5118007-02 Computer Architecture

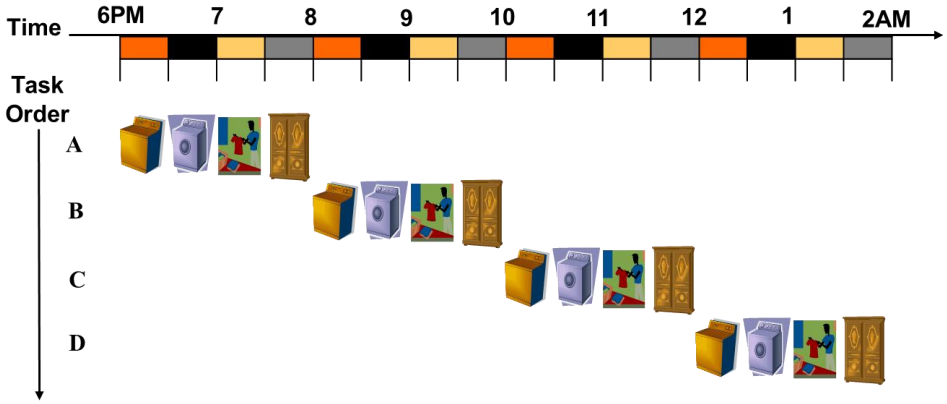
Ch. 4 The Processor

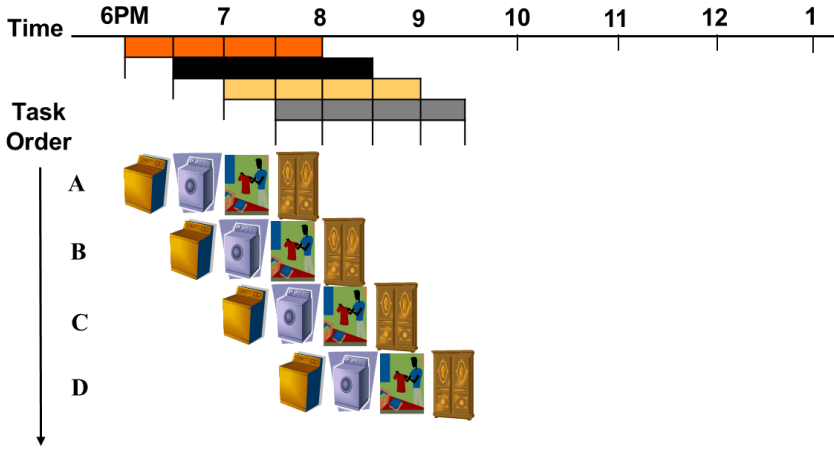
14 May 2024

Shin Hong

Pipelining

- Several instruction-executions can be overlapped to enhance CPU throughput
- Ex. Laundry analogy





Ideas

- Multiple aspects of a task are operating simultaneously on different components of a machine
  - does not help latency, but enhance throughput
  - potential speed-up is proportional to the number of stages
  - pipeline rate is determined by the slowest stage

Speed-Up by Pipelining

Suppose there are *n* jobs to execute and each job takes *T* seconds.

(a) Without pipelining

total execution time *T*s = *n***T*

(b) With pipelining

suppose we have *k* stage pipe and each stage takes *T*/*k* seconds

total execution time *T*p = (*n*+*k*-1) * *T*/*k*

Therefore speedup *S*p = *T*s / *T*p = (*n***k*) / (*n*+*k*-1)

If *n* goes infinity, *S*p becomes *k*

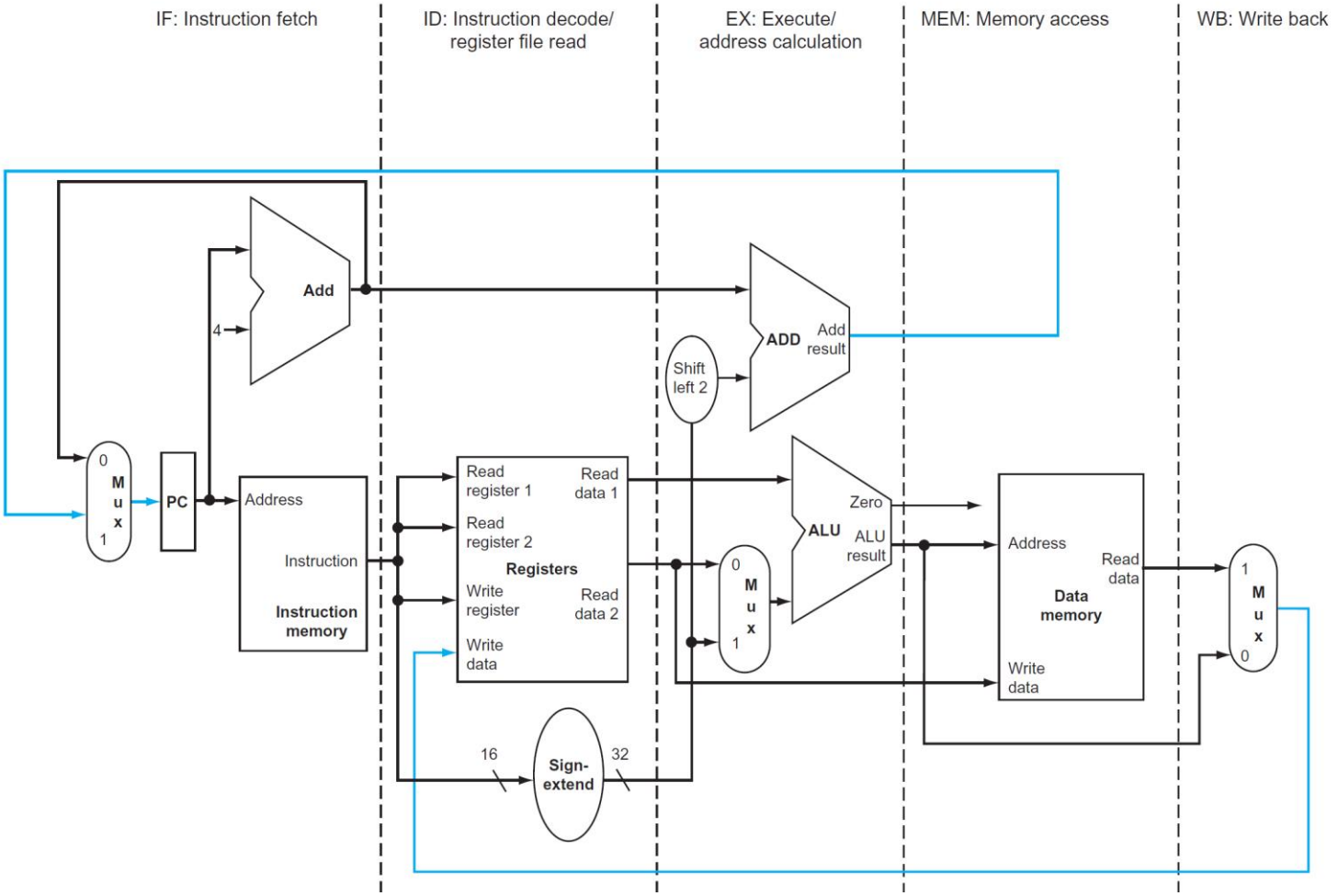
Ex) *n*=100, *T*=10

(a) *T*s = 100*10 = 1000sec

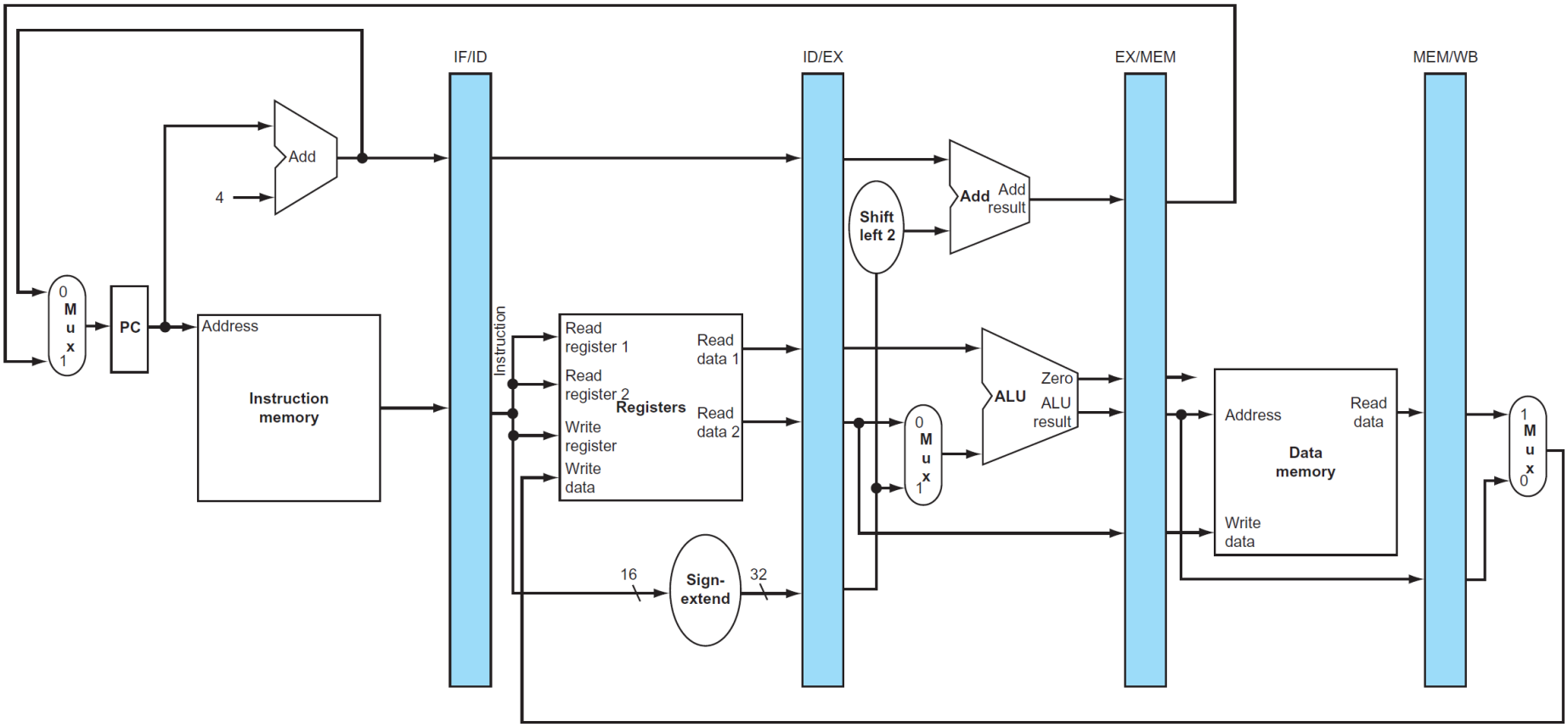
(b) *k*=5, *T*p = (100+5-1) * 10/5 = 208 seconds

(c) *k*=10, *T*p = (100+10-1) * 10/10 = 109 seconds

Stages in Instruction-Execution



Pipelined Datapath



Basic Steps

Step 1. Instruction fetch step (IF)

Step 2. Instruction decode/register fetch step (ID)

Step 3. Execution/effective address step (EX)

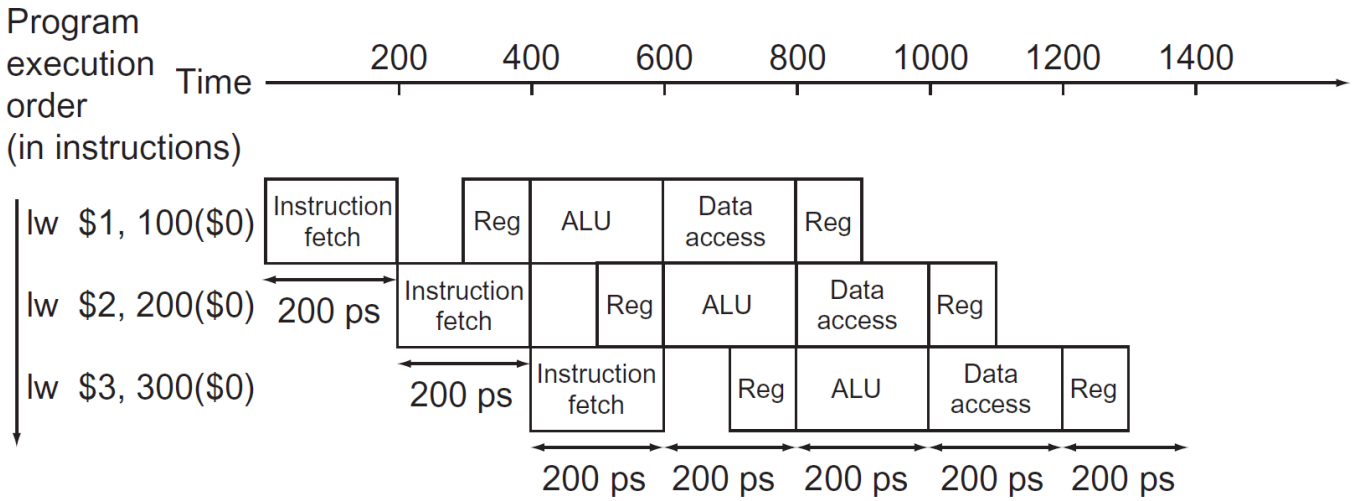
Step 4. Memory access (MEM)

Step 5. Register write-back step (WB)

Pipelined Instruction Execution

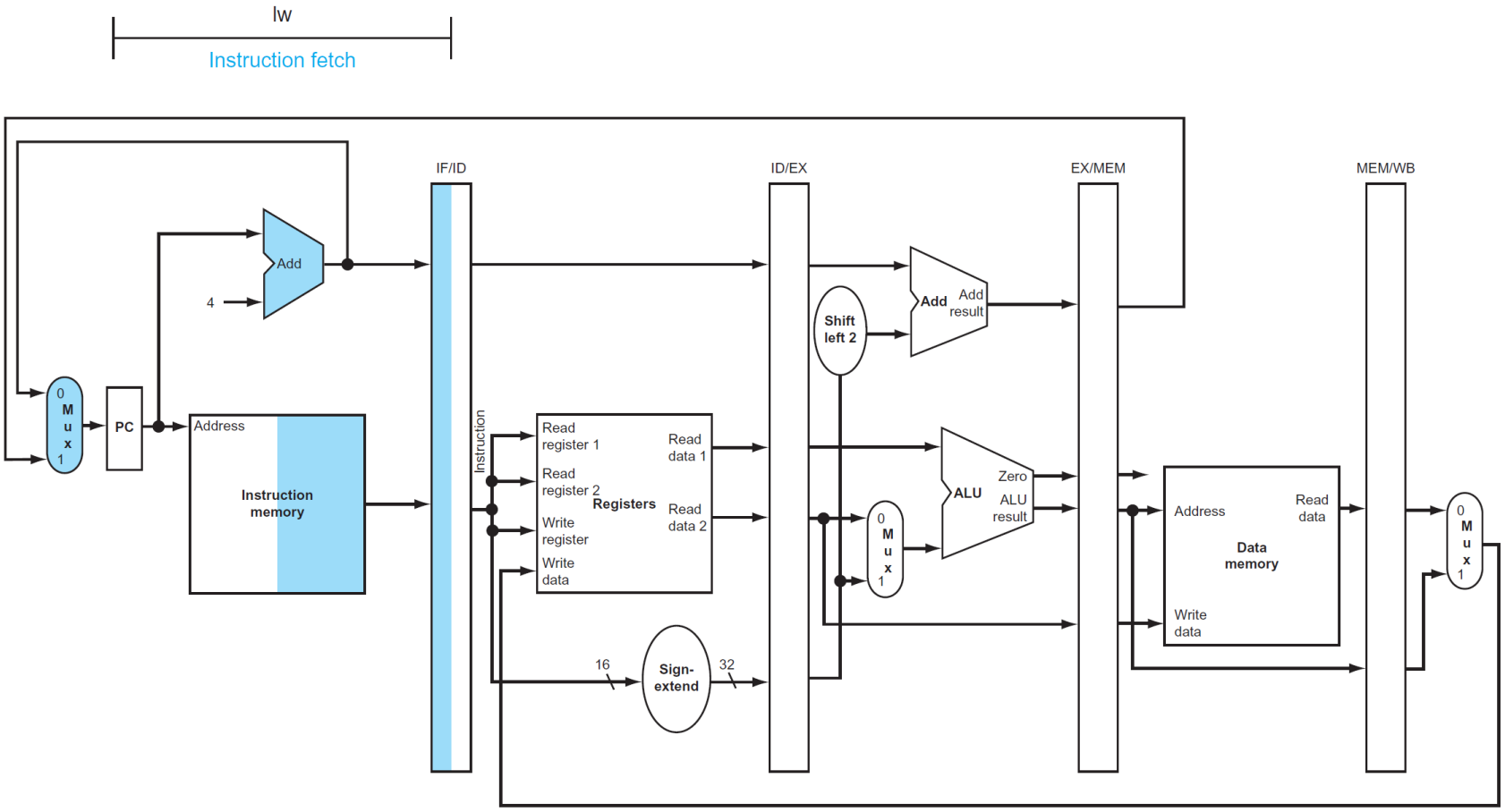
Instruction number	Clock number								
	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction i + 1		IF	ID	EX	MEM	WB			
Instruction i + 2			IF	ID	EX	MEM	WB		
Instruction i + 3				IF	ID	EX	MEM	WB	
Instruction i + 4					IF	ID	EX	MEM	WB

Pipelined Instruction Execution

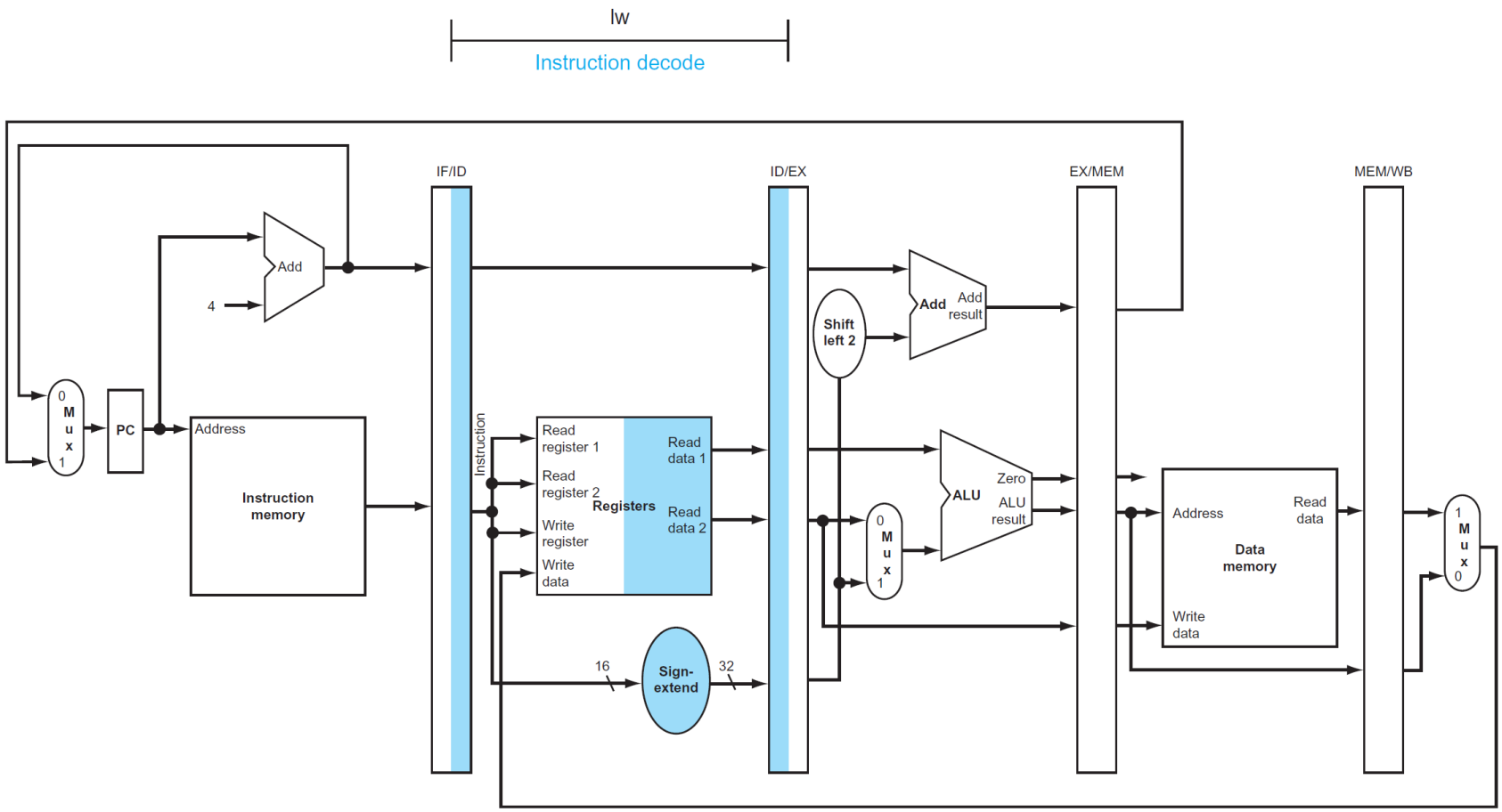


Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

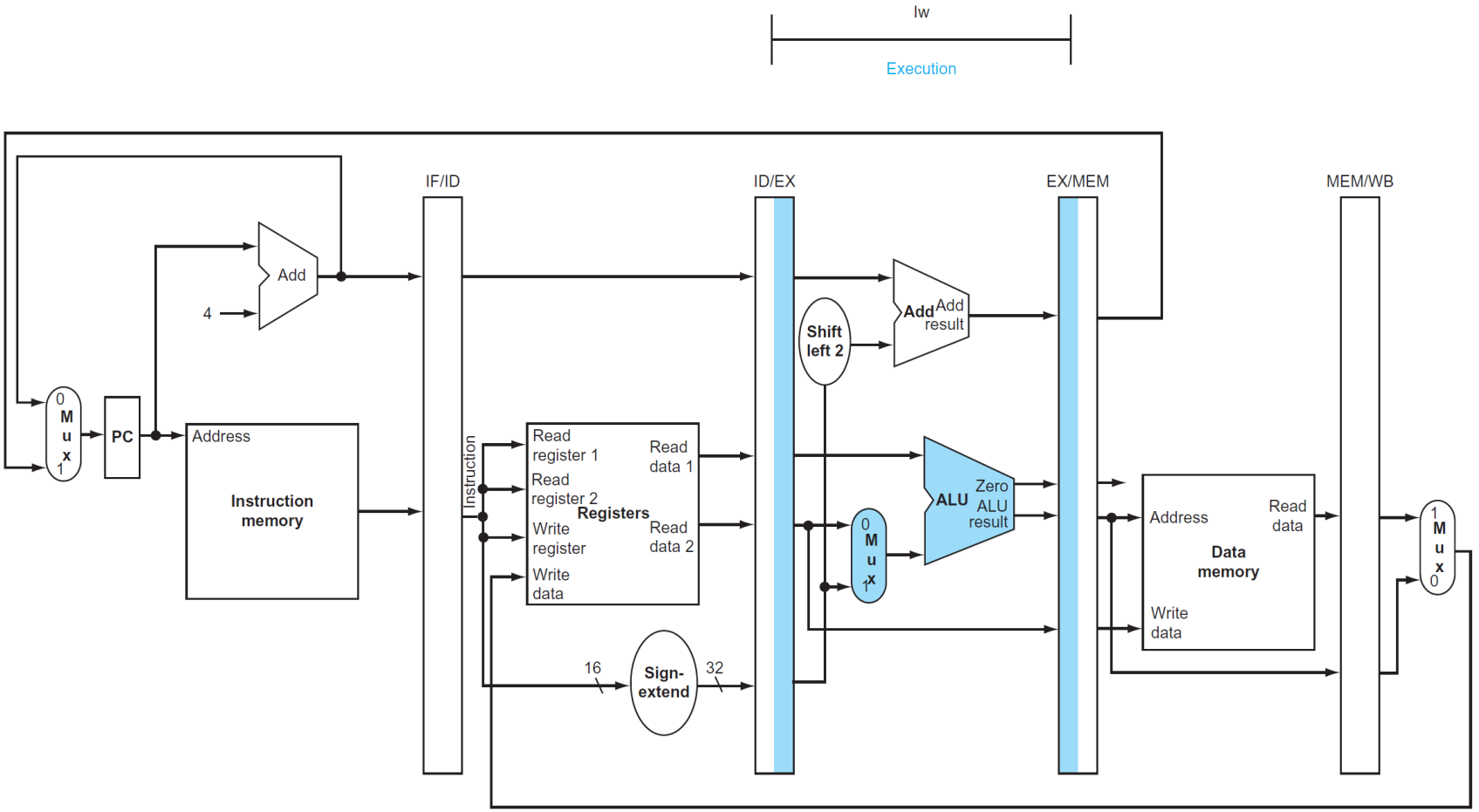
Pipeline Flow: IF



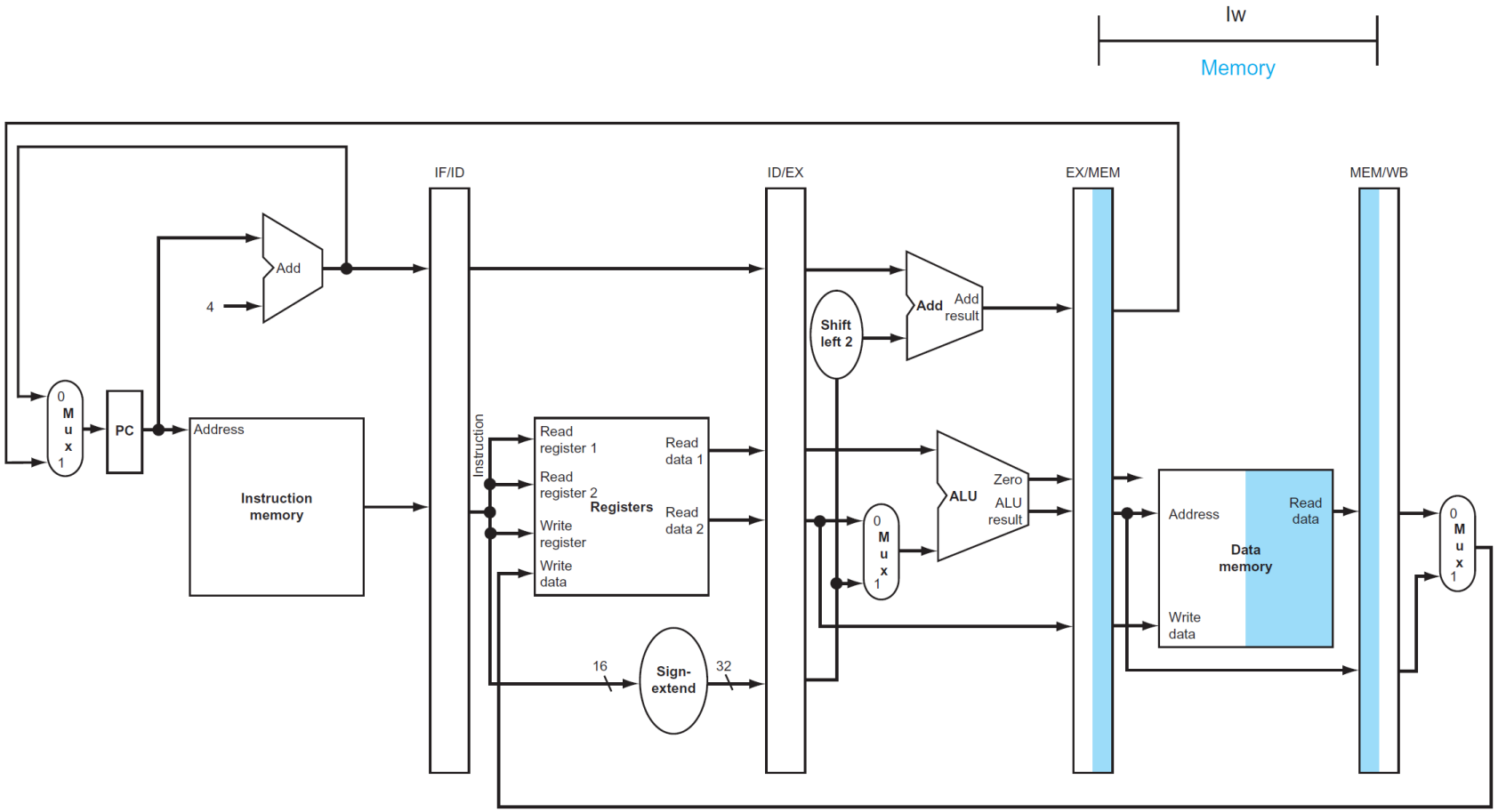
Pipeline Flow: ID



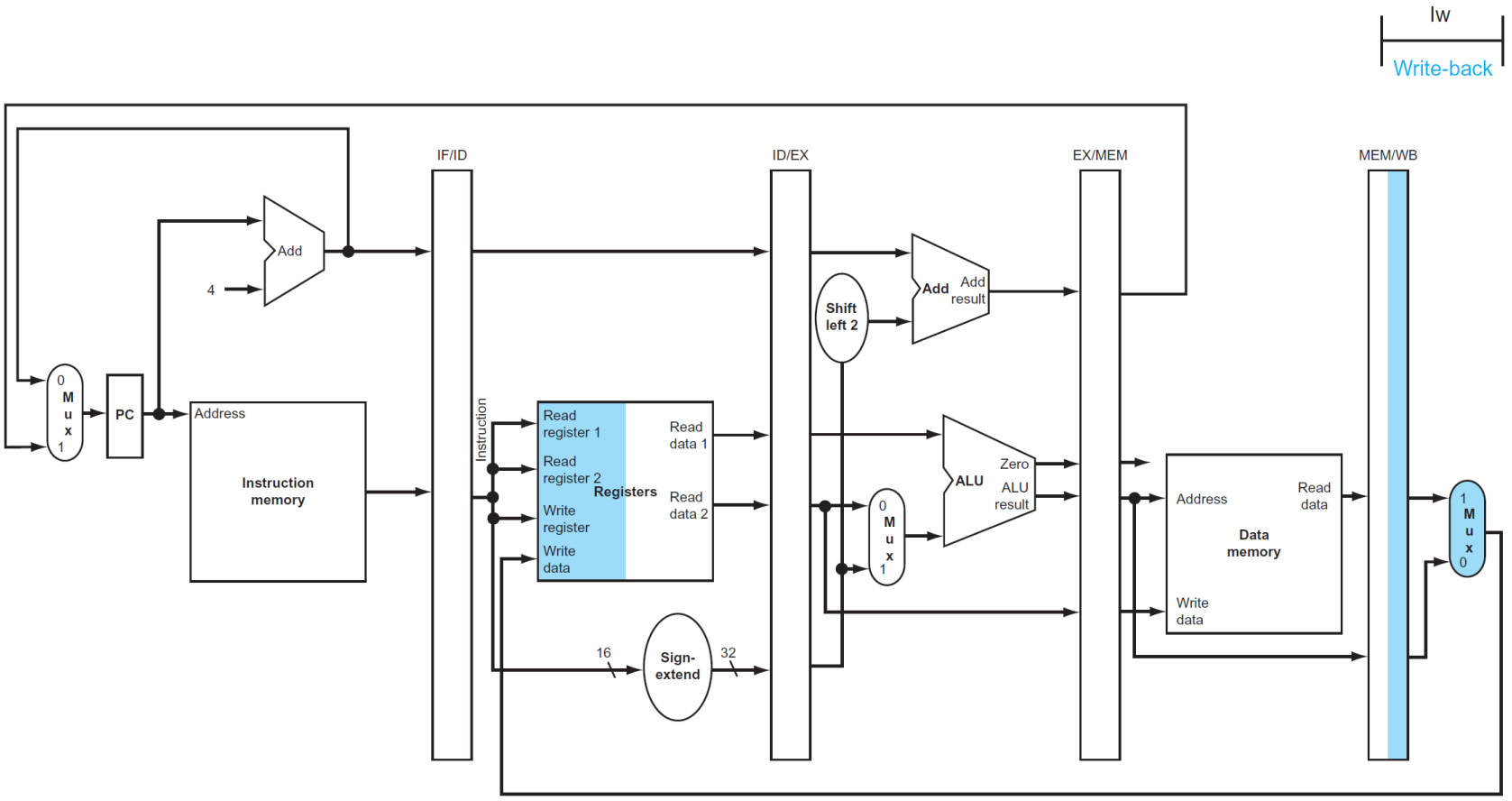
Pipeline Flow: EX



Pipeline Flow: MEM



Pipeline Flow: WB



In extra, write register number is passed from ID to MEM/WB, and then used for defining Write Register at WB

MIPS Approach to Pipelining

- All instructions are the same length
- A few instruction formats are used
- Instruction formats share common structures
- Memory operands appear only in loads and stores

Hazard

- A hazard occurs when the next instruction cannot get executed in the following clock cycle (or, result in anomaly)

- Structural hazard
- Data hazard
- Control hazard

Structural Hazard

- Occurs when two components do not work independently
- Having a single memory instead of two separate ones (data memory and instruction memory)

- Solution: Resource Duplication
  - Separate I and D memories (caches)

Data Hazard

- Occurs when one step must wait for another to complete
  - i.e., the pipeline stalls
- In MIPS, data hazard is possible when the next instruction reads a register written by the earlier instruction in the pipe
  - Ex.
```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

Solution (1)

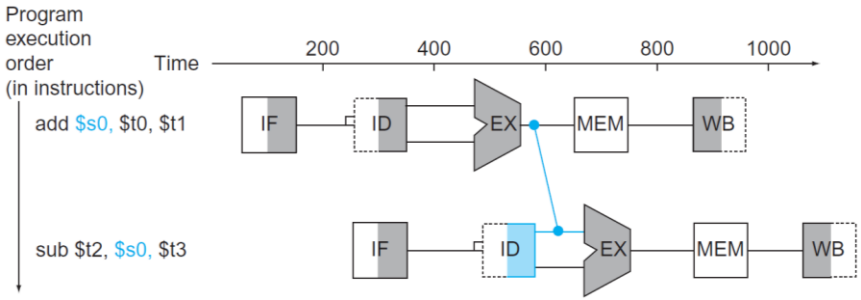
- Instruction re-ordering (via compiler)
  - Ex. `a = b + e;`
    `c = b + f;`

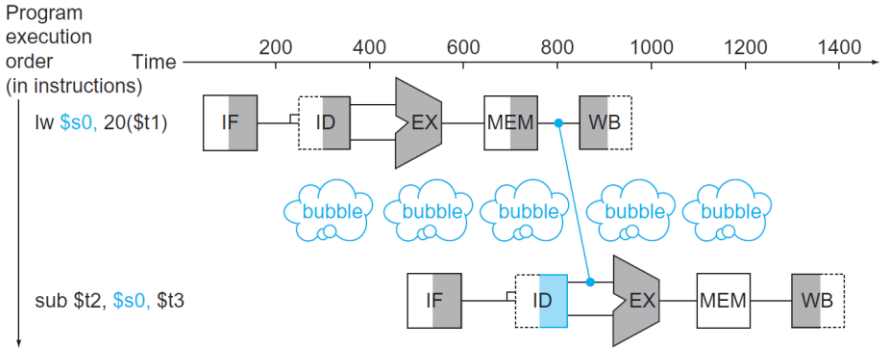
```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add   $t3, $t1,$t2
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add   $t5, $t1,$t4
sw    $t5, 16($t0)
```

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

Solution (2)

- Instruction re-ordering (via compiler)
- Forwarding
  - add extra hardware to retrieve the needed data early





Control Hazards

- Occurs when a decision must be made based on the result of an earlier execution in the pipe
- Solutions
  - stall on branch
  - branch predict