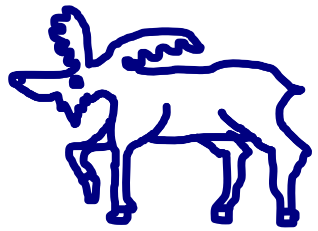
Lecture 14
Pipelining

Byung-gi Kim

School of Computer Science and Engineering

Soongsil University



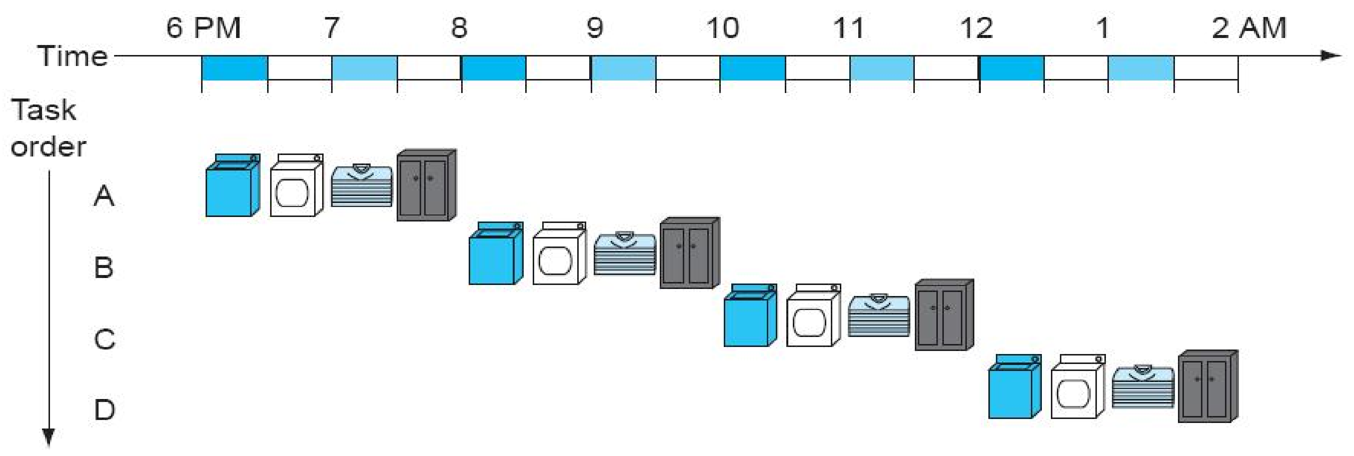
4. The Processor

4.5 An Overview of Pipelining

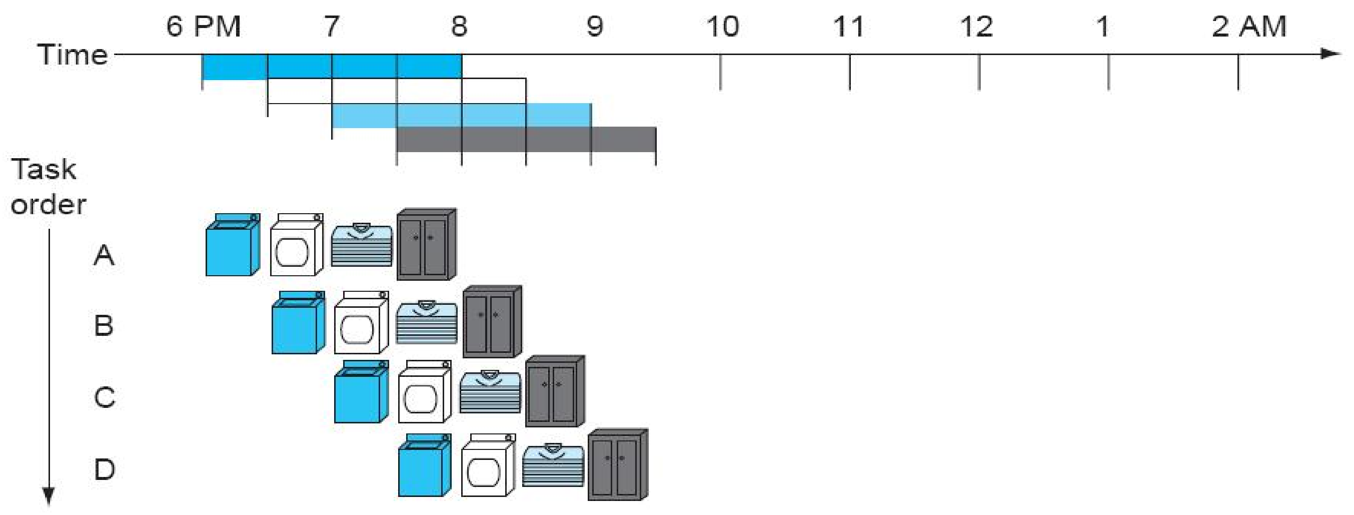
- **Pipelining**
  - Implementation technique in which multiple instructions are overlapped in execution
  - Exploits parallelism among the instructions in a sequential instruction stream
  - Improves instruction **throughput** rather than individual instruction **execution time**

The Laundry Analogy for Pipelining

Figure 4.25



8 hours for 4 tasks



3.5 hours for 4 tasks

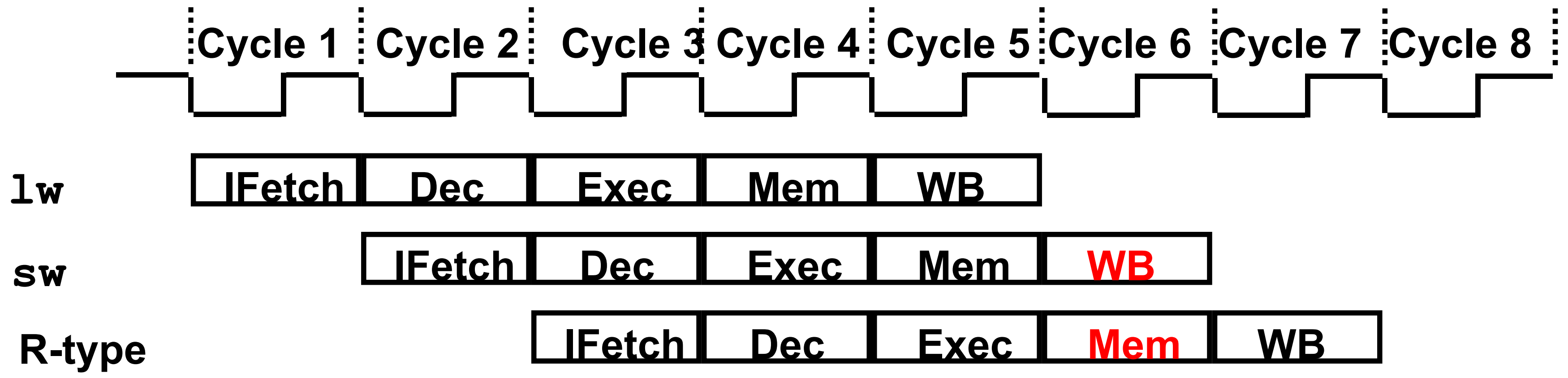
speedup = 8/3.5 = 2.3

5 Stages of Instruction Execution

1. Fetch instruction.
2. Read registers while decoding the instruction.
3. Execute the operation or calculate an address.
4. Access an operand in data memory.
5. Write the result into a register.

A Pipelined MIPS Processor

- Start the next instruction before the current one has completed
  - improves throughput - total amount of work done in a given time
  - instruction latency (execution time, delay time, response time - time from the start of an instruction to its completion) is *not* reduced

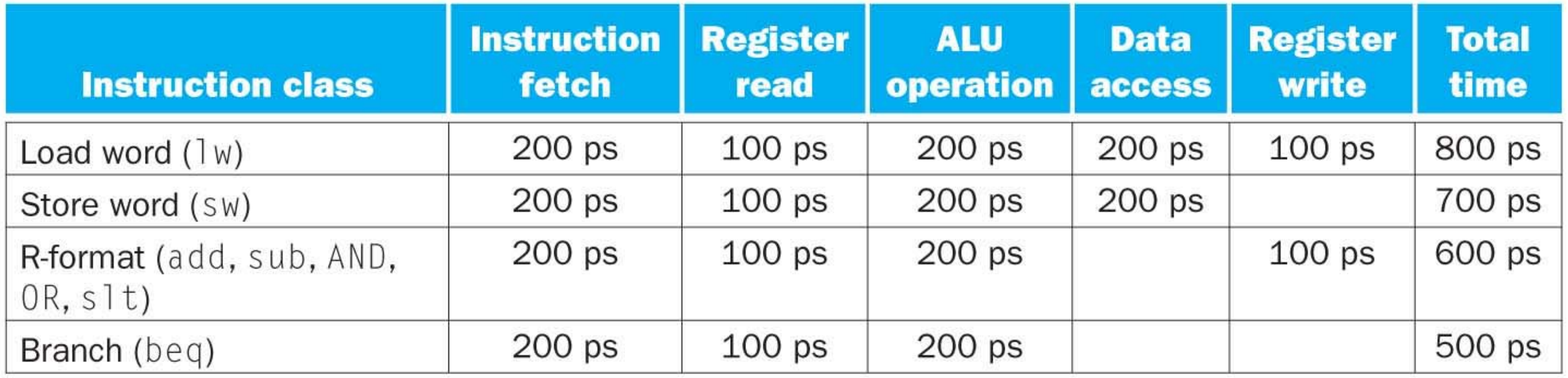


- clock cycle (pipeline stage time) is limited by the slowest stage
- for some stages don't need the whole clock cycle (e.g., WB)
- for some instructions, some stages are wasted cycles (i.e., nothing is done during that cycle for that instruction

Example: Single-Cycle vs. Pipelined

- Operation times for the major functional units
  - Memory access and ALU operation: 200 ps
  - Register file read or write: 100 ps
- **Compare the average time between instructions of a single-cycle implementation to a pipelined implementation.**

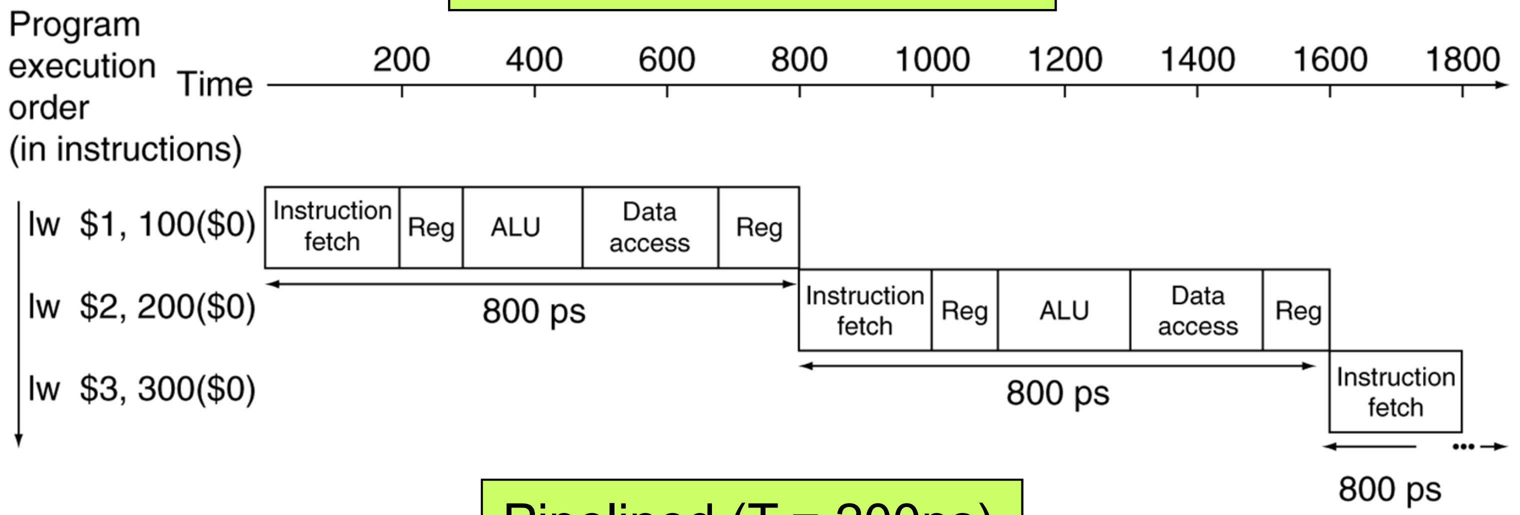
[Answer]



Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

Figure 4.26

Single-cycle (T_c= 800ps)



Pipelined (T_c= 200ps)

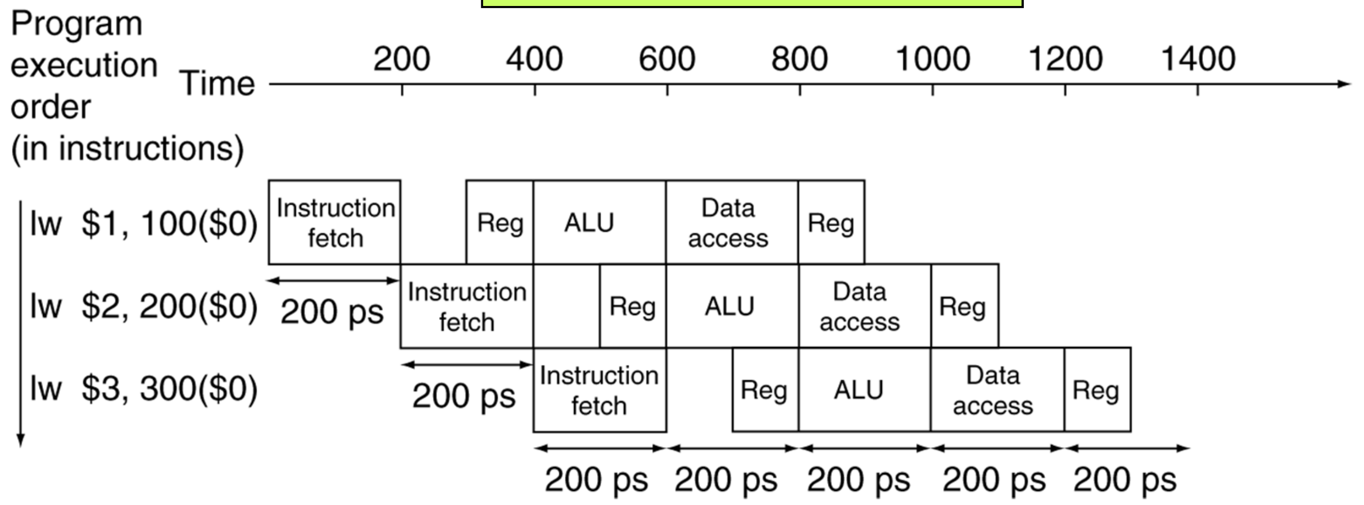


Figure 4.27

Pipeline Performance

- Clock cycle is determined by the time required for the slowest pipe stage.
- With perfectly balanced pipeline stages,

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

- Speedup of *k*-stage pipeline with clock cycle time=*t*
  - For n instructions,

$$\text{speedup} = \frac{n \times k \times t}{(k-1) \times t + n \times t} = \frac{n \times k \times t}{k \times t + (n-1) \times t}$$

  - For infinite number of instructions (i.e. $n \rightarrow \infty$),

speedup = k

Pipeline Hazards

- Situations that prevent starting the next instruction in the next cycle
- **Structural hazards**
  - different instructions in different stages conflicting for the same resource
  - Solution : increase the number of resources
- **Data hazards**
  - Instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction
  - Solutions: pipeline stall and forwarding (= bypassing)
- **Control hazards**
  - fetch cannot continue because it does not know the outcome of an earlier branch
  - Solutions: pipeline stall, branch prediction, delayed branch

4.6 Pipelined Datapath and Control

- **5 stages of instruction pipeline**
  - IF: Instruction fetch
  - ID: Instruction decode and register file read
  - EX: Execution and address calculation
  - MEM: Data memory access
  - WB: Write back

5 Steps of MIPS Datapath

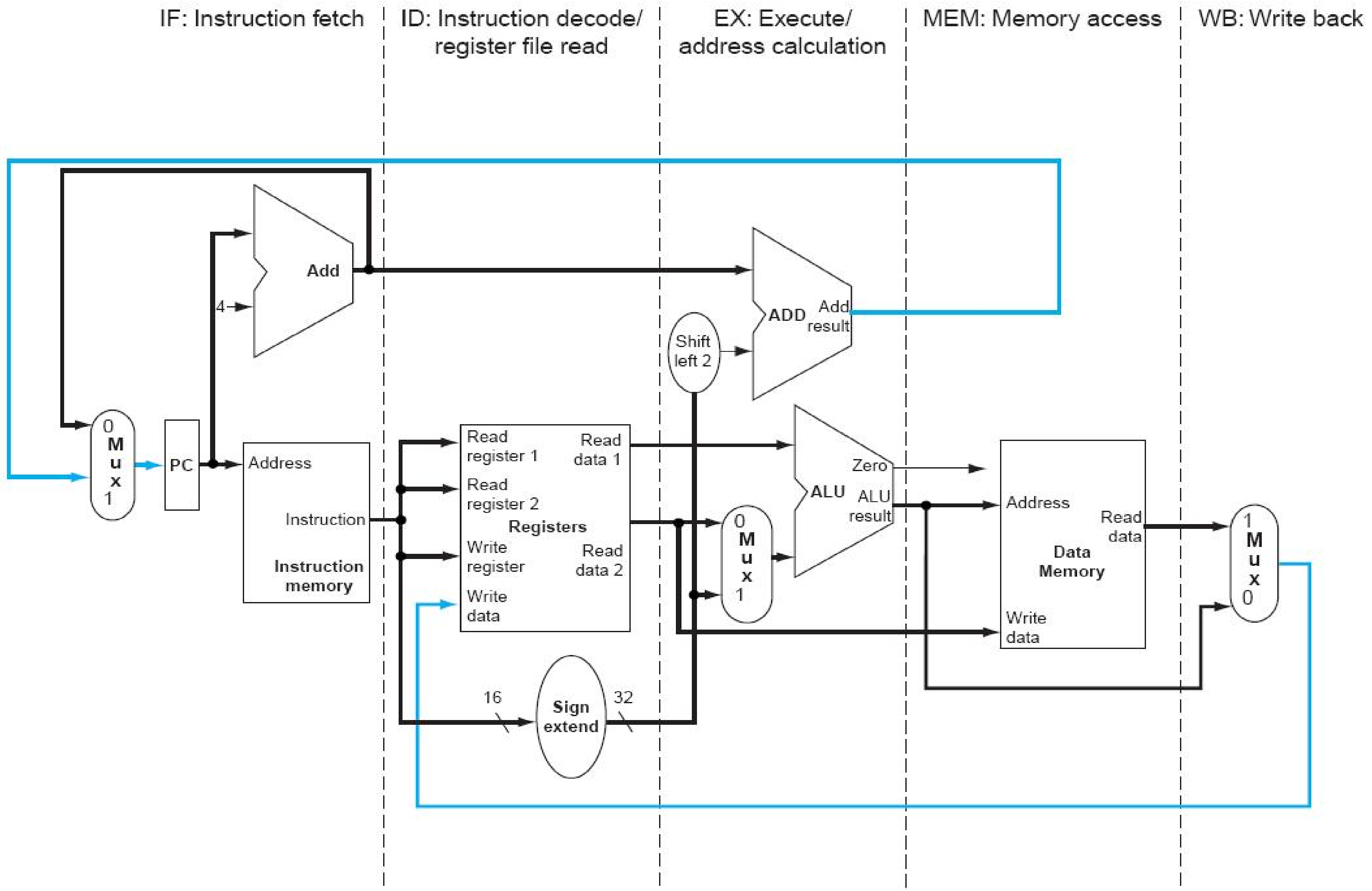


Figure 4.33

Pipelined Execution

- **2 exceptions to the left-to-right flow of instructions**
  - Write-back stage: Send ALU result back to the register file
    $\Longrightarrow$ Data hazard
  - Next PC select: Incremented PC or the branch address from MEM stage
    $\Longrightarrow$ Control hazard

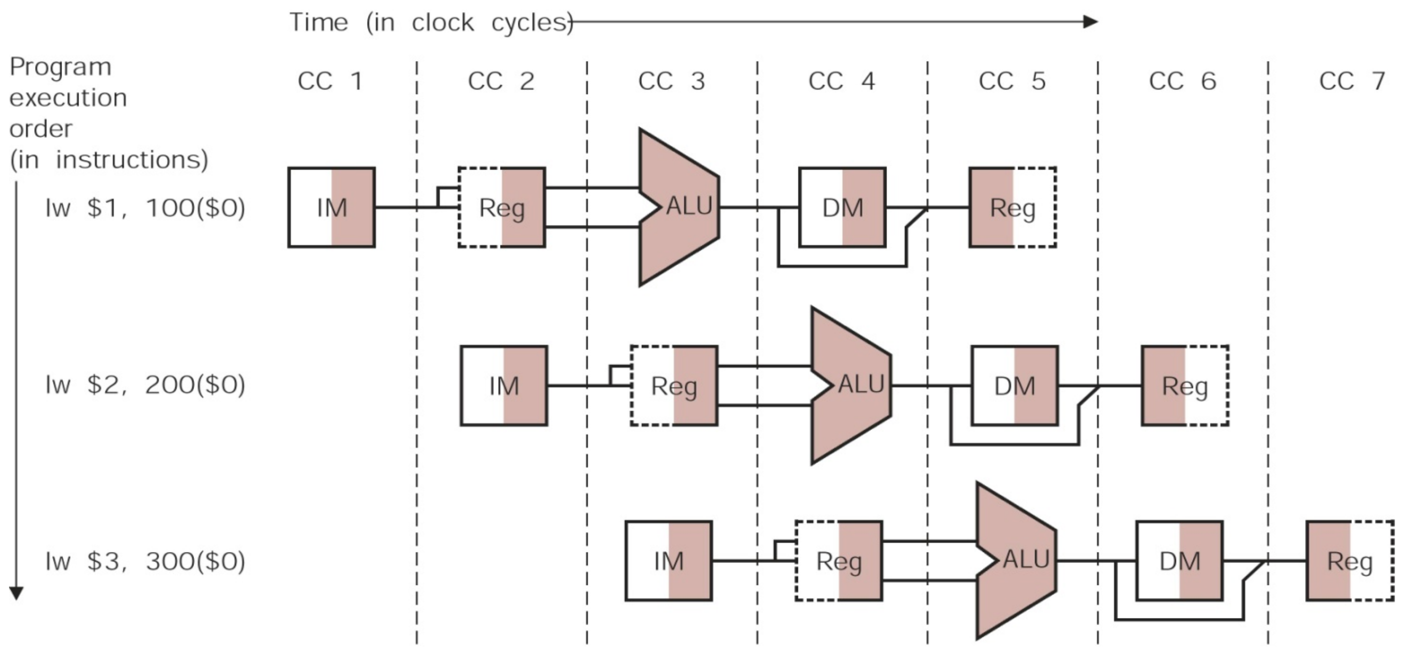
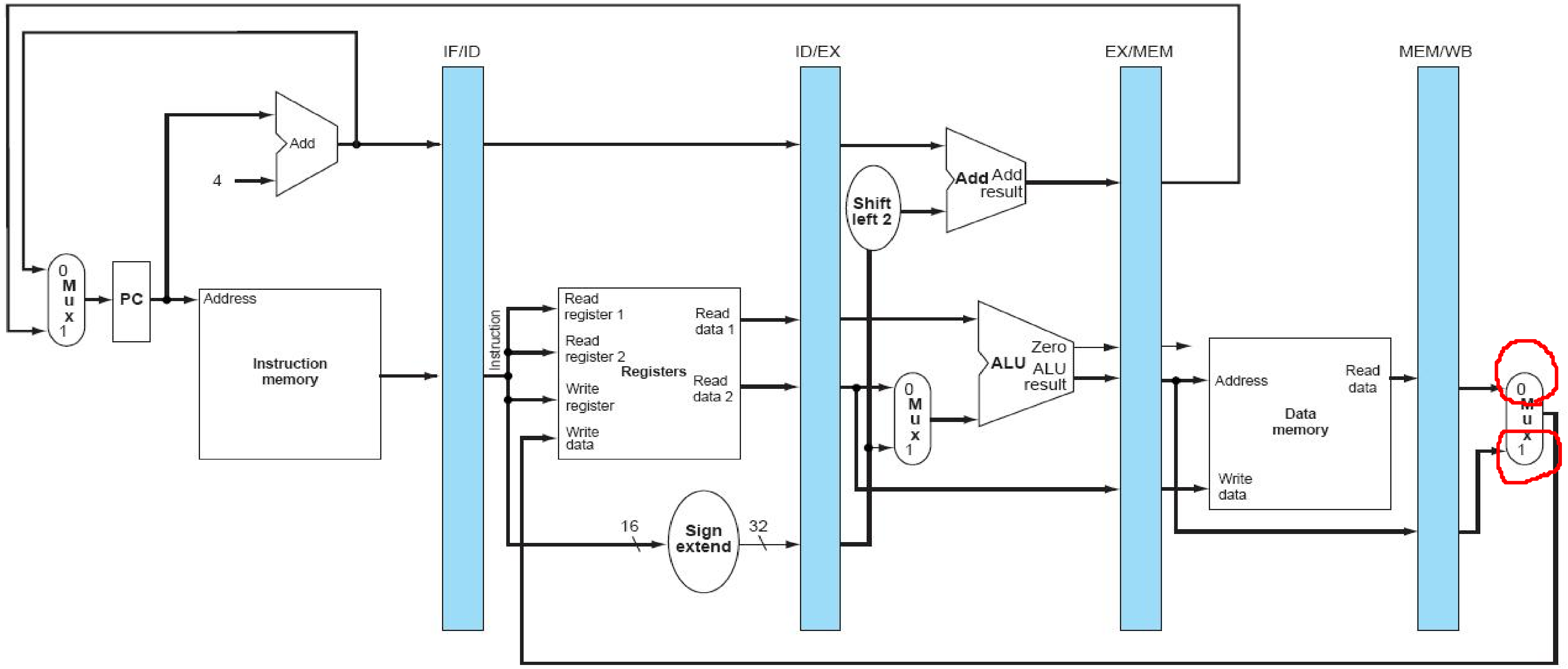


Figure 4.34

Pipelined Version of the Datapath

- **Pipeline register**
  - Separation of the two stages
  - Hold information produced in previous cycle

Figure 4.35



Graphically Representing Pipelines

- **Multiple-clock-cycle pipeline diagram**
  - Showing resource usage

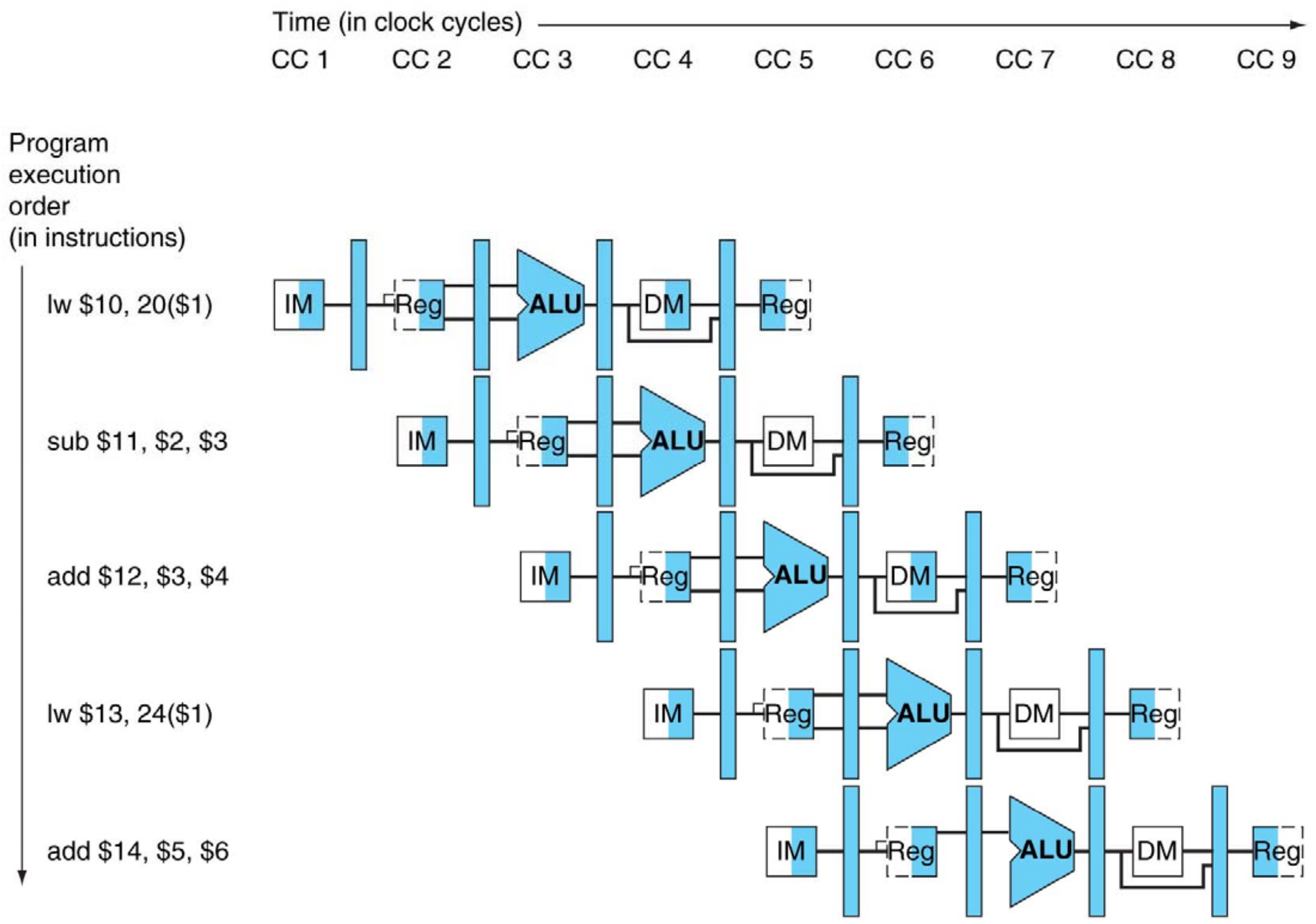
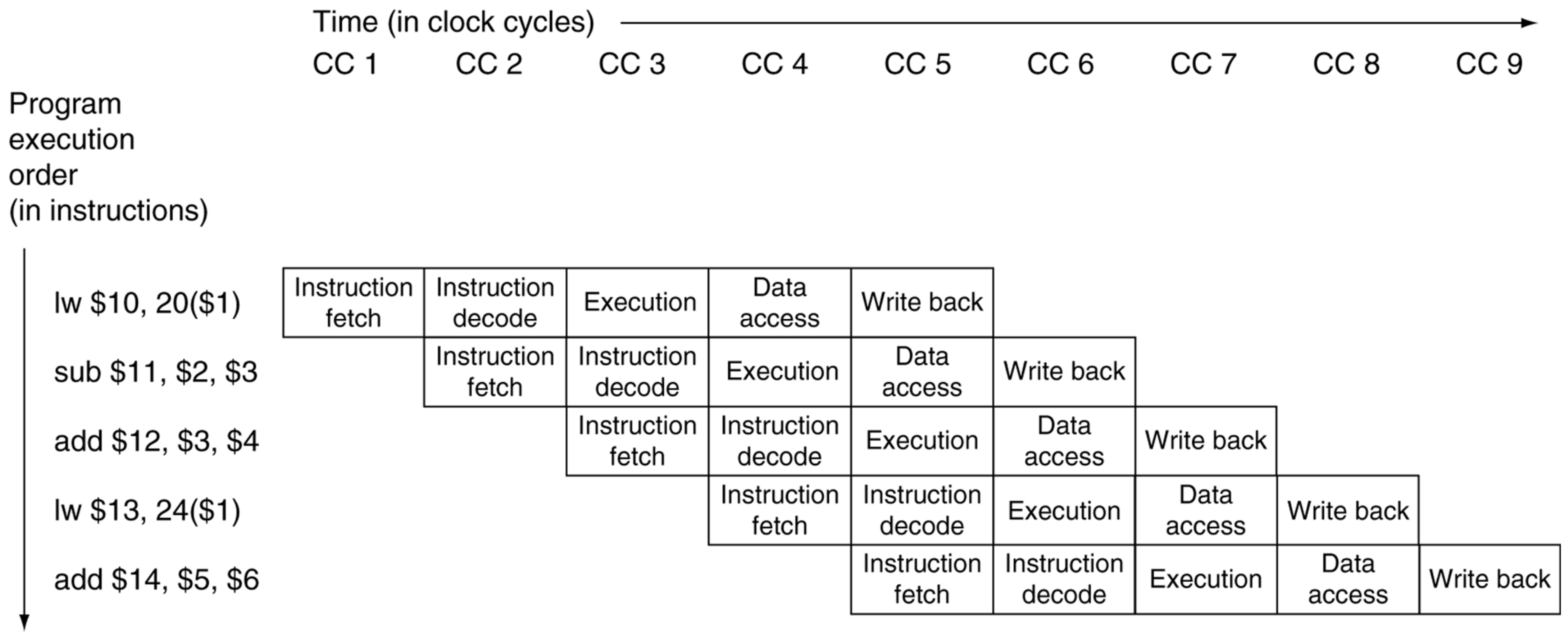


Figure 4.43

Traditional Multi-Cycle Pipeline Diagram



Time (in clock cycles) →

Program execution order (in instructions) ↓

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
lw $10, 20($1)	Instruction fetch	Instruction decode	Execution	Data access	Write back				
sub $11, $2, $3		Instruction fetch	Instruction decode	Execution	Data access	Write back			
add $12, $3, $4			Instruction fetch	Instruction decode	Execution	Data access	Write back		
lw $13, 24($1)				Instruction fetch	Instruction decode	Execution	Data access	Write back	
add $14, $5, $6					Instruction fetch	Instruction decode	Execution	Data access	Write back

Figure 4.44

Single-Cycle Pipeline Diagram

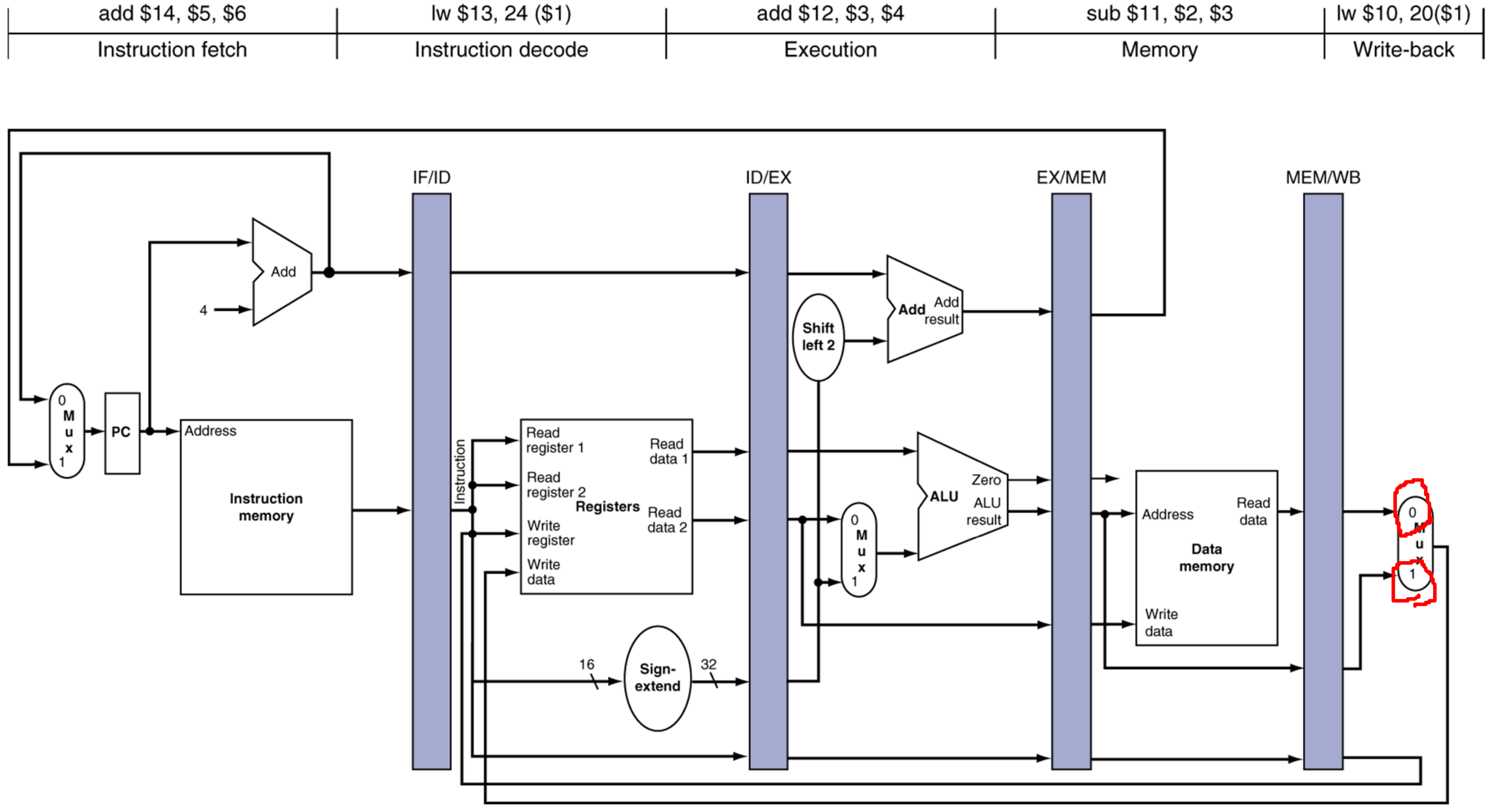


Figure 4.45

Simplified Single-Cycle Pipeline Diagram

	IF	ID	EX	MEM	WB
clock 1	I1				
clock 2	I2	I1			
clock 3	I3	I2	I1		
clock 4	I4	I3	I2	I1	
clock 5	I5	I4	I3	I2	I1
clock 6	I6	I5	I4	I3	I2
clock 7	I7	I6	I5	I4	I3