Lecture 15

Introduction to Pipelining

Guest Lecture by

Mahesh

THE UNIVERSITY OF UTAH

`$ whoami`

I’m ***Mahesh***, a PhD student in Computer Science @ the U.

What I do now….

I research optimizing compilers for high-performance machine learning.

What I want to do in the future….

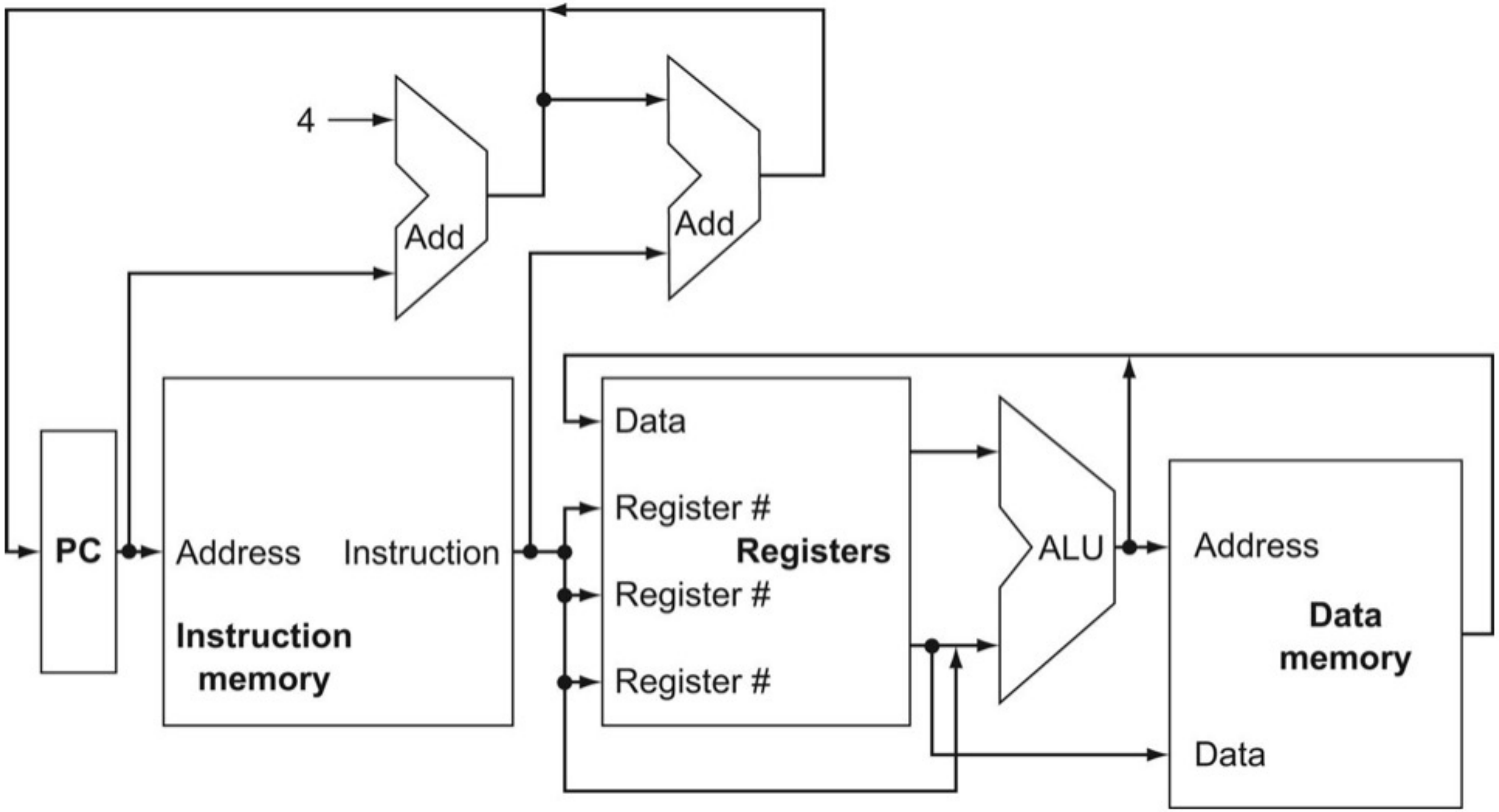
Become a Professor and teach CS to students like you!

Today's Lecture

- Recap Single-cycle CPU
- Multicycle CPU
- Pipelined Architecture
- Effects of Pipelining

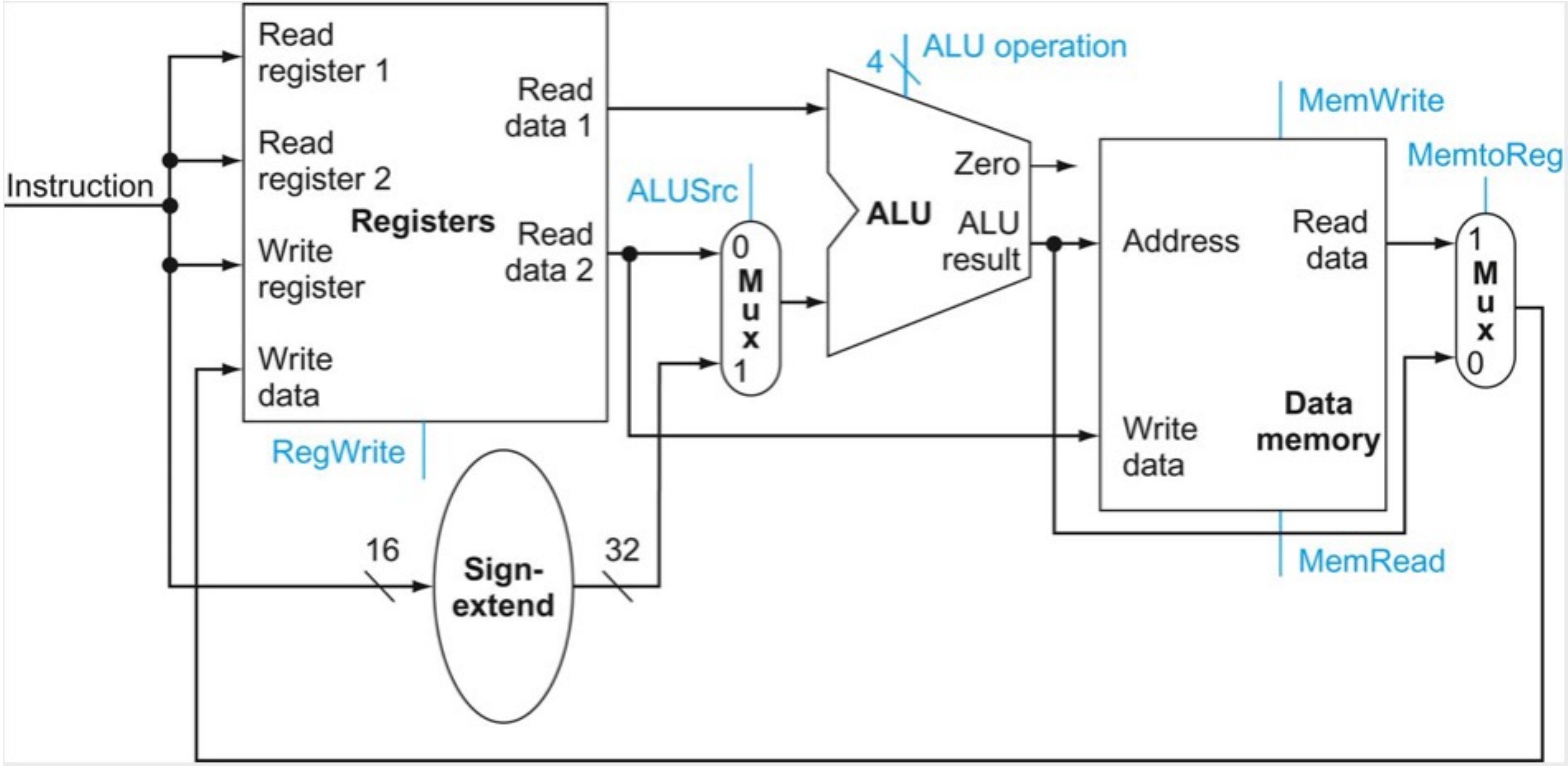
Recap: Single-cycle Processor

View from 30,000 feet



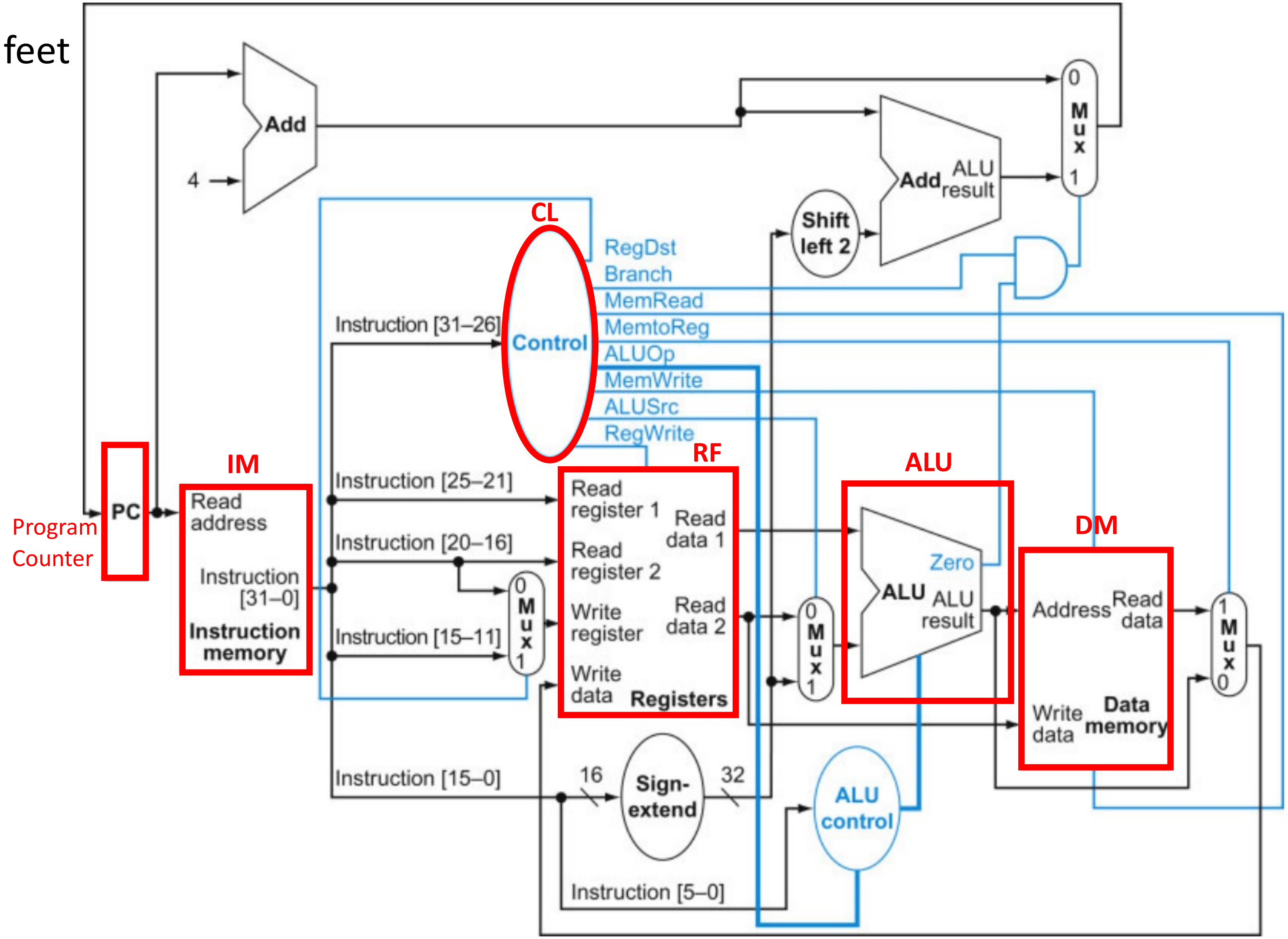
Recap: Single-cycle Processor

View from 10,000 feet



Recap: Single-cycle Processor

View from 5,000 feet

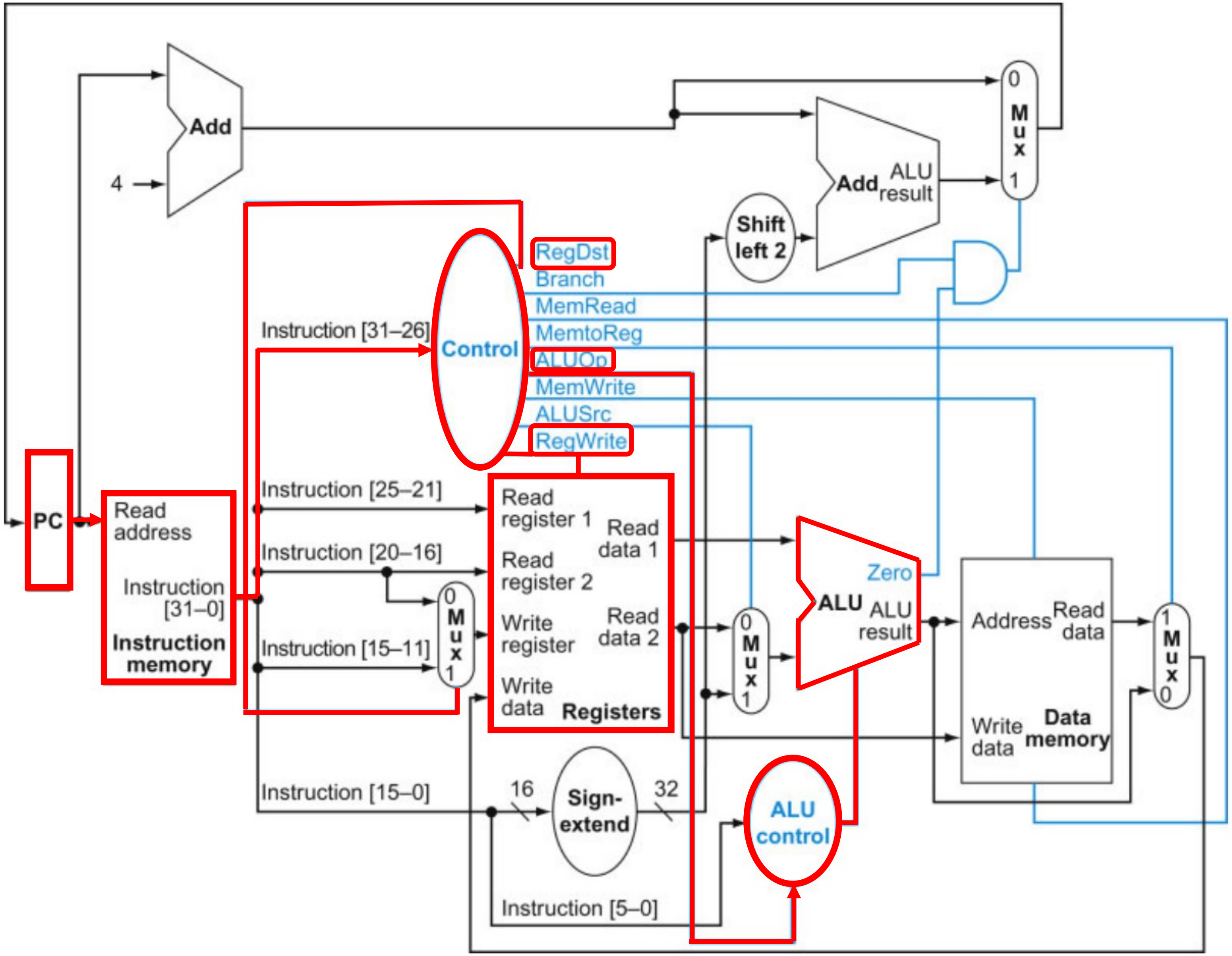


Recap: Single-cycle Processor

R-Type Instruction

```
add $1, $2, $3
```

	$2	$3	$1		
31:26	25:21	20:16	15:11	10:6	5:0
0	rs	rt	rd	shamt	funct
op	Source Registers		Destination Register	Shift Amount	Function Code

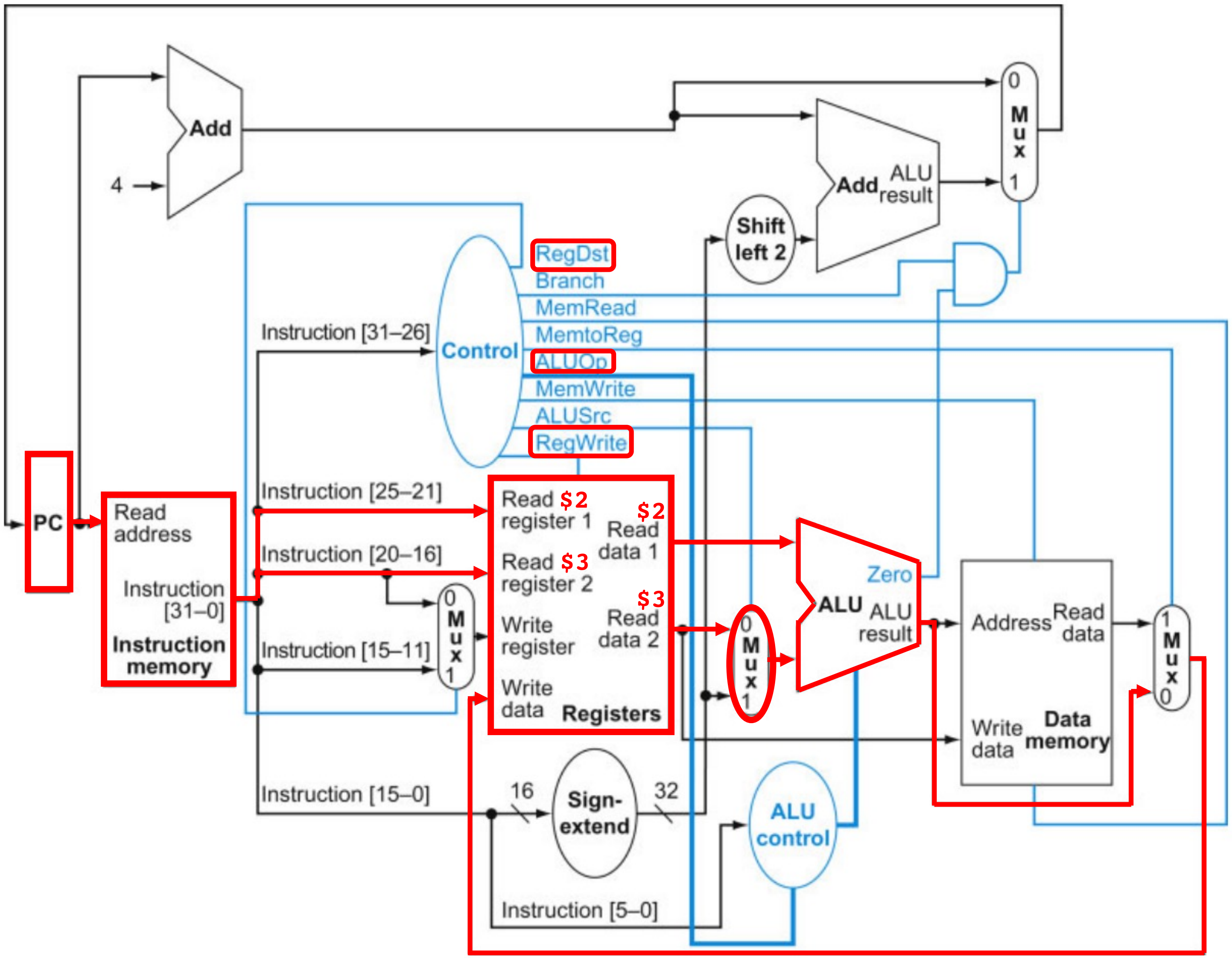


Recap: Single-cycle Processor

R-Type Instruction

`add $1, $2, $3`

	$2	$3	$1		
31:26	25:21	20:16	15:11	10:6	5:0
0	rs	rt	rd	shamt	funct
op	Source Registers		Destination Register	Shift Amount	Function Code

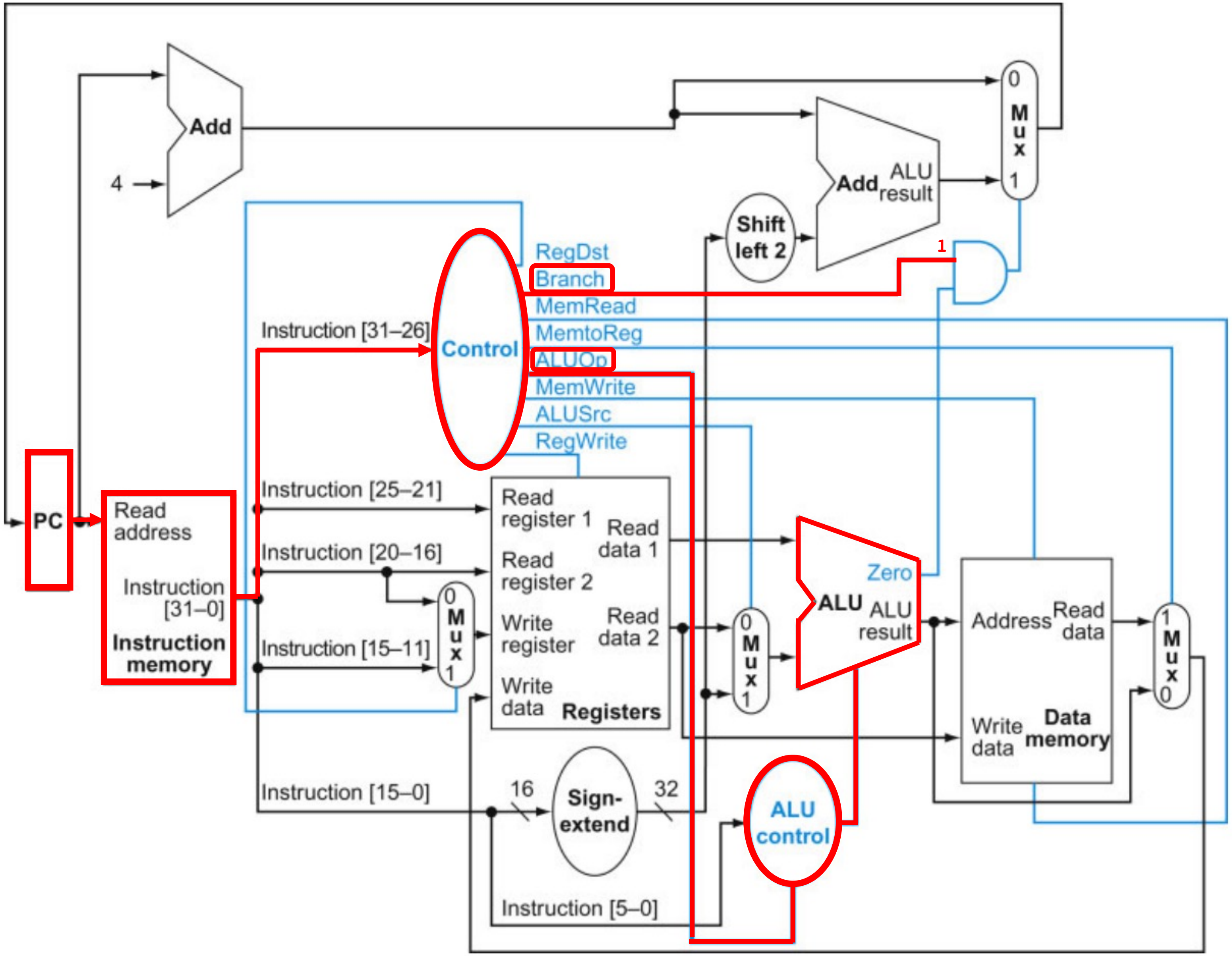


Recap: Single-cycle Processor

I-Type Instruction

```
beq $1, $2, 1000
```

	$1	$2	
31:26	25:21	20:16	15:0
4	rs	rt	offset
op	Source Registers		

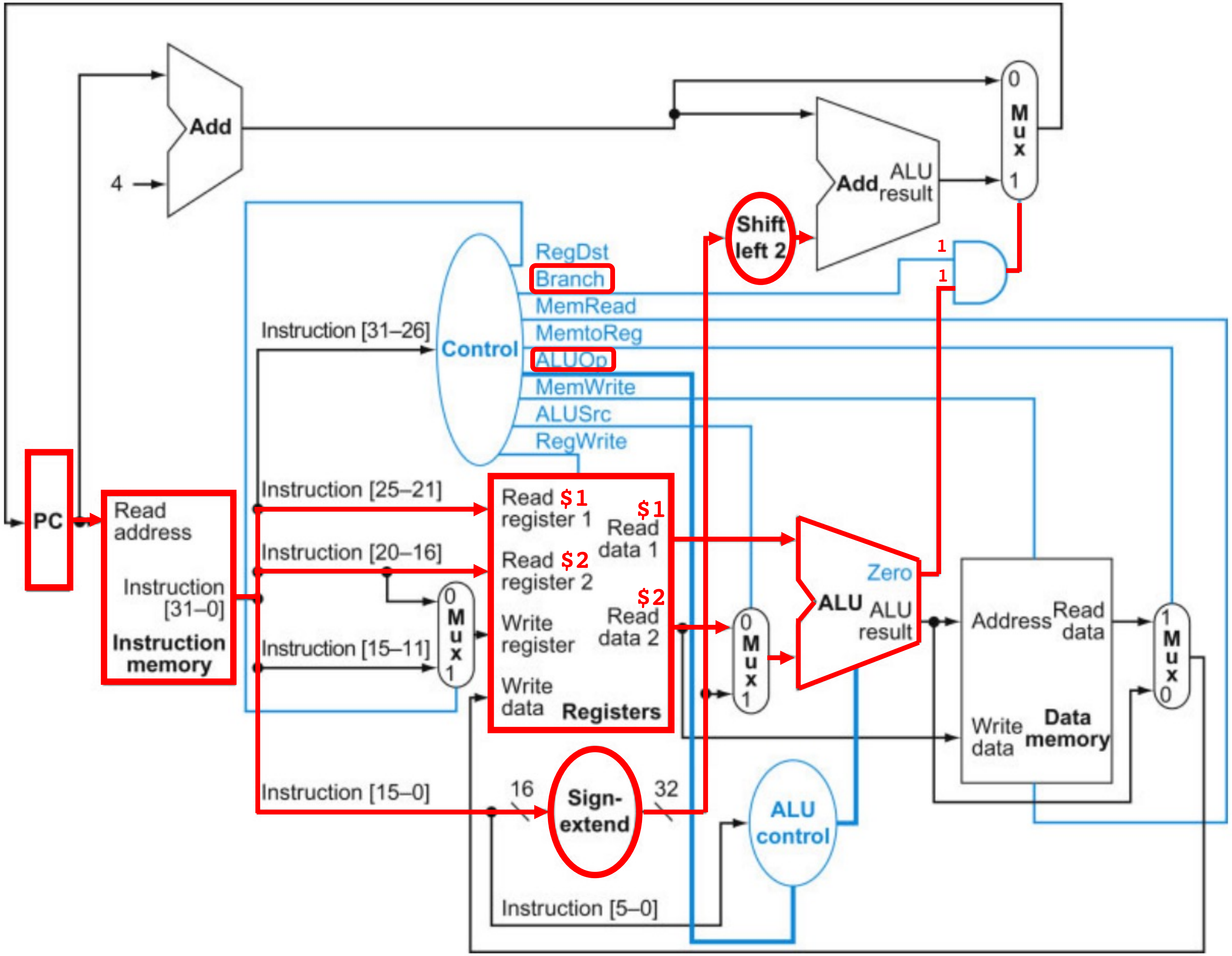


Recap: Single-cycle Processor

I-Type Instruction

```
beq $1, $2, 1000
```

	$1	$2	
31:26	25:21	20:16	15:0
4	rs	rt	offset
op	Source Registers		

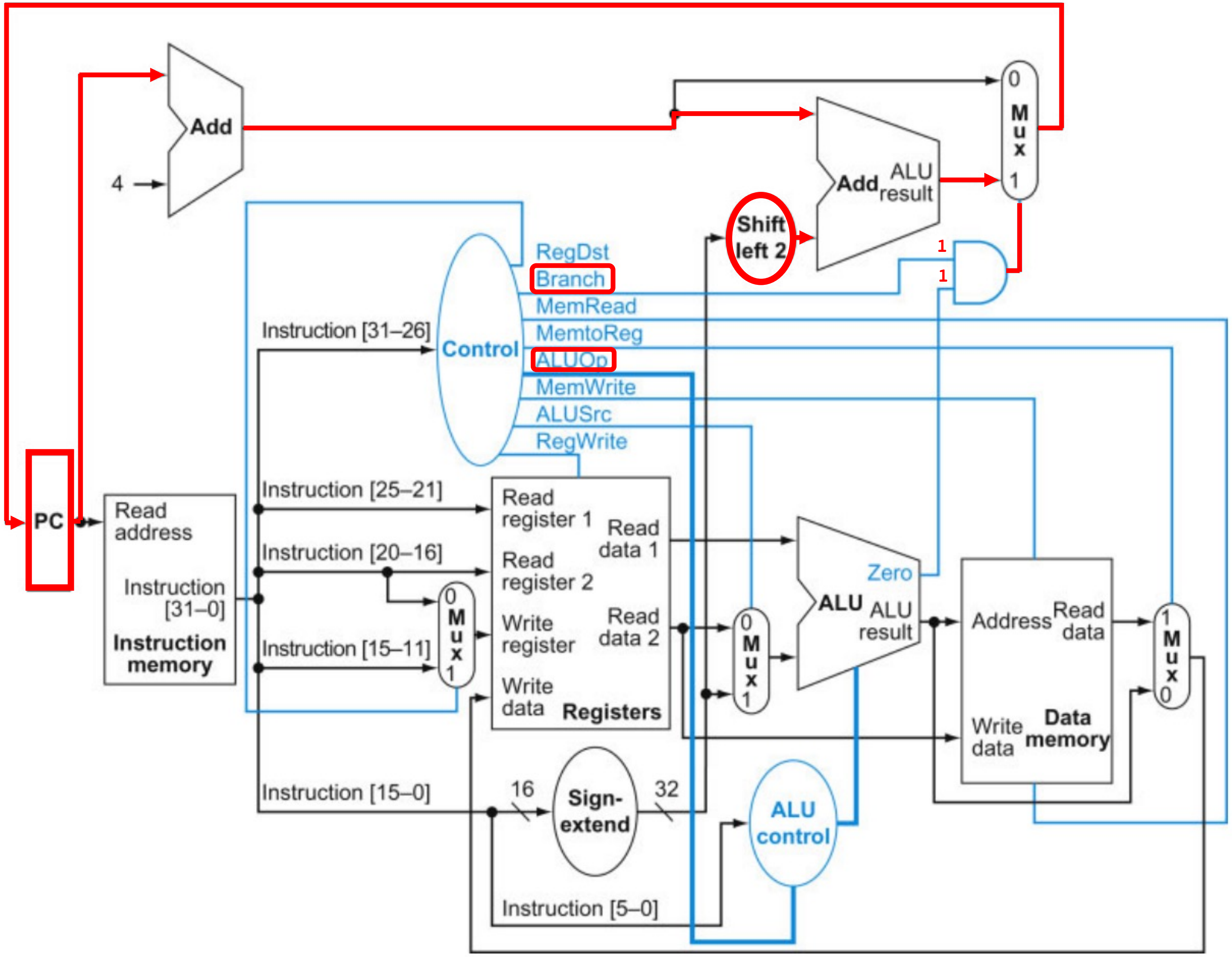


Recap: Single-cycle Processor

I-Type Instruction

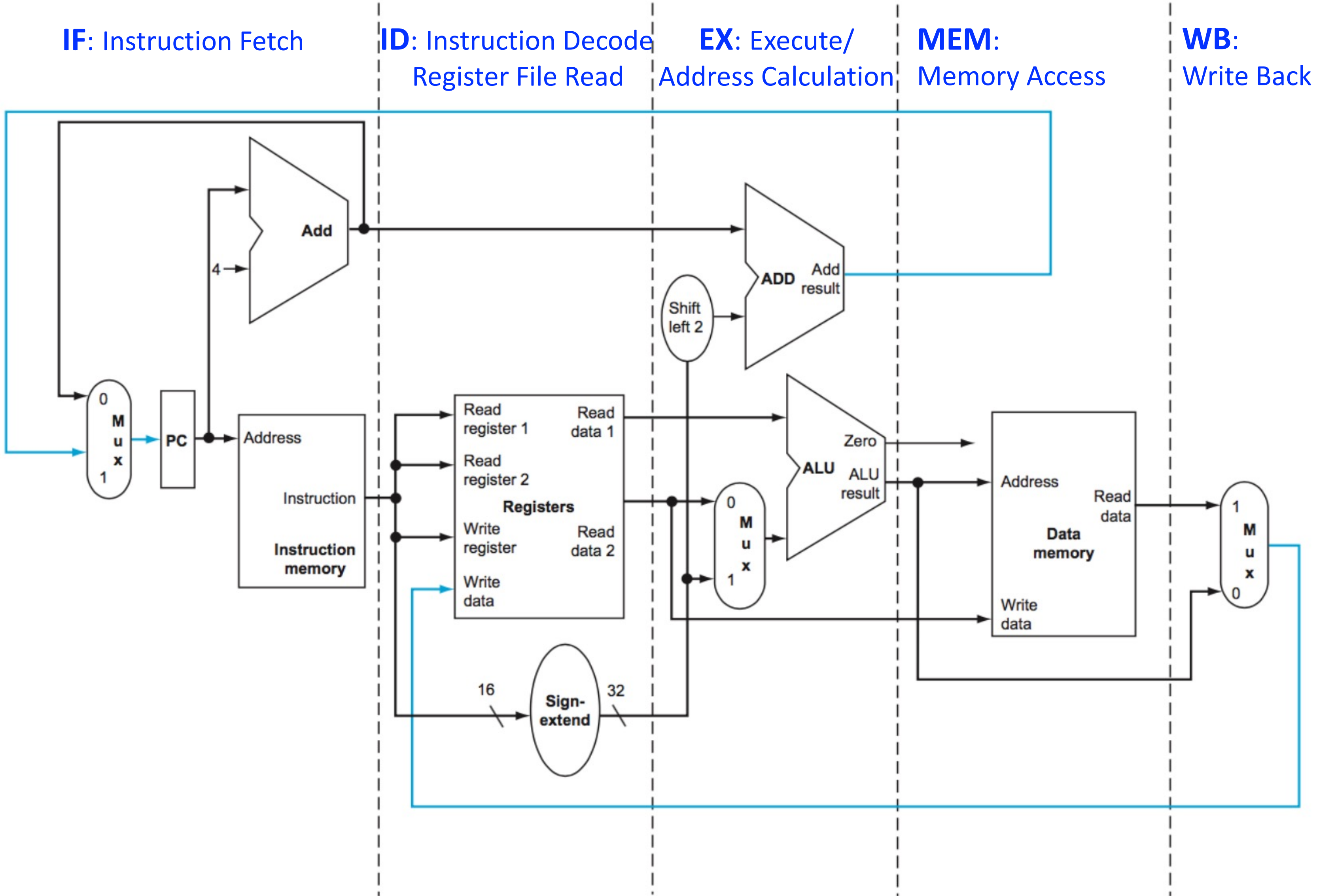
```
beq $1, $2, 1000
```

	$1	$2	
31:26	25:21	20:16	15:0
4	rs	rt	offset
op	Source Registers		



Processing Instructions

- A sequence of processing tasks per instruction

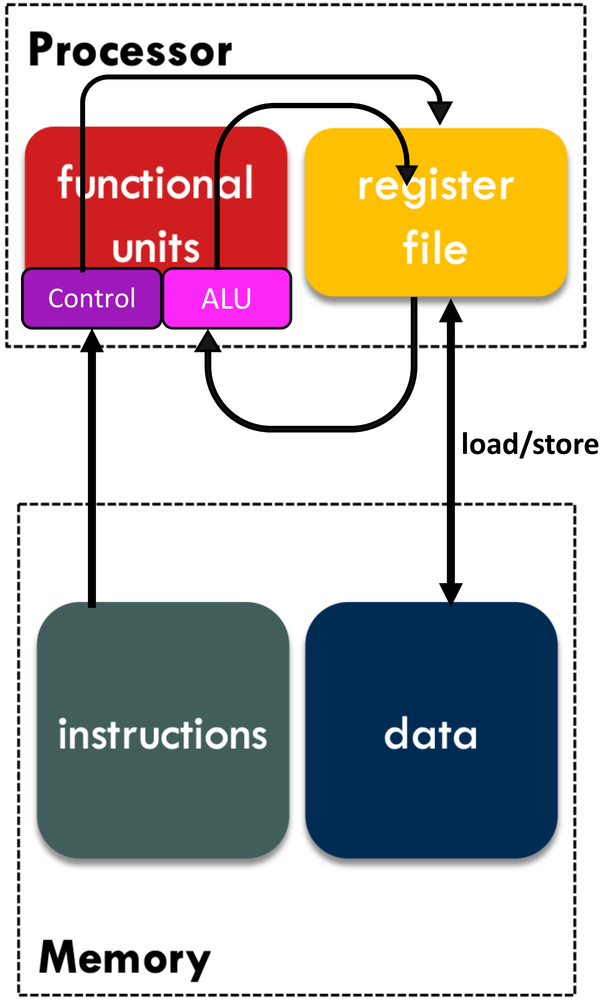


Processing Instructions

Every instruction may require multiple processing steps:

- **IF**: Instruction Fetch ✓
- **ID**: Instruction Decode ✓

  Register Read (RR) ✓
- **EXE**: Execute Instructions ✓
- **MEM**: Memory Access ✓
- **WB**: Register Write Back ✓

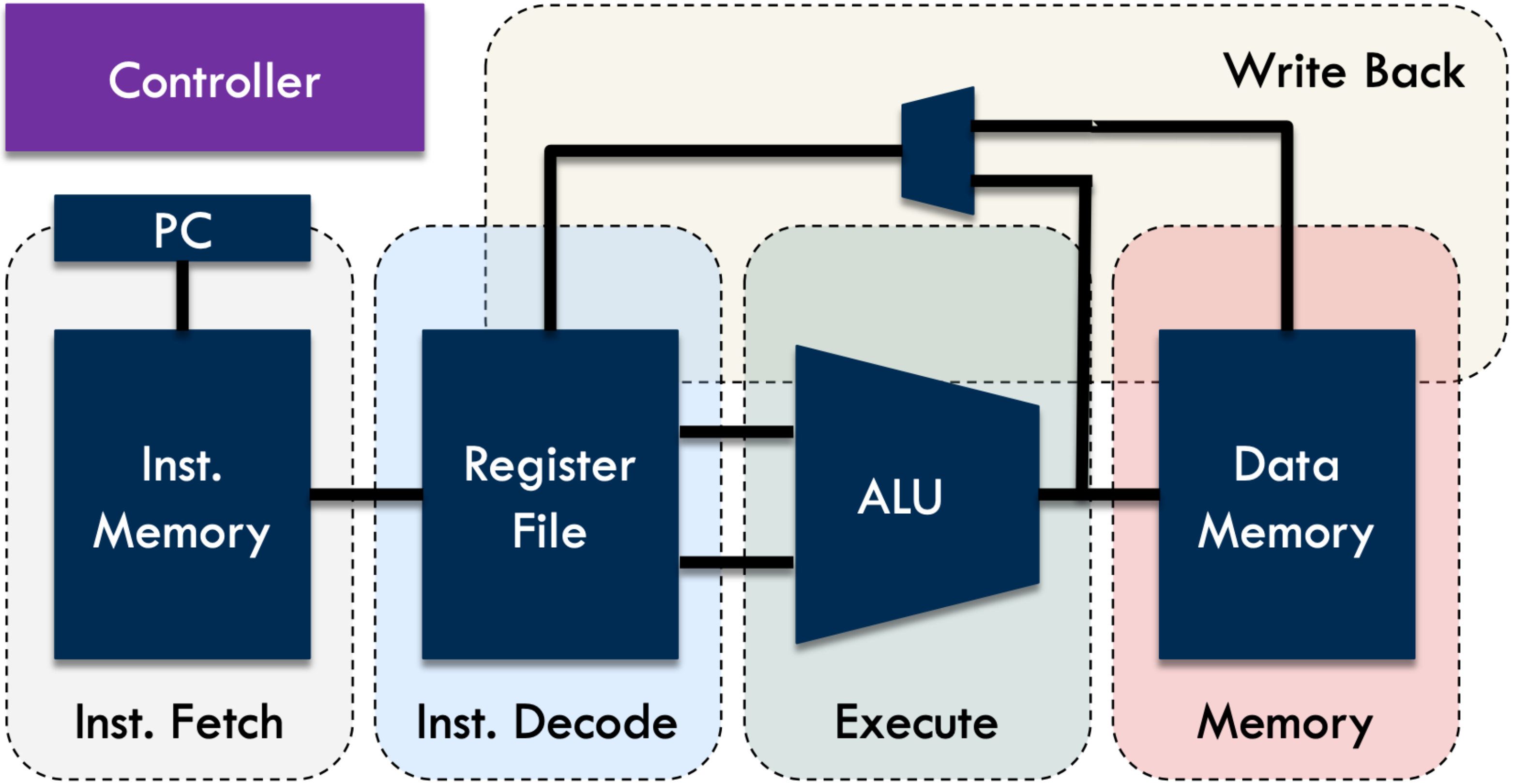


Single-cycle Architecture

Critical path

- Includes all of the processing steps
- Determines clock cycle time

`lw $1, 8($2)`

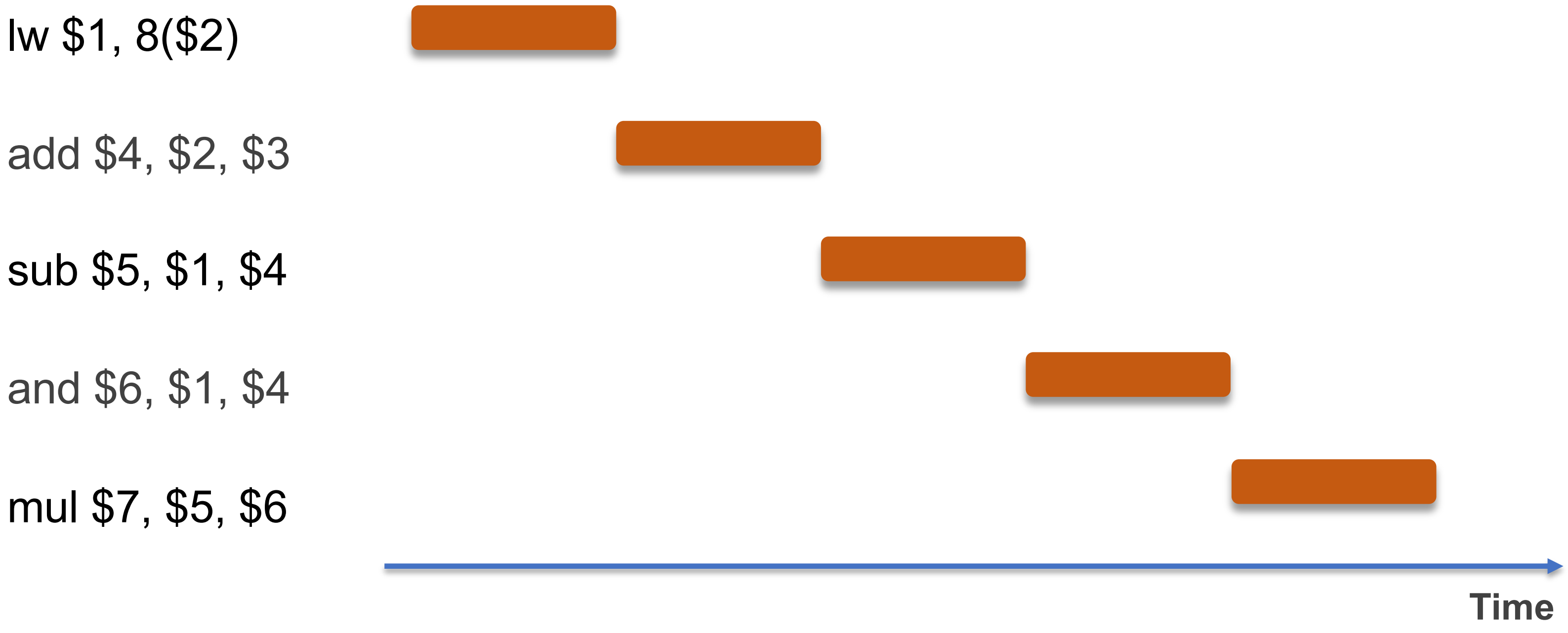


Single-cycle CPU Performance

- Example Program:

What is the CPU time for a Cycle Time of 6 ns?

CT = 6 ns; CPU Time = ?



```
lw $1, 8($2)
add $4, $2, $3
sub $5, $1, $4
and $6, $1, $4
mul $7, $5, $6
```

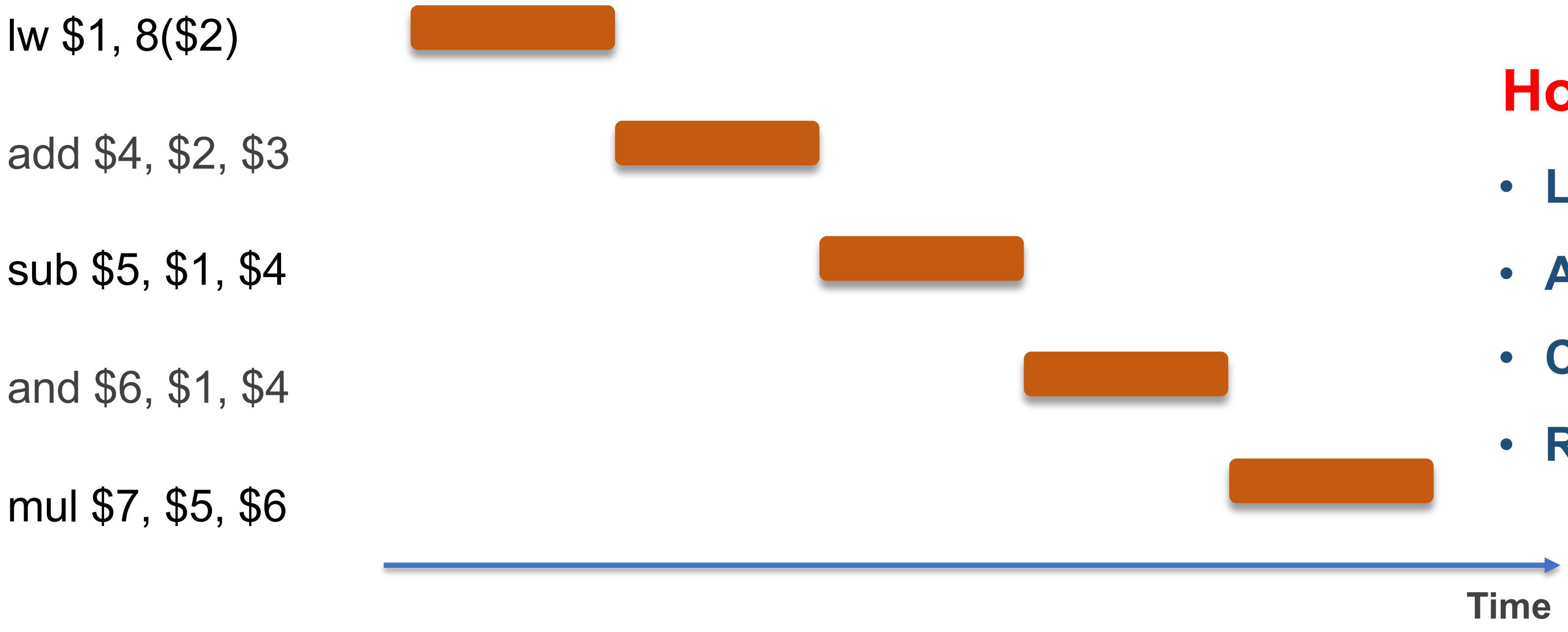
Single-cycle CPU Performance

$$CPU\ Time = IC \times CPI \times CT$$

- Example Program:

  What is the CPU time for a Cycle Time of 6 ns?

CT = 6 ns; CPU Time = 5 x 6 ns= 30 ns



lw $1, 8($2)

add $4, $2, $3

sub $5, $1, $4

and $6, $1, $4

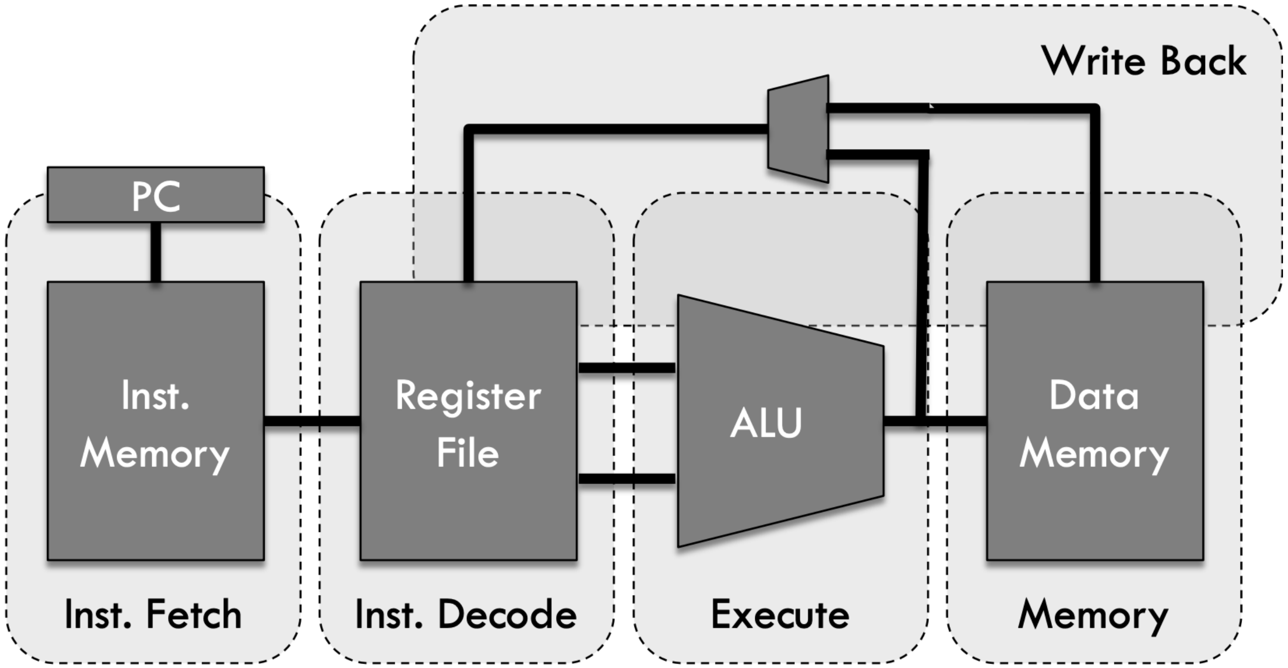
mul $7, $5, $6

How to improve?

- **Locality Optimization**
- **Amdahl’s Law**
- **Common Case Fast**
- **Reuse Idle Resources**

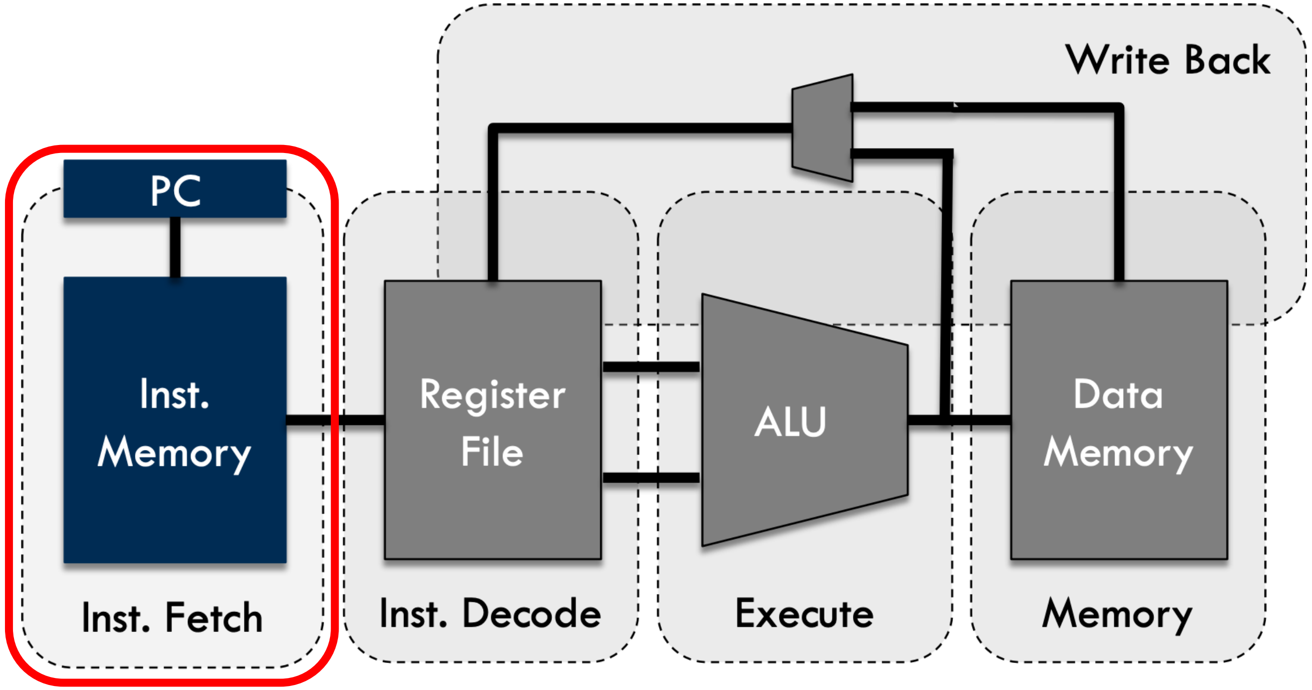
Reusing Idle Resources

- Each processing step finishes in a fraction of a cycle.
- Idle resources can be reused for processing



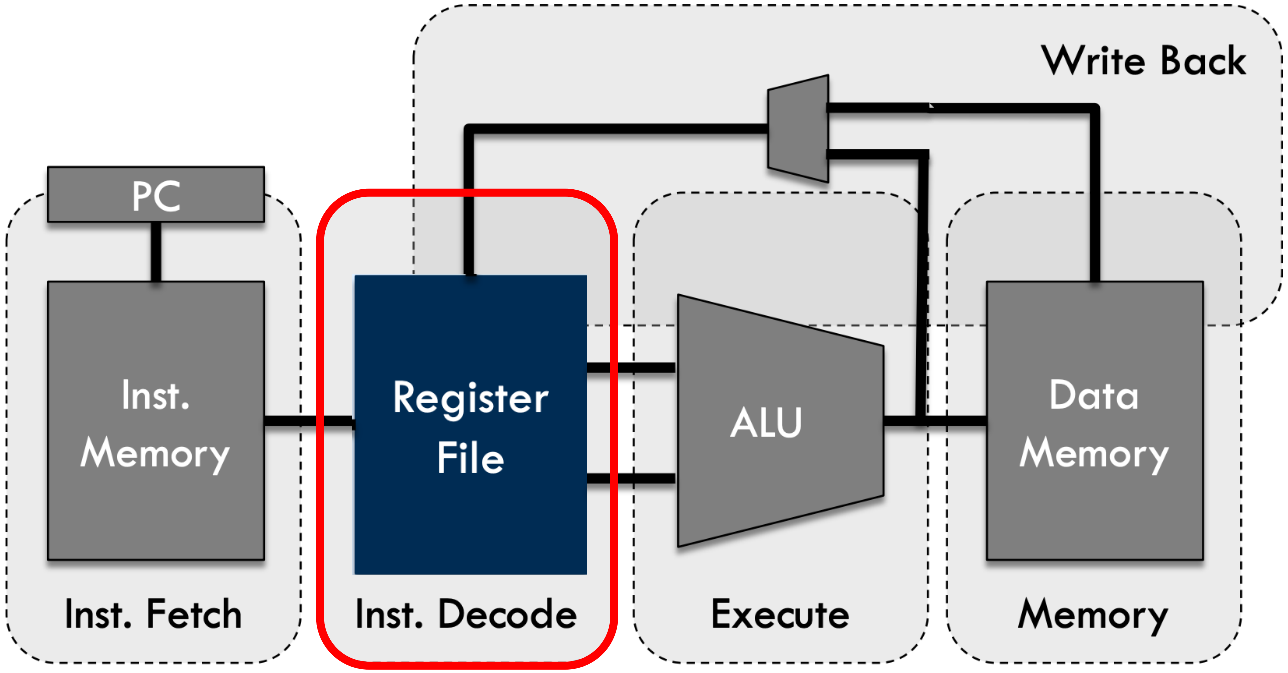
Reusing Idle Resources

- Each processing step finishes in a fraction of a cycle.
- Idle resources can be reused for processing



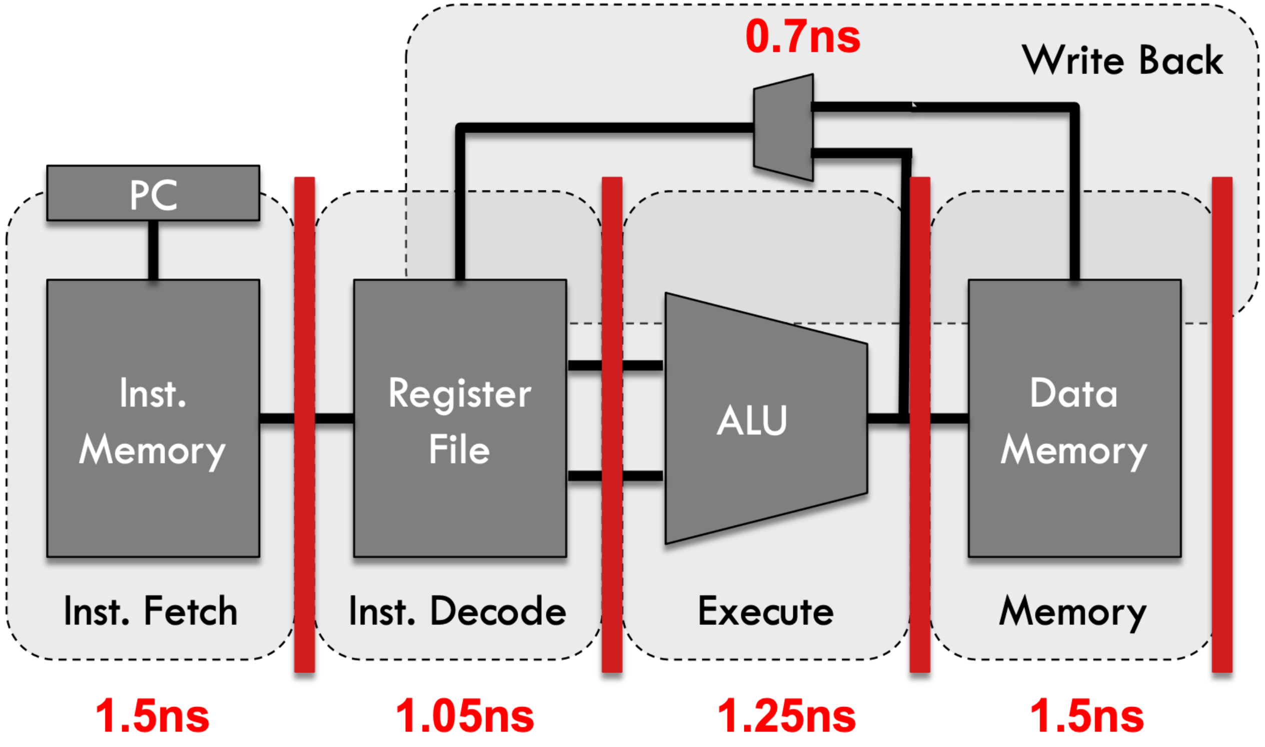
Reusing Idle Resources

- Each processing step finishes in a fraction of a cycle.
- Idle resources can be reused for processing

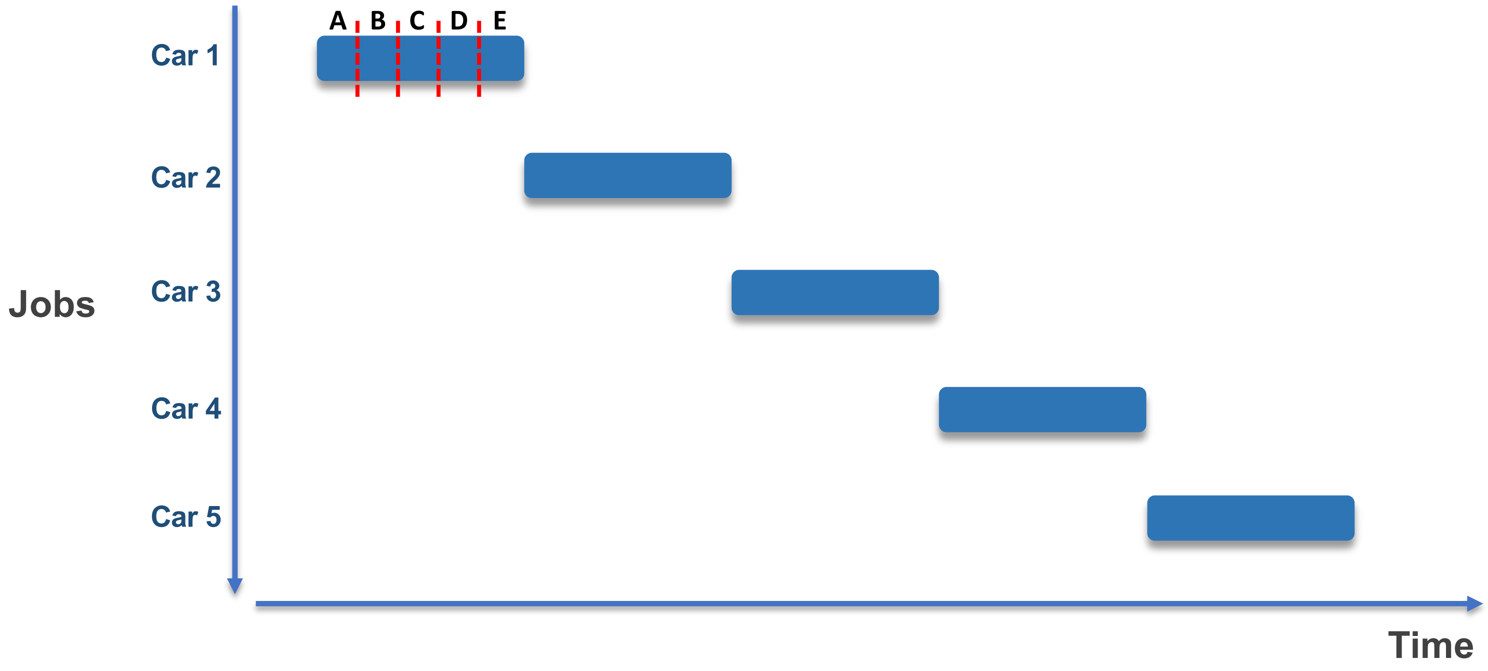


Multi-stage Circuit

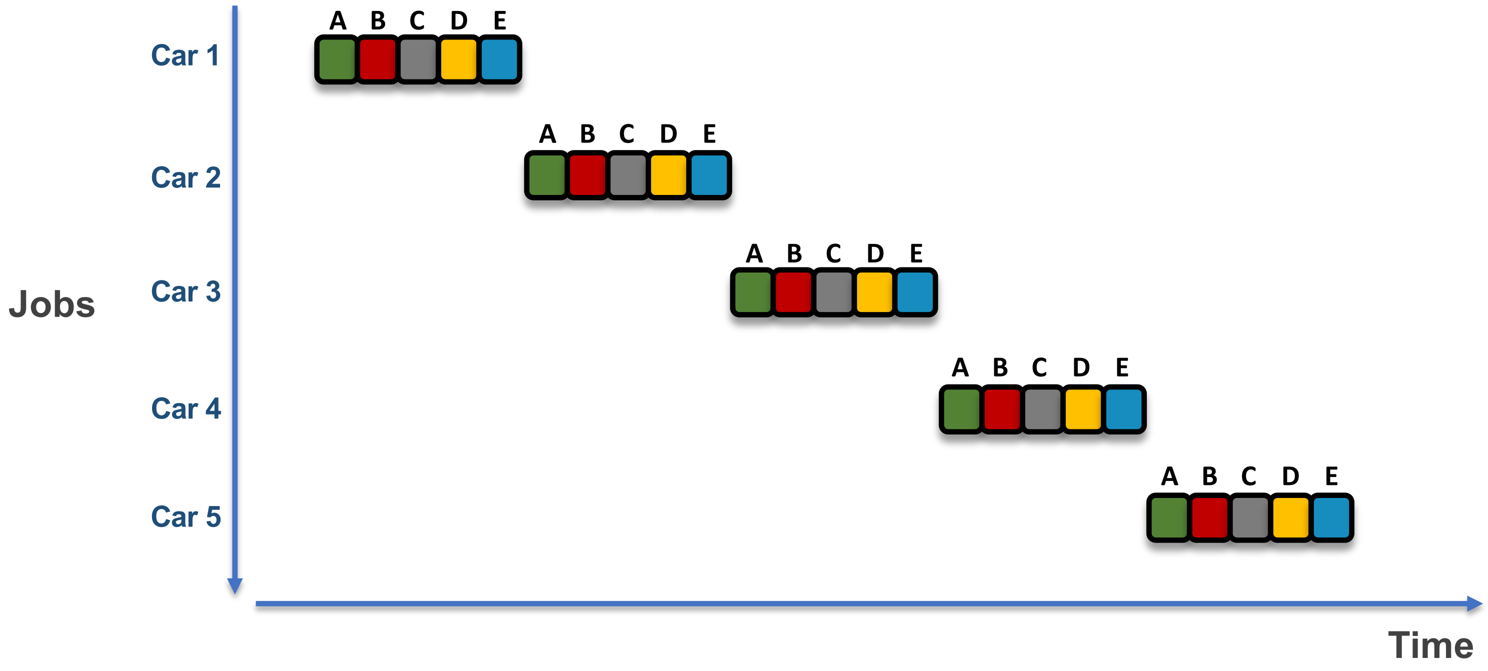
- Each processing step finishes in a fraction of a cycle.
- Idle resources can be reused for processing



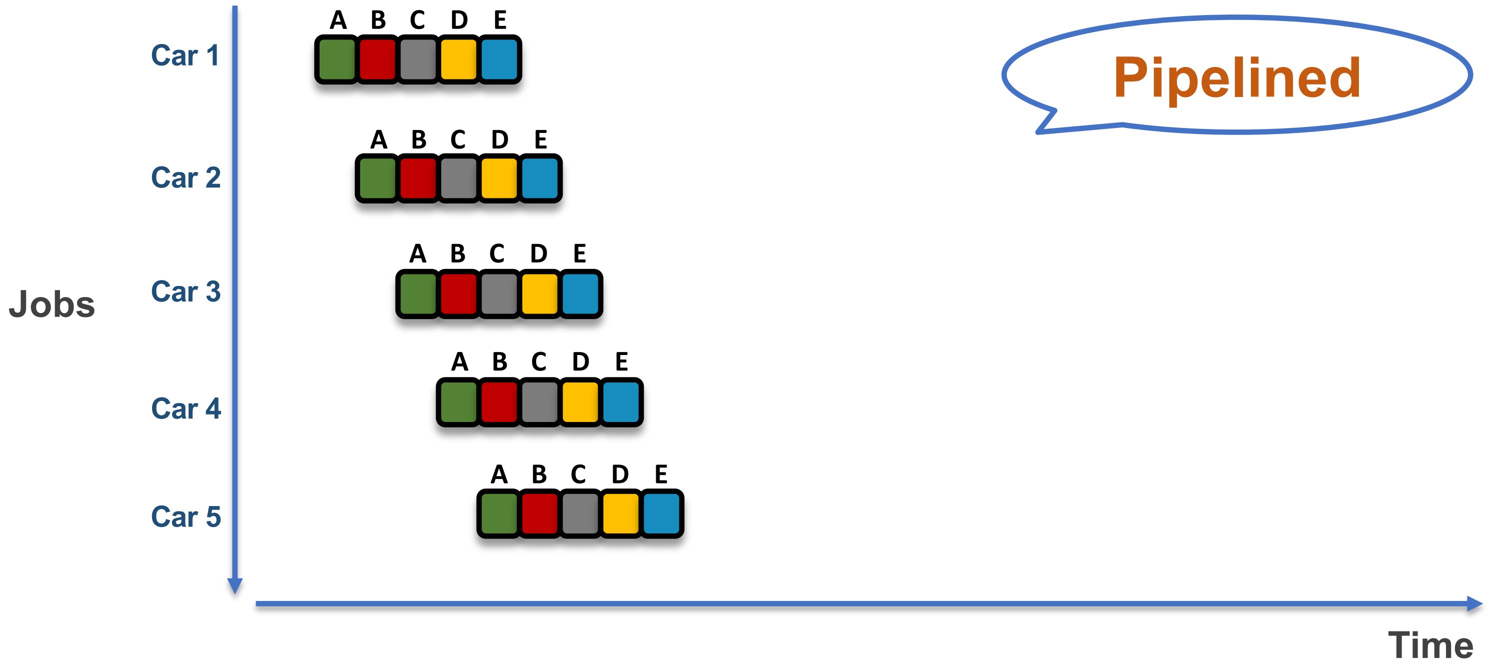
Analogy: Car Assembly Line



Analogy: Car Assembly Line

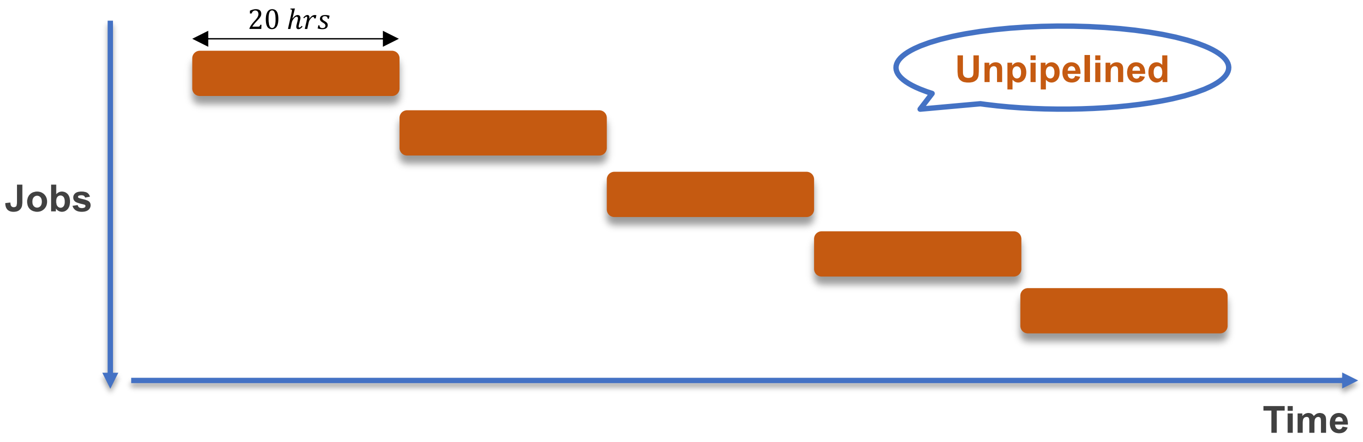


Analogy: Car Assembly Line

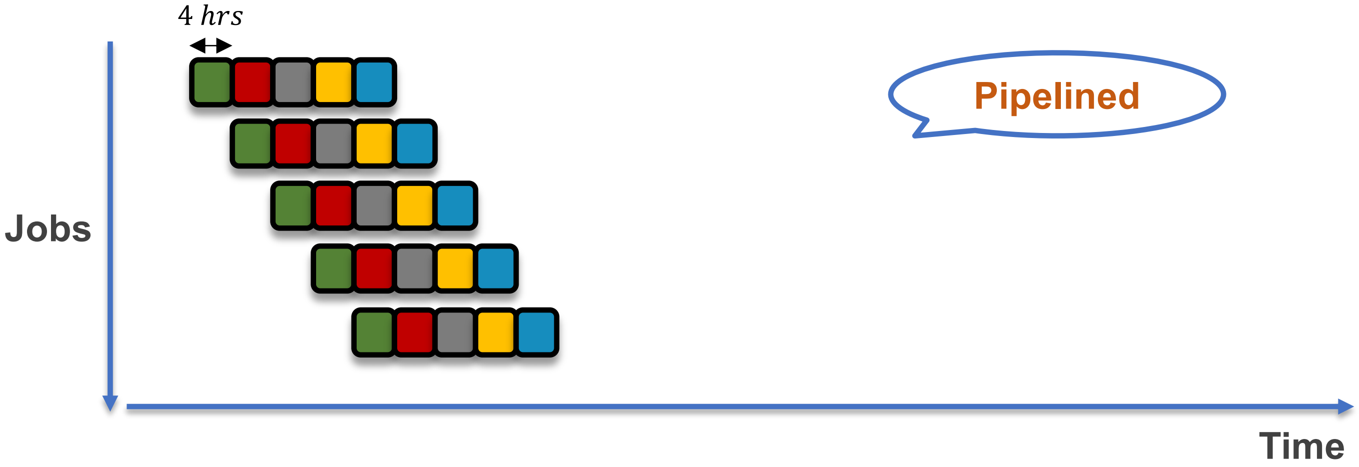


Pipelined

Car Assembly Line With Pipelining

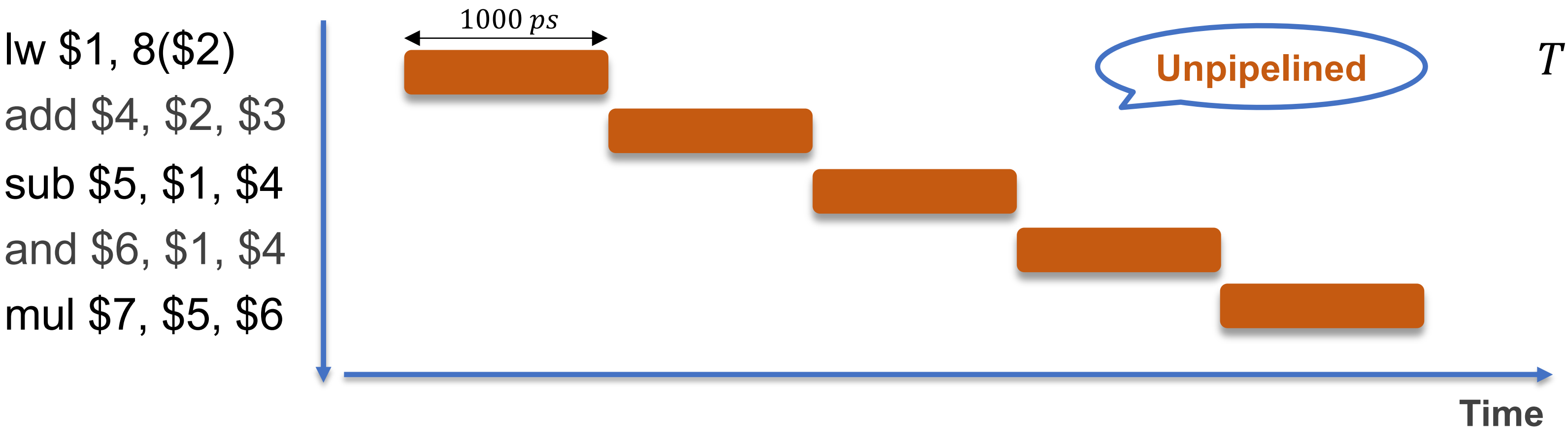


$$Throughput = \frac{1\ car}{20\ hrs}$$



$$Throughput = \frac{1\ car}{4\ hrs}$$

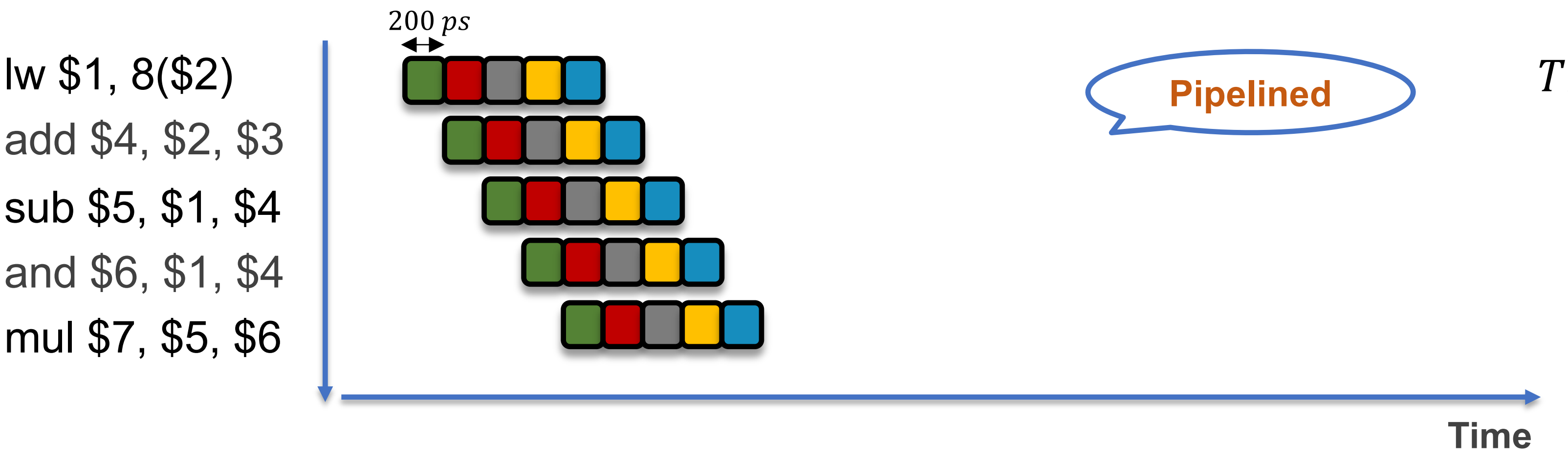
Pipelined Processor



lw $1, 8($2)
add $4, $2, $3
sub $5, $1, $4
and $6, $1, $4
mul $7, $5, $6

$$Throughput = \frac{1}{1000\ ps} = 1\ BIPS$$

$$\text{CPI} = 1$$



lw $1, 8($2)
add $4, $2, $3
sub $5, $1, $4
and $6, $1, $4
mul $7, $5, $6

$$Throughput = \frac{1}{200\ ps} = 5\ BIPS$$

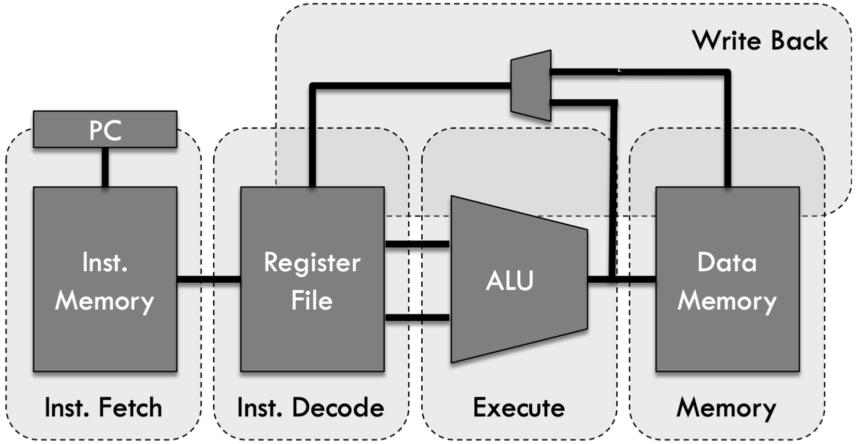
$$\text{CPI} = 1$$

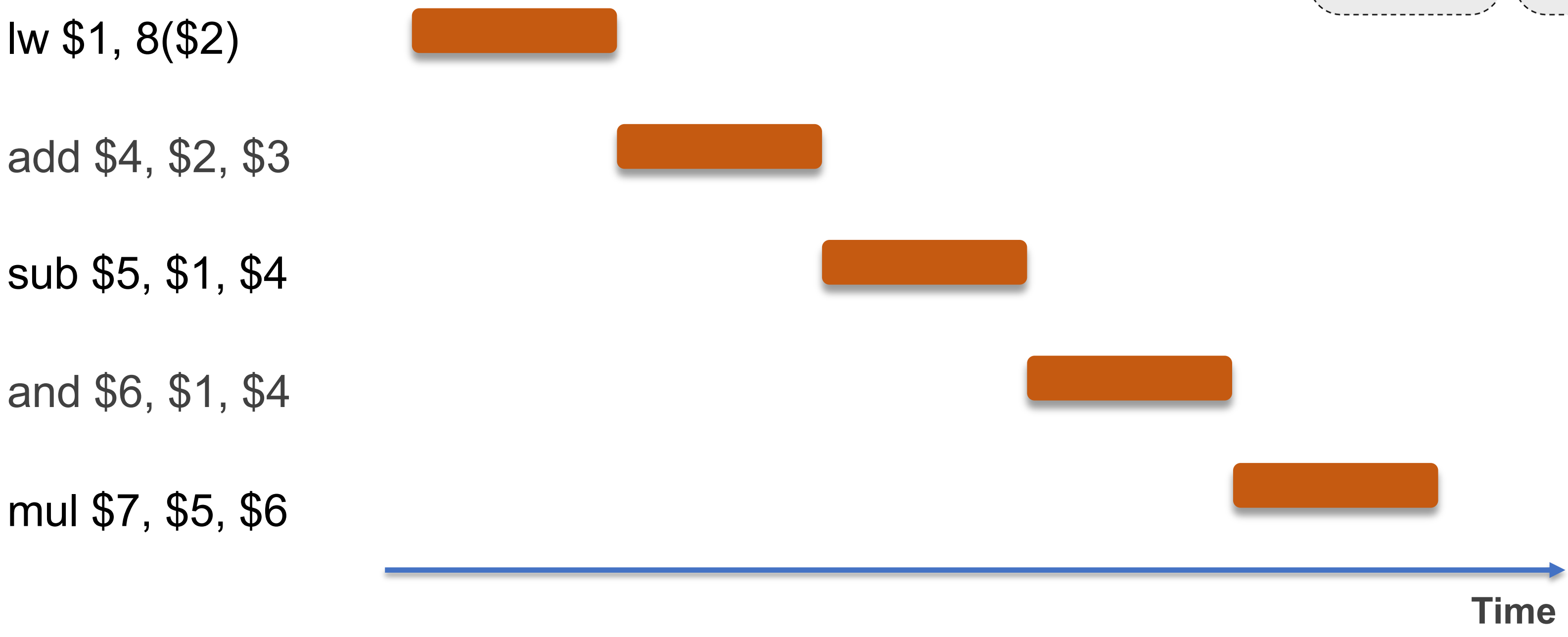
Recall: Single-cycle CPU Performance

- Example Program:

  What is the CPU time for a Cycle Time of 6 ns?

  CT = 6 ns;   CPU Time = 5 x 6 ns = 30 ns





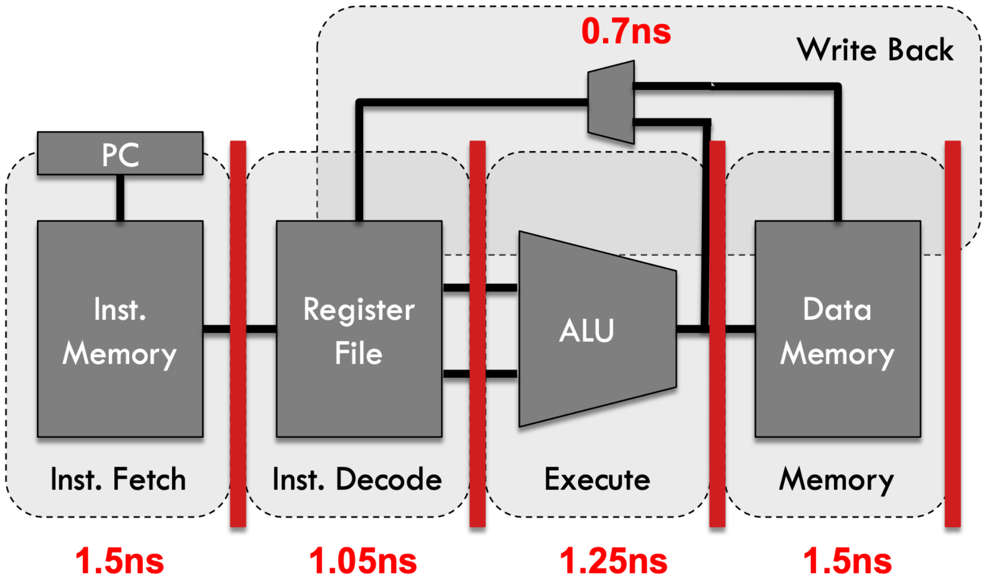
```
lw $1, 8($2)
add $4, $2, $3
sub $5, $1, $4
and $6, $1, $4
mul $7, $5, $6
```

Pipelined CPU Performance

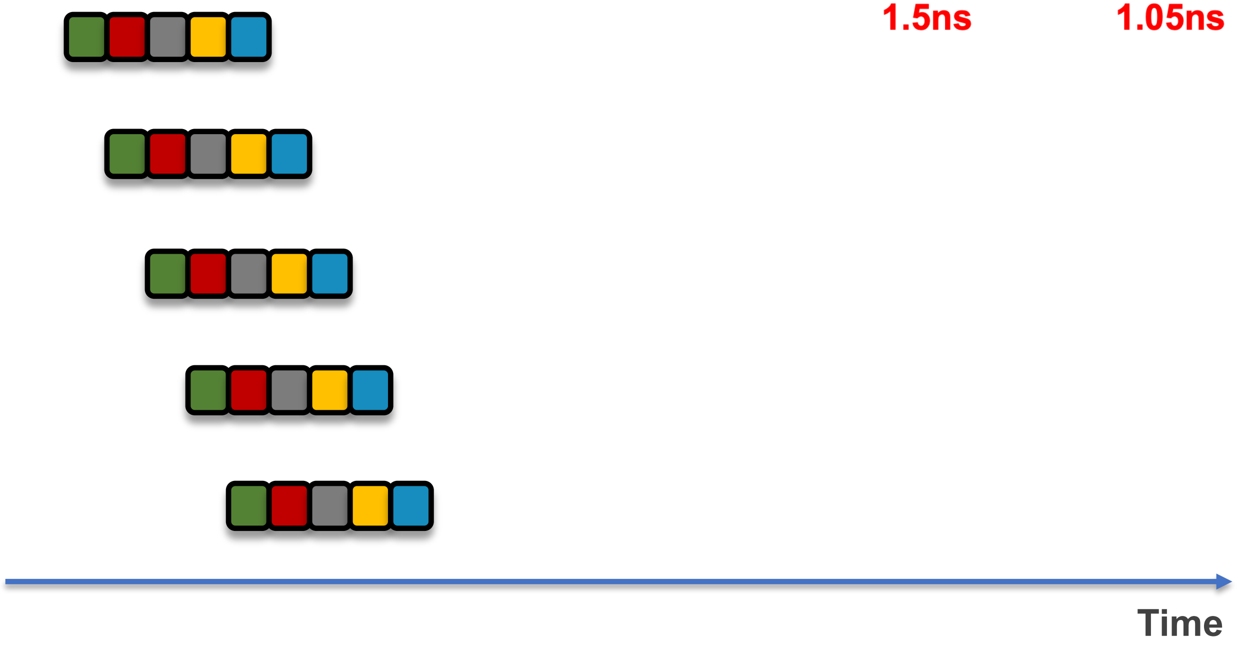
- Example Program:

  What is the CPU time for a Cycle Time of 1.5 ns?

  CT = 1.5 ns;   CPU Time = 9 x 1.5 ns = 13.5 ns



```
lw $1, 8($2)
add $4, $2, $3
sub $5, $1, $4
and $6, $1, $4
mul $7, $5, $6
```



Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

  No, it takes the same or even more time depending on CT

Performance Impacts of Pipelining

- Does it take longer to finish each individual job?

  No, it takes the same or even more time depending on CT

- Does it take shorter to finish a series of jobs?

Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

  No, it takes the same or even more time depending on CT

- Does it take shorter to finish a series of jobs?

  Yes, the throughput has increased by using a 5-stage pipeline

Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

  No, it takes the same or even more time depending on CT

- Does it take shorter to finish a series of jobs?

  Yes, the throughput has increased by using a 5-stage pipeline

- What assumptions were made while answering these questions?

Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

  No, it takes the same or even more time depending on CT

- Does it take shorter to finish a series of jobs?

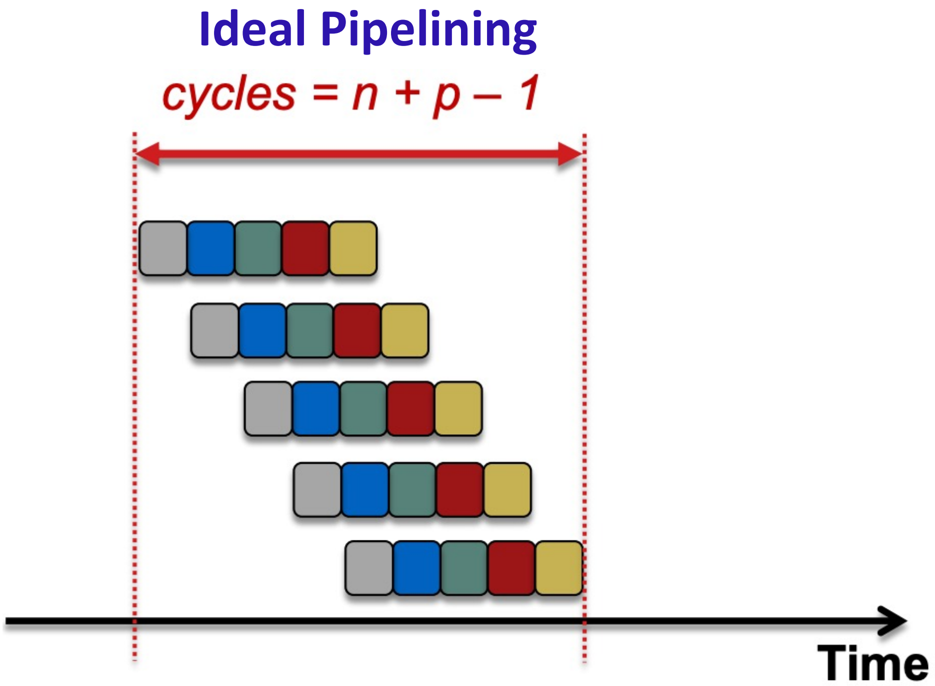
  Yes, the throughput has increased by using a 5-stage pipeline

- What assumptions were made while answering these questions?
  - No data dependencies

Performance Impacts of Pipelining

- Stall cycles to resolve data dependencies

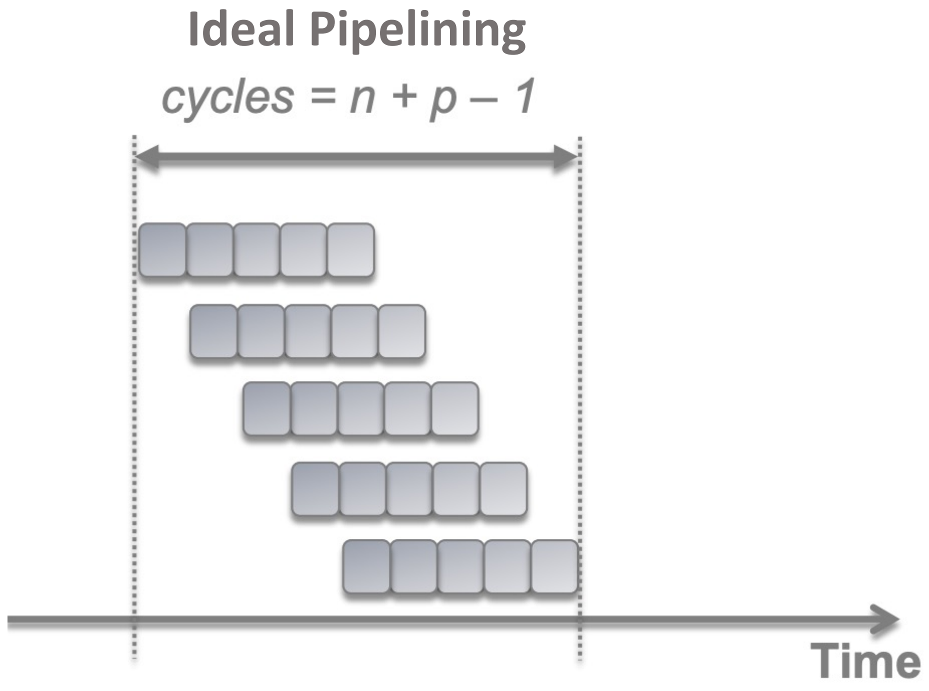
$$n = \#\ instructions, \qquad p = \#\ pipeline\ stages$$

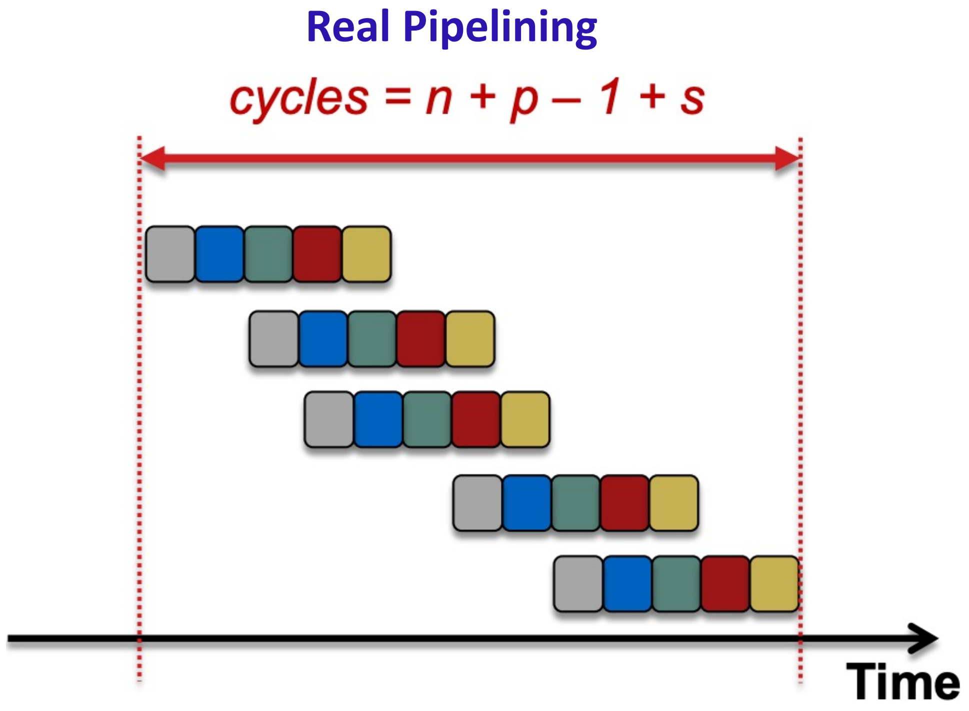


Performance Impacts of Pipelining

- Stall cycles to resolve data dependencies

$$n = \#\ instructions, \qquad p = \#\ pipeline\ stages, \qquad s = \#stall\ cycles$$





Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

  No, it takes the same or even more time depending on CT

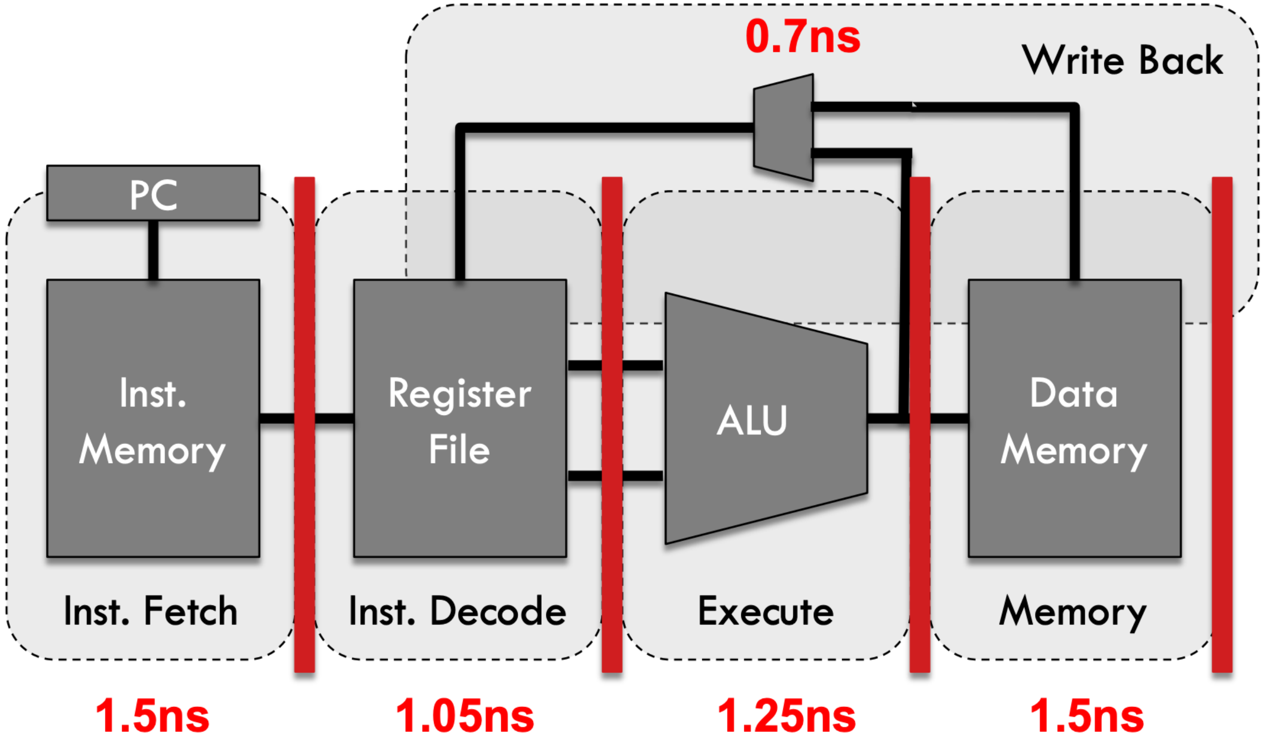
- Does it take shorter to finish a series of jobs?

  Yes, the throughput has increased by using a 5-stage pipeline

- What assumptions were made while answering these questions?
  - No data dependencies

Performance Impacts of Pipelining

- Latch overhead in pipelining



Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

  No, it takes the same or even more time depending on CT

- Does it take shorter to finish a series of jobs?

  Yes, the throughput has increased by using a 5-stage pipeline

- What assumptions were made while answering these questions?
  - No data dependencies
  - No latch overhead

Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

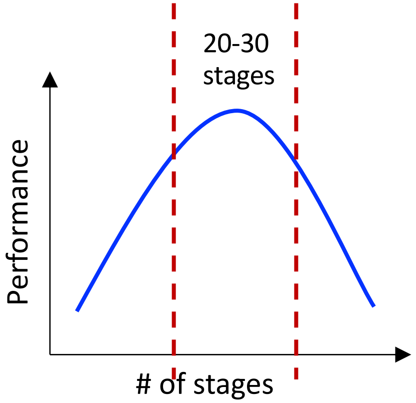
  No, it takes the same or even more time depending on CT

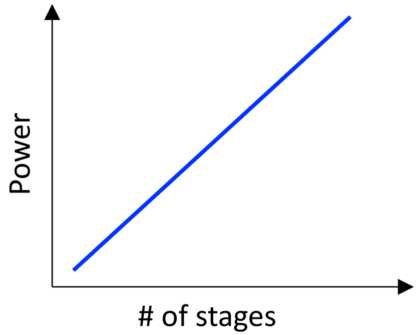
- Does it take shorter to finish a series of jobs?

  Yes, the throughput has increased by using a 5-stage pipeline

- What assumptions were made while answering these questions?
  - No data dependencies
  - No latch overhead

- Is a 50-stage pipeline better than a 5-stage pipeline?





Performance Impacts of Pipelining

- Does it take shorter to finish each individual job?

  No, it takes the same or even more time depending on CT

- Does it take shorter to finish a series of jobs?

  Yes, the throughput has increased by using a 5-stage pipeline

- What assumptions were made while answering these questions?
  - No data dependencies
  - No latch overhead

- Is a 50-stage pipeline better than a 5-stage pipeline?

  No, performance degrades with more stages due to latch overhead and data dependencies

Quantitative Effects of Pipelining

As a result of pipelining:

- Time in ns per instruction *goes up*
- Each instruction takes *more cycles* to execute
- But…average CPI remains roughly the same
- Clock speed *goes up*
- Total execution time *goes down*, resulting in *lower* average time per instruction
- Under <u>ideal</u> conditions,

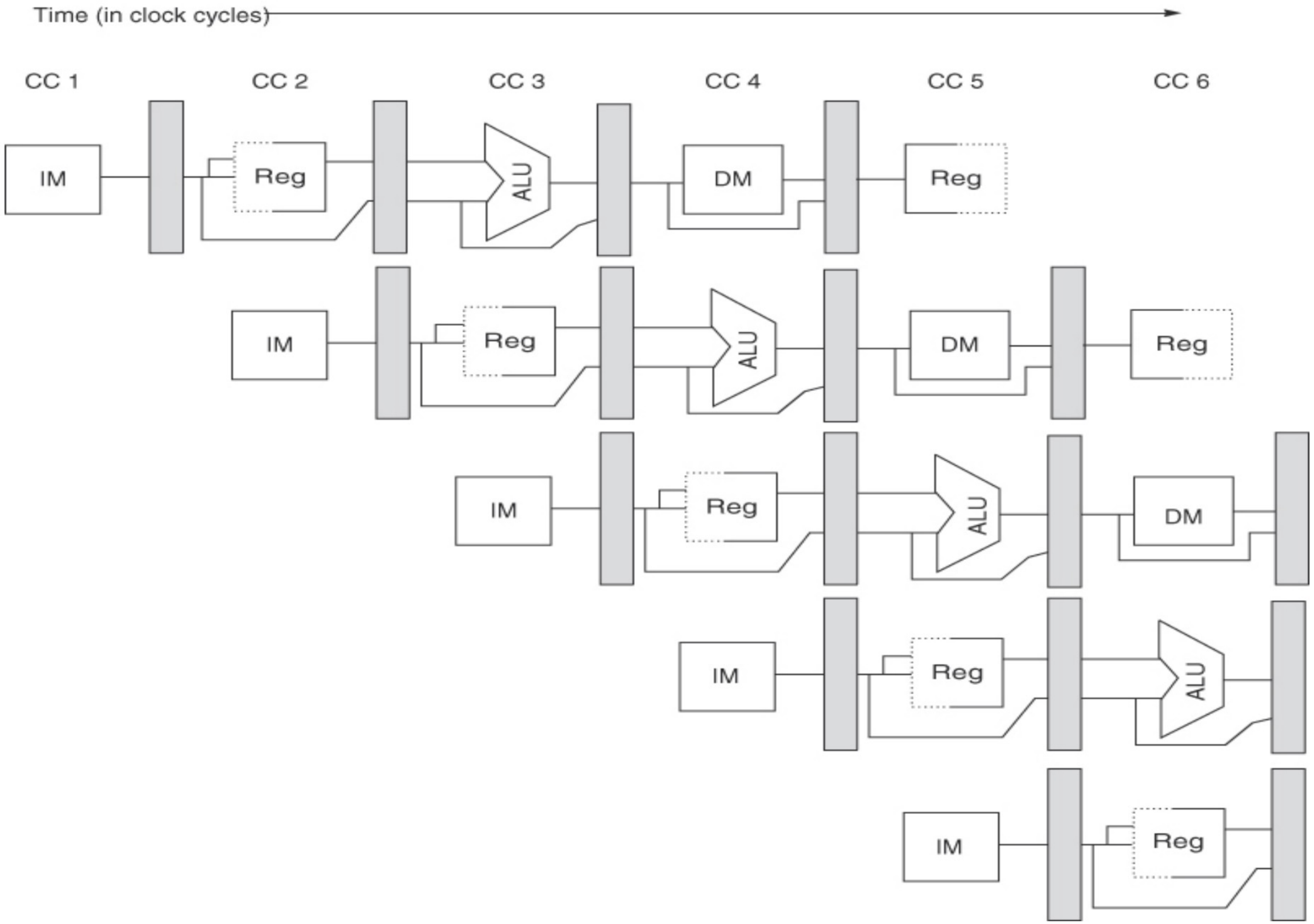
  Speedup

  = ratio of *elapsed times between successive instruction completions*

  = increase in clock speed

  = *number of pipeline stages*

Designing a 5-stage Pipeline

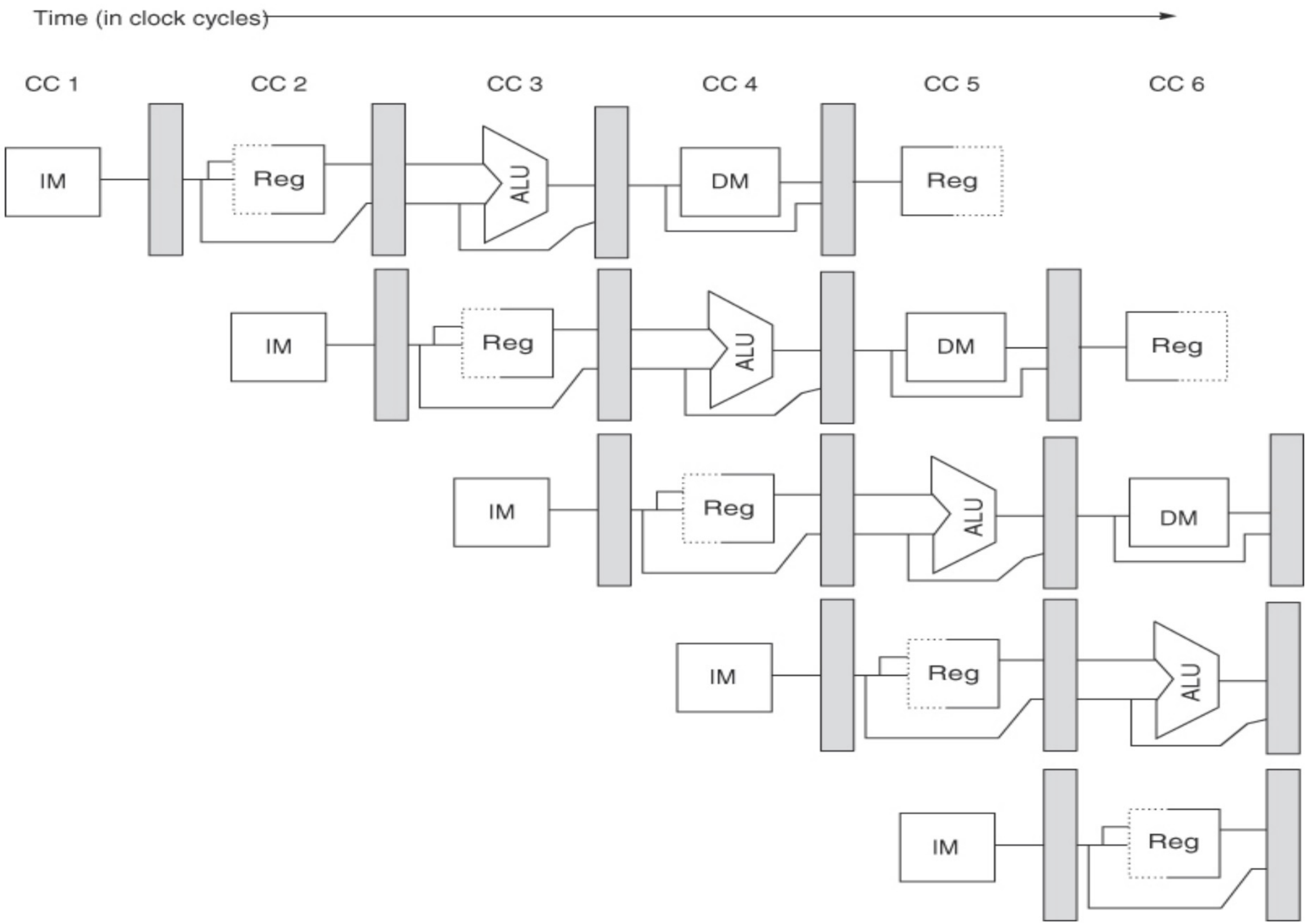


Designing a 5-stage Pipeline

lw $1, 6($2)

add $3, $5, $0

sw $4, 8($6)



Next time…

- Read chapter 4.5 – 4.7 from textbook
- Next lecture
    - Introduction to pipelining hazards
    - Structural Hazards
    - Data Hazards
    - Resolving Structural and Data Hazards

maheshl@cs.utah.edu