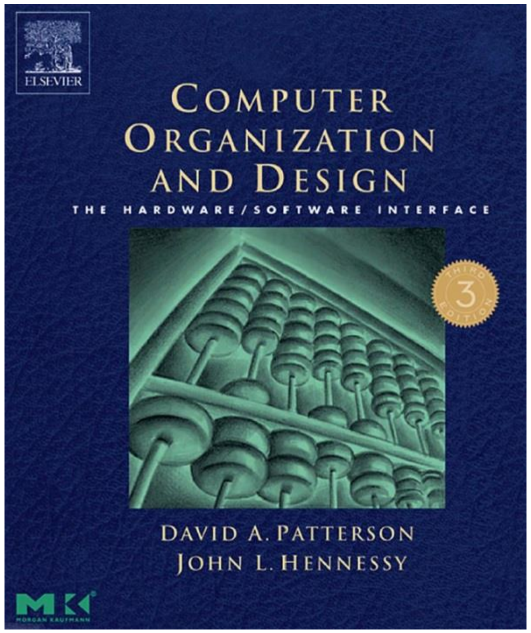
MIPS Overview

prepared and instructed by
Shmuel Wimer
Eng. Faculty, Bar-Ilan University



Architecture

- We consider the simplest MIPS **Instruction Set Architecture (ISA)** introduced by standard undergrad courses.
- 32-bit byte addressing, namely, memory space is 2^{32} bytes, or 2^{30} words (4 bytes).
- Access is always aligned to address (mod 4) = 0.
- **Register File (RF)** comprises 32 registers of 32 bits.
- 32 bit ISA, independent of microarchitecture (single cycle, multi cycle, pipelined).

Arithmetic Instructions (R-type, ALU)

C code: **a = b + c**

compiler

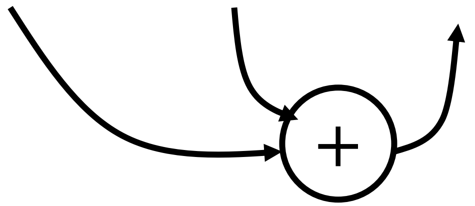
Assembly form: **add $t0, $s1, $s2**

General form: **add rd, rs, rt : rd = rs + rt**

assembler

Machine instruction:

000000	10001	10010	01000	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



Memory-Register Instructions (I-type)

Load from Mem to Reg \ Store from Reg to Mem

Assembly form: **lw rt, offset(rs)**

General form: **rt = mem[rs + offset]**

Example: **lw $t0, 32($s3)**

assembler

Machine instruction:

100011	10011	01000	0000000000100000
op	**rs**	**rt**	**immediate value / address offset**
6 bits	5 bits	5 bits	16 bits

Also for ALU operations with immediate values

Branch Instructions

Controls program flow, conditions, loops, etc.

Jumps to memory offset from **program counter (PC)**

C++ code: **if (x1 == x2) cout << "…"; else cout << "…";**

compiler

Assembly form: **bne $s1, $s2, Label_1**

assembler

Machine instruction:

offset from PC

op	s1	s2	offset
6 bits	5 bits	5 bits	16 bits

Unconditional Jumps

Controls program flow, jumps to other code of functions, subroutine etc.

Jumps to target address.

FORTRAN code: **GO TO n**

compiler

Assembly form: **J Target**

assembler

Machine instruction:

op	**jump target address**
6 bits	26 bits

Target address: **PC[31:28] || instruction[25:0] || ’00’**

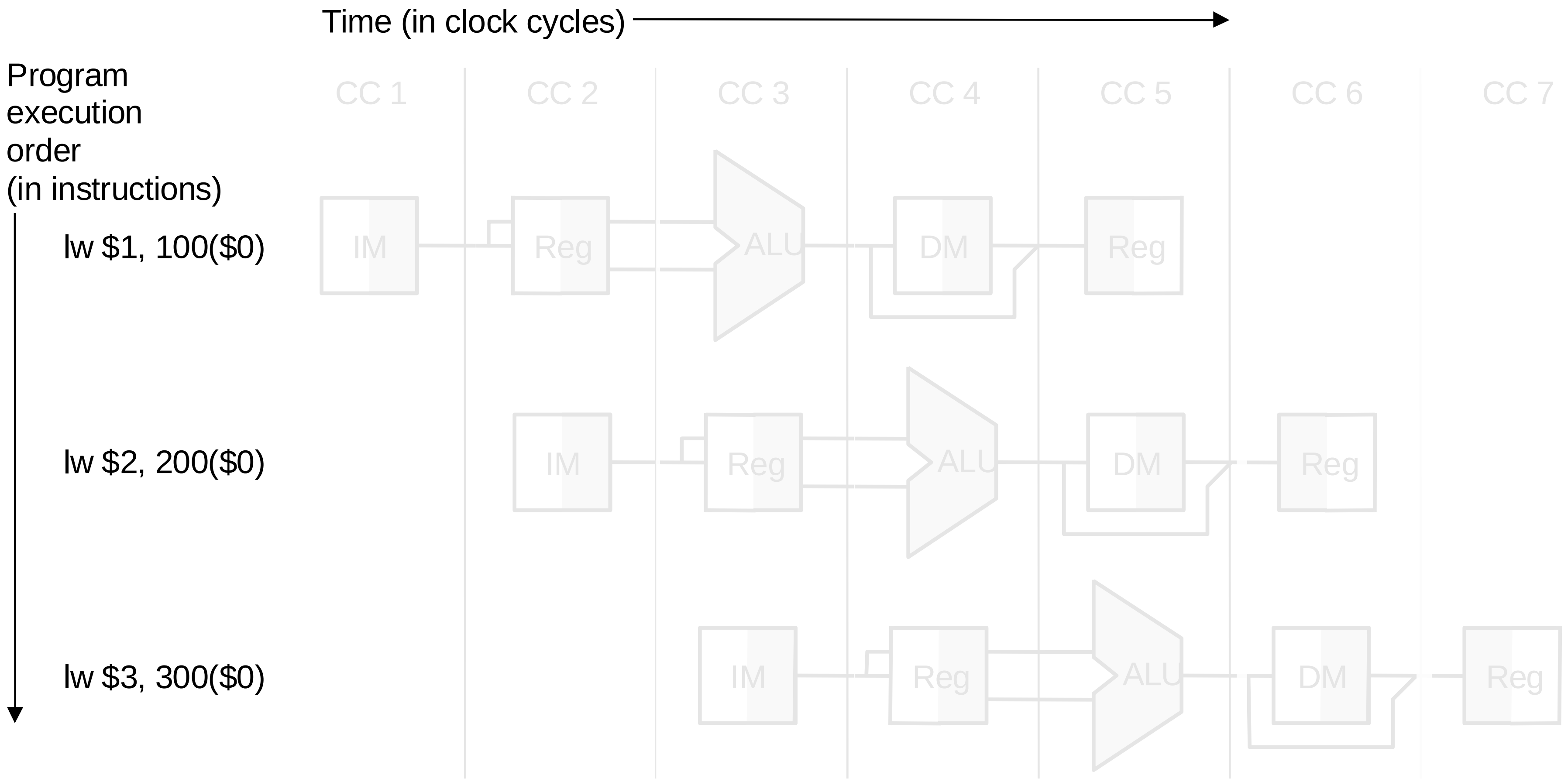
Memory Map of Code

Assembly code ⇨ Assembler ⇨ Memory map

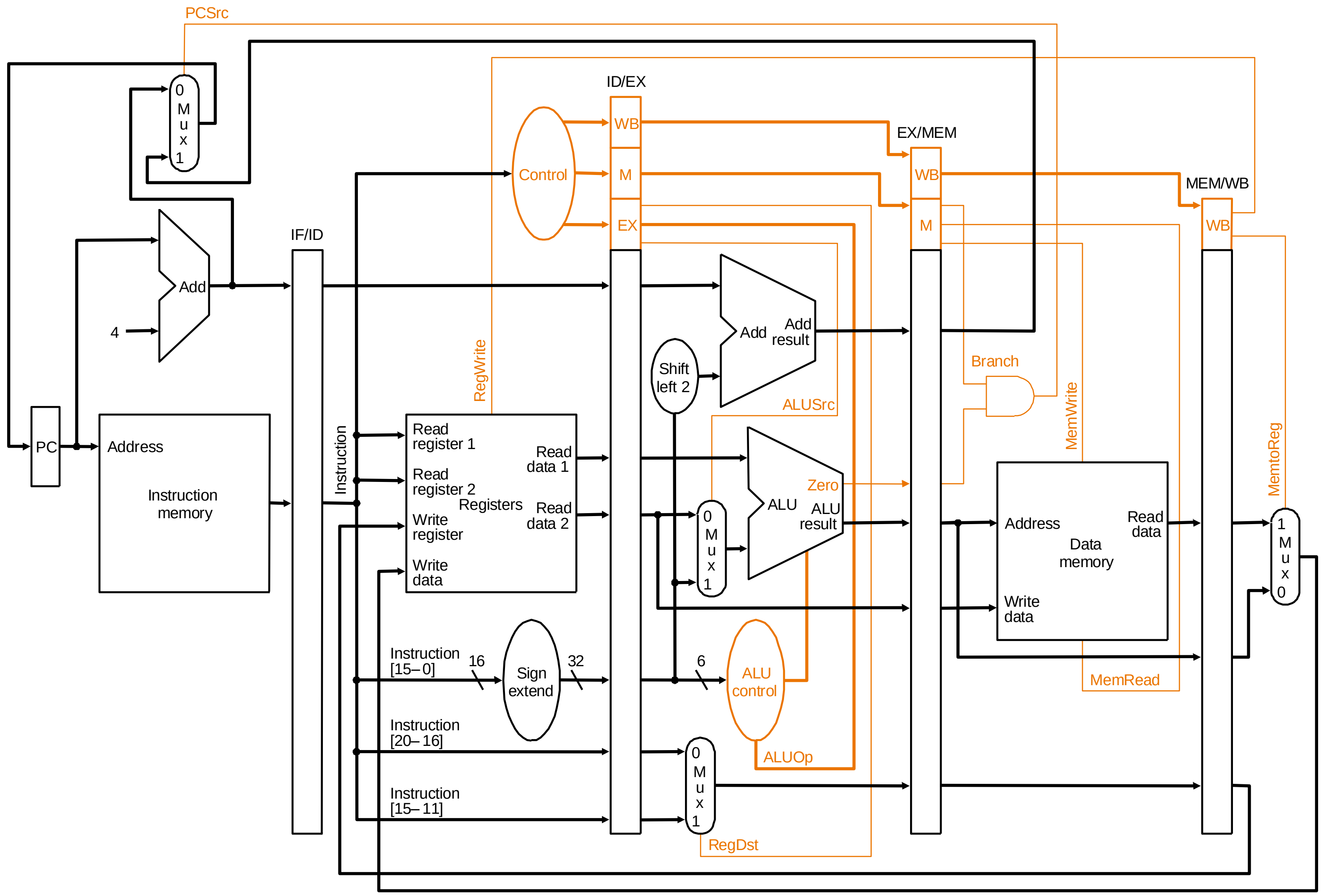
```
        # starts from 80000
Loop:       add  $t1, $s3, $s3
            add  $t1, $t1, $t1
            add  $t1, $t1, $s6
            lw   $t0, 0($t1)
            bne  $t0, $s5, Exit
            add  $s3, $s3, $s4
            j    Loop
Exit:
```

	6	5	5	5	5	6	
80000	0	19	19	9	0	0	R-type
80004	0	9	9	9	0	0	R-type
80008	0	9	22	9	0	0	R-type
80012	35	9	8	0			I-type
80016	5	8	21	2			I-type
80020	0	19	20	19	0	0	R-type
80024	2	20000					J-type
80028							

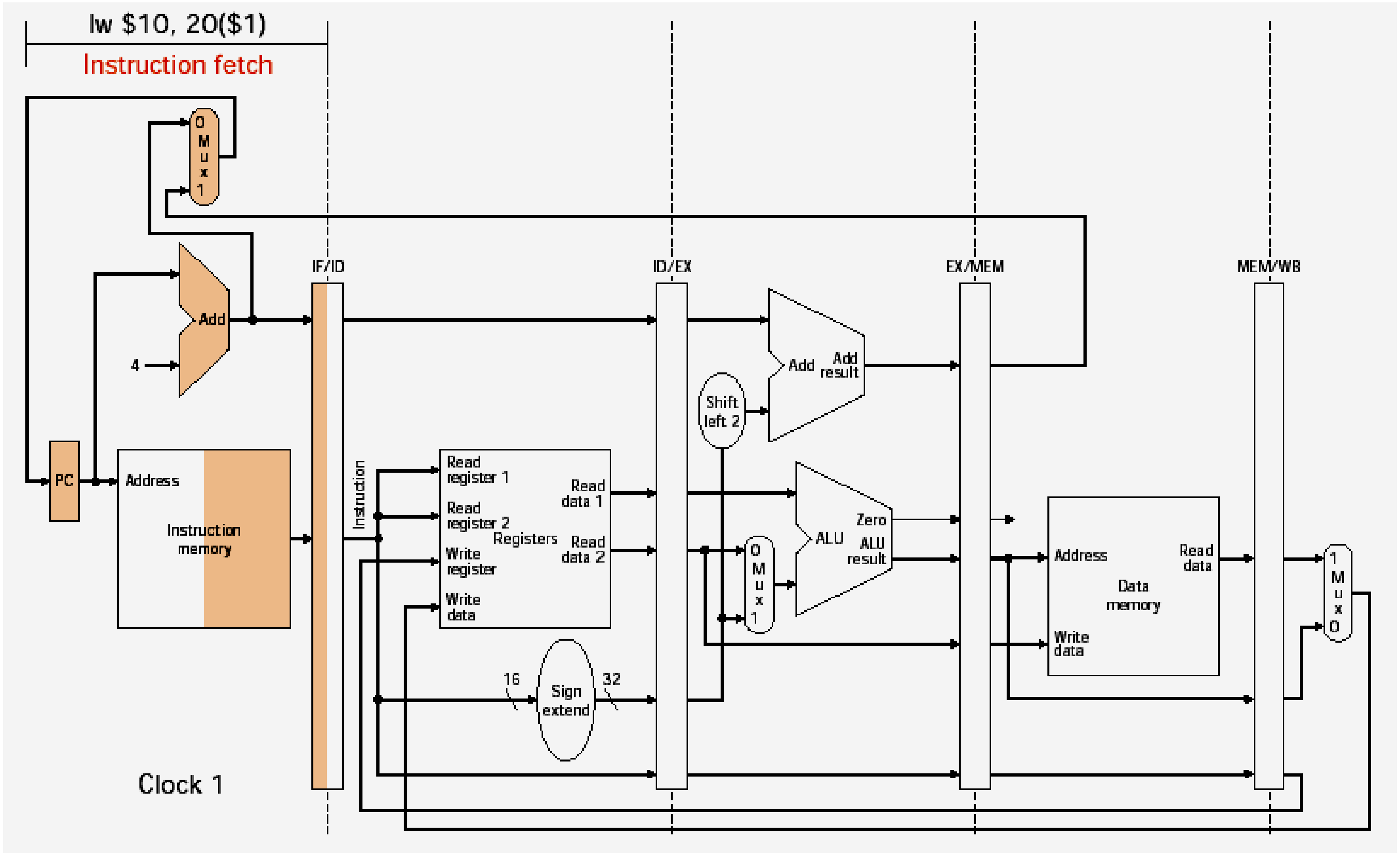
Pipelined MIPS



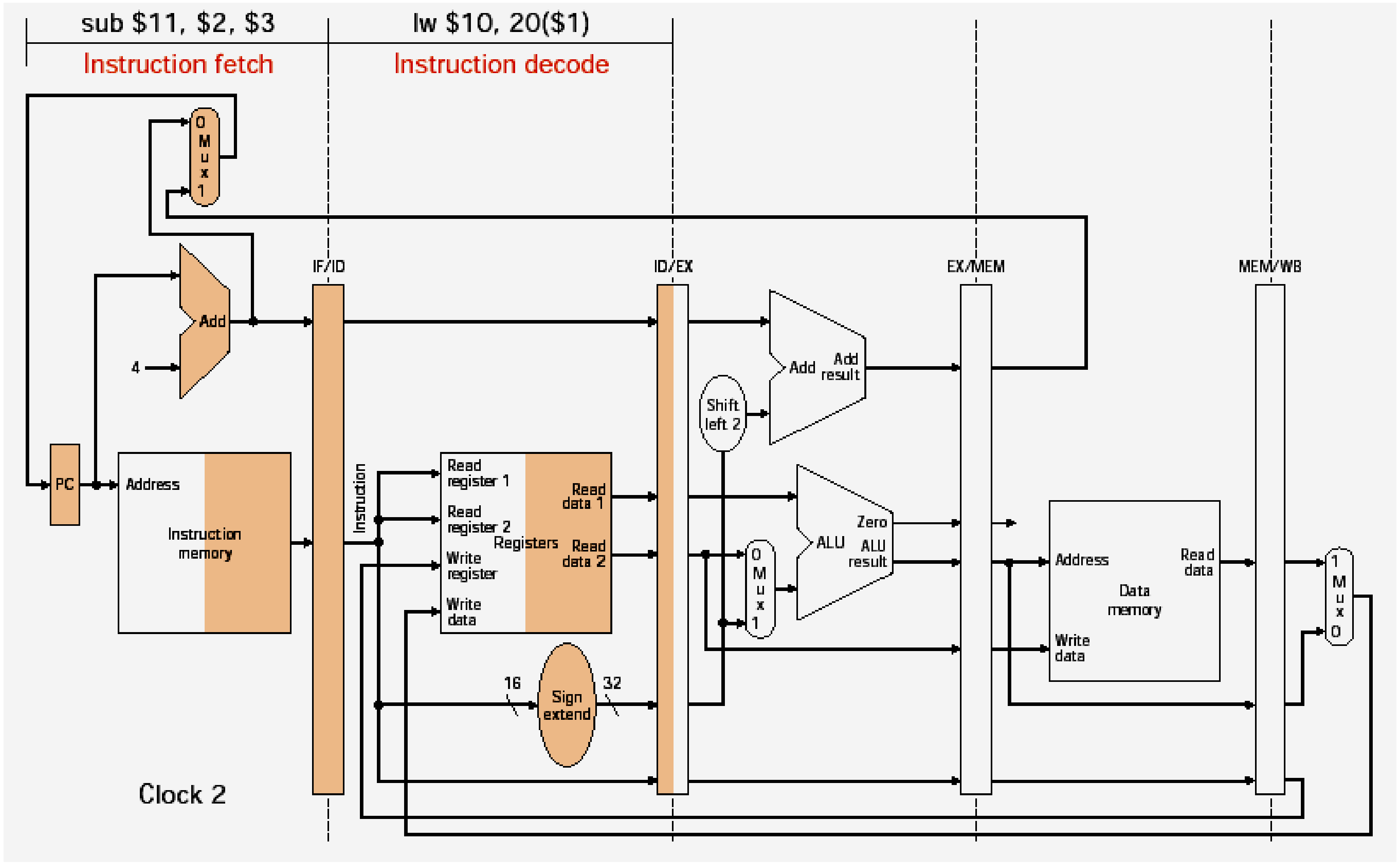
Data-path + Control Hardware



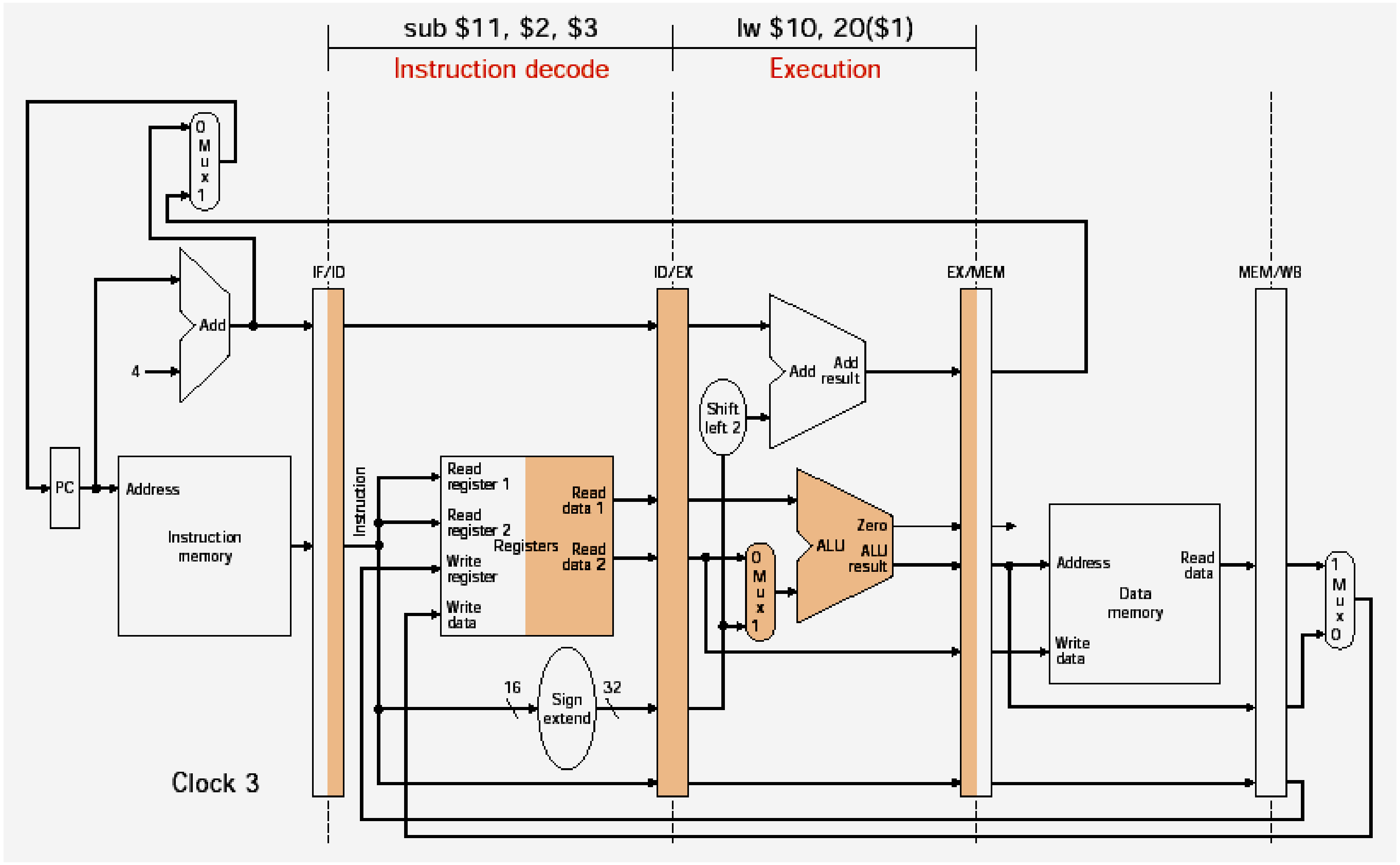
lw Instruction Fetch (IF)



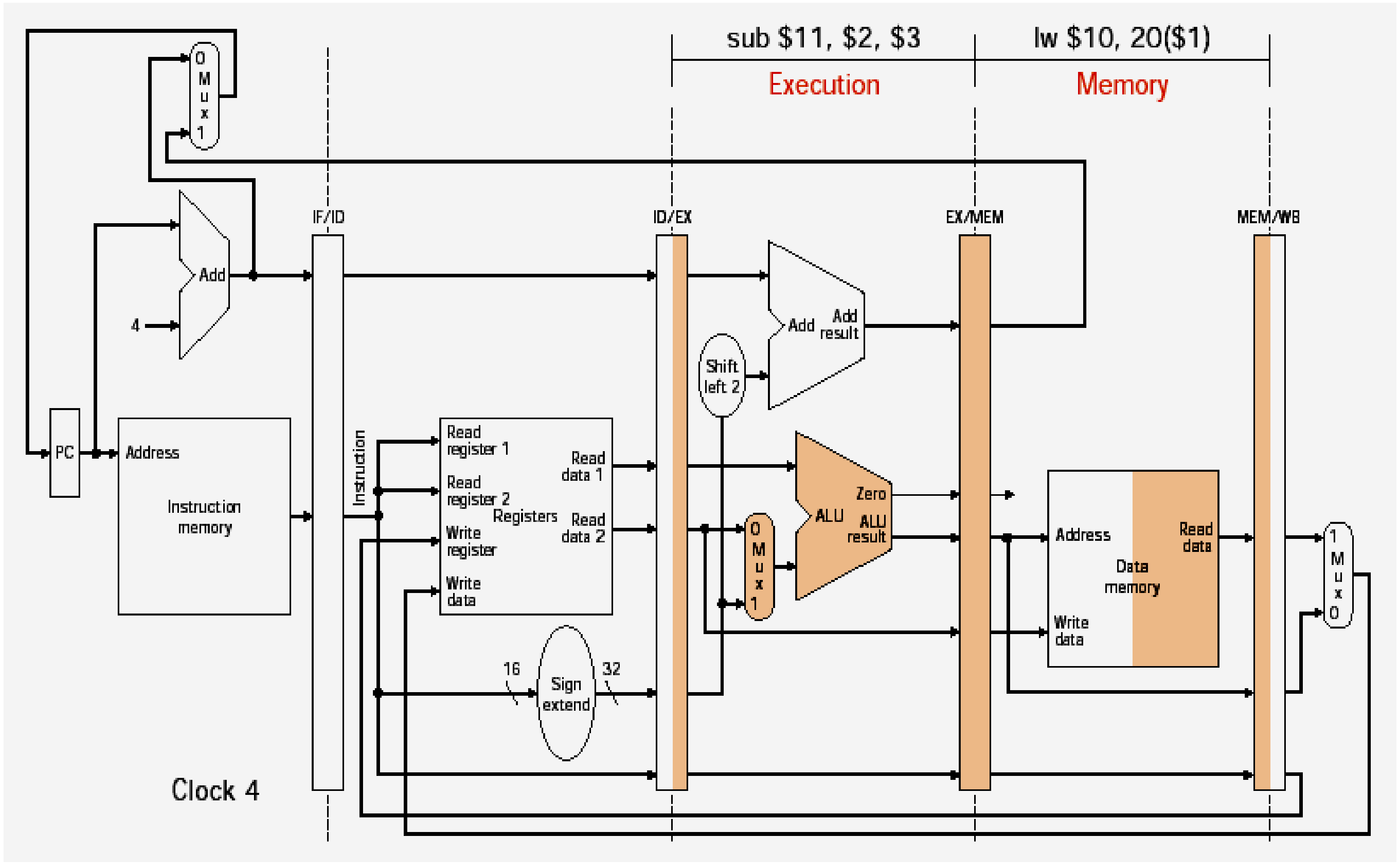
sub IF → lw Instruction Decode (ID)



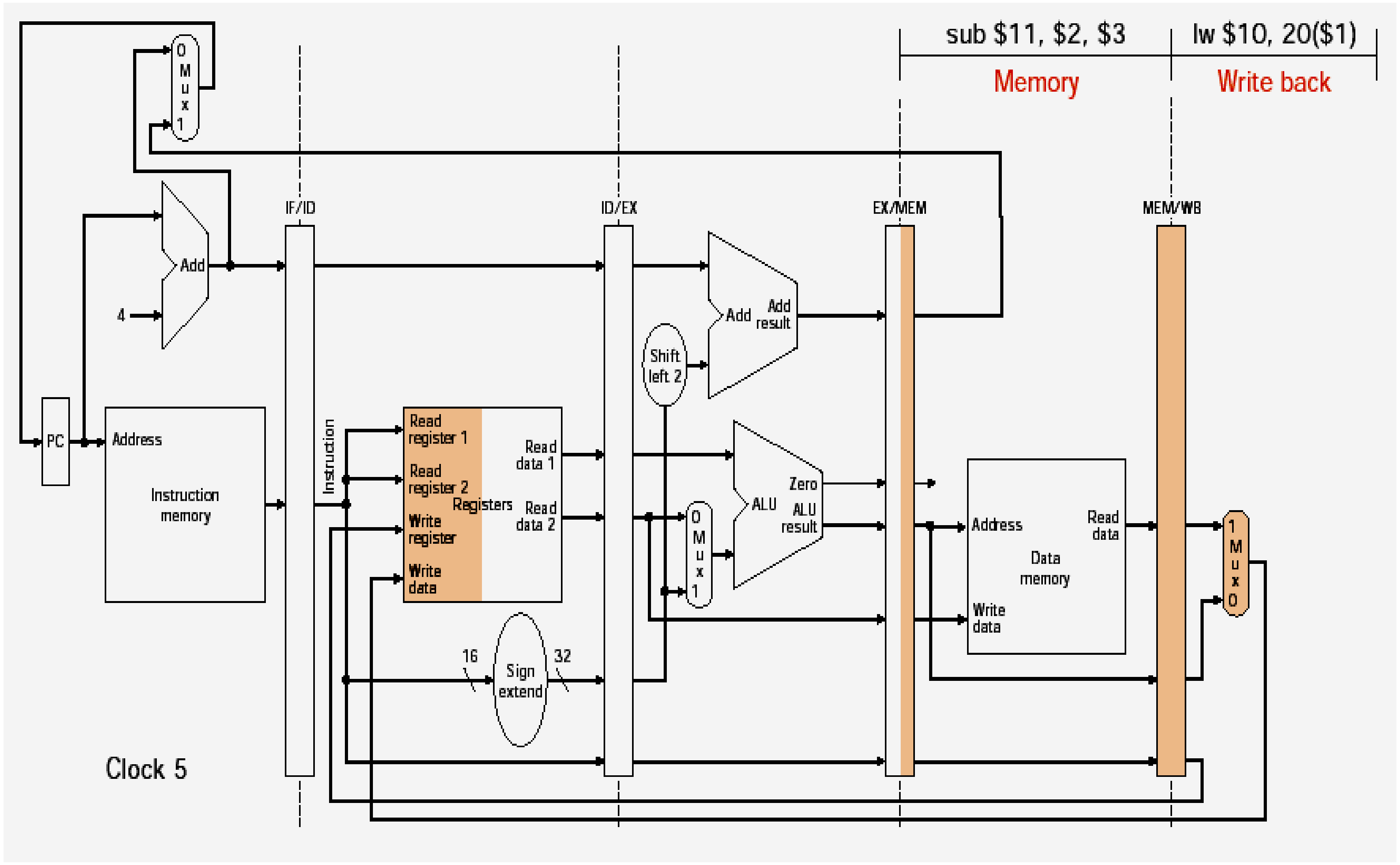
sub ID → lw Execution (EX)



sub EX → lw Memory (MEM)

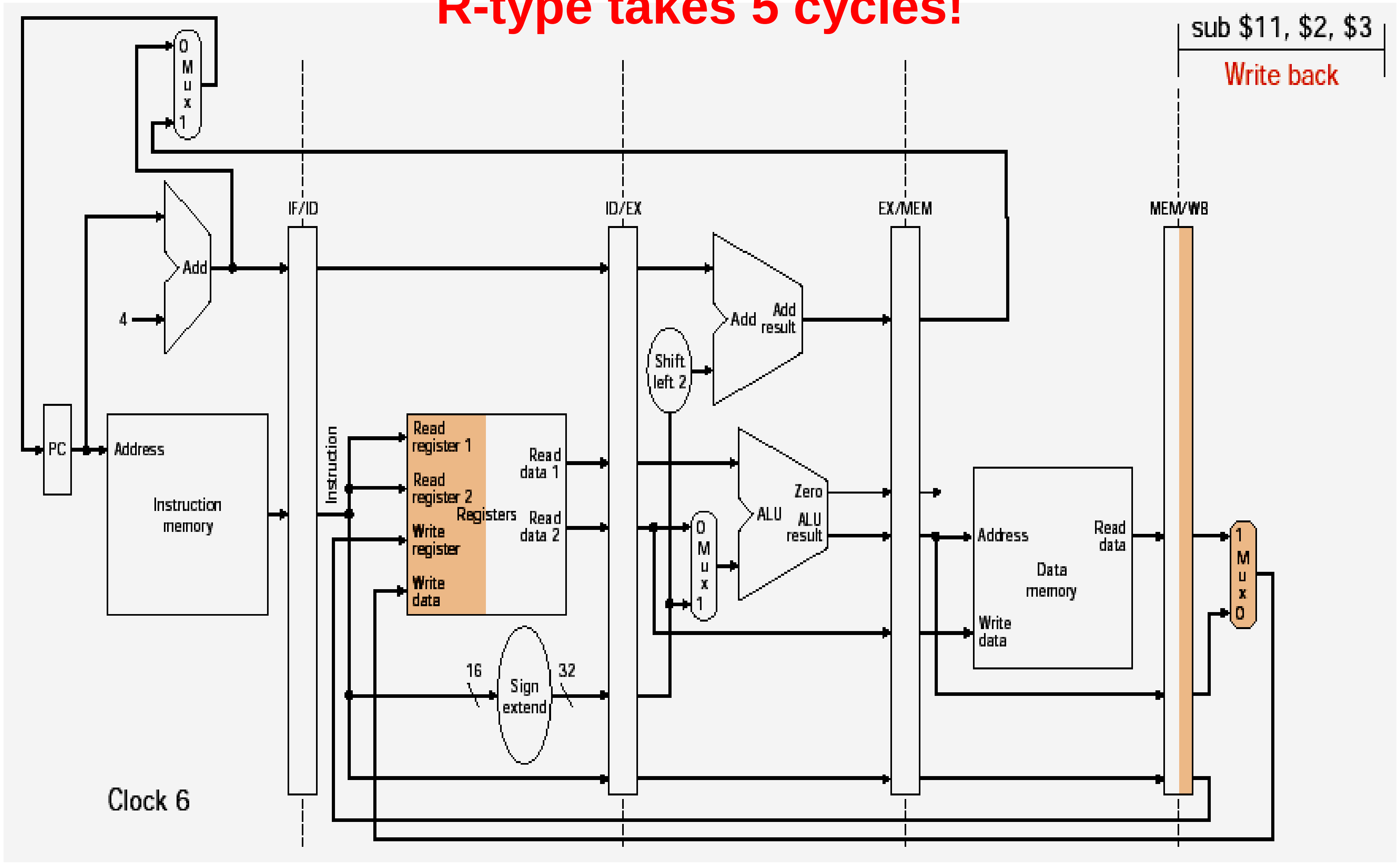


sub MEM → lw Writeback (WB)



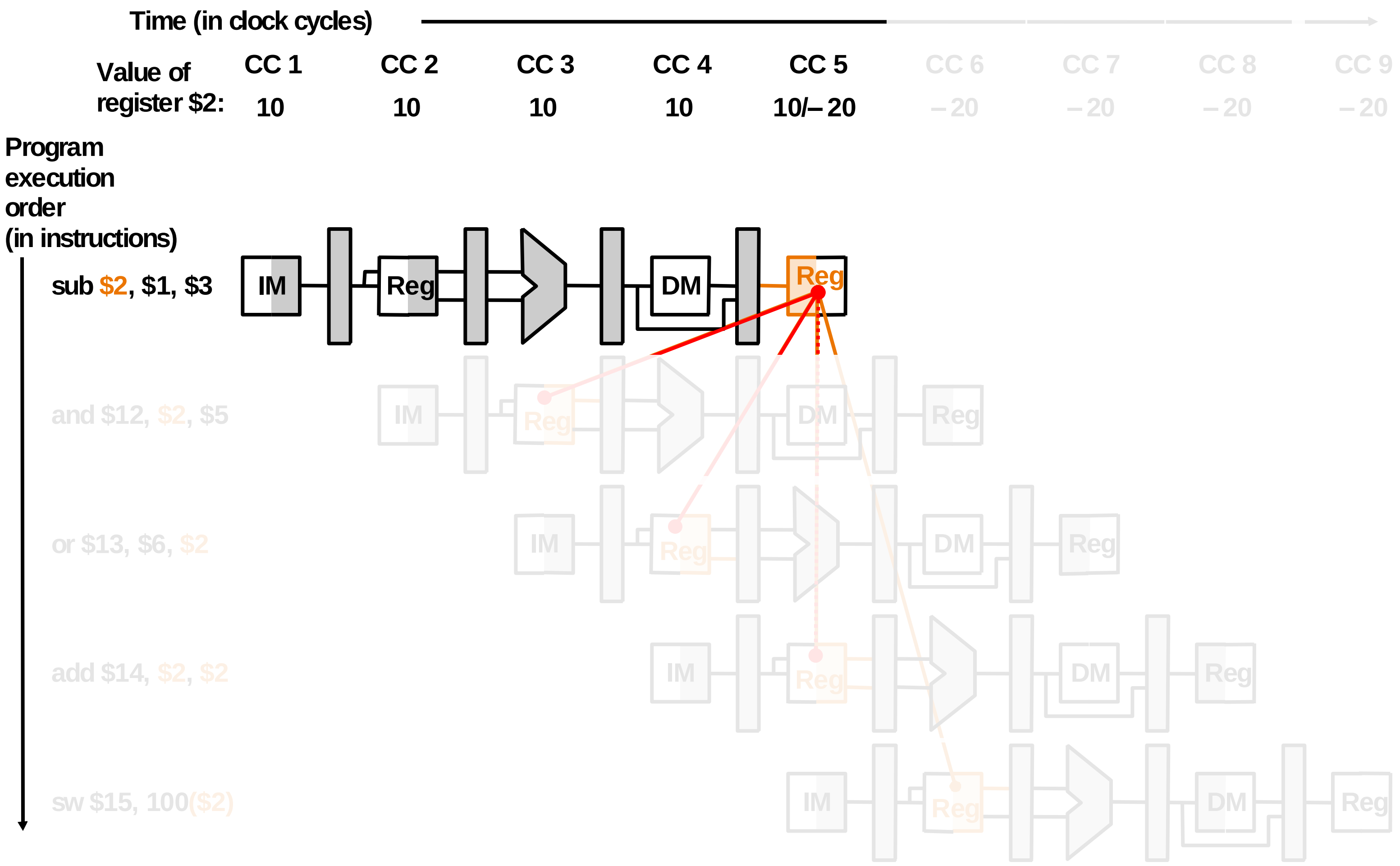
sub WB

R-type takes 5 cycles!



Data Hazards

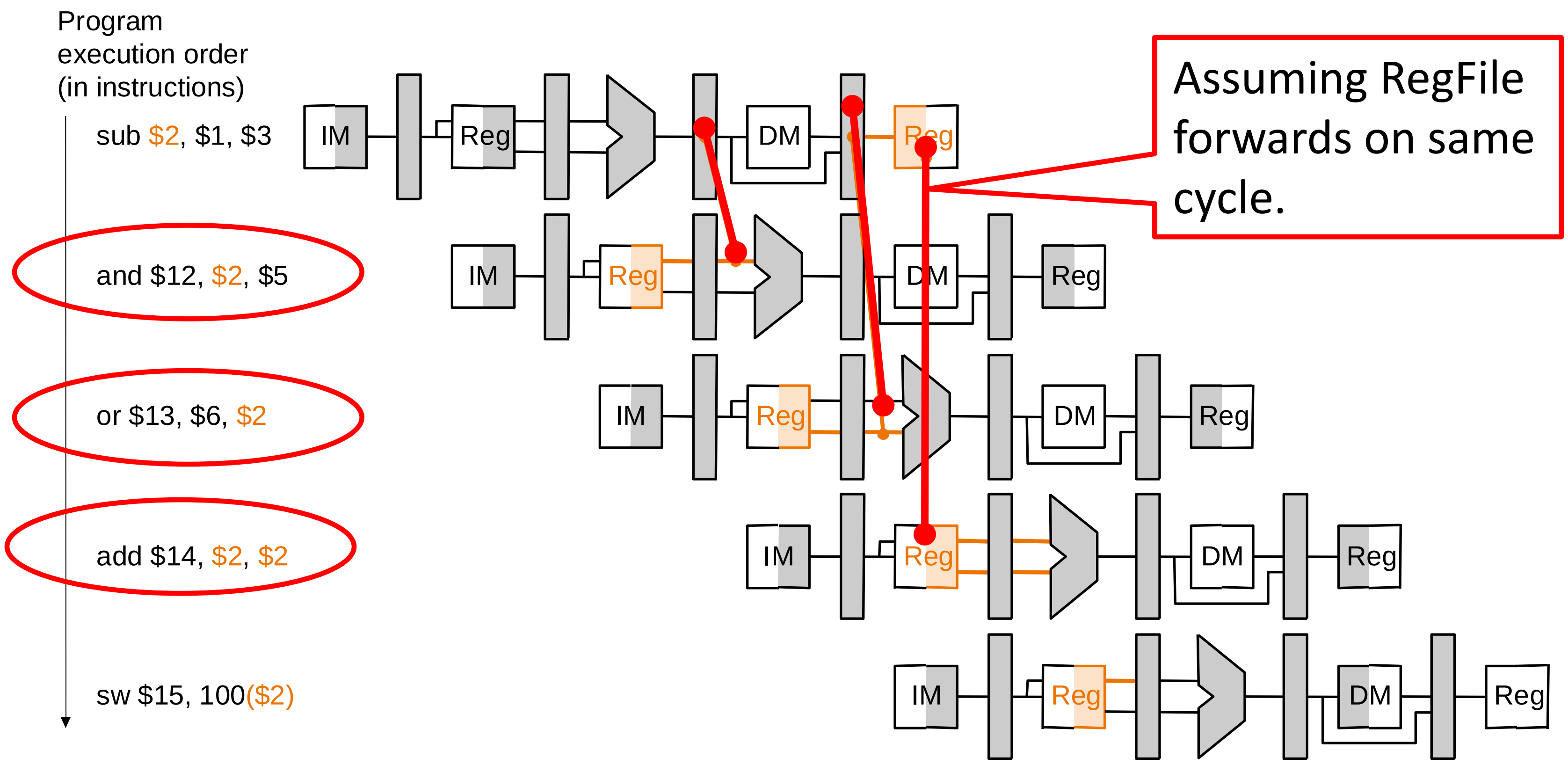
Using data produced by earlier R-type instruction.



Bypass \ Forwarding

Time (in clock cycles) →

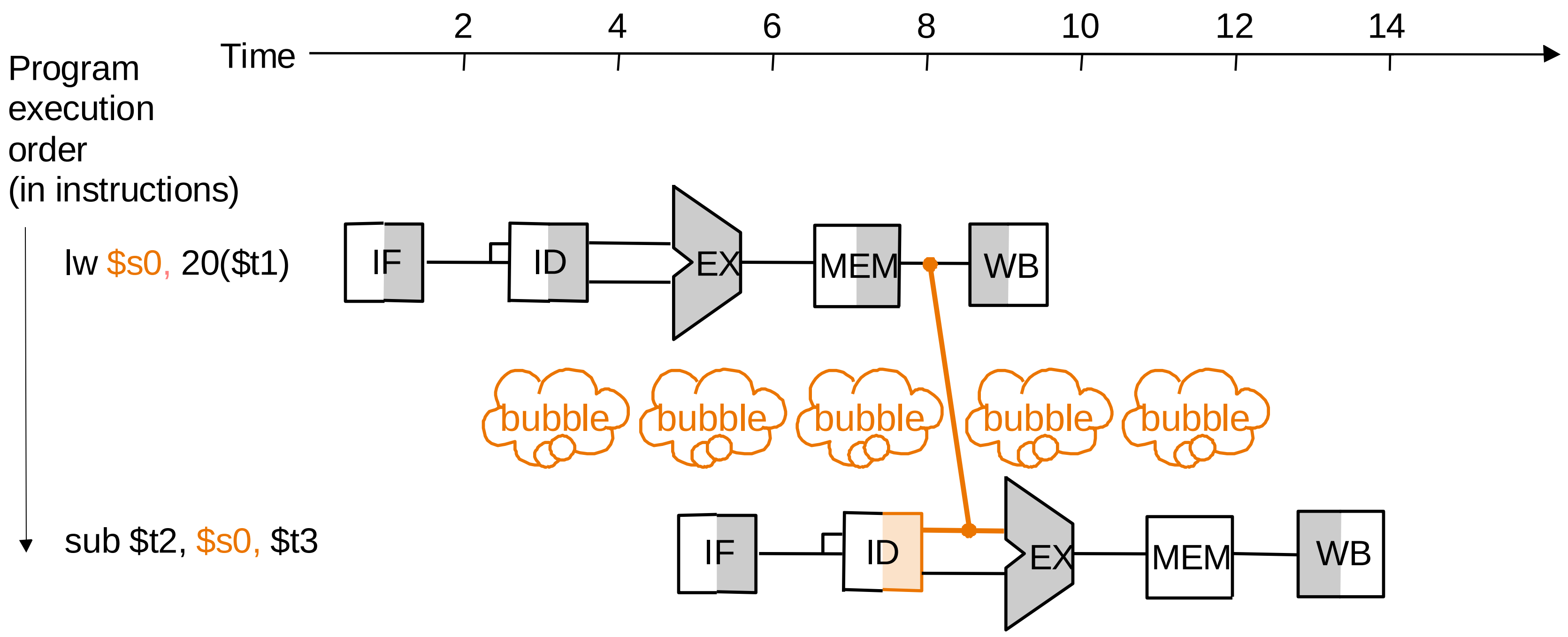
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/– 20	– 20	– 20	– 20	– 20
Value of EX/MEM :	X	X	X	– 20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	– 20	X	X	X	X



Assuming RegFile forwards on same cycle.

Load Hazards

Using data produced by earlier LW instruction.

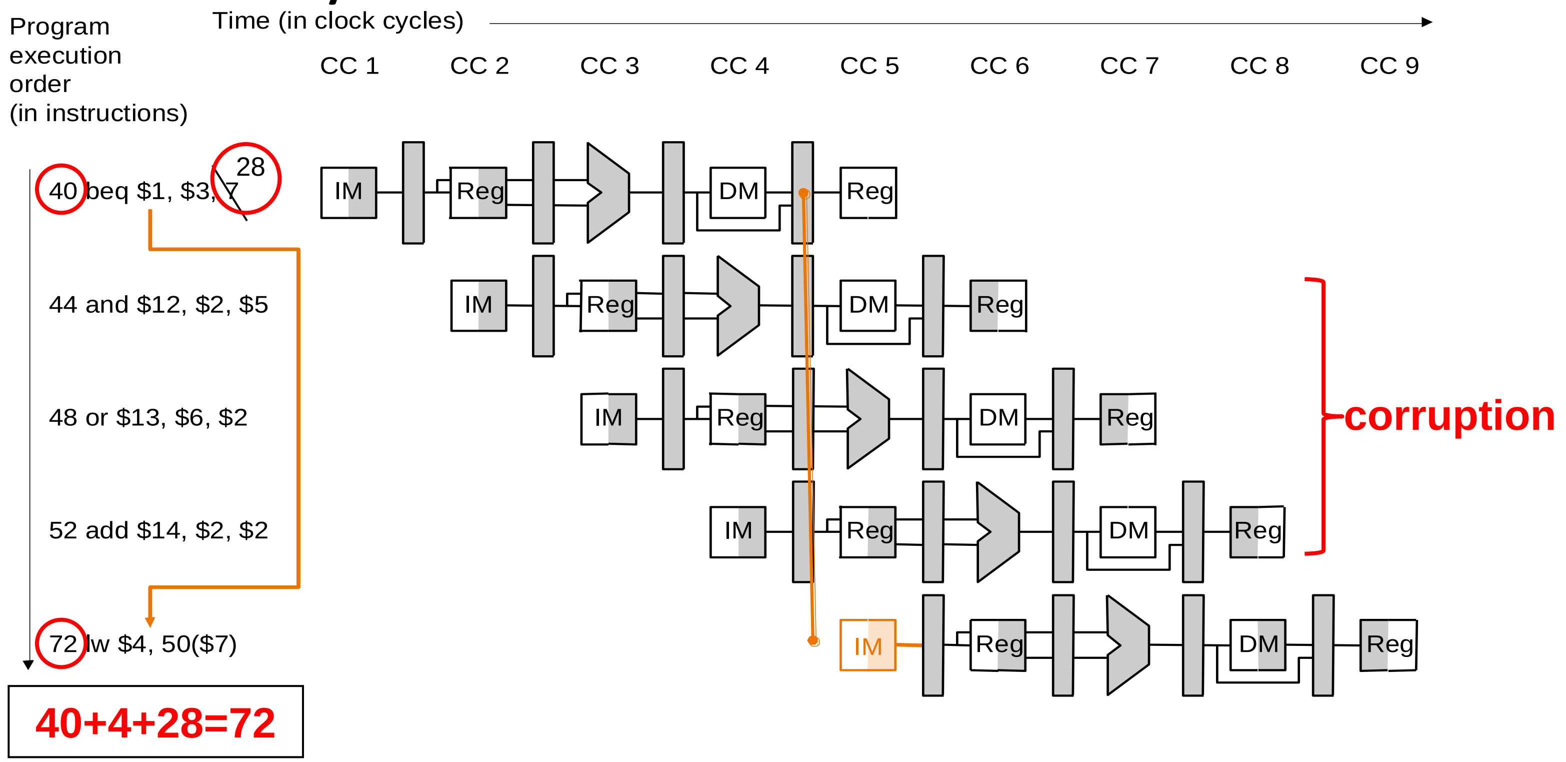


Bubble is a must. Can use NOP (SW) or CLK idle by hazard detection unit (HW).

Use same forwarding unit.

Control Hazards

Takes 3 cycles to resolve branch condition.



Solved by SW bubbles (simple, inefficient), instruction reorder (compiler, limited), by HW (efficient, complex) or combinations.

Exceptions and Interrupts

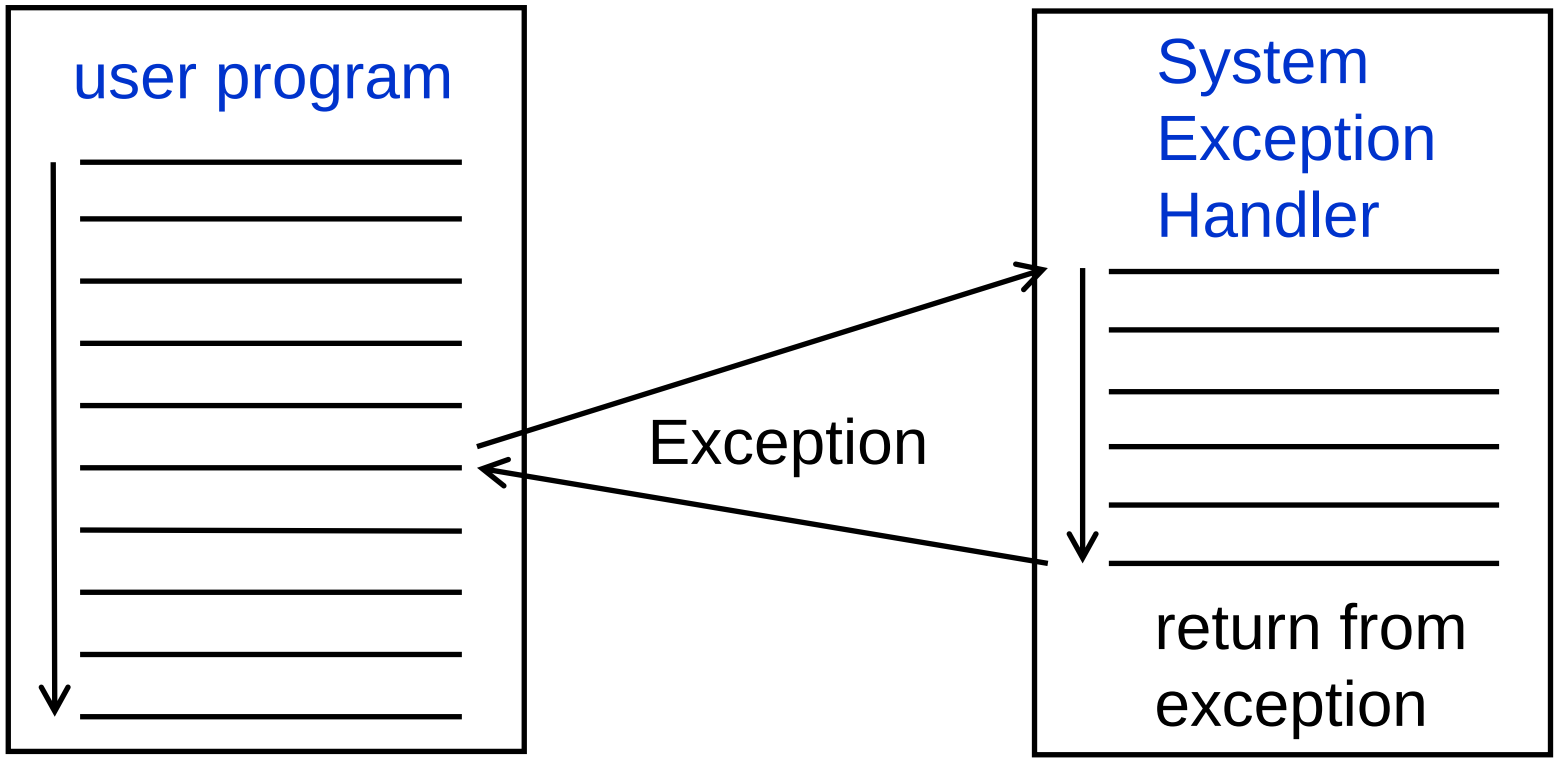
- **Exception**: any unexpected change in the internal control flow
  - Invoking an **Operating System (OS)** service from user program
  - Integer arithmetic overflow
  - Using an undefined or unimplemented instruction
  - Hardware malfunctions
- **Interrupt**: event is externally caused
  - I/O device request
  - Tracing instruction execution
  - Breakpoint (programmer-requested interrupt)

Exceptions in MIPS

stage	Problem exceptions occurring
IF	Page fault on IF, misaligned memory access, memory protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch, misaligned memory access, memory protection violation
WB	None

What Happens During an Exception?

- An exception occurs
- Operating system trap
- Saving the PC where the exception happens
- Save the operating system state
- Run exception code
- Resume the last instruction before it traps, or terminate the program



normal control flow:
sequential, jumps, branches, calls, returns

Multicycle Operations

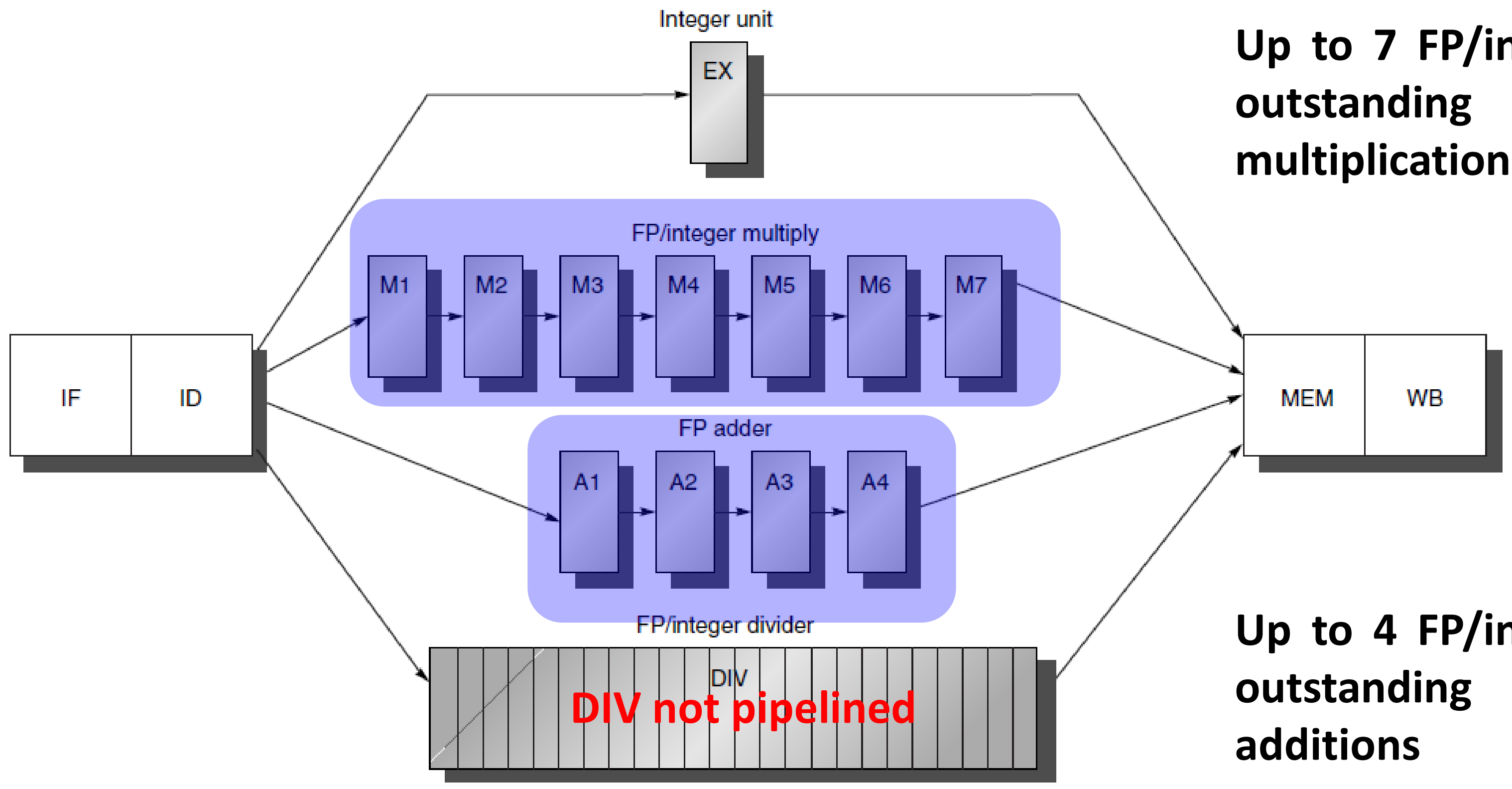
Supporting **floating point (FP)** operations, EX takes few clock cycles.

- Single cycle FP implies slow clock or enormous hardware, both undesired.

Support multiple FP units working simultaneously.

A stall occurs if an issued instruction causes **structural hazard** or **data hazard**.

Four separate functional units operated in parallel.

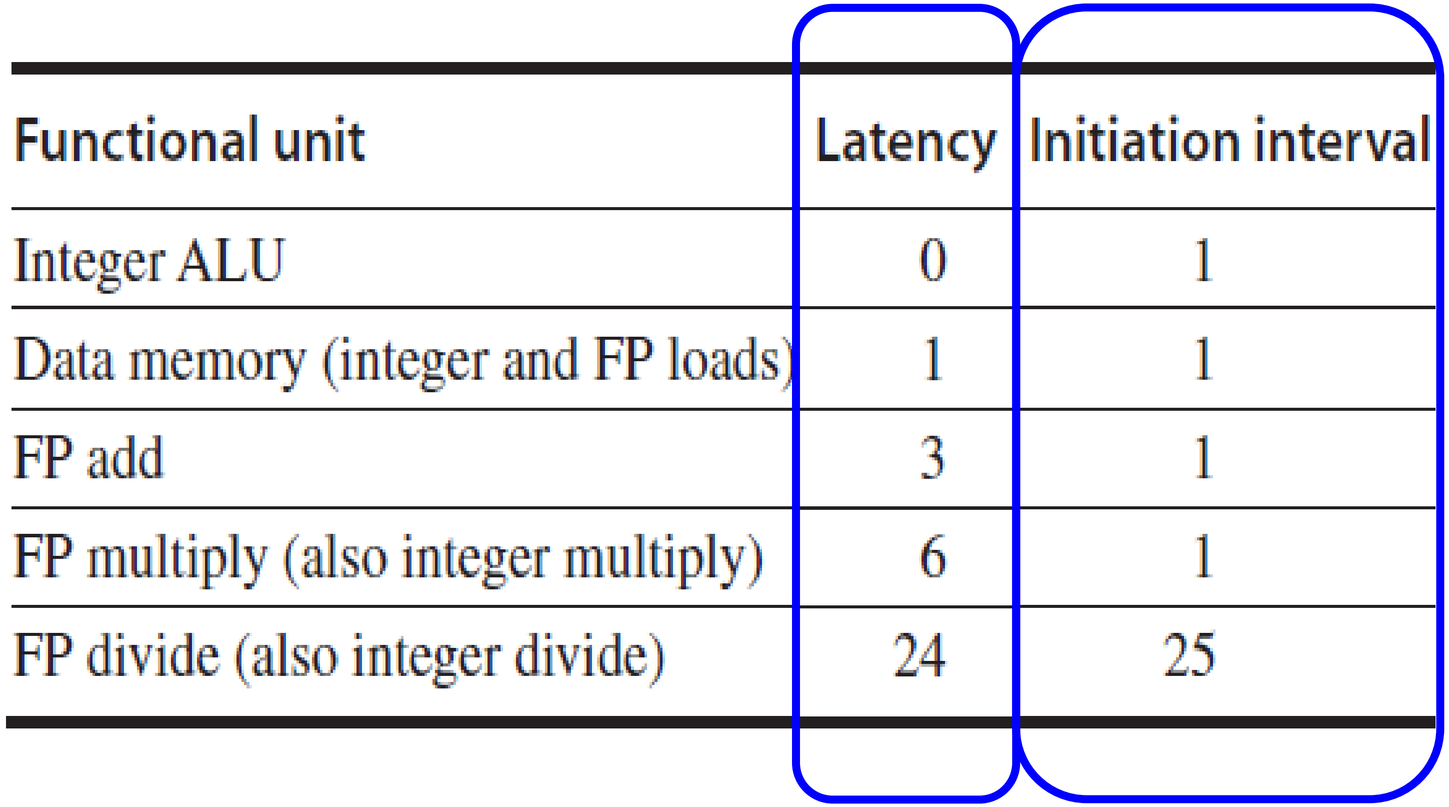


Up to 7 FP/int outstanding multiplications

Up to 4 FP/int outstanding additions

Pipeline Regs (A1/A2,…, A3/A4), (M1/M2,…, M6/M7)

ID/EX Reg placed by ID/EX, ID/DIV, ID/M1, and ID/A1.



Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

Latency : # **intervening** cycles between an instruction that produces a result and an instruction that uses it.

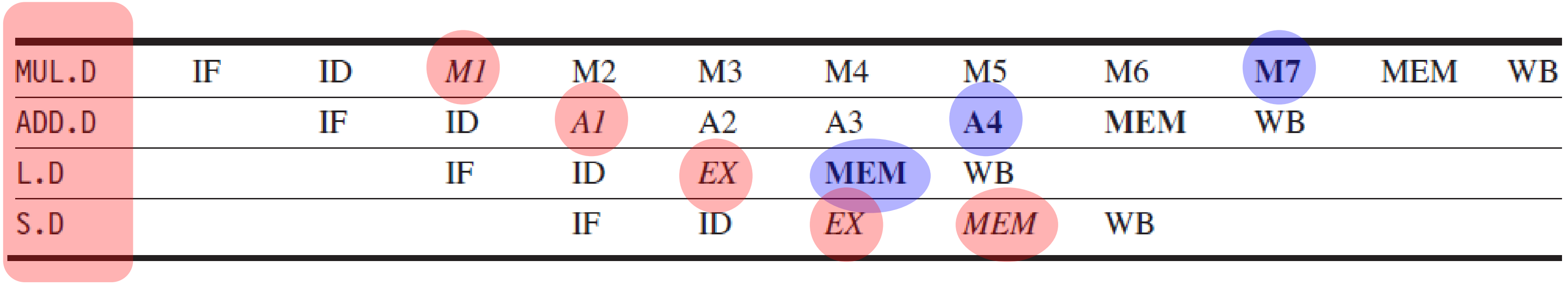
Initiation: # cycles that must **elapse** between issuing two operations of a given type.

Integer ALU latency is 0, since its result can be used on next clock cycle.

Loads latency is 1, since its results can be used after one intervening cycle.

".D" extension of instruction stands for **double-precision** (64-bit) FP operations.

data needed result available



MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	**A4**	MEM	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

Data Hazards

Created when a dependence between instructions is close enough.

- **Program order** must be preserved.

SW and HW techniques exploit parallelism by preserving program order **only where it affects the outcome of the program**.

Hazards are classified by the order of reads and writes.

Consider two instructions ***i*** and ***j***, with ***i*** preceding ***j***. Three possible data hazards.

RAW (read after write). ***j*** reads a source before written by ***i*** (**true dependence**).

- Most common, program order must be preserved.

WAW (write after write). ***j*** writes an operand before written by ***i*** (**output dependence**).

- Wrong write order, leaving value written by ***i*** rather than ***j***.
- Writing in **more than one pipe** stage or allow instruction to proceed when previous one is stalled.

WAR (write after read). ***j*** writes a destination before read by ***i***, ***i*** gets wrong value (**anti dependence**).

- Occurs when instructions **reordered**. Not in static issue pipelines since all **reads are early** (ID) and all **writes are late** (WB).

Structural hazards occurs by HW conflicts.

- Divide unit is **not pipelined**, causing structural hazards. Must detect and stall instructions issue.
- Varying running times of instruction may result in few register writes in a cycle.

Instructions can complete in a different order than they were issued, causing problems with **exceptions**.

Because of longer latency of operations, stalls for **RAW hazards** will be more frequent.

Instructions below depends on their previous and proceeds as soon as data are available, assuming that the pipeline has full bypassing and forwarding.

	Clock cycle number																
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`L.D F4,0(R2)`	IF	ID	EX	MEM	WB												
`MUL.D F0,F4,F6`		IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
`ADD.D F2,F0,F8`			IF	stall	ID	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	MEM	WB
`S.D F2,0(R2)`					IF	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	MEM

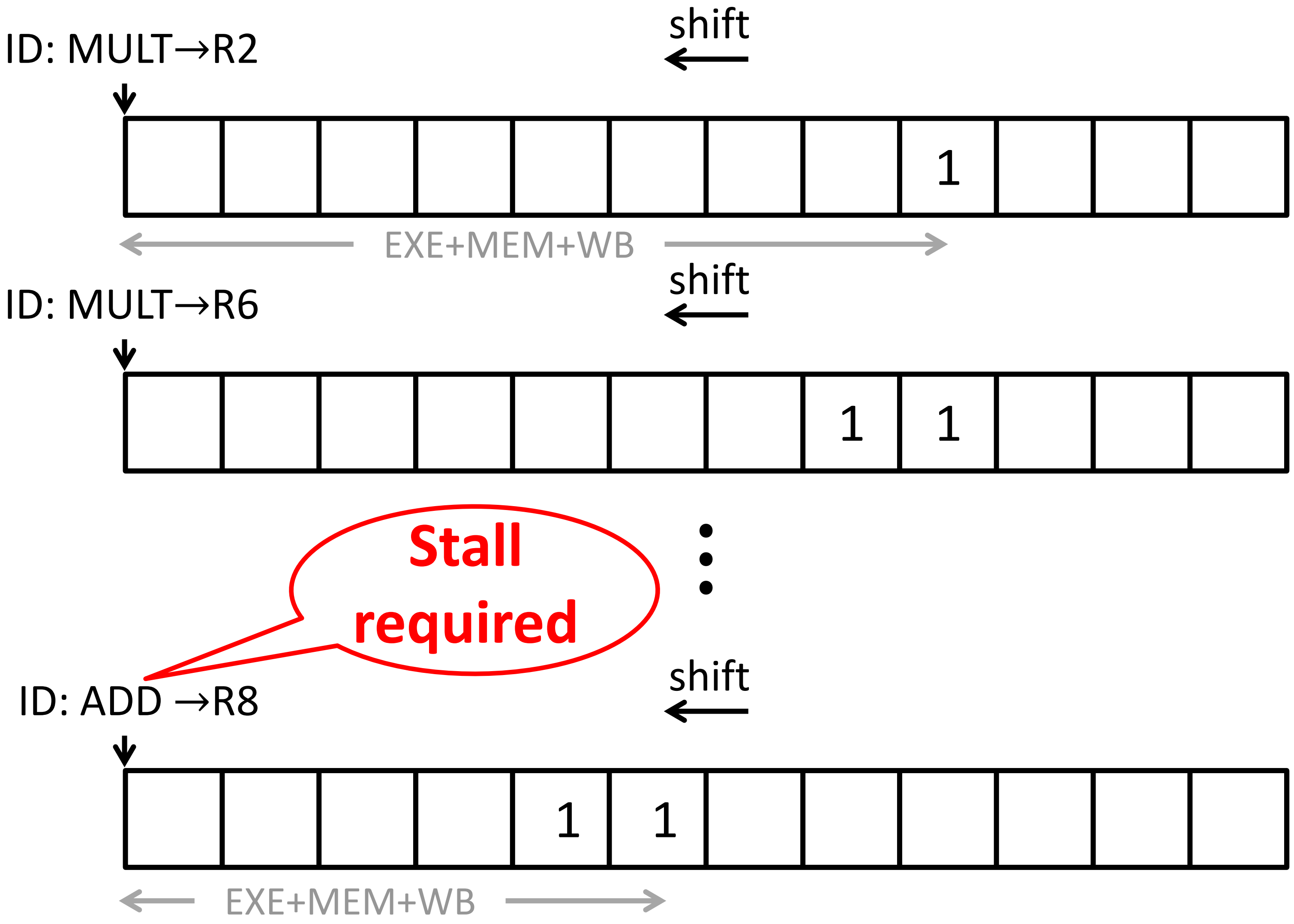
S.D is stalled extra cycle so that it does not conflict with ADD.D in MEM. Needs extra HW to handle this.

	Clock cycle number										
Instruction	1	2	3	4	5	6	7	8	9	10	11
`MUL.D F0,F4,F6`	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
`ADD.D F2,F4,F6`				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
`L.D  F2,0(R2)`							IF	ID	EX	MEM	WB

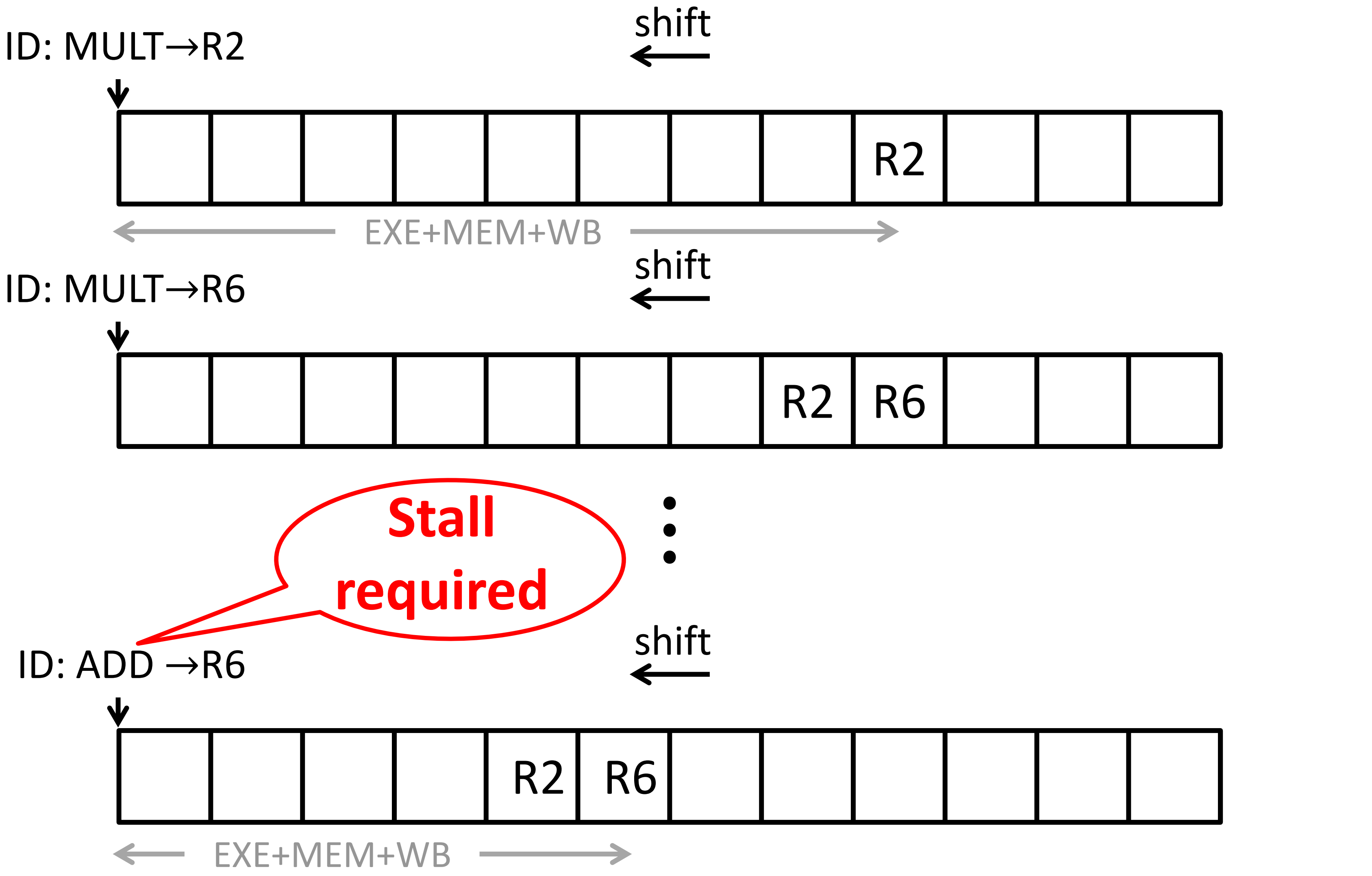
Three instructions in MEM. Is it a **structural** hazard? No. The first two MEM do not write to MEM.

Instructions are in WB, resulting in a structural hazard. The processor must serialize the WB. Write ports could be increased, but it may not pay (only rarely used 2nd).

single port RF



dual port RF


shift
ID: MULT→R2
R2
EXE+MEM+WB
shift
ID: MULT→R6
R2
R6
Stall required
ID: ADD →R6
shift
R2
R6
EXE+MEM+WB

Alternative solution detects conflicts at **MEM** or **WB** stage, a case where either instruction can be stalled.

Simple heuristic gives priority to unit with longer latency, since it is the most likely to cause other stalls due to **RAW hazards**.

Advantage is the simple implementation.

Disadvantage is that it complicates pipeline control, as **stalls** can now arise from **two places**.