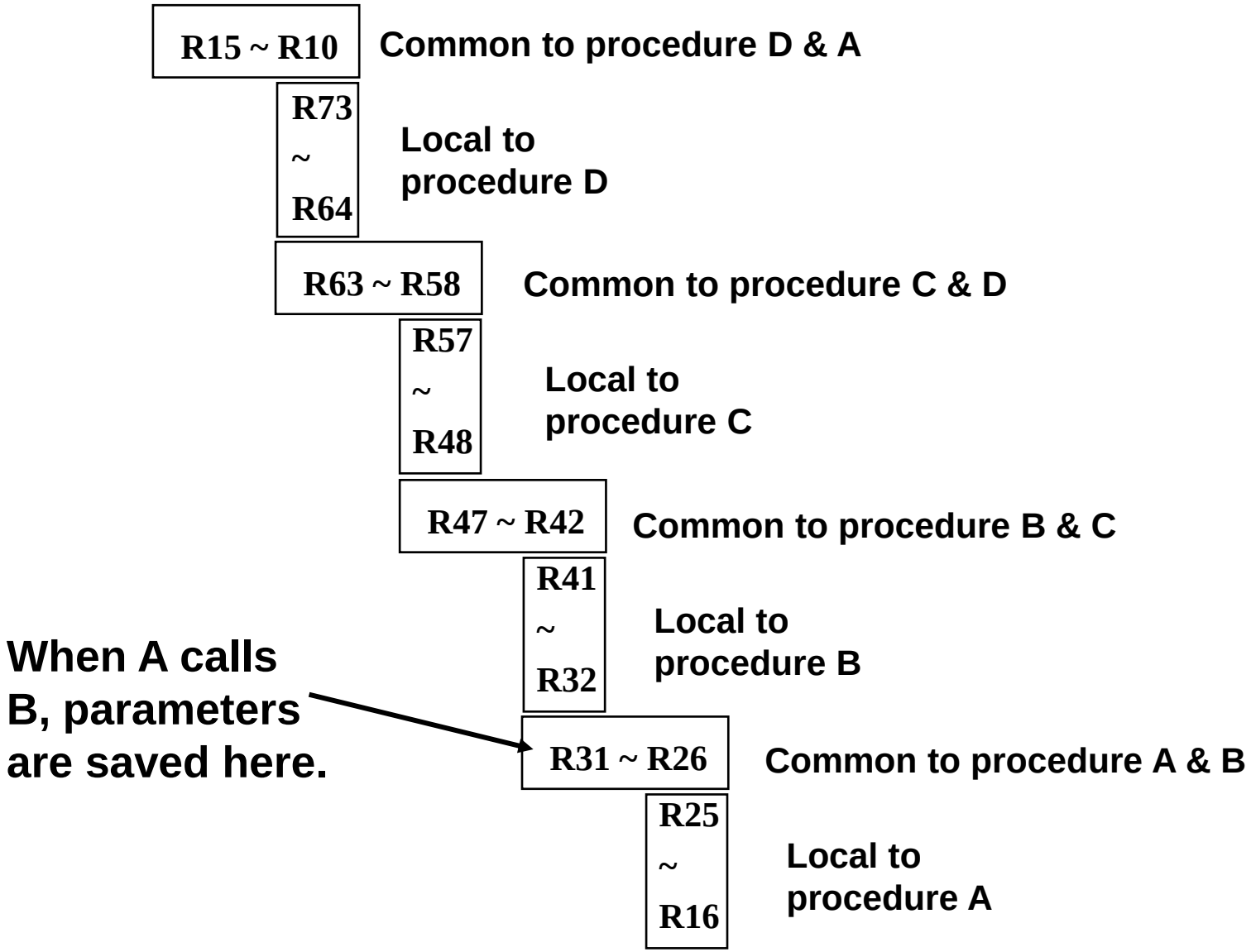
➔ use overlapped register windows

ex) Suppose the computer has 74 registers

R0 ~ R9 : common to all procedures

64 registers are divided into 4 windows

RISC


R15 ~ R10
Common to procedure D & A
R73 ~ R64
Local to procedure D
R63 ~ R58
Common to procedure C & D
R57 ~ R48
Local to procedure C
R47 ~ R42
Common to procedure B & C
R41 ~ R32
Local to procedure B
When A calls B, parameters are saved here.
R31 ~ R26
Common to procedure A & B
R25 ~ R16
Local to procedure A

RISC

4. Most operations are done within registers

   ➔ Faster execution time

   ➔ Almost all instructions have simple register addressing mode.

5. Fixed-length, easily decoded instruction format

   ➔ Aligned on word boundaries

   ➔ Simple control logic

6. Hardwired rather than micro-programmed control : faster

7. One instruction per clock cycle

   ➔ Due to pipelining