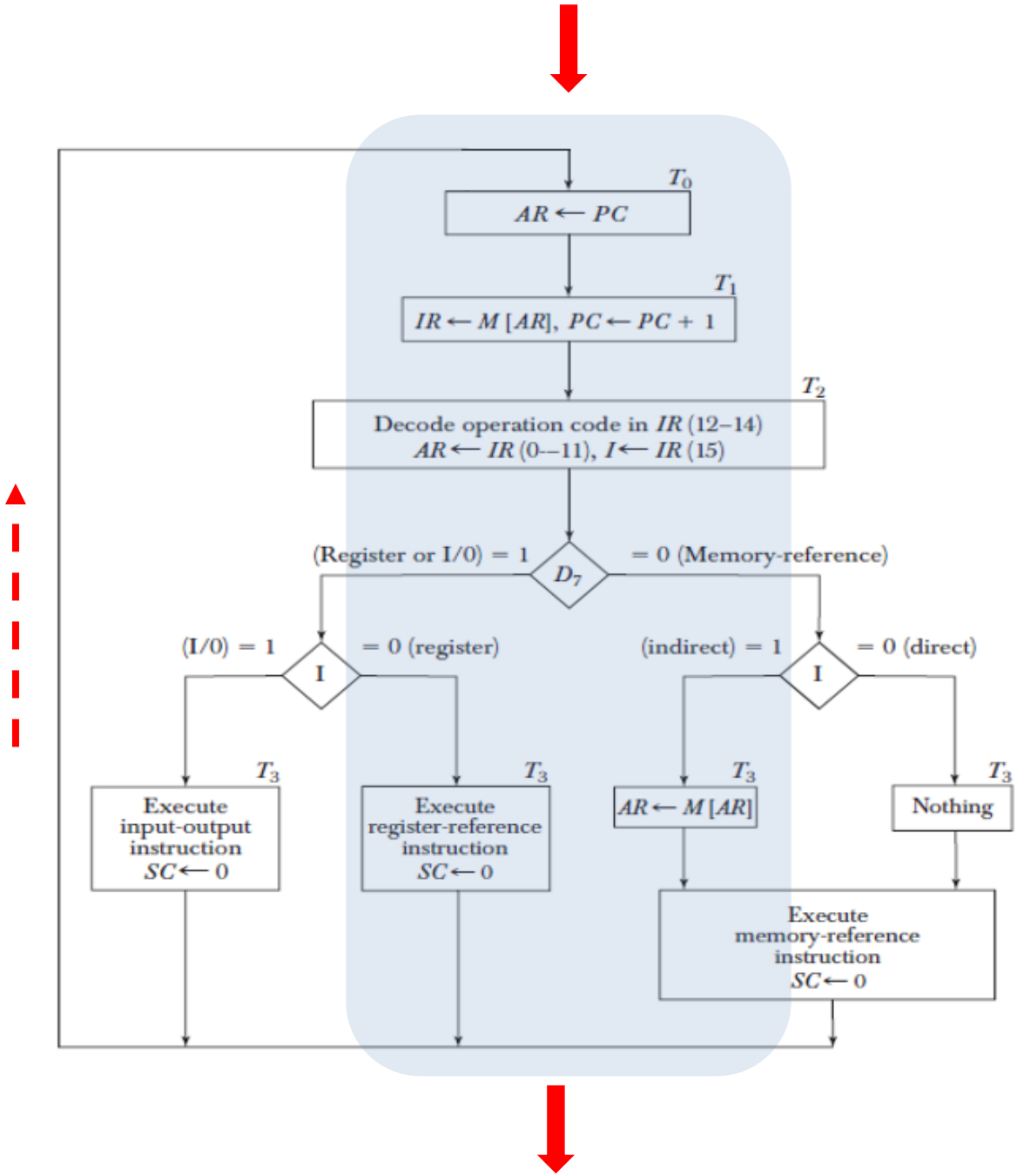
به نام خدا

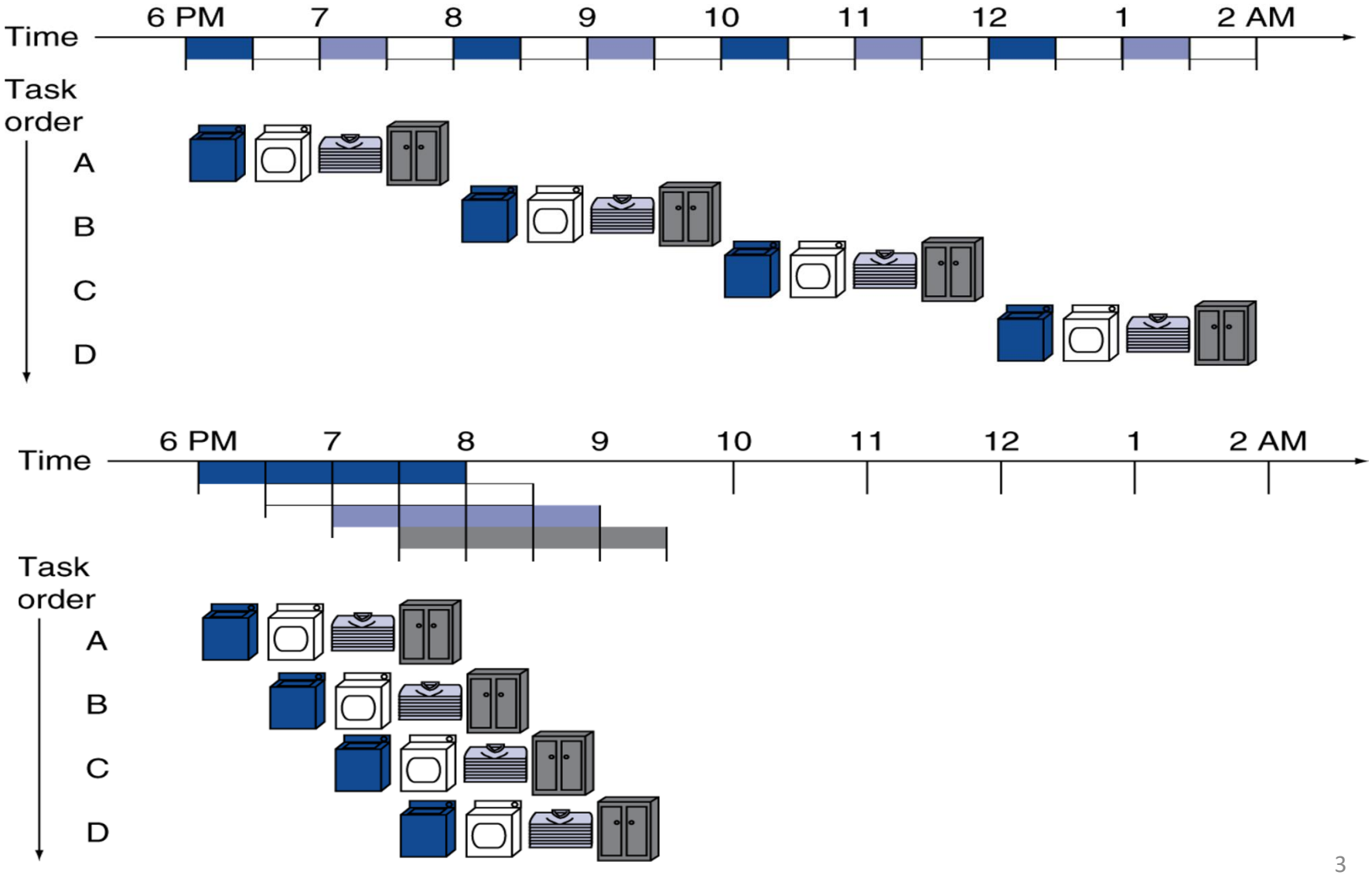
بهبودکارایی پردازنده

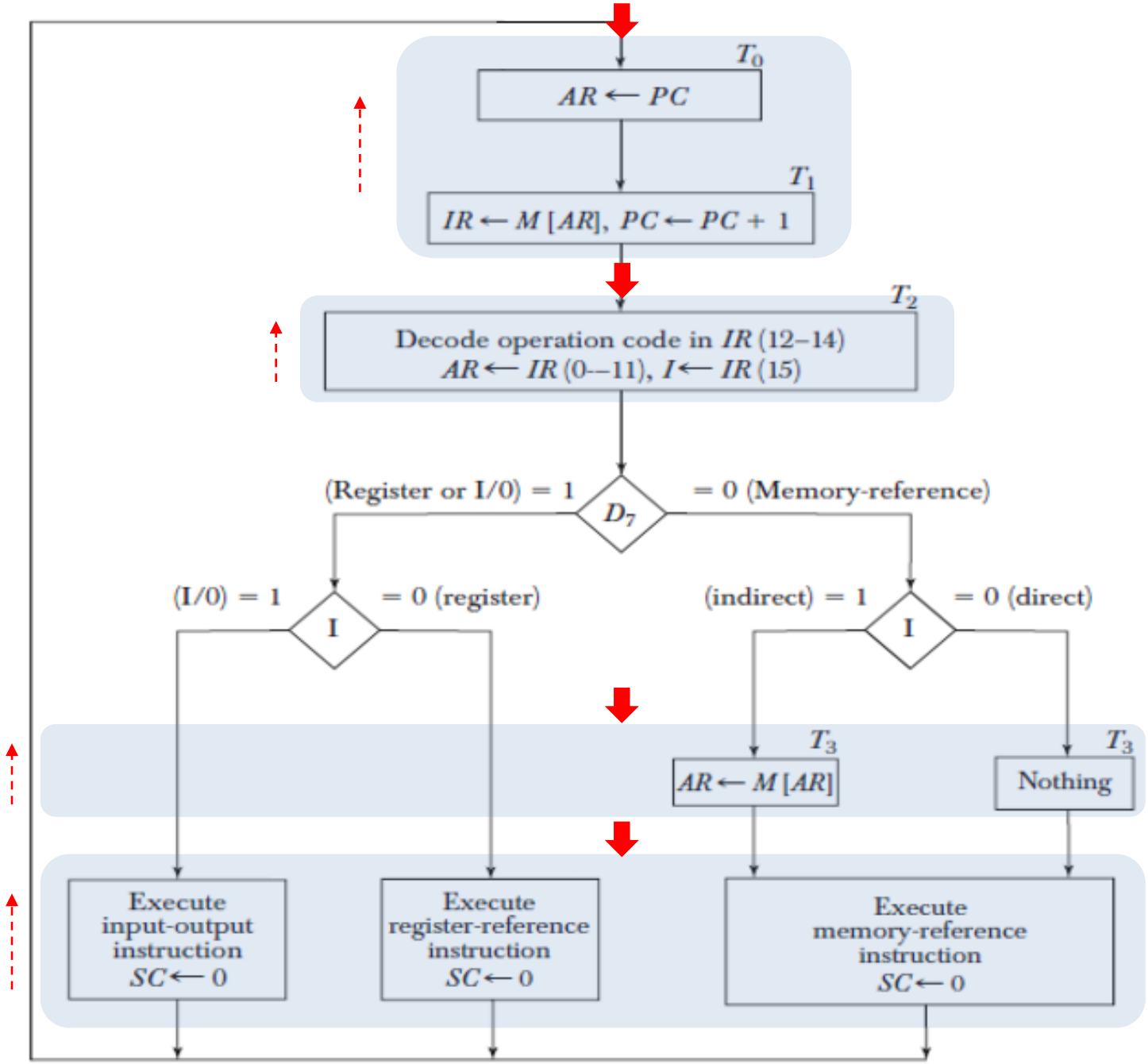
Pipeline

Dr. Aref Karimiafshar
A.karimiafshar@iut.ac.ir


T_0
$AR \leftarrow PC$
T_1
$IR \leftarrow M[AR],\ PC \leftarrow PC + 1$
T_2
Decode operation code in IR (12–14)
$AR \leftarrow IR$ (0–11), $I \leftarrow IR$ (15)
(Register or I/O) = 1
D_7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T_3
Execute input-output instruction
$SC \leftarrow 0$
T_3
Execute register-reference instruction
$SC \leftarrow 0$
T_3
$AR \leftarrow M[AR]$
T_3
Nothing
Execute memory-reference instruction
$SC \leftarrow 0$

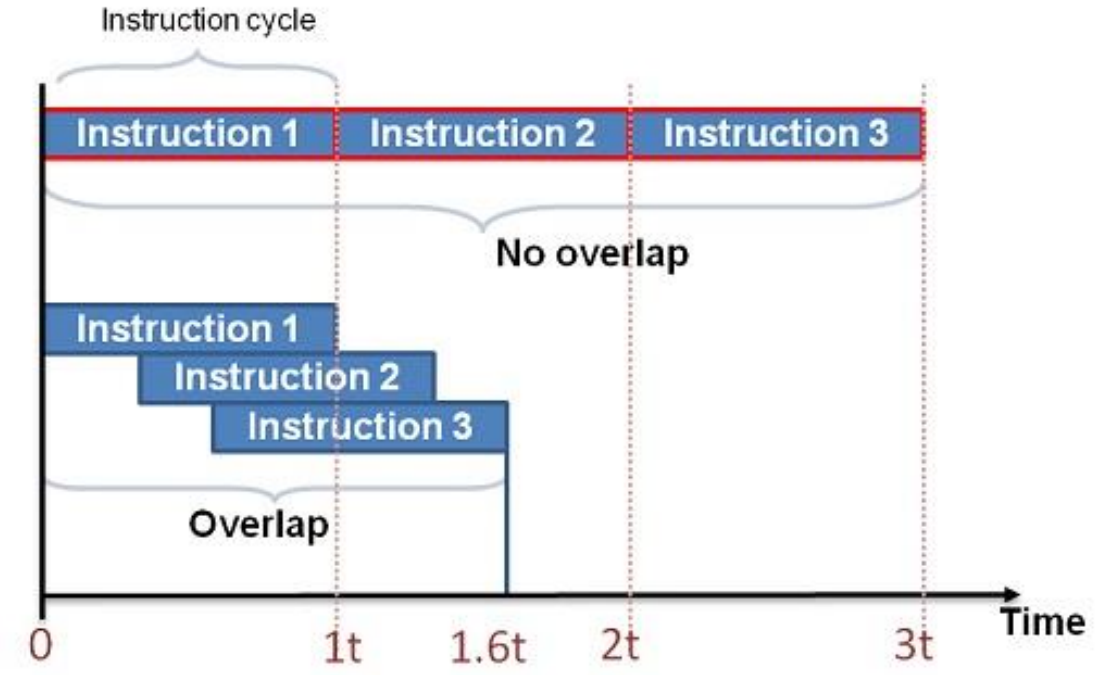
Pipelining Analogy

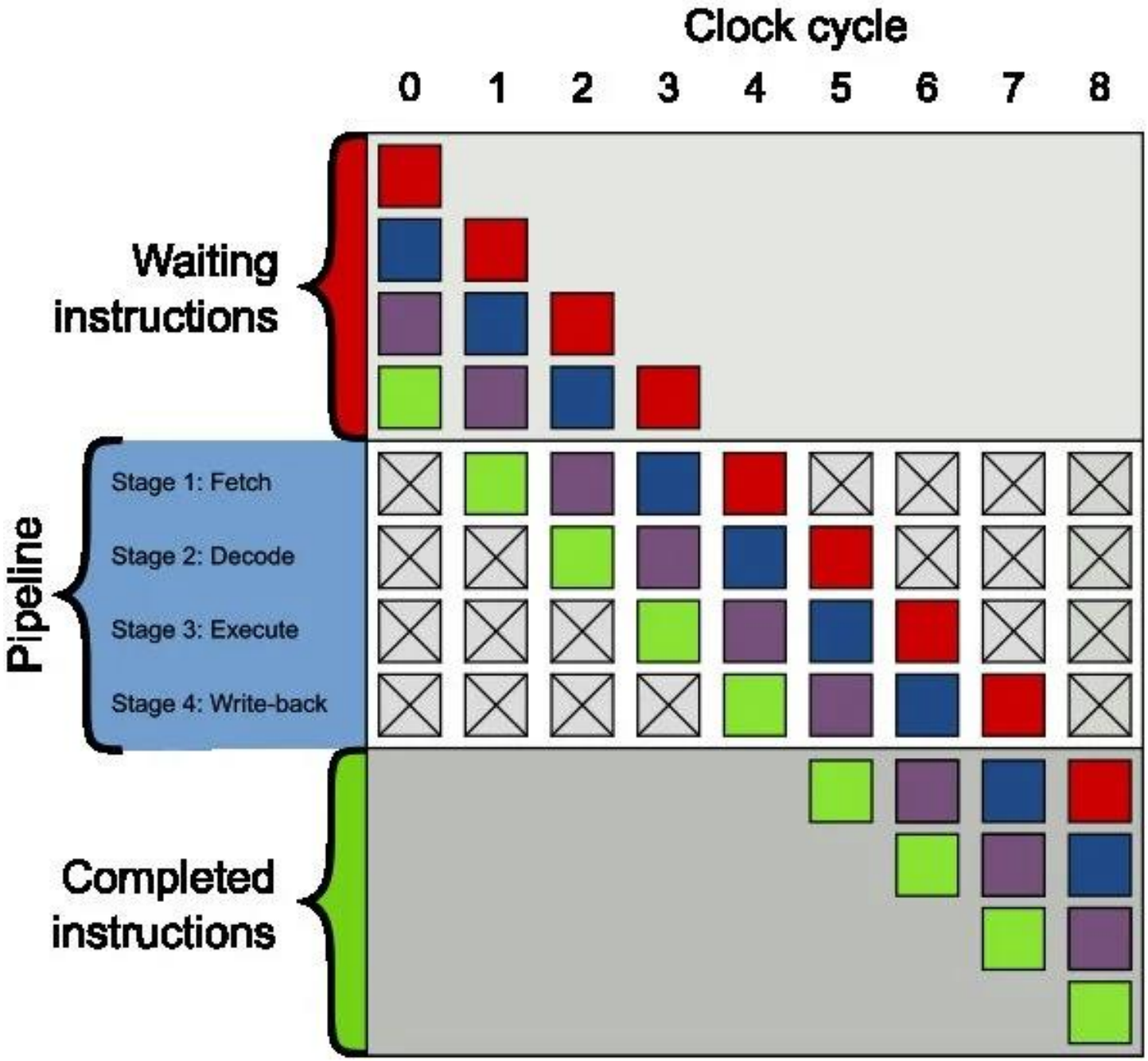



T_0
$AR \leftarrow PC$
T_1
$IR \leftarrow M[AR], PC \leftarrow PC + 1$
T_2
Decode operation code in IR (12–14)
$AR \leftarrow IR$ (0–11), $I \leftarrow IR$ (15)
(Register or I/0) = 1
D_7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T_3
$AR \leftarrow M[AR]$
T_3
Nothing
Execute input-output instruction $SC \leftarrow 0$
Execute register-reference instruction $SC \leftarrow 0$
Execute memory-reference instruction $SC \leftarrow 0$

What is Pipelining?

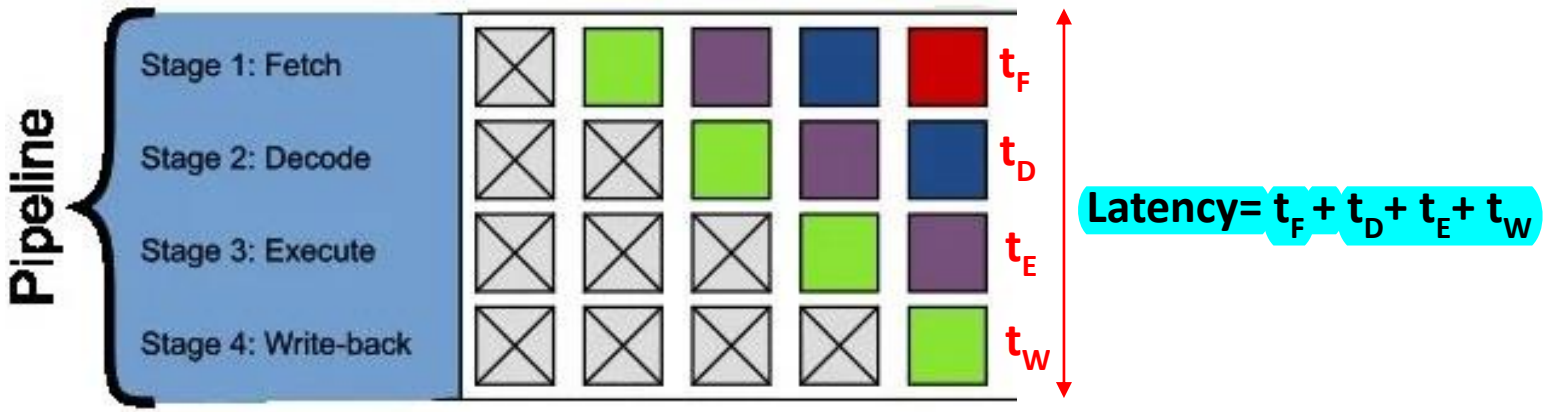
- A way of speeding up execution of instructions
- Key idea:
  - Overlap execution of multiple instructions




Clock cycle
0
1
2
3
4
5
6
7
8
Waiting instructions
Pipeline
Stage 1: Fetch
Stage 2: Decode
Stage 3: Execute
Stage 4: Write-back
Completed instructions

Latency of a Pipeline

- The latency of a pipeline is defined as the Time required for an instruction to propagate through the pipeline
  - Its unit is time unit like microseconds, nanoseconds

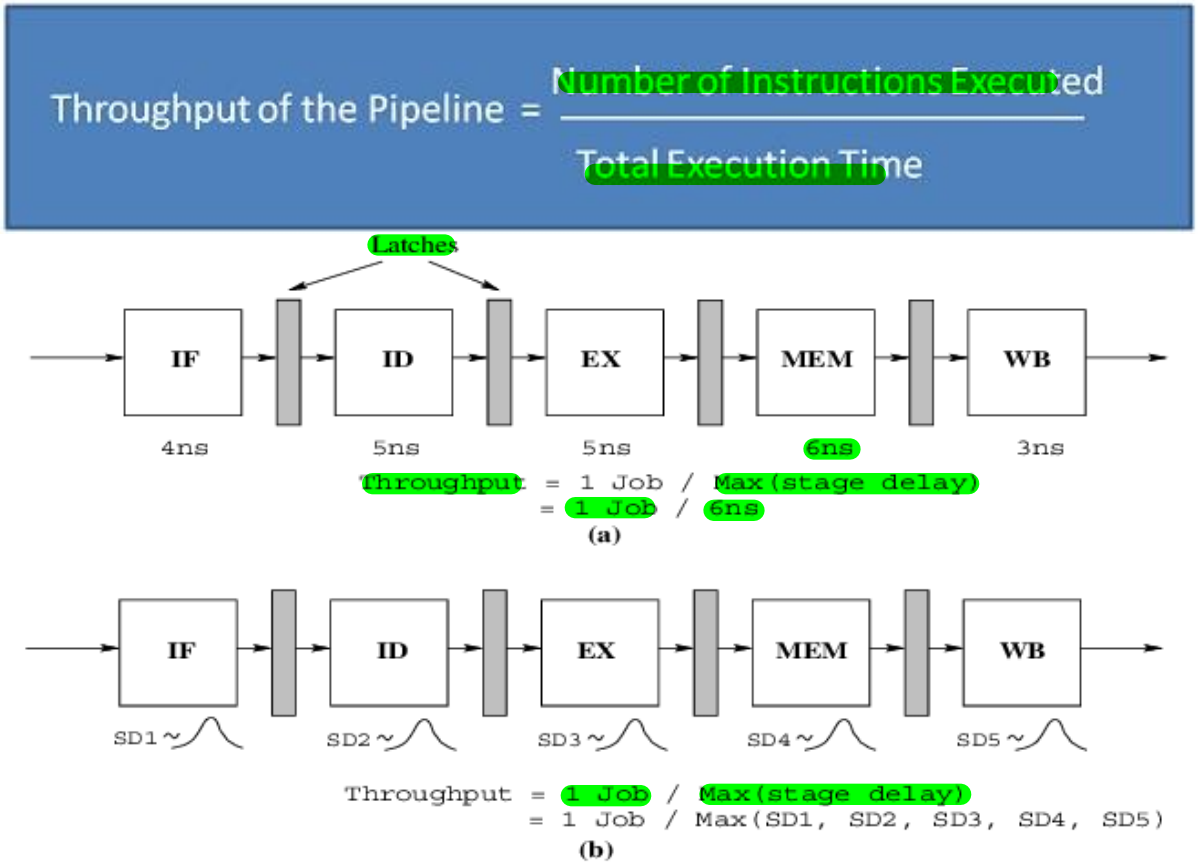


IF $t_F = t_D = t_E = t_W$ then:

Latency = Number of pipeline Stages (m) x Cycle Time (T)

Throughput of a pipeline

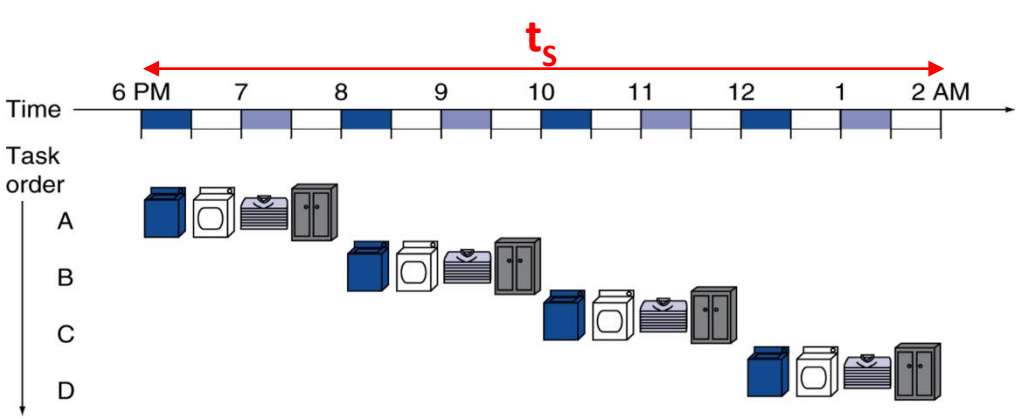
- Throughput of a pipeline is the rate at which instructions can start and finish
  - i.e. the number of instructions finished per second

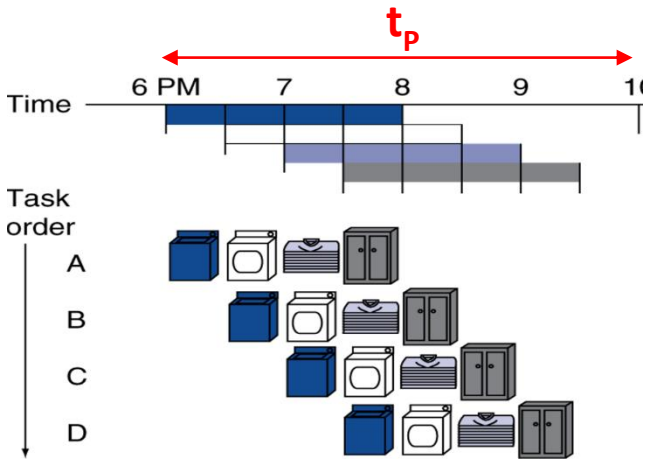


Speedup

- Speedup is a quantitative measure of the performance achievement

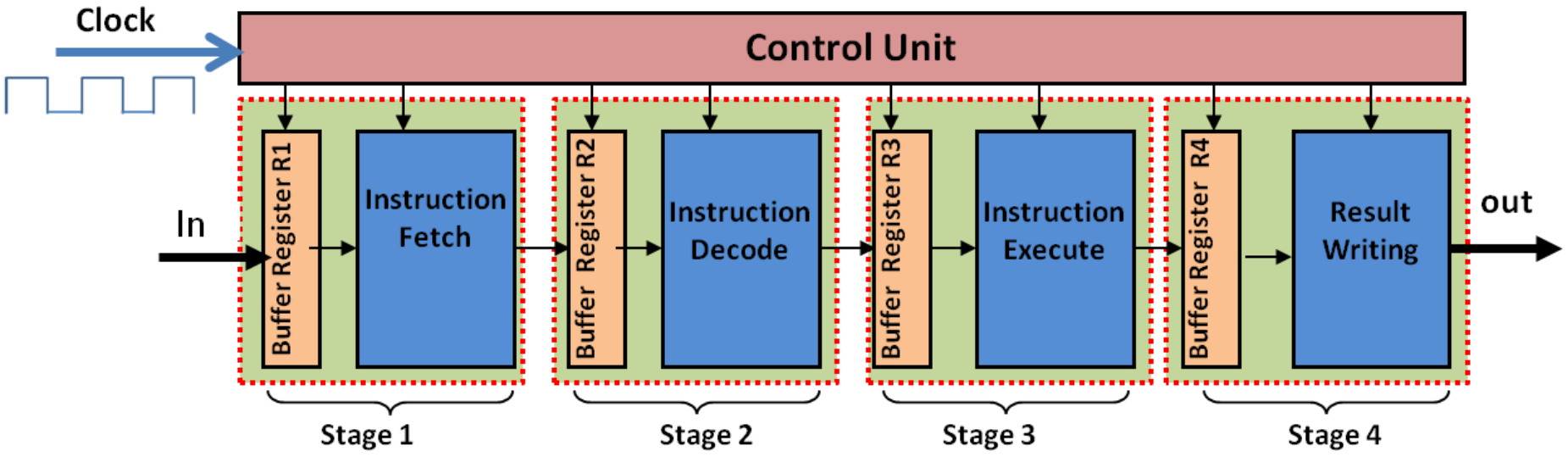
$$\text{Speed Up achieved due to Pipeline} = \frac{\text{Non Pipelined Execution Time}}{\text{Pipelined Execution Time}}$$



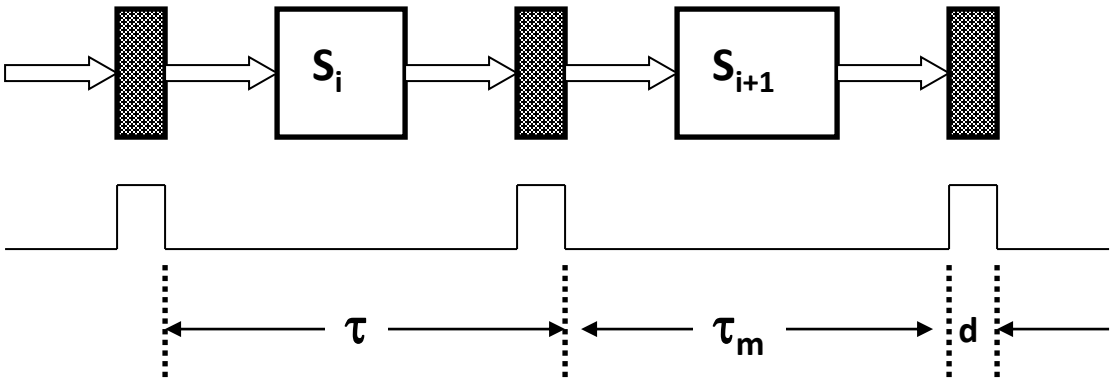


$$\textbf{Speed Up} = \frac{t_s}{t_p} = \frac{8}{3.5} = 2.3$$

Four Stage Instruction Pipeline



Pipeline Clock and Timing



Clock cycle of the pipeline : τ

Latch delay : d

$$\tau = \max\{\tau_m\} + d$$

Pipeline frequency : f

$$f = 1 / \tau$$

Speedup and Efficiency

- *k-stage* pipeline processes *n* tasks in k + (n-1) clock cycles:
  - *k* cycles for the first task
  - *n-1* cycles for the remaining *n-1* tasks
- Total time to process *n* tasks

$$T_k = [\, k + (n-1)] \, \tau$$

- For the non-pipelined processor

$$T_1 = n \, k \, \tau$$

- Speedup factor

$$S_k = \frac{T_1}{T_k} = \frac{n \, k \, \tau}{[\, k + (n-1)] \, \tau} = \frac{n \, k}{k + (n-1)}$$

Pipeline Performance: Example

- Task has 4 subtasks with time: t1=60, t2=50, t3=90, and t4=80 ns (nanoseconds)
- latch delay = 10
- Pipeline cycle time = 90+10 = 100 ns
- For non-pipelined execution
  - time = 60+50+90+80 = 280 ns
- Speedup for above case is: 280/100 = 2.8 !!
- Pipeline Time for 1000 tasks = 1000 + 4-1= 1003*100 ns
- Sequential time = 1000*280ns
- Throughput= 1000/1003

Pipeline Hazards

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: two different instructions use same h/w in same cycle
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Pipelining of branches & other instructions that change the PC

MIPS Pipeline

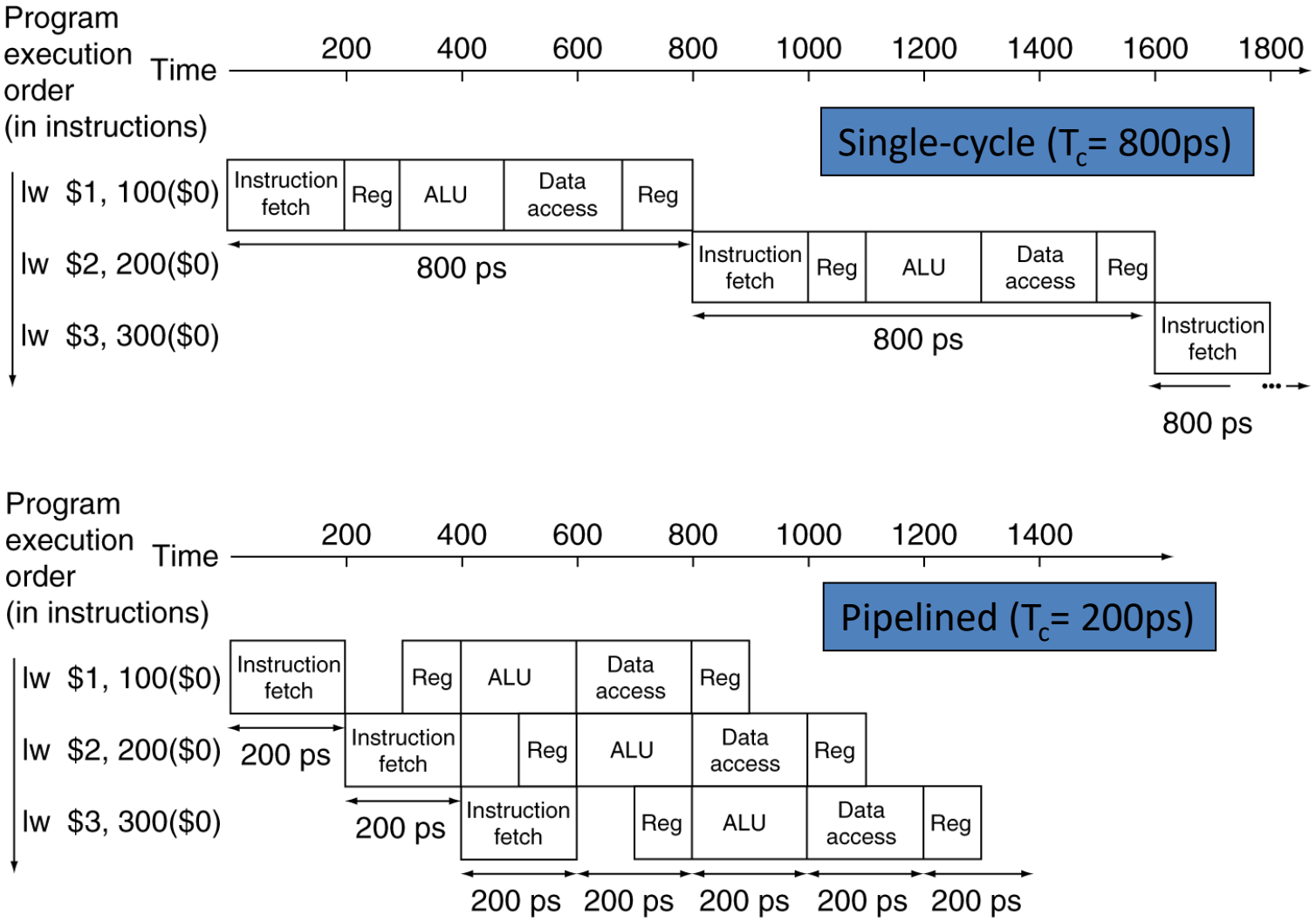
- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register

Pipeline Performance

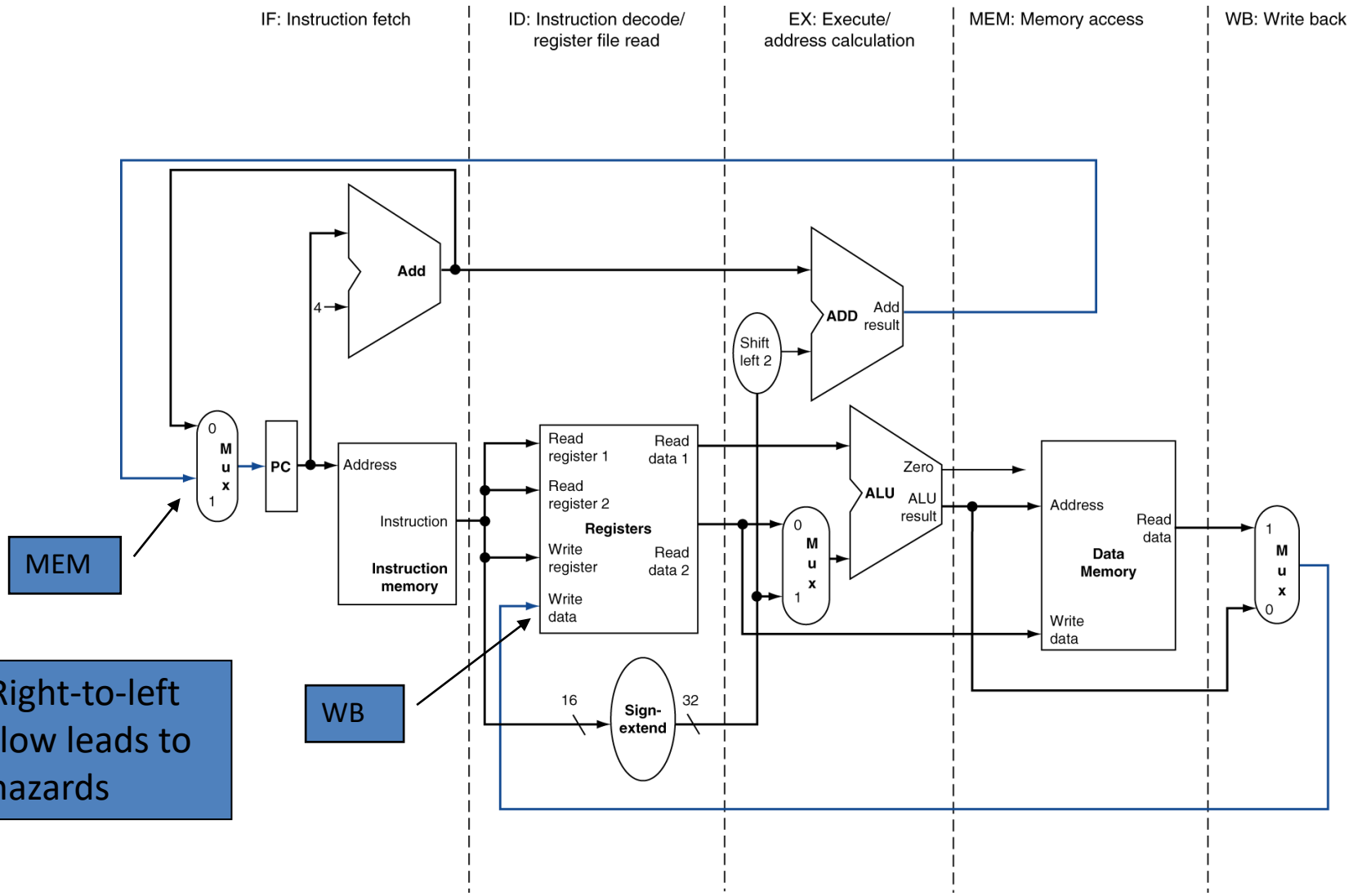
- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance

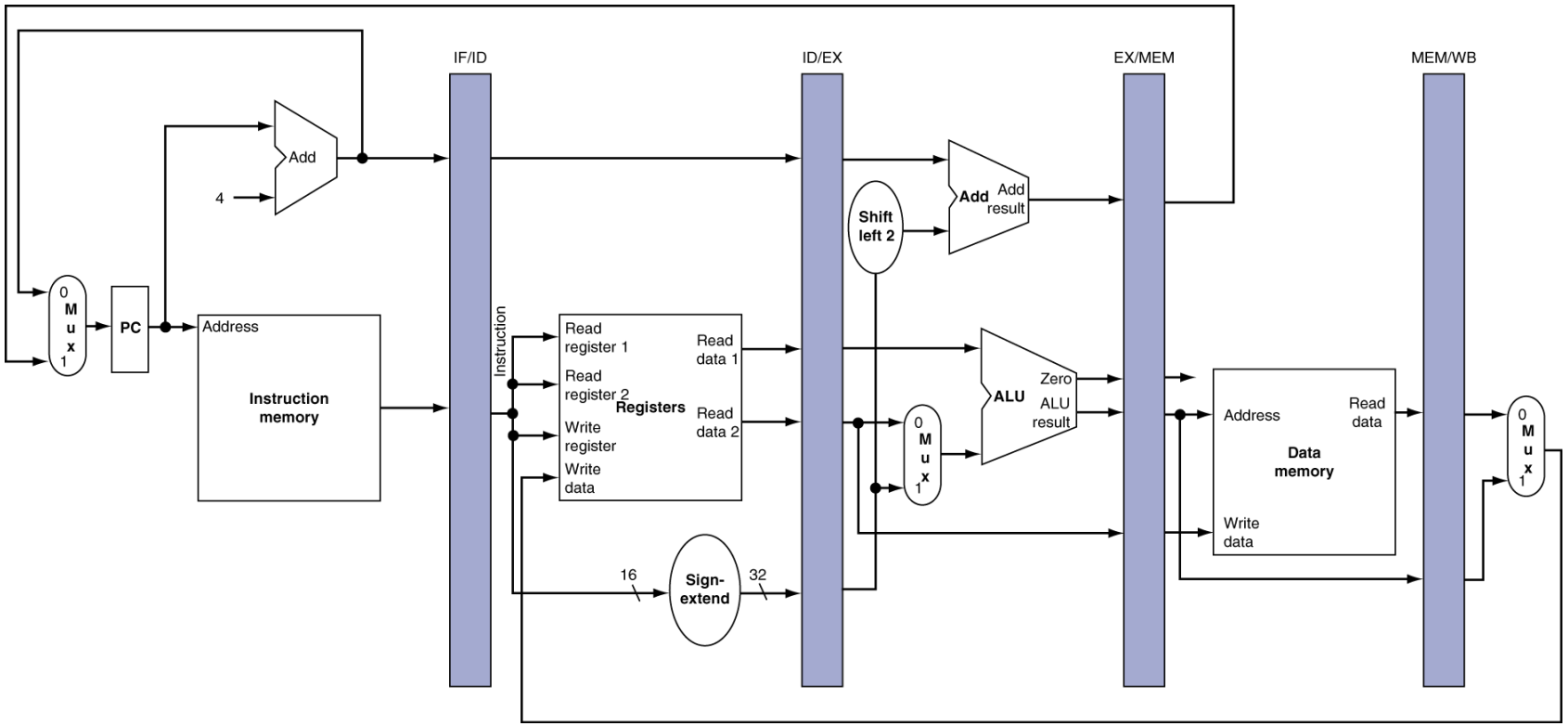


MIPS Pipelined Datapath



Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle



Comments about Pipelining

- The good news
  - Multiple instructions are being processed at same time
  - This works because stages are isolated by registers
  - Best case speedup of N
- The bad news
  - Instructions interfere with each other - hazards
- Example: different instructions may need the same piece of hardware (e.g., memory) in same clock cycle
- Example: instruction may require a result produced by an earlier instruction that is not yet complete

پایان

موفق و پیروز باشید