CALIFORNIA STATE UNIVERSITY
FULLERTON

CPSC-440 Computer System Architecture

Lecture 4

Processor Structure and Instruction Cycles

Pipelining Basic

What is a Program?

- A sequence of instructions (steps)
  - An instruction
    - A binary number
    - Consisting of two parts
      - Opcode: what to do
      - Operand: what data should be used
  - Example:

Opcode	Operand

- For each instruction (step), an arithmetic or logical operation is done
- For each operation, a different set of control signals is needed

Function of Control Unit

- Each operation is associated with a unique opcode
  - e.g., ADD, MOVE
- The hardware interprets the opcode and issues the control signals
- We have a computer!

Computer Components: Top Level View

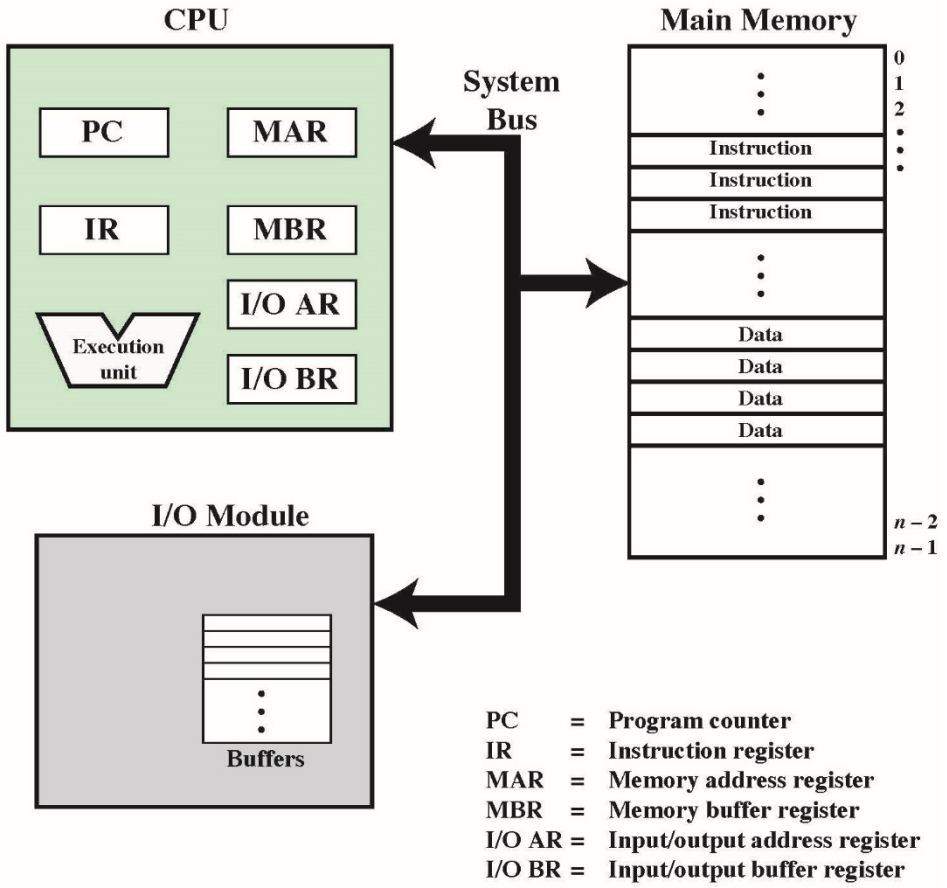
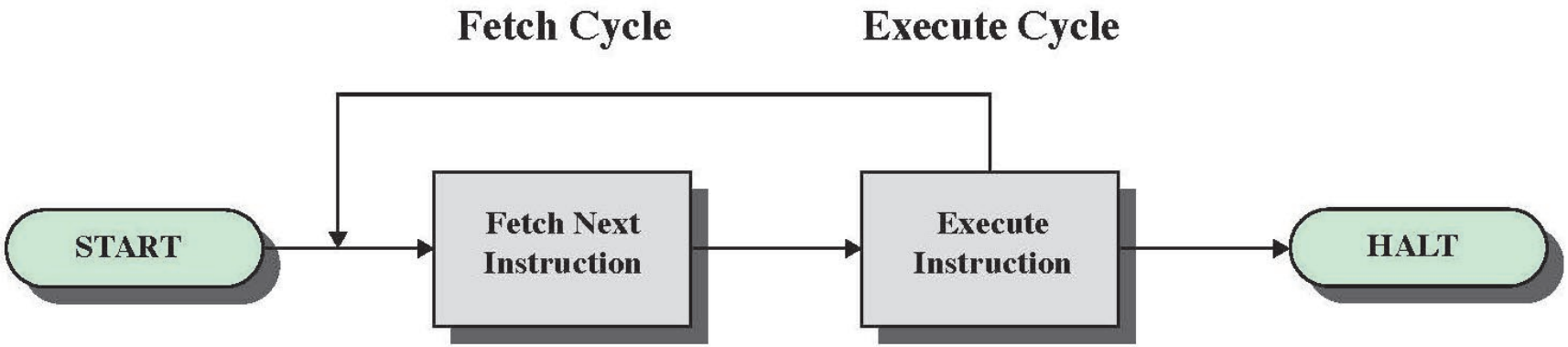


Figure 3.2 Computer Components: Top-Level View

Instruction Cycle

- Two basic steps:
  - Fetch
  - Execute



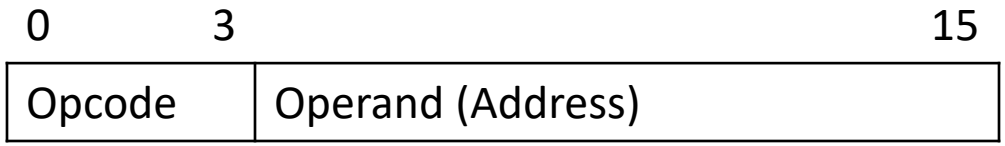
Fetch Cycle

- Program Counter (PC) holds address of next instruction to fetch
- CPU fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- CPU interprets instruction and performs required actions

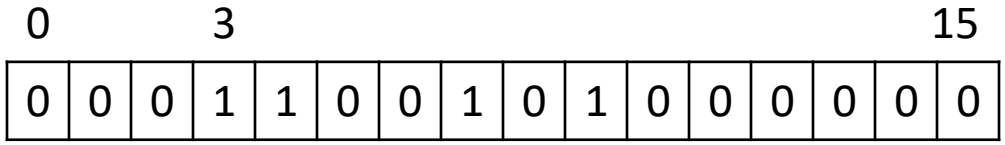
Execute Cycle

- Processor-Memory
  - Data transfer between CPU and main memory
- Processor-I/O
  - Data transfer between CPU and I/O module
- Data Processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g., jump
- An instruction's execution may involve a combination of above

Example of Program Execution



- Load AC from memory
  - Opcode: 0x1
  - Operand (Address): 0x940



- PC: program counter
- AC: accumulator
- IR: instruction register

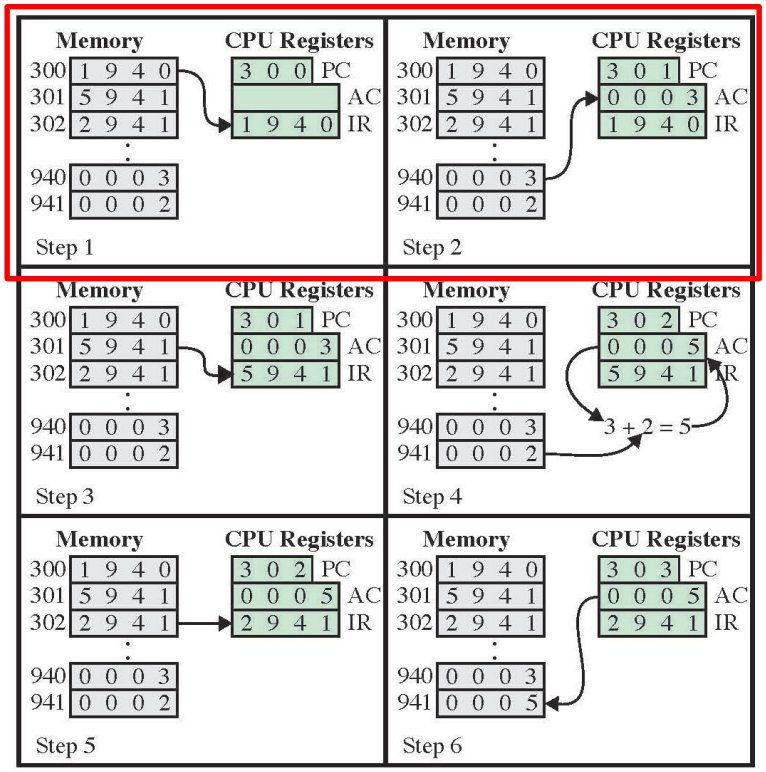
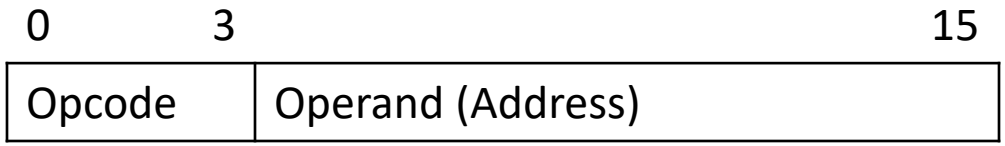
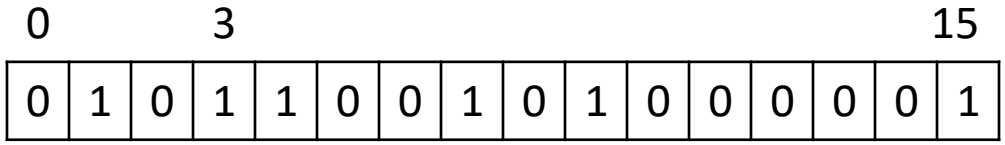


Figure 3.5 Example of Program Execution
(contents of memory and registers in hexadecimal)

Example of Program Execution



- Add to AC from memory
  - Opcode: 0x5
  - Operand (Address): 0x941



- PC: program counter
- AC: accumulator
- IR: instruction register

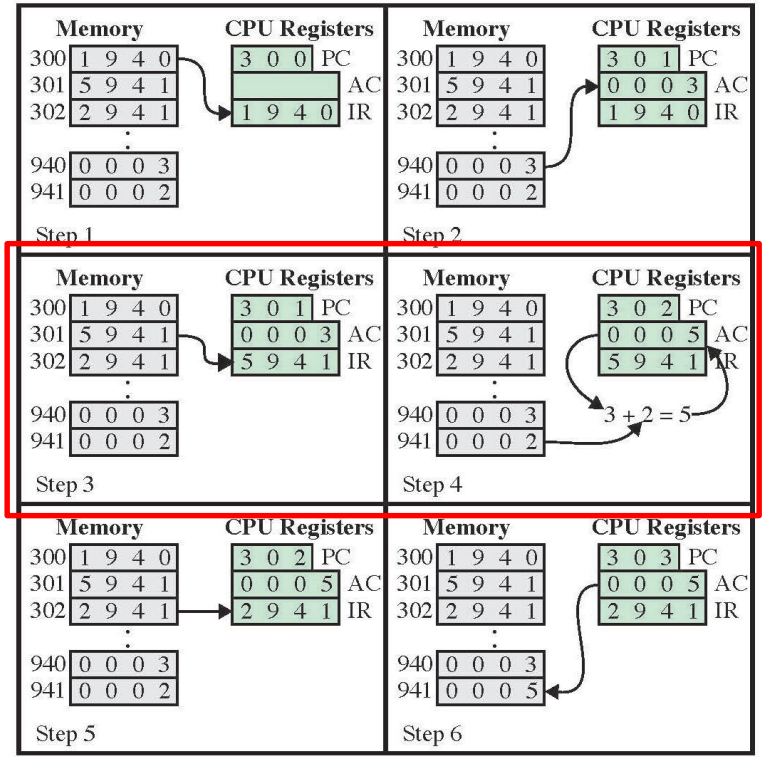
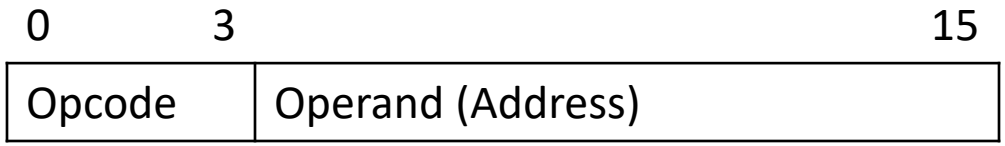
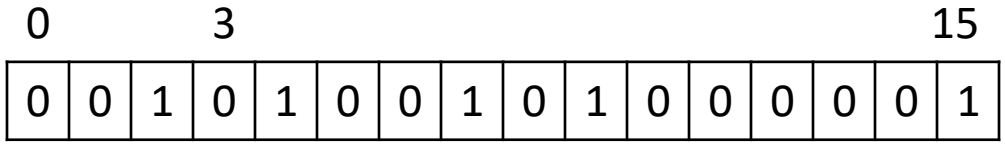


Figure 3.5 Example of Program Execution
(contents of memory and registers in hexadecimal)

Example of Program Execution



- Store AC to memory
  - Opcode: 0x2
  - Operand (Address): 0x941



- PC: program counter
- AC: accumulator
- IR: instruction register

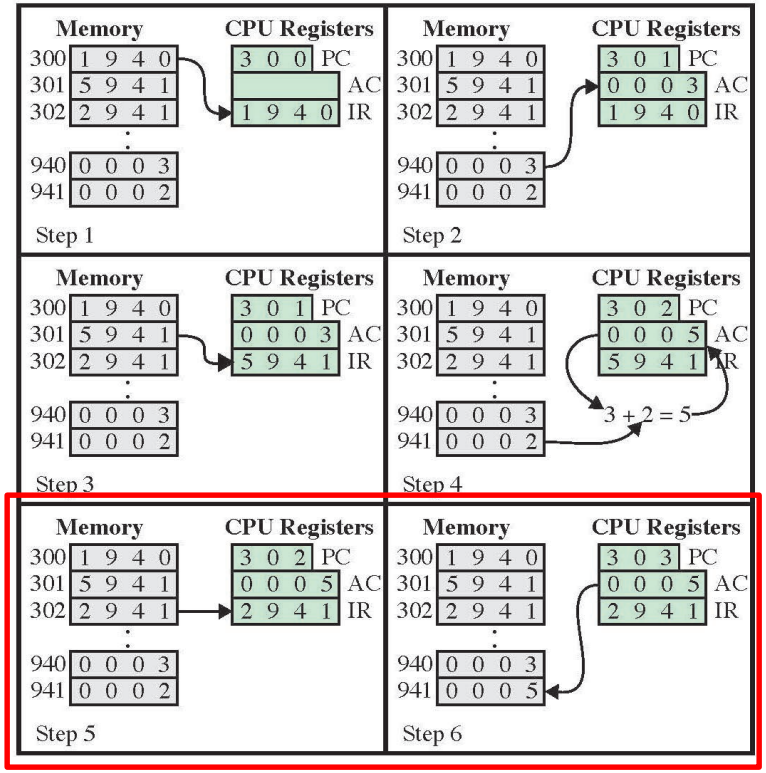
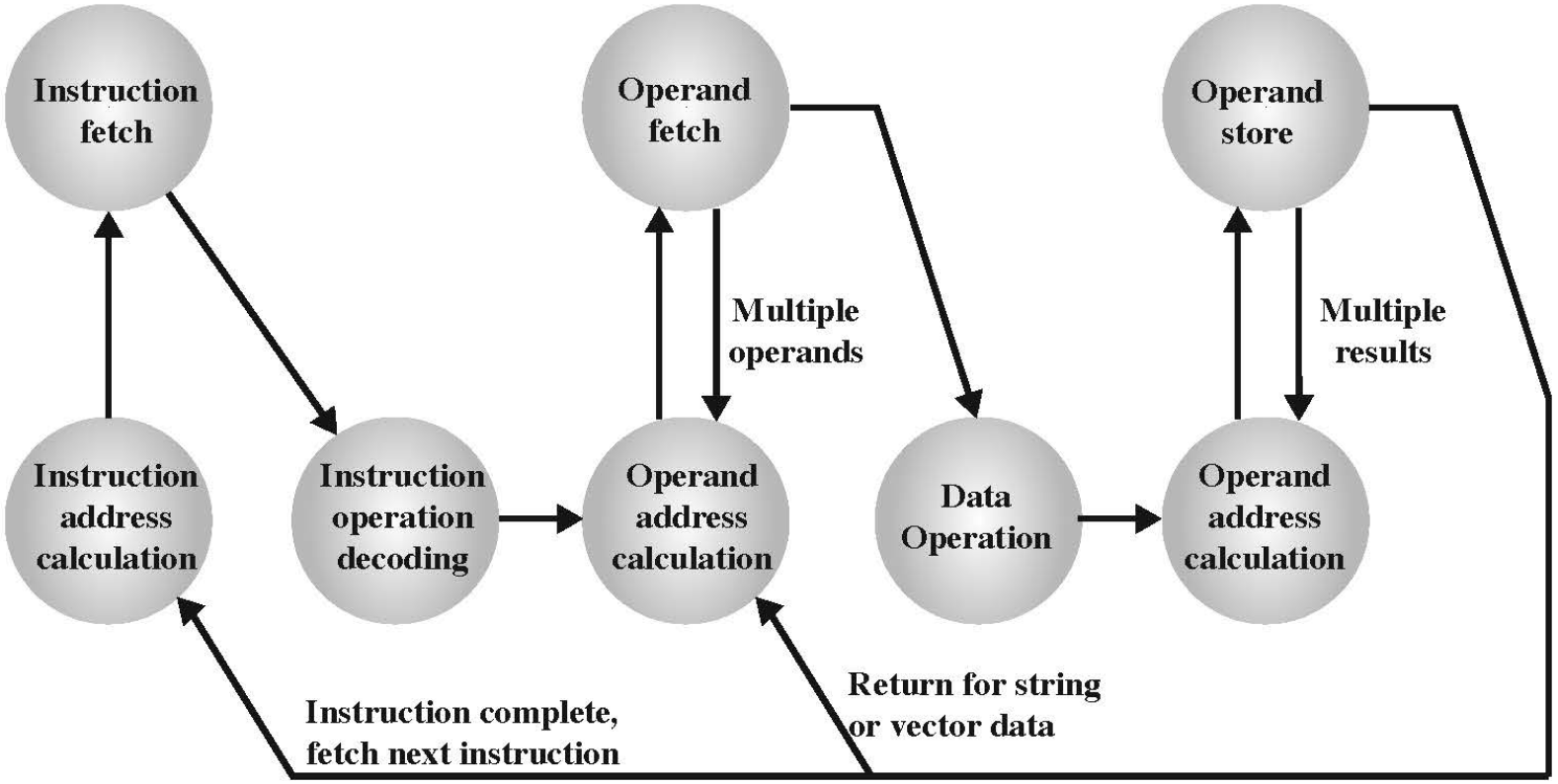


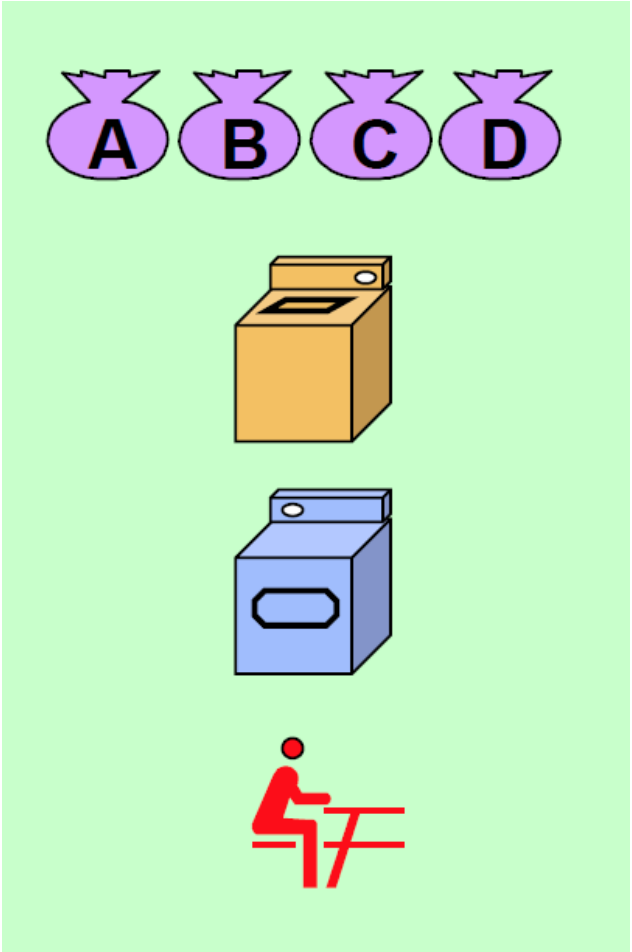
Figure 3.5 Example of Program Execution
(contents of memory and registers in hexadecimal)

Instruction Cycle State Diagram

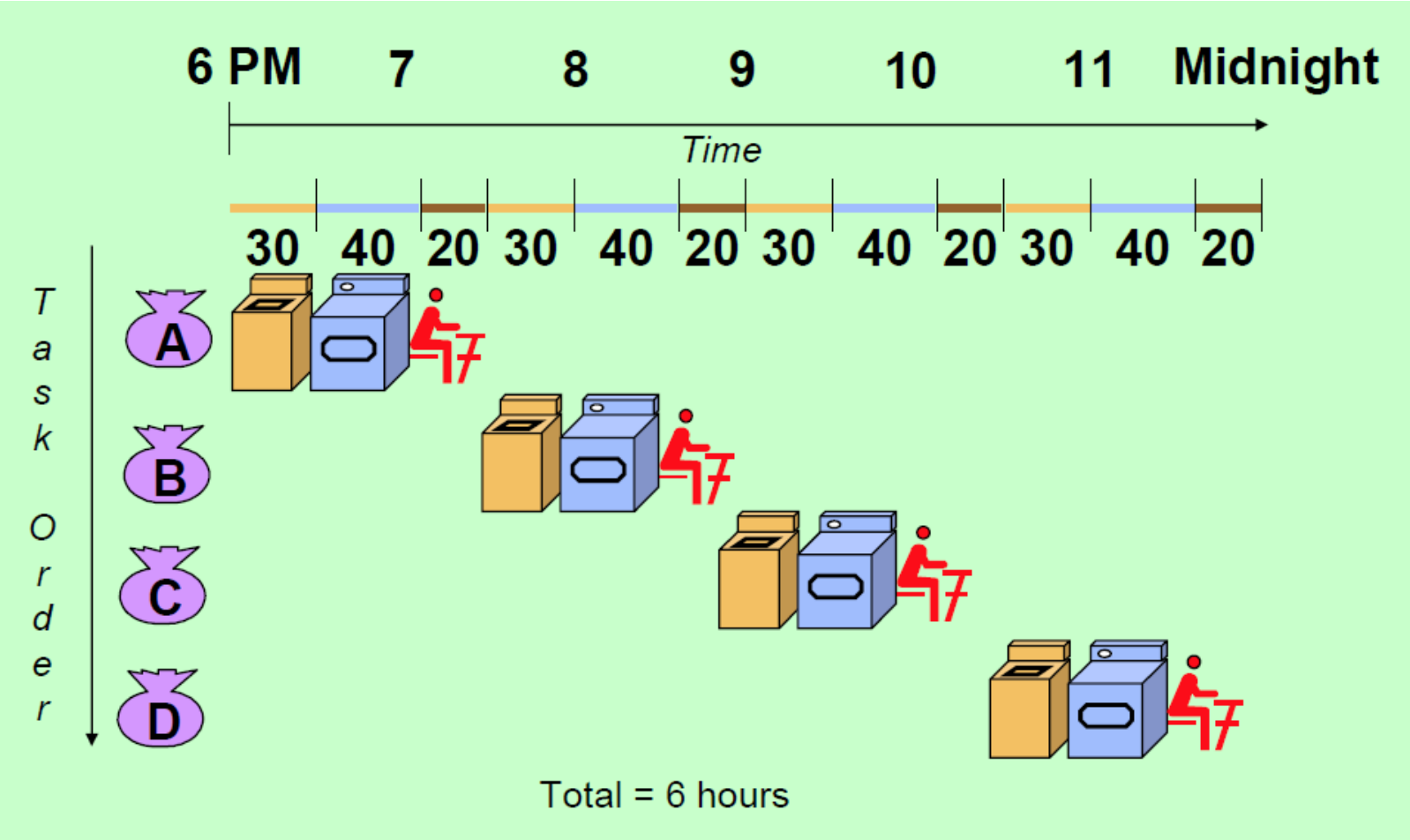


What is Pipeline?

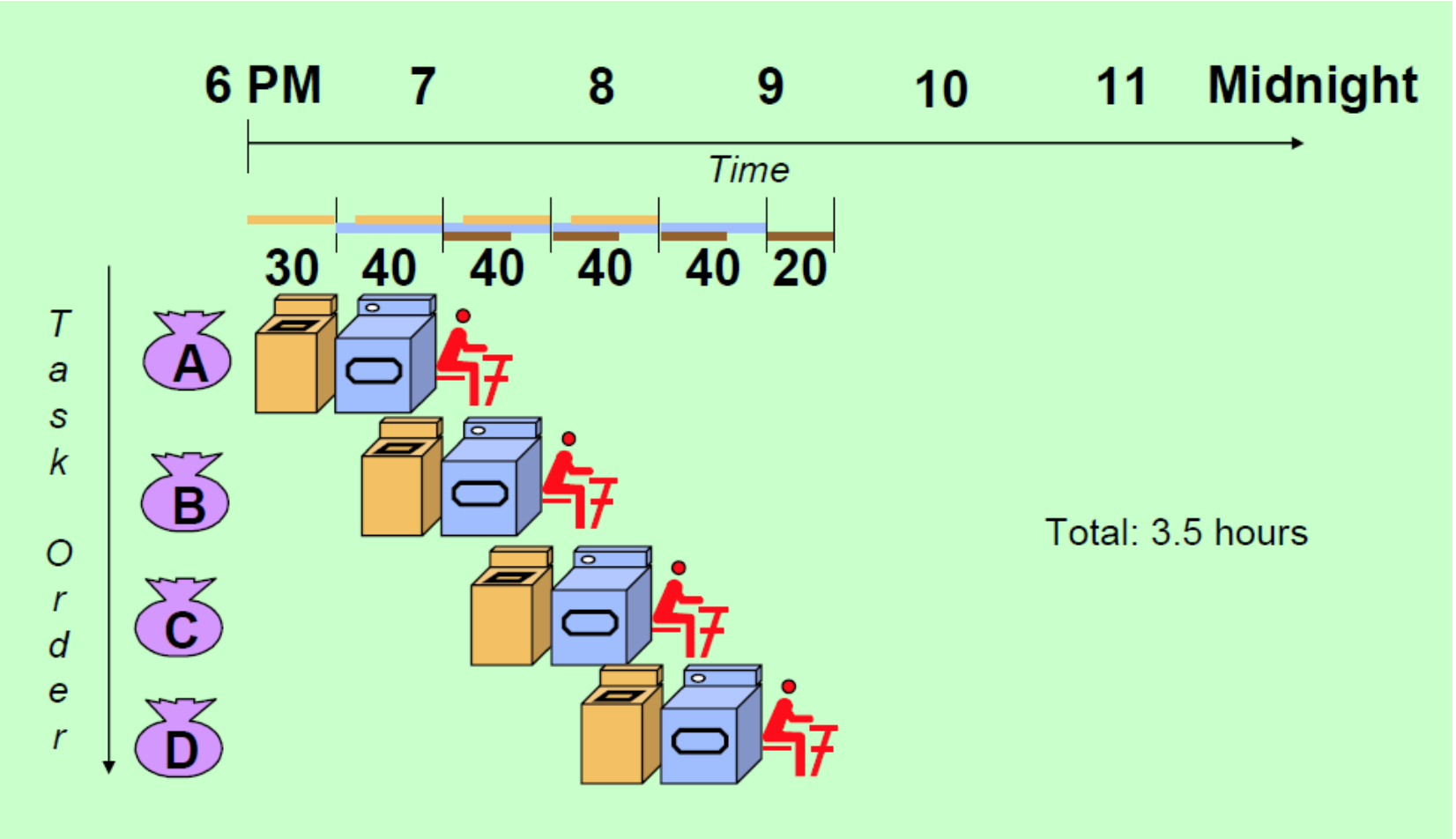
- Laundry Example
  - Ann, Brian, Cathy, Dave, each has one load of clothes to wash, dry, and fold
    - Washer takes 30 minutes
    - Dryer takes 40 minutes
    - Folding takes 20 minutes



Sequential Laundry



Pipelined Laundry: Start ASAP



Pipelining

- Overlap the executions of multiple instructions
  - Fetch Instruction (FI)
  - Decode Instruction (DI)
  - Calculate Operands (CO)
    - Or Effective Address (EA)
  - Fetch Operands (FO)
  - Execute Instructions (EI)
  - Write Operand (WO)

Timing Diagram for Instruction Pipeline Operation

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

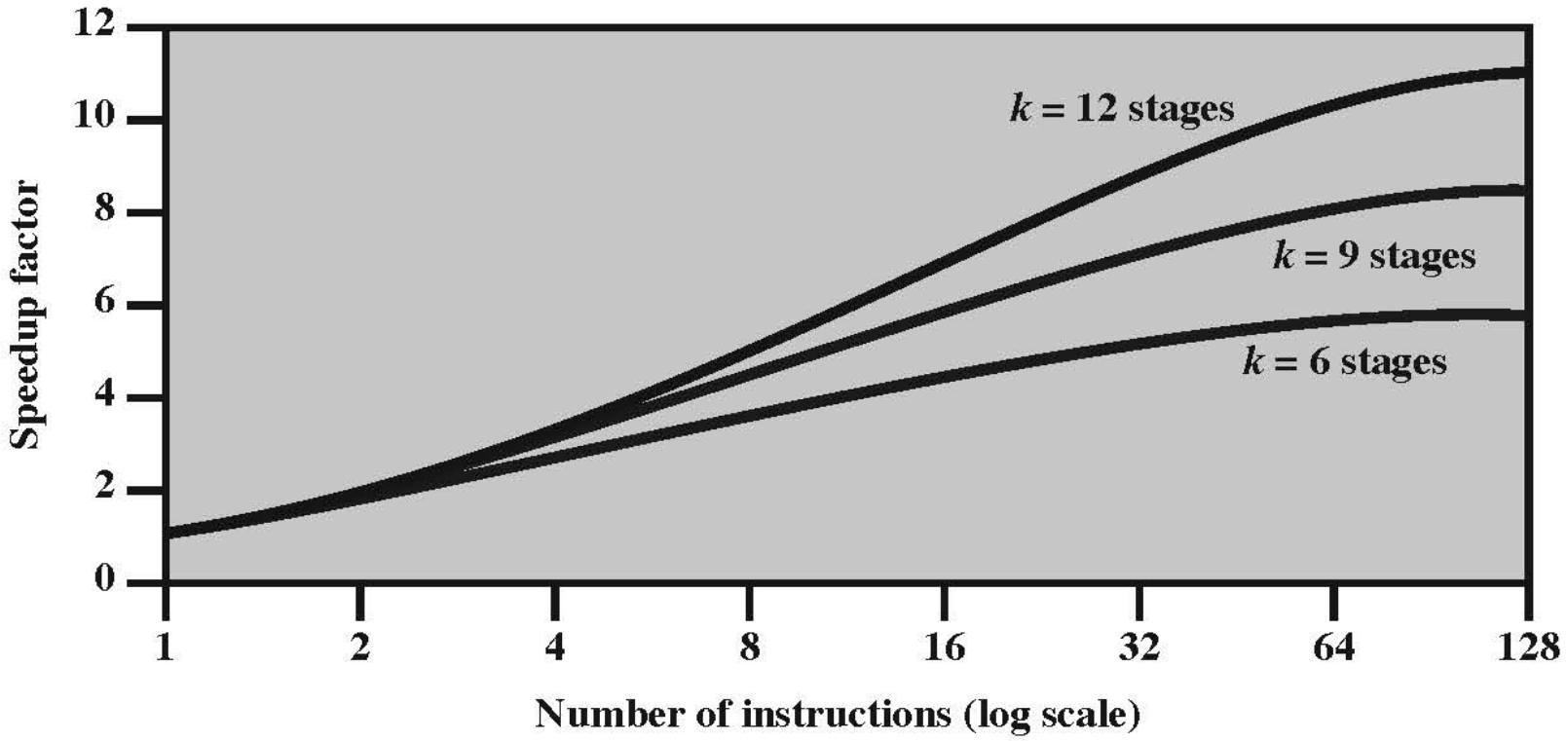
Pipeline Performance

- Speedup factor
  - Assume…
    - An instruction takes k stages
    - Each stage takes 1 clock cycle
  - T_1
    - Without pipeline, a program with n instructions will take nk cycles
  - T_k
    - With a k-stage pipeline, only the first instruction takes k cycles and the rest of them take only 1 cycle
  - Therefore…

$$S_k = \frac{T_1}{T_k} = \frac{nk}{k + (n-1)} \approx k$$

    - Where $n \to \infty$

Pipeline Performance



Speedup Ratio Example

- A non-pipeline system takes 100ns to process a task
- The same task can be processed in a 5 segment pipeline into 20ns each
- Determine the speedup ratio of the pipeline for 1000 tasks?

$$S_k = \frac{T_1}{T_k} = \frac{nk}{k + (n-1)} = \frac{1000 * 5}{5 + (1000 - 1)}$$
$$= 4.98$$

Pipeline Hazards

- Pipeline Hazards
  - Caused when some portion (or all) of the pipeline must stall
  - a.k.a., “pipeline bubble”
- Types of hazards
  - Resource
    - Two (or more) instructions in pipeline need same resource
    - e.g., both instructions need multipliers
  - Data
    - Data dependency
    - e.g., one instruction needs the results from the previous one that has not been available yet
  - Control
    - The next instruction is not at PC+1
    - e.g., branch, procedure call, and return

Data Hazard Example

MIPS Assembly	1	2	3	4	5	6	7	8	9	10
add r16, r1, r2	FI	DI	FO	EI	WO					
sub r15, r16, r3		FI	DI	IDLE	IDLE	FO	EI	WO		
I3			FI			DI	FO	EI	WO	
I4						FI	DI	FO	EI	WO

- Read After Write (RAW) (True Dependency)
  - An instruction modifies a register or memory location
  - Succeeding instruction reads data in memory or register location
  - Hazard occurs if the read takes place before write operation is complete

Pipeline Principles

- The pipeline rate limited by **slowest** pipeline stage
- A pipeline is an operation of **multiple** tasks; run simultaneously
- Potential speedup = **number of pipe stages (k)**
- Sometimes, pipeline has to be stalled due to the pipeline hazard