CPE 431/531

Chapter 4 – The Processor

Dr. Rhonda Kay Gaede

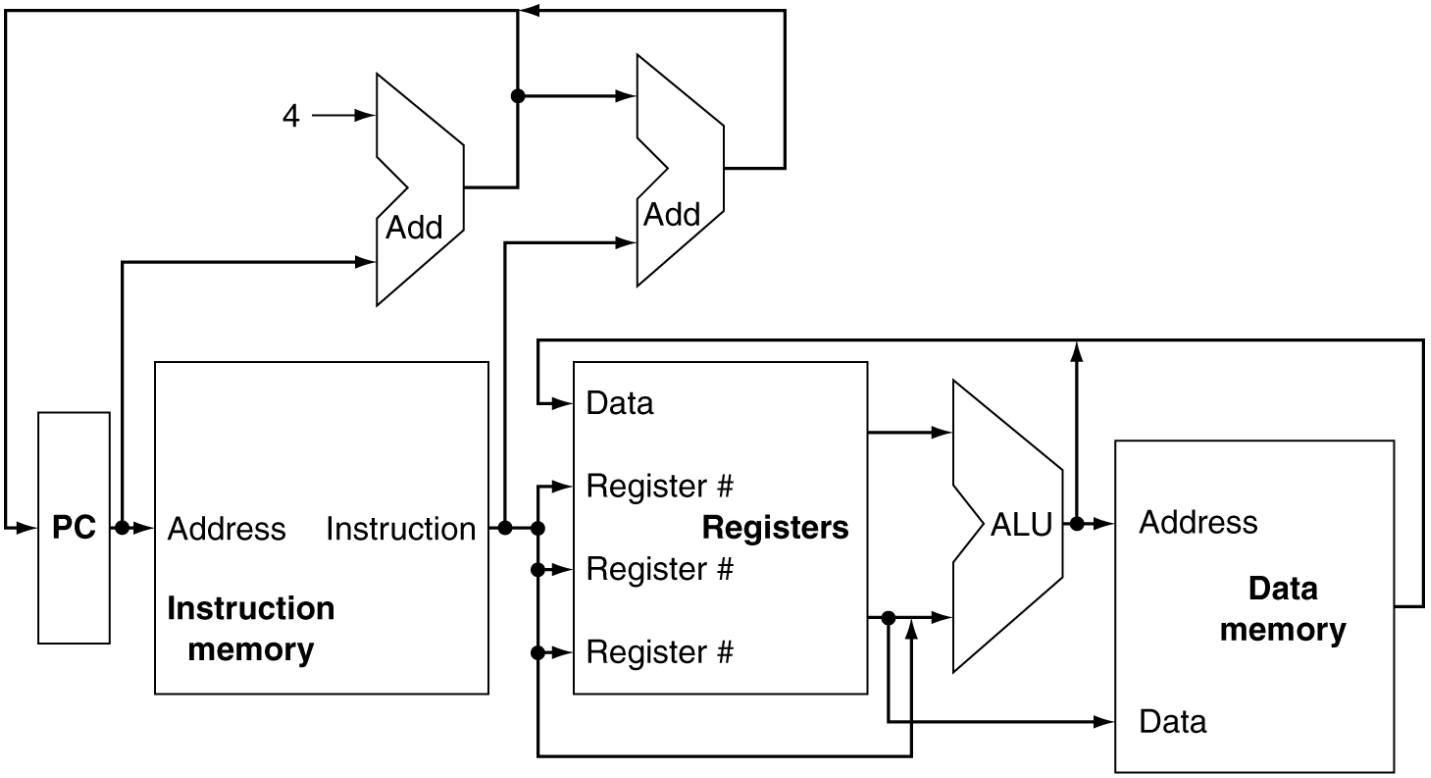
4.1 Implementation Basics

- Performance Factors
  - Instruction Count
  - Cycle Time
  - CPI
- A Basic MIPS Implementation
  - Simple subset: **`lw`**, **`sw`**, **`add`**, **`sub`**, **`and`**, **`or`**, **`slt`**, **`beq`**, **`j`**

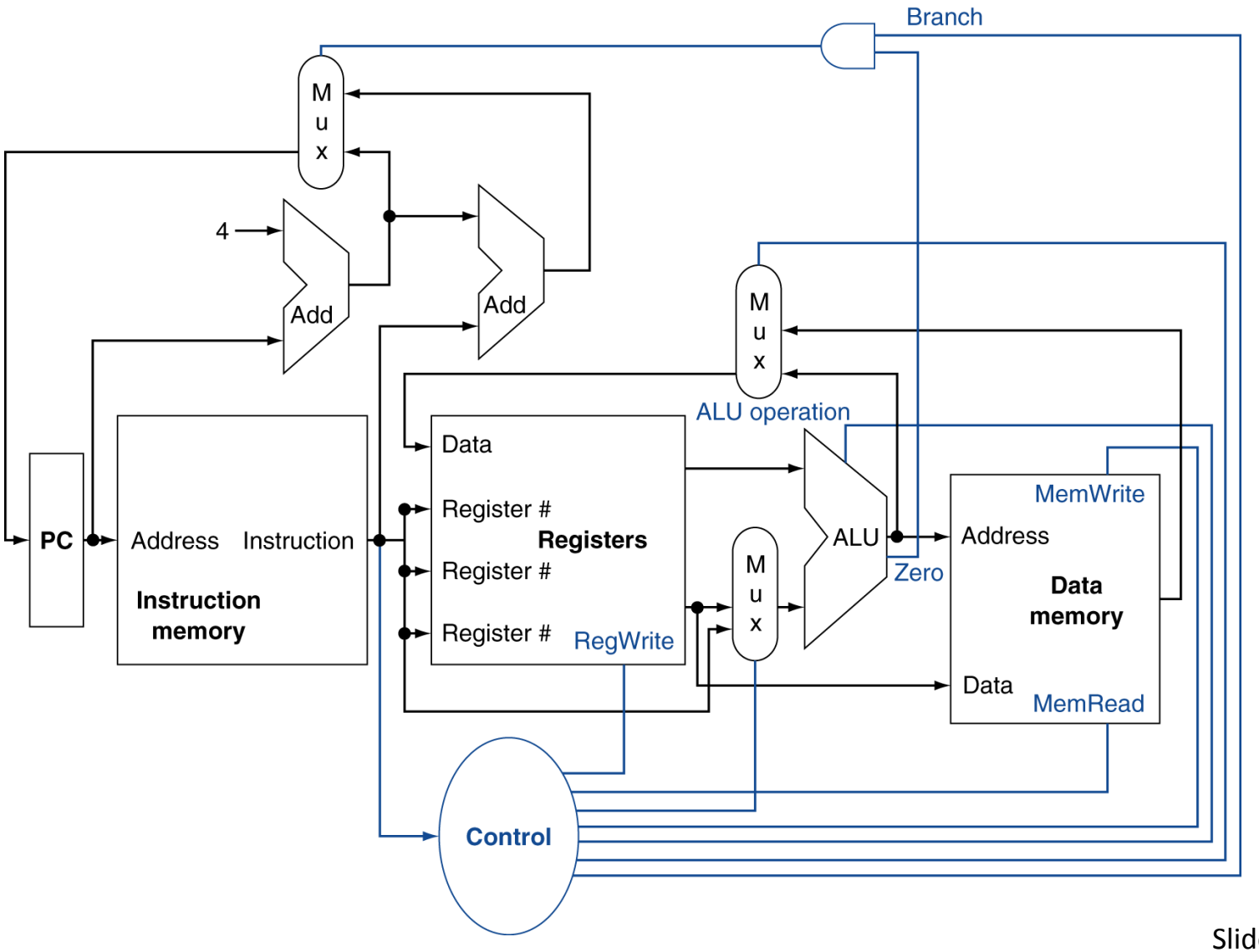
4.1 Implementation Overview

- All instructions begin the same way
  - _____
  - _____
  - _____
- Then, it depends on the instruction
  - **`lw`**
  - **`sw`**
  - **`add`** et.al.
  - **`beq`**

4.1 Datapath


4
Add
Add
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data

4.1 Datapath + Control

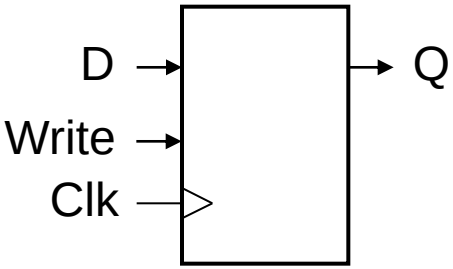


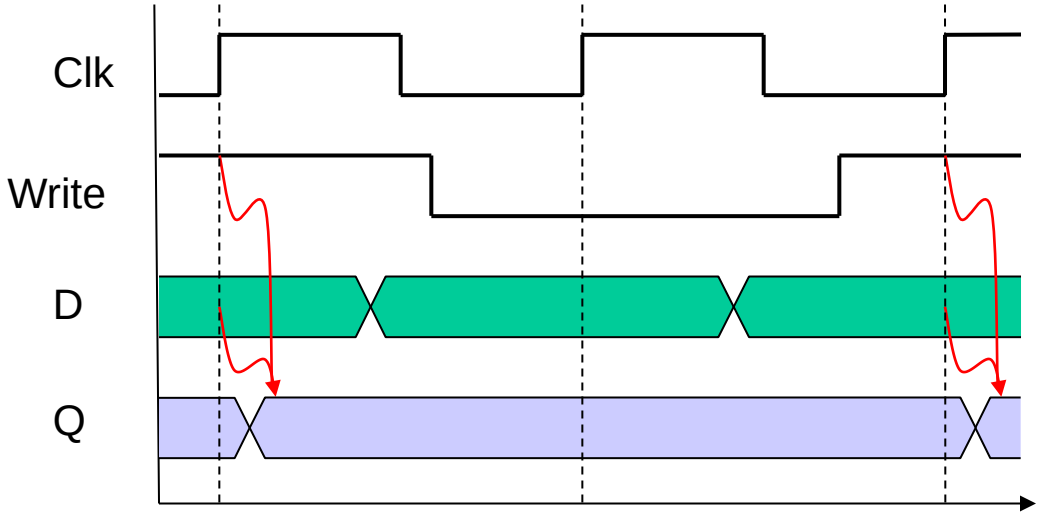
4.2 Classes and Values

- Two classes of logic
  - _____
  - _____|
- Two logic values
  - _____
  - _____-

4.2 Sequential Elements

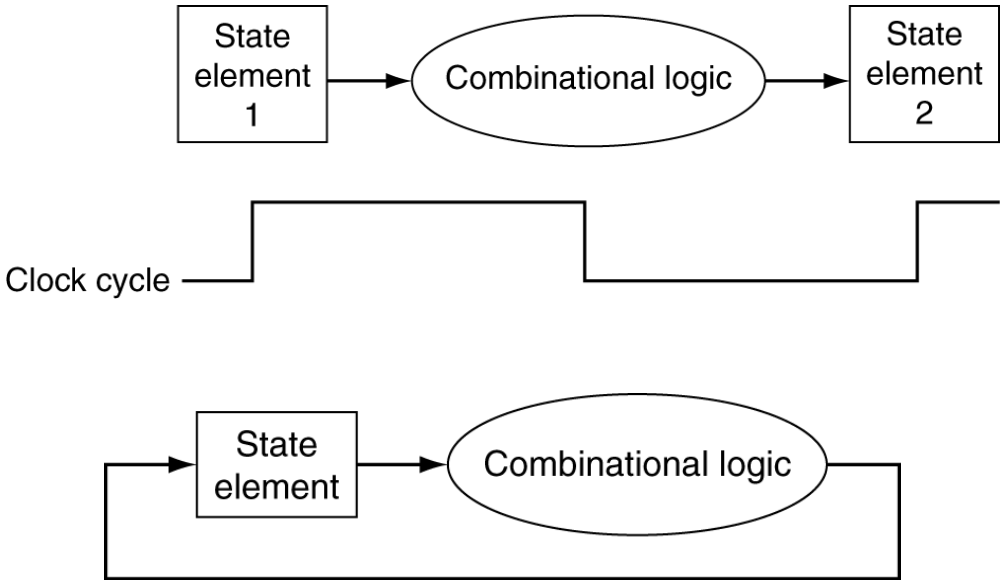
- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later.



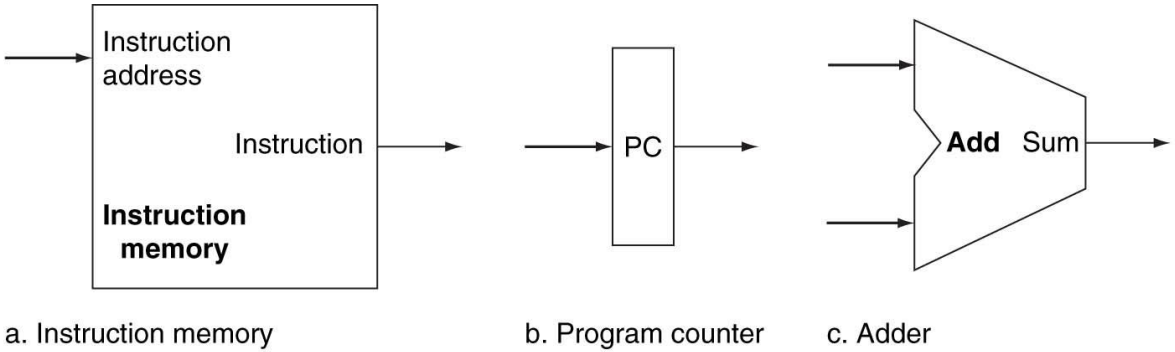


4.2 Clocking Methodology

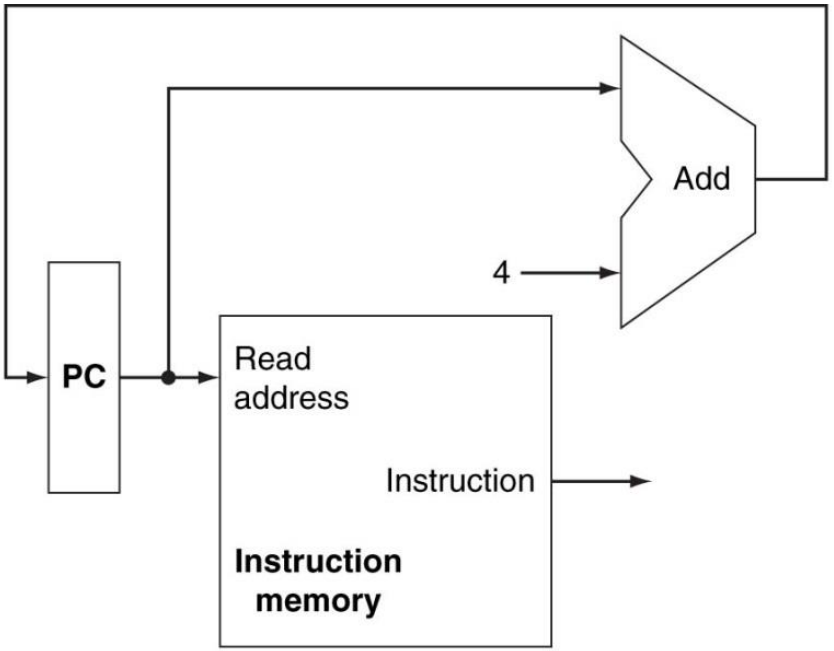
- A clocking methodology defines when signals can be _____ and when they can be _____.
- We assume an _____ clocking methodology.
- Longest delay determines clock period



4.3 Instruction Fetch and Default Sequencing

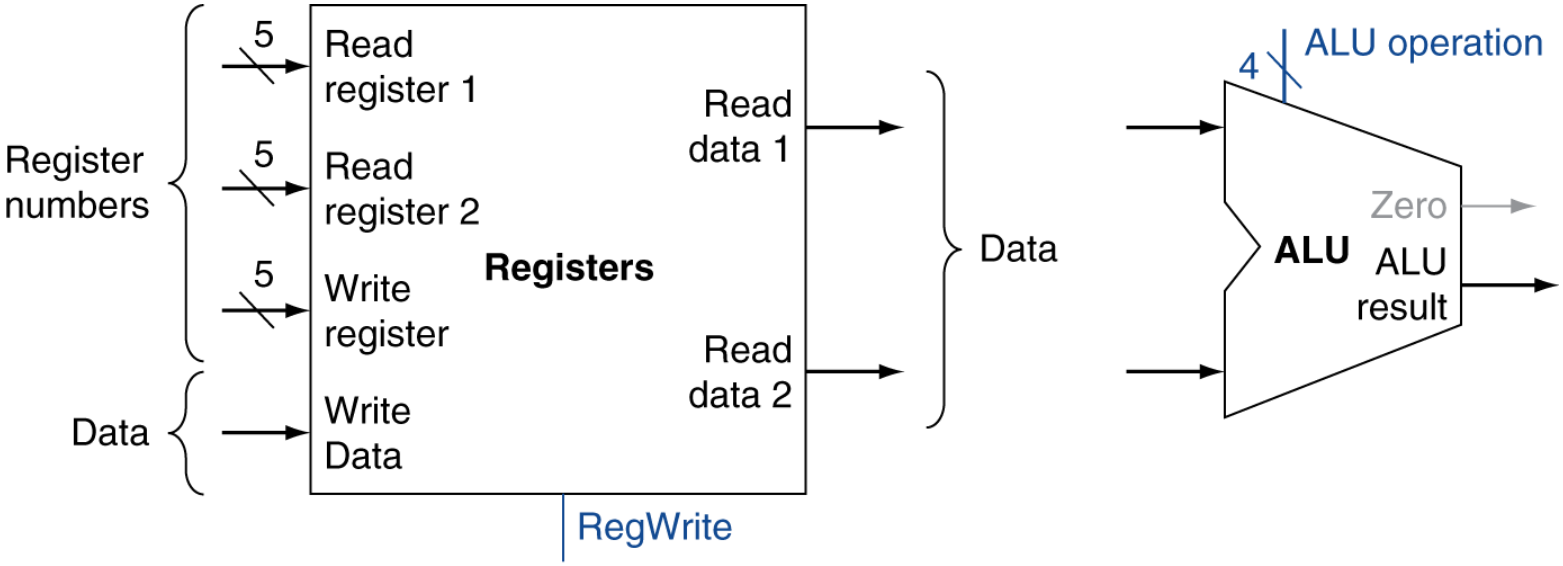


a. Instruction memory b. Program counter c. Adder



4.3 R-type Instruction Requirements

- Read ____ register operands
- Perform _____ operation
- _____ register results

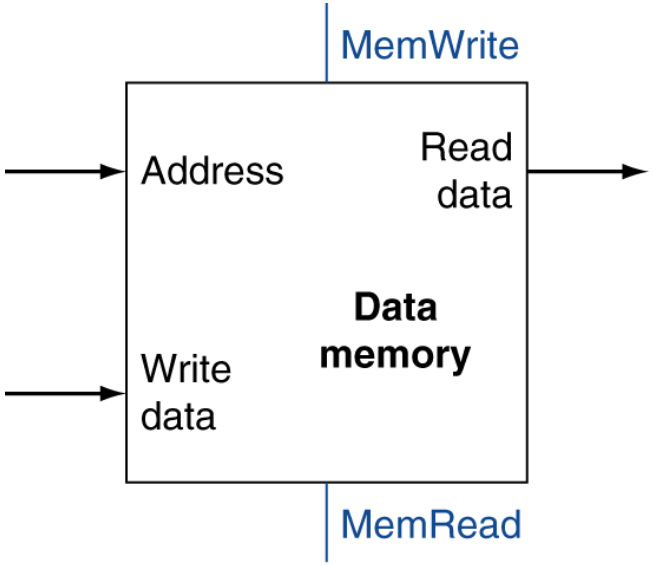


a. Registers

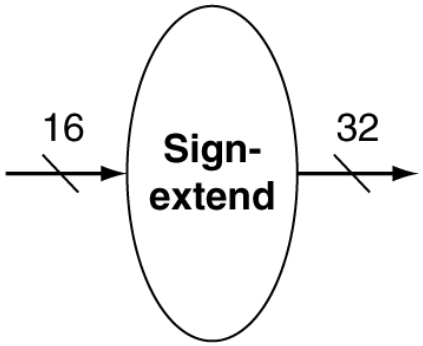
b. ALU

4.3 `lw/sw` Instruction Requirements

- Read register operands
- Calculate _____ _____ using 16-bit offset
- Load: _____ memory and _____ register
- Store: _____ register value to _____



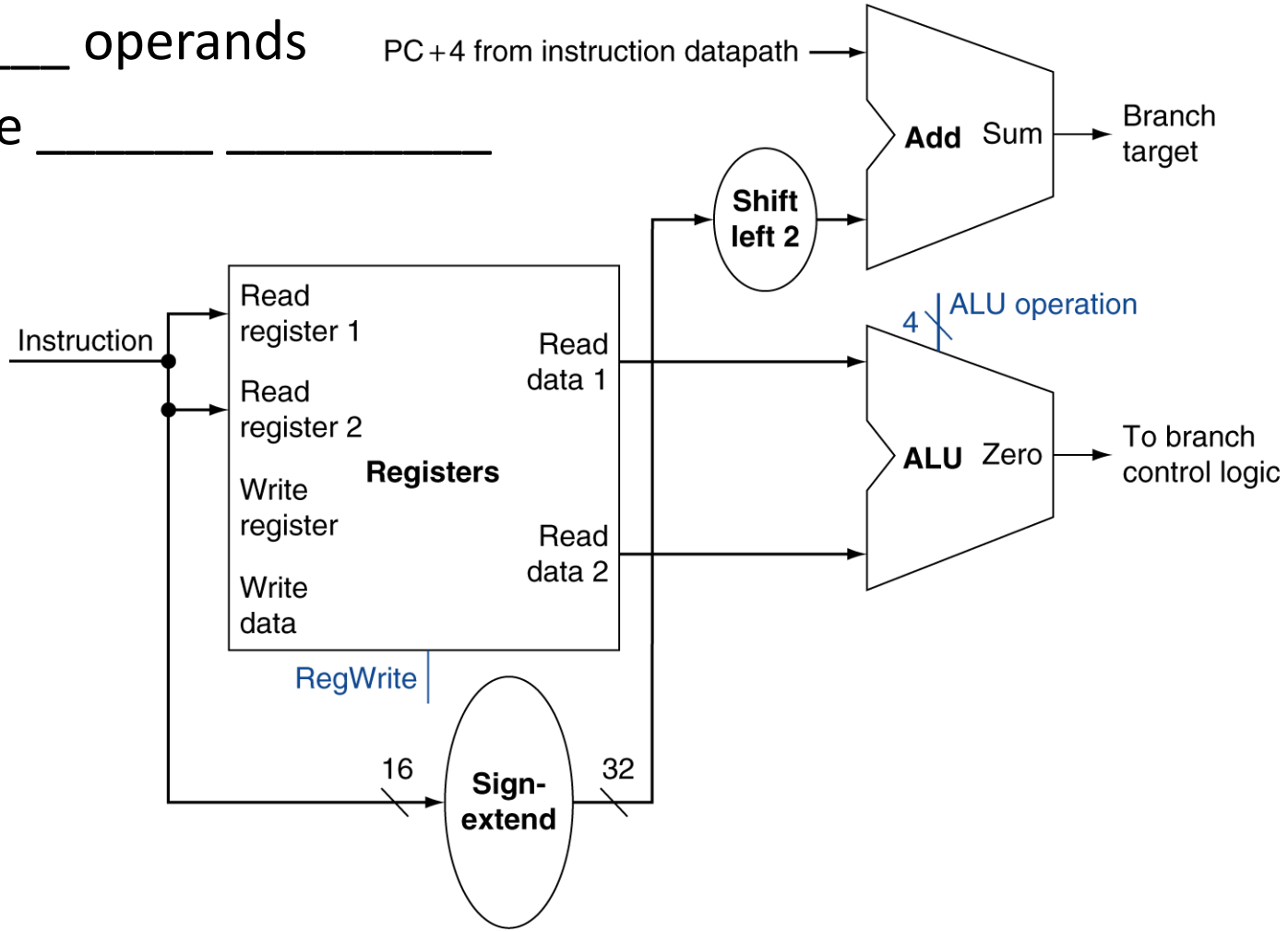
a. Data memory unit



b. Sign extension unit

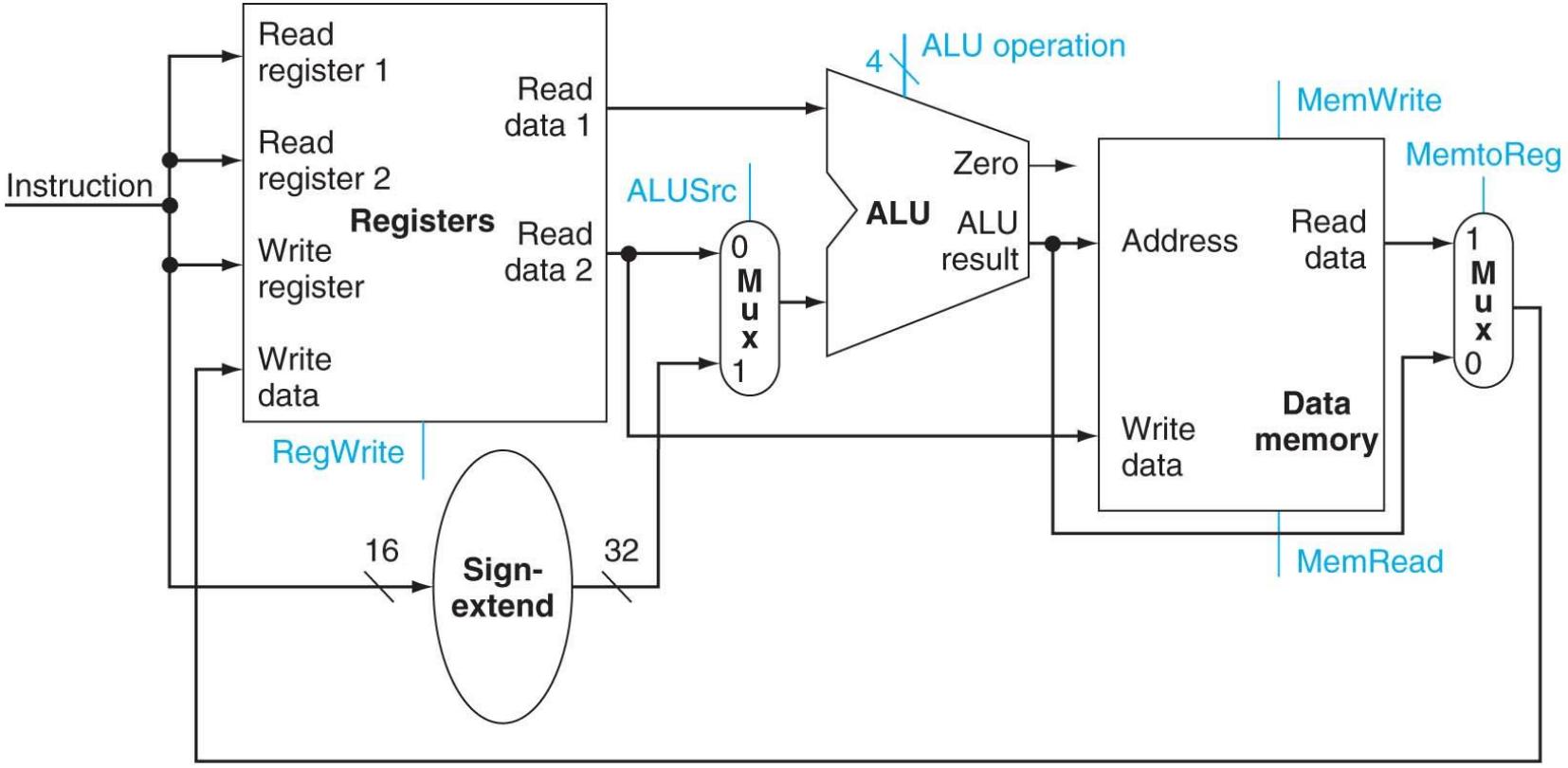
4.3 `beq` Instruction Requirements

- Read register operands
- _____ operands
- Calculate _____ _____

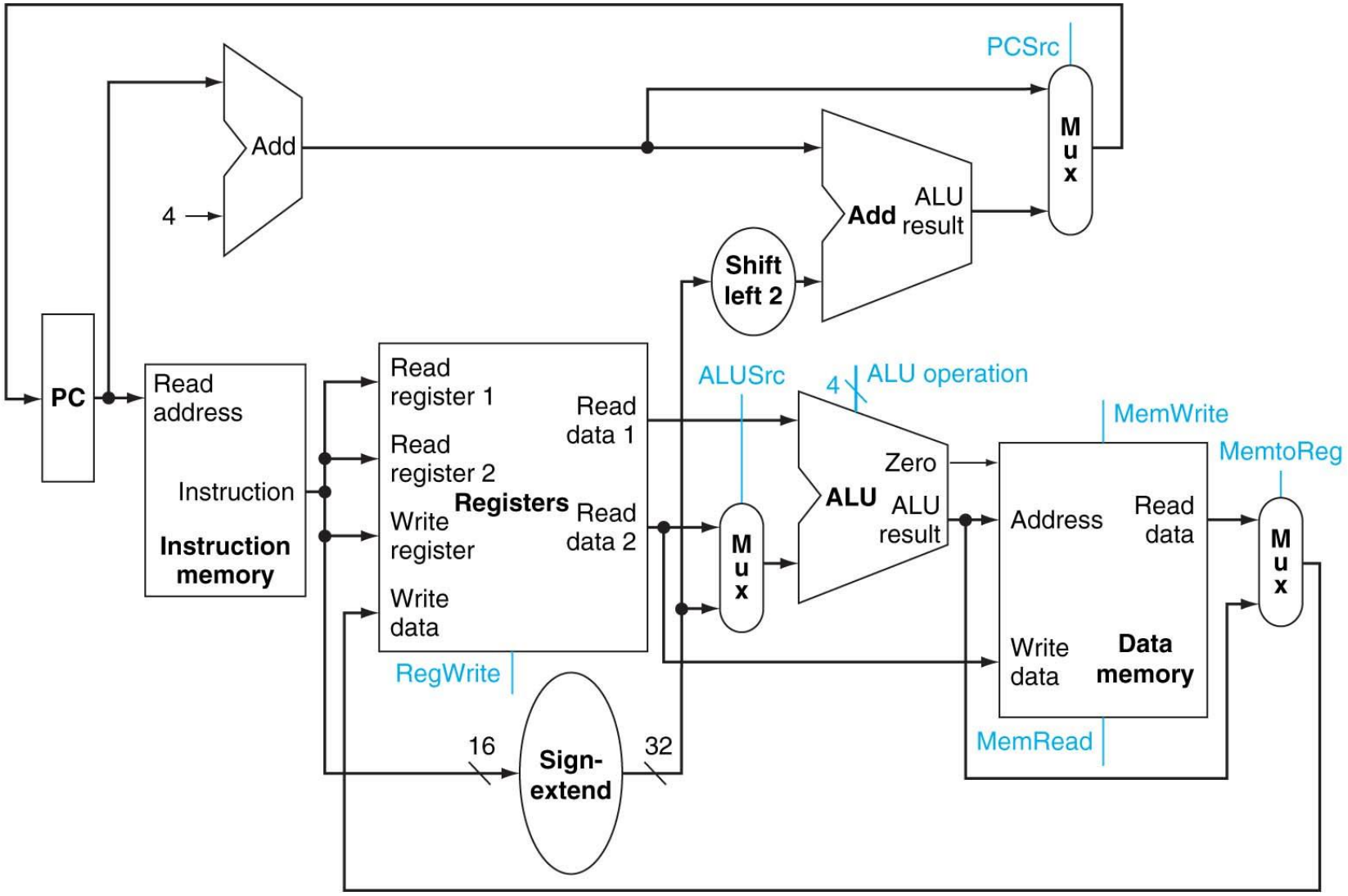


4.3 Creating a Single Datapath: R-type + `lw/sw`

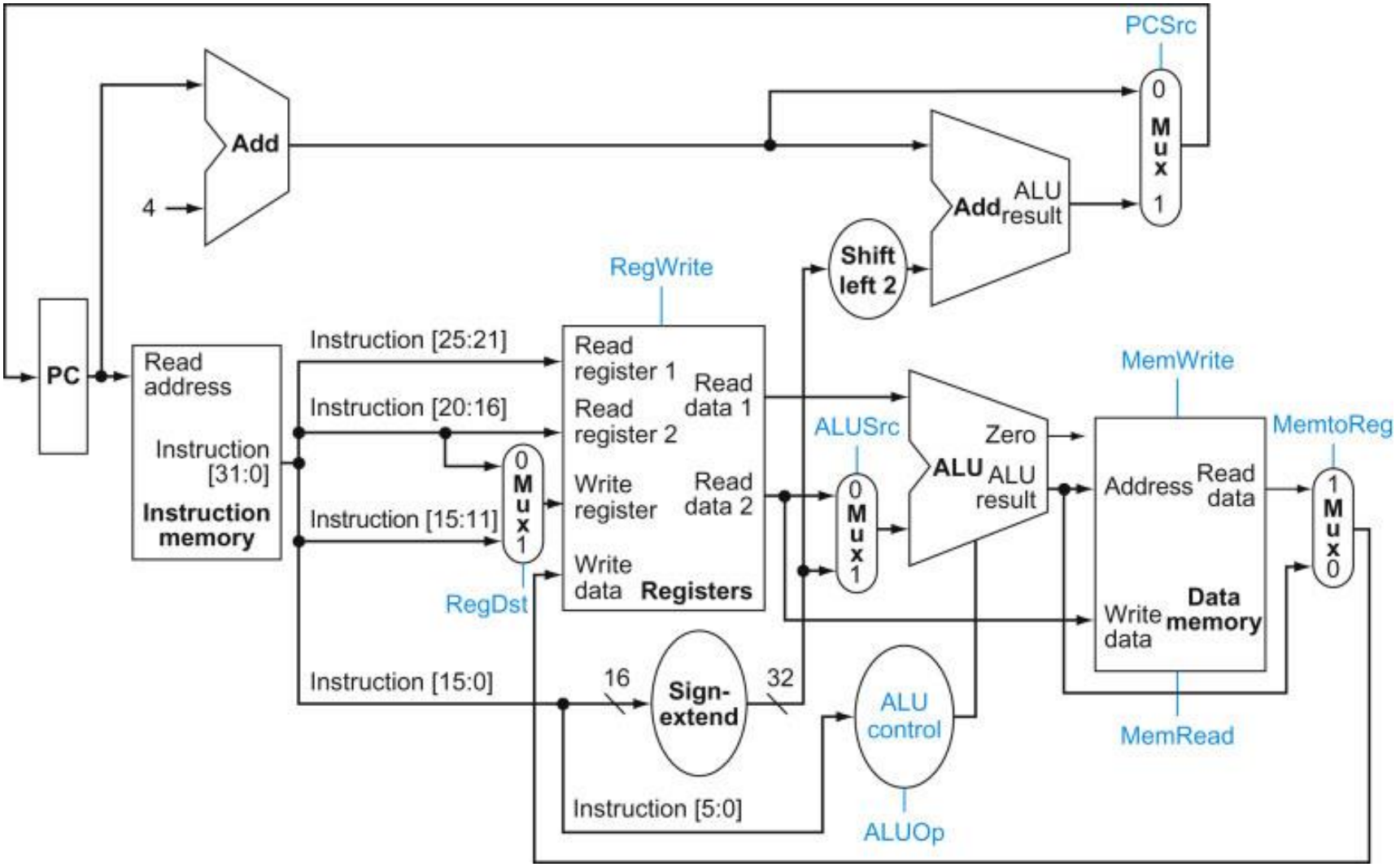
- We want a _____ implementation.
- Separate _____ and ____ memories are required.
- When I say _____, you say ____.



4.3 Adding `beq` to the R-type + `lw/sw` Datapath

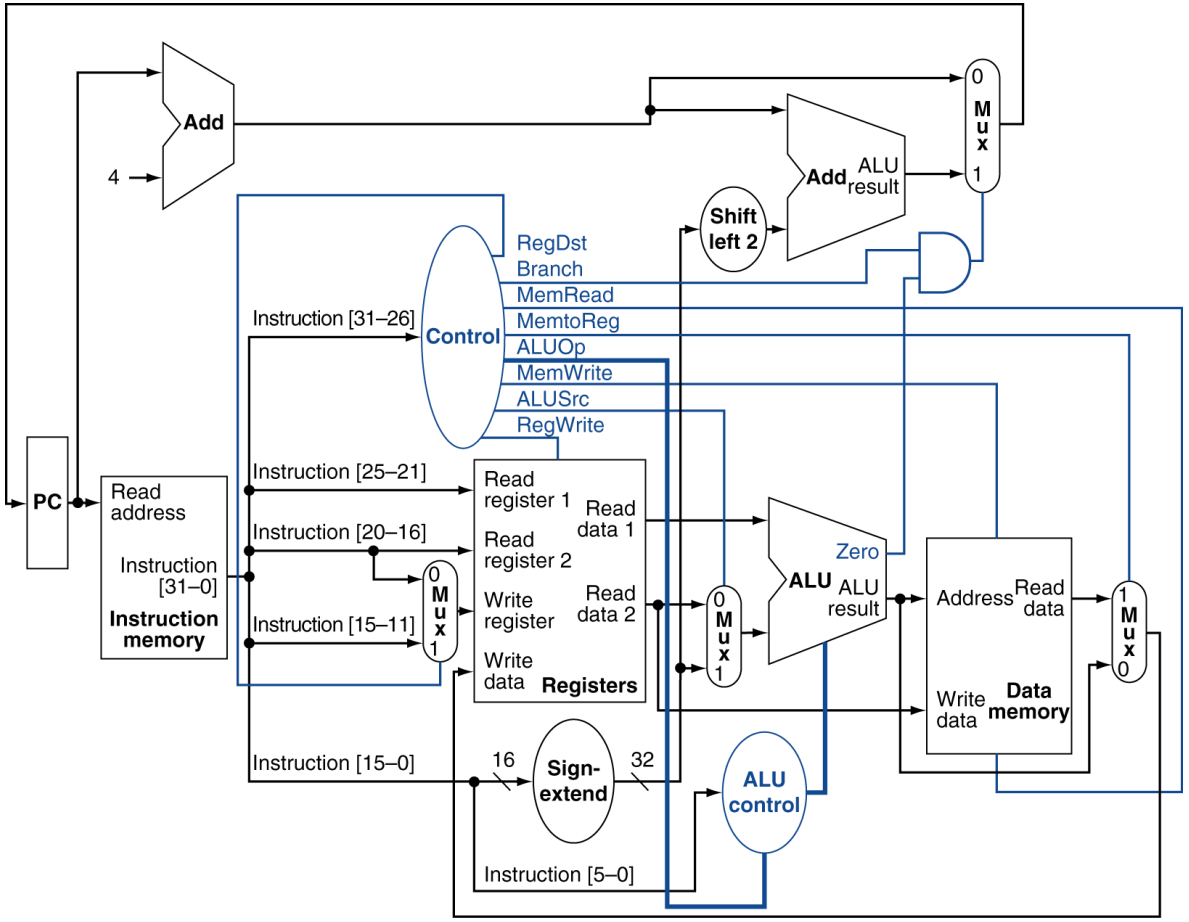


4.4 Defining Necessary Control

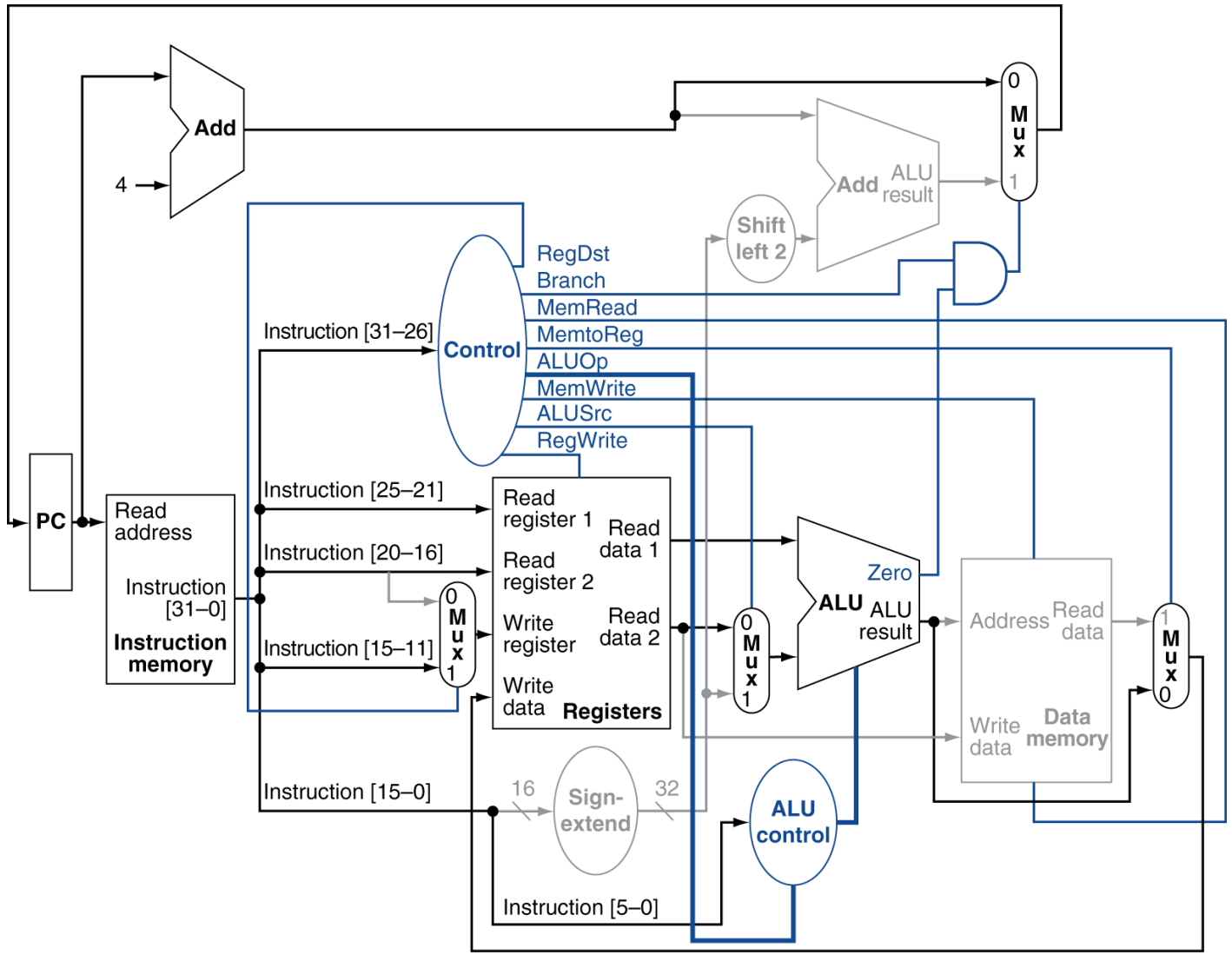


4.4 Adding a Control Unit

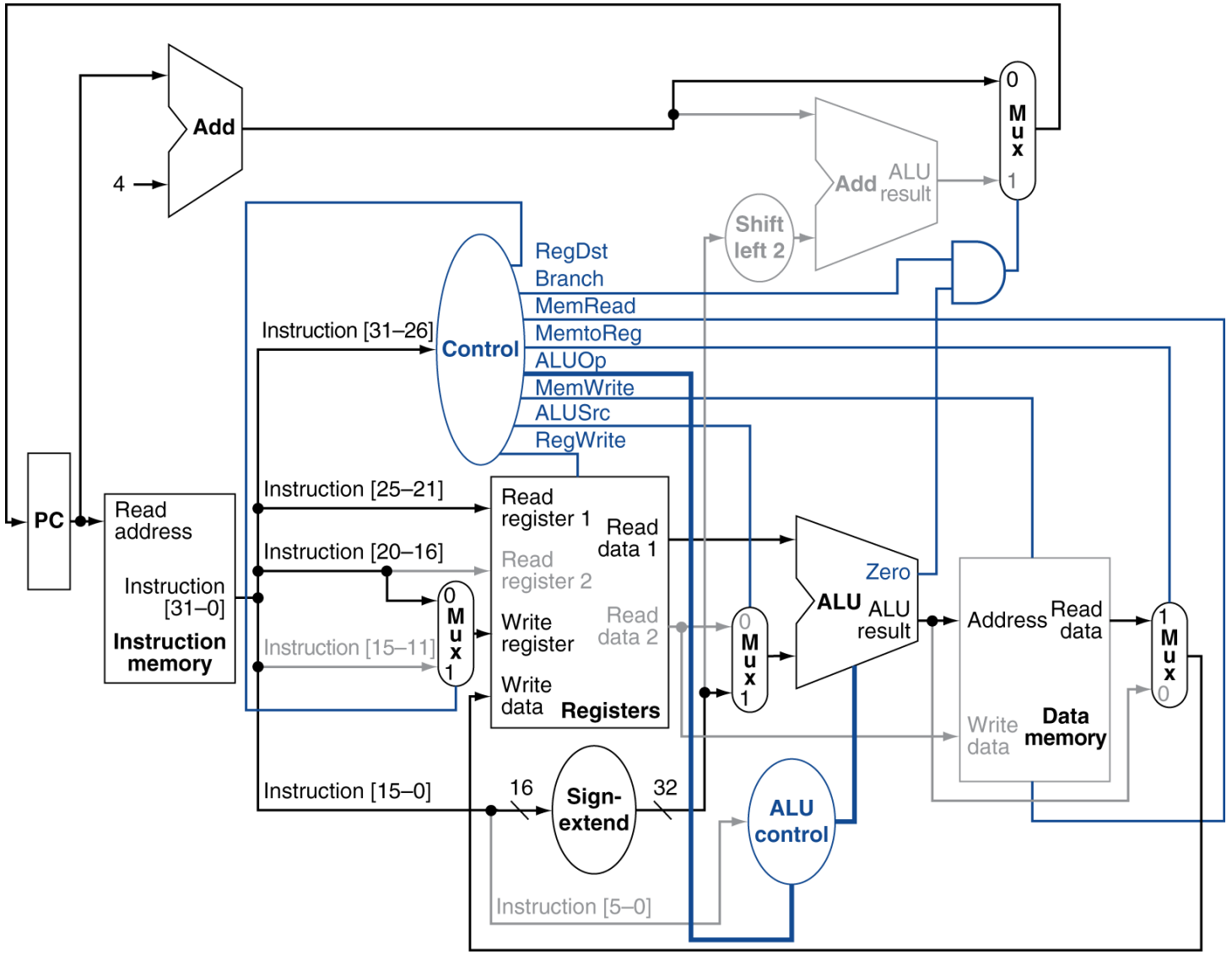
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1



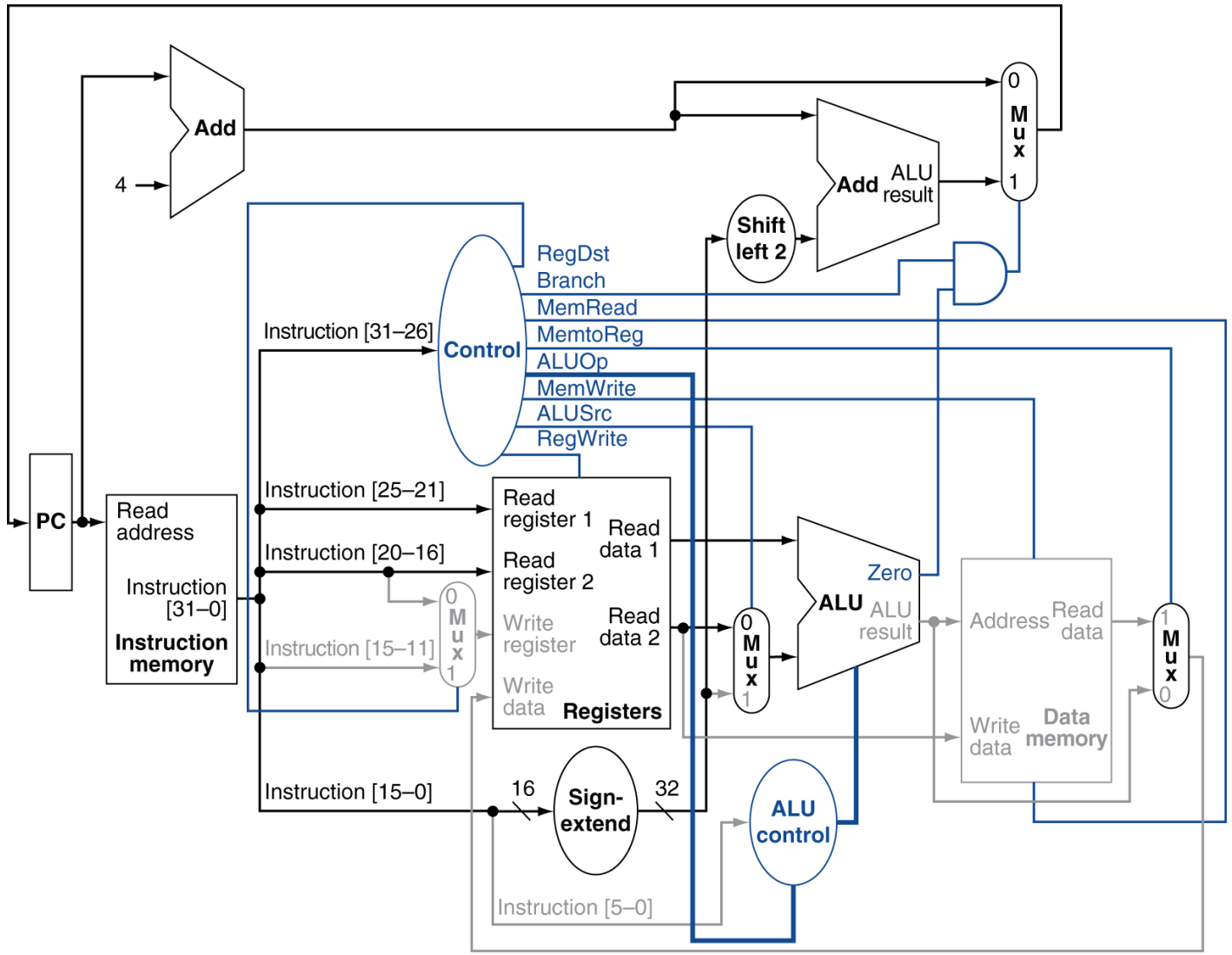
4.4 R-type Instruction Execution



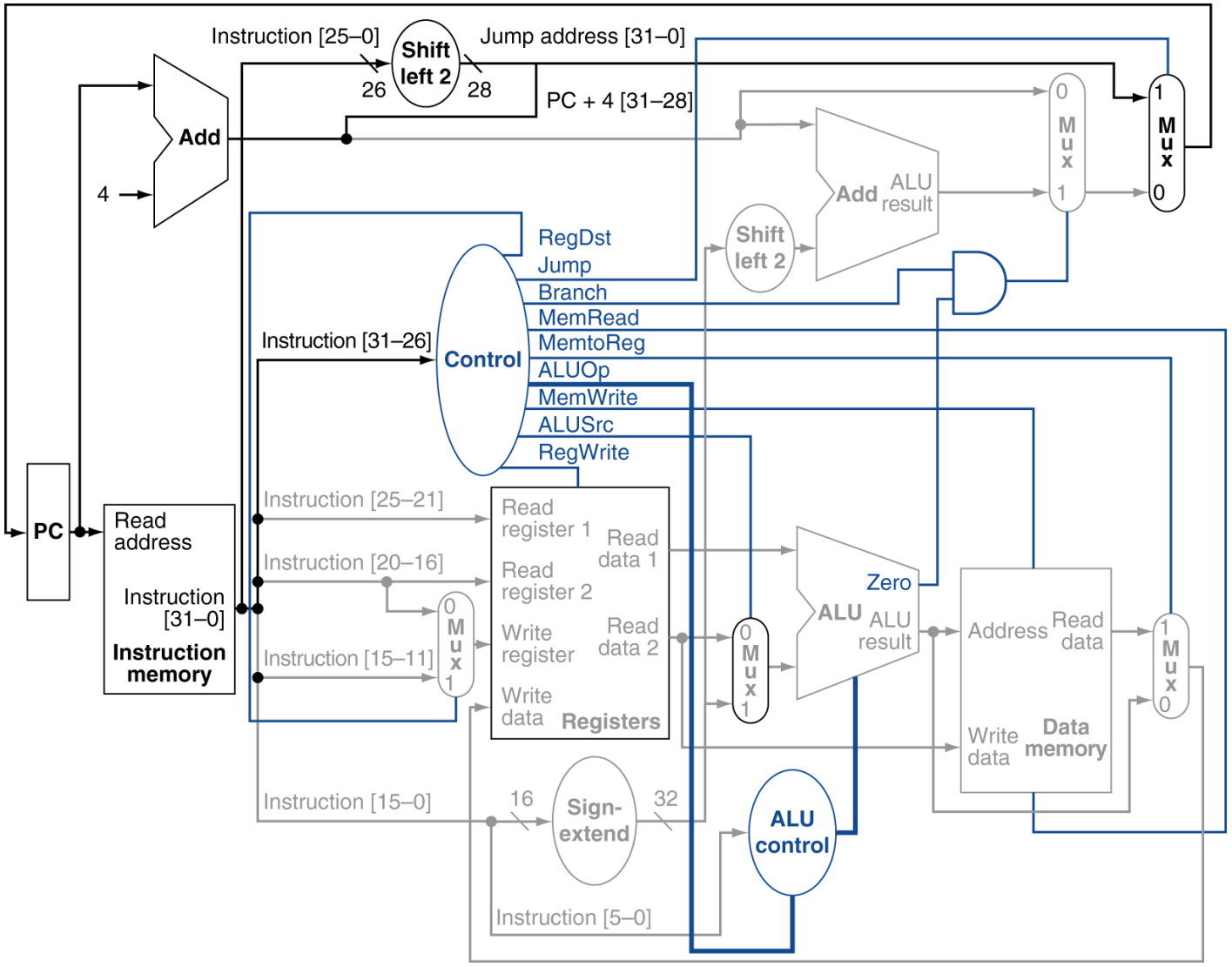
4.4 `lw` Instruction Execution



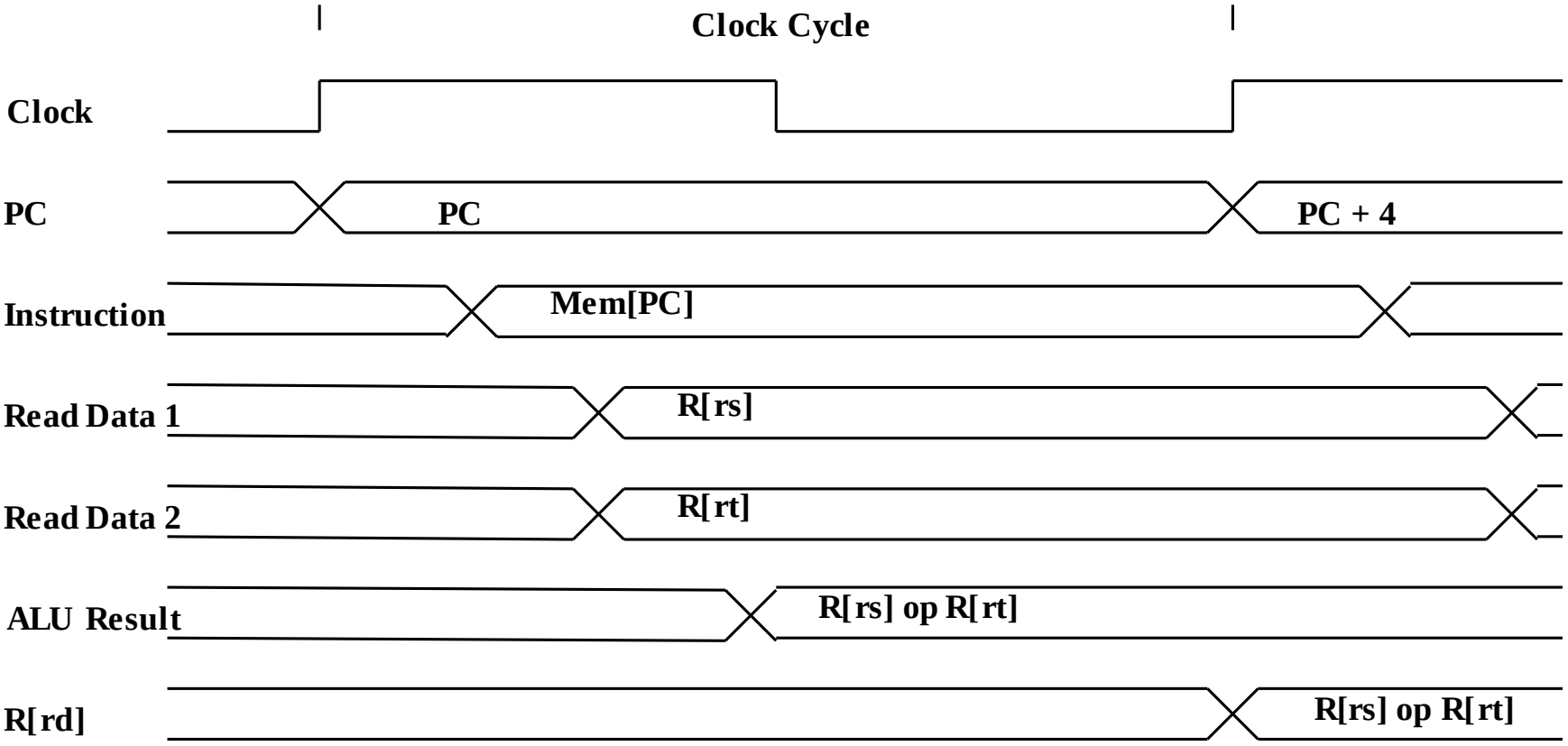
4.4 beq Instruction Execution



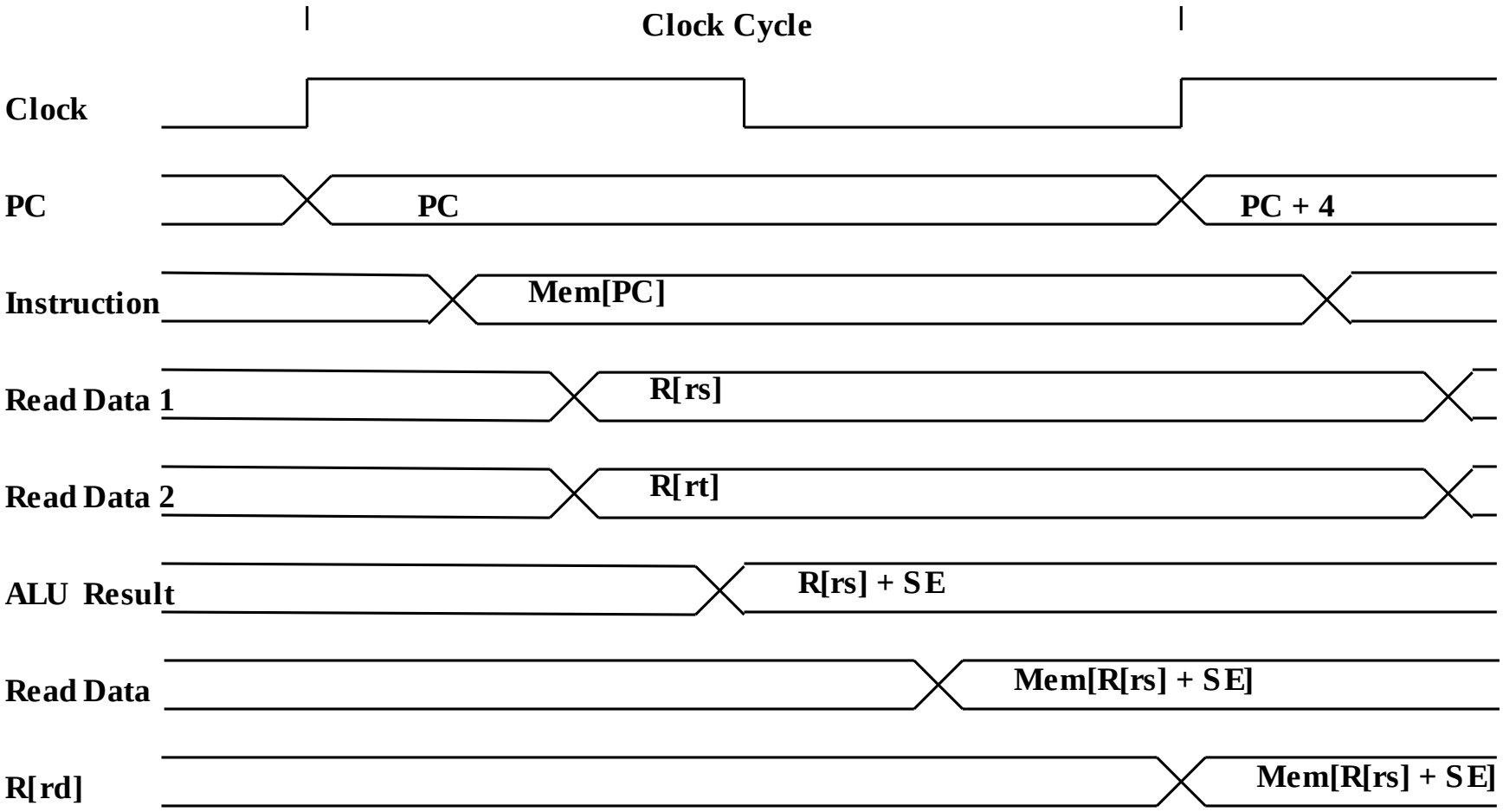
4.4 Implementing Jumps



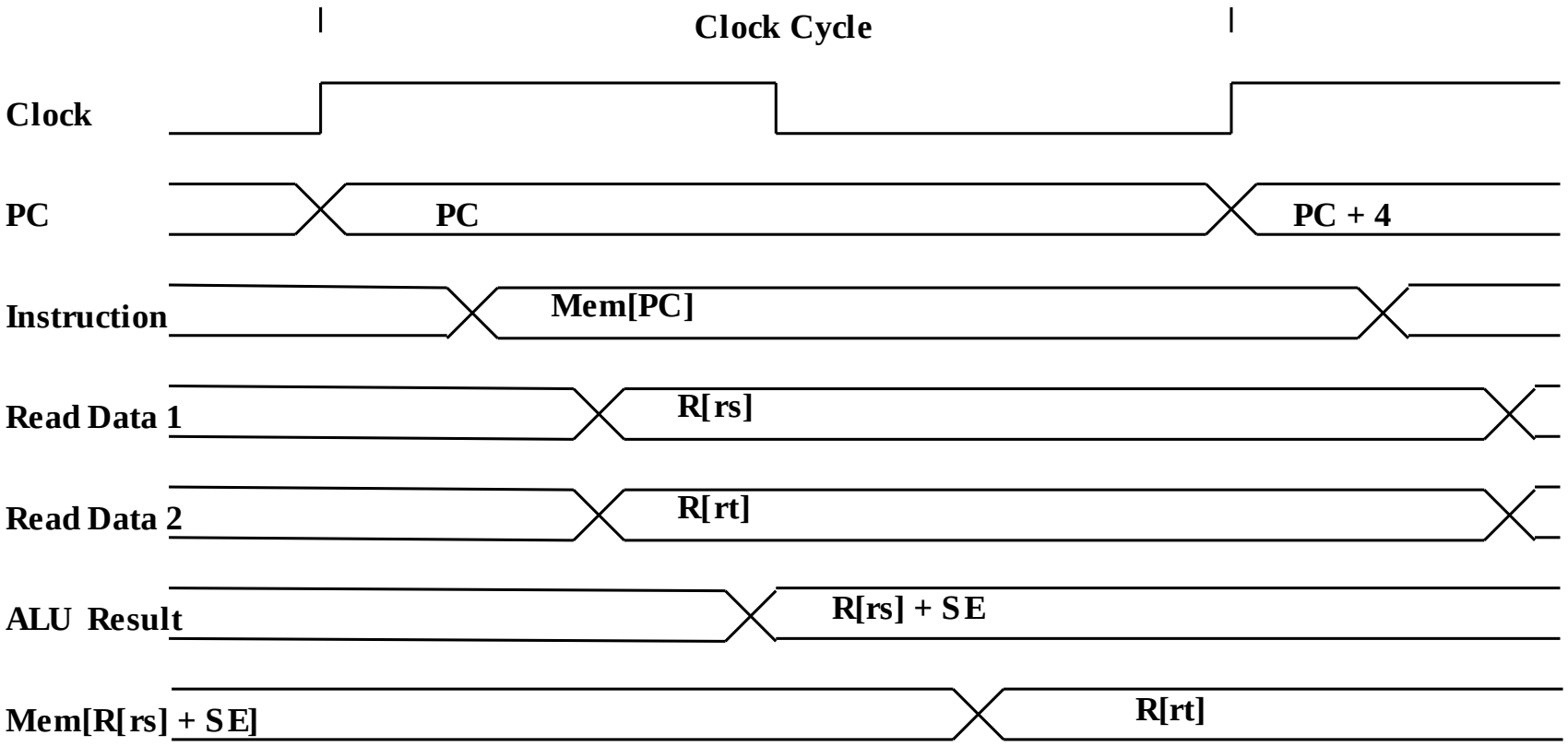
4.4 R-type Instruction Timing



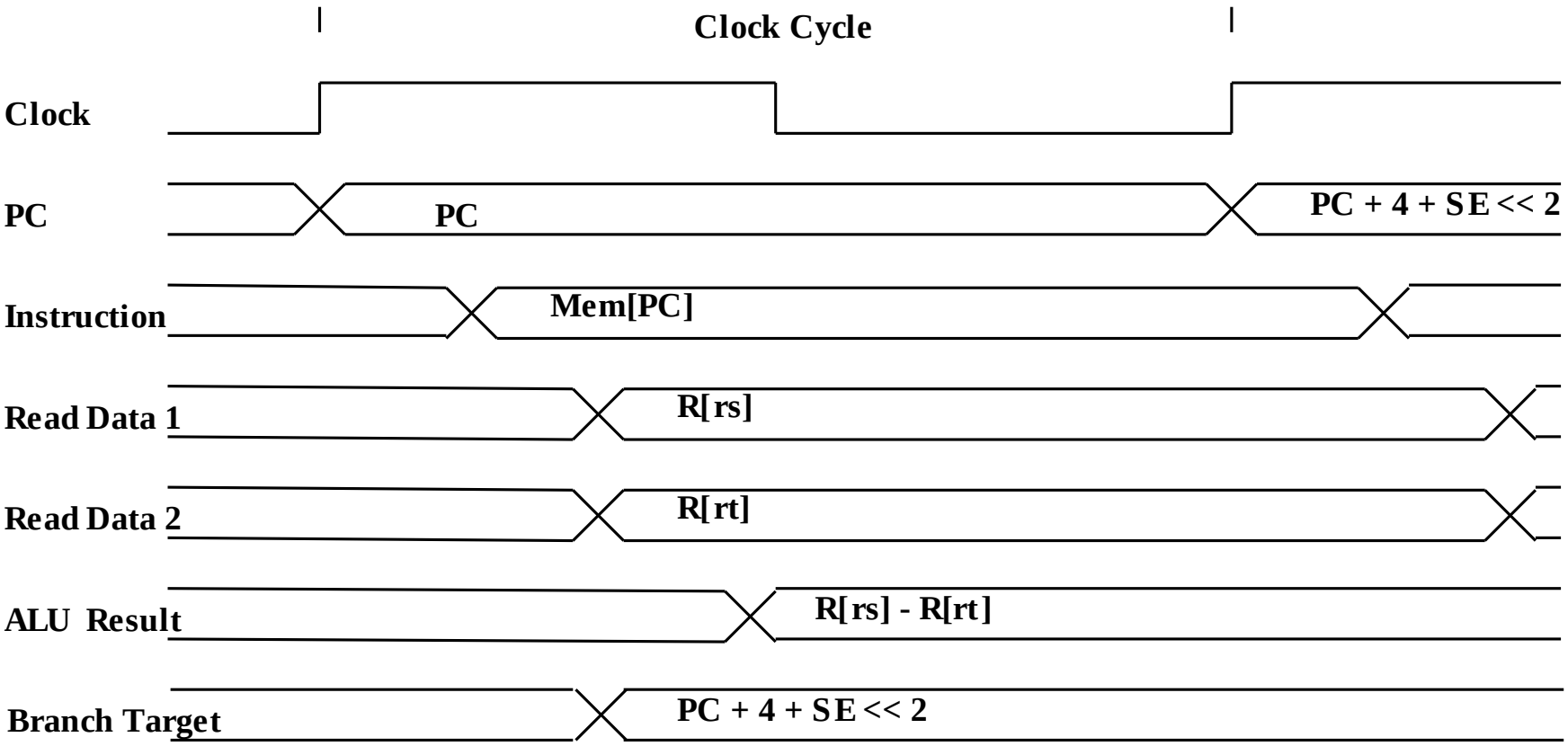
4.4 `lw` Instruction Timing



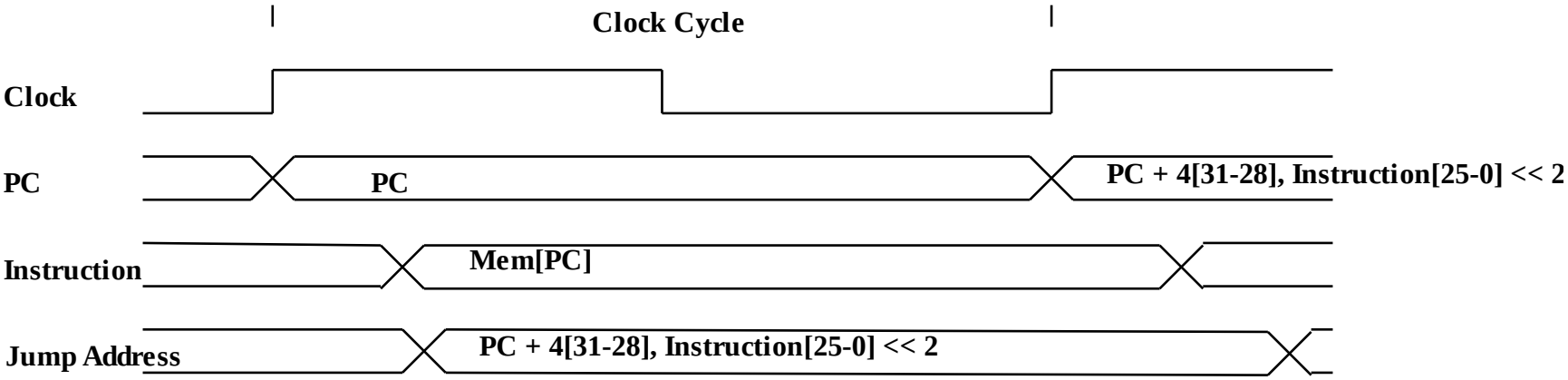
4.4 `sw` Instruction Timing



4.4 `beq` Instruction Timing



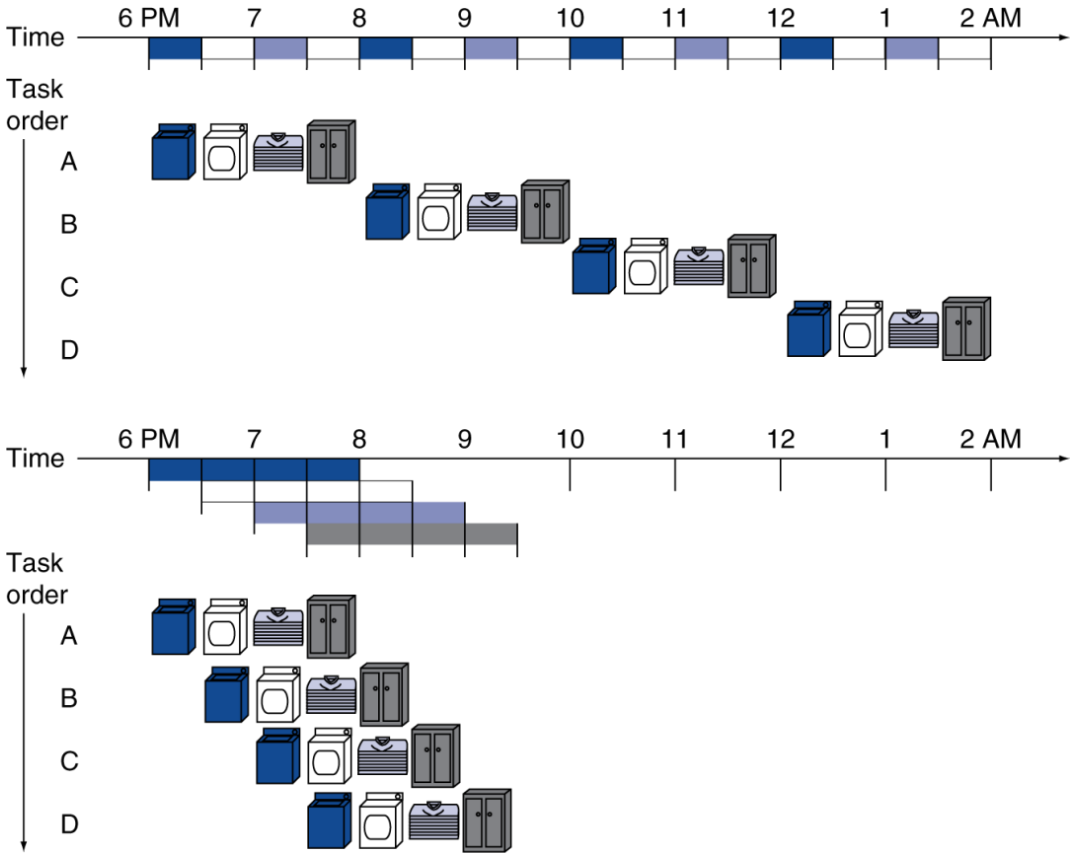
4.4 j Instruction Timing



4.5 An Overview of Pipelining

- Pipelining is an implementation technique in which _____ _____are _____in execution.
- Pipelining helps _____, not individual _____ _____.
- You can't ____ a stage.

4.5 The Laundry Analogy



Speedup

- One Load
- Four Loads
- N Loads as N $\rightarrow \infty$

4.5 MIPS Processor Stages

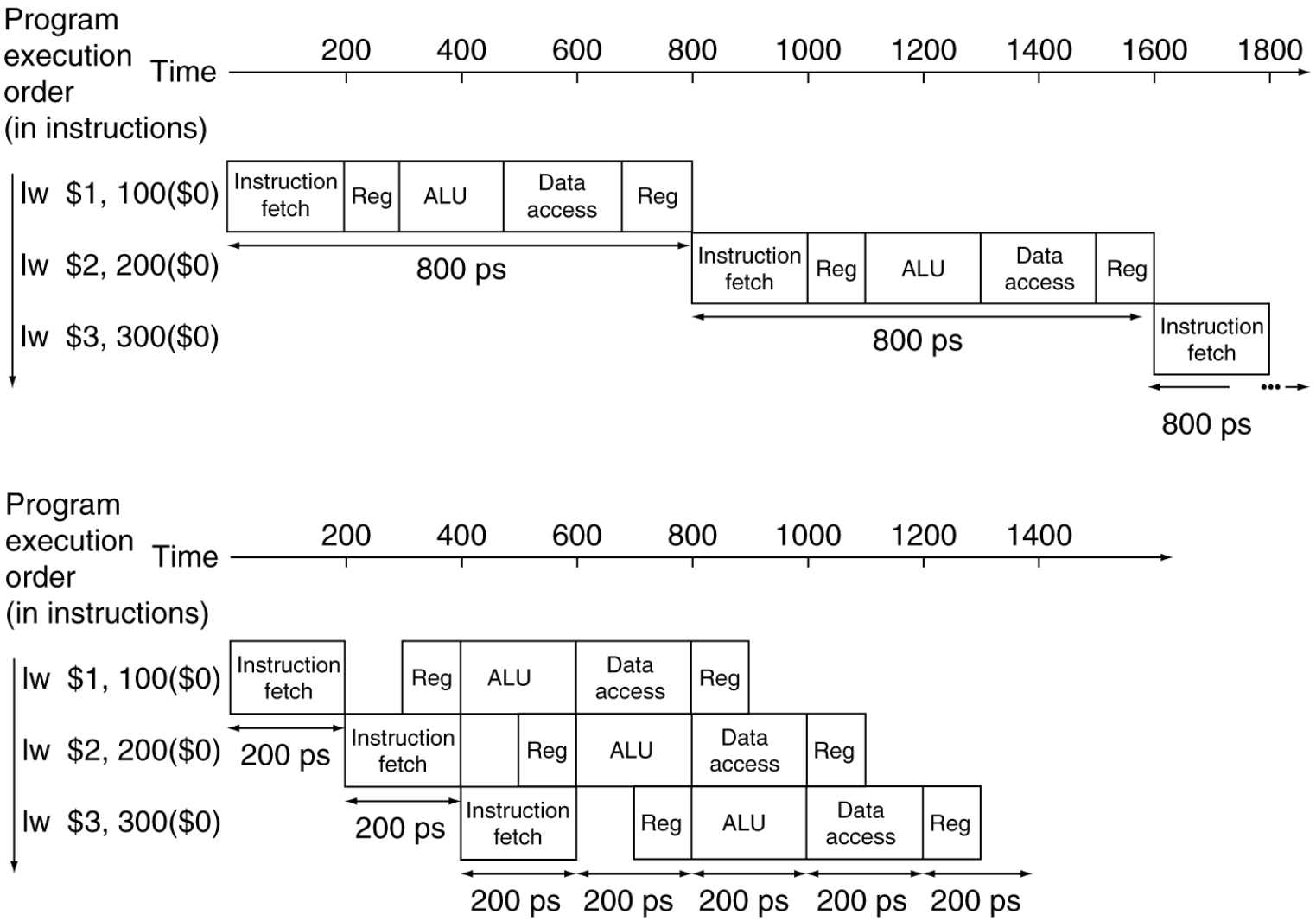
- Instruction fetch (IF)
- Instruction decode and register read (ID)
- Execution (calculate address) (EX)
- Memory access (MEM)
- Register write (WB)

4.5 Single Cycle vs. Pipelined Performance

- Consider: **`lw, sw, add, sub, and, or, slt, beq`**
- Operation times: memory, ALU 200 ps, register 100 ps

Instr	Instruction 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

4.5 Single Cycle vs. Pipelined Timeline



4.5 Designing Instruction Sets for Pipelining

- All MIPS instructions are the same _____.
- MIPS has only a few _____ _____.
- Memory operands appear only in _____.
- Operands must be _____ in memory.

4.5 Pipeline Hazards

- Structural Hazard - not enough _____.
- Data Hazards – one _____ needs the _____ of another
- Control Hazard – _____ aren't made
  - Conservative Approach – _____
  - Alternative Approach – _____

4.5 Data Hazards

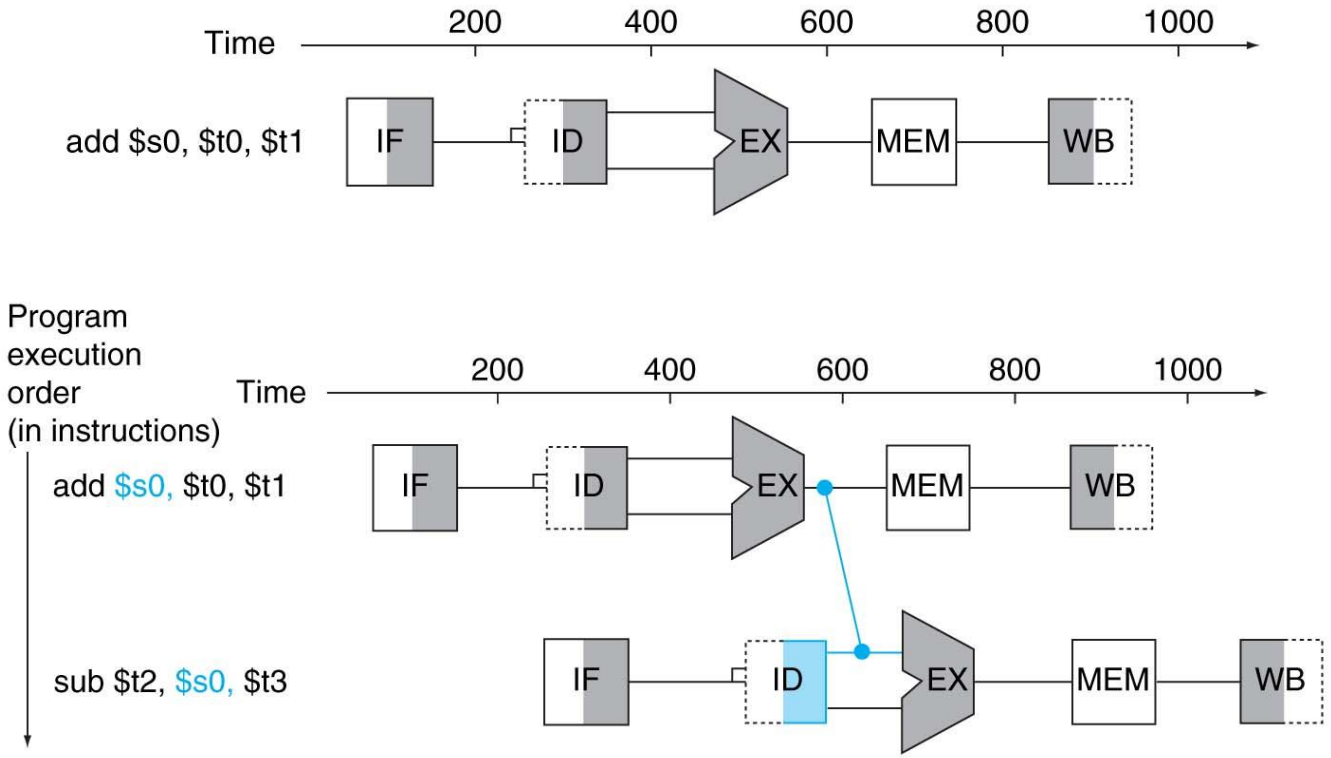
- A ____ hazard occurs when a _____ _____ has not yet been _____ to the _____ ____.
- Consider

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

- Though the result is not _____ until ___, it is available after the `add` has finished the EX stage, _____ it to the right place.

4.5 Two Instruction Forwarding

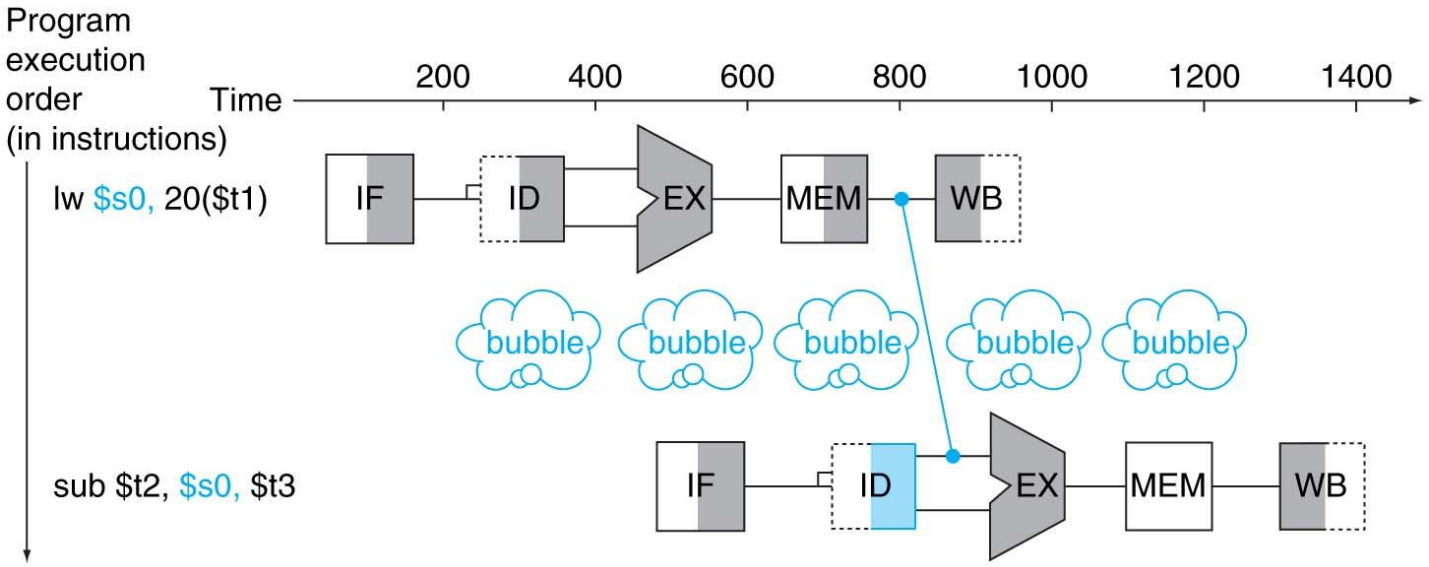
- Forwarding paths are _____ only if the destination stage is _____ than the source stage.



4.5 More on Forwarding

- Forwarding can't fix everything.
- Consider

```
lw      $s0, 20($t1)
sub     $t2, $s0, $t3
```

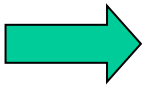


4.2 The Compiler Can Help

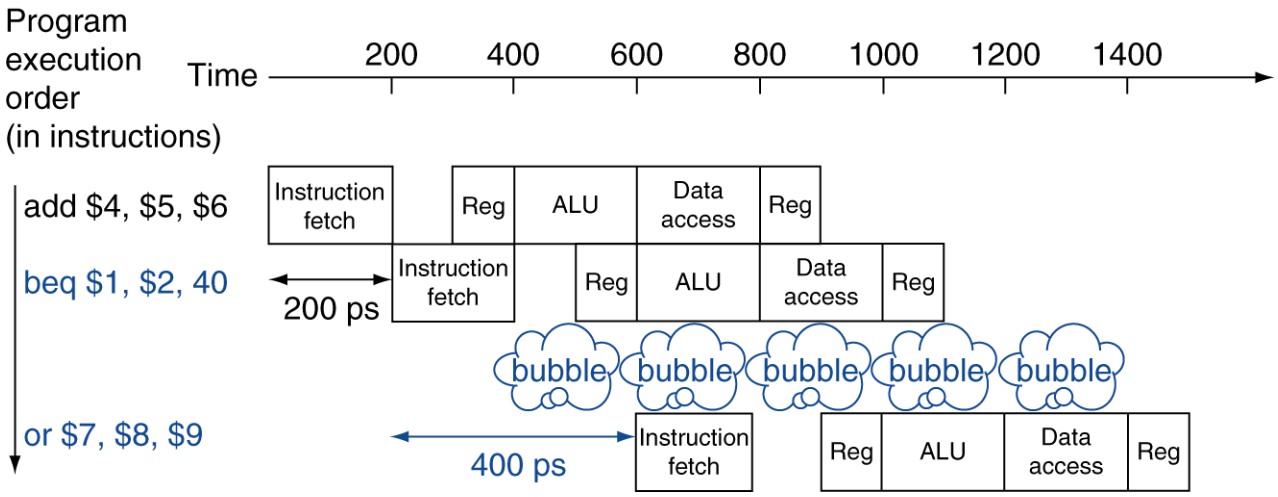
- Consider the following:

```
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)
```

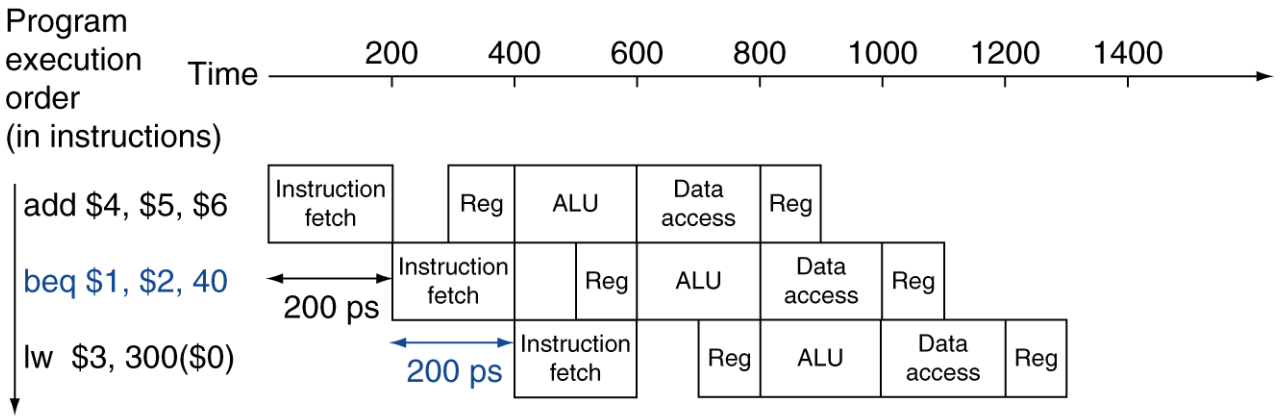


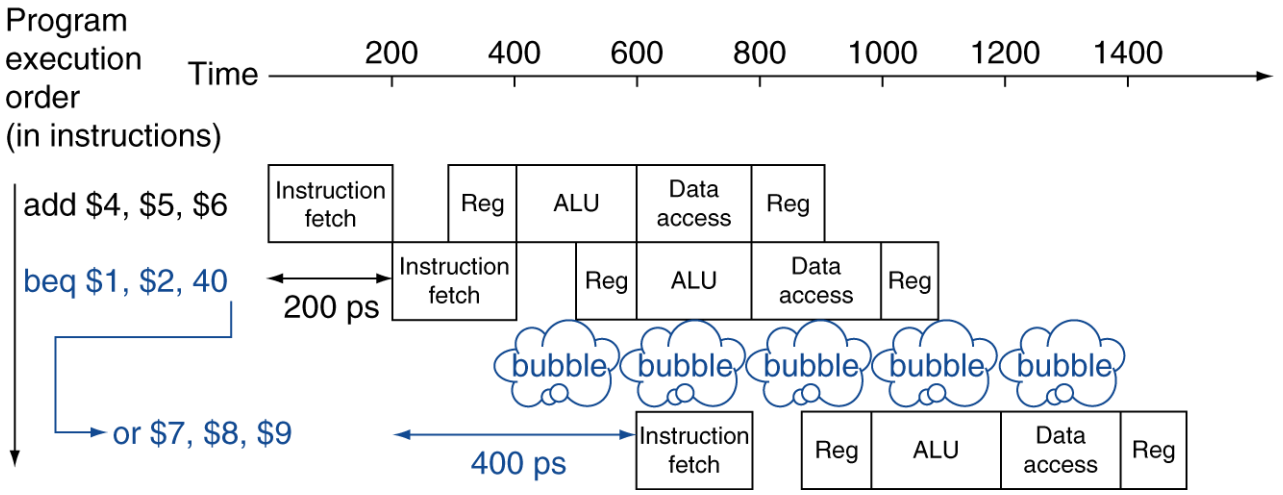
4.2 Stalling on Control Hazards



Performance of "Stall on Branch"

4.2 Prediction for Control Hazards

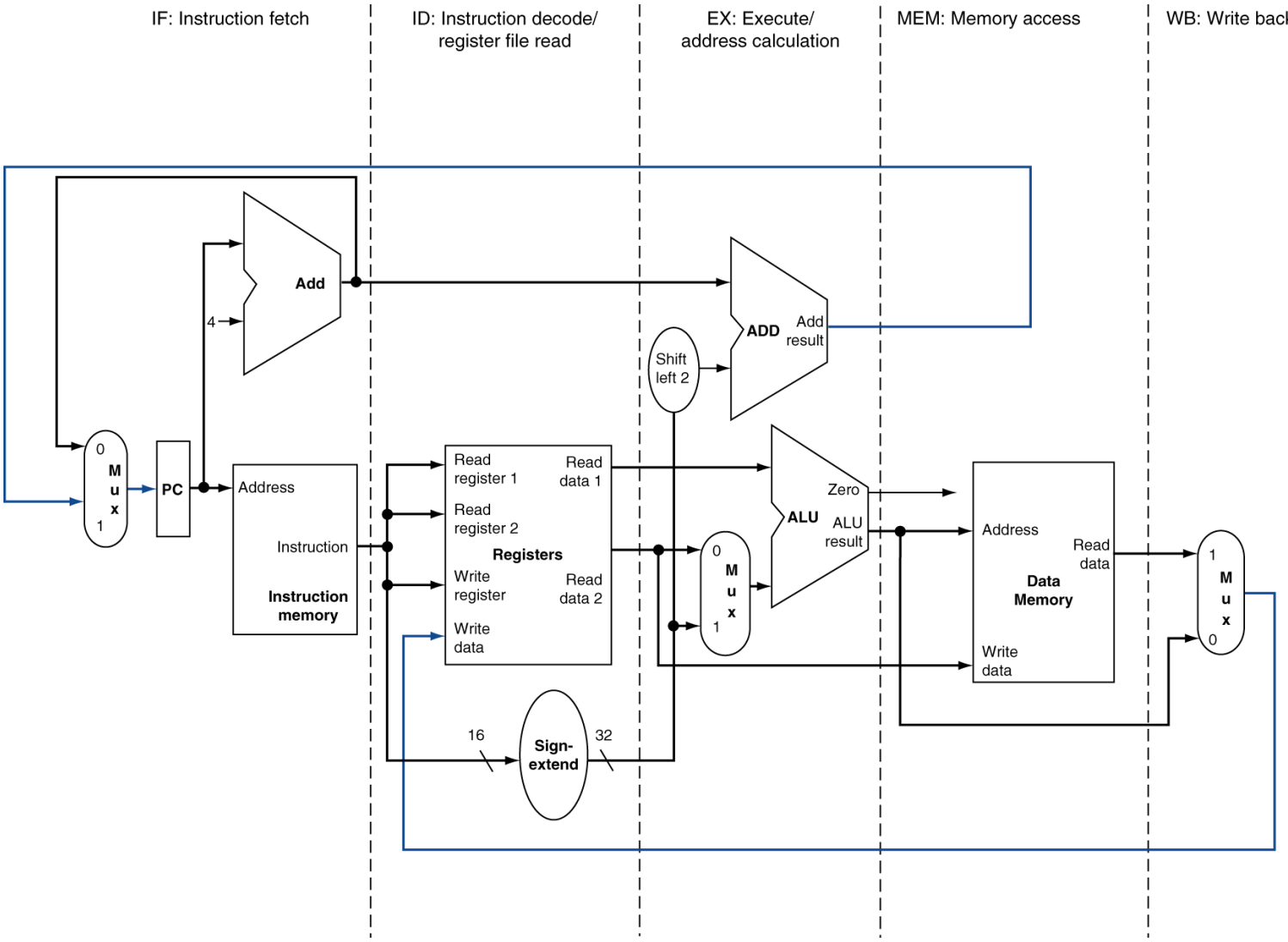




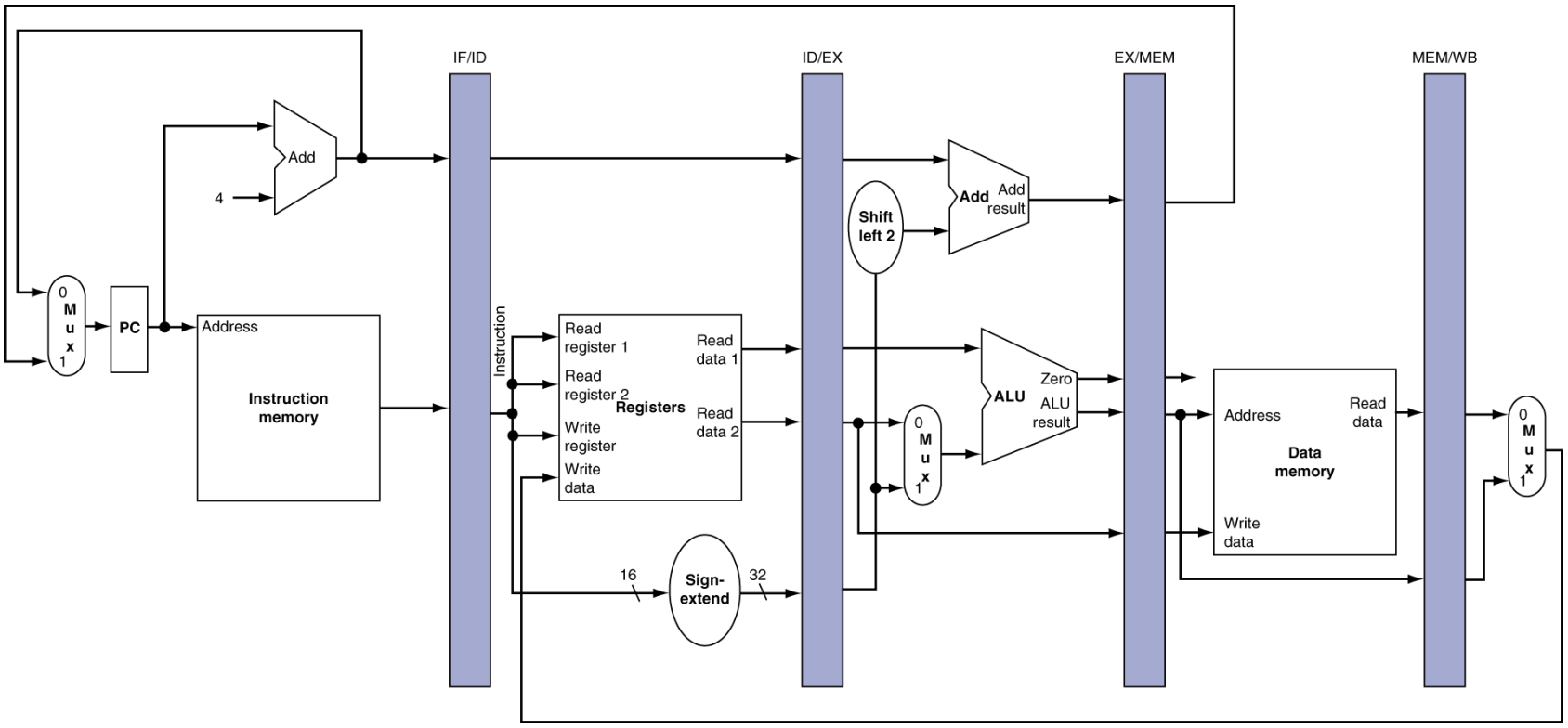
4.5 More Realistic Branch Prediction

- _____ branch prediction
  - Based on _____ branch behavior
  - Example: ____ branches
    - Predict _____ branches _____
    - Predict _____ branches _____
- _____ branch prediction
  - _____ measures _____ branch behavior
    - Record _____ _____ of each branch
  - Assume _____ behavior will continue the trend
    - When _____, stall while re-fetching, and update _____

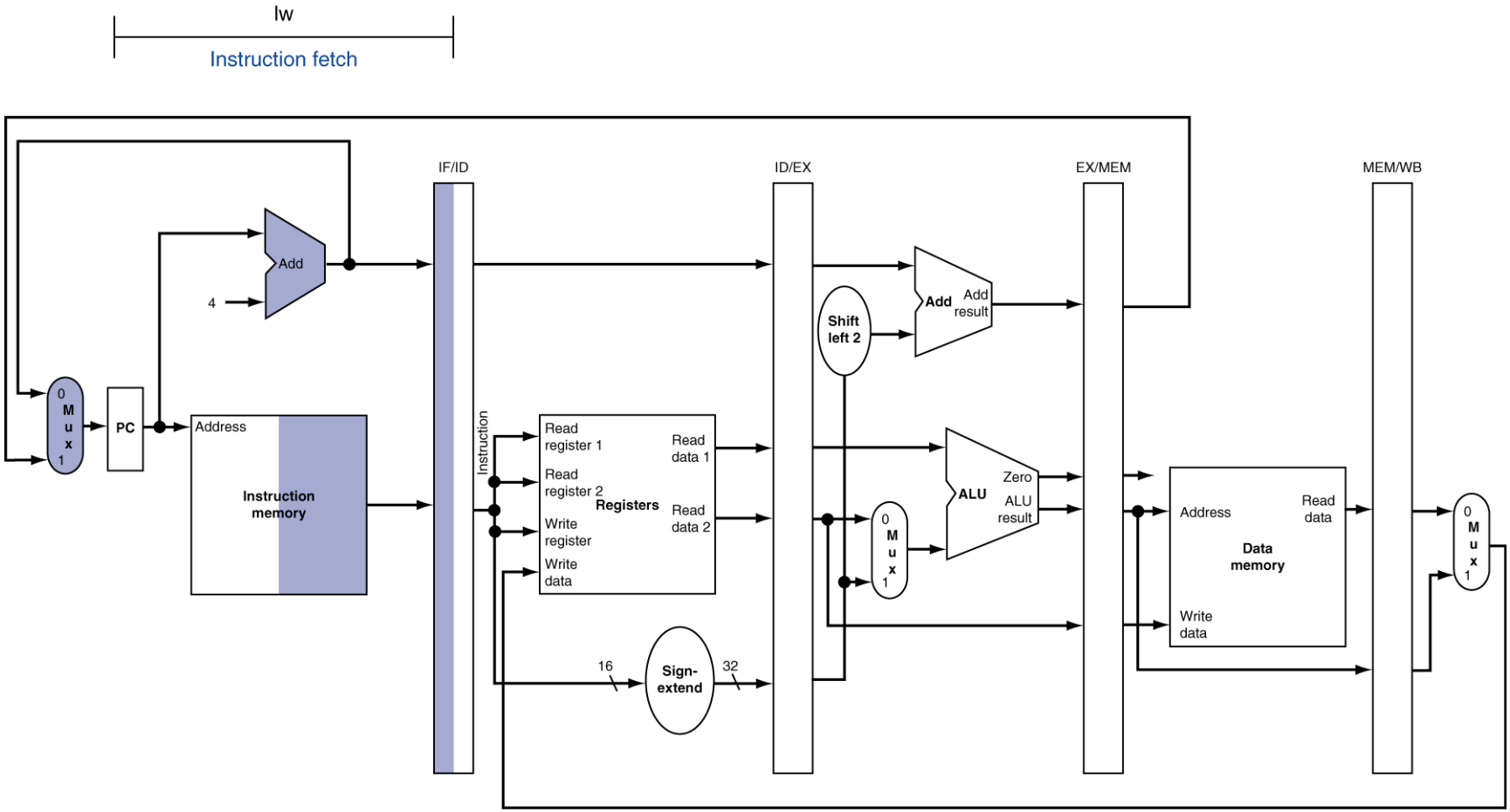
4.6 Identifying the Pipeline Stages



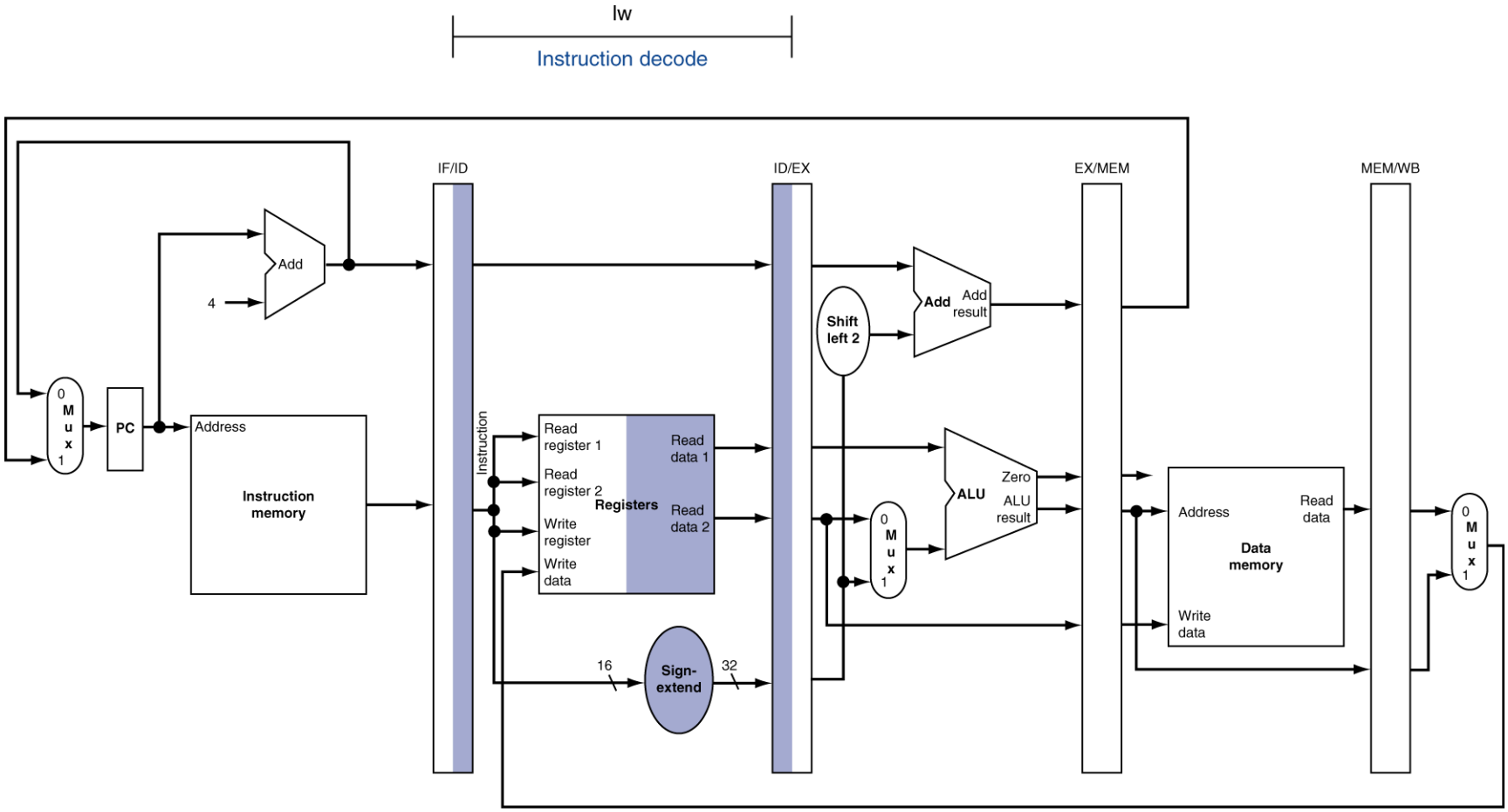
4.6 Adding Pipeline Registers



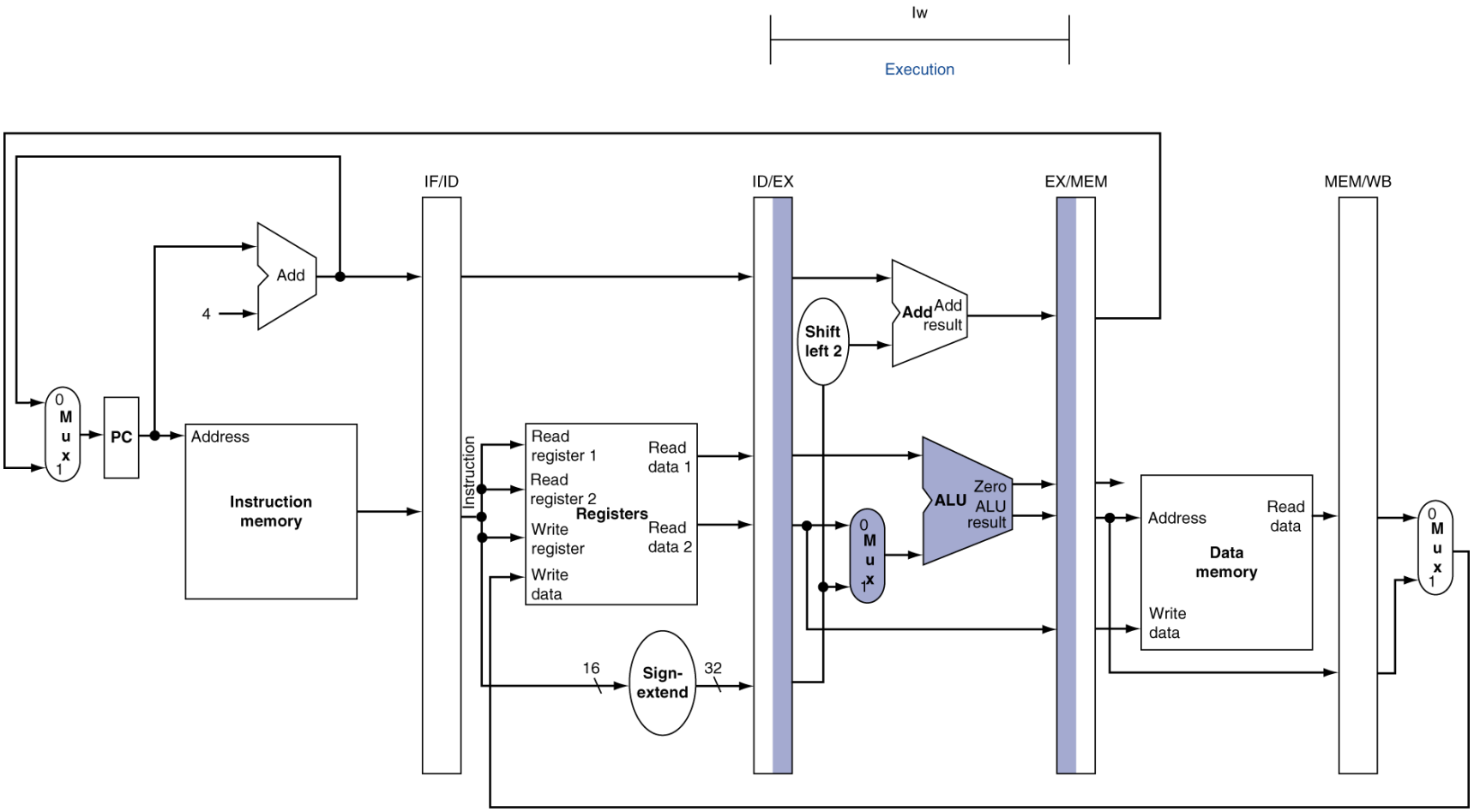
4.6 `lw` Instruction Execution: IF Stage



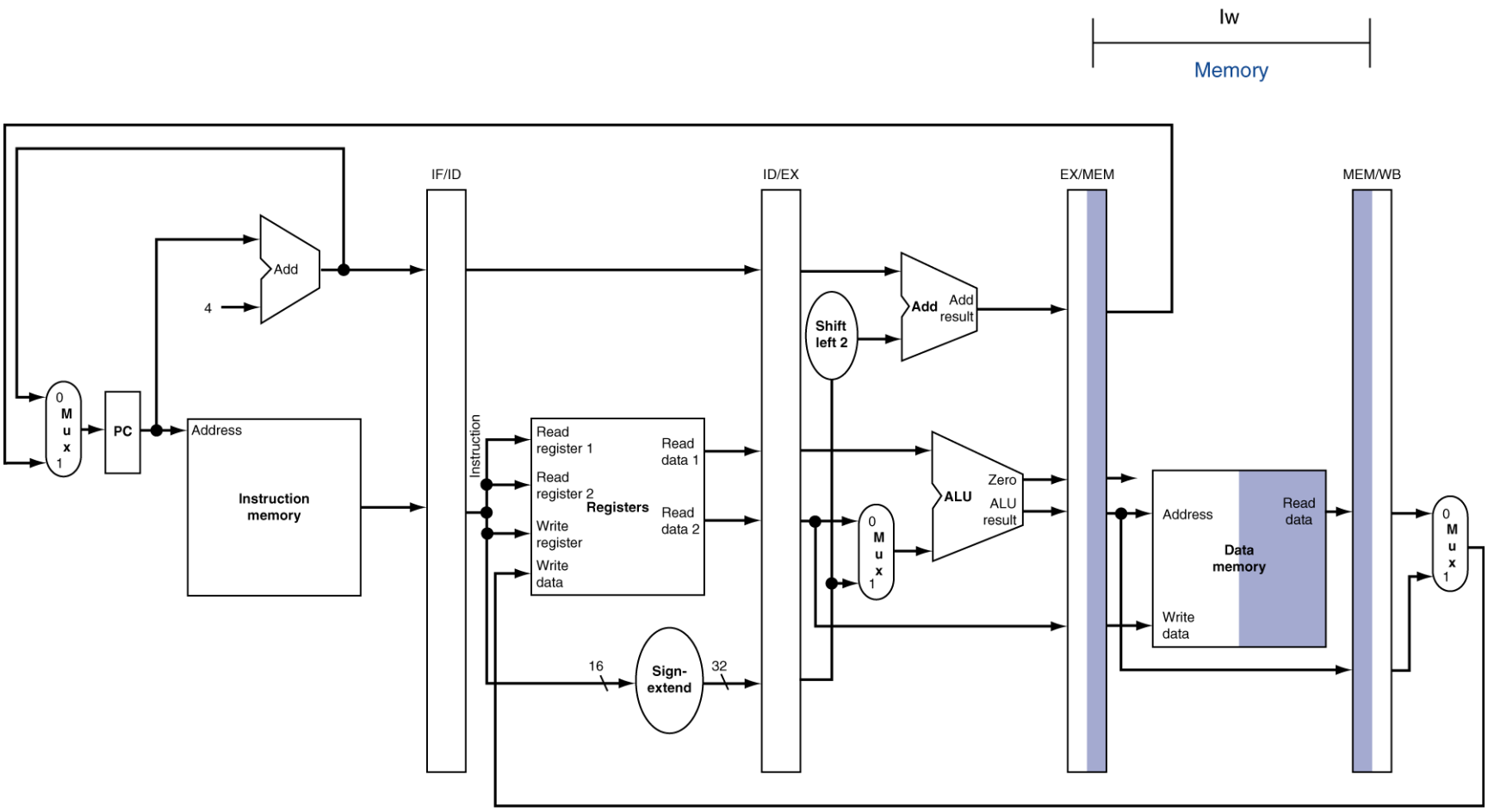
4.6 `lw` Instruction Execution: ID Stage



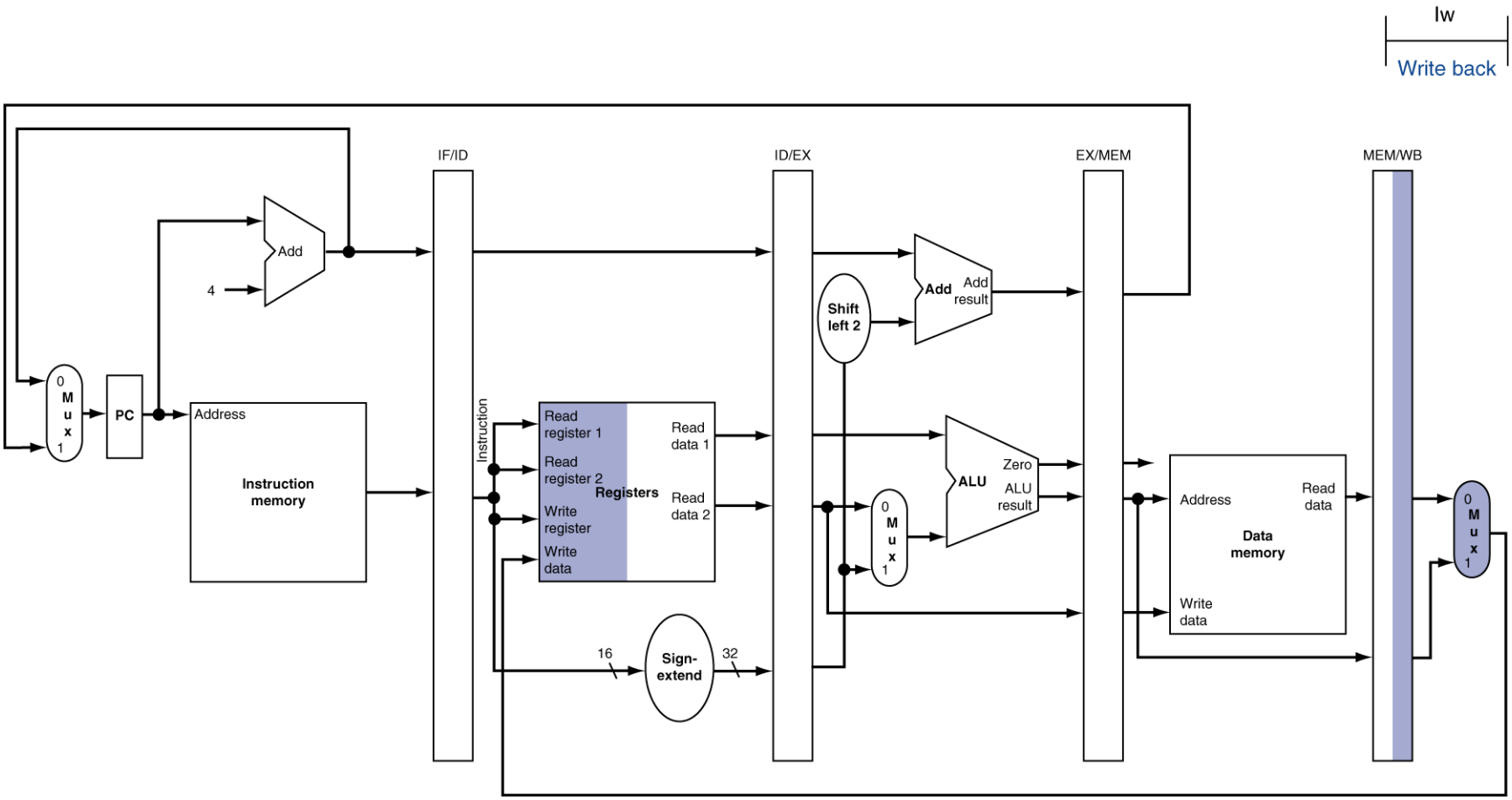
4.6 `lw` Instruction Execution: EX Stage



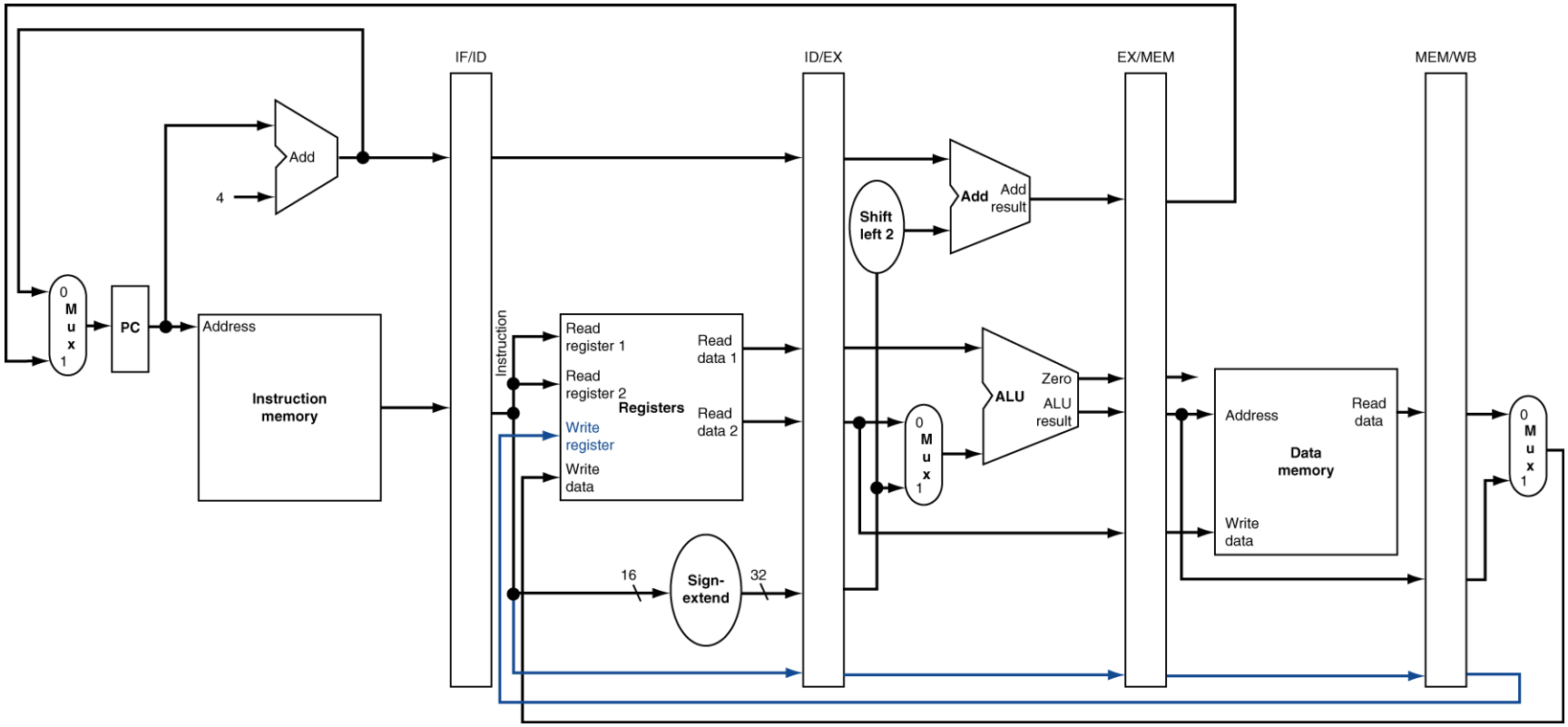
4.6 `lw` Instruction Execution: MEM Stage



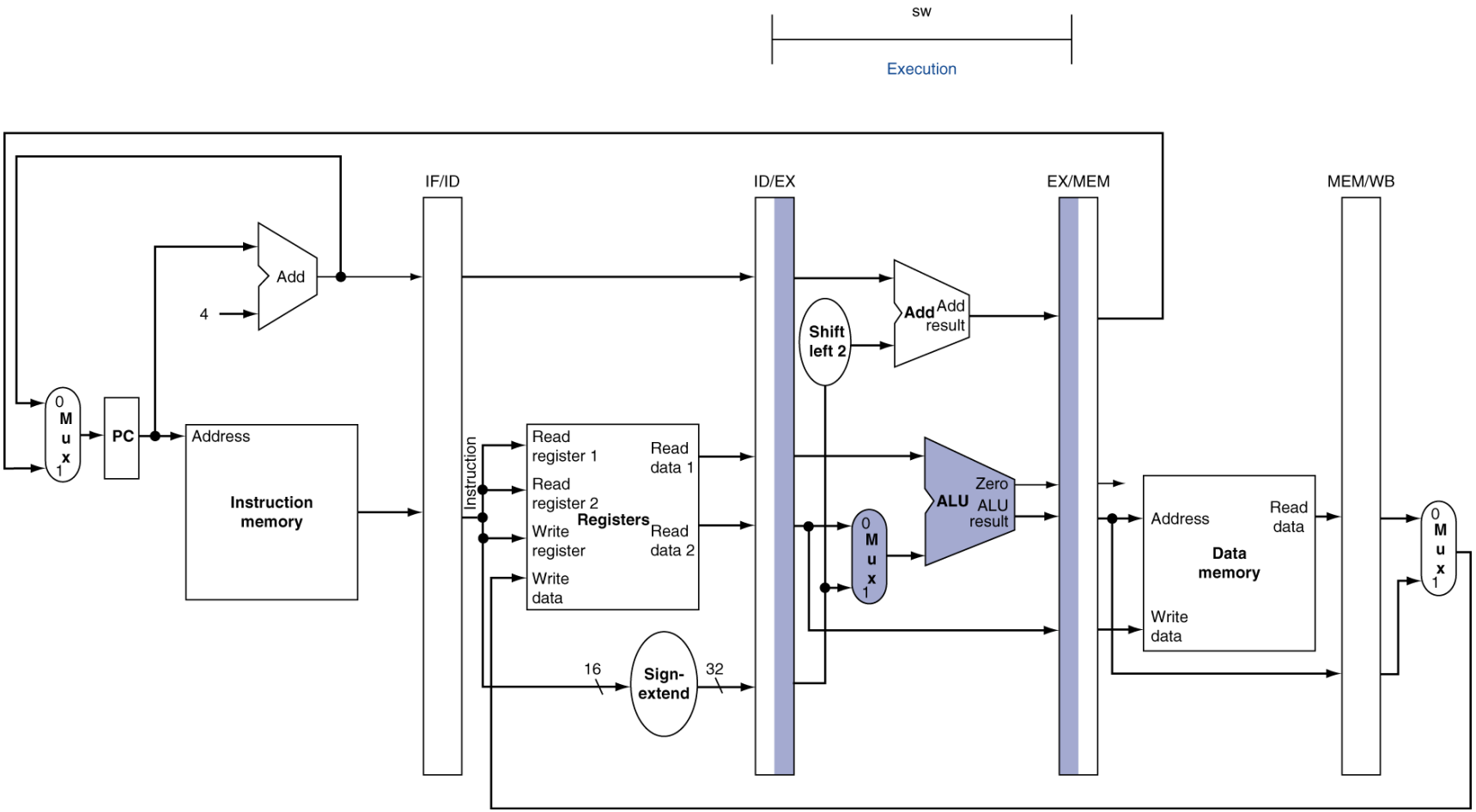
4.6 `lw` Instruction Execution: WB Stage



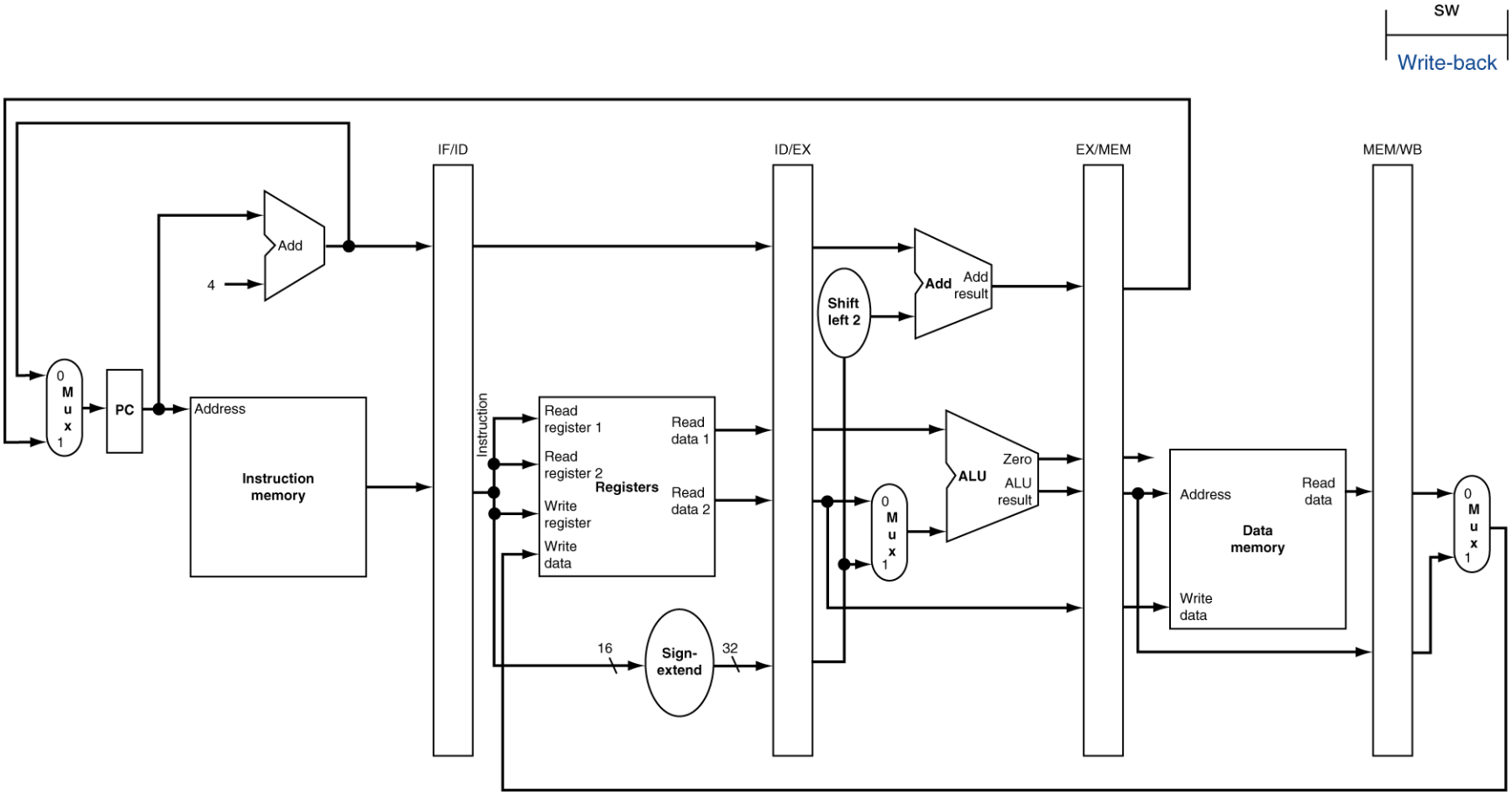
4.6 Corrected Datapath for `lw`



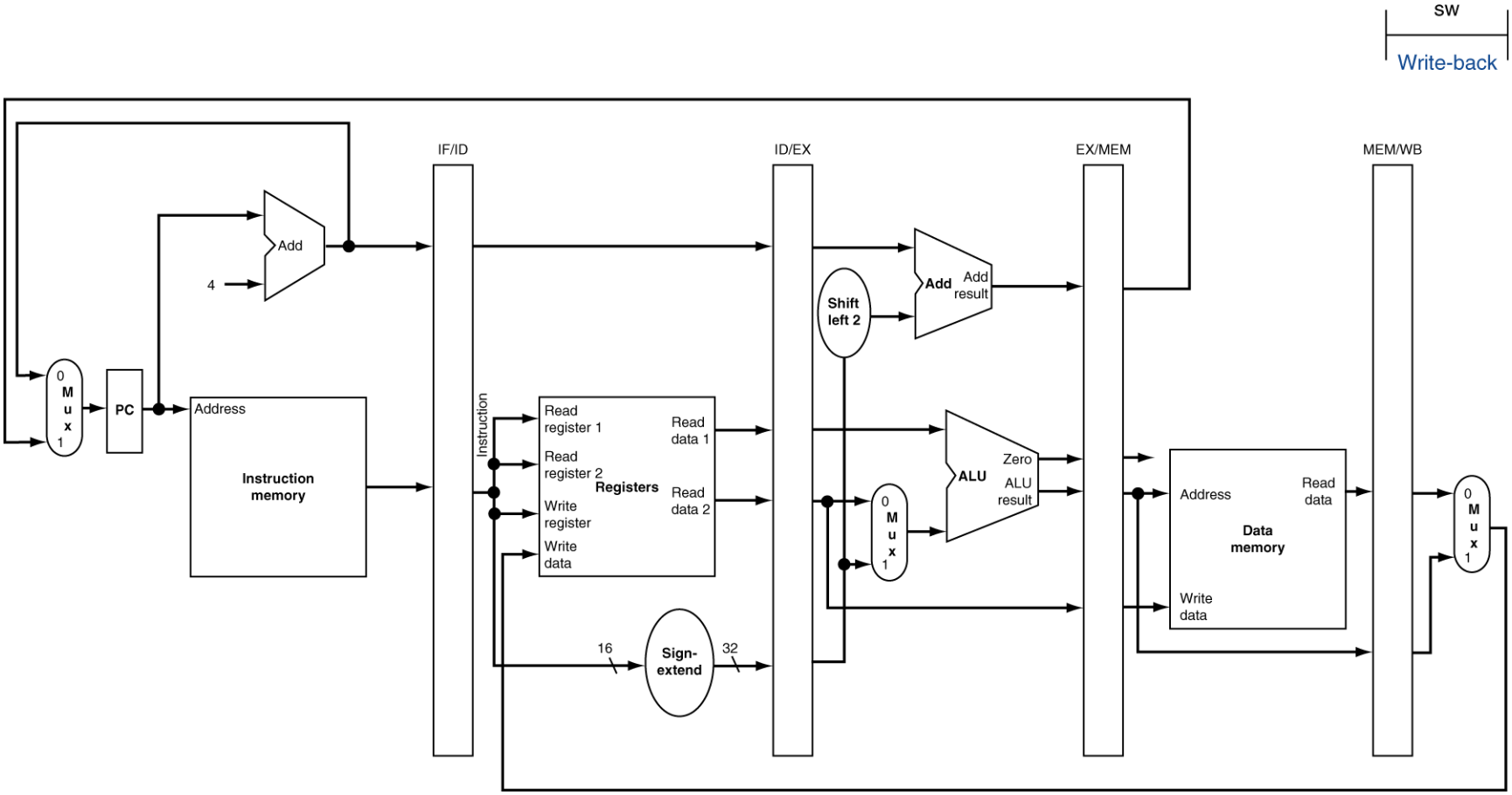
4.6 sw Instruction Execution: EX Stage



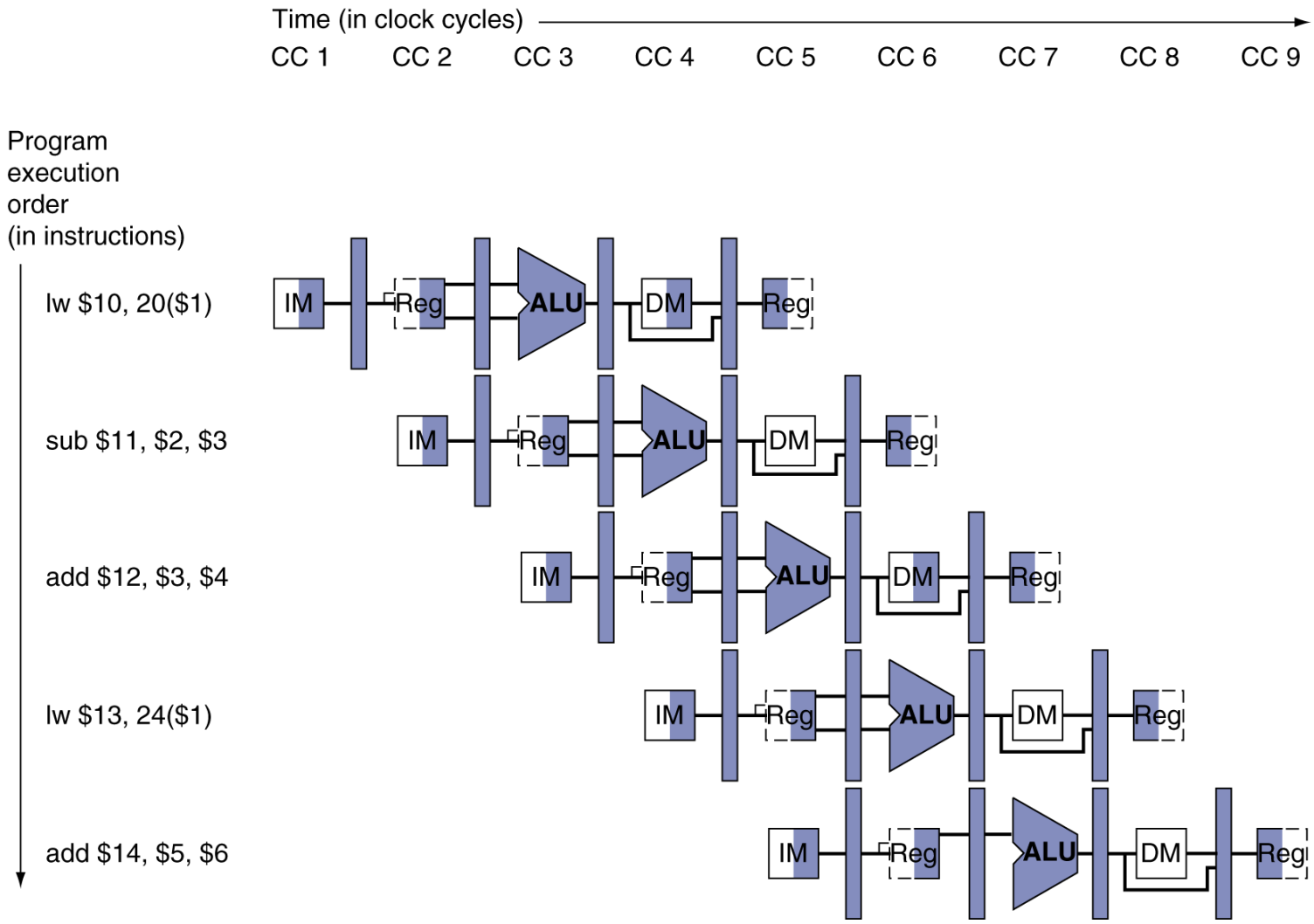
4.6 sw Instruction Execution: MEM Stage



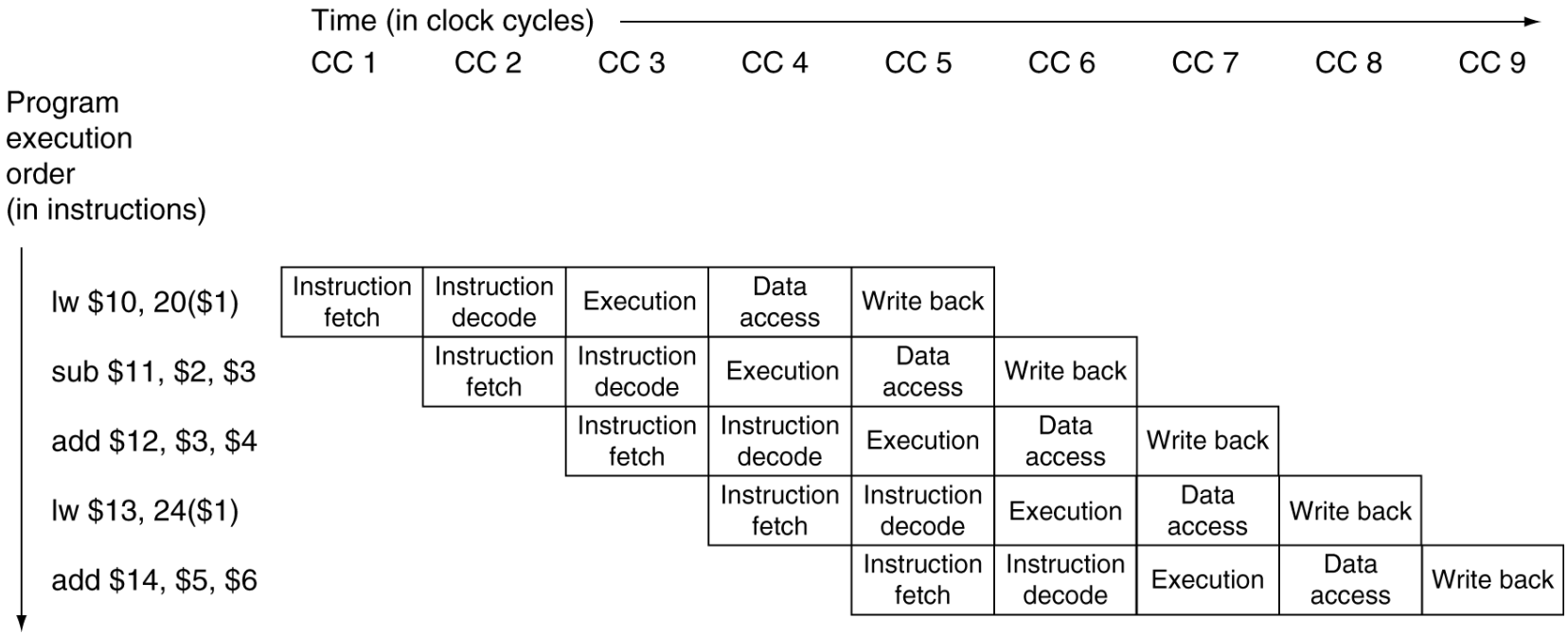
4.6 `sw` Instruction Execution: WB Stage



4.6 Stylized Multiple Clock Cycle Diagrams



4.6 Traditional Multiple Clock Cycle Diagrams

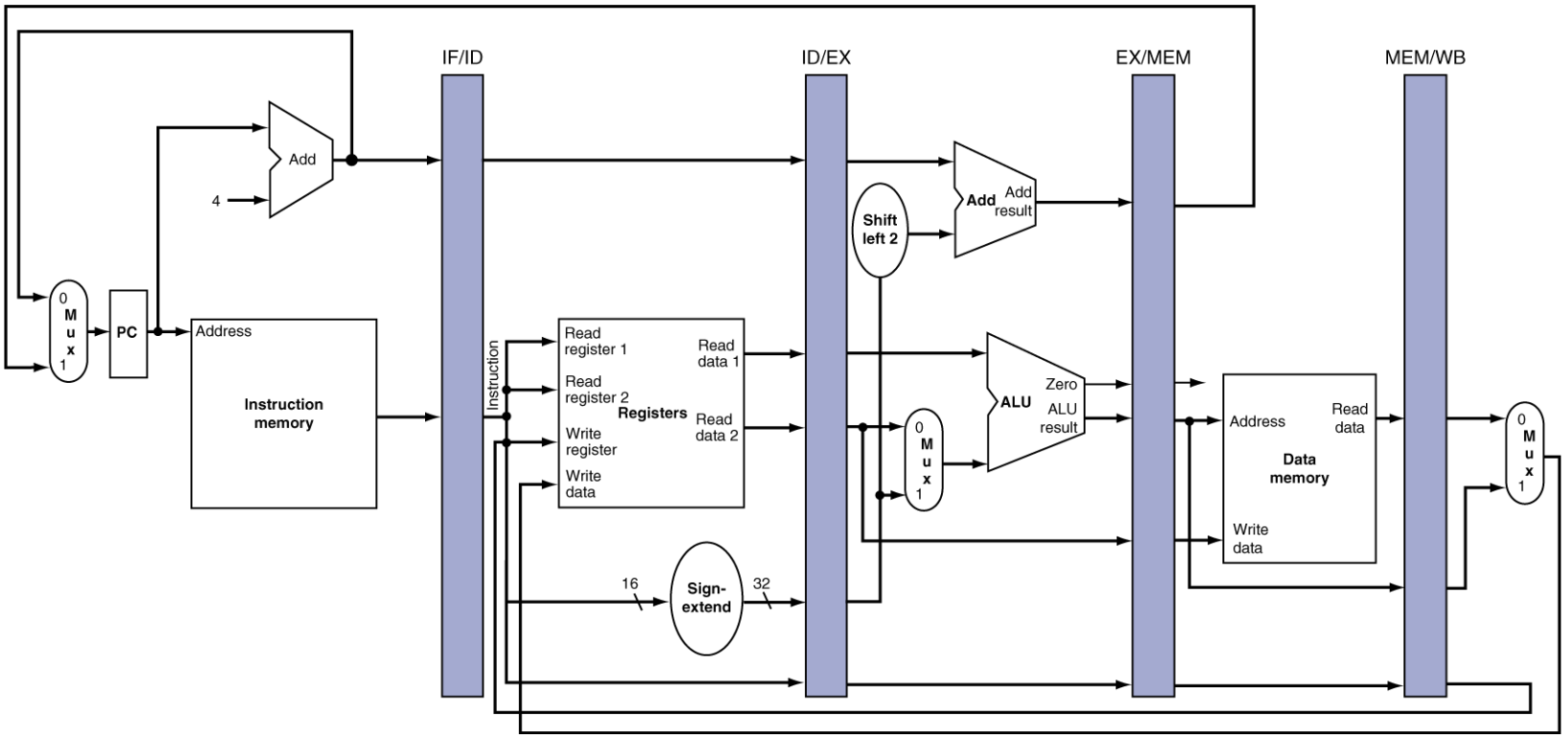


4.6 Alternate Multiple Clock Cycle Diagrams

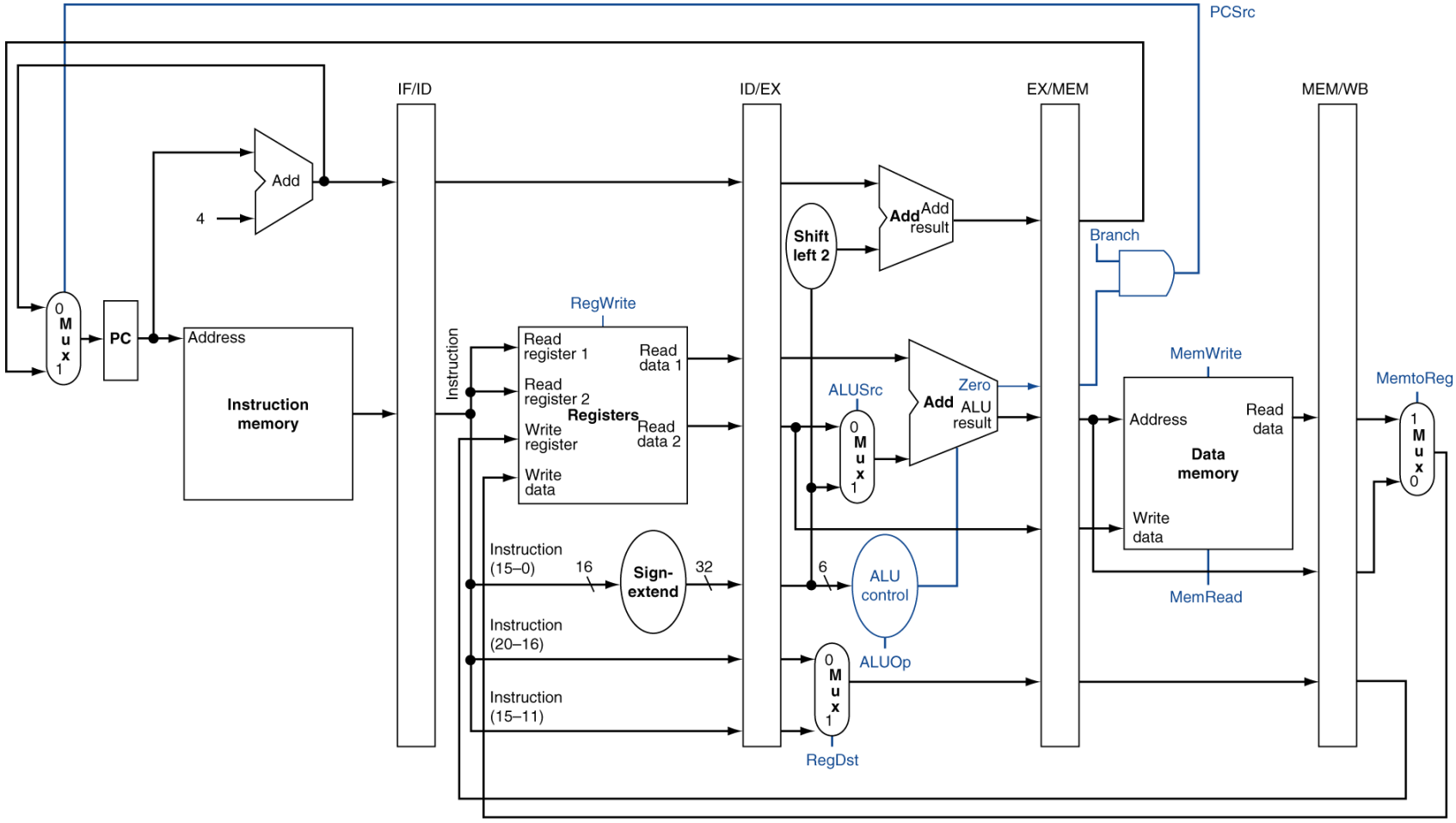
Cycle	IF	ID	EX	MEM	WB
1	lw $10	ahead1	ahead2	ahead3	ahead4
2	sub $11	lw $10	ahead1	ahead2	ahead3
3	add $12	sub $11	lw $10	ahead1	ahead2
4	lw $13	add $12	sub $11	lw $10	ahead1
5	add $14	lw $13	add $12	sub $11	lw $10
6	after1	add $14	lw $13	add $12	sub $11
7	after2	after1	add $14	lw $13	add $12
8	after3	after2	after1	add $14	lw $13
9	after4	after3	after2	after1	add $14

4.6 Cycle 5 Slice

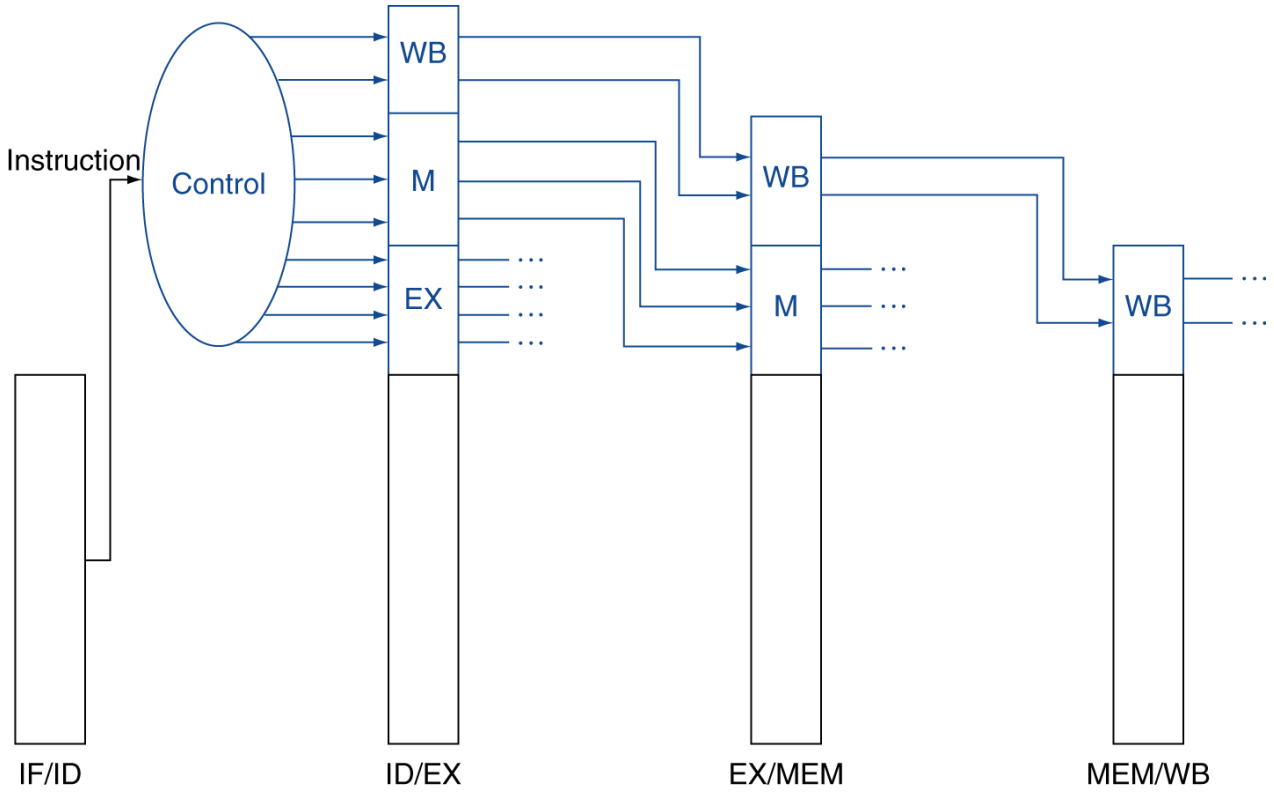
add $14, $5, $6	lw $13, 24 ($1)	add $12, $3, $4	sub $11, $2, $3	lw $10, 20($1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back



4.6 Identifying Pipelined Control Lines

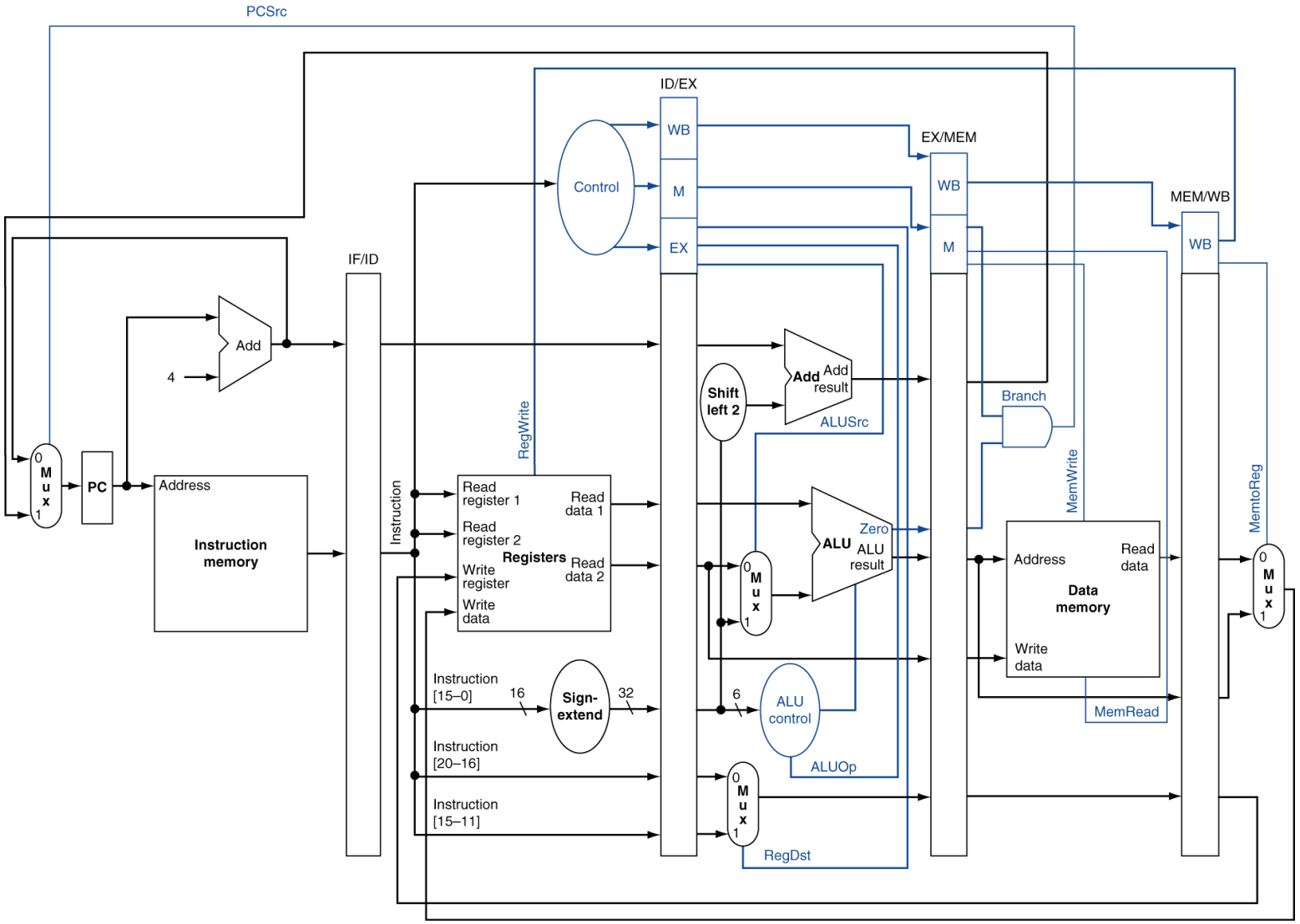


4.6 Generating and Saving Control Lines



- EX
- MEM
- WB

4.6 Putting it all Together

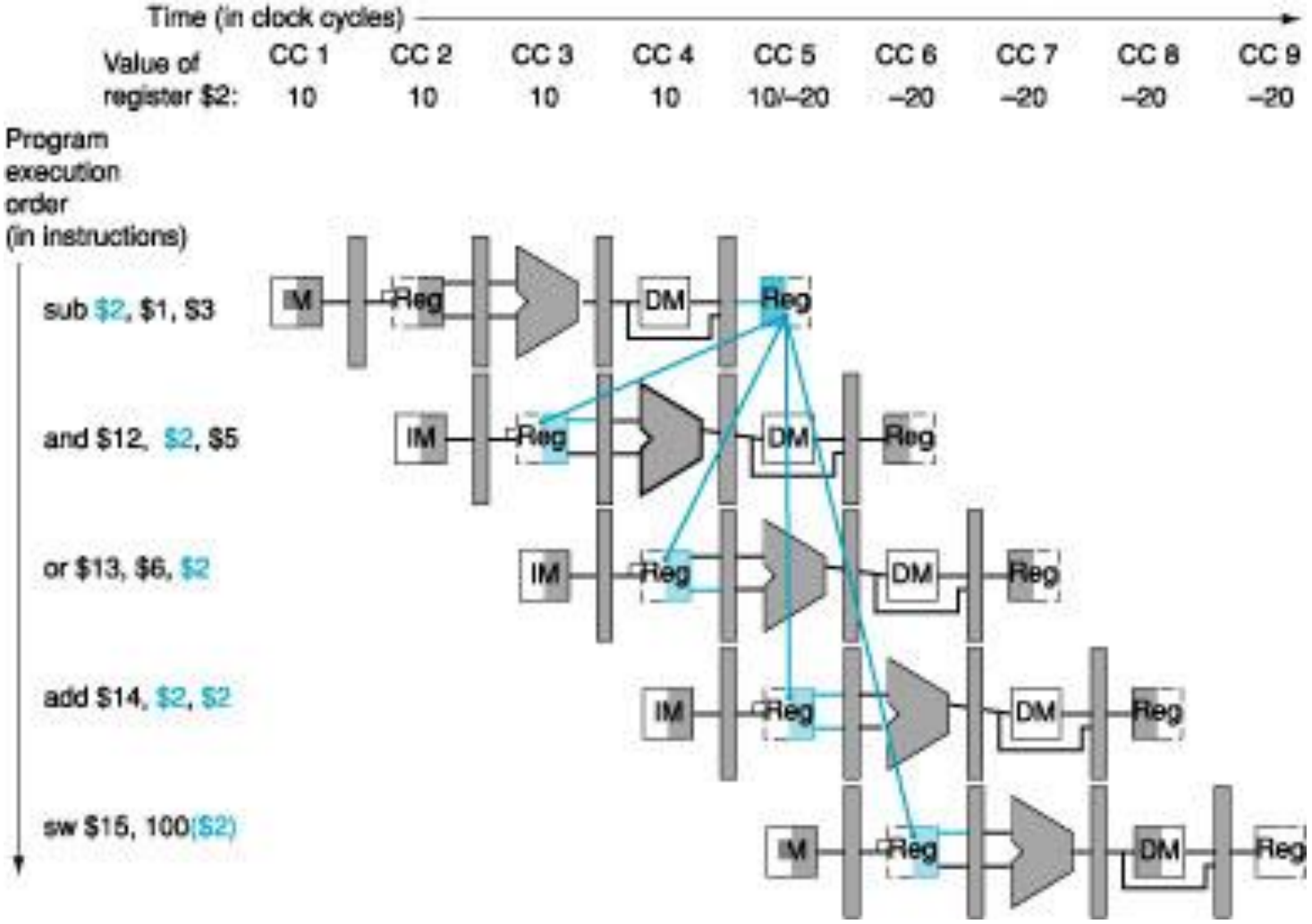


4.7 Data Dependencies

In the previous example, there were no data dependencies. Now, the rest of the story.

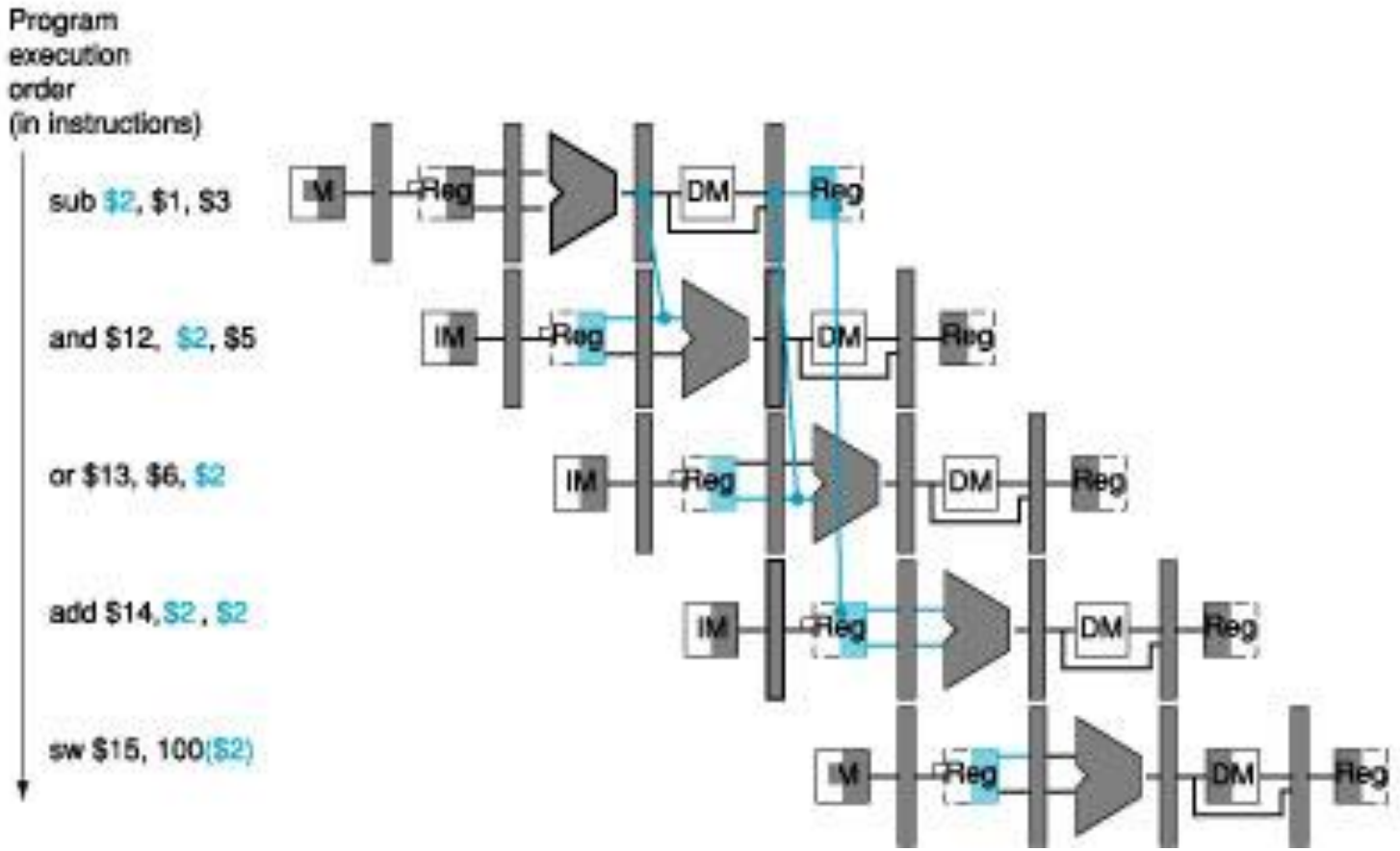
```
1   sub   $2, $1, $3
2   and   $12, $2, $5
3   or    $13, $6, $2,
4   add   $14, $2, $2
5   sw    $15, 100($2)
```

4.7 Which Data Dependencies are Hazards

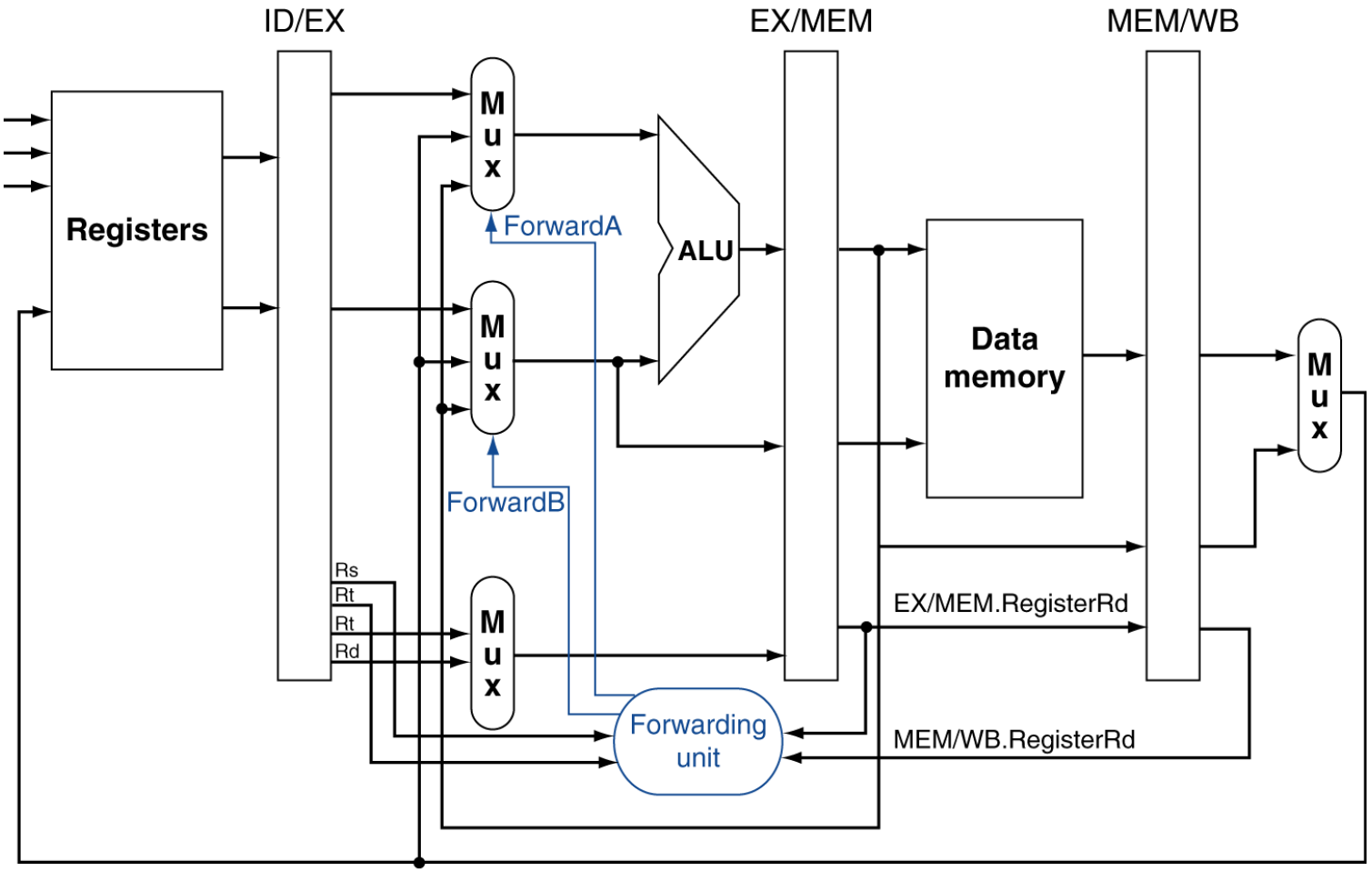


4.7 Forwarding in Action

Time (in clock cycles)	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X



4.7 Forwarding Paths



b. With forwarding

4.7 Forwarding Unit: Classifying Hazards

- Type 1 – The information needed in the ___ stage by an instruction is the result of the instruction ___ stage ahead (found in the _____ pipeline register)
  - a. The information is needed in R[rs]
  - b. The information is needed as R[rt]

- Type 2 – The information needed in the ___ stage by an instruction is the result of the instruction ___ stages ahead (found in the _____ pipeline register)
  - a. The information is needed in R[rs]
  - b. The information is needed as R[rt]

4.7 Forwarding Unit: Type 1 Hazards

Type 1 Hazard

If (ahead1 writes to register and (ahead1 doesn't have $zero as the destination register) and (ahead1 is writing to a register read by Current Instruction)) Forward from EX/MEM a) Forward to RegisterRs, b) Forward to RegisterRd

ahead1 writes to register

ahead1 doesn't have $zero as the destination register

ahead1 is writing to a register read by Current Instruction

a)

b)

4.7 Forwarding Unit: Type 2 Hazards

Type 2 Hazard

If (ahead2 writes to register and (ahead2 doesn't have $zero as the destination register) and (ahead2 is writing to a register read by Current Instruction)) Forward from MEM/WB a) Forward to RegisterRs, b) Forward to RegisterRd

ahead2 writes to register

ahead2 doesn't have $zero as the destination register

ahead2 is writing to a register read by Current Instruction

a)

b)

4.7 Double Data Hazard Jeopardy

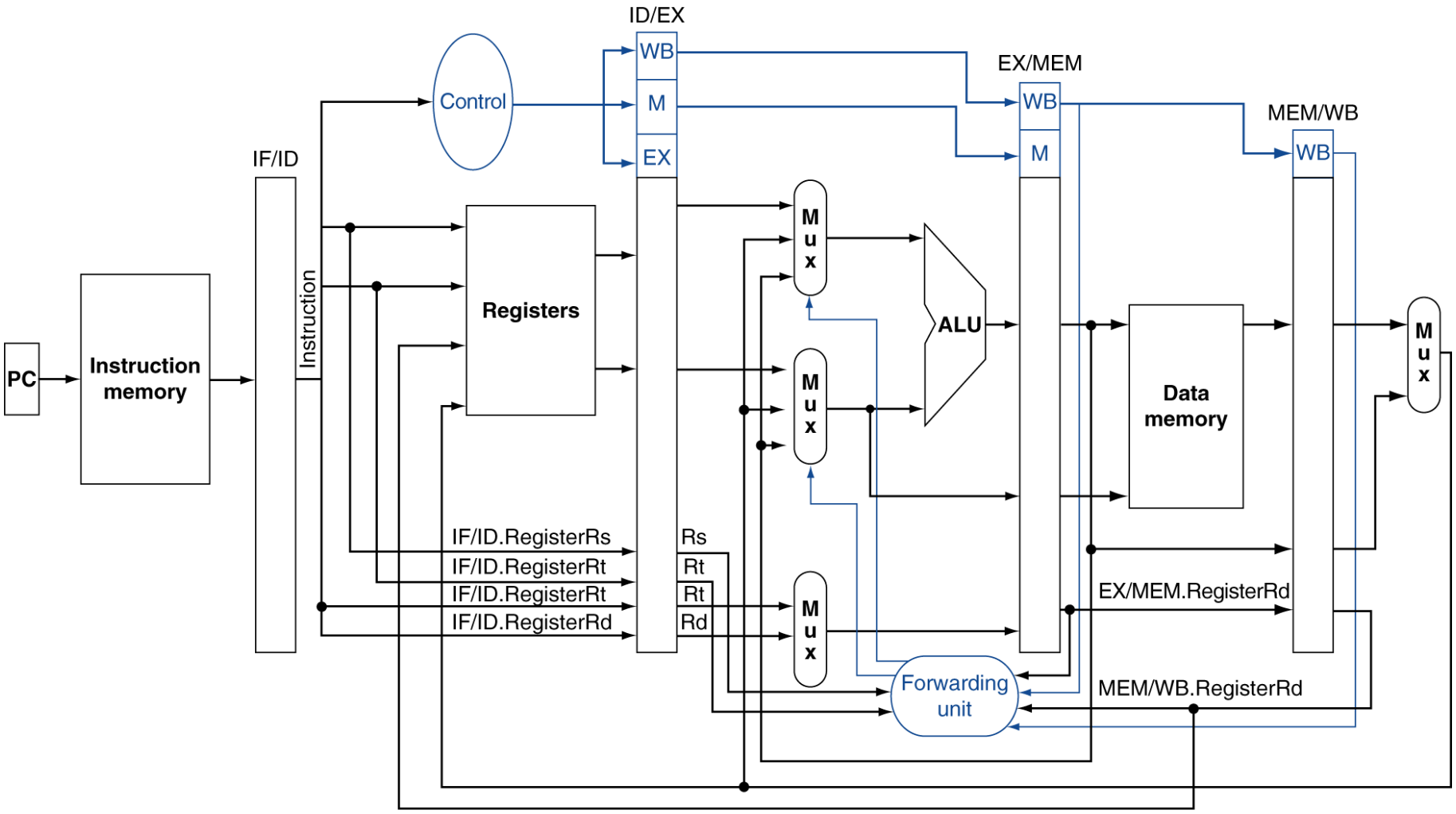
Consider

```
add  $1, $1, $2
add  $1, $1, $3
add  $1, $1, $4
```

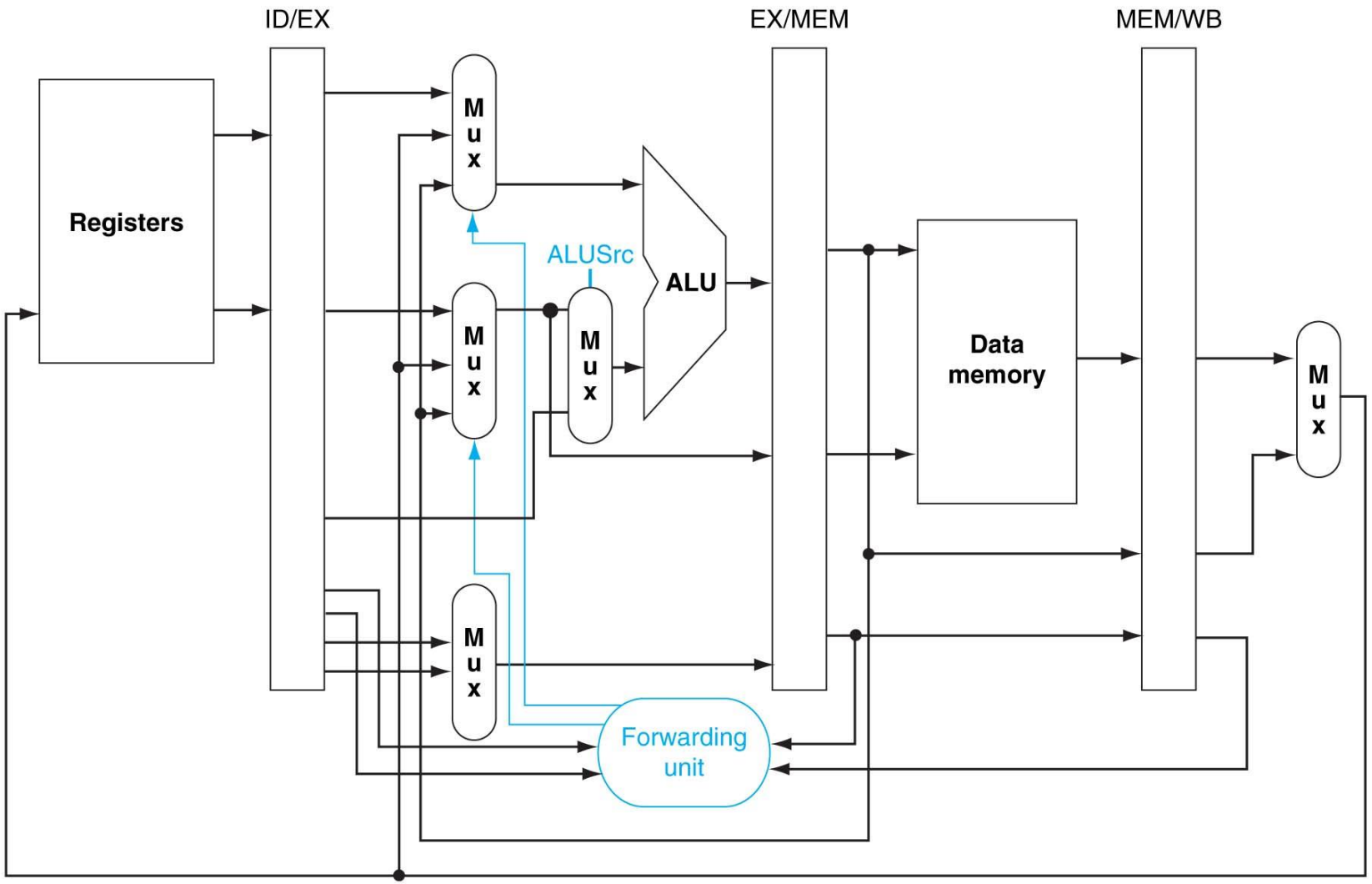
Type 2 Hazard

If (ahead2 writes to register and (ahead2 doesn't have $zero as the destination register) and (ahead2 is writing to a register read by Current Instruction) and (no Type 1 Hazard)) Forward from MEM/WB a) Forward to RegisterRs, b) Forward to RegisterRd

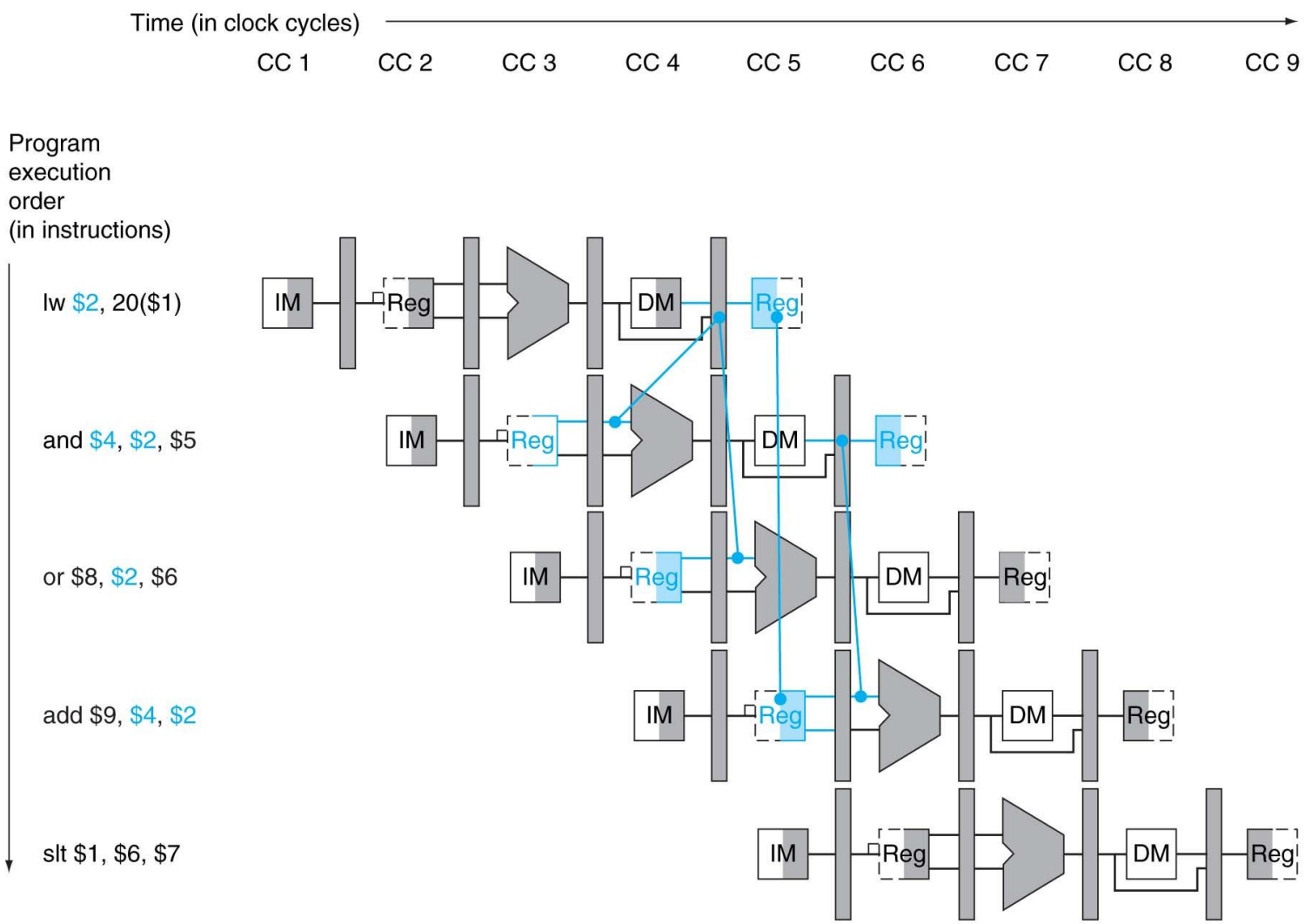
4.7 Forwarding Datapath with Control



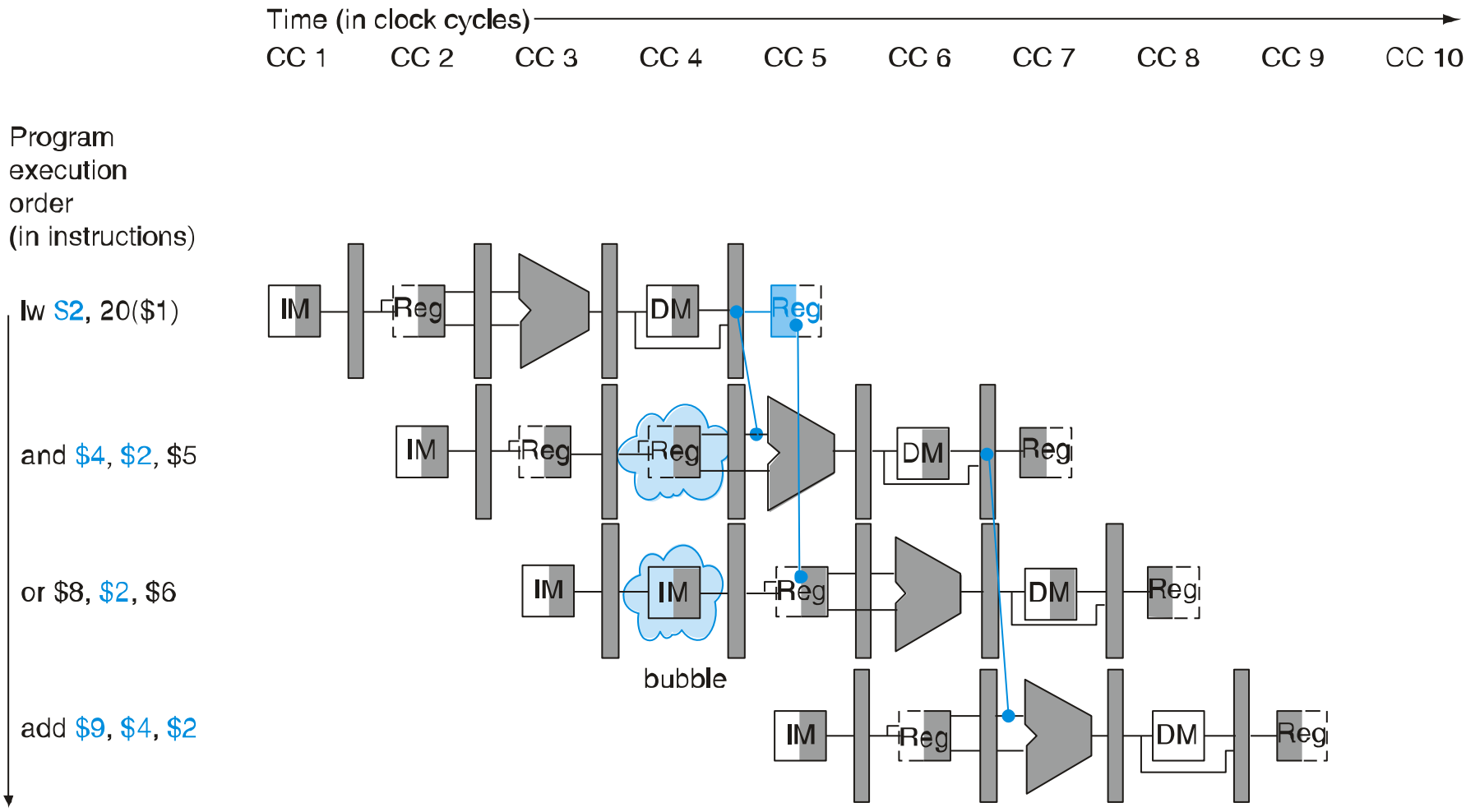
4.7 EX Forwarding Completed



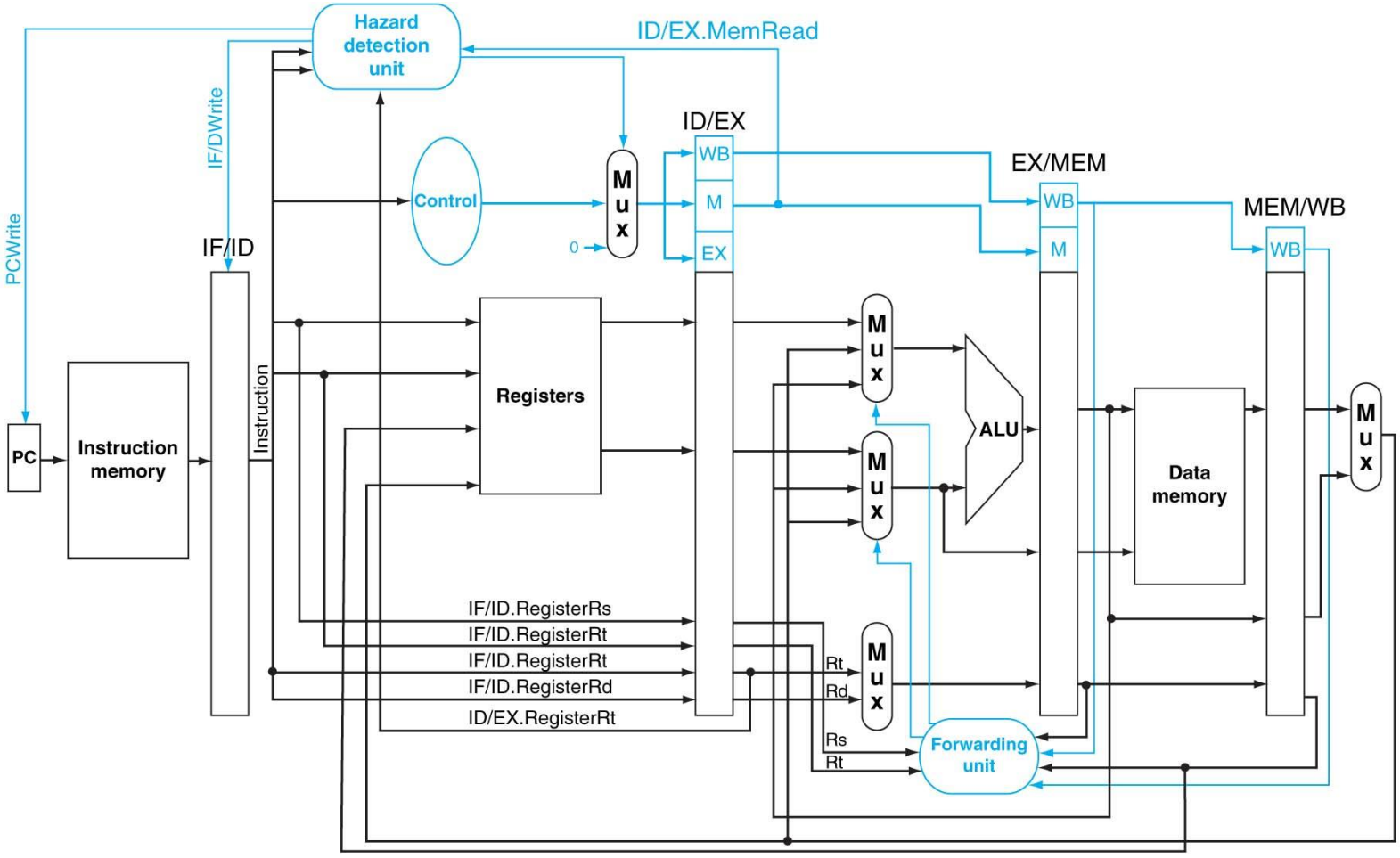
4.7 Forwarding Can't Always Save the Day



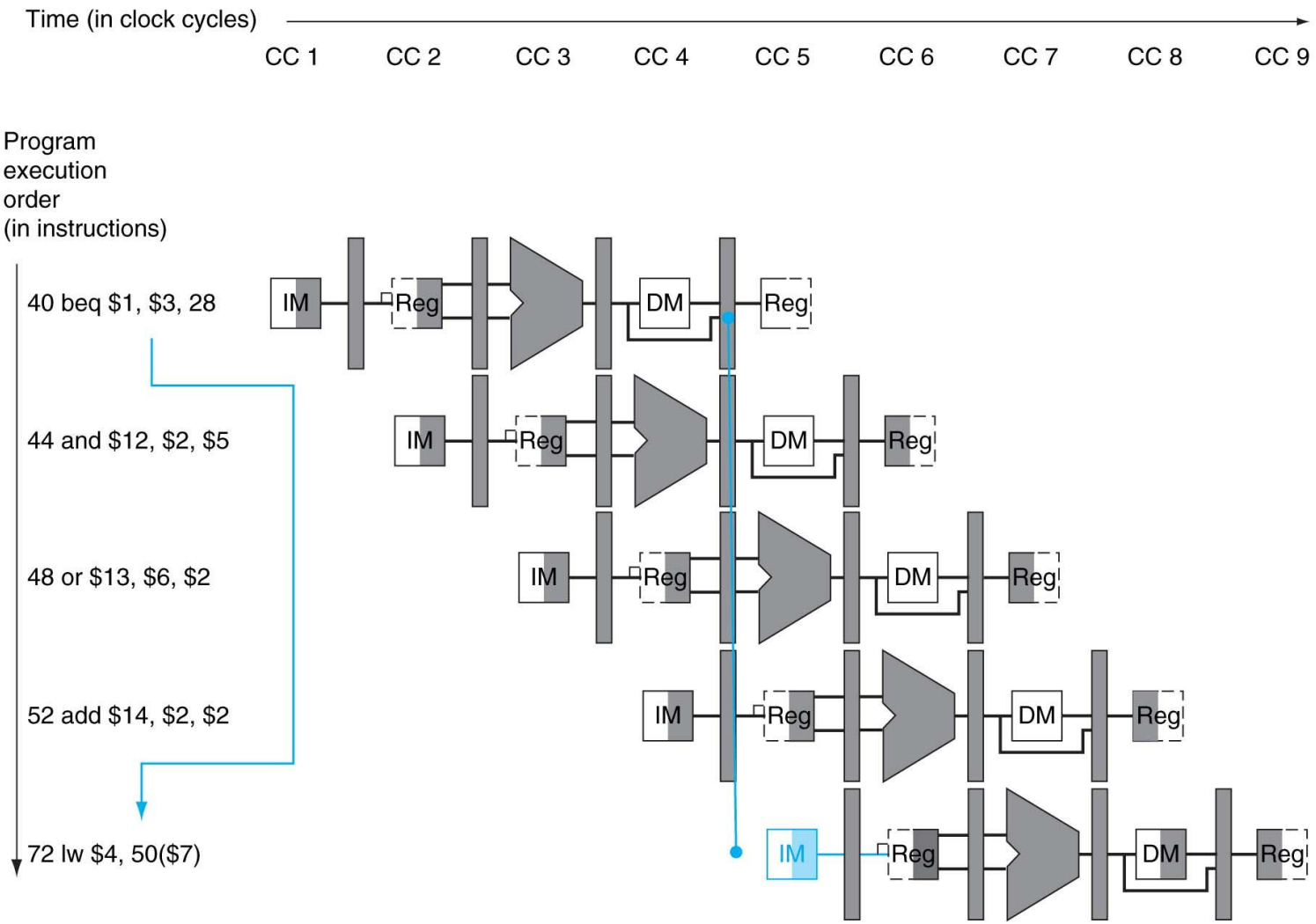
4.7 Stalling in Action



4.7 Hazard Detection Unit



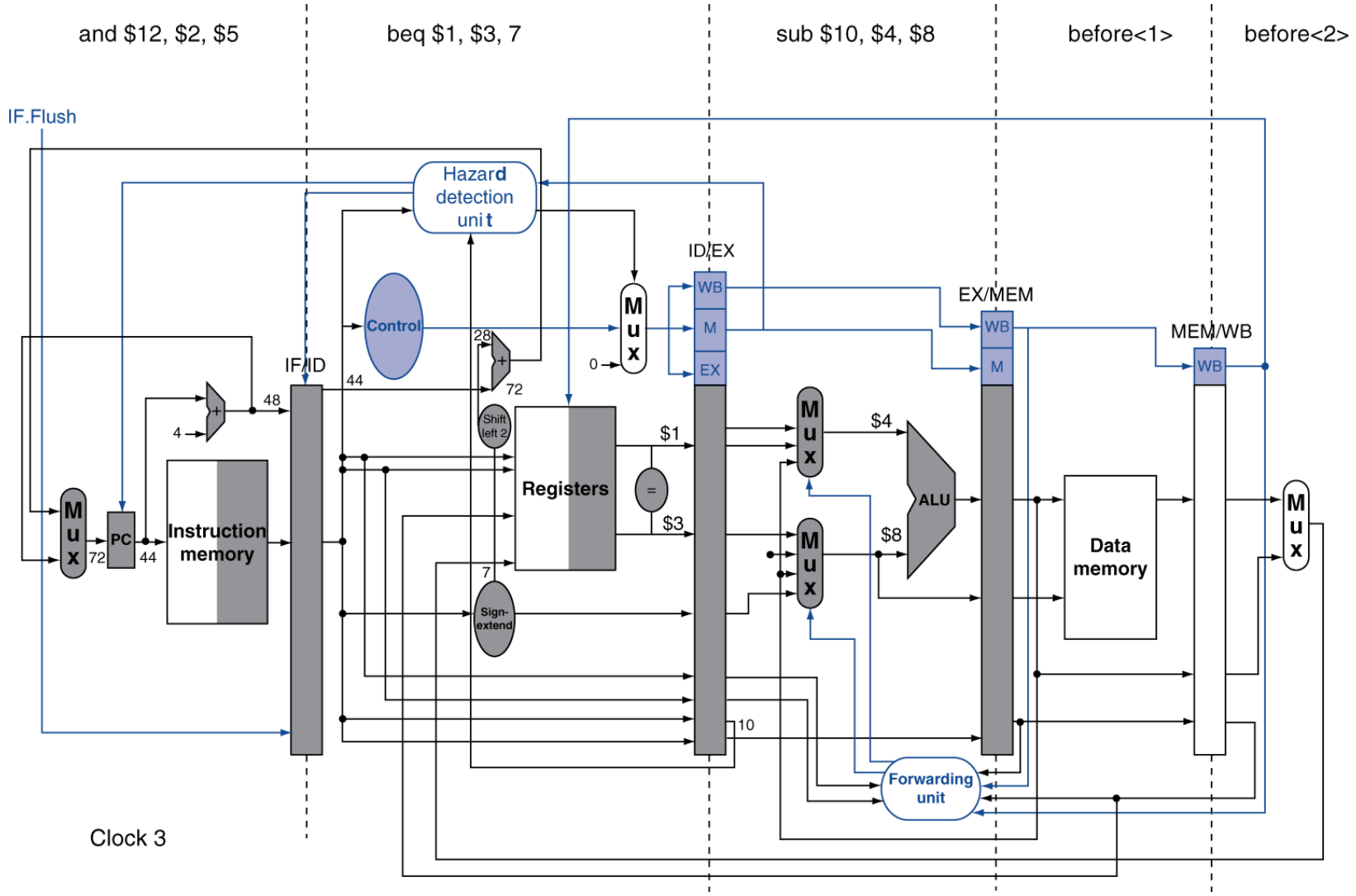
4.8 Control Hazard Example



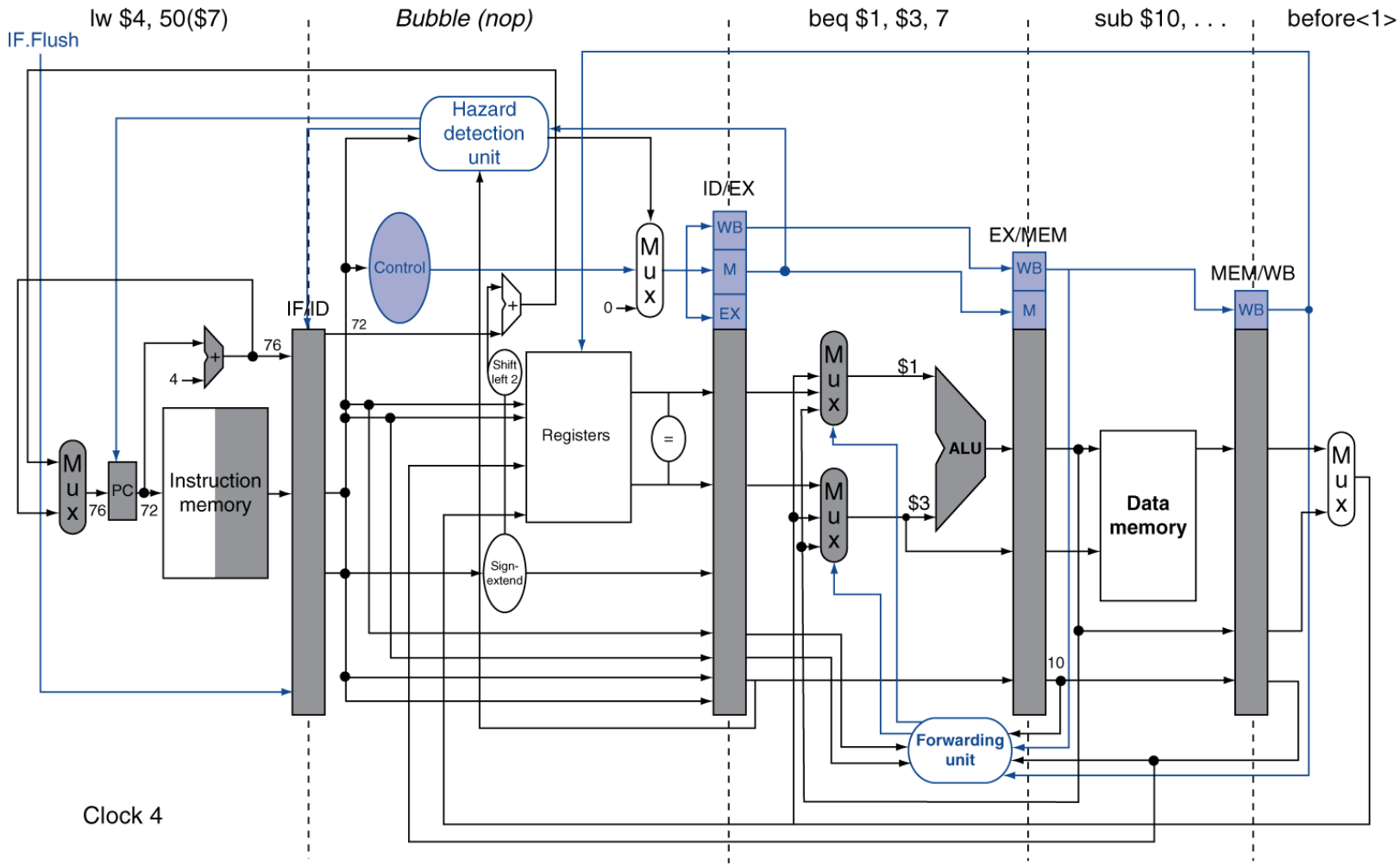
4.8 Approaches to Control Hazards

- Assume Branch Not Taken
  - _____
  - _____
- Reducing the Delay of Branches – Two Items Needed
  - _____
  - _____
  - _____
  - _____

4.8 Changes to Reduce Branch Delay



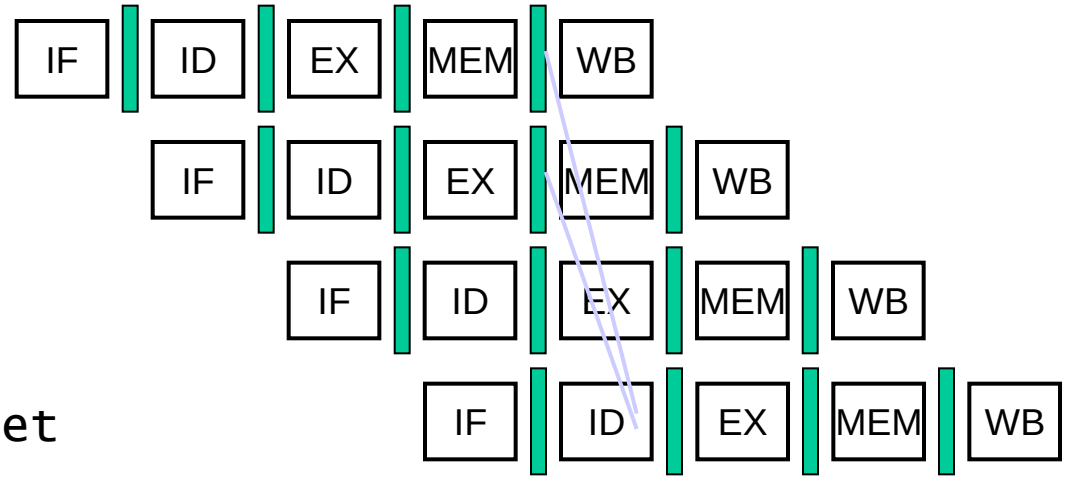
4.8 Branch Taken Example



4.8 Data Hazards for Branch (1)

If a _____ register is a _____ of 2nd or 3rd preceding ALU instruction

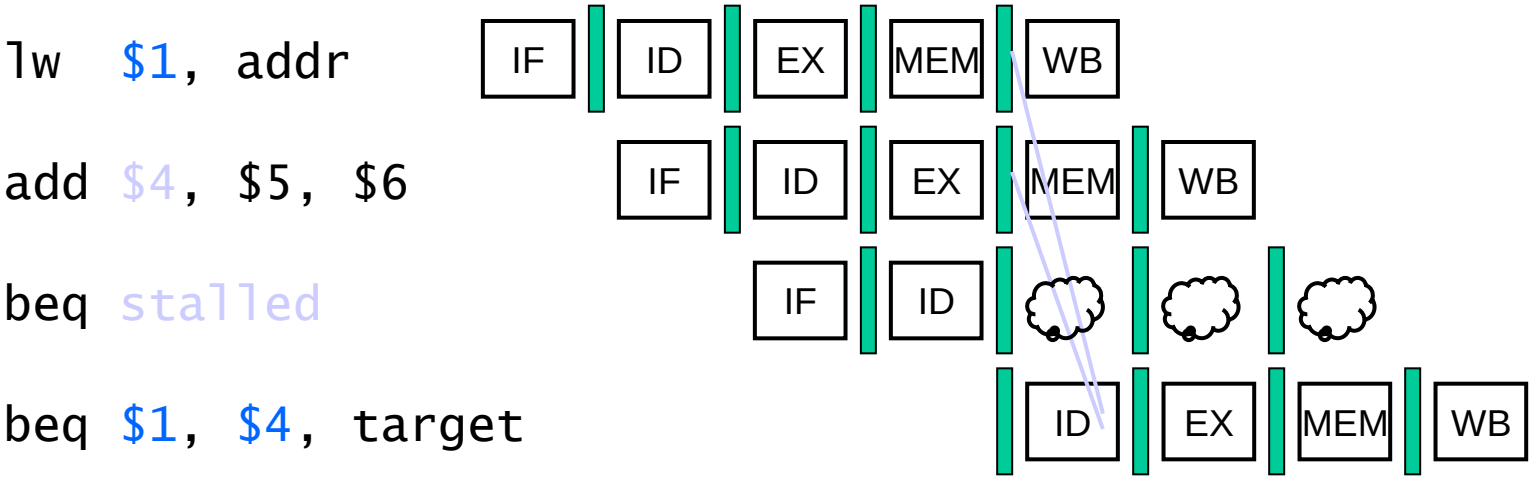
```
add $1, $2, $3
add $4, $5, $6
…
beq $1, $4, target
```



Can resolve using _____

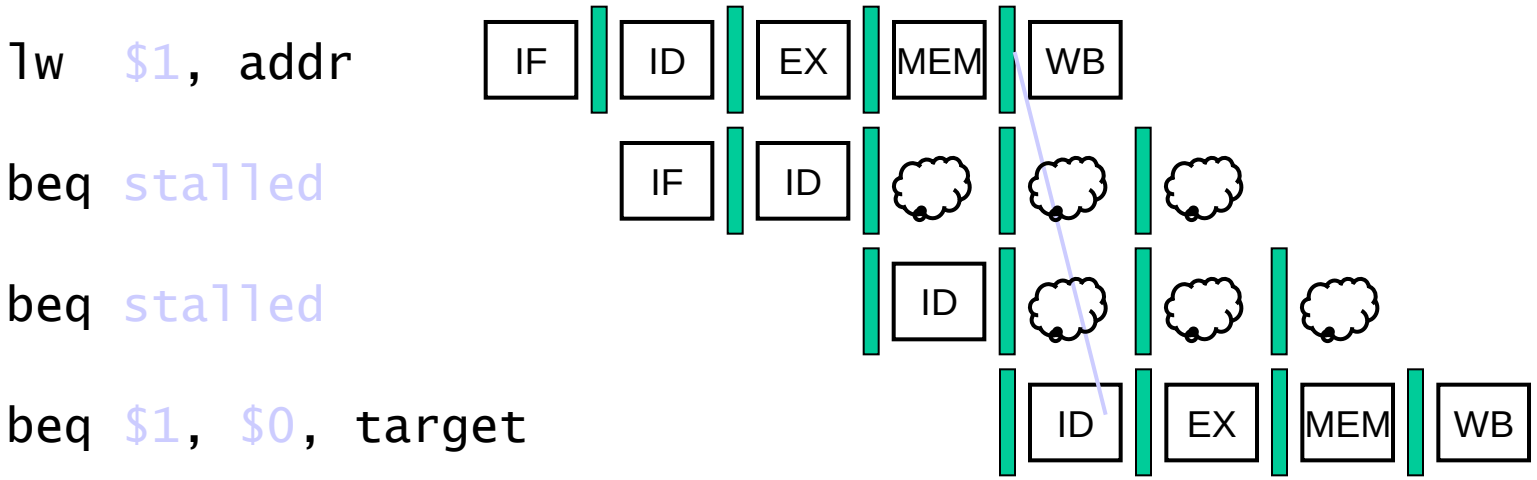
4.8 Data Hazards for Branches (2)

- If a comparison register is a destination of _____ ALU instruction or _____ instruction
  - Need _____



4.8 Data Hazards for Branches (3)

- If a comparison register is a destination of _____ _____ _____ instruction
  - Need _____-



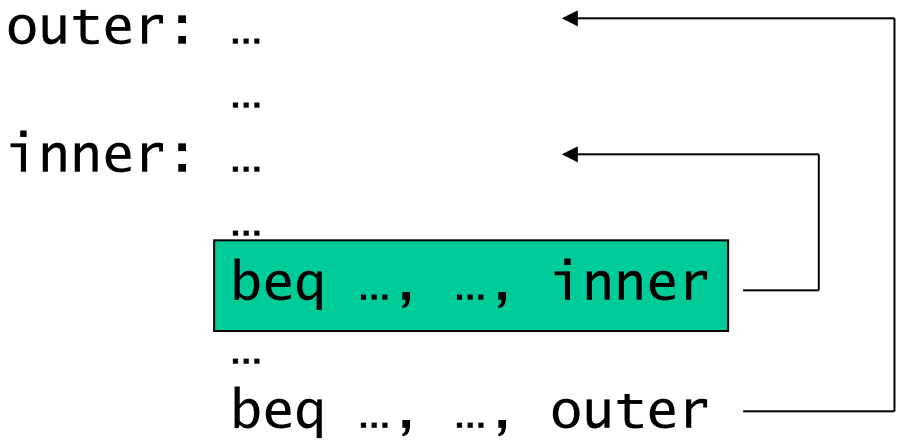
4.8 Dynamic Branch Prediction

- In _____ and _____ pipelines, branch _____ is more significant
- Use _____ prediction
  - Branch _____ table
  - _____ by _____ branch instruction addresses
  - Stores _____
  - To execute a branch
    - Check _____, expect the _____ outcome
    - Fetch from _____
    - Correct if necessary and _____ table

4.8 Shortcoming of 1-Bit History

- Mispredict as taken on ____ iteration of _____ loop
- Then mispredict as not taken on _____ iteration of _____ loop _____ _____

Outer Inner Predict Actual



- Inner loop branches mispredicted twice!

4.9 Exceptions and Interrupts

- _____ events requiring change in _____ of control
  - Different ISAs use the terms differently
- Exception
  - Arises _____, e.g., undefined opcode, overflow, syscall, …
- Interrupt
  - From an _____ I/O controller
- Dealing with them without sacrificing performance is hard

4.9 Handling Exceptions

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of _____(or _____) instruction
  - In MIPS: Exception Program Counter (EPC)
- Save indication of the problem
  - In MIPS: _____ register
  - We'll assume 1-bit, 0 for undefined opcode, 1 for overflow
- Jump to handler at _____

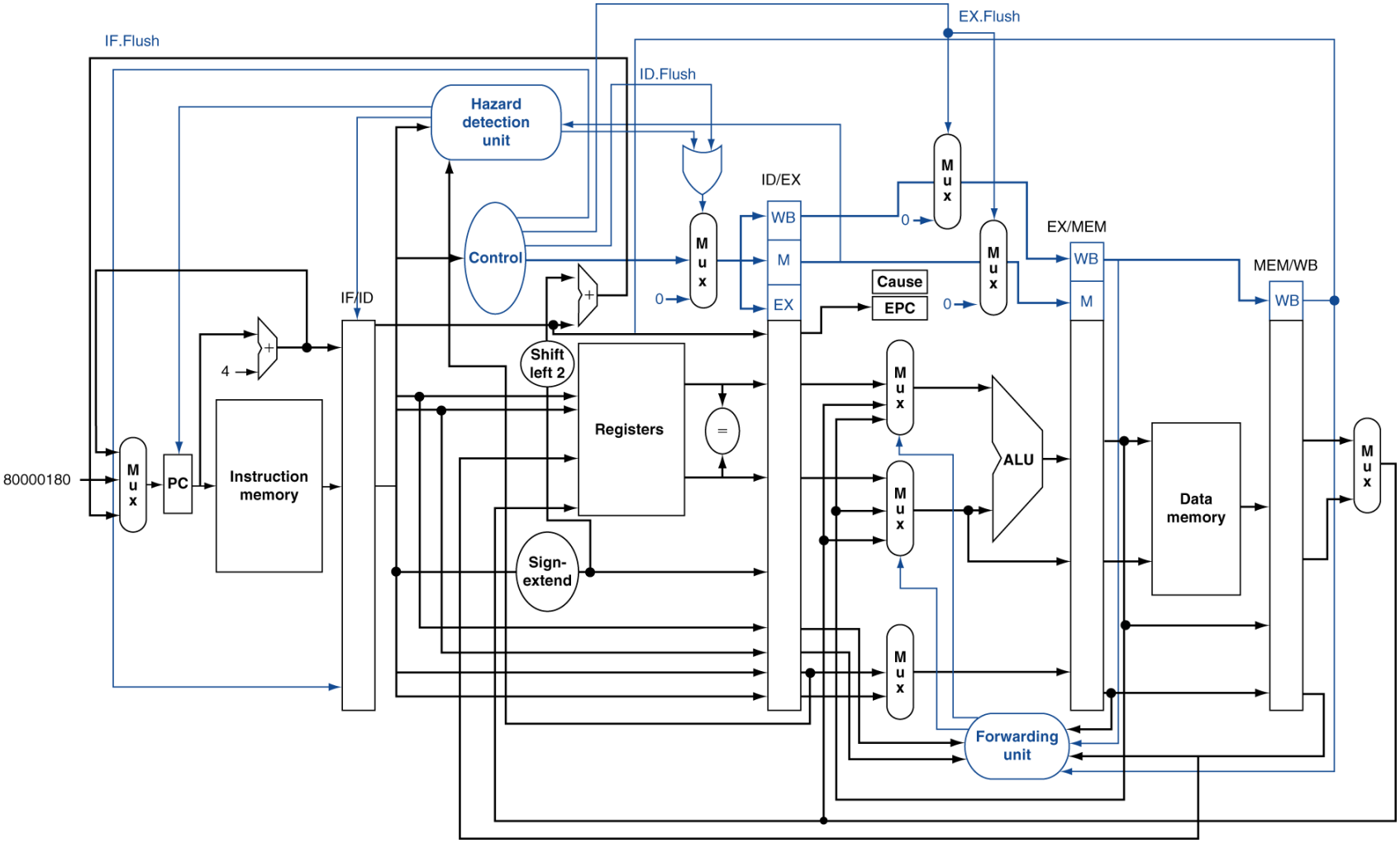
4.9 An Alternate Mechanism

- _____ Interrupts
  - Handler _____ determined by the _____
- Example:
  - Undefined opcode: C000 0000
  - Overflow: C000 0020
  - …: C000 0040
- Instructions __ _____ _____ either
  - _____ _____ the interrupt, or
  - _____ to _____ handler

4.9 Handler Actions

- ____ _____, and transfer to _____ handler
- If _____
  - Take corrective action
  - use _____ to return to _____
- Otherwise
  - _____ program
  - Report _____ using EPC, cause, …

4.9 Pipeline with Exceptions



4.9 Exception Properties

- _____ exceptions
  - Pipeline can ____ the instruction
  - Handler executes, then returns to the instruction
    - _____
- PC saved in ____ register
  - Identifies _____ instruction
  - Actually _____ is saved
    - _____ must adjust

4.9 Exception Example (1)

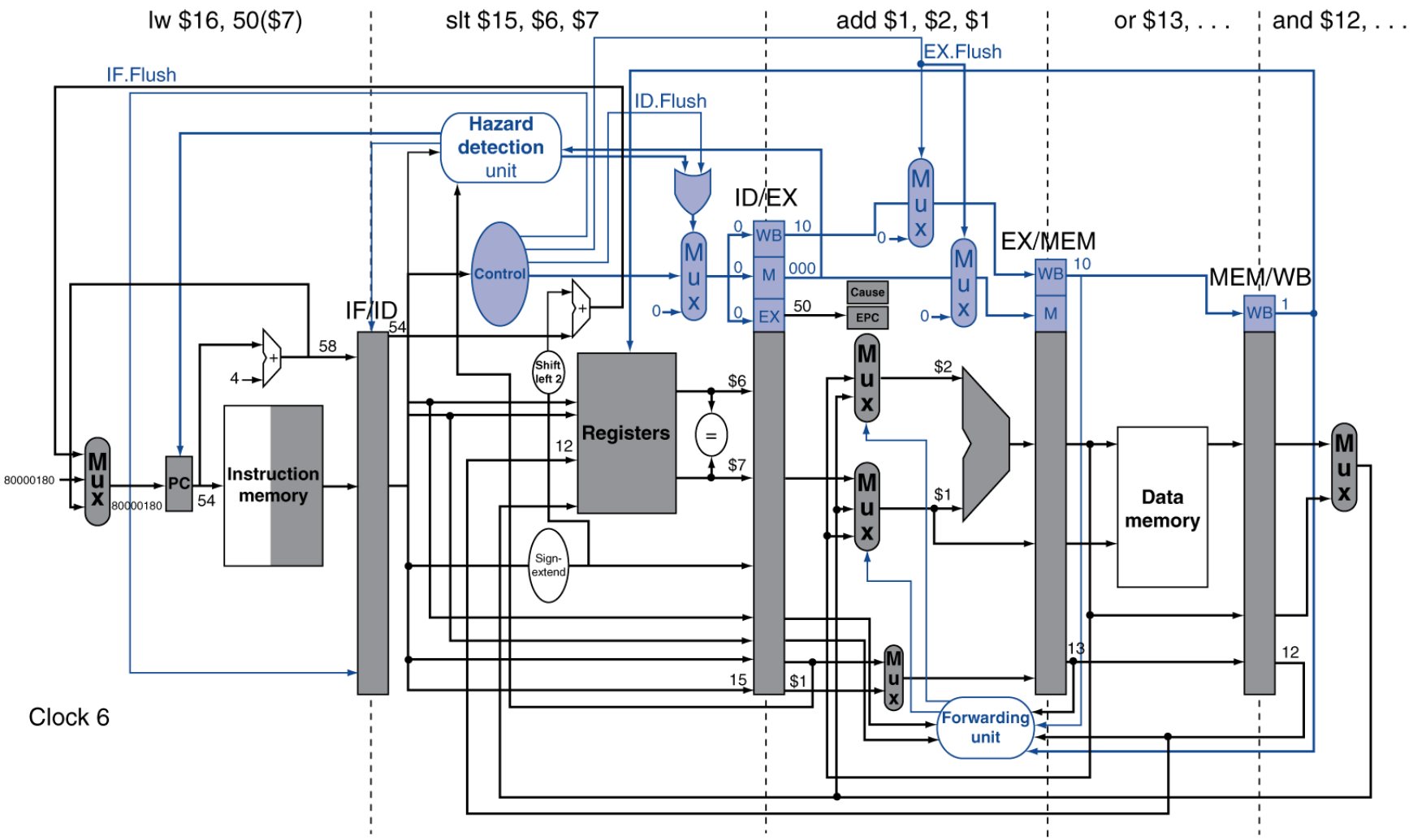
- Exception on `add` in

```
40      sub   $11, $2, $4
44      and   $12, $2, $5
48      or    $13, $2, $6
4C      add   $1,  $2, $1
50      slt   $15, $6, $7
54      lw    $16, 50($7)
…
```

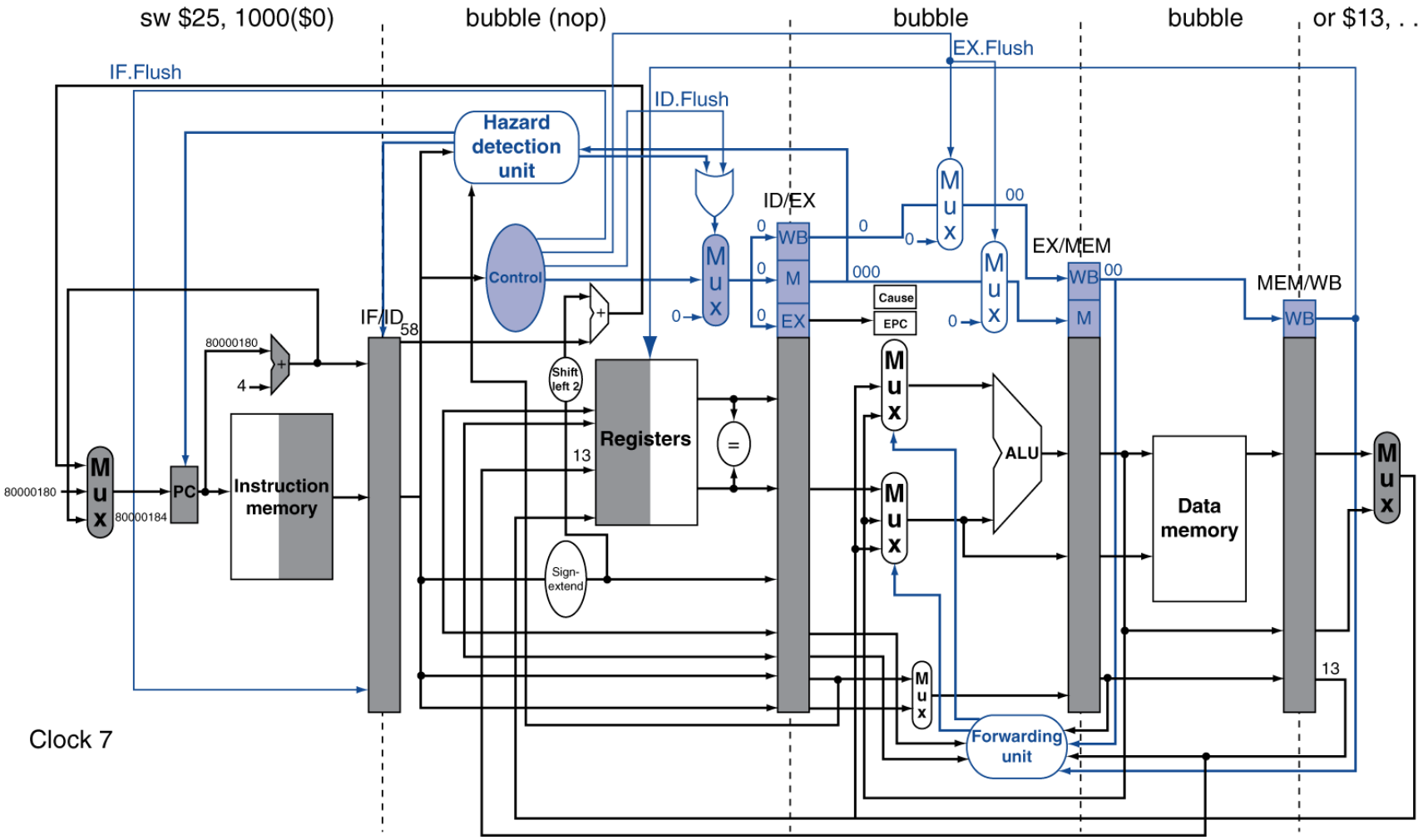
- Handler

```
80000180    sw    $25, 1000($0)
80000184    sw    $26, 1004($0)
```

4.9 Exception Example (2)



4.9 Exception Example (3)



4.9 Multiple Exceptions

- Pipelining _____ multiple instructions
  - Could have _____ exceptions __ _____
- Simple approach: deal with exception from _____ instruction
  - Flush _____ instructions
  - _____ exceptions
- In complex pipelines
  - _____ instructions issued ___ _____
  - _____ completion
  - Maintaining _____ exceptions is _____!

4.9 Imprecise Exceptions

- Just _____ pipeline and ____ _____, including exception cause(s)
- Let the handler work out
  - _____
  - _____
- Simplifies _____, but _____ _____ _____ software
- Not feasible for complex multiple-issue out-of-order pipelines

4.10 Parallelism via Instructions

- Pipelining: executing multiple instructions in parallel
- To increase ILP (_____)
  - Deeper pipeline
    - Less work per stage ⇒ shorter clock cycle
  - Multiple _____
    - _____ pipeline stages ⇒ _____ pipelines
    - Start _____ instructions per clock cycle
    - CPI < 1, so use _____
    - E.g., 4GHz 4-way multiple-issue, 16 BIPS, peak CPI = 0.25, peak ____ = 4
    - But _____ reduce this in practice

4.10 Multiple Issue

- _____ multiple issue
  - _____ groups instructions to be _____ together
  - _____ them into _____ _____
  - _____ detects and _____ hazards
- _____ multiple issue
  - _____ examines instruction stream and _____ _____ to issue _____ cycle
  - _____ can help by _____ instructions
  - CPU _____ hazards using advanced techniques at _____

4.10 Speculation

- "Guess" what to do with an instruction
  - Start operation as soon as possible
  - _____ whether guess was right
    - If so, _____ the operation
    - If not, _____ and do the right thing
- _____ to static and dynamic multiple issue
- Examples
  - Speculate on branch outcome
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated

4.10 Compiler/Hardware Speculation

- Compiler can _____ instructions
  - e.g., move _____ before _____
  - Can include _____ instructions to _____ from _____ guess
- Hardware can ____ _____ for instructions to execute
  - _____ results until it determines they are _____ _____
  - _____ buffers on _____ speculation

4.10 Static Multiple Issue

- _____ groups instructions into _____ _____
  - _____ of instructions that can be issued on a _____ cycle
  - Determined by _____ _____ required
- Think of an _____ _____ as a very long instruction
  - Specifies _____ _____ operations
  - ⇒ _____ (VLIW)

4.10 Scheduling Static Multiple Issue

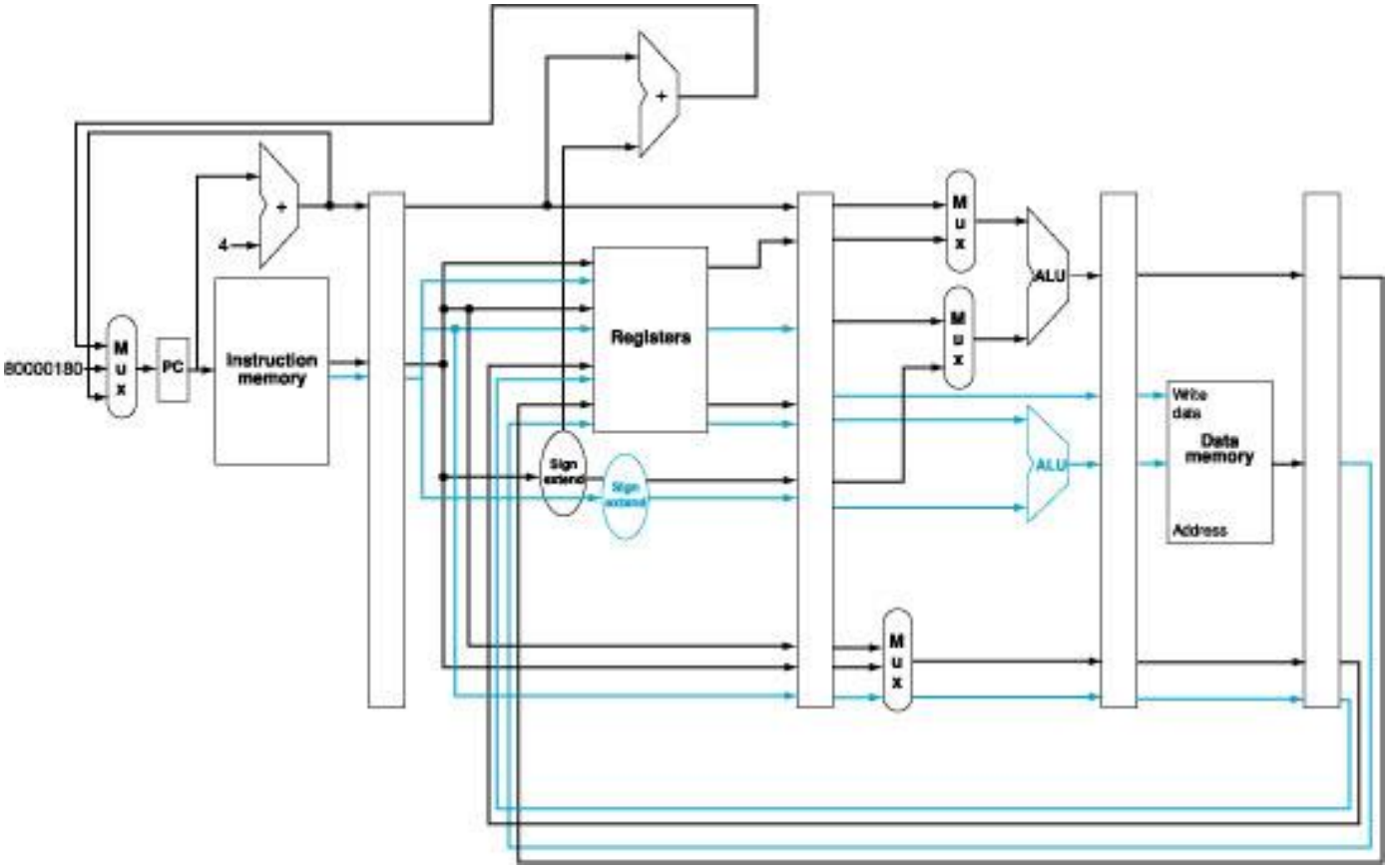
- Compiler must remove _____ hazards
  - _____ instructions into issue packets
  - No dependencies _____ a packet
  - Possibly some dependencies _____ packets
    - _____; compiler must know!
  - Pad with _____ if necessary

4.10 MIPS Static Dual Issue (1)

- Two-issue packets
  - One _____ instruction
  - One _____ instruction

Instruction type	Pipeline Stages						
ALU/branch	IF	ID	EX	MEM	WB		
Load/store	IF	ID	EX	MEM	WB		
ALU/branch		IF	ID	EX	MEM	WB	
Load/store		IF	ID	EX	MEM	WB	
ALU/branch			IF	ID	EX	MEM	WB
Load/store			IF	ID	EX	MEM	WB

4.10 MIPS Static Dual Issue (2)



4.10 Scheduling Example

```
Loop: lw   $t0, 0($s1)       # $t0=array element
      addu $t0, $t0, $s2     # add scalar in $s2
      sw   $t0, 0($s1)       # store result
      addi $s1, $s1,-4       # decrement pointer
      bne  $s1, $zero, Loop  # branch $s1!=0
```

	ALU/branch	Load/store	cycle
Loop:			1
			2
			3
			4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

4.10 Loop Unrolling (1)

```
Loop:
```

4.10 Loop Unrolling (2)

```
Loop:
```

4.10 Loop Unrolling (3)

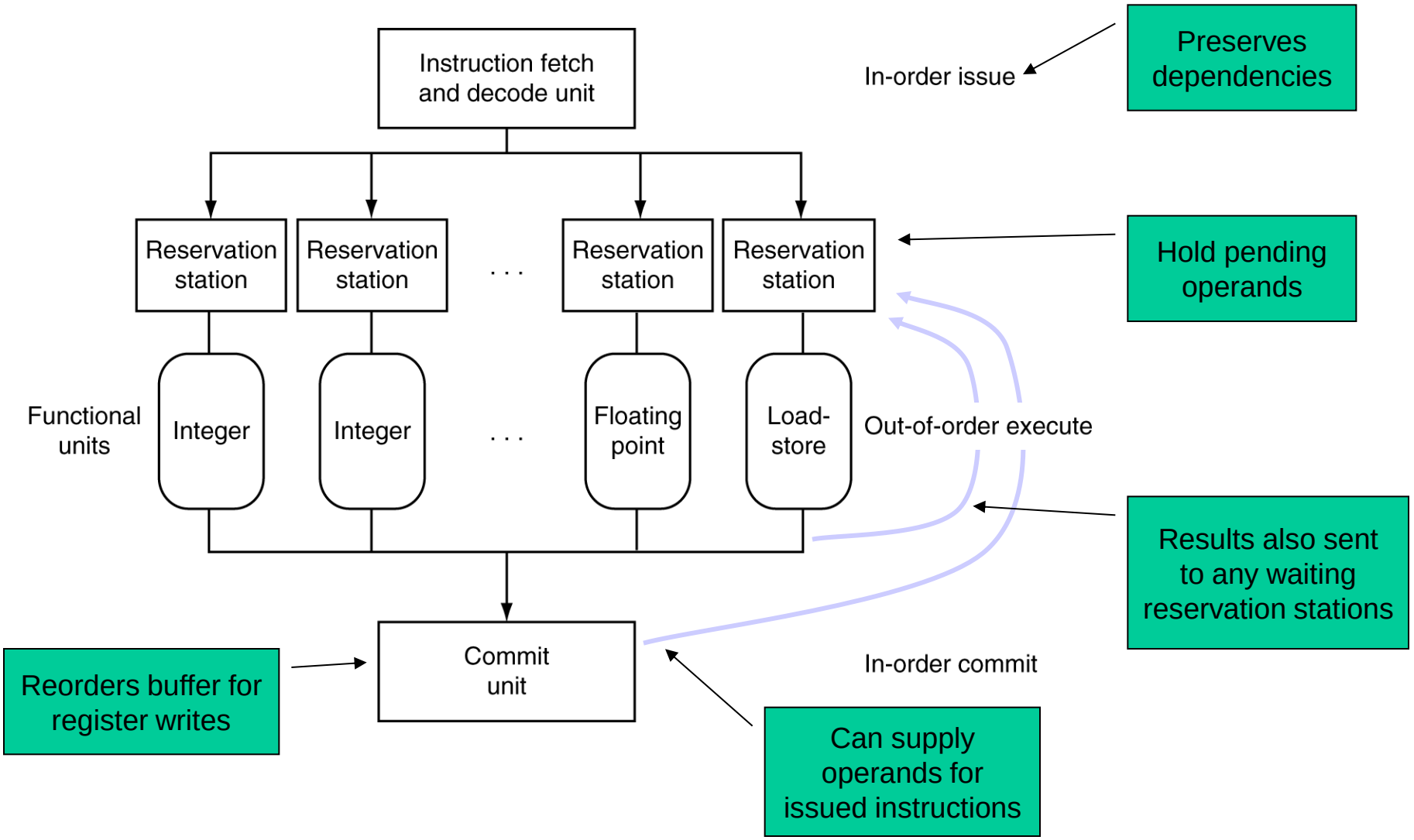
```
Loop:
```

4.10 Scheduling Unrolled Loop

- Replicate loop body to expose more parallelism
  - Reduces loop-control overhead
- Use different registers per replication
  - Called “register renaming”

	ALU/branch	Load/store	cycle
Loop:			1
			2
			3
			4
			5
			6
			7
			8

4.10 Dynamically Scheduled CPU



4.10 Speculation

- _____ branch and continue _____
  - Don't _____ until branch _____ determined
- _____ speculation
  - Avoid load and _____ ____ delay
    - Predict the _____ address
    - Predict _____ value
    - Load before completing _____ stores
    - _____ stored values to load unit
  - Don't _____ load until _____ cleared

4.10 Does Multiple Issue Work?

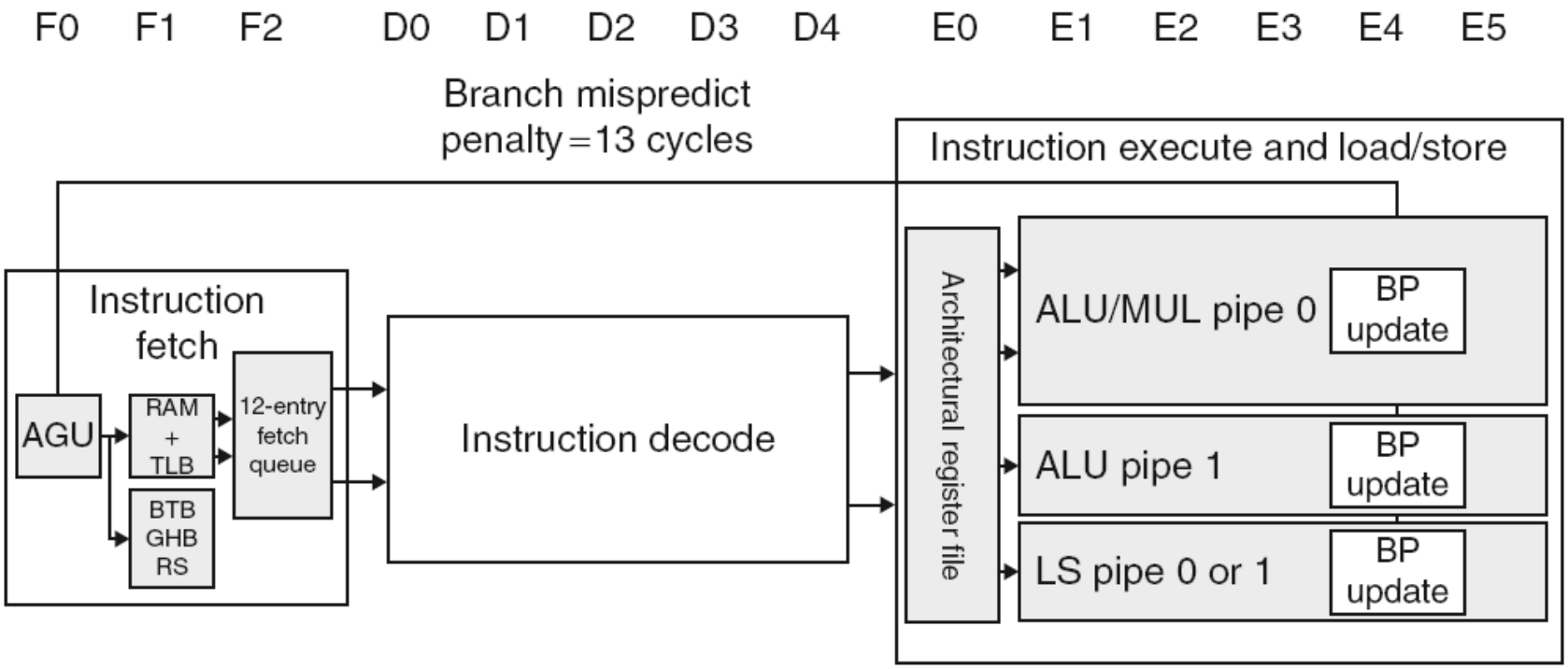
- Yes, but not as much as we'd like
- Programs have real _____ that limit ILP
- Some _____ are hard to _____
  - _____
- Some _____ is hard to expose
  - Limited _____ ____during instruction issue
- Memory delays and _____ _____
  - Hard to keep pipelines ____
- _____ can help if done well

4.10 Power Efficiency

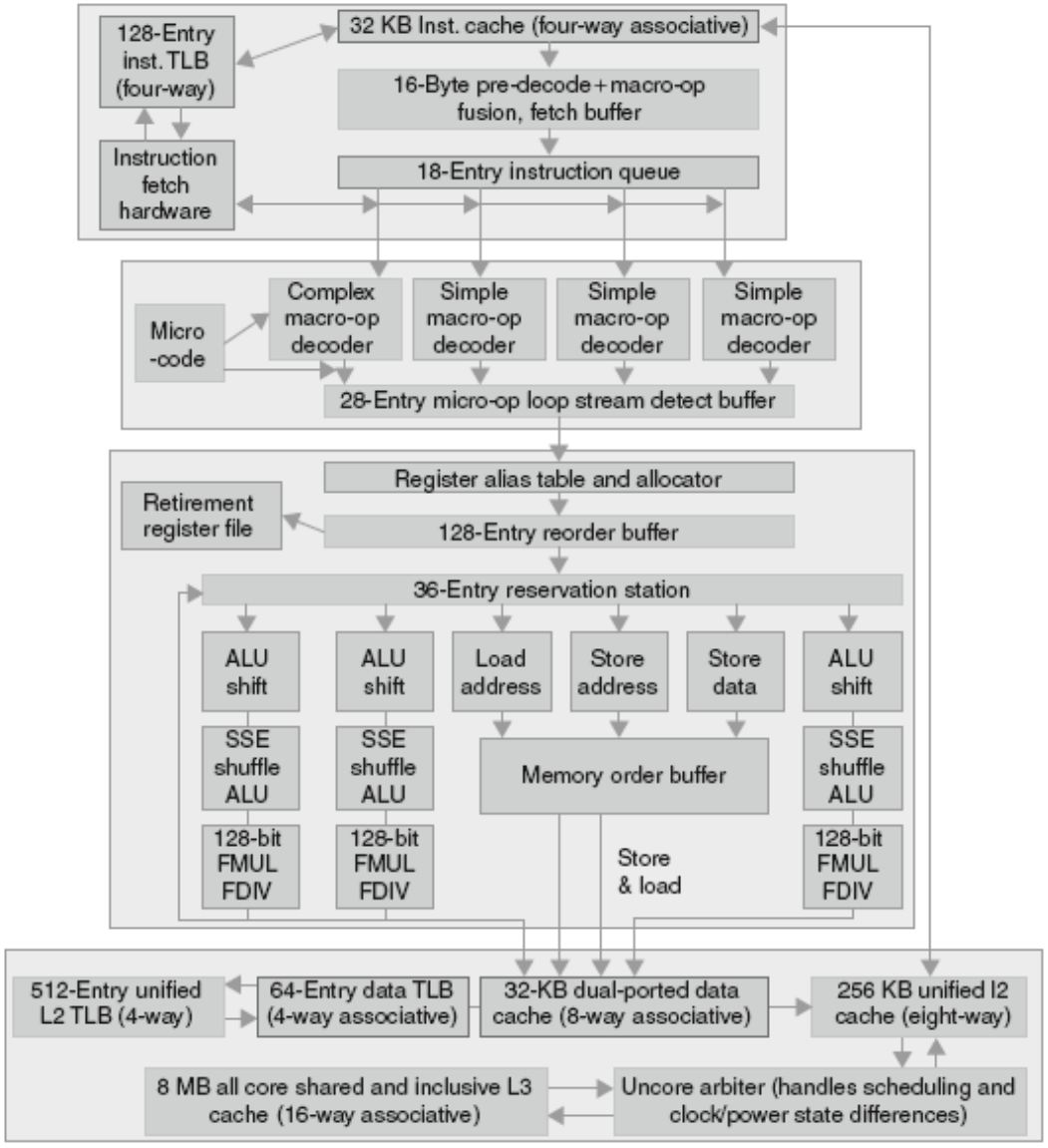
- Complexity of dynamic scheduling and speculations requires power
- Multiple simpler cores may be better

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

4.11 ARM Cortex A8 Pipeline



4.11 Core i7 Pipeline



4.15 Concluding Remarks

- ____ influences design of _____ and _____
- _____ and _____influence design of ___
- Pipelining improves _____ _____ using parallelism
  - More _____ completed ___ _____
  - _____ for each instruction ___ _____
- Hazards: _____, _____, _____
- Multiple issue and dynamic scheduling (ILP)
  - _____ limit achievable parallelism
  - _____leads to the power wall