pollev.com/kakiryan

COMP311: *COMPUTER ORGANIZATION!*

Lecture 25: A 5-stage Pipeline & Performance Analysis

tinyurl.com/comp311-fa25

SIT IN YOUR ASSIGNED SEAT FROM LAST QUIZ!

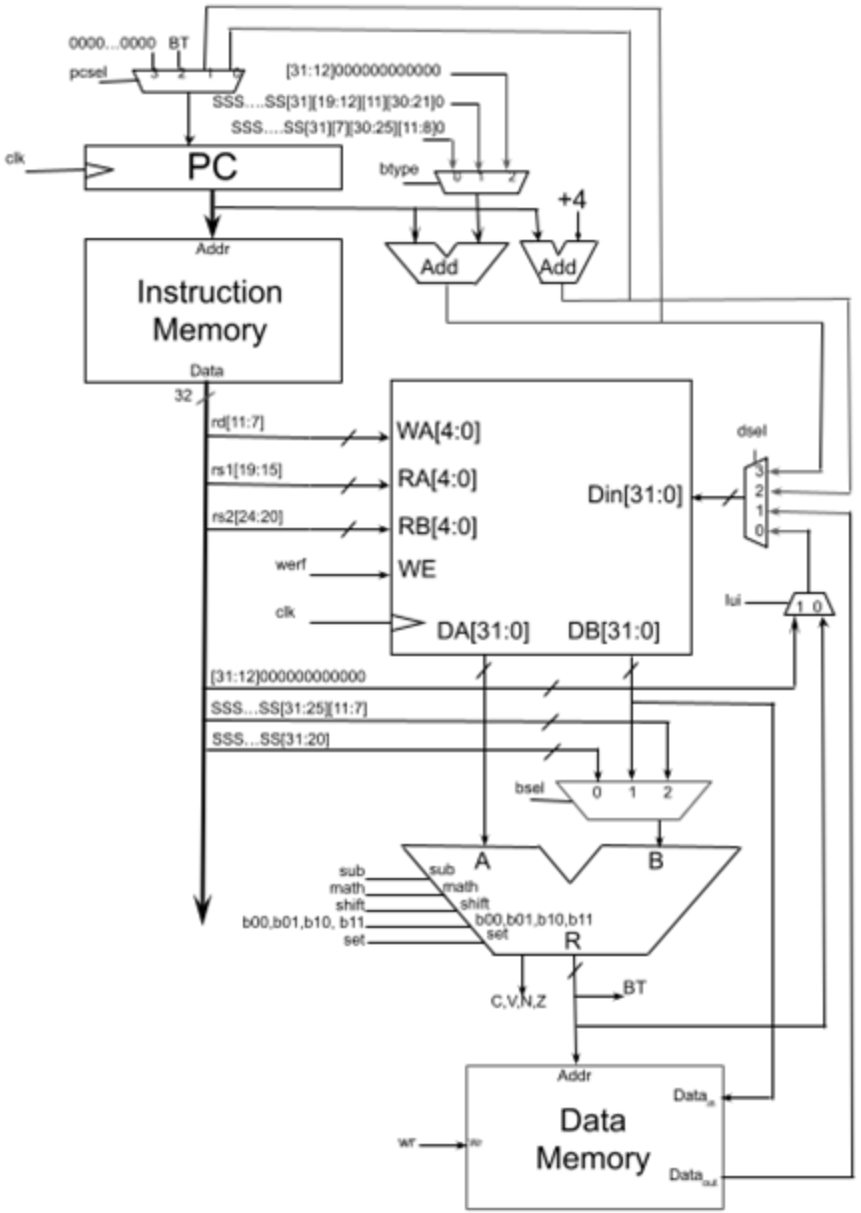
PIPELINING REVIEW!

How to Pipeline This?

A CPU is a digital circuit like any other. Thus, we should be able to pipeline it to increase its throughput.

However, there are a few tricky issues.

- It already has registers that get updated on each clock (register file and PC)
- It has feedback, the ALU or Data Memory outputs are routed back to the register file



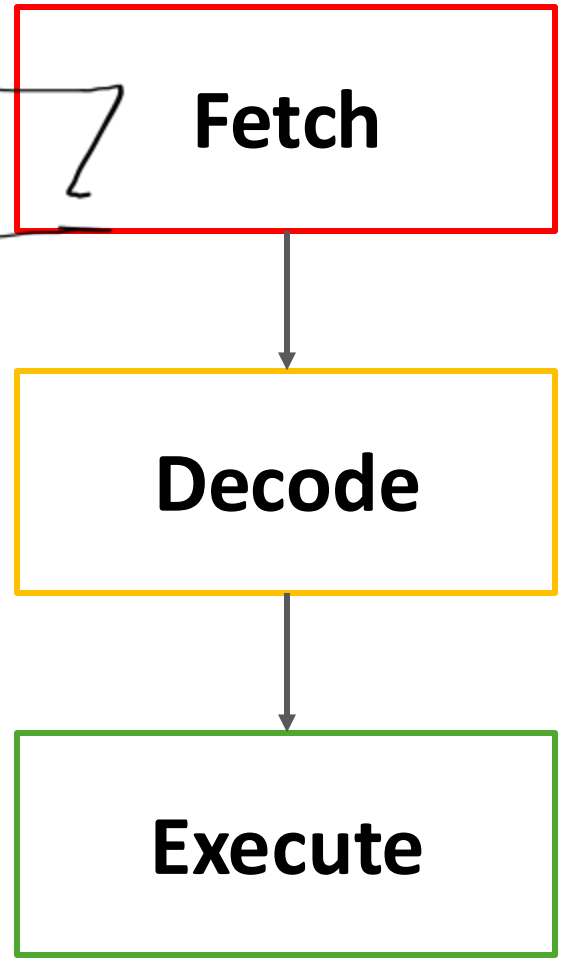
Our Goal

A simple 3-stage ~~pipeline~~:

Fetch:
Instruction memory access

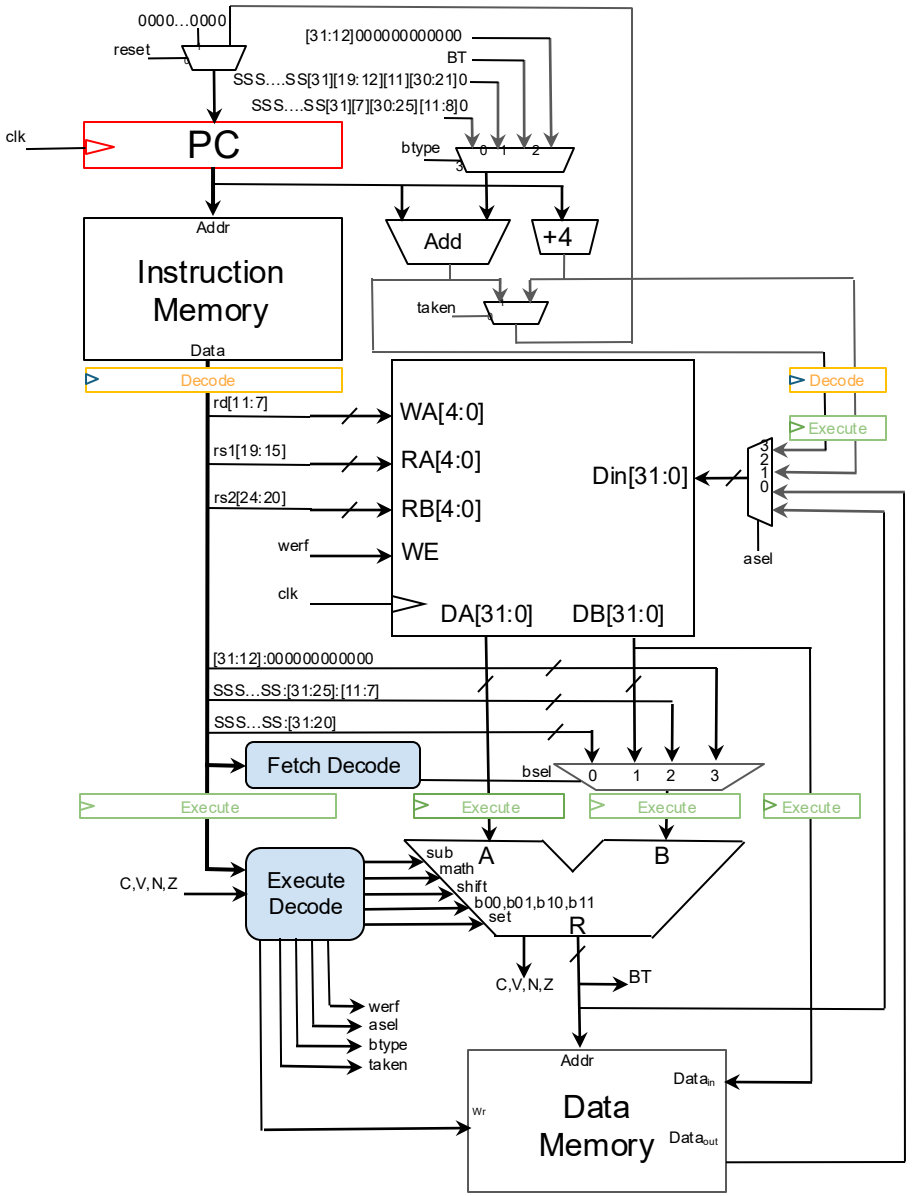
Decode:
Decode instruction
Get source register operands

Execute:
ALU operation
Write-back destination register



Pipelining the Datapath!

- Fetch, Decode, and Execute Stages
- Instructions are decoded in both the Fetch and Decode stages
- Register ports are “Read” in the Decode stage and “Written” at in the end of
  the Execute stage
- PC+4, and PC relative calculations are “delayed”
  for use in later stages

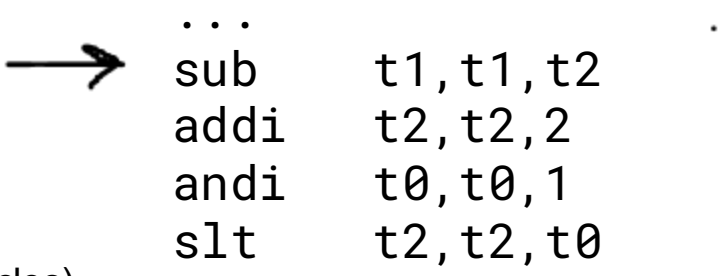


How Instructions flow

Consider the following instruction sequence:

Progress in a three-stage pipeline

Once filled, at every clock there are 3 instructions at various stages of execution.



Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	sub t1,t1,t2					
Decode						
Execute						

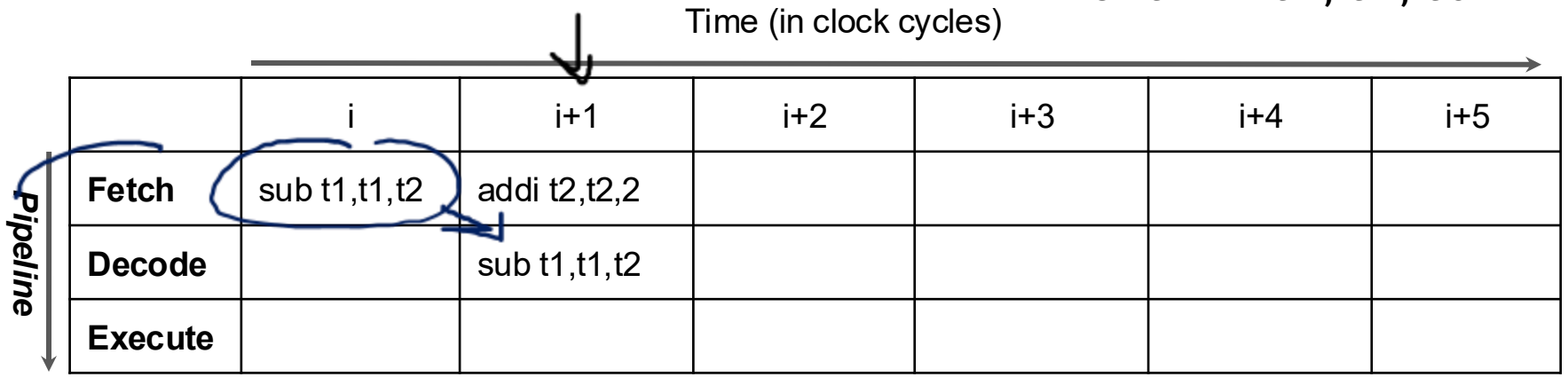
How Instructions flow

Consider the following instruction sequence:

Progress in a three-stage pipeline

Once filled, at every clock there are 3 instructions at various stages of execution.

```
...
sub    t1,t1,t2
addi   t2,t2,2
andi   t0,t0,1
slt    t2,t2,t0
```



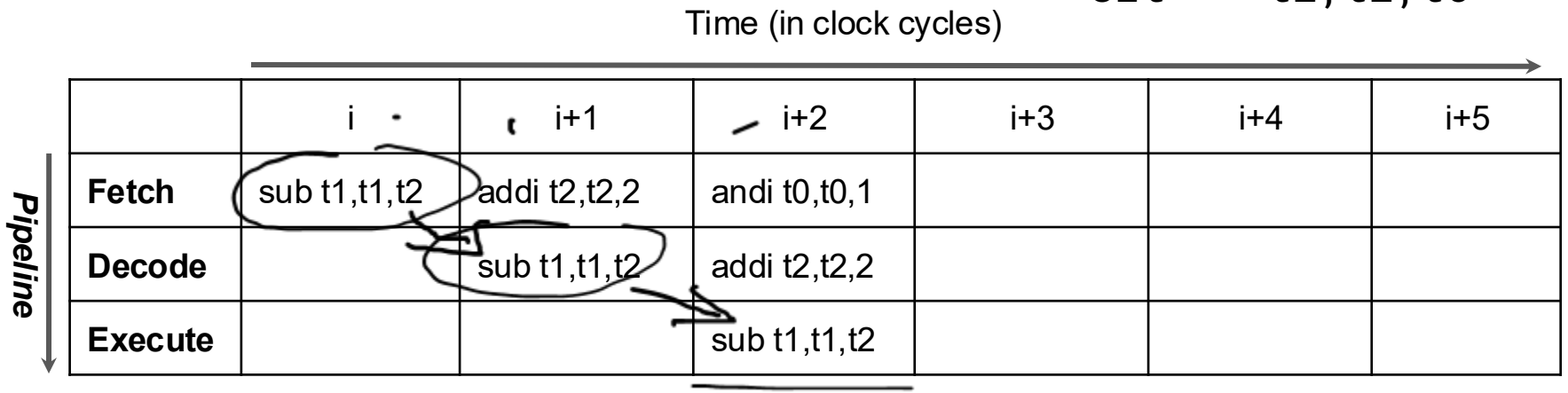
How Instructions flow

Consider the following instruction sequence:

Progress in a three-stage pipeline

Once filled, at every clock there are 3 instructions at various stages of execution.

```
...
sub    t1,t1,t2
addi   t2,t2,2
andi   t0,t0,1
slt    t2,t2,t0
```



How Instructions flow

Consider the following instruction sequence:

Progress in a three-stage pipeline

Once filled, at every clock there are 3 instructions at various stages of execution.

```
...
sub     t1,t1,t2
addi    t2,t2,2
andi    t0,t0,1
slt     t2,t2,t0
```

Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0		
Decode		sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0	
Execute			sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0

Source operands are fetched in this stage

How Instructions flow

Consider the following instruction sequence:

Progress in a three-stage pipeline

Once filled, at every clock there are 3 instructions at various stages of execution.

```
...
sub     t1,t1,t2
addi    t2,t2,2
andi    t0,t0,1
slt     t2,t2,t0
```

Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0		
Decode		sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0	
Execute			sub t1,t1,t2	addi t2,t2,2	andi t0,t0,1	slt t2,t2,t0

Source operands are fetched in this stage

Destination operands are updated in this stage

Simple Instruction flow

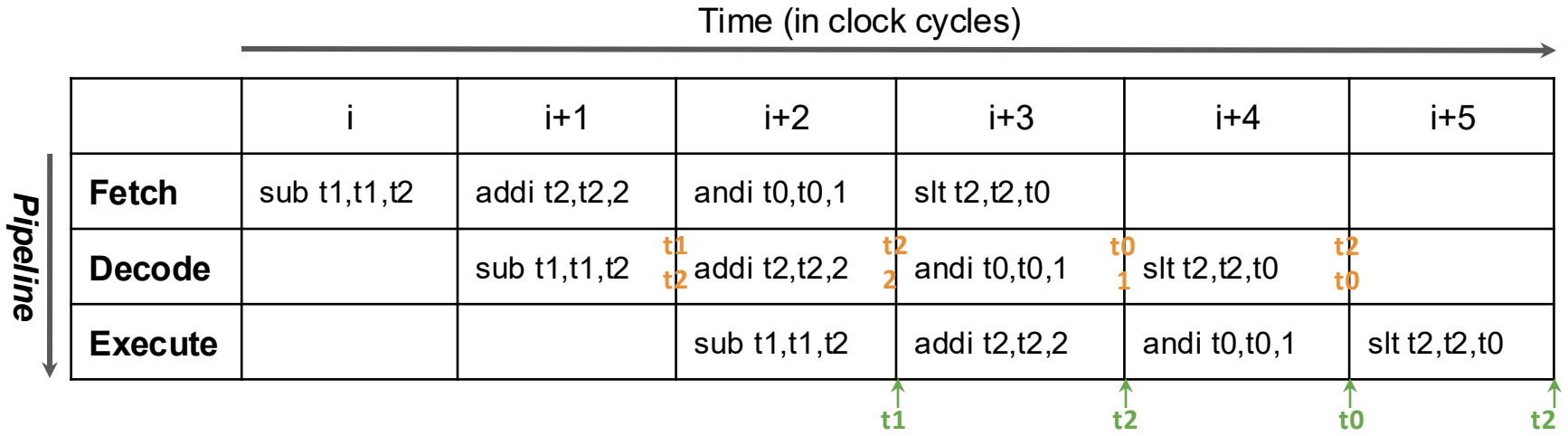
Consider the following instruction sequence:

Instruction becomes available at the end of the Fetch stage

Operands at the end of Decode

Destination and ALU flags are updated at the end of Execute

```
...
sub     t1,t1,t2
addi    t2,t2,2
andi    t0,t0,1
slt     t2,t2,t0
```

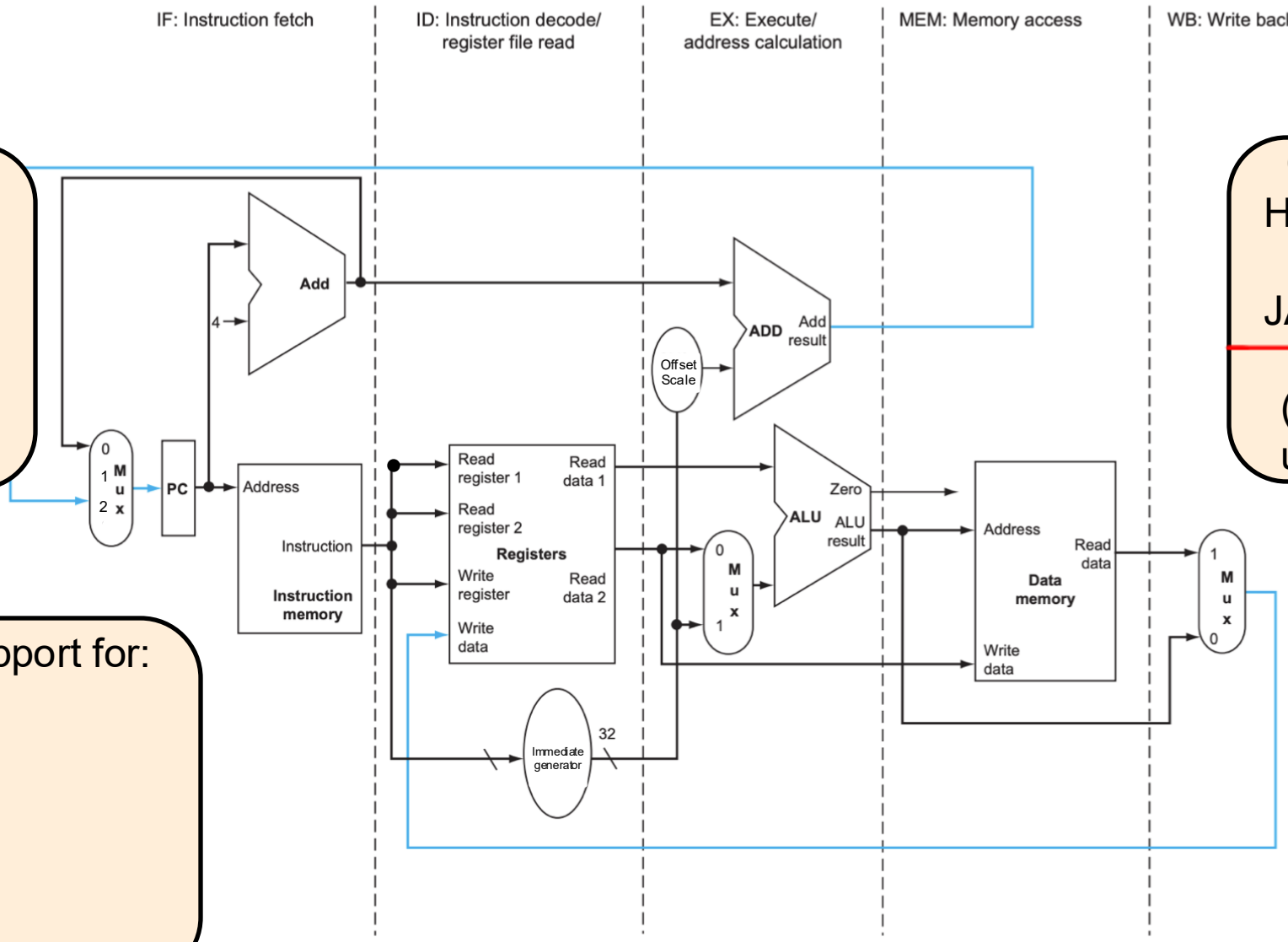


Splitting our Datapath into 5 Stages

- Instruction Fetch
  - Reading the instruction from memory
- Instruction Decode
  - Splitting up the instruction, generating control signals, and reading the register file
  - Jump address calculation
- Execute
  - ALU Operation
  - Branch address calculation
- Memory
  - Reading/writing data memory
- Writeback
  - Writing the result to the register file

Datapath Split into 5 Stages

This version of the full data path has some details abstracted away for readability/tracing purposes.



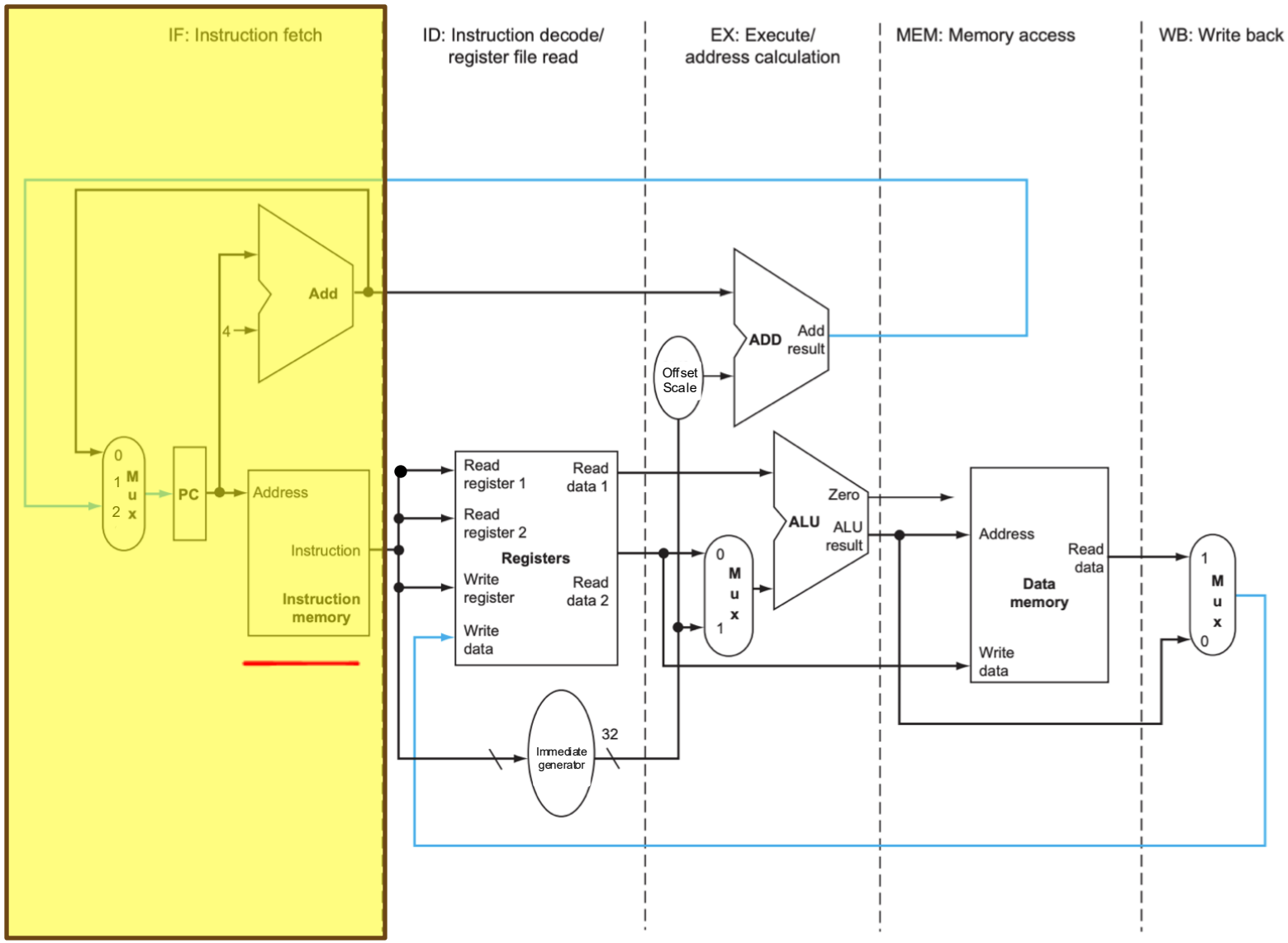
Challenge Q: What HW would need to be added to support JALR? Other types of branches?
(the full ALU control unit also missing…)

This datapath has support for:

- R-type arithmetic
- I-type arithmetic
- Loads
- Stores
- BEQ
- JAL

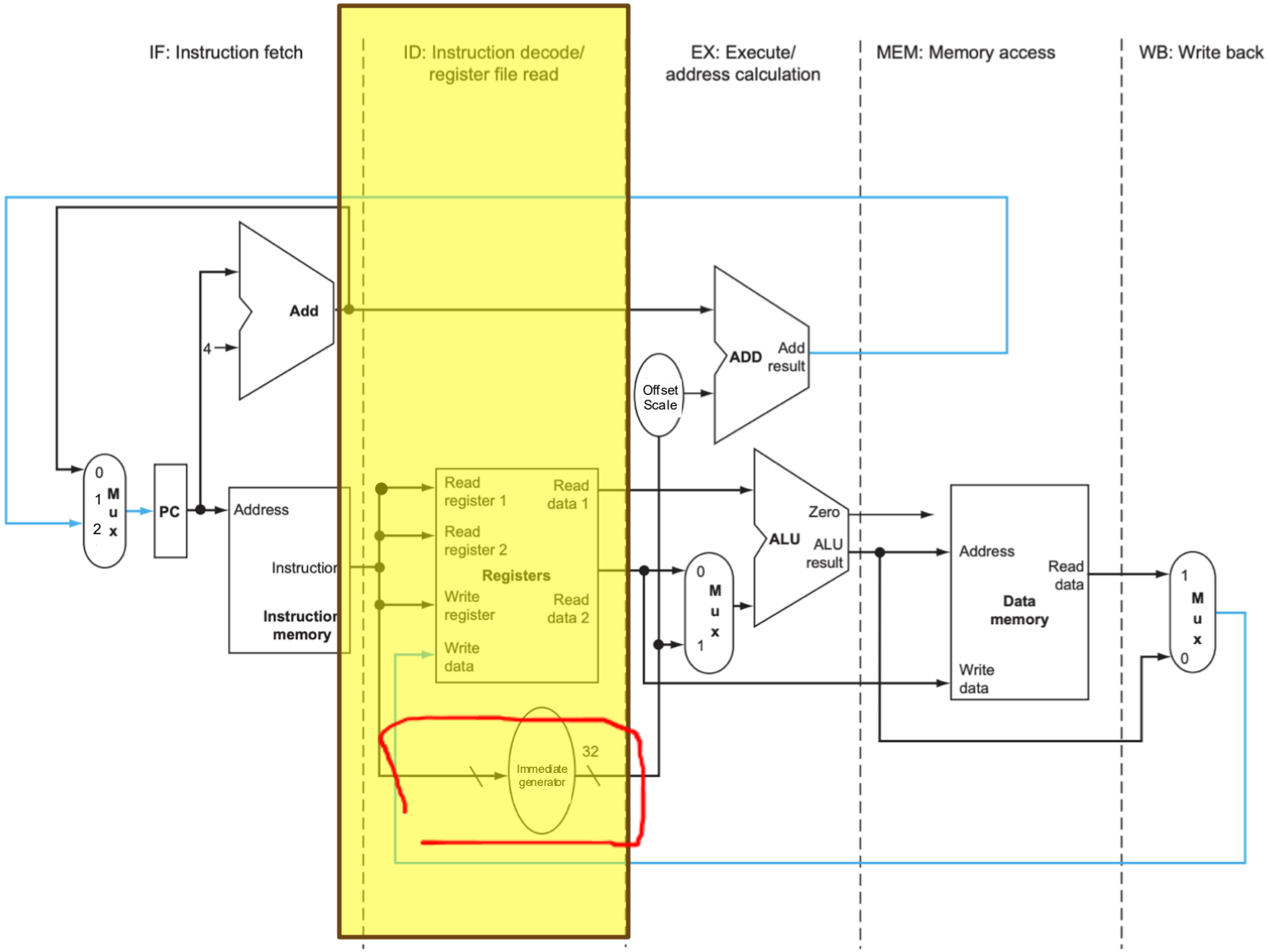
Instruction Fetch

Reading the instruction from memory



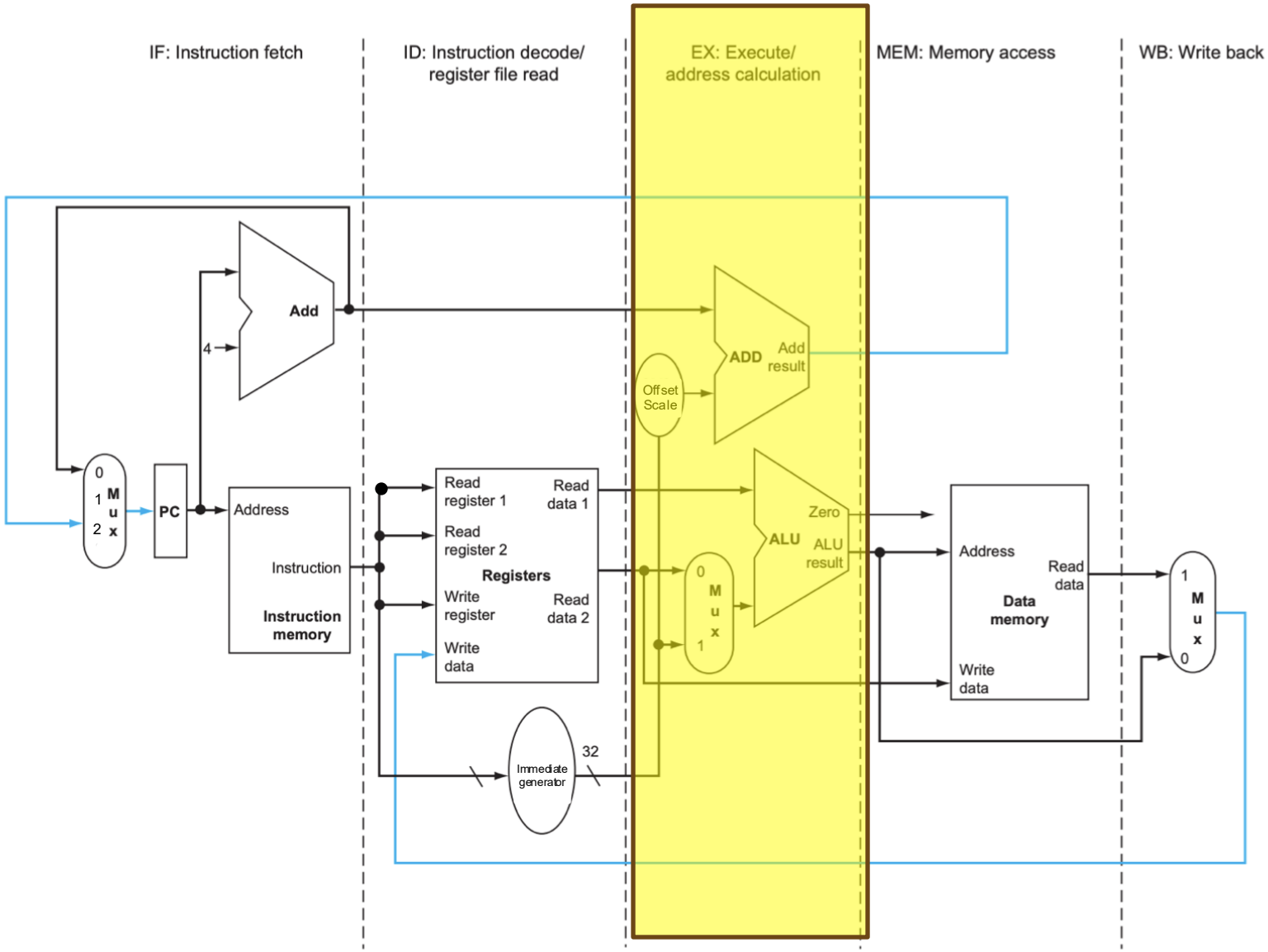
Instruction Decode

- Splitting up the instruction fields
- Generating control signals
- Reading the register file
- Jump address calculation



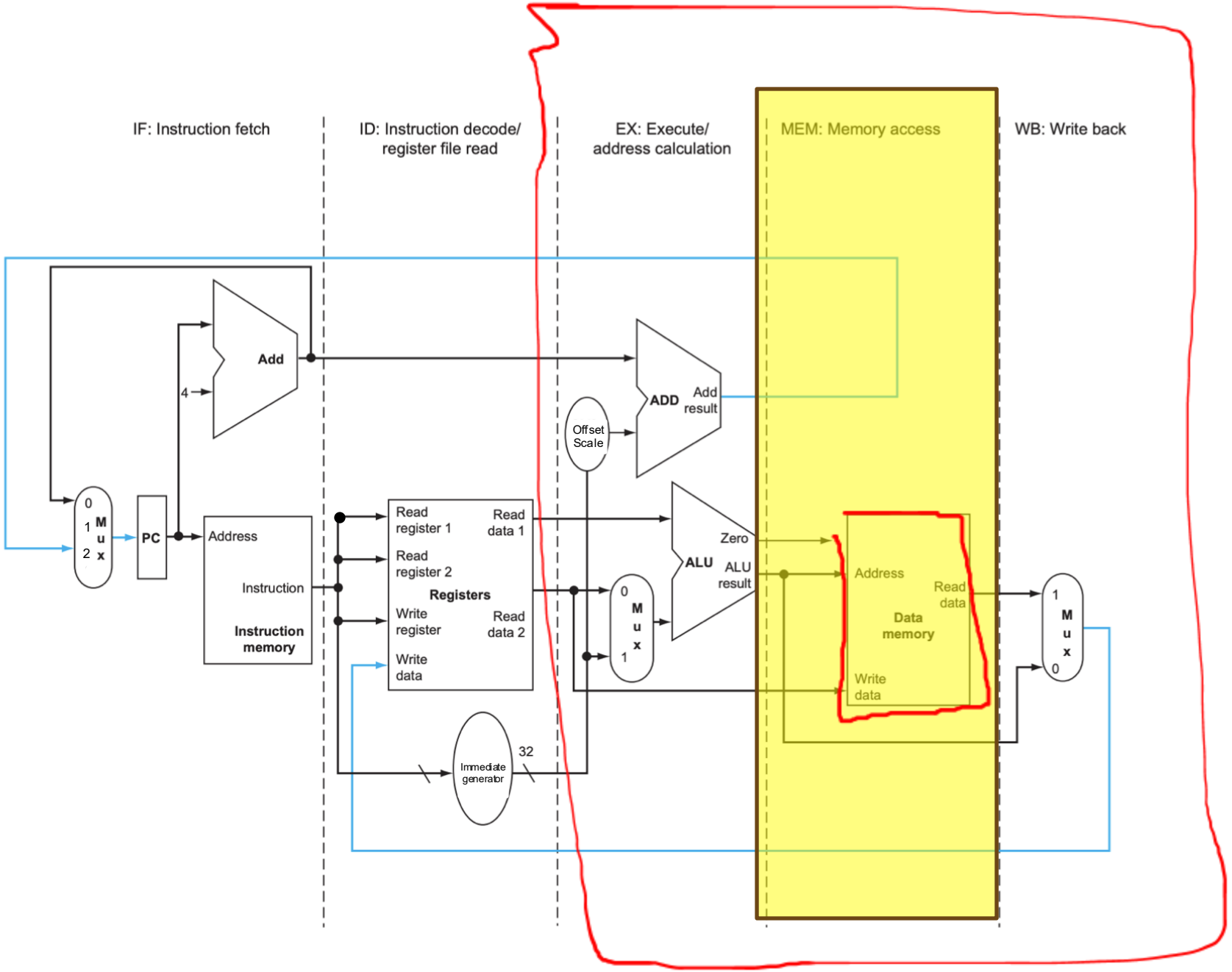
Execute

- ALU Operation
- Branch address calculation



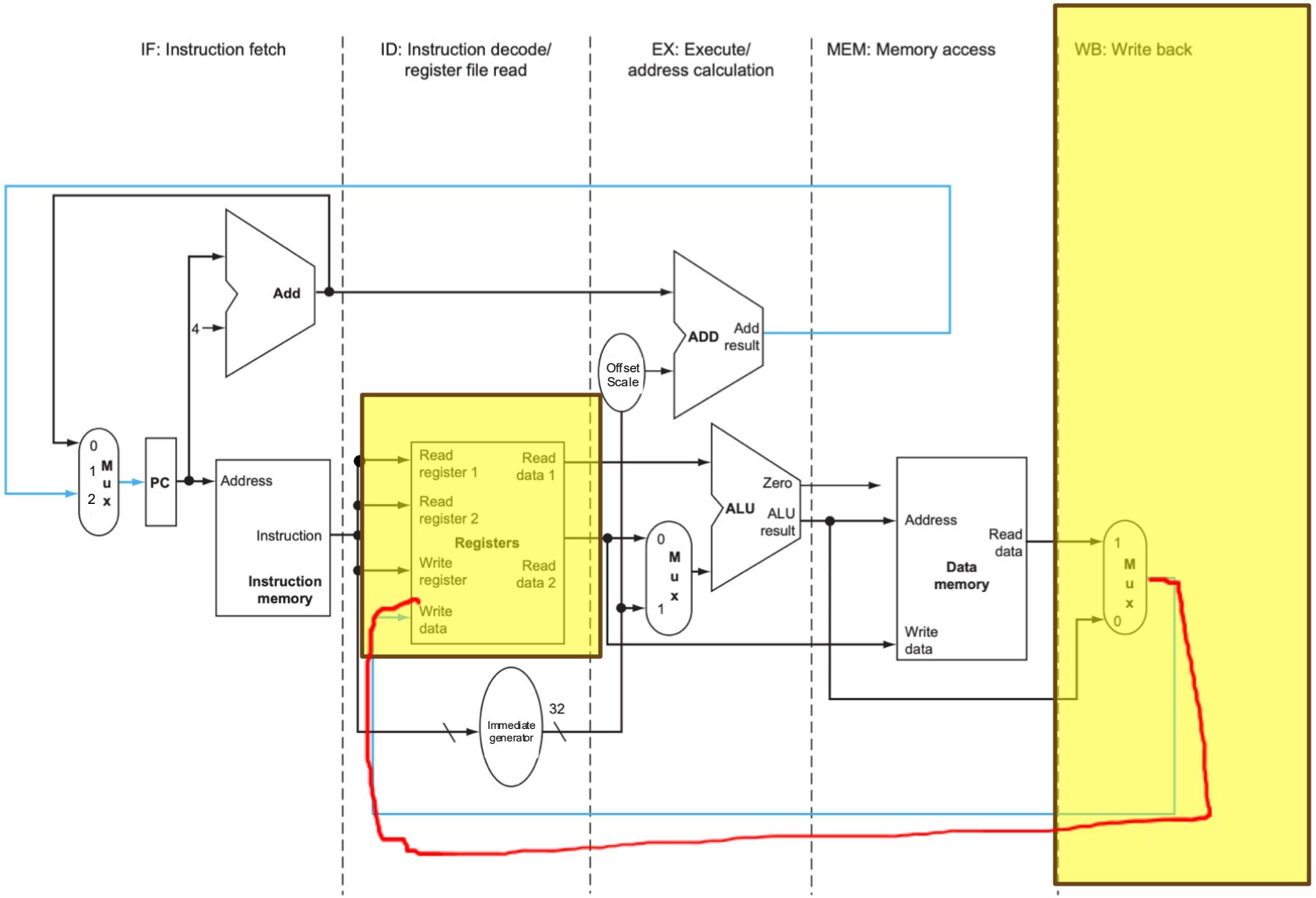
Memory

Reading/writing data memory

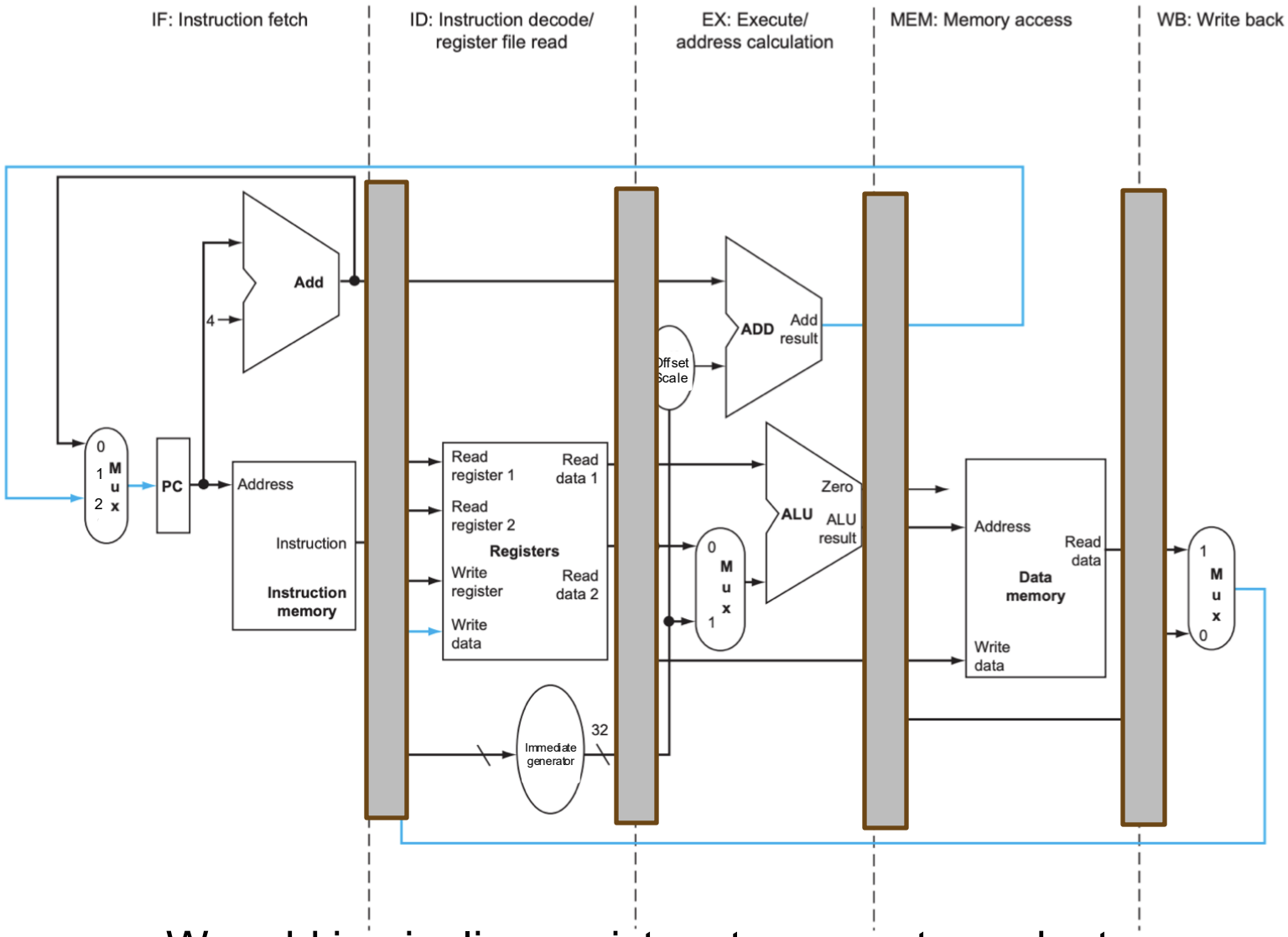


Writeback

Writing the result to the register file



5-Stage Pipeline



We add in pipeline registers to separate each stage

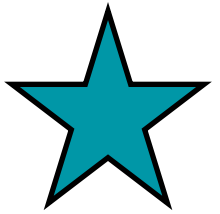
Stages Used by Each Instruction

Instruction	Fetch	Decode	Execute	Memory	Writeback
arithmetic and logical (add, addi, or, etc)					
sw					
lw					
branch					
j					

Stages Used by Each Instruction

Instruction	Fetch	Decode	Execute	Memory	Writeback
arithmetic and logical (add, addi, or, etc)	X	X	X		X
sw					
lw					
branch					
j					

Stages Used by Each Instruction



Instruction	Fetch	Decode	Execute	Memory	Writeback
arithmetic and logical (add, addi, or, etc)	X	X	X		X
sw	X	X	X	X	
lw	X	X	X	X	X
branch	X	X	X		
j	X	X			

Time Needed for Each Stage

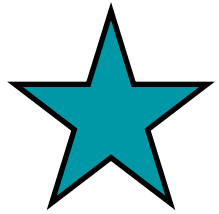
	Fetch	Decode	Execute	Memory	Writeback
Time	200ps	100ps	200ps	200ps	100ps

Time Needed for Each Instruction

	Fetch	Decode	Execute	Memory	Writeback
Time	200ps	100ps	200ps	200ps	100ps

Instruction	Fetch	Decode	Execute	Memory	Writeback	Total Time
arithmetic and logical (add, addi, or, etc)	X	X	X		X	200ps + 100ps + 200ps + 100ps = 600ps
sw	X	X	X	X		700ps
lw	X	X	X	X	X	
branch	X	X	X			
j	X	X				

Time Needed for Each Instruction



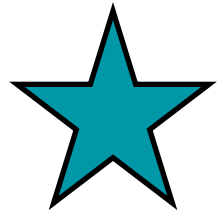
	Fetch	Decode	Execute	Memory	Writeback
Time	200ps	100ps	200ps	200ps	100ps

Instruction	Fetch	Decode	Execute	Memory	Writeback	Total Time
arithmetic and logical (add, addi, or, etc)	X	X	X		X	200ps + 100ps + 200ps + 100ps = 600ps
sw	X	X	X	X		200 + 100 + 200 + 200 = 700ps
lw	X	X	X	X	X	200 + 100 + 200 + 200 + 100 = 800ps
branch	X	X	X			200 + 100 + 200 = 500ps
j	X	X				200 + 100 = 300ps

Min Clock Period

- What is the minimum clock period that will work for all instructions?
  - Assuming a clock-to-q delay of 20ps

Minimum Clock Period



- To execute an arithmetic or logical (add, addi, or, etc) instruction, the clock period must be at least 620ps
- To execute a sw instruction, the clock period must be at least 720ps
- To execute a lw instruction, the clock period must be at least 820ps
- To execute a branch instruction, the clock period must be at least 520ps
- To execute a jump instruction, the clock period must be at least 320ps

- Load word is the slowest instruction
  - Therefore, the shortest clock period that will work for all instructions is 820ps

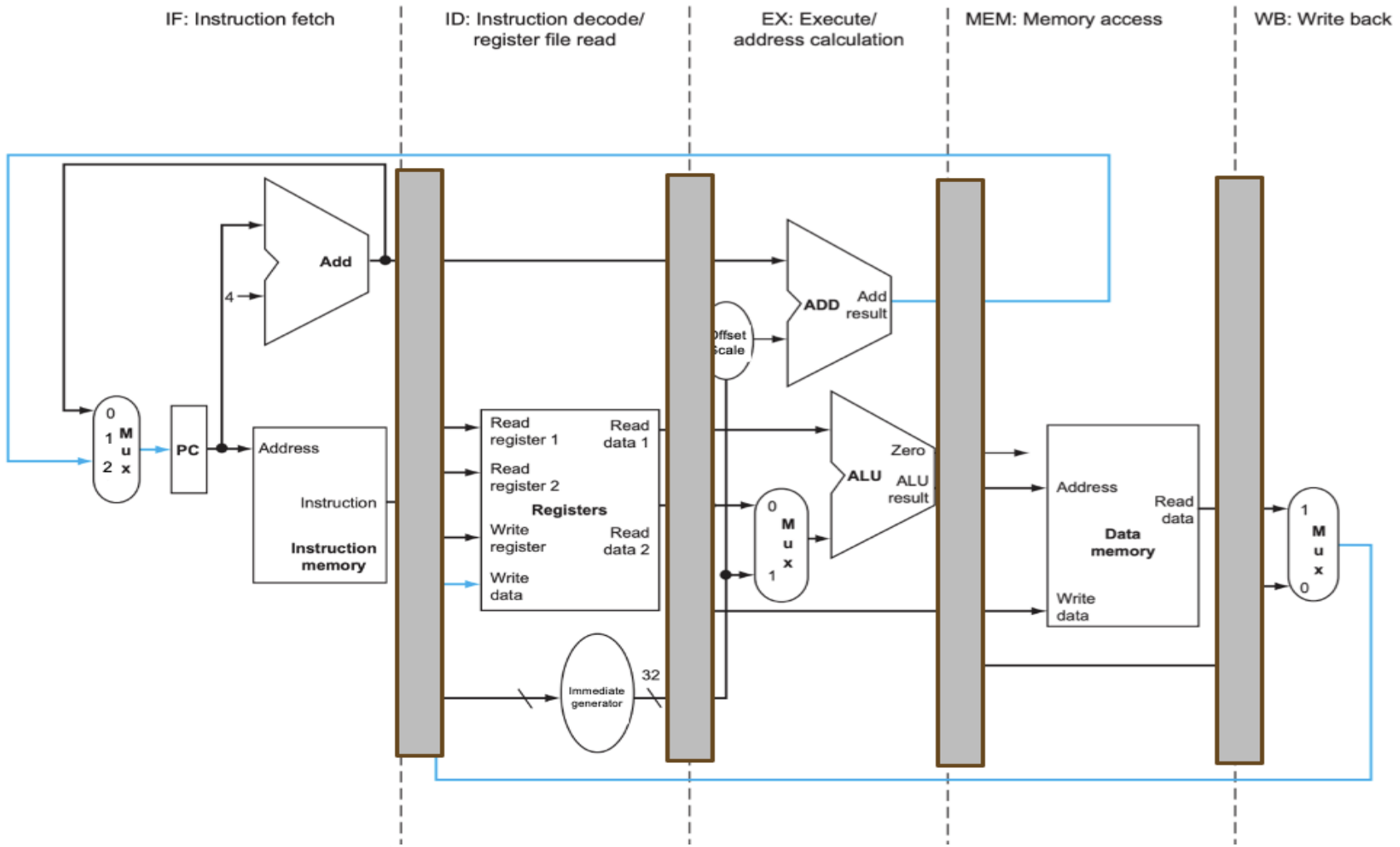
Single Cycle Datapath Timing

- It takes one clock cycle to execute one instruction
- If we need to execute 100 instructions, it will take 100 clock cycles
- If we set our clock period to 820ps, it will take
  - 820ps to execute one instruction
  - 82,000ps to execute 100 instructions

Executing Instructions on Our Pipelined Datapath

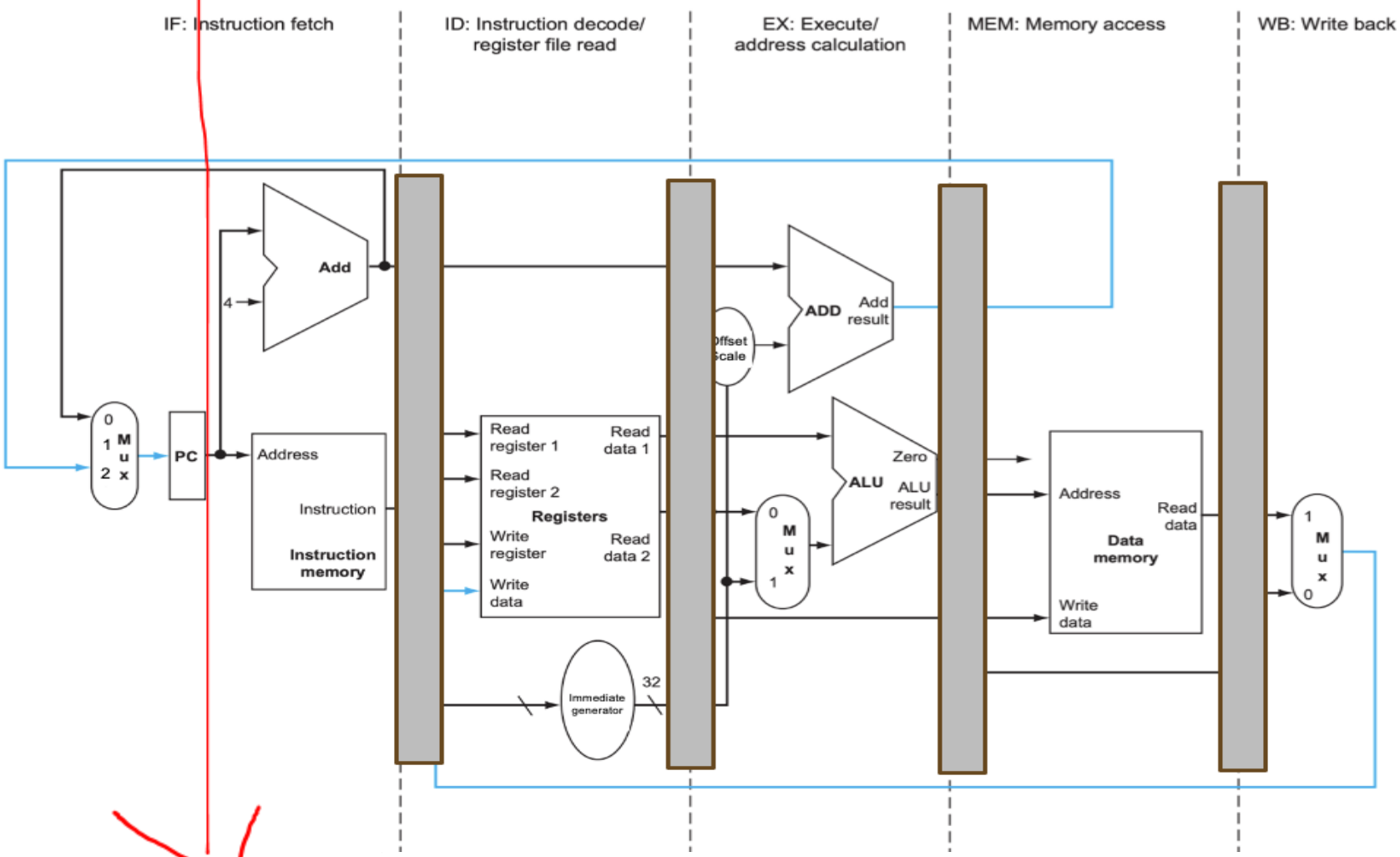
```
add x8,   x6,   x7
lw  x9,   100(x0)
sub x8,   x9,   x20
sw  x10,  0(x20)
or  x21,  x11,  x12
```

Cycle 0




```
add x8, x6, x7
```

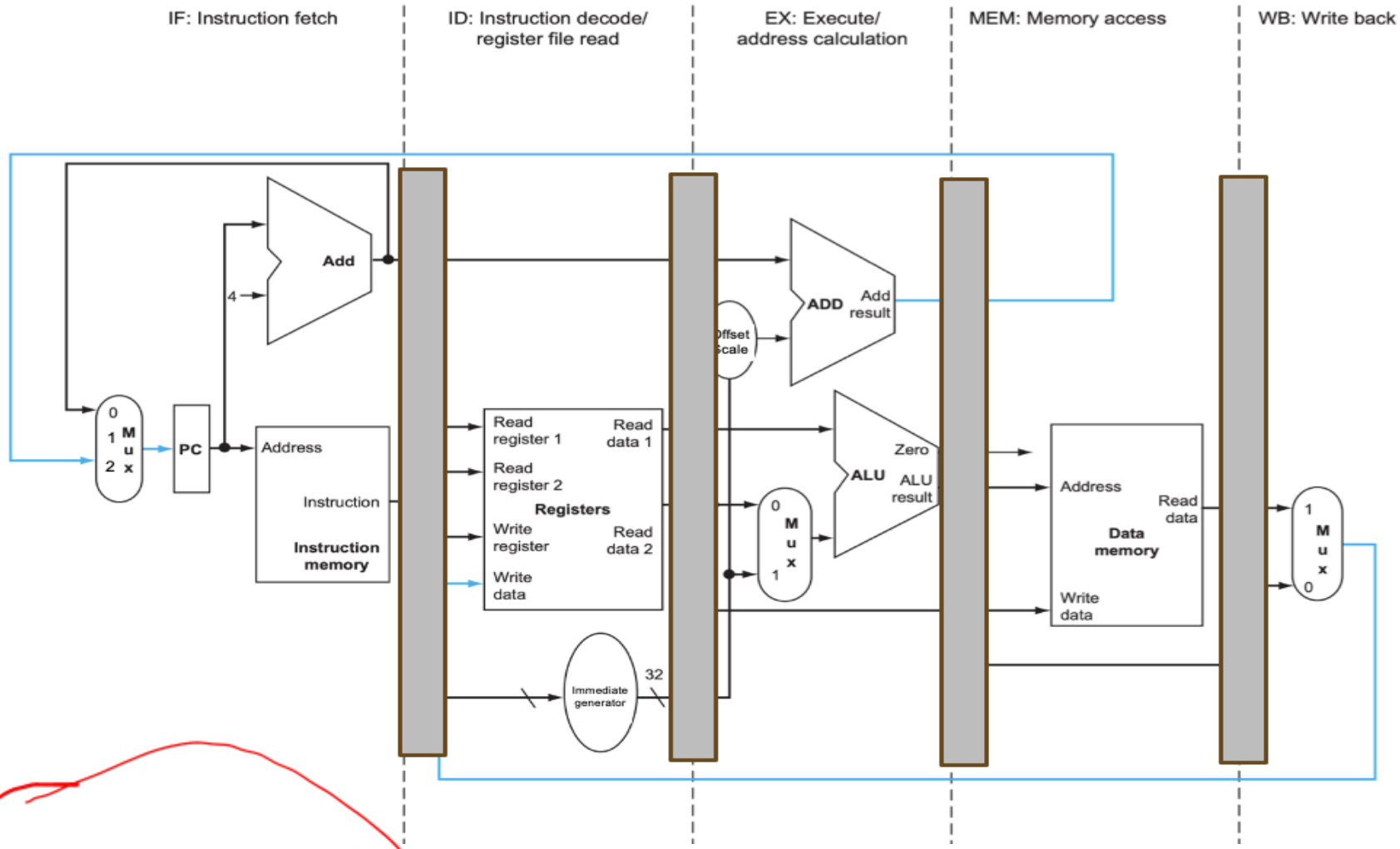
Cycle 1



`lw x9, 100(x0)`

`add x8, x6, x7`

Cycle 2

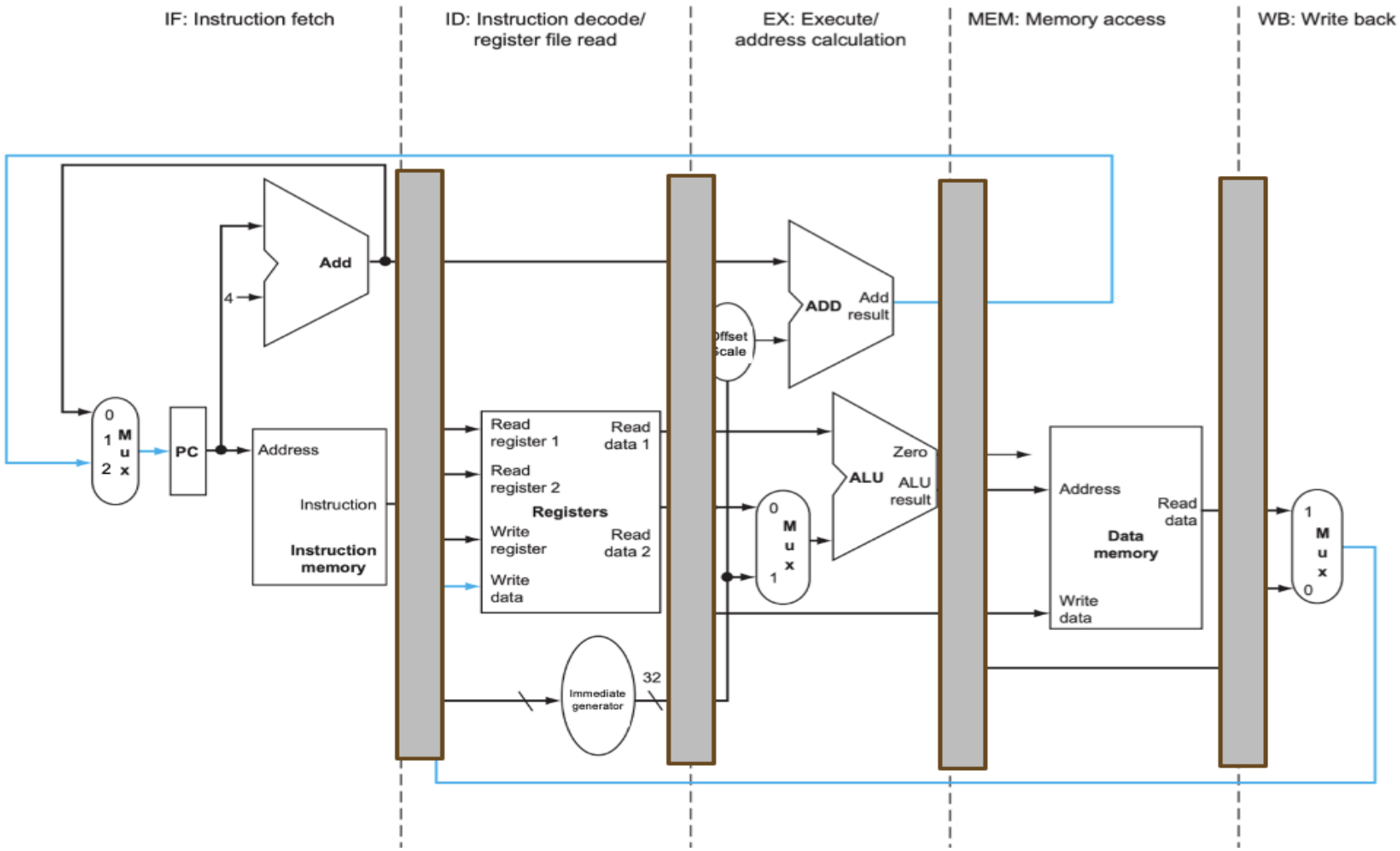



```
sub x8, x9, x20
```

```
lw x9, 100(x0)
```

```
add x8, x6, x7
```

Cycle 3

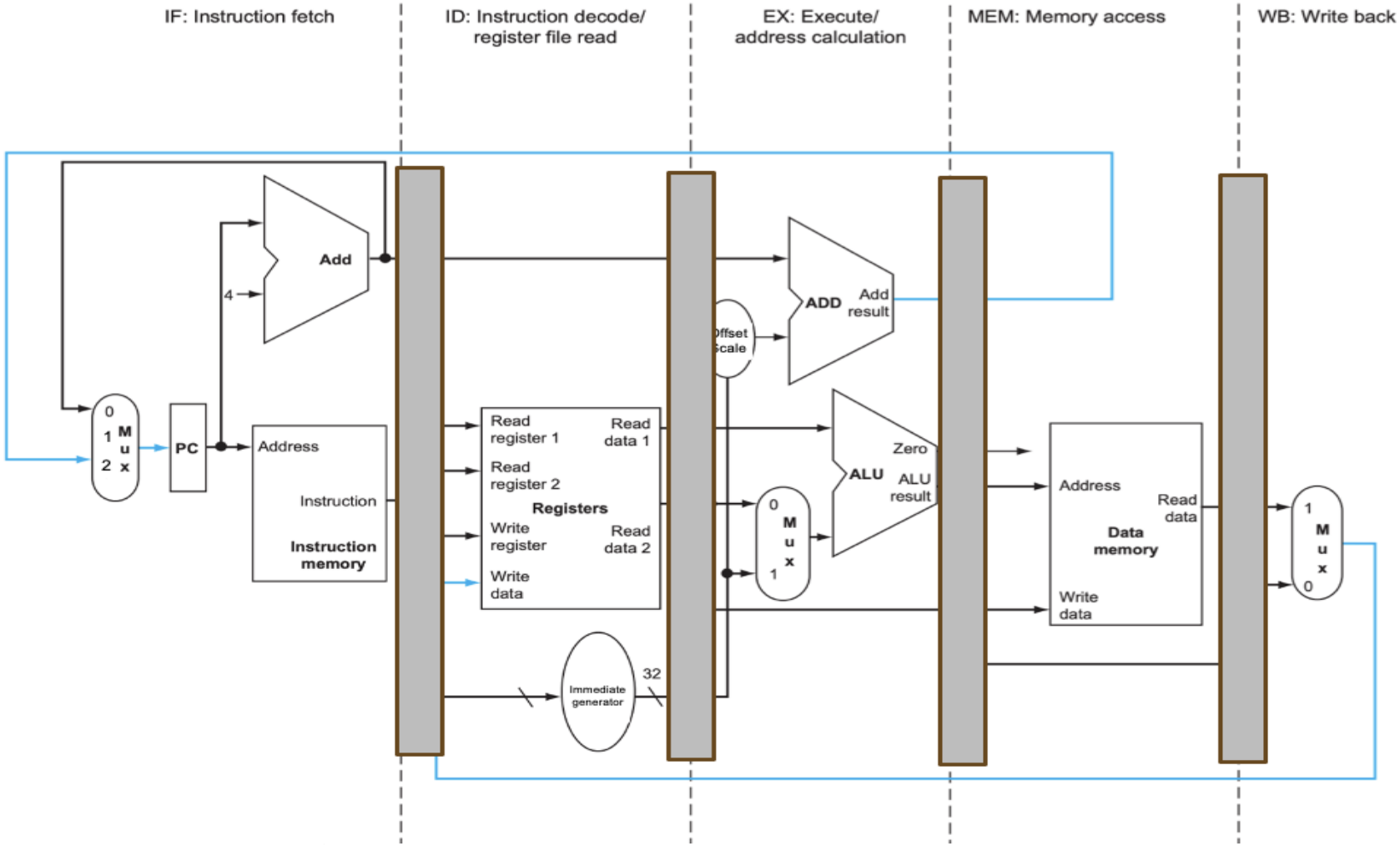


sw x10, 0(x20)

sub x8, x9, x20

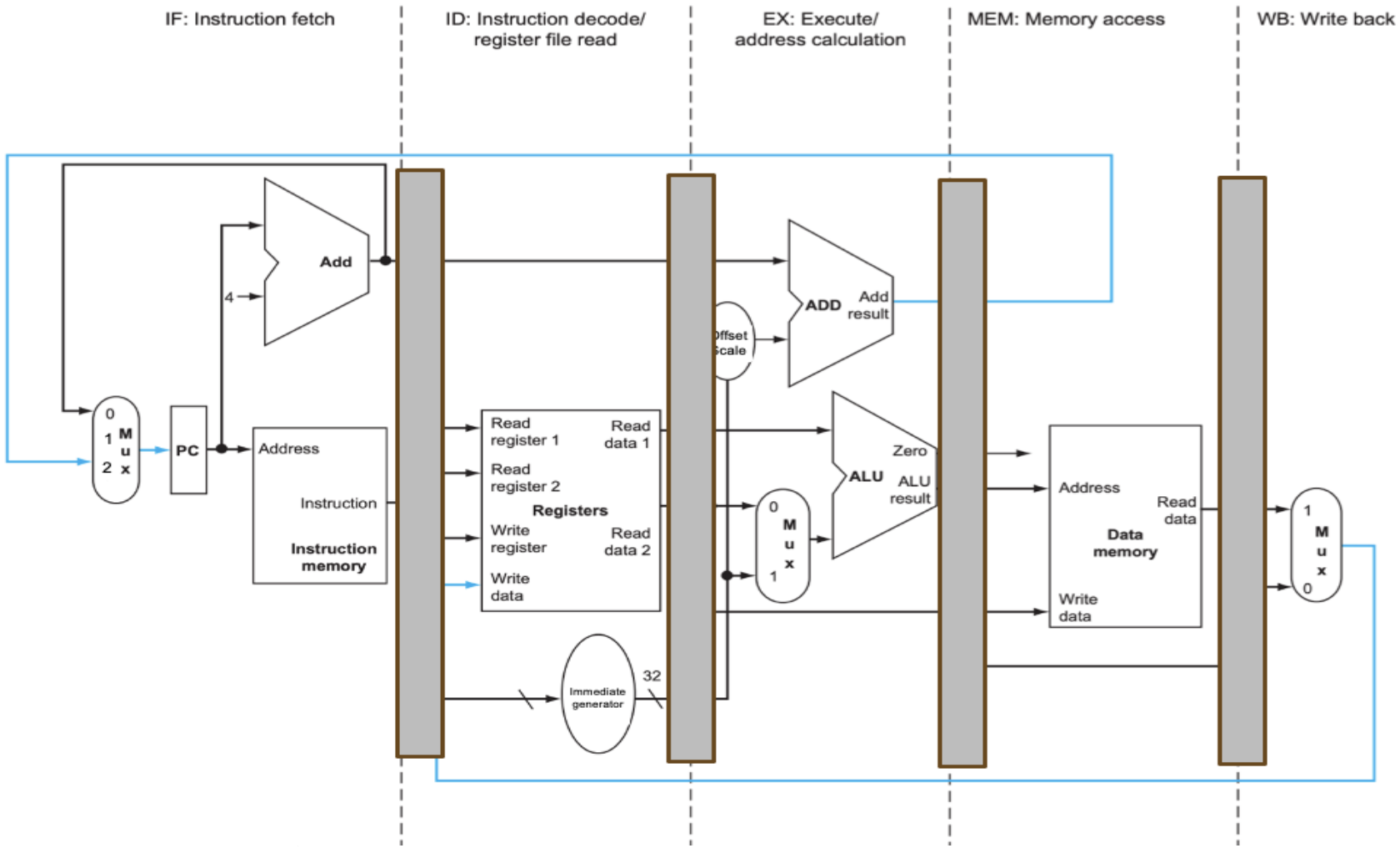
lw x9, 100(x0) add x8, x6, x7

Cycle 4



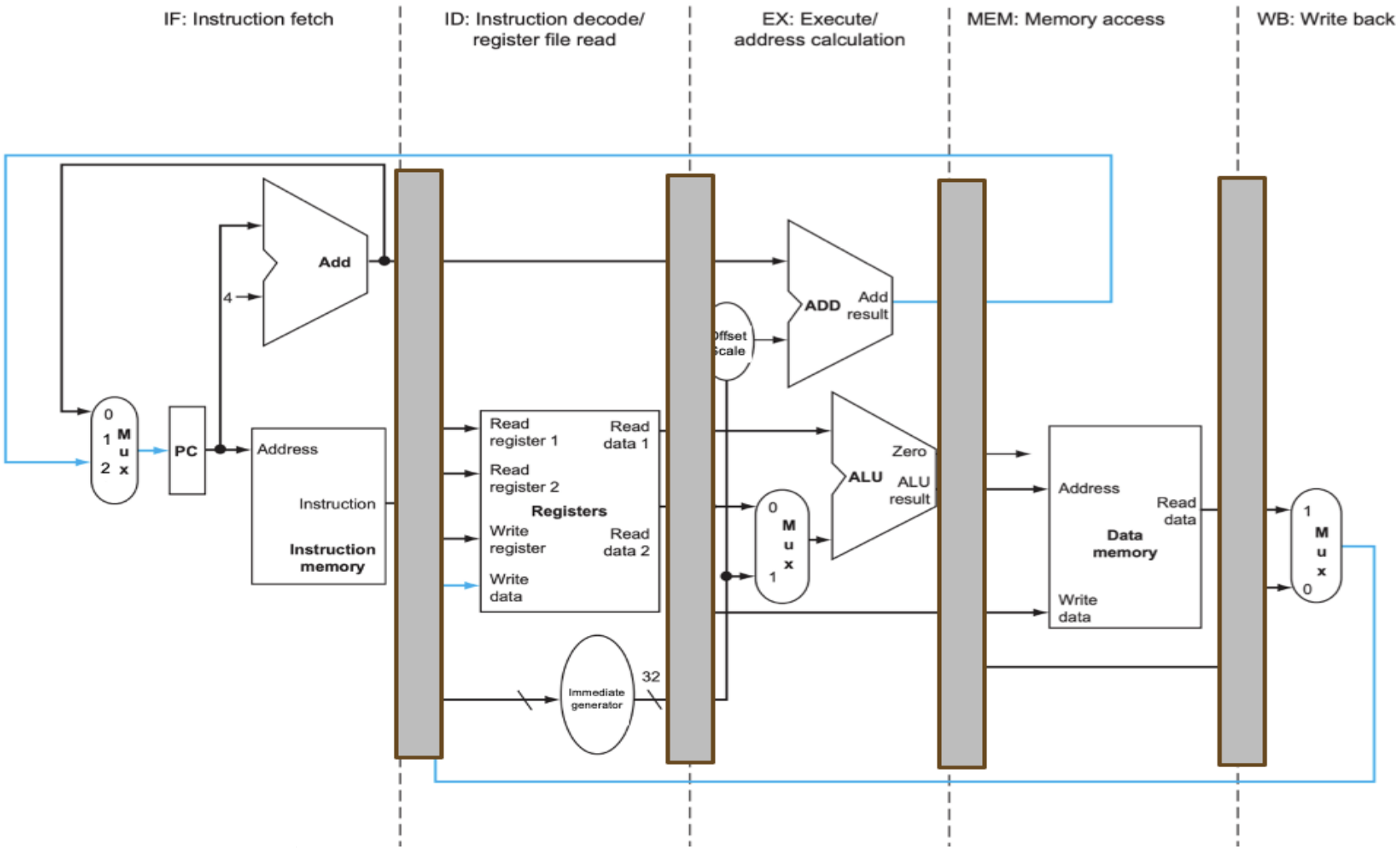
or x21, x11, x12 sw x10, 0(x20) sub x8, x9, x20 lw x9, 100(x0) add x8, x6, x7

Cycle 5



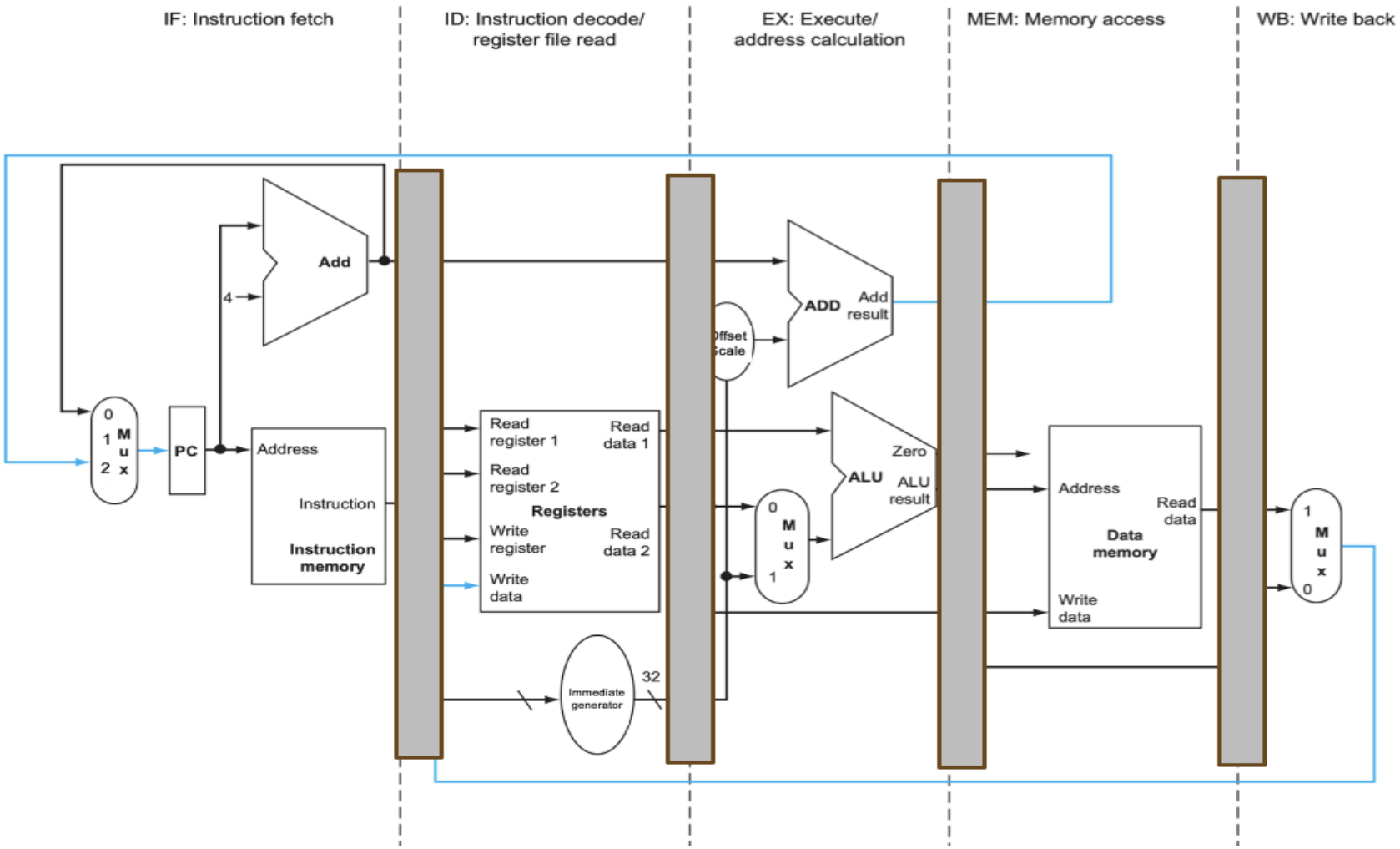
or x21, x11, x12 sw x10, 0(x20) sub x8, x9, x20 lw x9, 100(x0)

Cycle 6



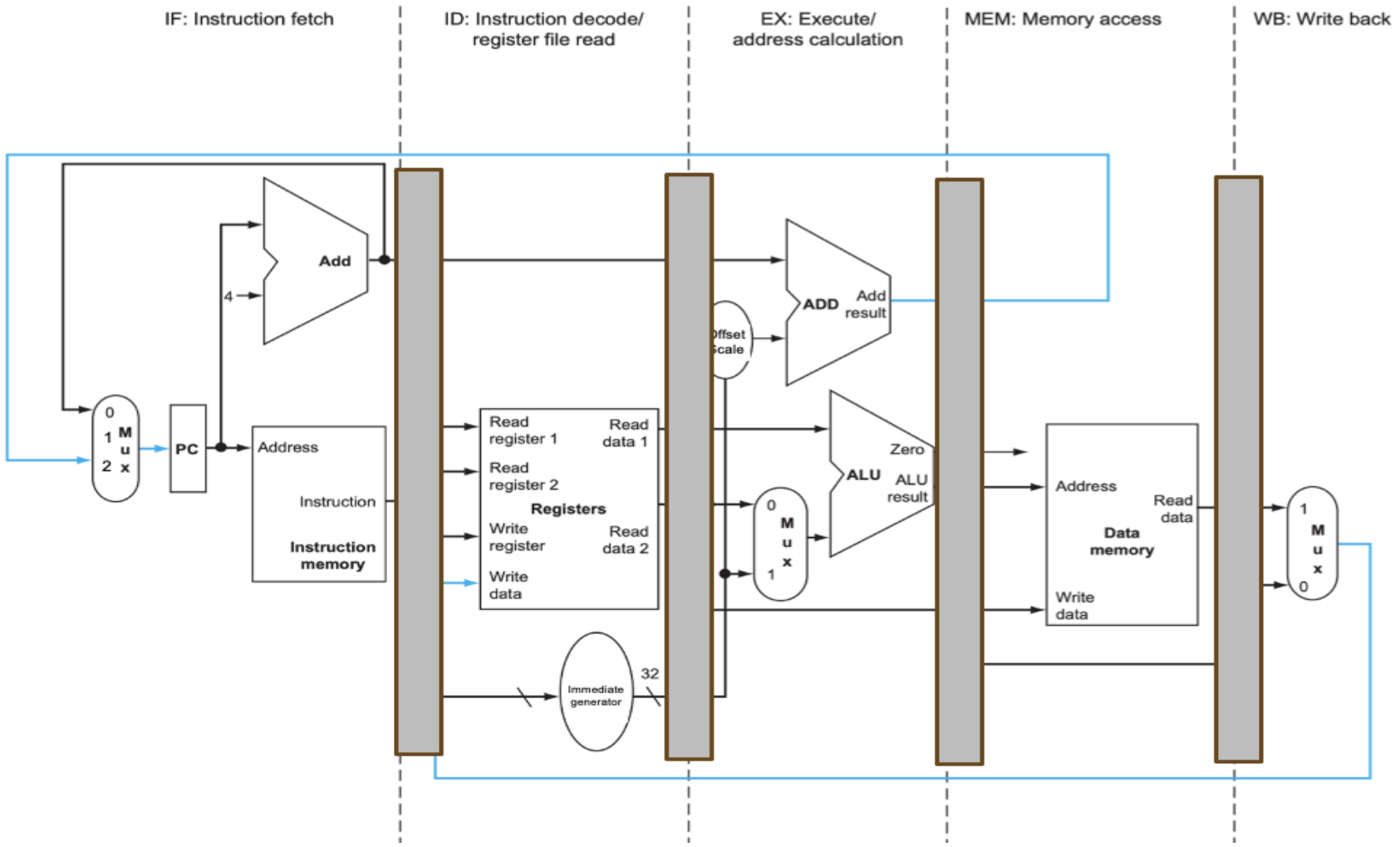
or x21, x11, x12 sw x10, 0(x20) `sub x8, x9, x20`

Cycle 7



or x21, x11, x12 sw x10, 0(x20)

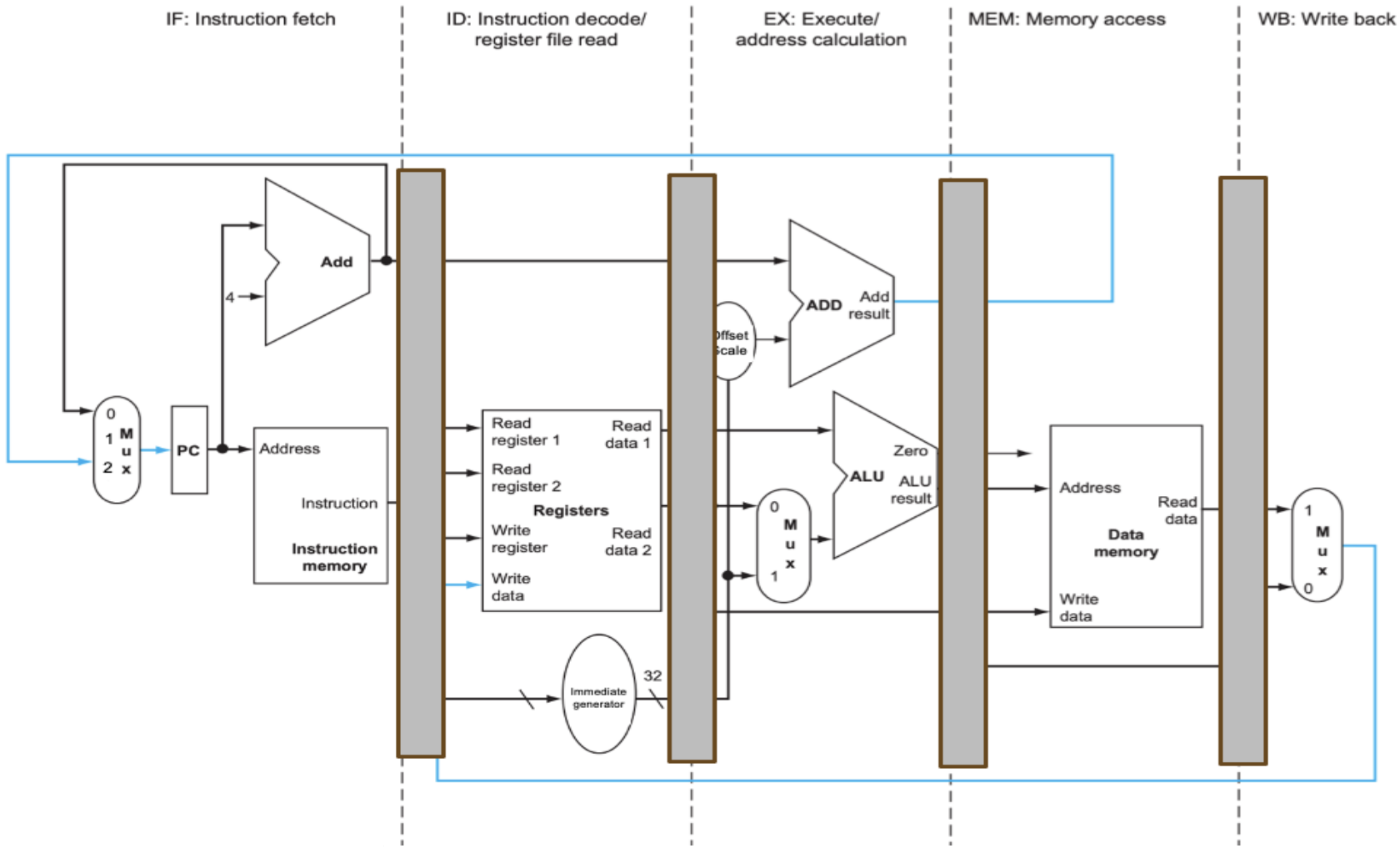
Cycle 8



or x21, x11, x12

Determine the Min Clock Period

The clock-to-q delay of a pipeline register is 20ps



200ps 100ps 200ps 200ps 100ps

Latency

- What is the latency of executing a single instruction on
  - *The single cycle datapath with a clock period of 820ps?*
  - *The 5-stage pipelined datapath with a clock period of 220ps?*

Speedup

- What is the speedup of executing the following set of instructions on a single-cycle vs pipelined datapath assuming
  - *single-cycle clock period = 820 ps*
  - *pipelined clock period = 220 ps*

$$speedup = \frac{single\ cyle\ execution\ time}{pipelined\ execution\ time}$$