CSEN 702

Microprocessors

Lecture 2

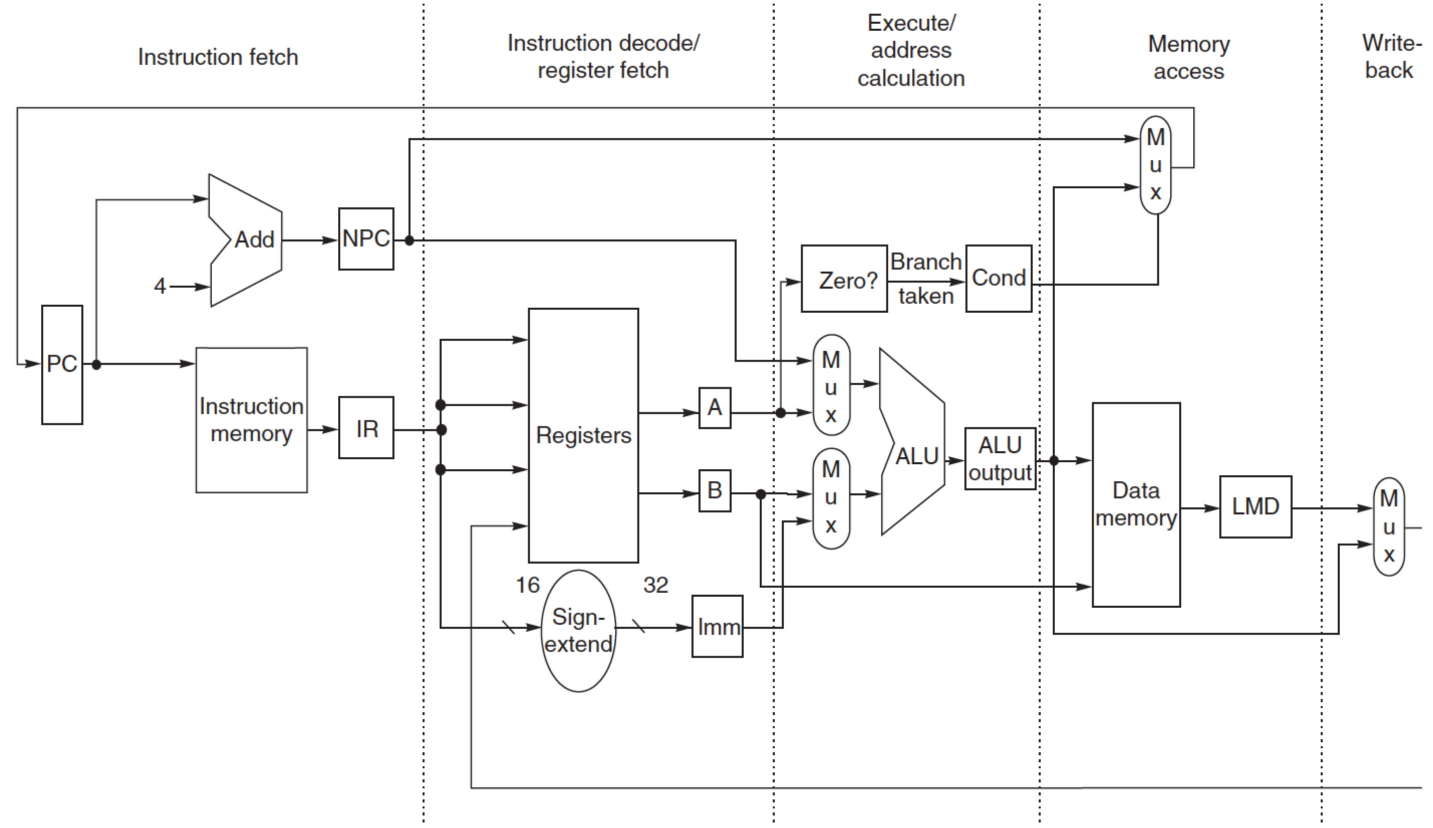
Review on MIPS64 pipelining and floating point (part 1)

Reading Textbook Chapter 1.1, 1.2, 1.3, Appendix A, Appendix C

Part 1 Pipelining

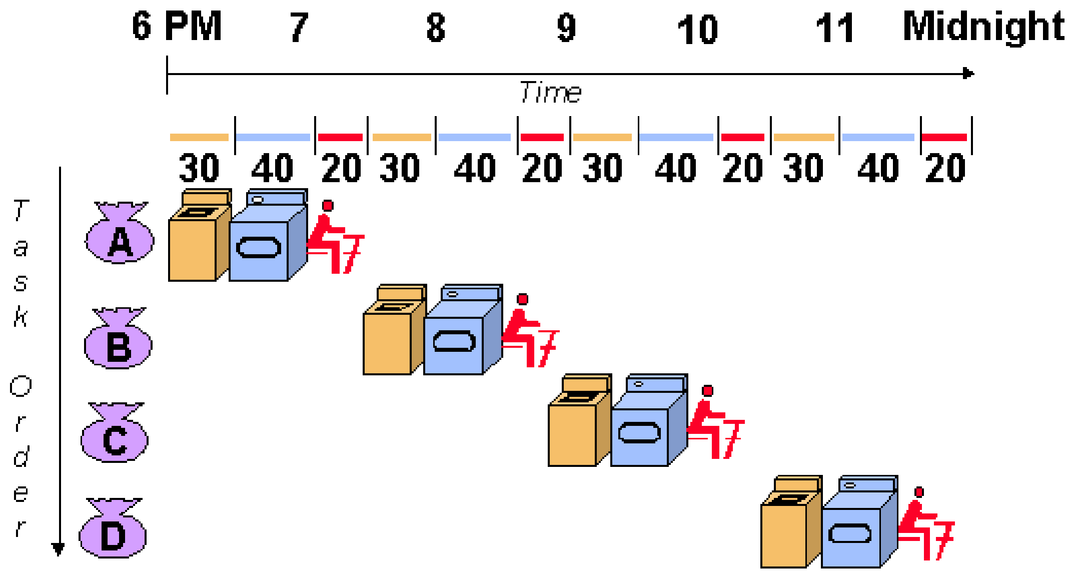
Normal datapath from lecture 1

- Once an instruction is being executed, all the blocks in the architecture are actually working for that instruction.
- Some of them finish the job early and remain unnecessary (idle) for the rest of the time

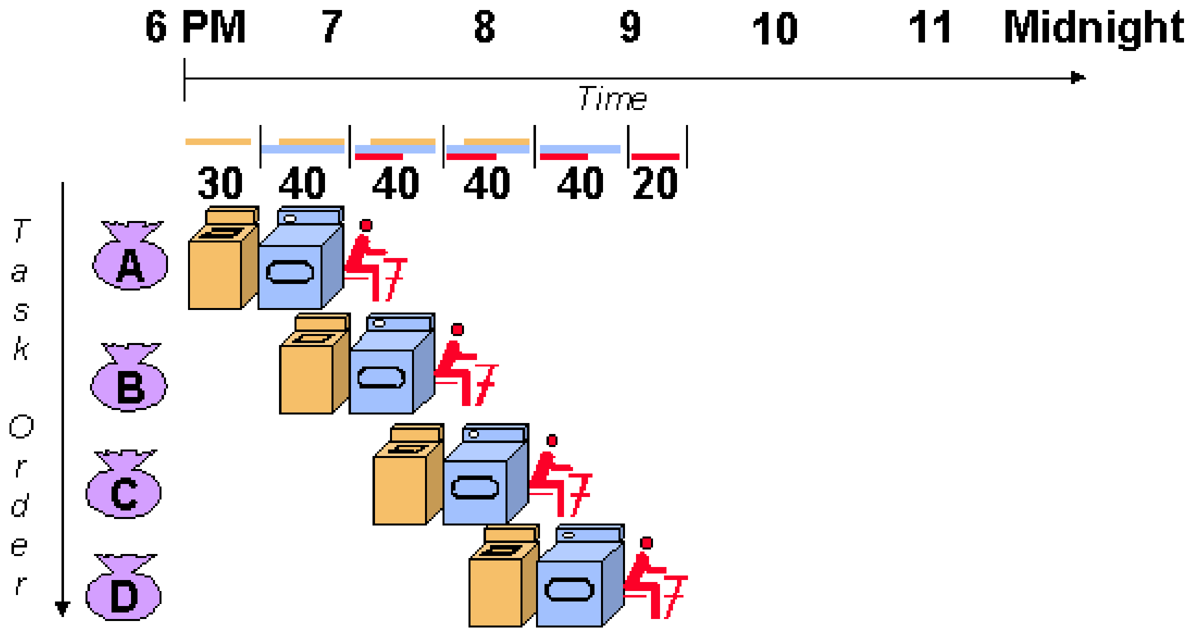


The concept of Pipelining

- Assume we have 4 people A, B, C, and D wanting to wash their clothes at a laundromat.
- Washing takes 30 min, drying takes 40 min, and folding takes 20 min.



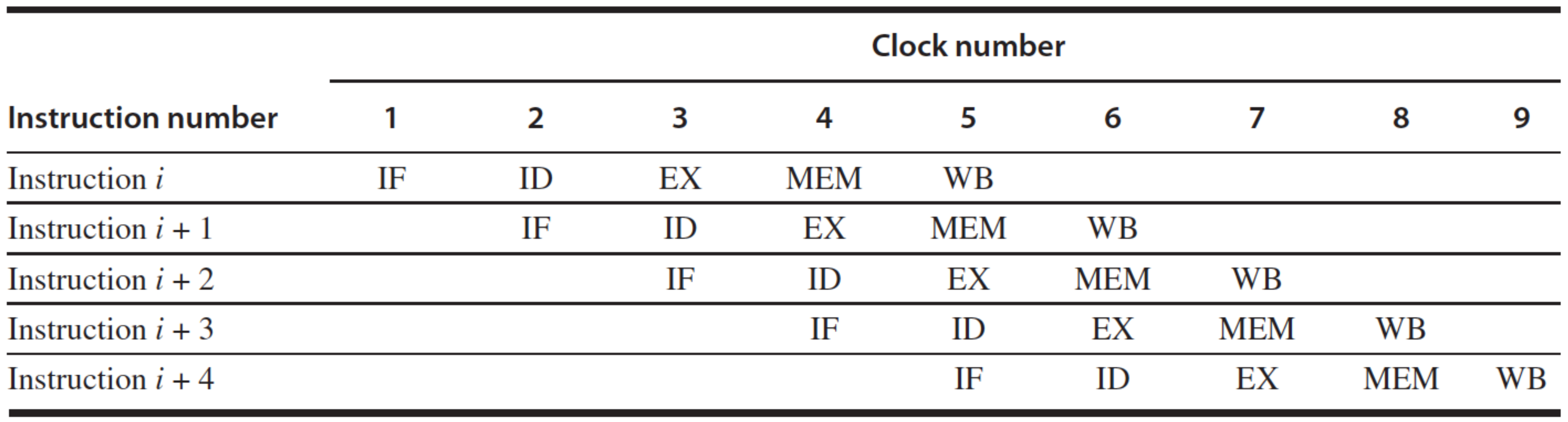
Un-pipelined process



Pipelined process

Pipelining the processor

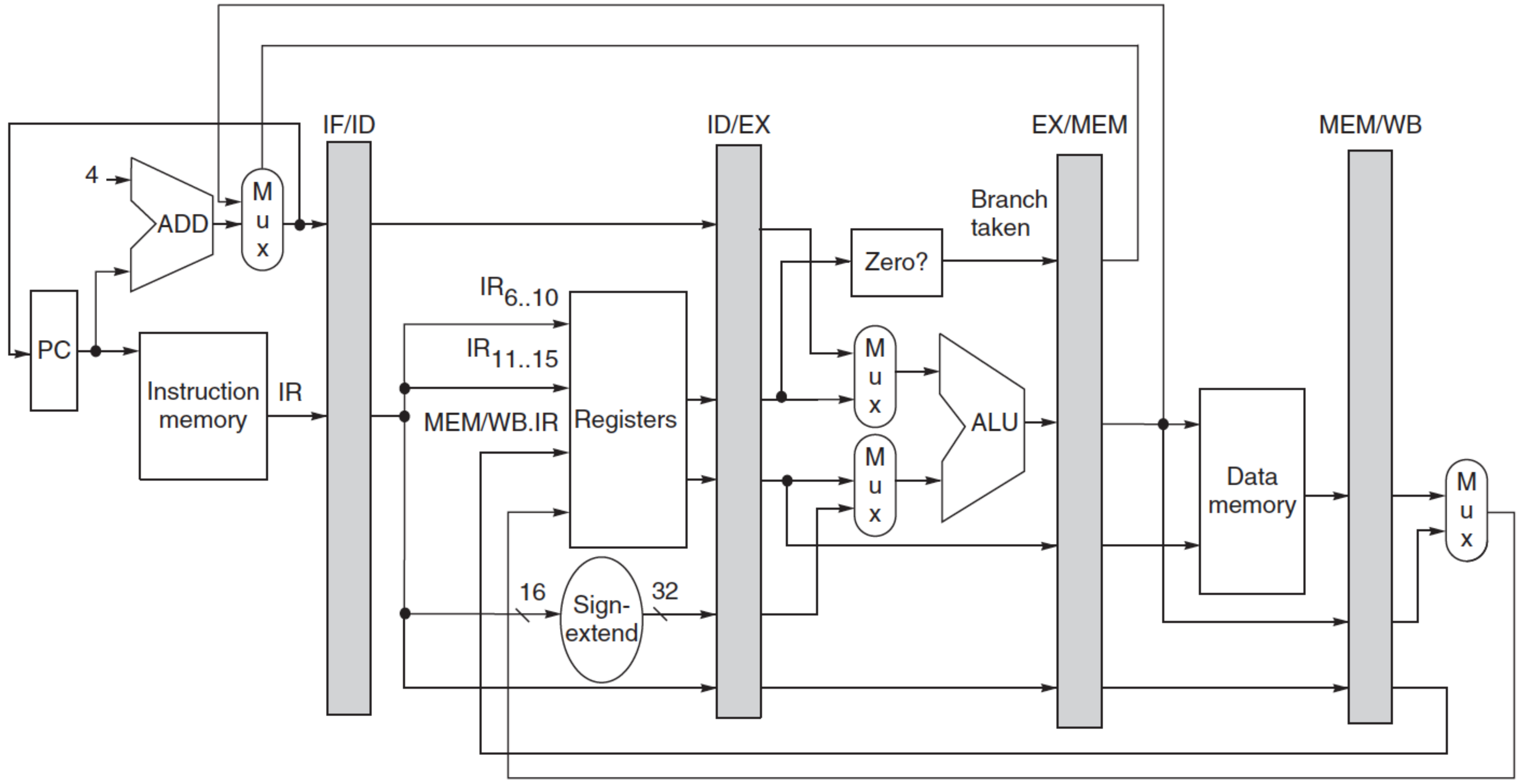
- The idea of splitting the job into tasks, and execute the tasks in parallel to speed up the overall process.



	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction i	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB

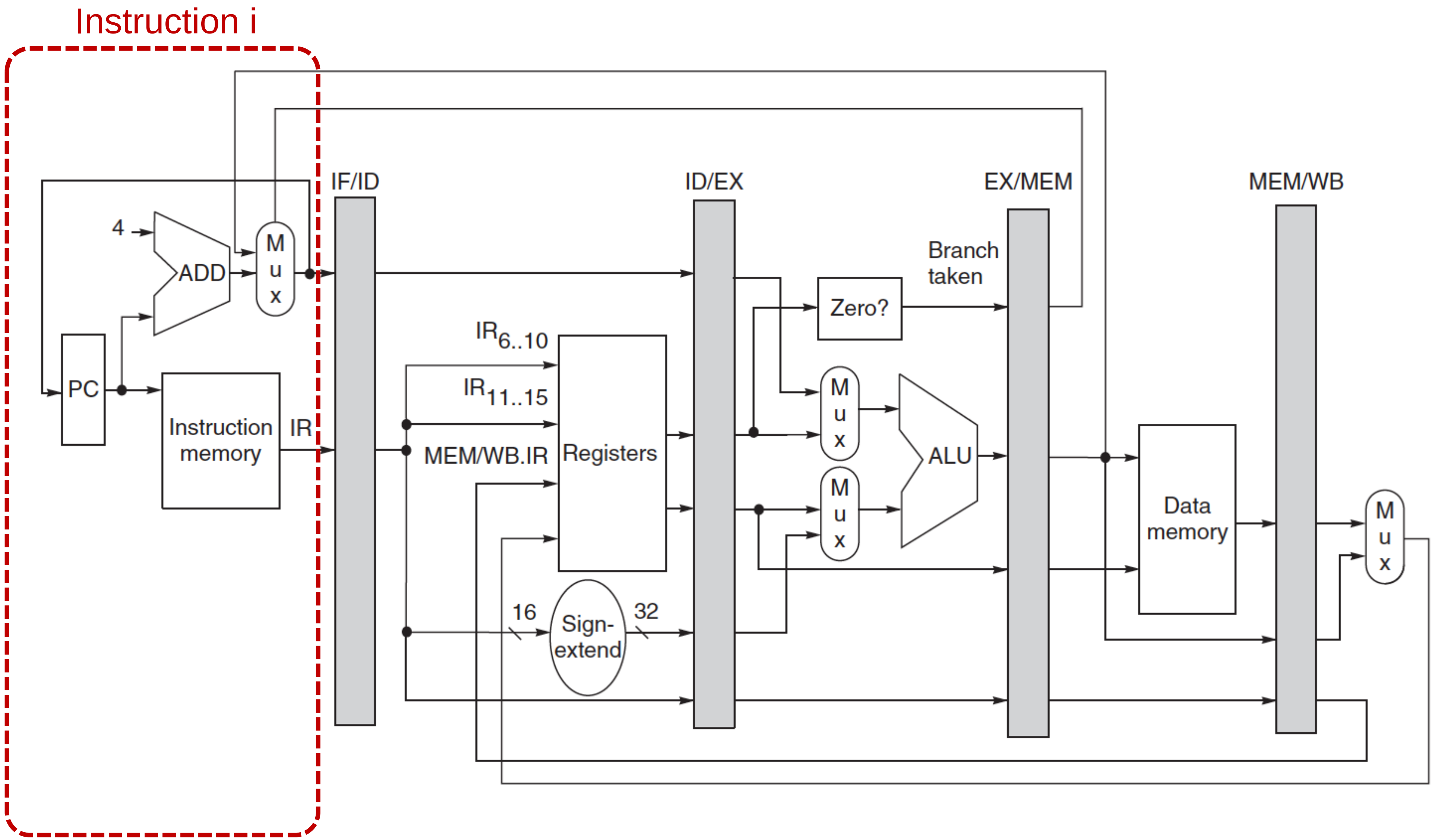
- So, while instruction i is being decoded, instruction i+1 is fetched.
- Or when instruction i is executing, i+1 will be decoded, i+2 is being fetched and so on. → **we need registers to hold intermediate results!**

5-stage pipeline

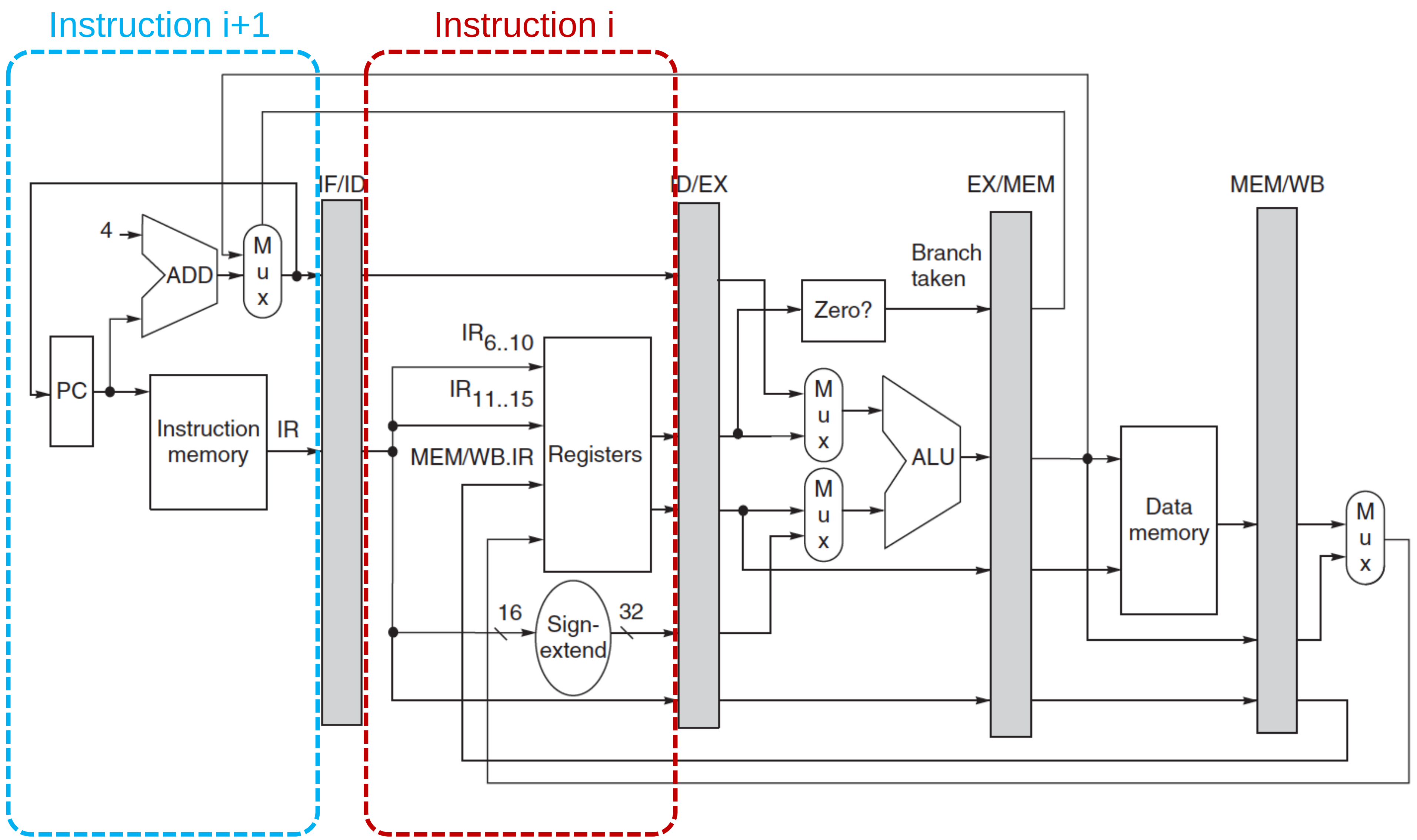


- we use separate instruction and data memories, which we would typically implement with separate instruction and data caches
- To handle reads and a write to the same register, we perform the register write in the first half of the clock cycle and the read in the second half.

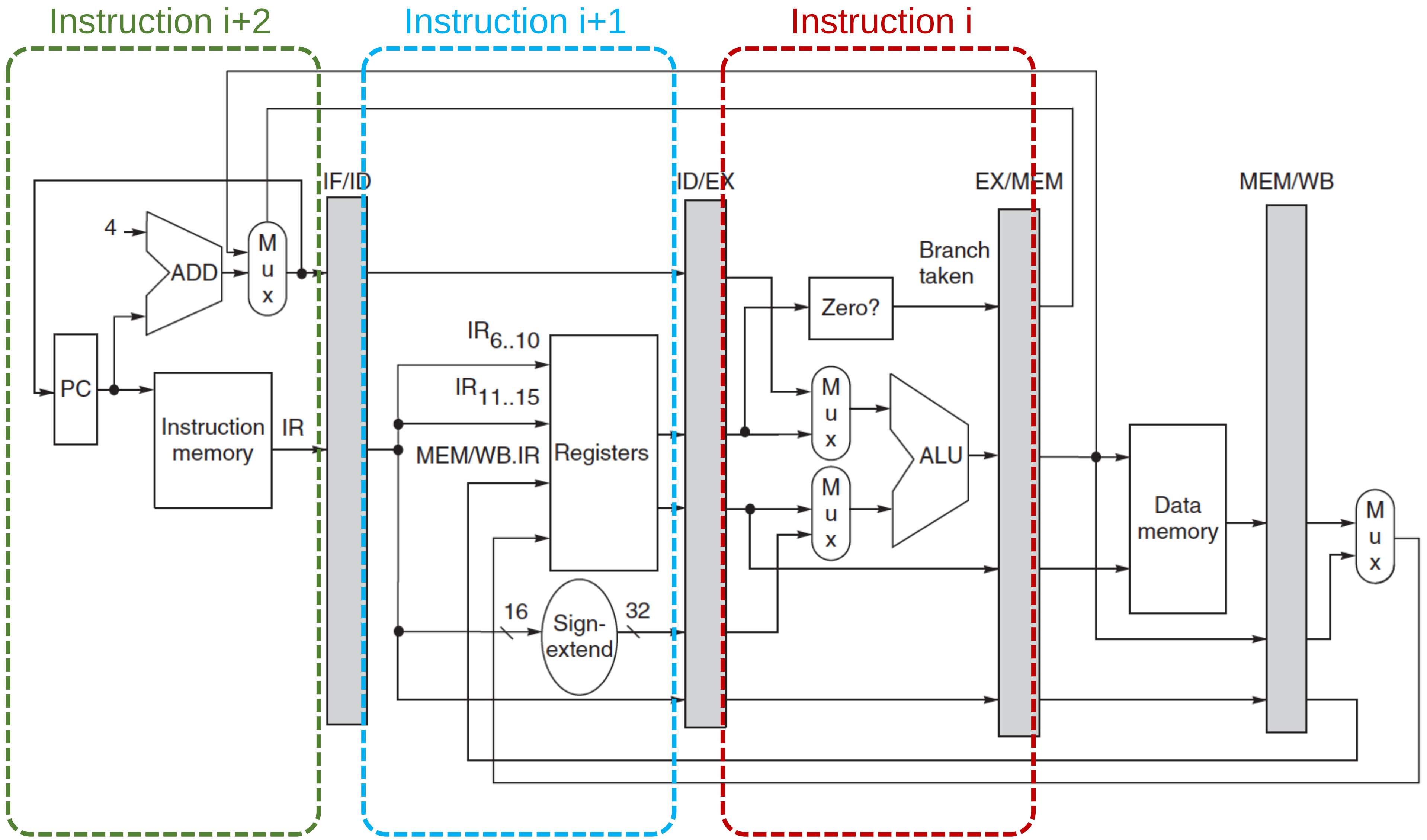
Pipelining in action: Cycle 1



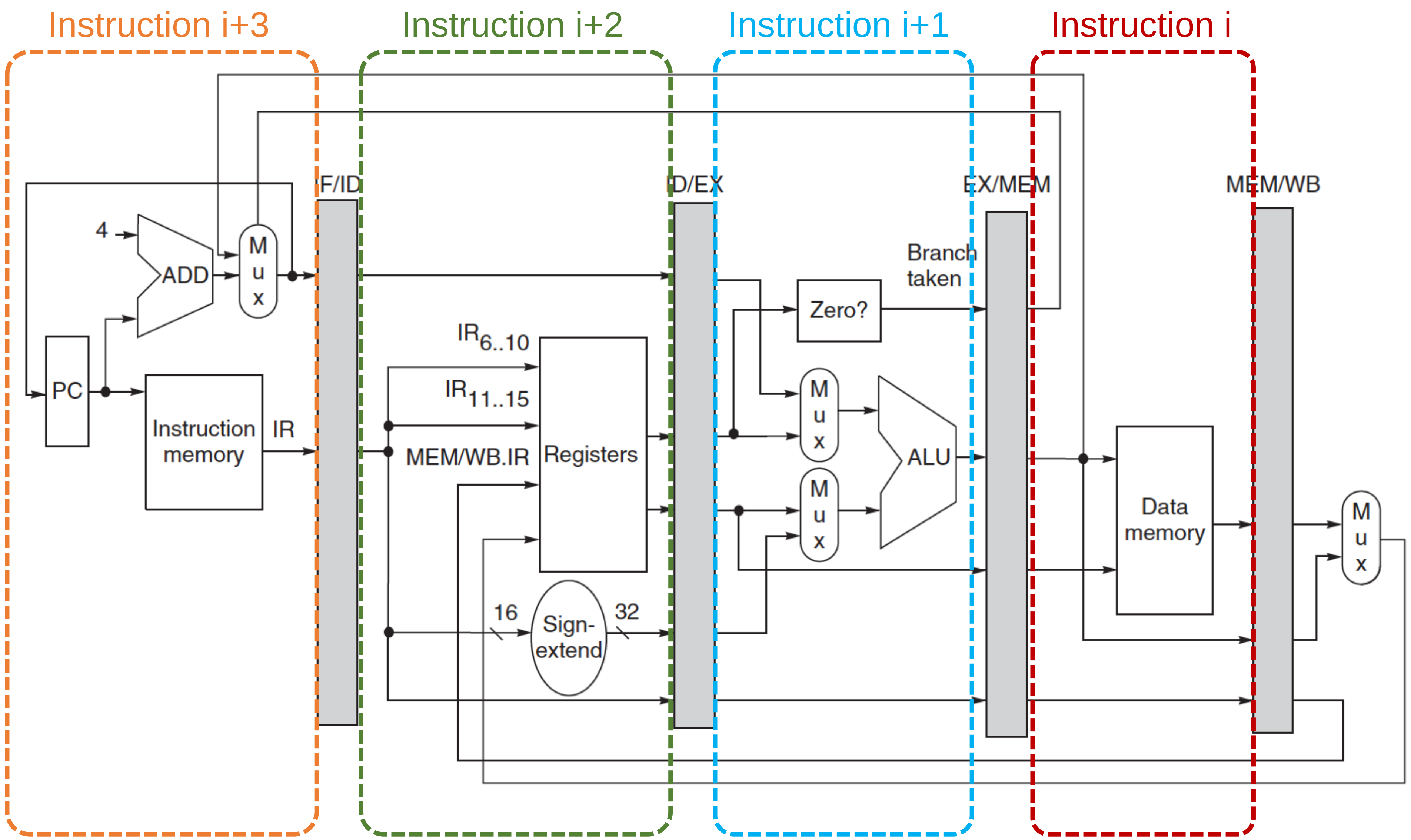
Pipelining in action: Cycle 2



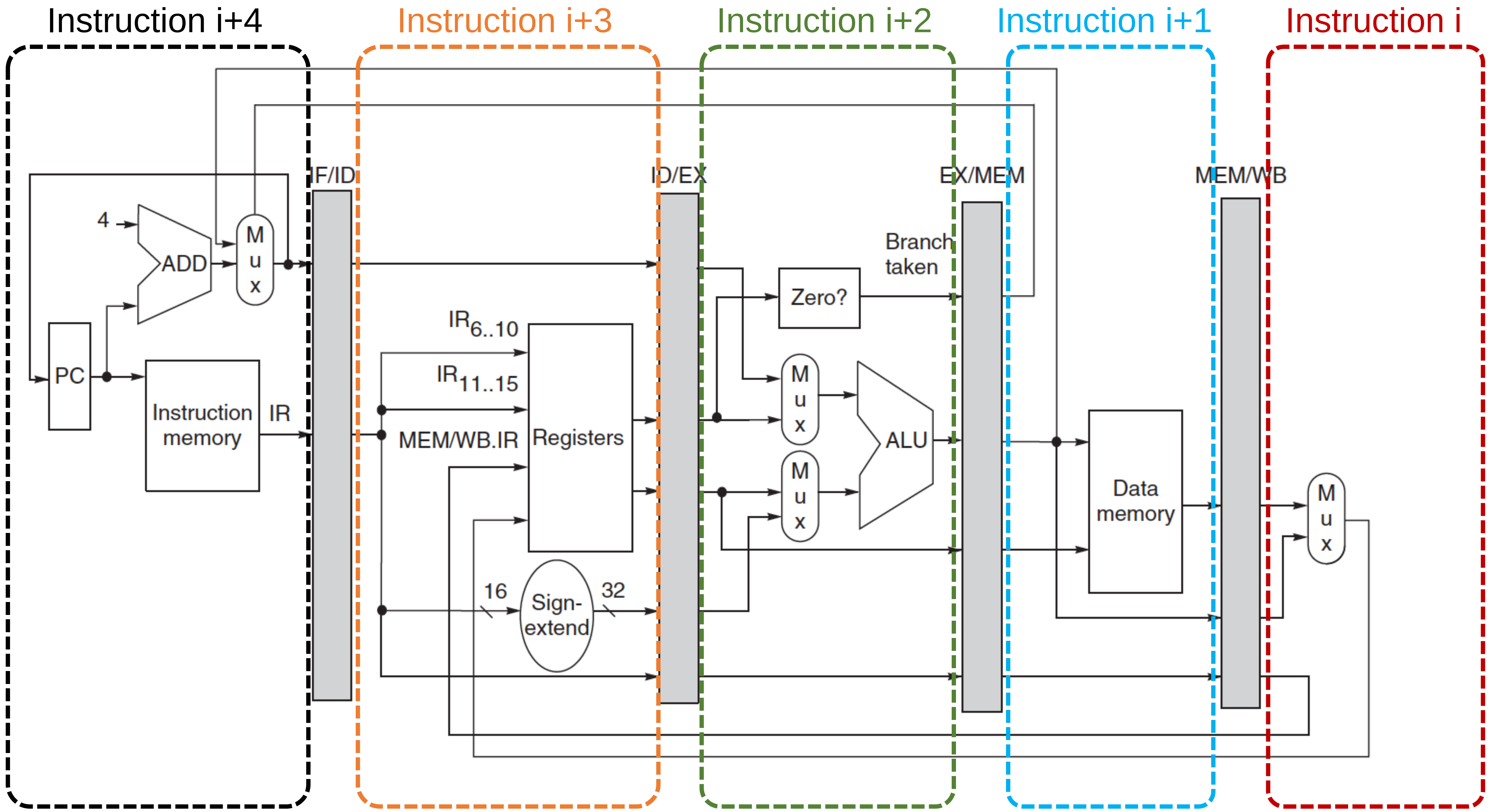
Pipelining in action: Cycle 3



Pipelining in action: Cycle 4



Pipelining in action: Cycle 4



Timing

- If the stages are perfectly balanced, then **the time per instruction** on the pipelined processor—assuming ideal conditions—is equal to:

$$\text{Time per instruction} = \boxed{\frac{\text{Time per instruction on unpipelined machine}}{\text{Number of pipe stages}}}$$

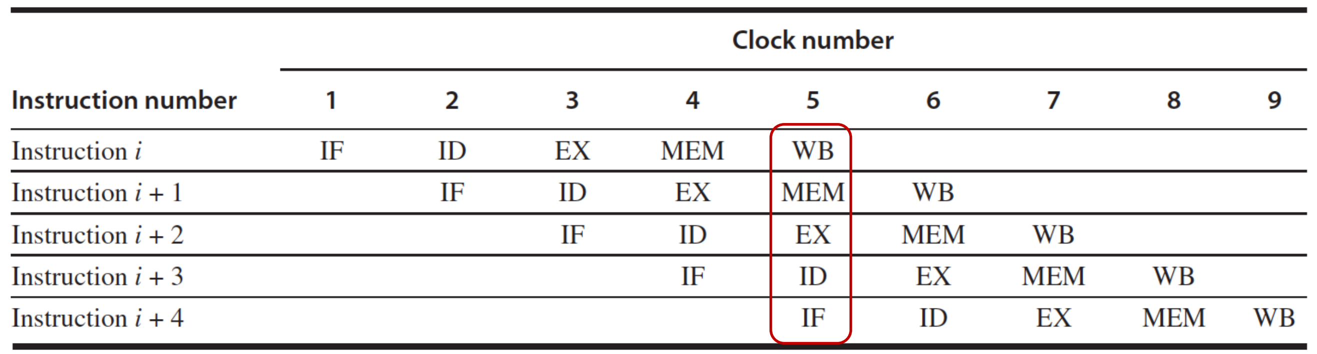
- **Goal**: Reduce CPI (Clock cycles per instruction) **or** clock cycle time
- Two approaches:
  - **A)** If we consider the un-pipelined version to execute an instruction in multiple clock cycles, then the goal is to reduce the CPI.
  - **B)** If we consider the un-pipelined version to execute an instruction in one long clock cycle, then the goal is reduce the clock cycle time.

Timing (2)

- Pipelining increases the CPU instruction throughput—the number of instructions completed per unit of time—but it does not reduce the execution time of an individual instruction

- Imbalance among the pipe stages reduces performance since the clock can run no faster than the time needed for the **slowest pipeline stage**.

- Pipeline overhead arises from the combination of pipeline register delay and clock skew.



	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

IDEAL CPI=1

Why?

→ If you look vertically in clock cycle 5, you notice that all stages are executed in 1 clock cycle. even if each stage from a different instruction, but overall, it's as if we are taking 1 clock cycle to execute an entire instruction.

→ In other words, we are able to finish an instruction every clock cycle.

Exercise

- Consider the un-pipelined processor that has a 1-ns clock cycle and that it uses:
- 4 cycles for ALU operations and branches
- 5 cycles for memory operations.
- Assume that the relative frequencies of these operations are 40%, 20%, and 40%, respectively.

- Suppose that due to clock skew and setup, pipelining the processor adds 0.2 ns of overhead to the clock.
- **Question: How much speedup in the instruction execution rate will we gain from a pipeline?**
- **Hint: calculate**
  speedup = (execution time per instruction unpipelined) / (execution time per instruction pipelined)

Exercise solution

- Consider the un-pipelined processor that has a 1-ns clock cycle and that it uses 4 cycles for ALU operations and branches and 5 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20%, and 40%, respectively.
- Suppose that due to clock skew and setup, pipelining the processor adds 0.2 ns of overhead to the clock.

Solution

Un-pipelined average instruction execution time = Clock cycle time x Average CPI
= 1 ns x [(40% + 20%) x 4 + 40% x 5]
= 1 ns x 4.4
= 4.4 ns

Pipelined average instruction execution time = Clock cycle time x Average CPI
= (1+0.2) ns x 1 ← **remember that CPI of pipelined = 1**
= 1.2 ns

Speedup = (avg. inst. time un-pipelined)/ (avg. inst. time pipelined)
= 4.4/1.2 = **3.7 times**

Pipeline hazards

- There are three classes of hazards:
    1. **Structural hazards** arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution.
    2. **Data hazards** arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.
    3. **Control hazards** arise from the pipelining of branches and other instructions that change the PC.

Solution:

- Stall the pipeline!
- Also all instructions issued after the stalled instruction, are also stalled.

1- Structural hazards

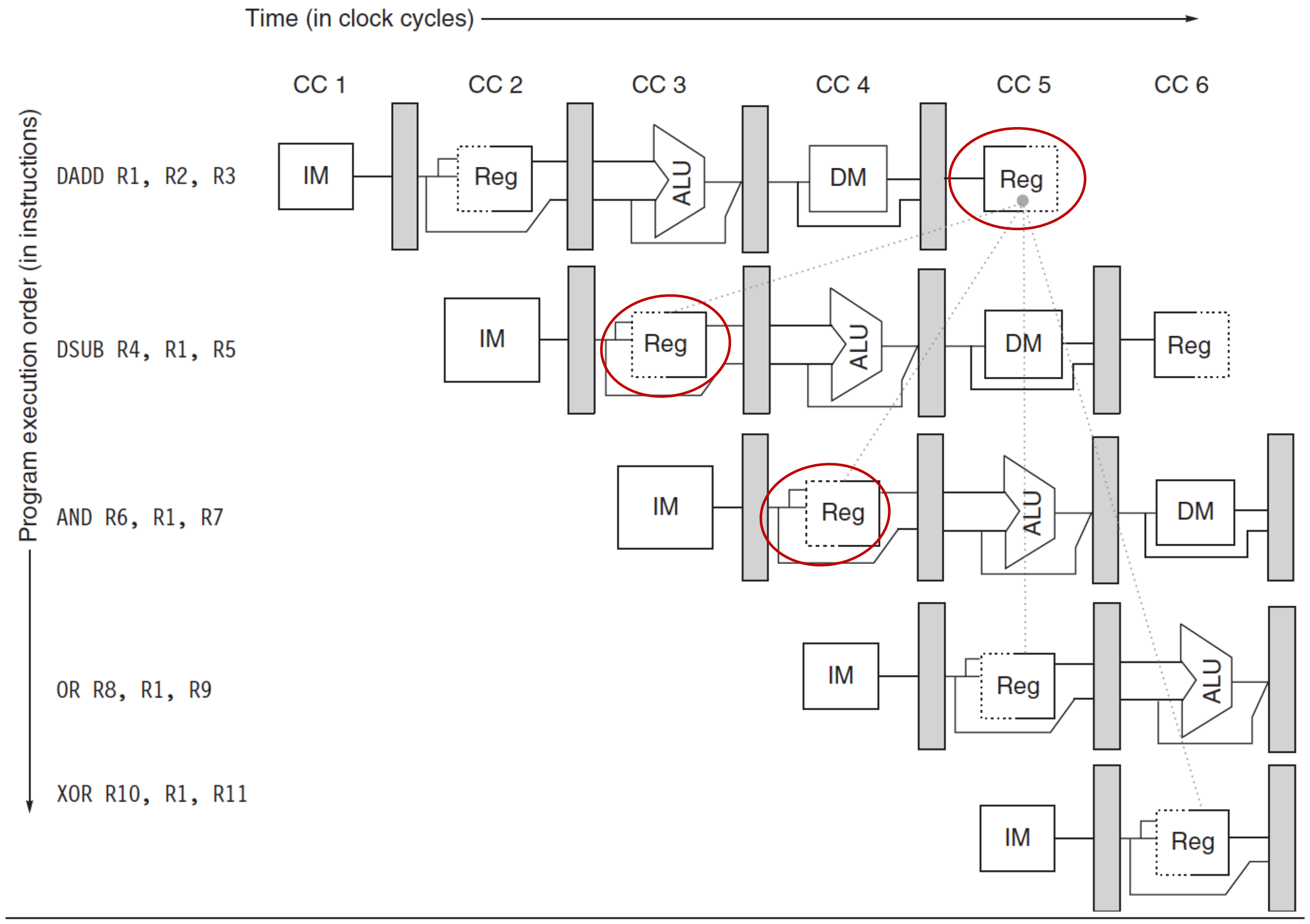
- When the hardware component cannot achieve two operations at the same time.
- **Example**: one memory for instructions and data, and two pipelined instructions are trying to read from the memory at the same time. (one is fetching an instruction and the other is reading data)

2- Data hazards

- When an instruction uses the result of a previous instruction.
- Consider this:

Example 1

```
DADD        R1,R2,R3
DSUB        R4,R1,R5
AND         R6,R1,R7
OR          R8,R1,R9
XOR         R10,R1,R11
```



DADD writes the value of R1 in the WB pipe stage, but the DSUB instruction reads the value during its ID stage. Also the AND will try to read it before it's written back.

→ This problem is called a **data hazard** .

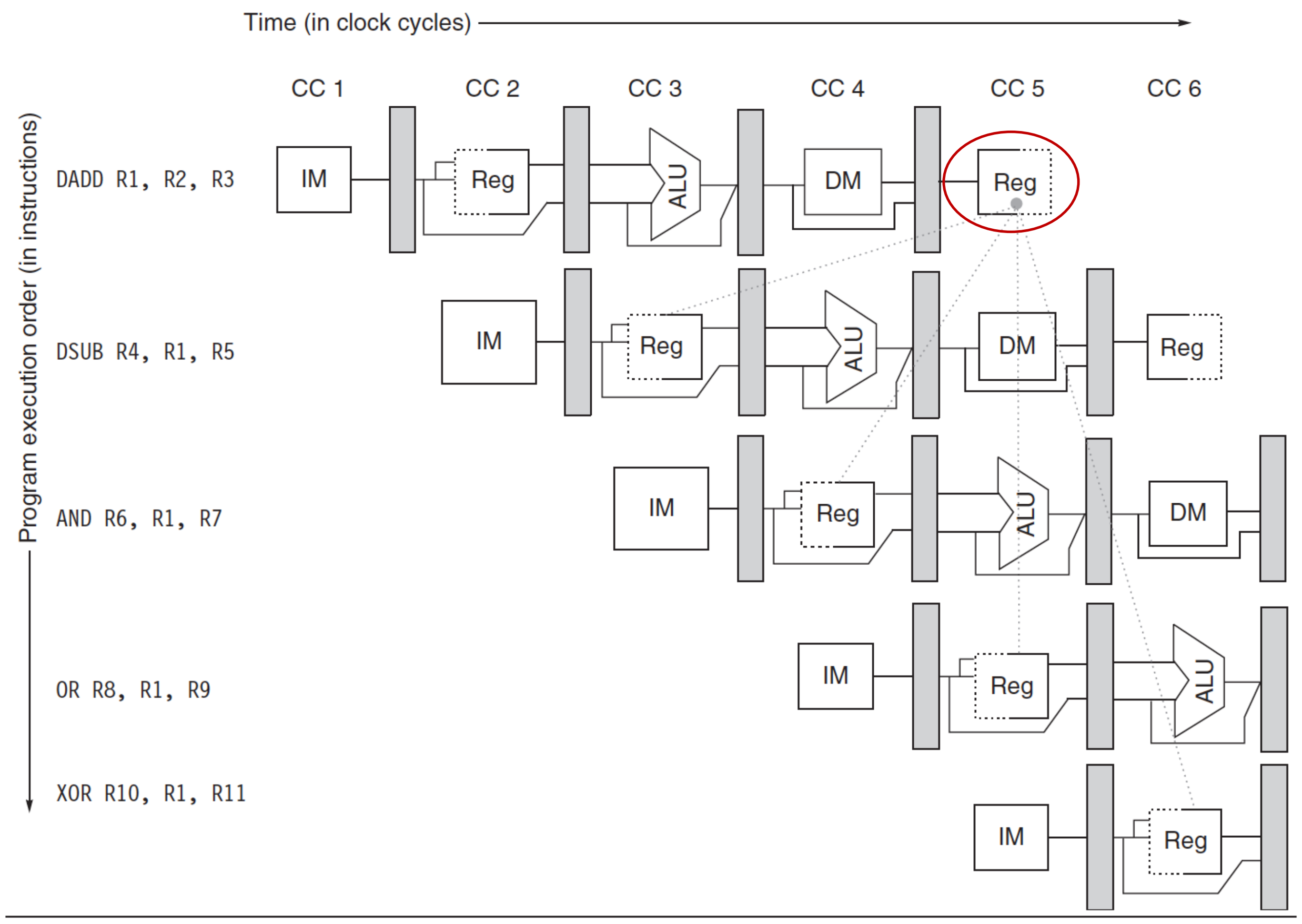
2- Data hazards (cont.)

- When an instruction uses the result of a previous instruction.
- Consider this:

Example 1

```
DADD    R1,R2,R3
DSUB    R4,R1,R5
AND     R6,R1,R7
OR      R8,R1,R9
XOR     R10,R1,R11
```

- **Does the XOR read wrong results of R1 too?**
- **What about the OR?**



End of lecture 2