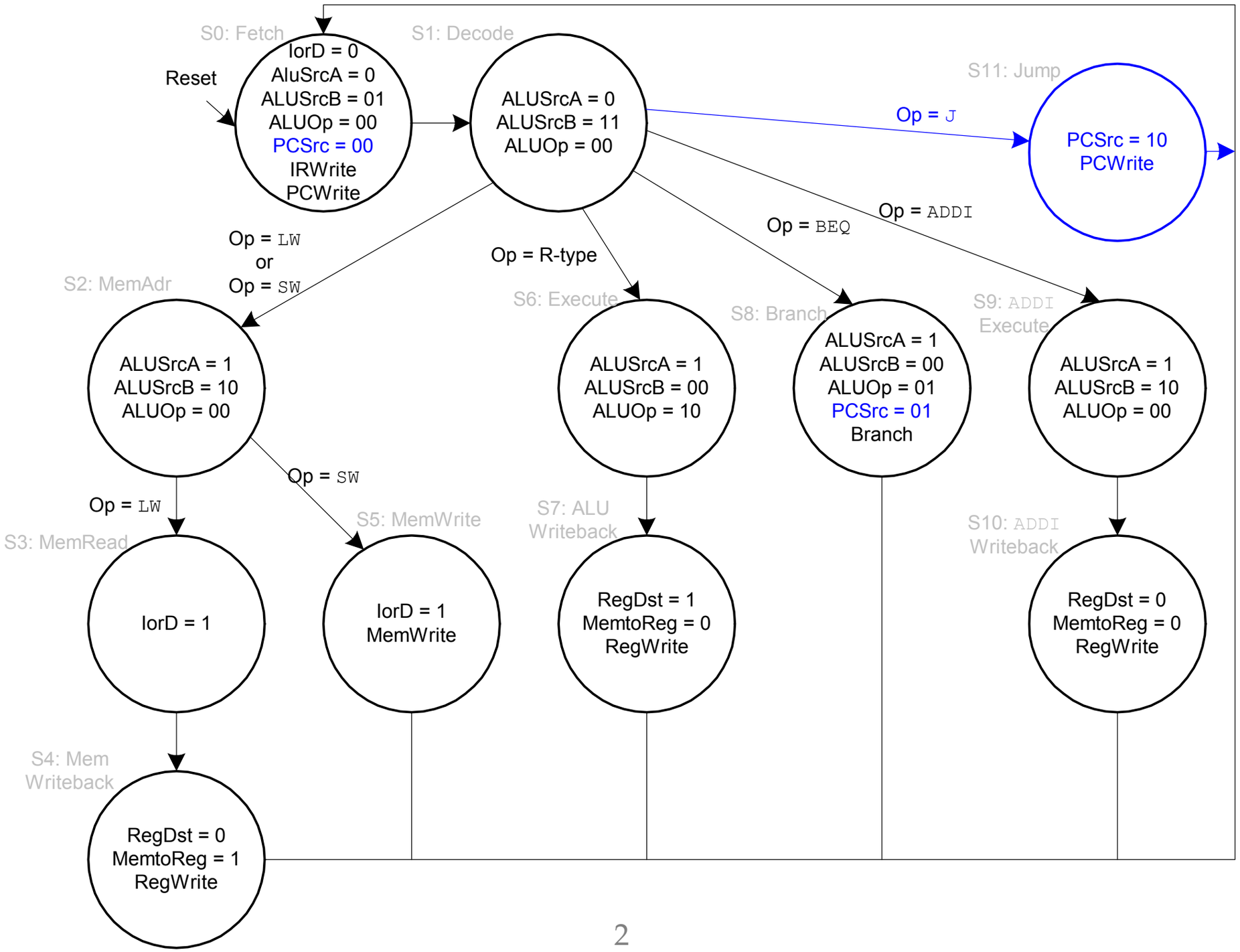
EECE 2322: Fundamentals of Digital Design and Computer Organization

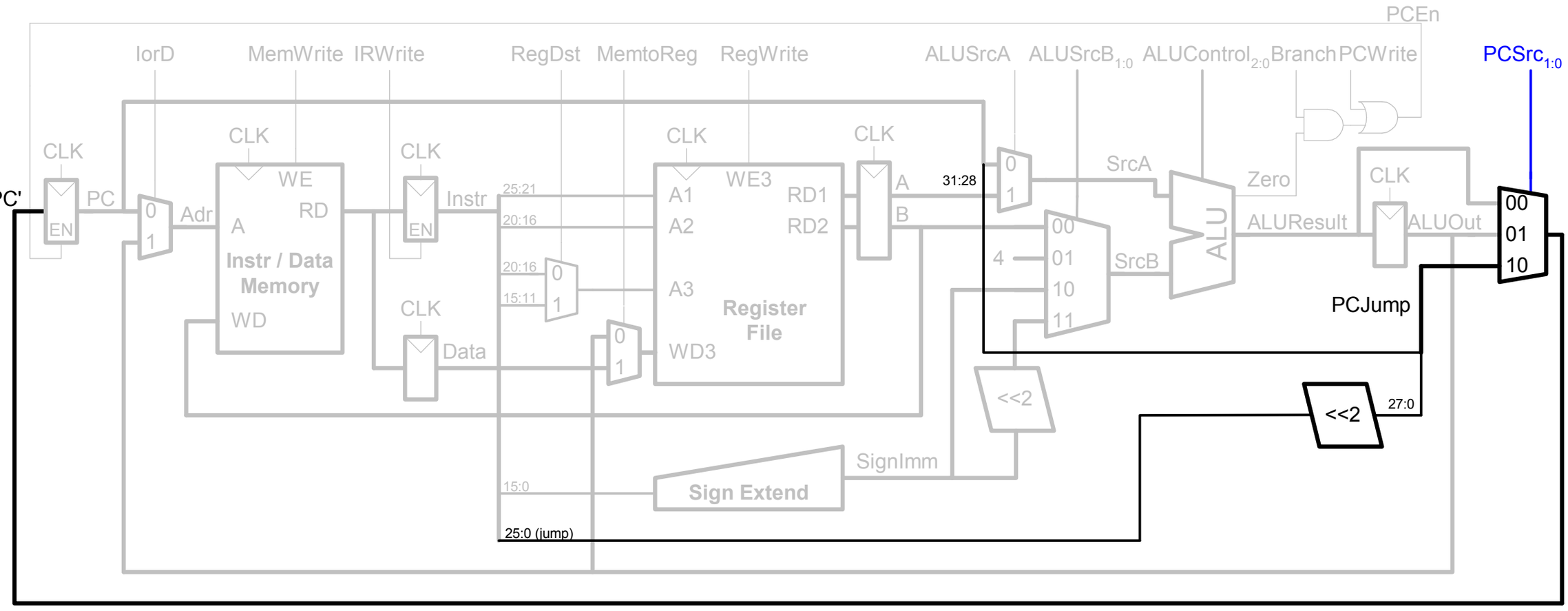
Lecture 13_3: Microarchitecture

Xiaolin Xu
Department of ECE
Northeastern University

Main Controller FSM: j



Extended Functionality: j



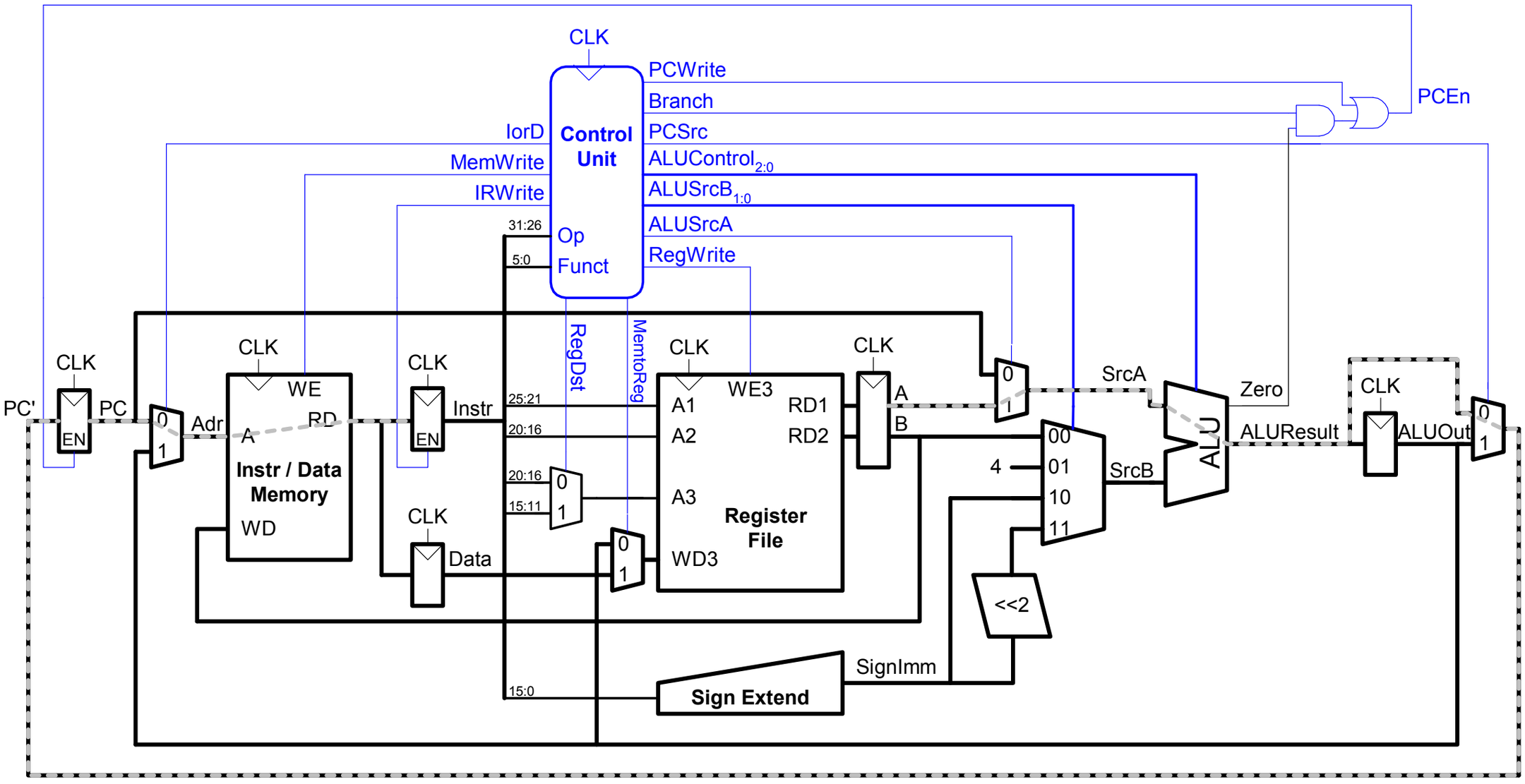
Multicycle Processor Performance

- Instructions take different number of cycles:
  - 3 cycles: `beq`, `j`
  - 4 cycles: R-Type, `sw`, `addi`
  - 5 cycles: `lw`
- CPI is weighted average
- SPECINT2000 benchmark
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- **Average CPI** = (0.11 + 0.2)(3) + (0.52 + 0.10)(4) + (0.25)(5) = **4.12**

Multicycle Processor Performance

Multicycle critical path:

$$T_c = t_{pcq} + t_{\text{mux}} + \max(t_{\text{ALU}} + t_{\text{mux}}, t_{\text{mem}}) + t_{\text{setup}}$$



Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	$t_{RF\text{read}}$	150
Register file setup	$t_{RF\text{setup}}$	20

$$T_c = t_{pcq\_PC} + t_{\text{mux}} + \max(t_{\text{ALU}} + t_{\text{mux}}, t_{\text{mem}}) + t_{\text{setup}}$$
$$= t_{pcq\_PC} + t_{\text{mux}} + t_{\text{mem}} + t_{\text{setup}}$$
$$= [30 + 25 + 250 + 20] \text{ ps}$$
$$= \mathbf{325 \text{ ps}}$$

Multicycle Performance Example

- For a program with 100 billion instructions executing on a multicycle MIPS processor
  - CPI = 4.12
  - $T_c = 325$ ps

Execution Time =

Multicycle Performance Example

- For a program with 100 billion instructions executing on a multicycle MIPS processor
  - CPI = 4.12
  - $T_c = 325$ ps

Execution Time = (# instructions) × CPI × T_c

= $(100 \times 10^9)(4.12)(325 \times 10^{-12})$

= 133.9 seconds

This is slower than the single-cycle processor (92.5 seconds). Why?

Multicycle Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) × CPI × T_c

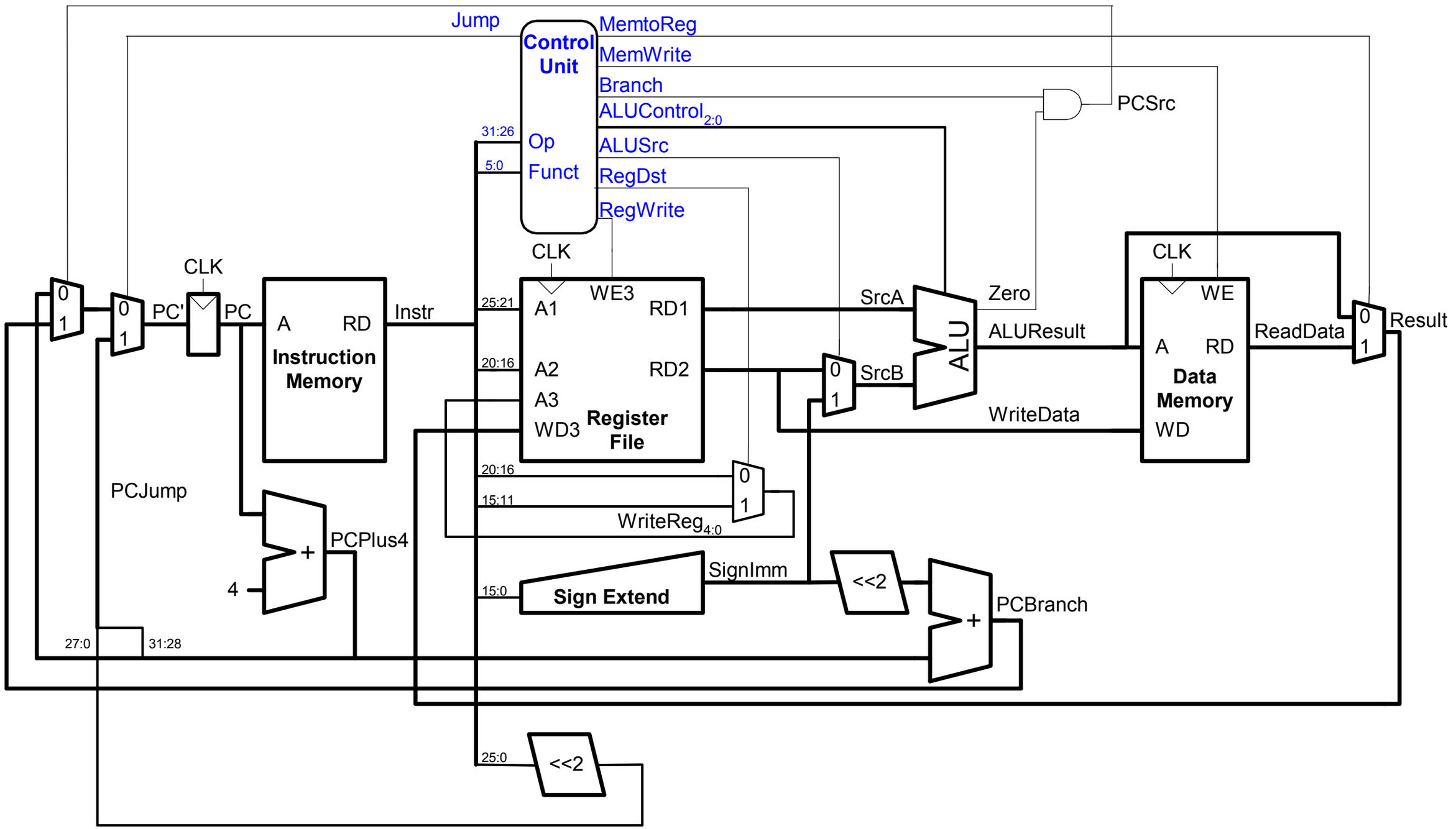
$= (100 \times 10^9)(4.12)(325 \times 10^{-12})$

$= 133.9$ seconds

This is **slower** than the single-cycle processor (92.5 seconds). Why?

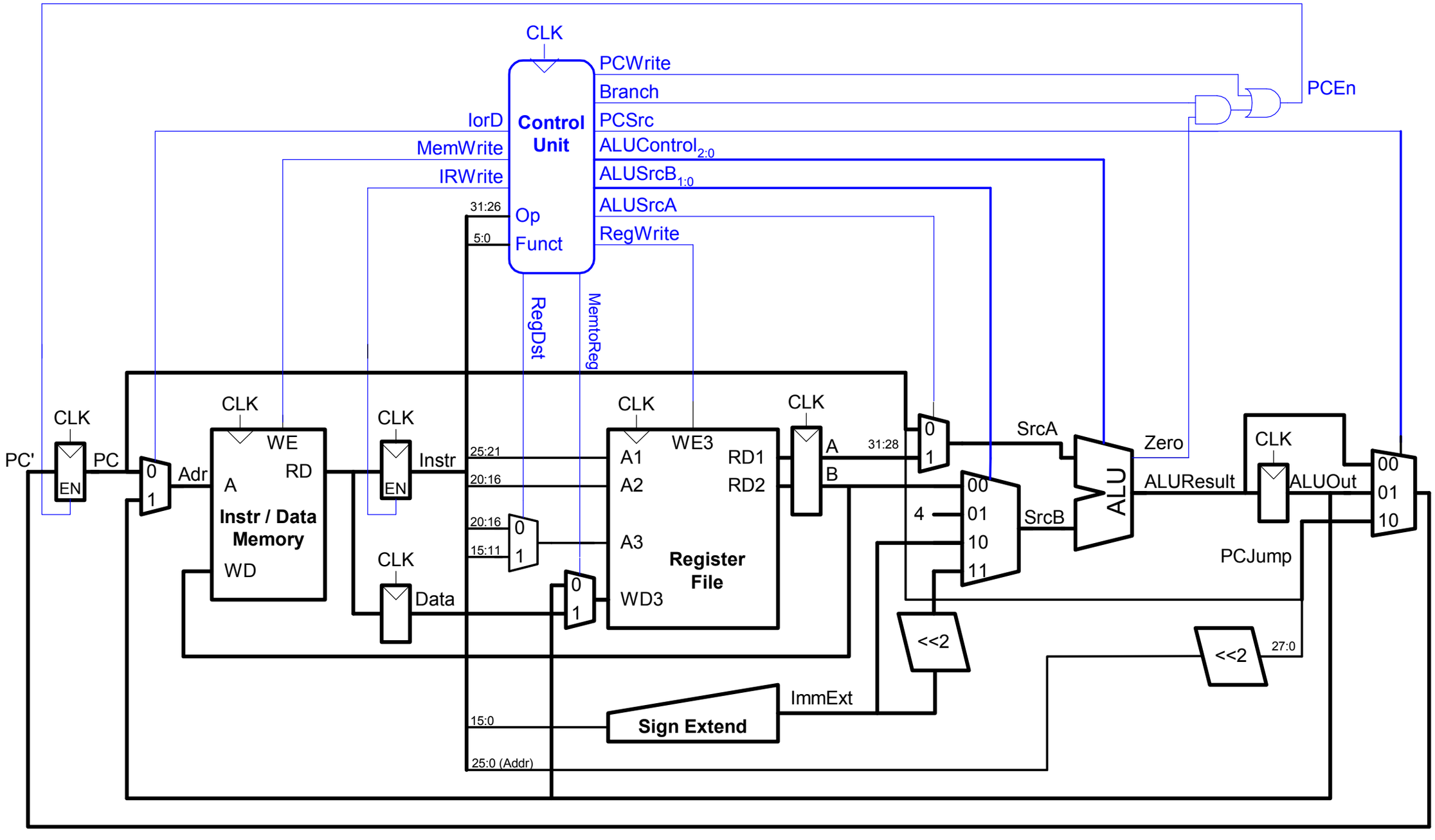
Not all steps same length
Sequencing overhead for each step ($t_{pcq} + t_{\text{setup}} = 50$ ps)

Review: Single-Cycle Processor



Clock cycle should be long enough to fit the slowest instruction, e.g., lw

Review: Multicycle Processor



Single Cycle and Multiple Cycle Datapath: Difference

- Single Cycle Datapaths
  - Data memory has only one Address input
  - Actual memory operation can be determined from the MemRead and MemWrite control signals
  - Separate memories for instructions and data
  - 2 adders for PC-based computations and one ALU

Single Cycle and Multiple Cycle Datapath: Difference

- Multiple Cycle Datapaths
  - Break instructions into separate steps
  - Each step takes a single clock cycle
  - Each functional unit can be used more than once in an instruction, as long as it is used in different clock cycles
  - Reduces the amount of hardware needed
  - **Reduces average instruction time**

Single Cycle and Multiple Cycle Datapath: Difference

S.No.	Single Cycle Datapath	Multiple Cycle Datapath
1	Instructions are not subdivided.	Instructions are divided into arbitrary number of steps.
2	Clock cycles are long enough for the lowest instruction.	Clock cycles are short but long enough for the lowest instruction.
3	There are only 1 instruction that can be executed at the same time.	There are only 1 instruction that can be executed at the same time.
4	There is 1 cycle per instruction, i, e., CPI = 1.	There is a variable number of clock cycles per instructions.
5	Control unit generates signals for the entire instruction.	Control unit generates signals for the instruction's current step and keeps track of the current step.
6	There is duplicate hardware, because we can use a functional unit for at most one subtask per instruction.	There is no duplicate hardware, because the instructions generally are broken into single FU steps.
7	Extra registers are not required.	Extra registers are required to hold the result of one step for use in the next step.
8	Performance is baseline.	Performance is slightly slower to moderately faster than single cycle, latter when the instructions steps are well balanced and a significantly fractions of the instructions take less than the maximum number of cycles.

Parallelism

- **Two types of parallelism:**
  - **Spatial parallelism**
    - duplicate hardware performs multiple tasks at once
  - **Temporal parallelism**
    - task is broken into multiple stages
    - also called pipelining
    - for example, an assembly line

Parallelism Definitions

- **Token:** Group of inputs processed to produce group of outputs
- **Latency:** Time for one token to pass from start to end
- **Throughput:** Number of tokens produced per unit time

Parallelism increases throughput

Parallelism Example

- Ben bakes cookies
  - 5 minutes to roll cookies
  - 15 minutes to bake
- What is the latency and throughput without parallelism?

Parallelism Example

- Ben bakes cookies
  - 5 minutes to roll cookies
  - 15 minutes to bake
- What is the latency and throughput without parallelism?

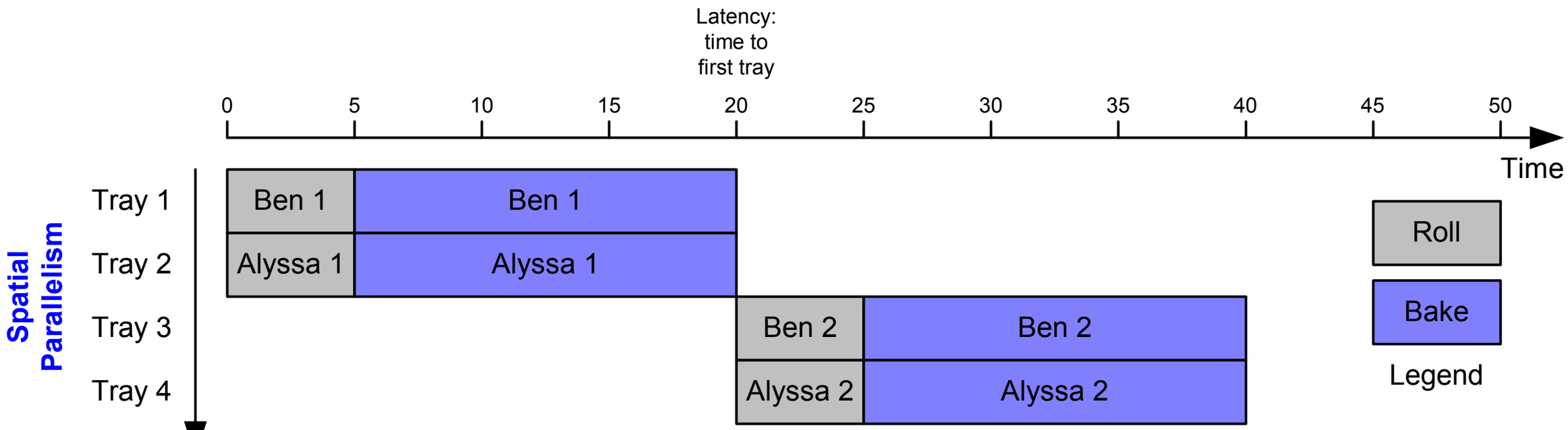
Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 1 tray/ 1/3 hour = **3 trays/hour**

Parallelism Example

- What is the latency and throughput if using parallelism?
  - **Spatial parallelism:** Using another oven from Alyssa
  - **Temporal parallelism:**
    - Two stages: rolling and baking
    - Using two trays
    - While first batch is baking, rolling the second batch, etc.

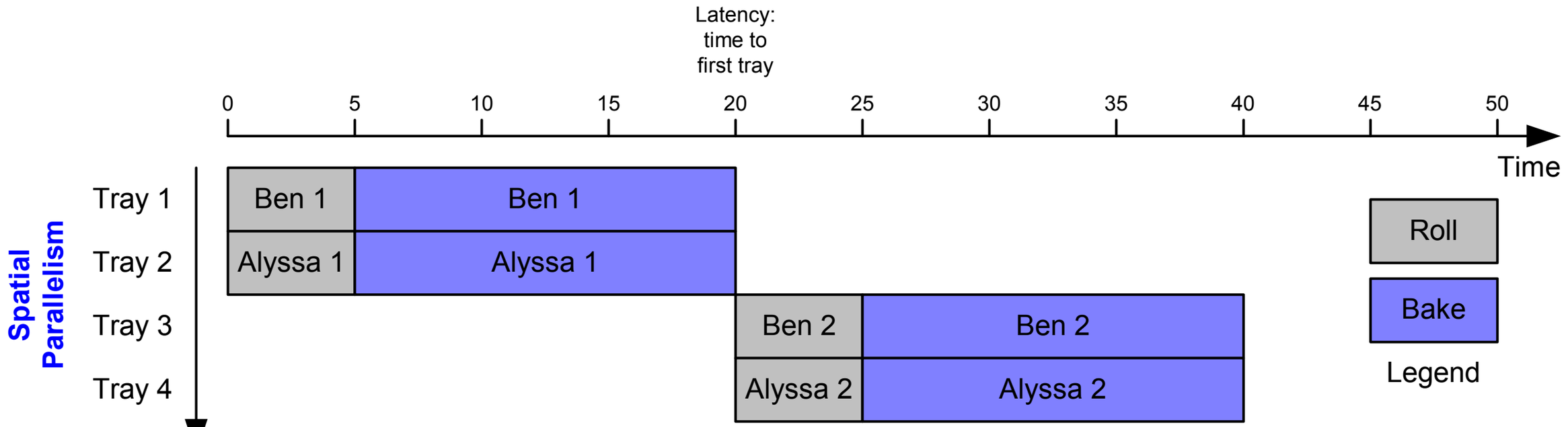
Spatial Parallelism



Latency = ?

Throughput = ?

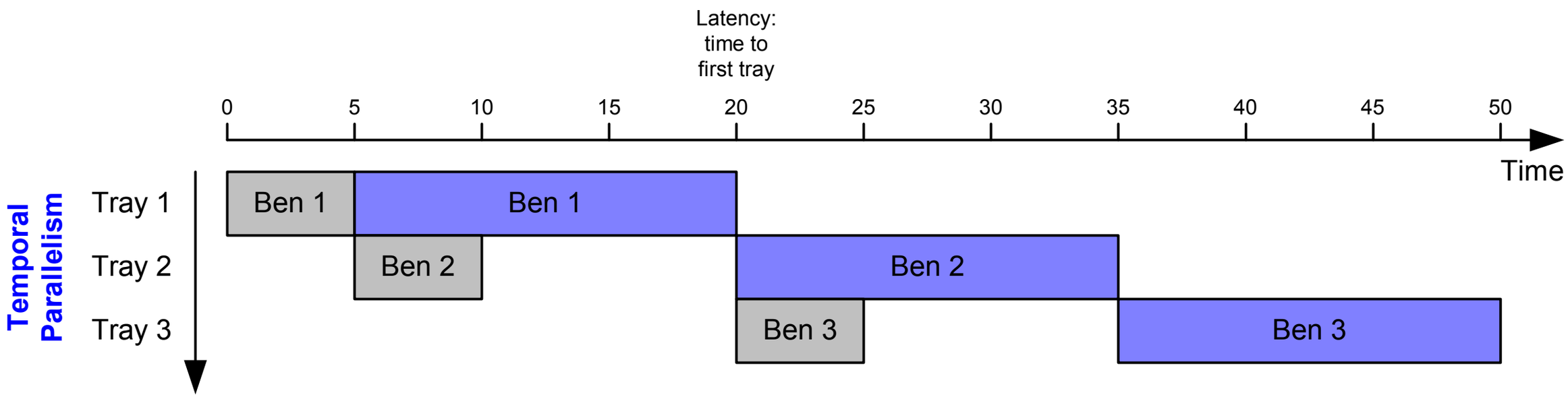
Spatial Parallelism



Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 2 trays/ 1/3 hour = **6 trays/hour**

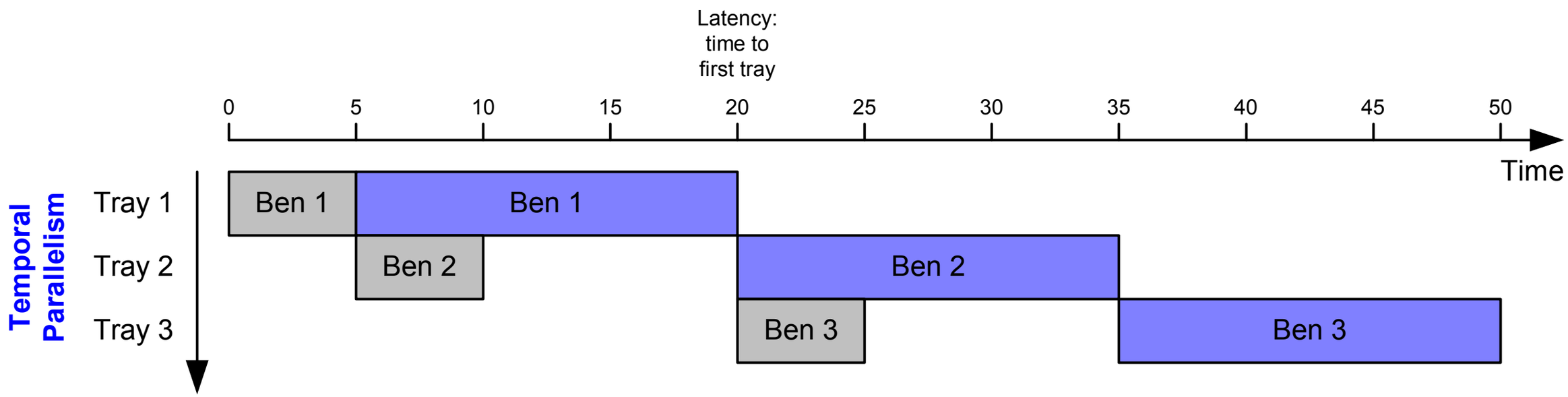
Temporal Parallelism



Latency = ?

Throughput = ?

Temporal Parallelism



Latency = 15 minutes = **1/4 hour**

Throughput = 1 trays/ 1/4 hour = **4 trays/hour**

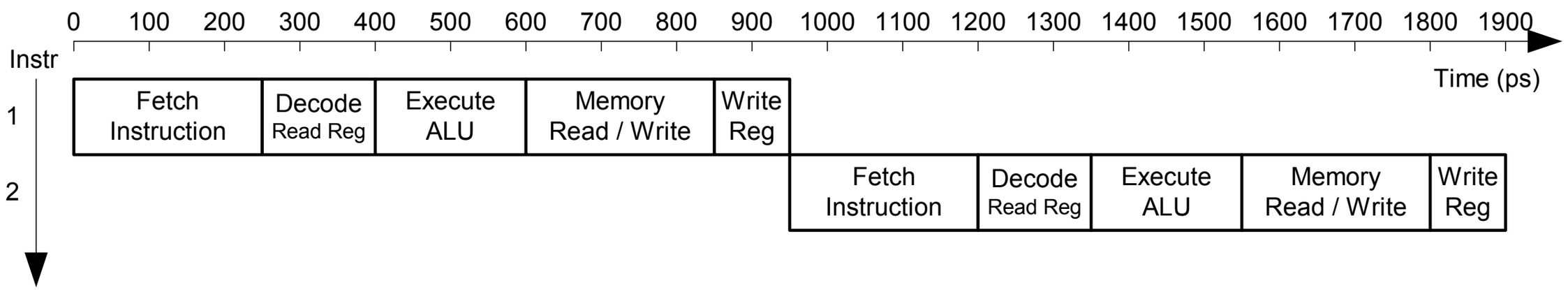
Using both techniques, the throughput would be **8 trays/hour**

Pipelined MIPS Processor

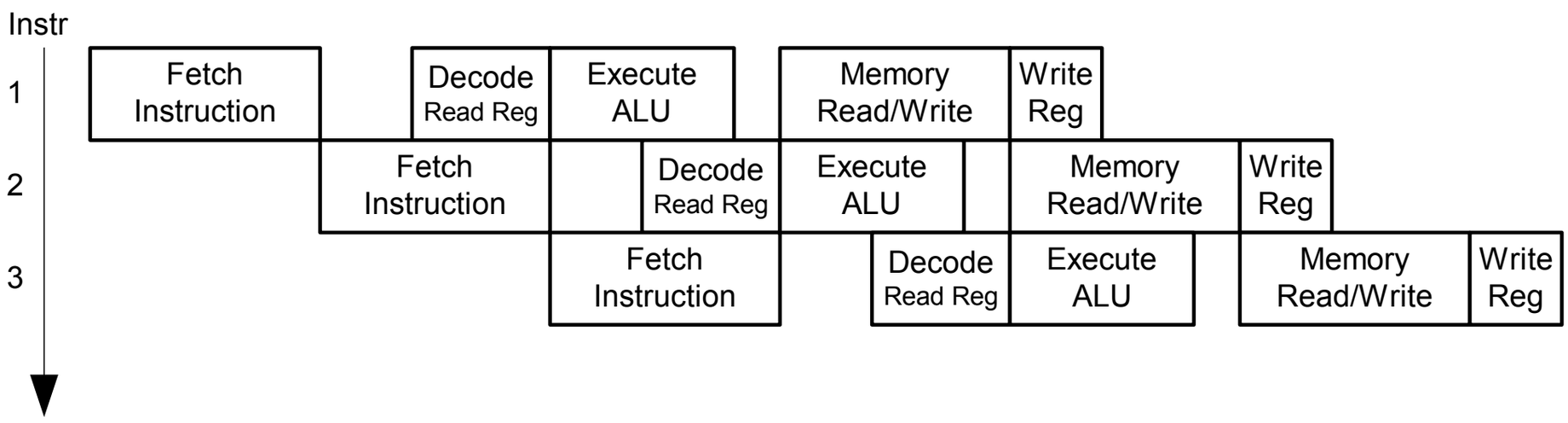
- Temporal parallelism
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add pipeline registers between stages

Single-Cycle vs. Pipelined

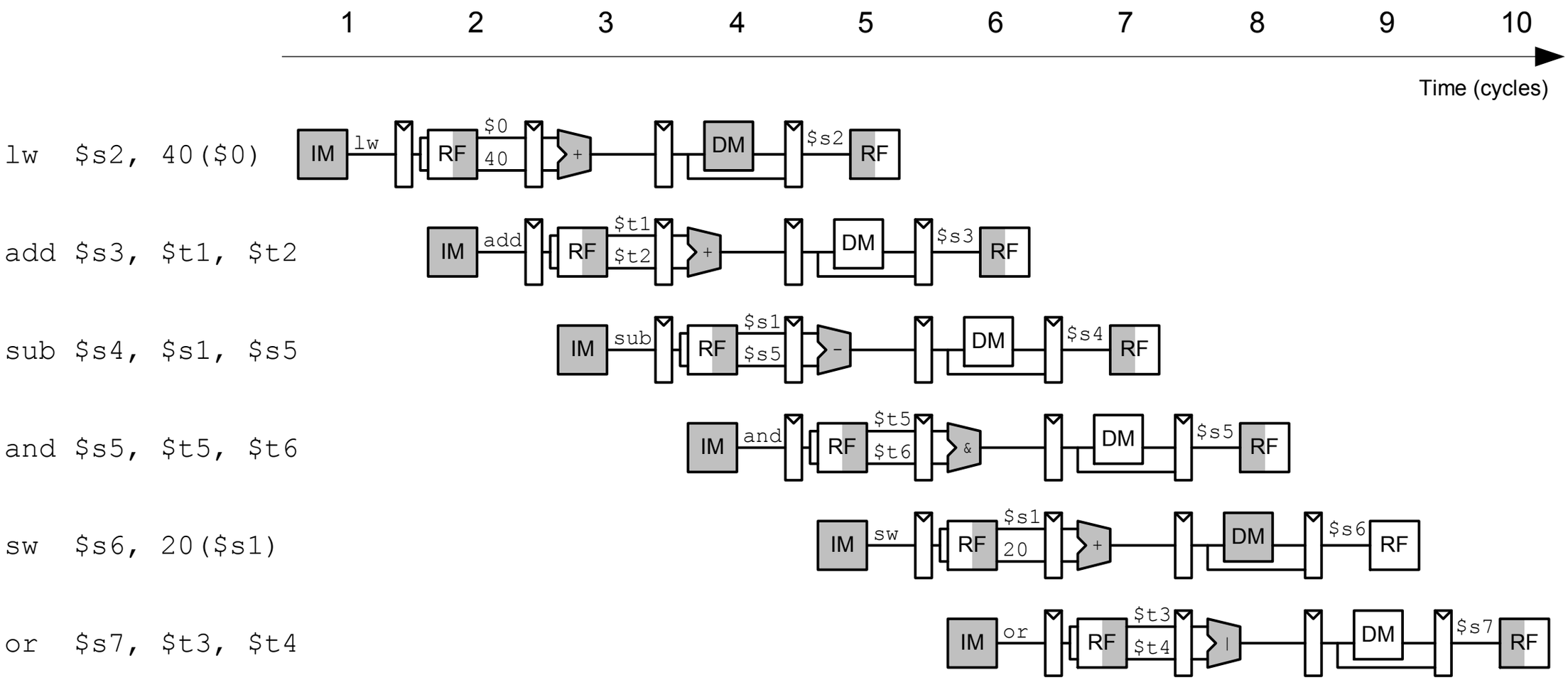
Single-Cycle



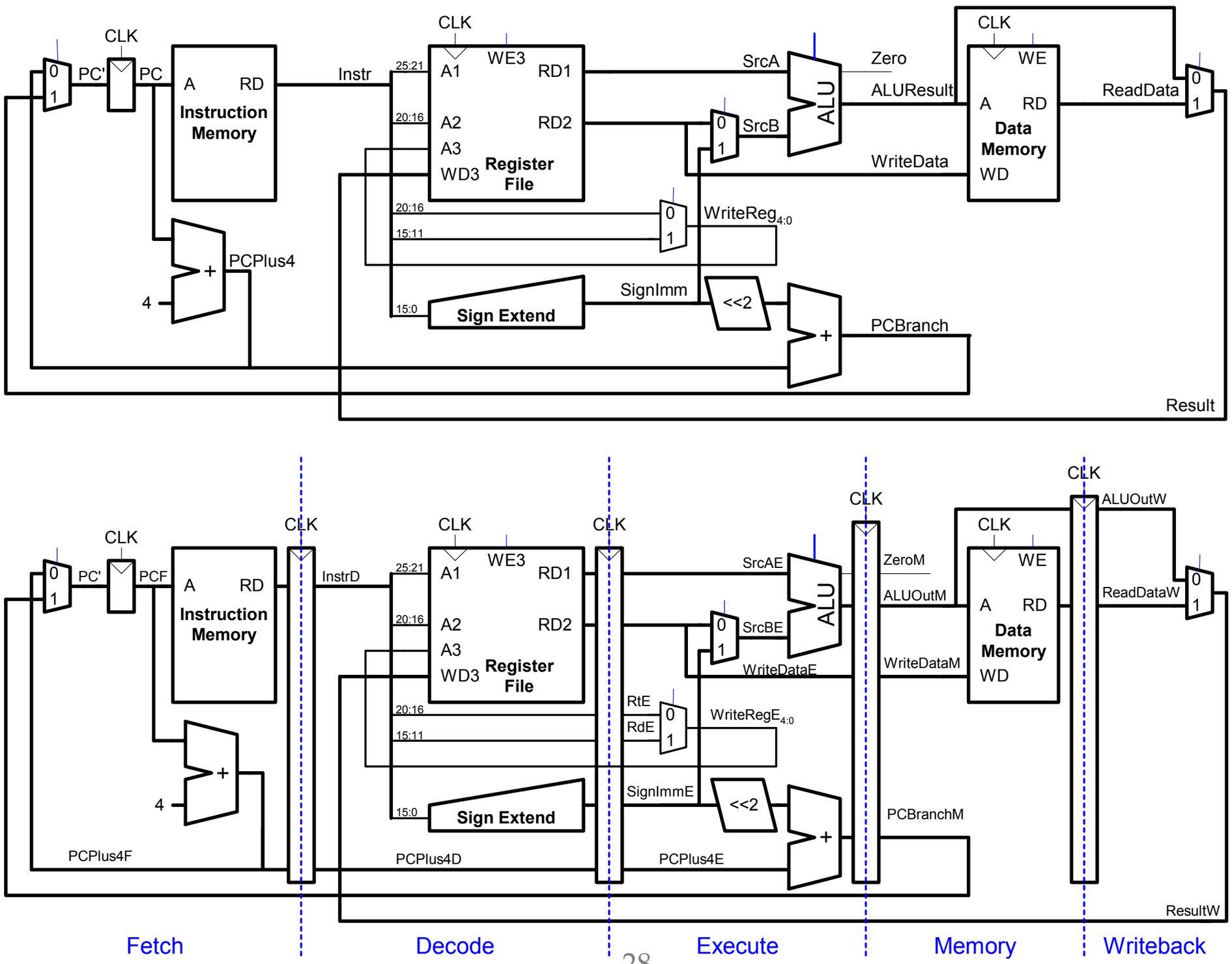
Pipelined



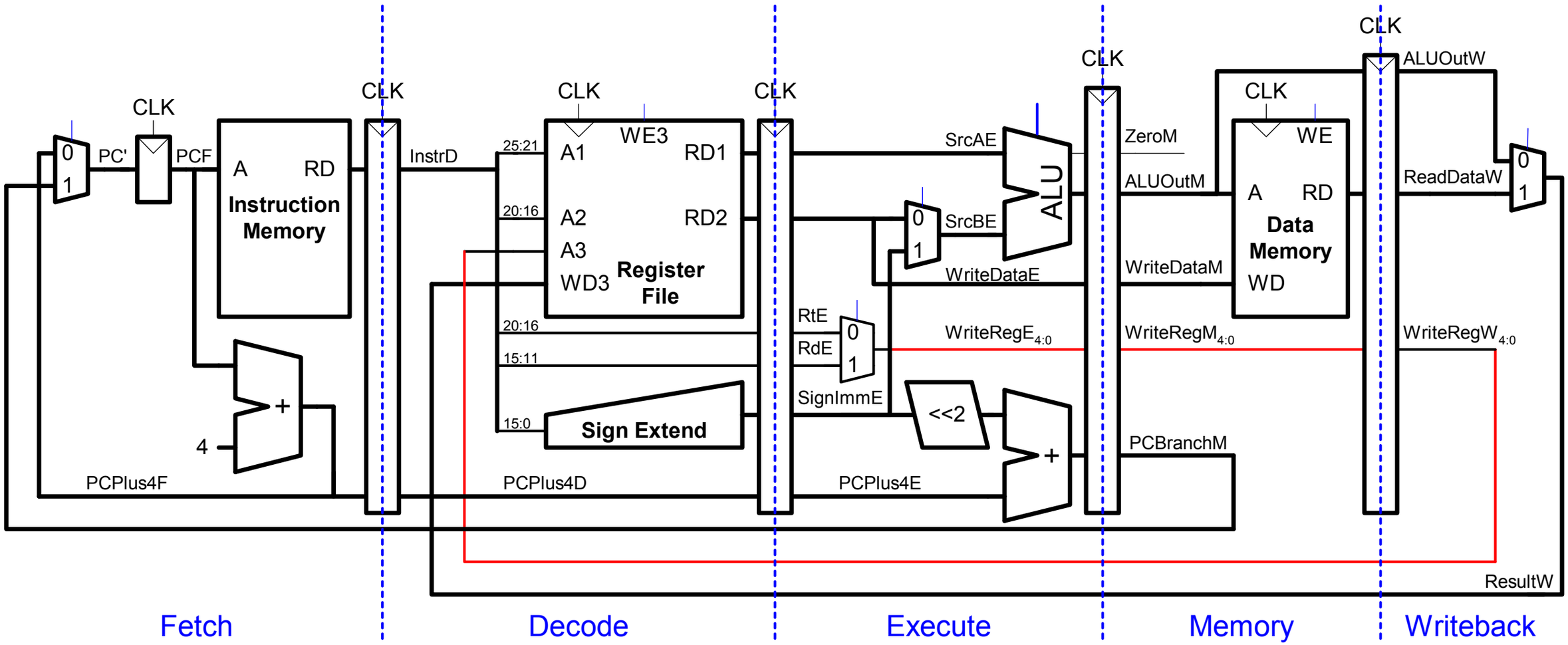
Pipelined Processor Abstraction



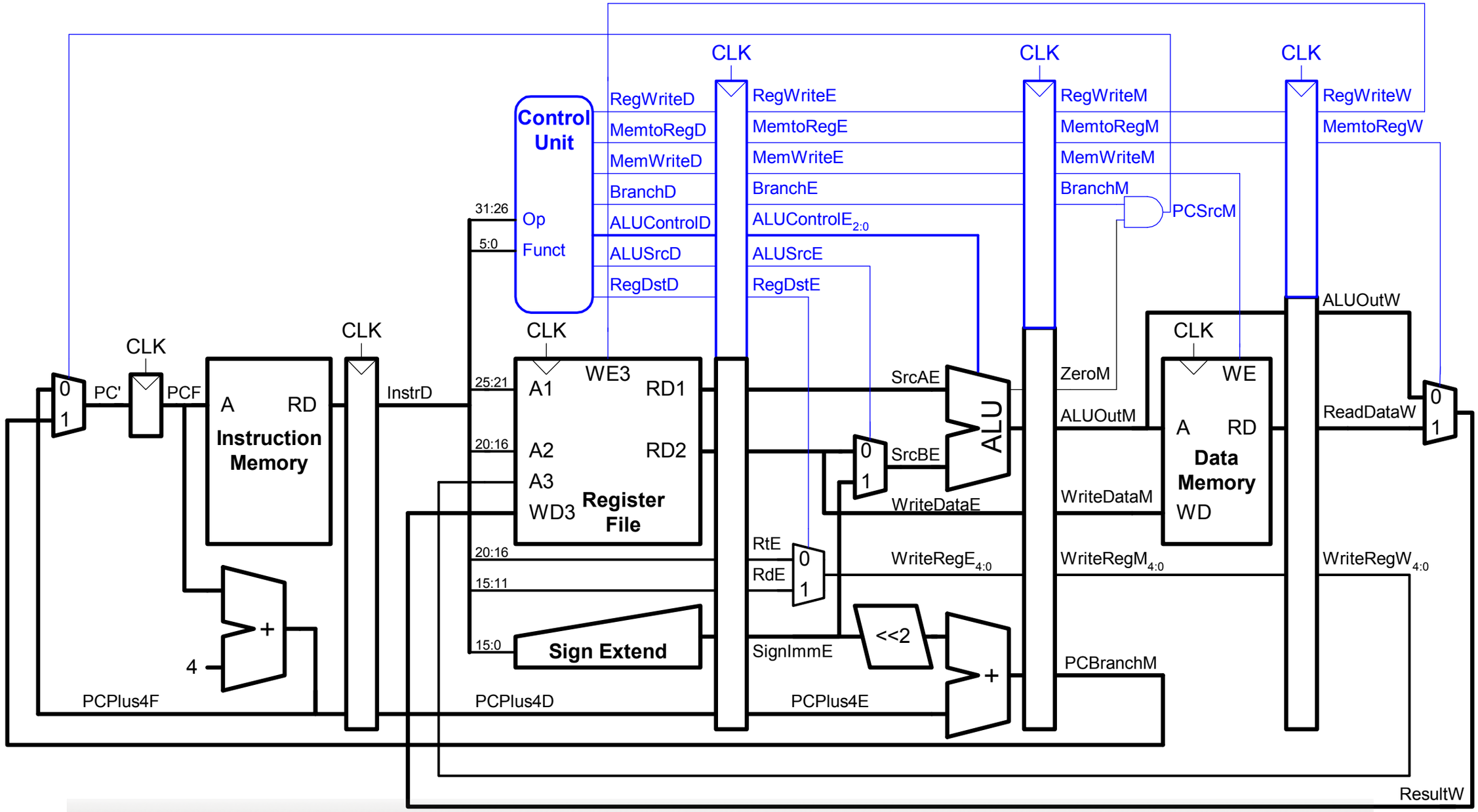
Single-Cycle & Pipelined Datapath



Corrected Pipelined Datapath



Pipelined Processor Control



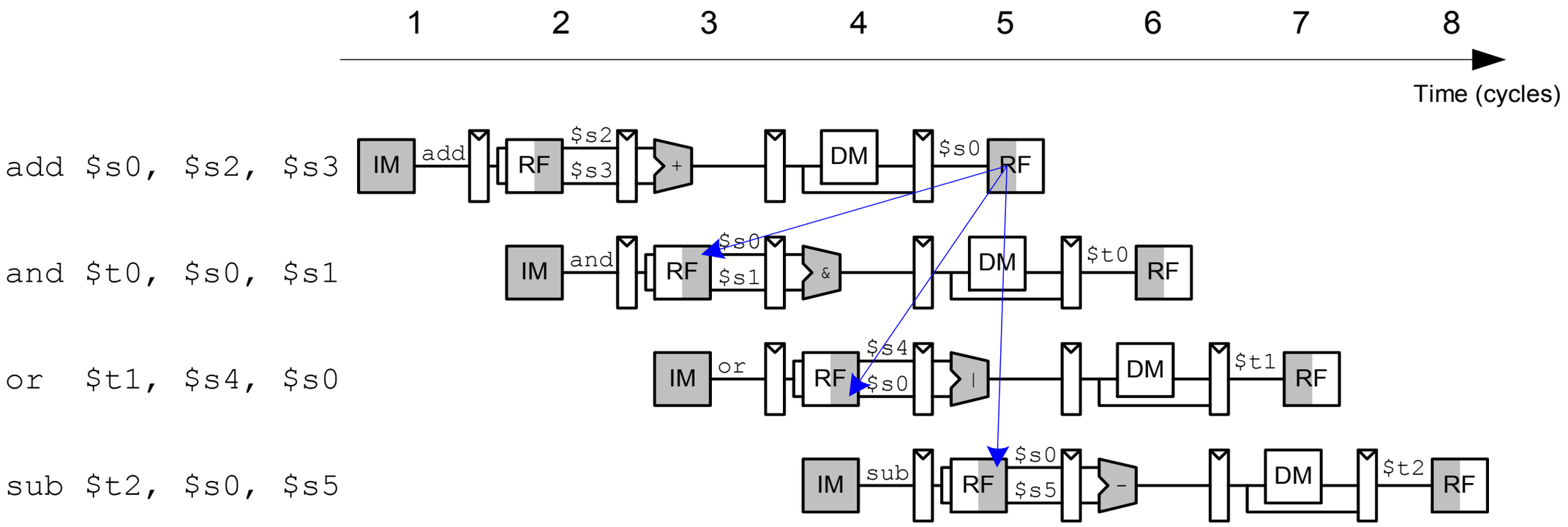
Same control unit as single-cycle processor

Control delayed to proper pipeline stage

Pipeline Hazards

- When an instruction depends on result from instruction that hasn't completed
- Types:
  - **Data hazard:** register value not yet written back to register file
  - **Control hazard:** next instruction not decided yet (caused by branches)

Data Hazard


1
2
3
4
5
6
7
8
Time (cycles)
add $s0, $s2, $s3
and $t0, $s0, $s1
or $t1, $s4, $s0
sub $t2, $s0, $s5
IM
add
RF
$s2
$s3
+
DM
$s0
RF
IM
and
RF
$s0
$s1
&
DM
$t0
RF
IM
or
RF
$s4
$s0
|
DM
$t1
RF
IM
sub
RF
$s0
$s5
-
DM
$t2
RF

Handling Data Hazards

- Insert `nop`s in code at compile time
- Rearrange code at compile time
- Forward data at run time
- Stall the processor at run time

Compile-Time Hazard Elimination

- Insert enough `nop`s for result to be ready
- Or move independent useful instructions forward