Computer Organization and Architecture

Lecture 5: Control Unit and Control Sequence of Instructions

Murtadha Hssayeni, Ph.D.

m.hssayeni@uobabylon.edu.iq

Outlines

- The Requirements Placed on The Processor
- The Control Unit (CU)
- Instruction Execution Cycle
- CPU Instruction Pipelining
- Pipelining Performance
  - Examples
- Decoding Instructions
  - Examples

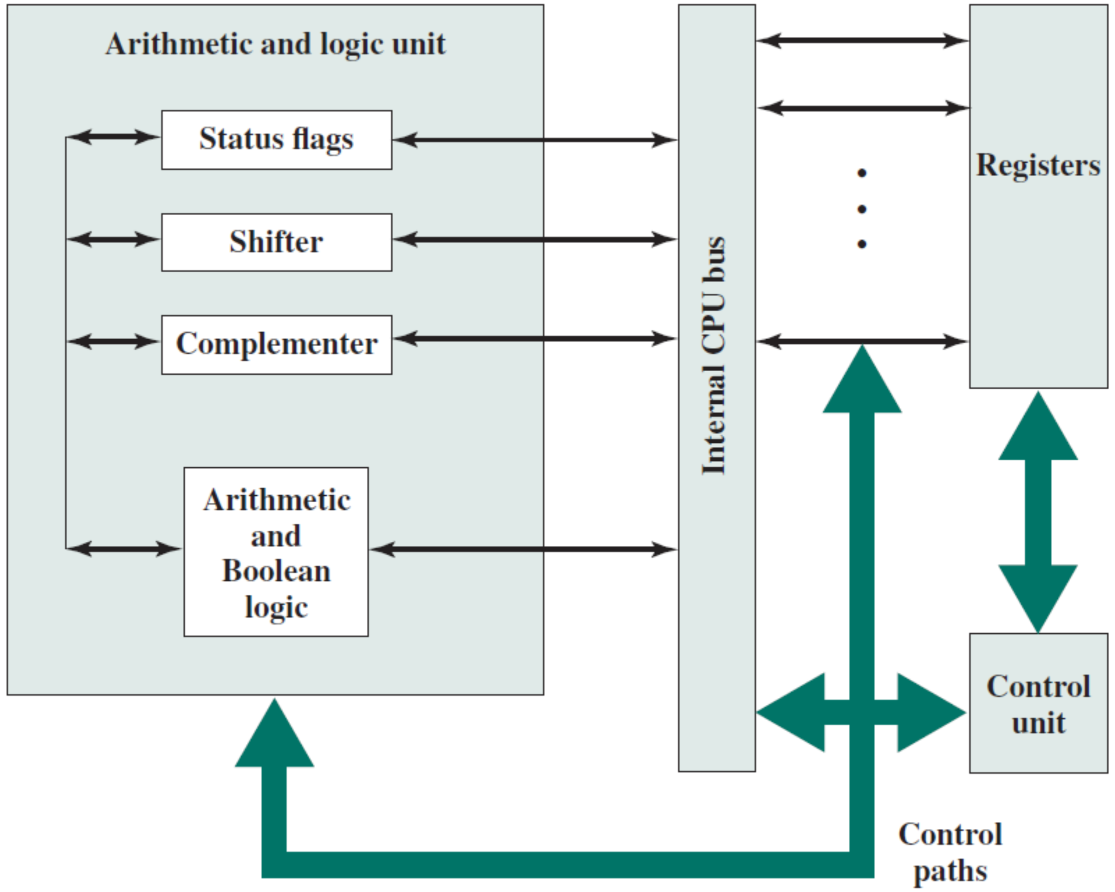
The Requirements Placed on The Processor

- **The operations that a CPU must do:**
  - **Fetch instruction**: The processor reads an instruction from memory (register, cache, main memory).
  - **Interpret instruction**: The instruction is decoded to determine what action is required.
  - **Fetch data**: The execution of an instruction may require reading data from memory or an I/O module.
  - **Process data**: The execution of an instruction may require performing some arithmetic or logical operation on data.
  - **Write data**: The results of an execution may require writing data to memory or an I/O module.

The Control Unit (CU)

- ❑CU is part of the CPU. Its role is:
  - ❑ It **controls the movement of data** and instructions into and out of the processor.
  - ❑It **coordinates** the sequencing of steps involved in **executing machine instructions including pipelining.**

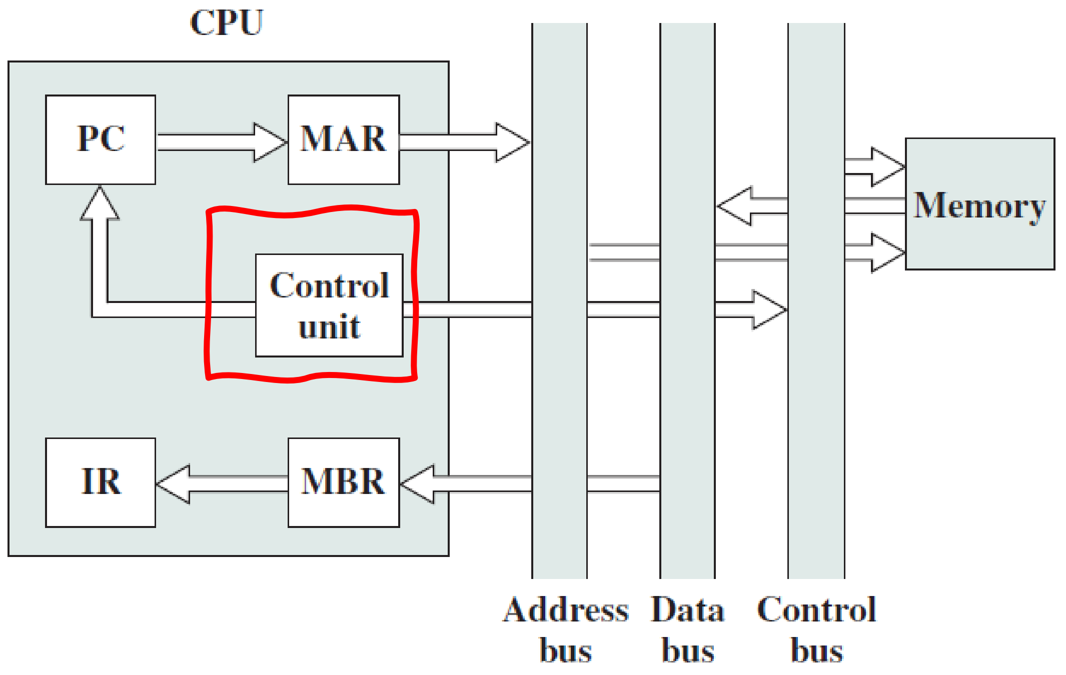
Internal Structure of the CPU:



CU controls the movement of data and instructions

1. The **program counter (PC)** contains the address of the next instruction to be fetched.
   - This address is mapped to the **Memory Address Register(MAR)** and placed on the address bus.
2. The **control unit** requests a memory read, and the result is placed on the data bus and copied into the **Memory Data Register (MDR)** and then moved to the **Instruction Register (IR)**.
3. Meanwhile, the PC is incremented by 1, preparatory for the next fetch.
4. Once the fetch cycle is over, the **control unit** examines the contents of the IR to determine if it contains an operand specifier using indirect addressing.

Data Flow: Fetch Cycle



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Instruction Execution Cycle

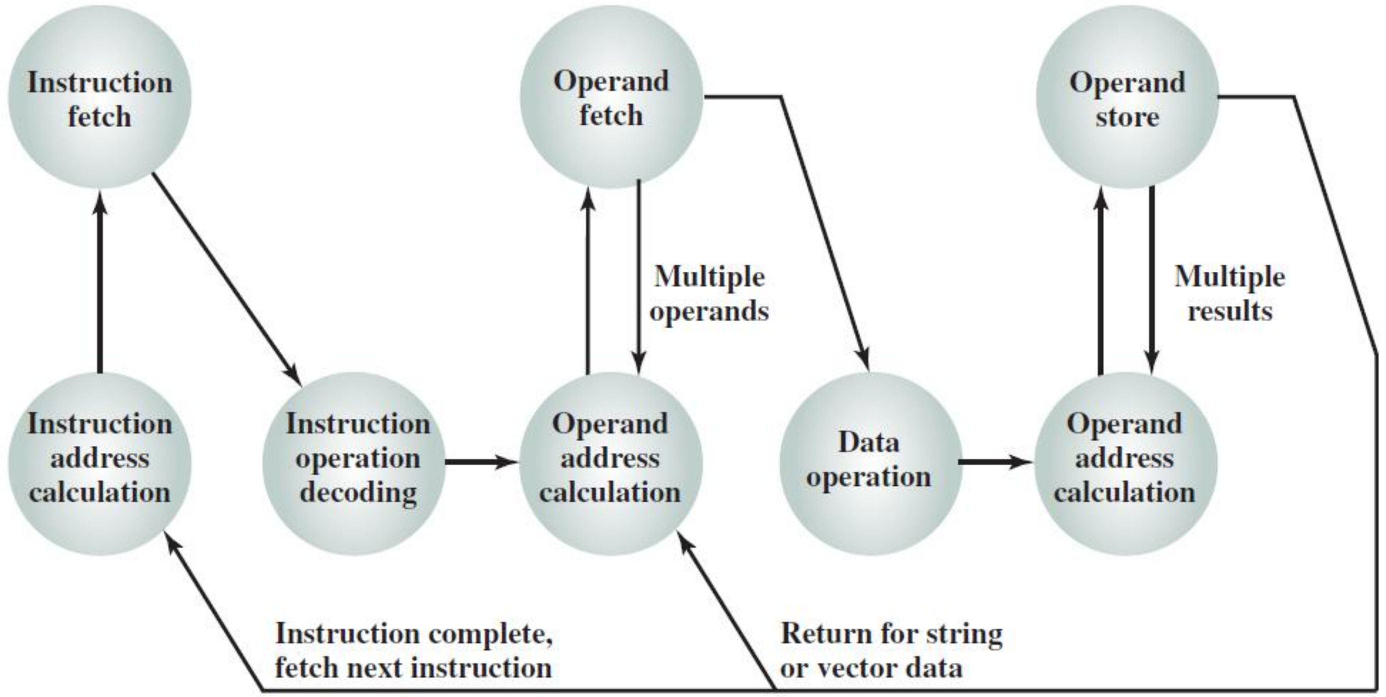
- ❑ CPU can have different cycles based on different instruction sets, but will be similar to the following cycle:

1. **Fetch Stage**:
   - The next instruction is fetched from the memory into the Instruction Register (IR).
2. **Decode Stage:**
   - During this stage, the encoded instruction presented in the instruction register is interpreted by the decoder as part of the control unit.
3. **Execute Stage:**
   - **The control unit of the CPU passes the decoded information** as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction.
   - If the ALU is involved, the result generated by the operation is stored in the main memory or sent to an output device.

Instruction Execution Diagram

- **Instruction Cycle State Diagram**

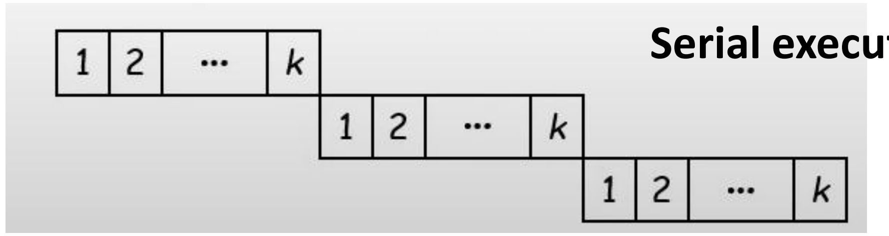
- Other states are added to this diagram such as the interrupt check and interrupt processing.



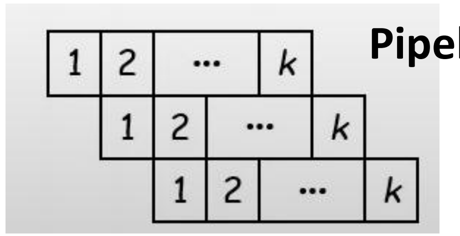
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

CPU Instruction Pipelining

- **instruction pipelining** is dividing the execution of instruction up into multiple phases, and executes separate instructions in each phase simultaneously**.**
- Consider a **task** that can be divided into **k subtasks**
  - The k subtasks are executed on **k different phases**
  - Each subtask requires one time unit



- **Pipelining is to overlap the execution**
  - The k phases work in parallel on k different tasks
  - Tasks enter/leave pipeline at the rate of one task per time unit



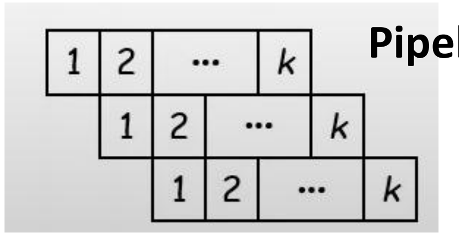
- **Clocked registers** are used to save the output from each stage.

Pipelining Performance

- **Pipelining is to overlap the execution**
  - The k phases work in parallel on k different tasks
  - Tasks enter/leave pipeline at the rate of one task per time unit
- A pipeline can process **n instructions** in (**k + (n – 1)) cycles**
  - k cycles are needed to complete the first instruction
  - n - 1 cycles are needed to complete the remaining n - 1 instructions
- The **speedup factor** for the instruction pipeline compared with serial execution:

$$\boldsymbol{Speedup\ factor}\ (\mathbf{S_k}) = \frac{\boldsymbol{n\ k}}{\boldsymbol{k + (n-1)}}$$

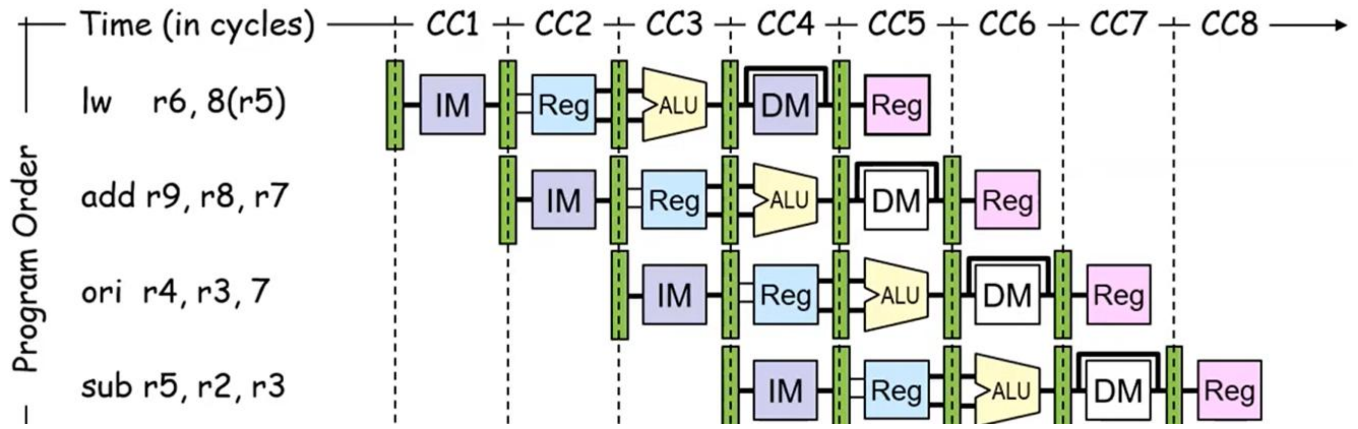
$S_k \rightarrow k$ for large n



Pipelined execution

Pipelining Performance: Example 1

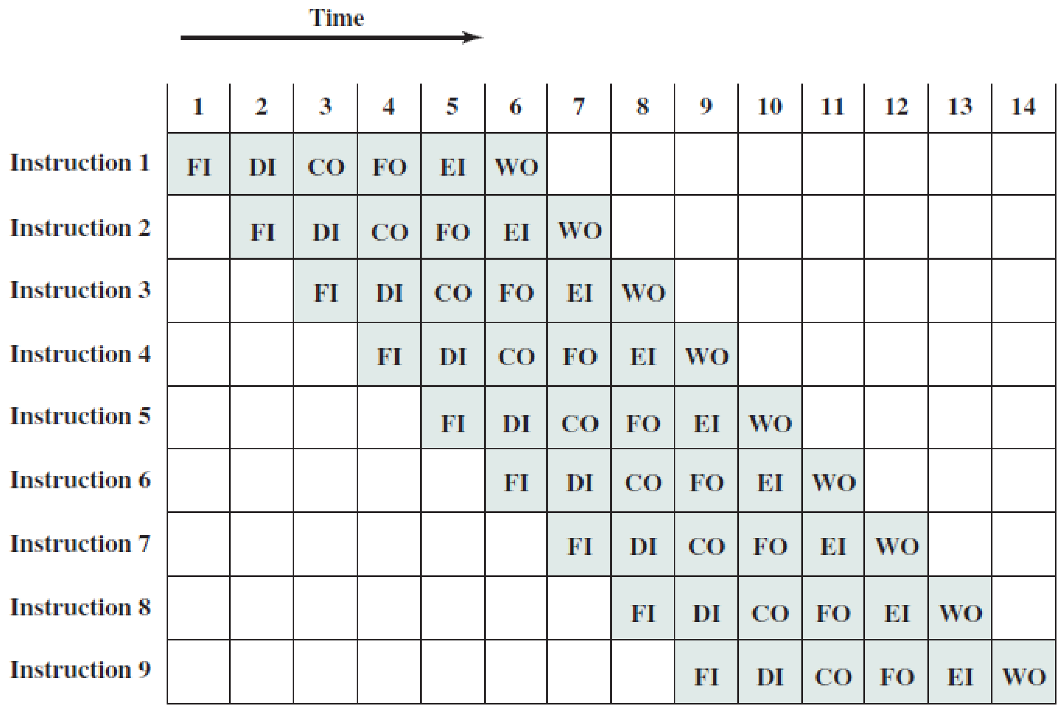
- Q: consider the following decomposition of the instruction processing:
  1. IF: Instruction Fetch from instruction memory
  2. ID: Instruction Decode
  3. EX: Execute operation
  4. MEM: Memory access for load and store only
  5. WB: Write Back result to register
- Draw a five-stage pipeline to execute 4 instructions, find the number of cycles required to execute them, and the speedup factor.



- n = 4 instructions, k = 5 stages
- Number of cycles required = (k+n-1) = 8
- Speedup factor = n x k / (k+n-1) = 20 / 8 = 2.5

Pipelining Performance: Example 2

- Q: consider the following decomposition of the instruction processing:
  1. Fetch instruction (FI): Read the next expected instruction into a buffer.
  2. Decode instruction (DI): Determine the opcode and the operand specifiers.
  3. Calculate operands (CO): Calculate the effective address of each source operand.
  4. Fetch operands (FO): Fetch each operand from memory.
  5. Execute instruction (EI): Perform the operation and store the result,
  6. Write operand (WO): Store the result in memory.
- Draw a six-stage pipeline to execute 9 instructions, find the number of cycles required to execute them, and the speedup factor.

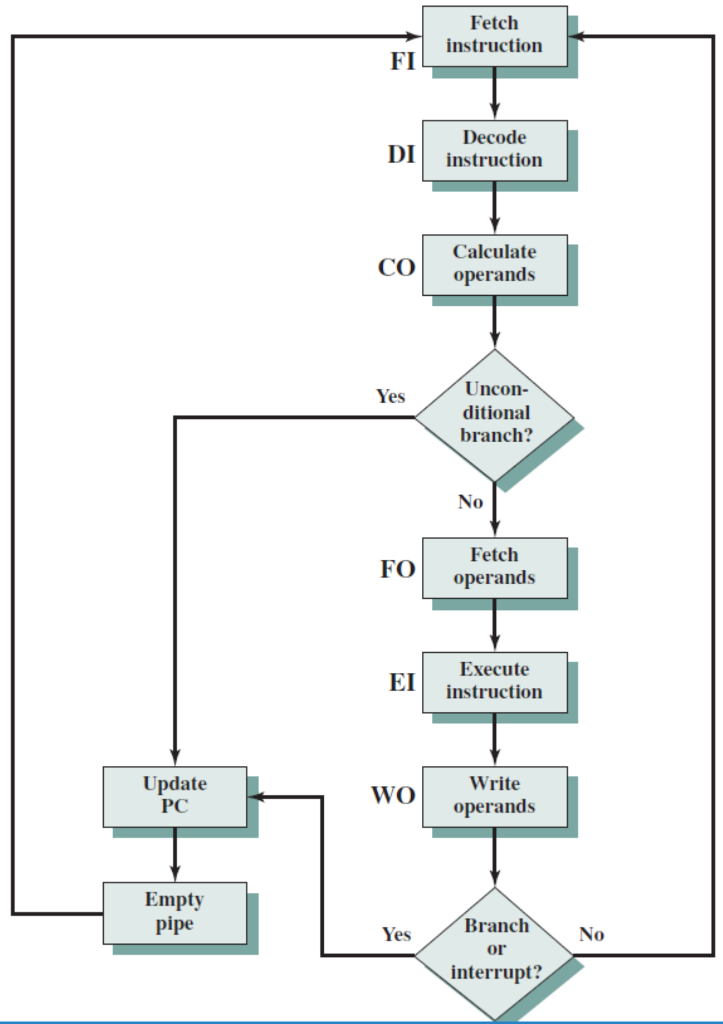


	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO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

- n = 9 instructions, k = 6 stages
- Number of cycles required = (k+n-1) = 14
- Speedup factor = n x k / (k+n-1) = 54 / 14 = 3.86
- Stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.

Several factors that limit the performance of Pipelining

1. If the **stages are not of equal duration**, there will be some waiting involved at various pipeline stages.
2. The **conditional branch instruction or interrupts**, which can invalidate several instruction fetches.

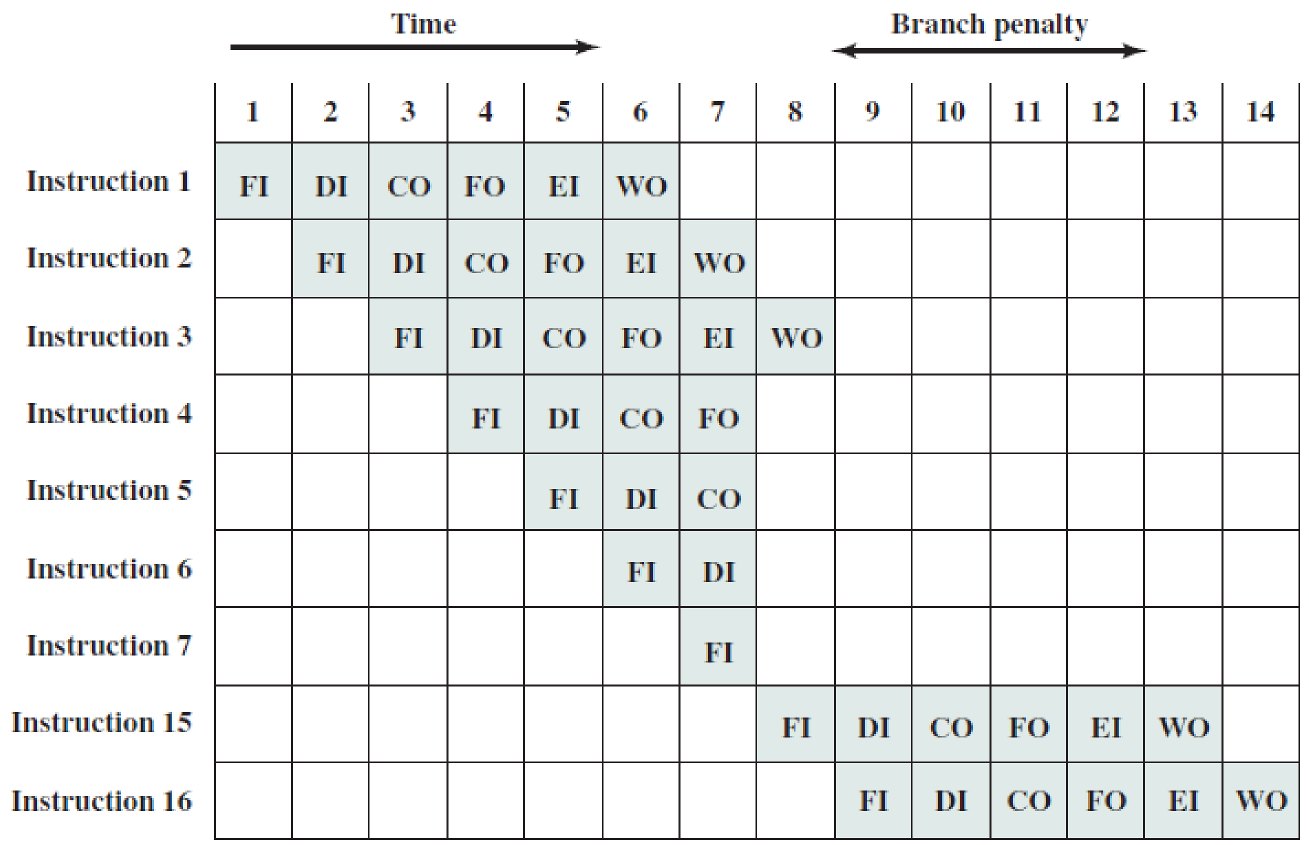


Six-Stage instruction pipeline with a branch or interrupt

Several factors that limit the performance of Pipelining (cont.)

- ❑Example on the effect of a conditional branch on instruction pipeline operation:
  - ❑Assume Instruction 3 is a conditional branch to instruction 15.
- ❑No instructions are completed during time units 9 through 12.
  - ❑This is the performance penalty.

A jump to instruction 15 from instruction 3


Time
Branch penalty

	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							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

Decoding Instructions

- **Decoding instructions** is interpreting each line of code to determine what <u>operation</u> needs to be performed and what <u>operands</u> are involved.
- Decode the following instruction:

SUB R1, [025F2H]

  - Opcode: sub
  - Source 1: R1
    - Mode 1: register addressing
  - Source 2: [025F2H]
    - Mode 2: direct addressing
  - Destination: R1

- Decode the following instruction:

ADD R1, [R2]

  - Opcode: ADD
  - Source 1: R1
    - Mode 1: register addressing
  - Source 2: [R2]
    - Mode 2: register indirect addressing
  - Destination: R1

Decoding Instructions: Cont.

- Decode the following instruction:

JMP labX

- Opcode: JMP
- Source: PC
  - Mode: relative addressing
- Destination: PC

- Decode the following instruction:

MOV [0A120H], [R1]

- Opcode: MOV
- Source: [R1]
  - Mode: register indirect addressing
- Destination: [0A120H]
  - Mode: direct addressing