香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Lecture 10: Pipelining

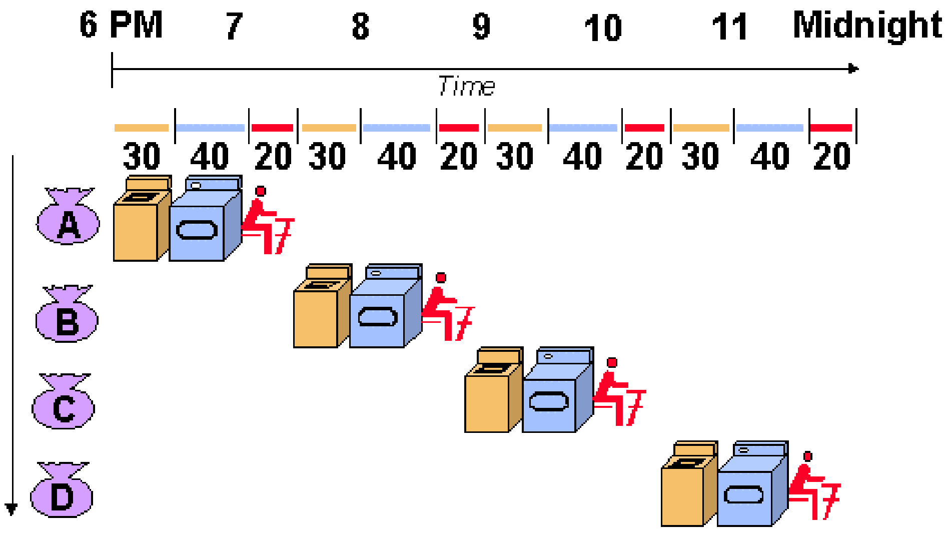
Ming-Chang YANG

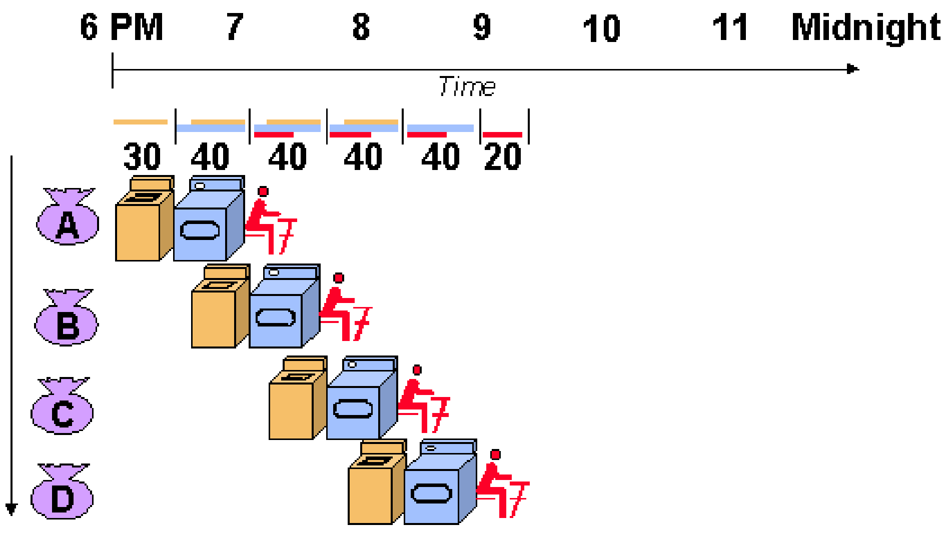
mcyang@cse.cuhk.edu.hk

Reading: Chap. 6

Why Pipelining?

- **Real-life Example**: Four loads of laundry that need to be *washed* (for 30 minutes), *dried* (for 40 minutes), and *folded* (for 20 minutes).





Without Pipelining

$$(30 + 40 + 20) * 4$$

$=$ 360 minutes in total

With Pipelining

$$30 + 40 * 4 + 20$$

$=$ 210 minutes in total

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/pipelining/index.html

Outline

- Pipelining in RISC-Style Processor
  - Pipeline Organization
  - Pipeline Stall: Hazards
    1) Data Dependencies
    2) Memory Delays
    3) Branch Delays
    4) Resource Limitations

- Pipelining in CISC-Style Processor

Recall: Five-Stage Organization (RISC)

- The execution can be arranged into five stages:

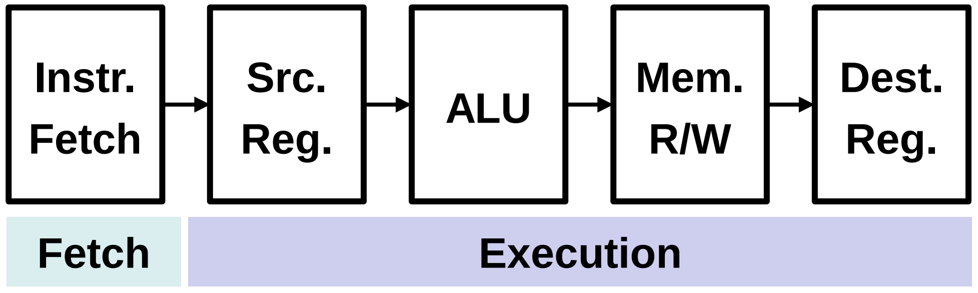
① Fetch an instruction and increment the PC.

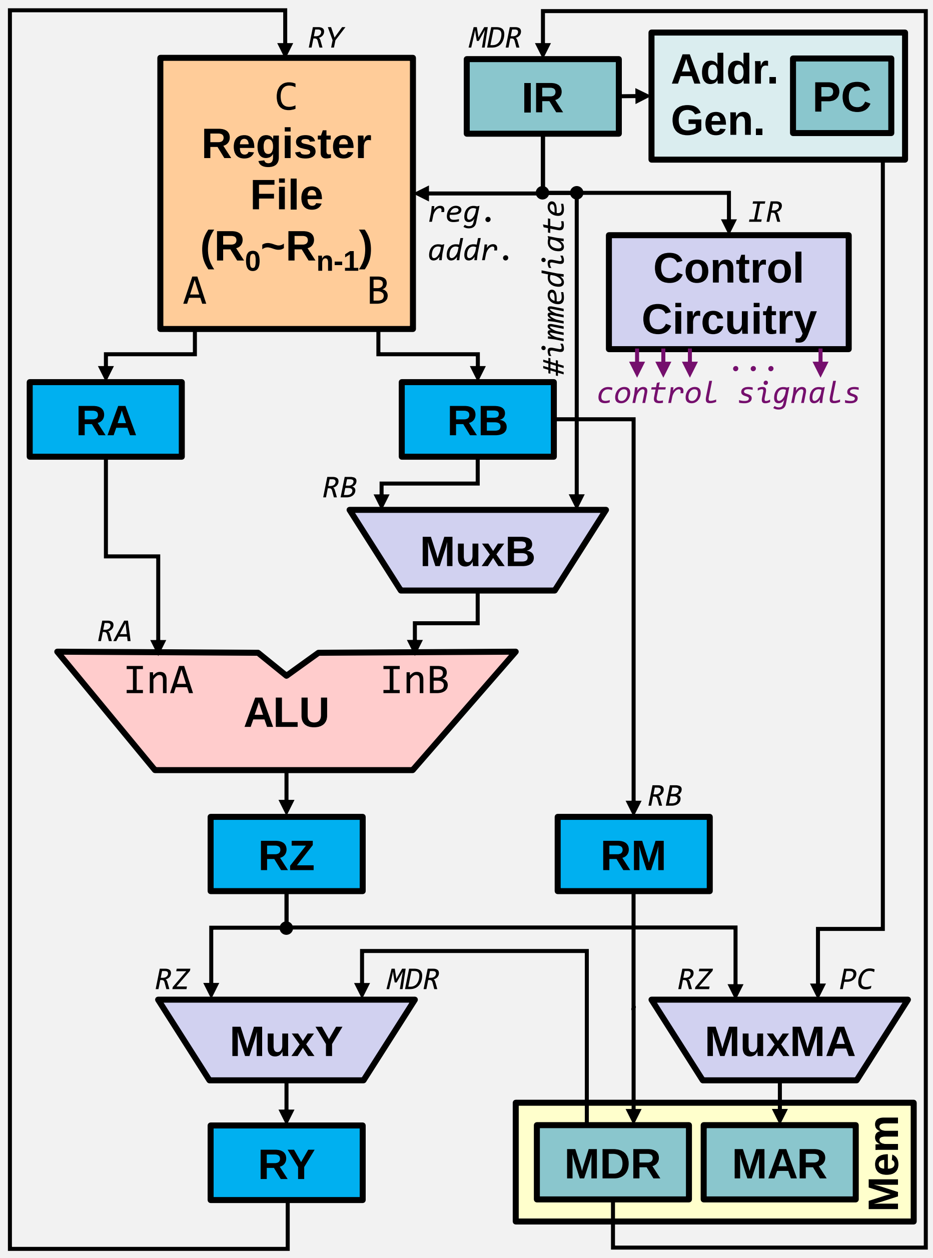
② Decode the instruction & read the source registers.

③ Perform an ALU operation.

④ Read/write memory data.

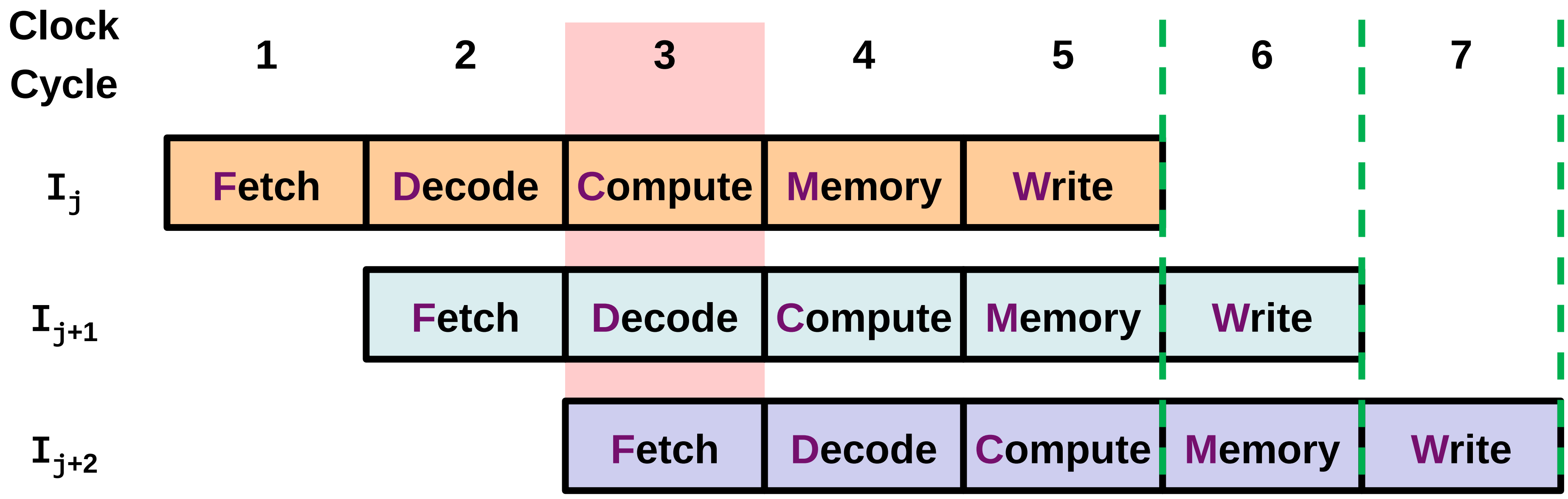
⑤ Write into the dest. reg.





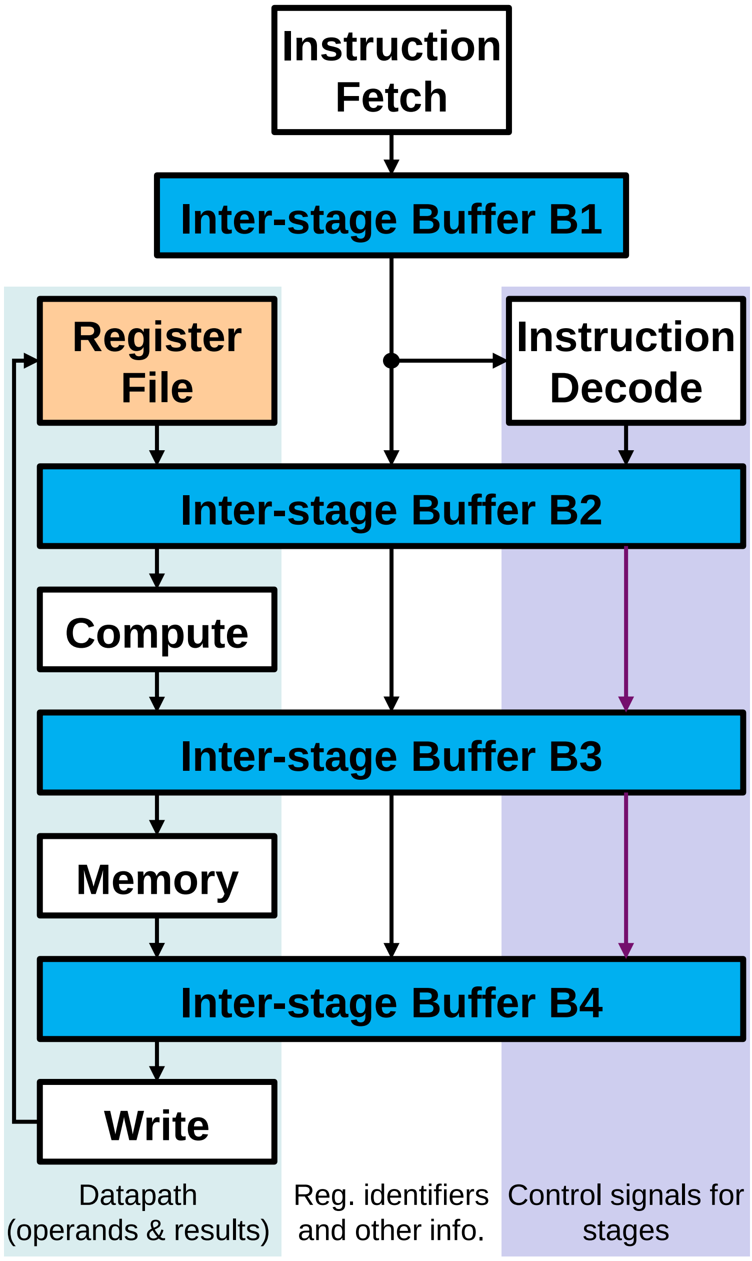
Pipelined Five-Stage Organization (1/2)

- The **five-stage organization** can allow instructions to be fetched and executed in a pipelined way easily.
  - The five stages are labeled as: **F**, **D**, **C**, **M**, and **W**.
  - At any time, each stage is working on a different instruction.
  - Ideally, instructions are done at the rate of **one per cycle**.
    - *Note: The time needed to perform any instruction is not changed: Any one instruction still takes (at least) five cycles to complete.*



Pipelined Five-Stage Organization (2/2)

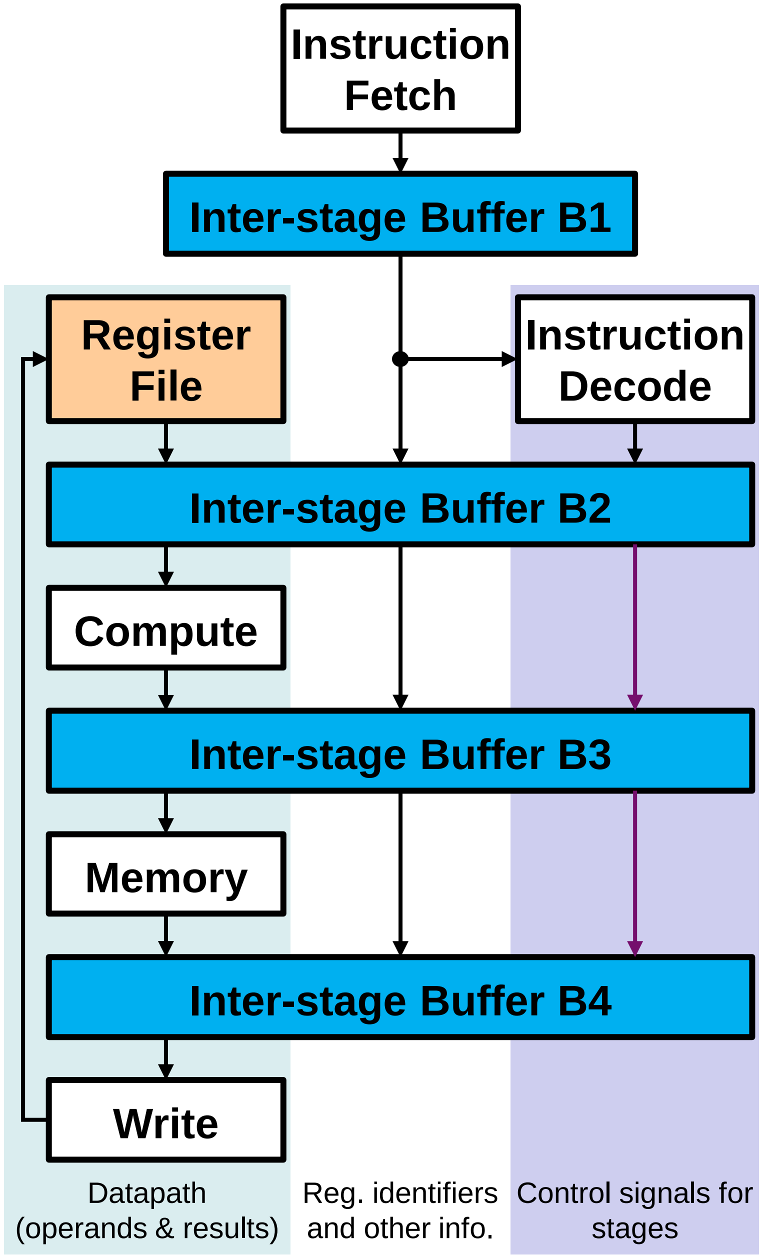
- **Inter-stage buffers** carry the info. from one stage to the next.
  - **B1** feeds **Decode** stage with the newly-fetched instruction.
  - **B2** feeds **Compute** stage with:
    - Two operands read from **Register File**;
    - The src./dest. register identifiers;
    - The immediate value from the instruction;
    - The control signals (which move though the entire pipeline via **B2**, **B3**, and **B4**).
  - **B3** holds the computed result or the data to be written to the memory.
  - **B4** feeds **Write** stage with the value to be written into **Register File**.
  - Note: **B1~B4** include the **inter-stage registers** (i.e., **RA/RB/RZ/RM/RY**).



Class Exercise 10.1

- During the clock cycle 5, what is the information held by the inter-stage buffers (i.e., B1 to B4), respectively?

Cycle	1	2	3	4	5	6	7	8	9
I_1	F	D	C	M	W				
I_2		F	D	C	M	W			
I_3			F	D	C	M	W		
I_4				F	D	C	M	W	
I_5					F	D	C	M	W



Outline

- Pipelining in RISC-Style Processor
  - Pipeline Organization
  - Pipeline Stall: Hazards
    1) Data Dependencies
    2) Memory Delays
    3) Branch Delays
    4) Resource Limitations

- Pipelining in CISC-Style Processor

Reality: The Pipeline May Stall

- If any pipeline stage requires more than 1 clock cycle, other stages must wait, causing the pipeline to stall.

Cycle	1	2	3	4	5	6	7	8	9	10
I_j	F	D	C	M	W					
I_{j+1}		F	D	C	M (cache miss)			W		
I_{j+2}			F	D	C	wait		M	W	
I_{j+3}				F	D	wait		C	M	W

- **Hazards**: Conditions that cause the pipeline to stall.
  - It might arise from ① *data dependencies*, ② *memory delays*, ③ *branch delays*, and ④ *resource limitations*.

1) Data Dependencies

- Pipeline may stall because of data dependencies.
- Consider the following two instructions:

```
Add   R2, R3, #50
Sub   R9, R2, #30
```

  - There is a data dependency since **R2** carries data from the first instruction to the second.
    - They must be performed in order to ensure the data consistency.
  - The **Decode** is stalled for three cycles to delay reading **R2** until cycle 6 by then the new value becomes available.

Cycle	1	2	3	4	5	6	7	8	9
Add	F	D	C	M	W_{R2}				
Sub		F	D				C	M	W

Hardware Sol.: Operand Forwarding

- **Operand forwarding** can alleviate the pipeline stalls due to data dependencies.
- Consider the following two instructions again:

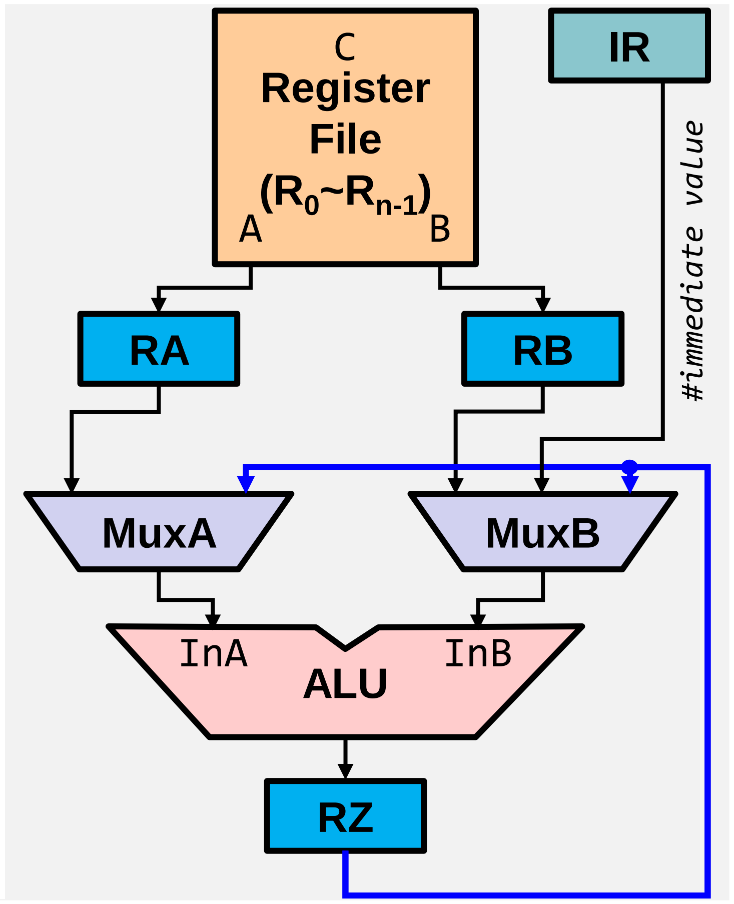
```
Add   R2, R3, #50
Sub   R9, R2, #30
```

  – The new value of **R2** is actually available at the end of cycle 3.

  – Rather than stalling **Sub**, the hardware can *forward* the value to where it is needed in cycle 4.

  – Additional hardware is needed to make such *forwarding* possible.

Cycle	1	2	3	4	5	6
Add	F	D	C_{R2}	M	W	
Sub		F	D	C	M	W



Class Exercise 10.2

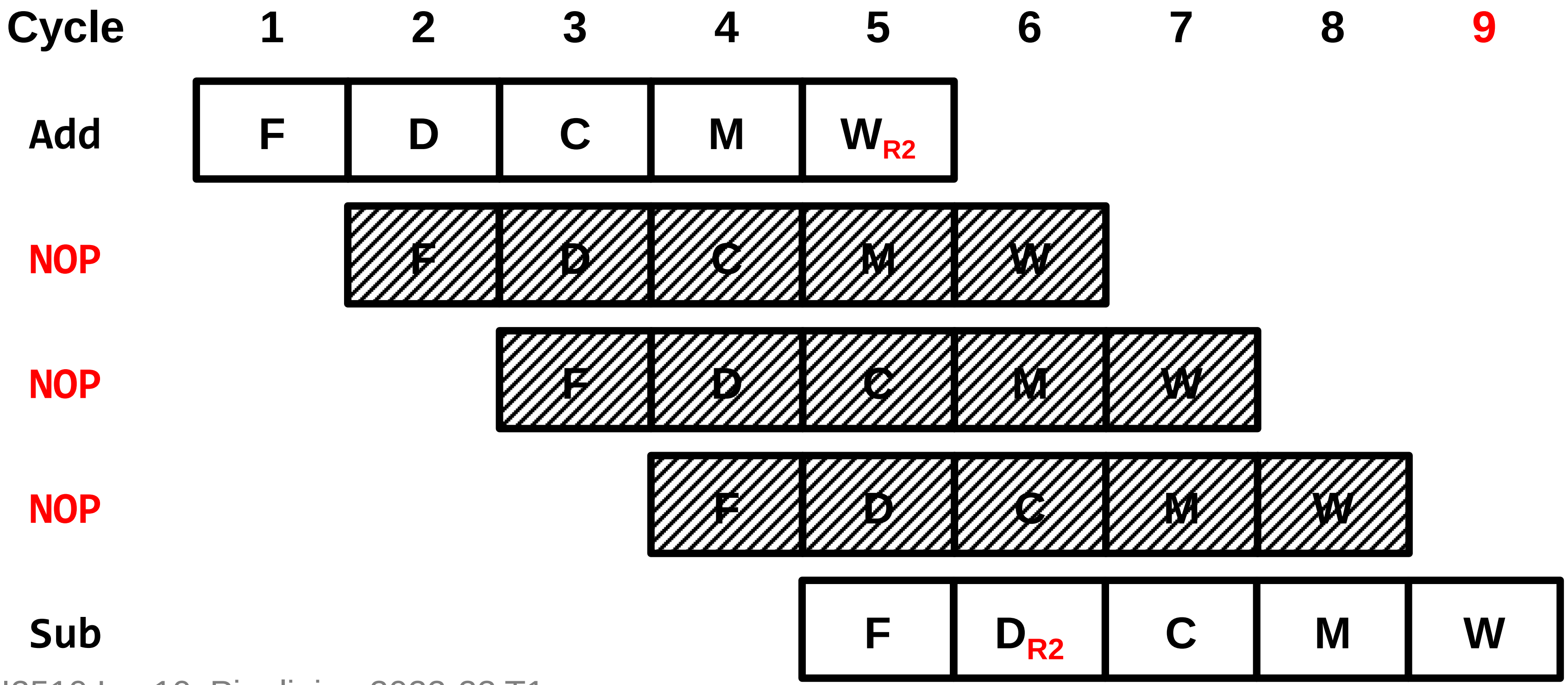
- Consider the following instructions:

```
Add  R2, R3, #100
Or   R4, R5, R6
Sub  R9, R2, #30
```

- How many clock cycles are required to complete the execution when the operand forwarding technique is not used or used, respectively?
  - *Note: The minimal number of cycles should be derived.*

Software Sol.: NOP Instruction

- The compiler can also identify the data dependency and insert **NOP** (**No-operation**) instructions to create idle clock cycles (also called *bubbles*).
  - **Pros**: simplified hardware
  - **Cons**: larger code size, “non-reducing” total execution time



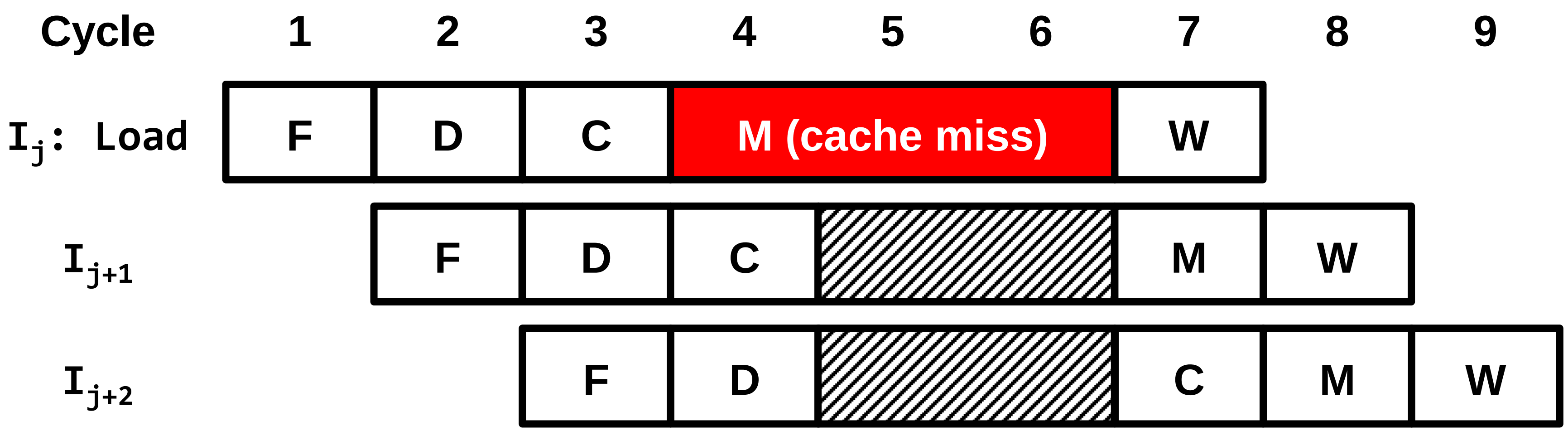
Software Sol.: Instruction Reordering

- The compiler can further move "useful instructions" into the NOP slots by **instruction reordering**.
  - It must carefully consider data dependencies still.
  - It can possibly improve performance and reduce code size.
    - Depending on the extent to which NOP slots can be usefully filled.

Cycle	1	2	3	4	5	6	7	8	9
Add	F	D	C	M	W_{R2}				
I_j		F	D	C	M	W			
I_{j+1}			F	D	C	M	W		
I_{j+2}				F	D	C	M	W	
Sub					F	D_{R2}	C	M	W

2) Memory Delays

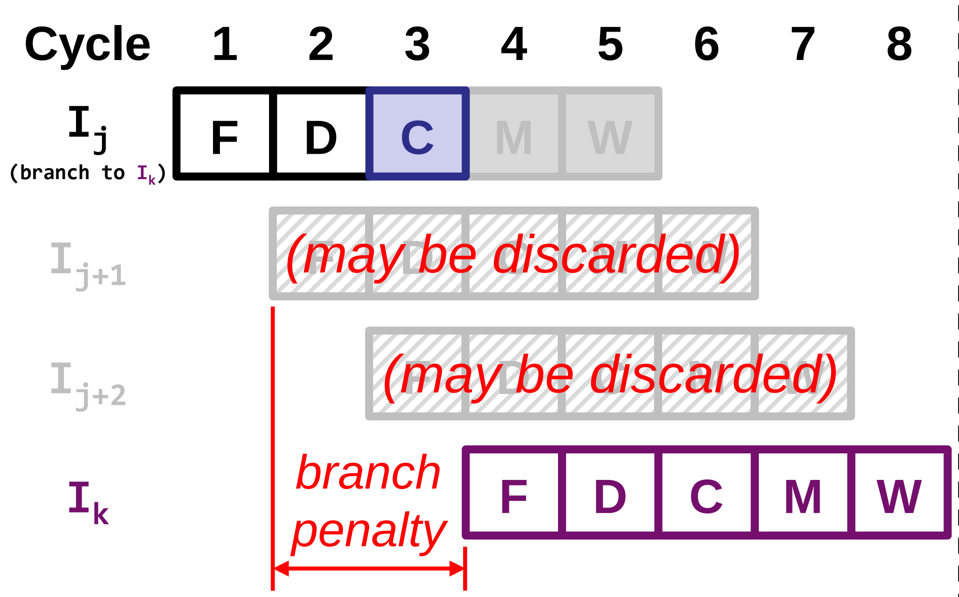
- Delays arising from memory accesses are another cause of pipeline stalls.
  - E.g., a **Load** instruction may require more than one cycle to obtain its operand from memory due to cache miss, which causes all subsequent instructions to be delayed.
    - *Note: A memory access may take more than ten cycles, but the figure shows only three cycles for simplicity.*



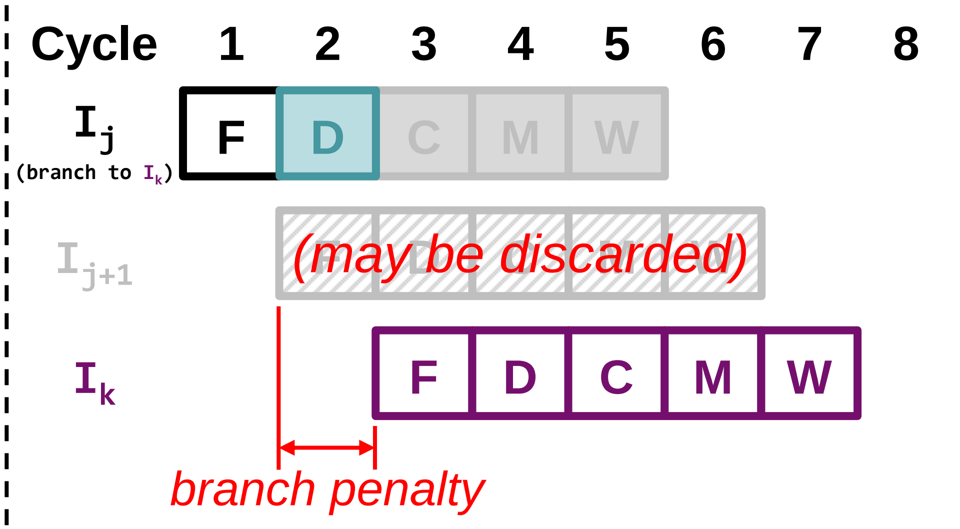
  - *Question: How can we alleviate such pipeline stalls?*

3) Branch Delays

- **Branch instructions** may also stall the pipeline.
  - They must first be decoded or executed to determine whether and where to branch.
  - **Branch Penalty**: The delays caused by a branch instruction.
    - It can be reduced by computing the branch target **earlier**.

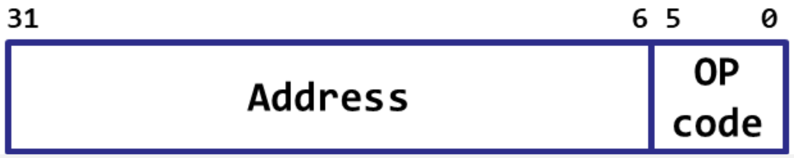


The branch target is computed in **C**.
Branch penalty: 2 clock cycles



The branch target is computed in **D**.
Branch penalty: 1 clock cycle
(The hardware must be modified.)

Recall: Branch



① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4 (shown here)

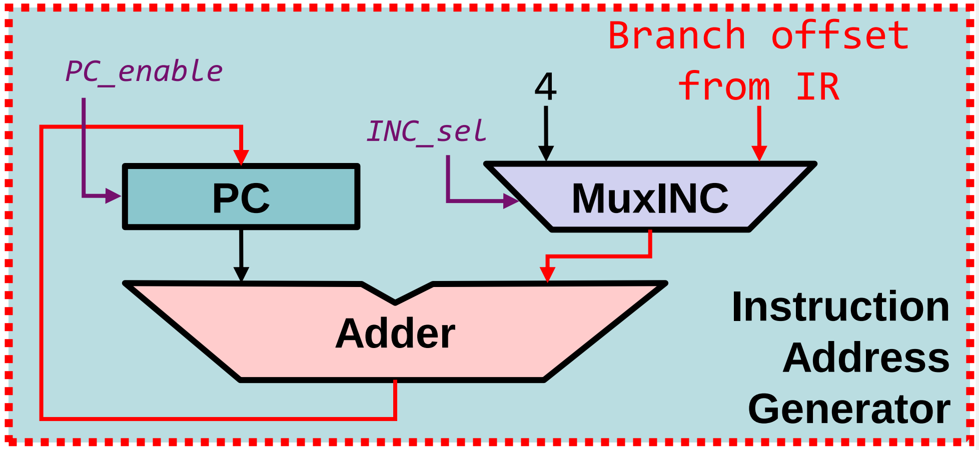
② Decode instruction

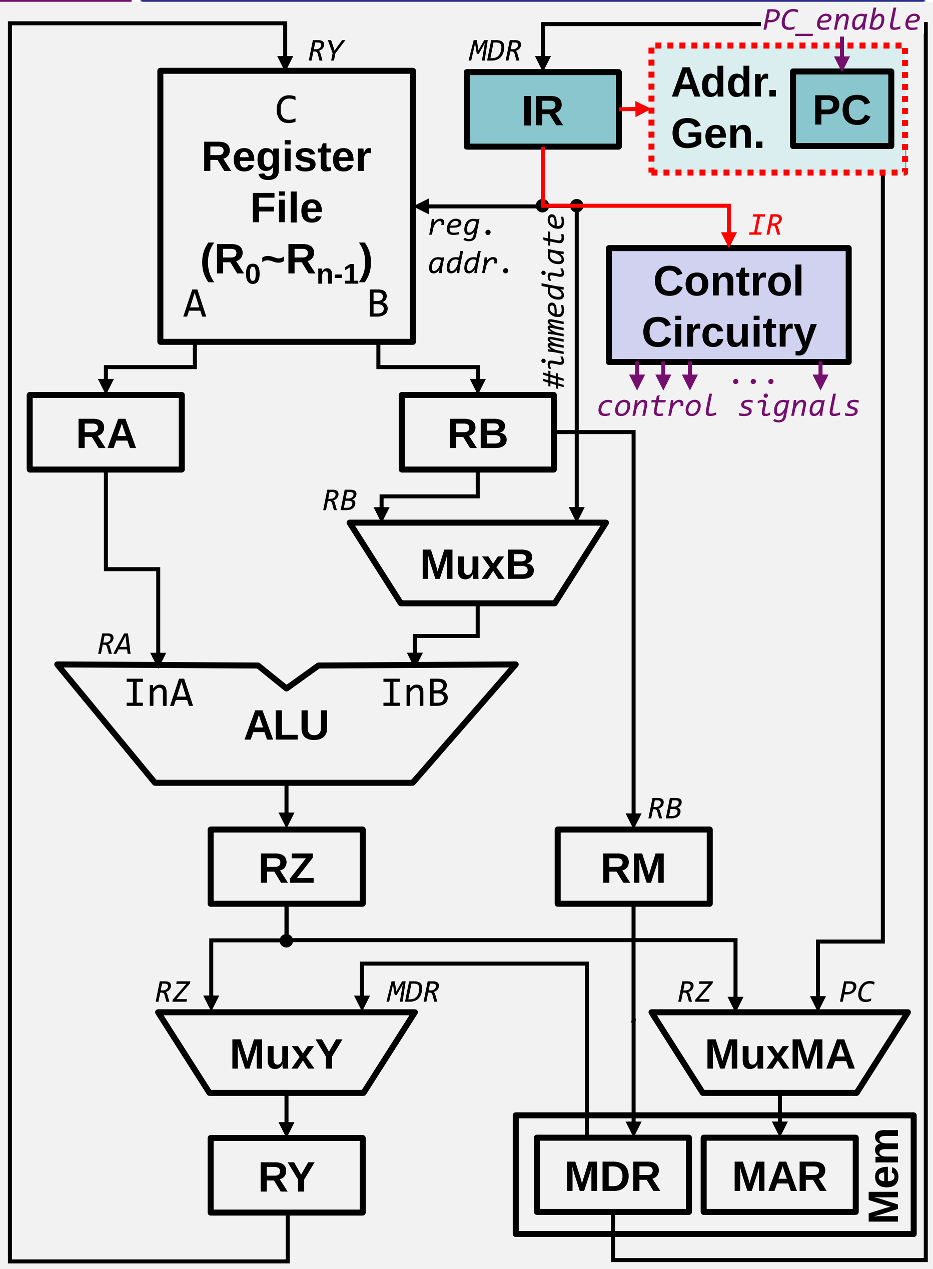
③ PC ← [PC] + branch offset

- The **branch offset** is from **IR**.
- **MuxINC** (in **Instruction Address Generator**) is set to select **offset**.

④ No action

⑤ No action





Solution: Delayed Branching

- The location(s) that follows a branch instruction is called the branch delay slot(s).
  - **Key Observation**: The instruction(s) in the delay slot(s) is always executed whether or not the branch is taken.
- **Delayed Branching**: The compiler may find a "suitable instruction(s)" to fill the delay slot(s).
  - One needed to be executed even when the branch is taken.

	Add R7, R8, R9
	I_j: Branch TARGET
	$\boxed{I_{j+1} \text{ (always executed)}}$
	...
TARGET	I_k

(a) Original sequence of instructions

	I_j: Branch TARGET
	$\boxed{\text{Add R7, R8, R9}}$
	I_{j+1}
	...
TARGET	I_k

(b) Placing the Add instruction in the branch delay slot

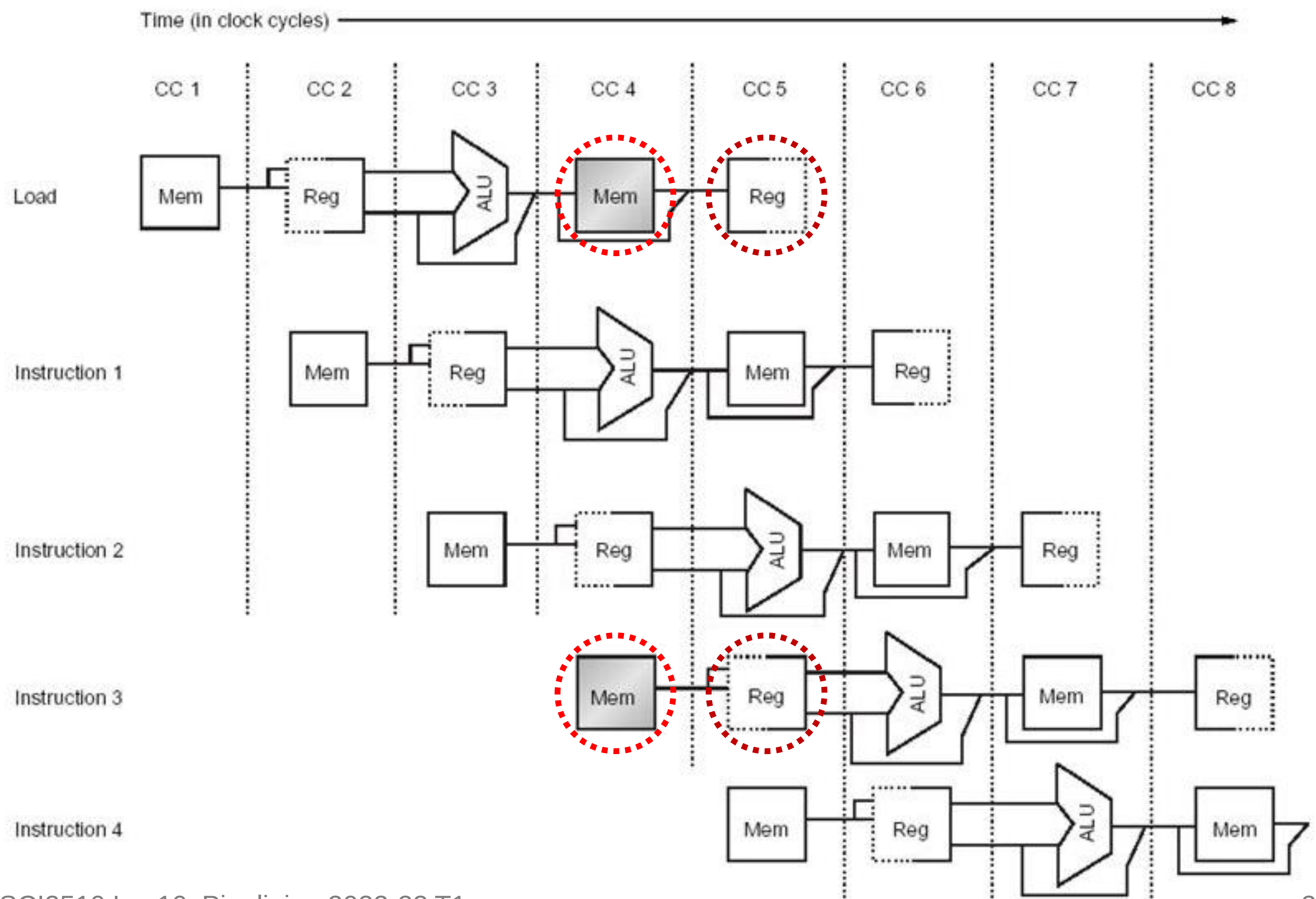
Class Exercise 10.3

- Suppose a pipelined processor has two branch delay slots but does not employ the delayed branch.
- If 20 percent of the instructions executed are branch instructions, what is the required number of clock cycles to complete 100 instructions?

4) Resource Limitations (1/2)

- The pipeline stalls when there are insufficient hardware resources to allow concurrent execution.
  - If two instructions need to access the same resource in the same clock cycle, one instruction must be stalled.
  - **Case 1**: One instruction is accessing memory during the **M** stage, while another is being fetched.
    - Possible Solution: Separating instruction & data caches.
  - **Case 2**: Two instructions require access to Register File at the same time.
    - Possible Solution: Equipping Register File with more input and output ports.
- In general, this can be prevented by providing additional hardware resources ($$$).

4) Resource Limitations (2/2)



Outline

- Pipelining in RISC-Style Processor
  - Pipeline Organization
  - Pipeline Stall: Hazards
    1) Data Dependencies
    2) Memory Delays
    3) Branch Delays
    4) Resource Limitations

- Pipelining in CISC-Style Processor

Pipelining in CISC-Style Processors?

- Complications arise for pipelining in CISC processors:
  - Reasons? CISC-style instructions are variable in size, may have multiple memory operands, and may have more complex addressing modes.

- Nonetheless, pipelined processors have still been implemented for CISC-style instruction sets.
  - For example, Core i7 architecture has a 14-stage pipeline.
  - To reduce internal complexity, CISC-style instructions are dynamically converted by the hardware into simpler RISC-style micro-operations.
    - This approach preserves code compatibility while making it possible to use the aggressive performance enhancement techniques that have been developed for RISC-style instruction sets.

Summary

- Pipelining in RISC-Style Processor
  - Pipeline Organization
  - Pipeline Stall: Hazards
    1) Data Dependencies
    2) Memory Delays
    3) Branch Delays
    4) Resource Limitations

- Pipelining in CISC-Style Processor