Processor Structure and Function

Contents of Lecture:

- Processor Organization
- Register Organization
  - ✓ User-Visible Registers
  - ✓ Control and Status Registers
- Instruction Cycle
- Pipelining Strategy

References for This Lecture:

- ✓ William Stallings, Computer Organization and Architecture Designing For Performance, 9th Edition, Chapter 14: *Processor Structure and Function*

Processor Organization:

Processor Requirements: the things that it 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

In order to do these things the processor needs to store some data temporarily and therefore needs a small internal memory

Register Organization:

- Within the processor there is a set of registers that function as a level of memory above main memory and cache in the hierarchy. The registers in the processor perform two roles:

  - ✓ **User-visible registers**: Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.

  - ✓ **Control and status registers**: Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

User-Visible Registers:

- A user-visible register is one that may be referenced by means of the machine language that the processor executes.
- We can characterize these in the following categories:
  - ✓ General purpose
  - ✓ Data
  - ✓ Address
  - ✓ Condition codes

Control and Status Registers:

Four registers are essential to instruction execution:

- **Program counter (PC):** Contains the address of an instruction to be fetched
- **Instruction register (IR):** Contains the instruction most recently fetched
- **Memory address register (MAR):** Contains the address of a location in memory
- **Memory buffer register (MBR):** Contains a word of data to be written to memory or the word most recently read

Instruction Cycle:

- To recall, an instruction cycle includes the following stages:
  - ✓ **Fetch**: Read the next instruction from memory into the processor.
  - ✓ **Execute**: Interpret the opcode and perform the indicated operation.
  - ✓ **Interrupt**: If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.

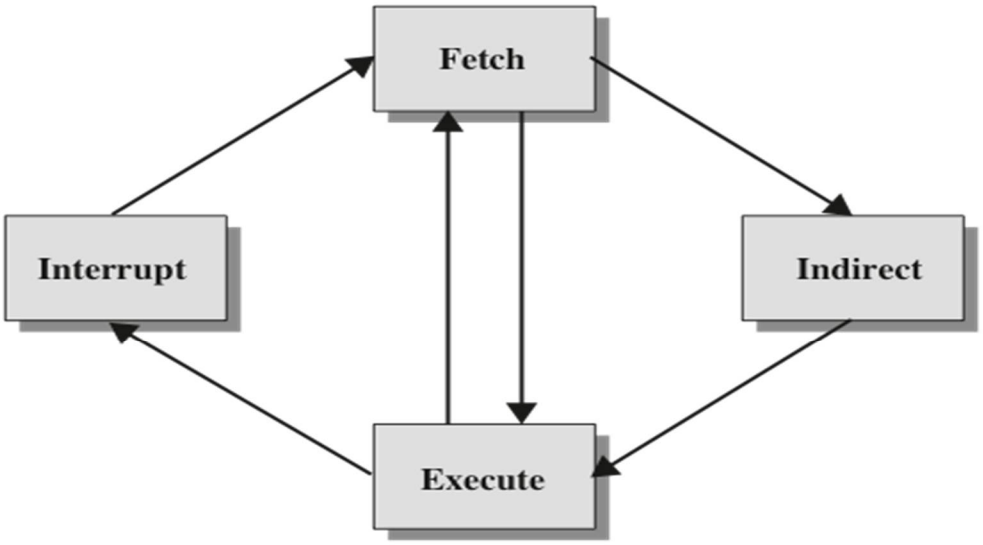

Figure 14.4 The Instruction Cycle

❖ Fetch Cycle:

- ✓ During the fetch cycle, an instruction is read from memory. The PC contains the address of the next instruction to be fetched.
- ✓ This address is moved to the MAR and placed on the address bus.
- ✓ The control unit requests a memory read, and the result is placed on the data bus and copied into the MBR and then moved to the IR. Meanwhile, the PC is incremented by 1, preparatory for the next fetch.
- ✓ Following figure shows the flow of data during this cycle.

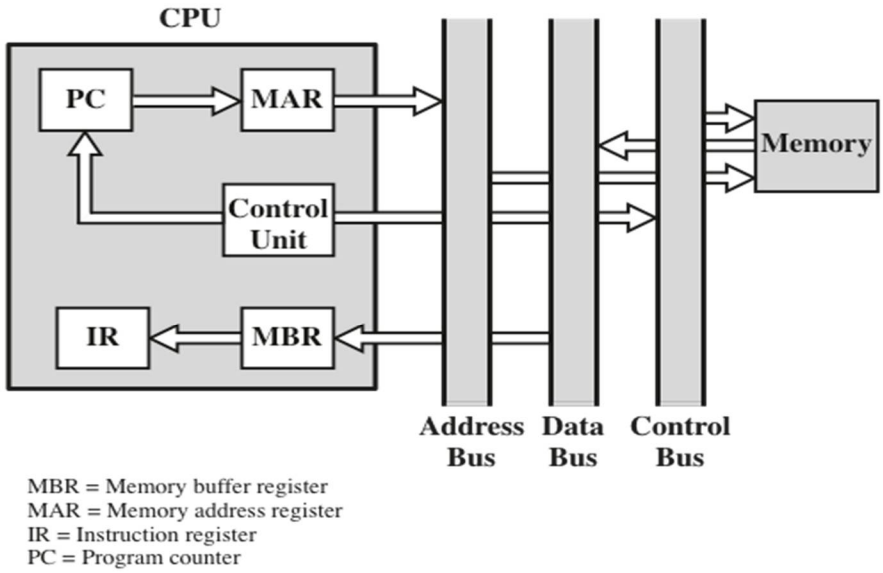

Figure 14.6 Data Flow, Fetch Cycle

❖ Indirect Cycle:

- ✓ 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.
- ✓ If so, an indirect cycle is performed, it is a simple cycle.

- ✓ The rightmost N bits of the MBR, which contain the address reference, are transferred to the MAR.

- ✓ Then the control unit requests a memory read, to get the desired address of the operand into the MBR.

- ✓ The fetch and indirect cycles are simple and predictable.
- ✓ As shown in following figure.

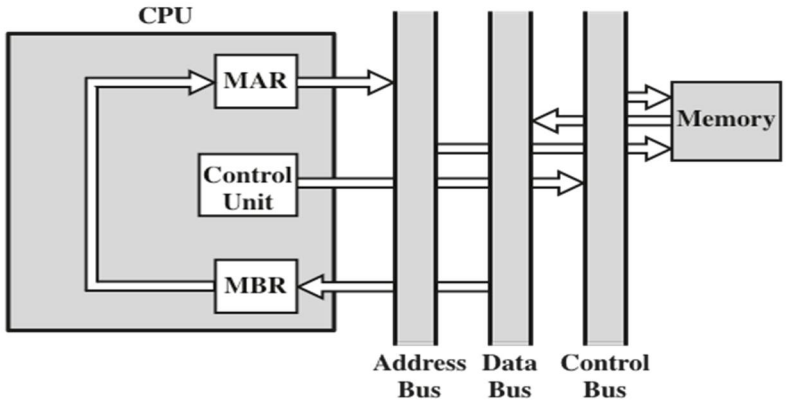

Figure 14.7 Data Flow, Indirect Cycle

❖ Execute cycle:

- ✓ The execute cycle takes many forms; the form depends on which of the various machine instructions is in the IR.
- ✓ This cycle may involve transferring data among registers, read or write from memory or I/O, and/or the invocation of the ALU.

❖ Interrupt Cycle:

- ✓ The interrupt cycle is simple and predictable.
- ✓ The current contents of the PC must be saved so that the processor can resume normal activity after the interrupt.
- ✓ Thus, the contents of the PC are transferred to the MBR to be written into memory.
- ✓ The special memory location reserved for this purpose is loaded into the MAR from the control unit.
- ✓ It might, for example, be a stack pointer.
- ✓ The PC is loaded with the address of the interrupt routine.
- ✓ As a result, the next instruction cycle will begin by fetching the appropriate instruction

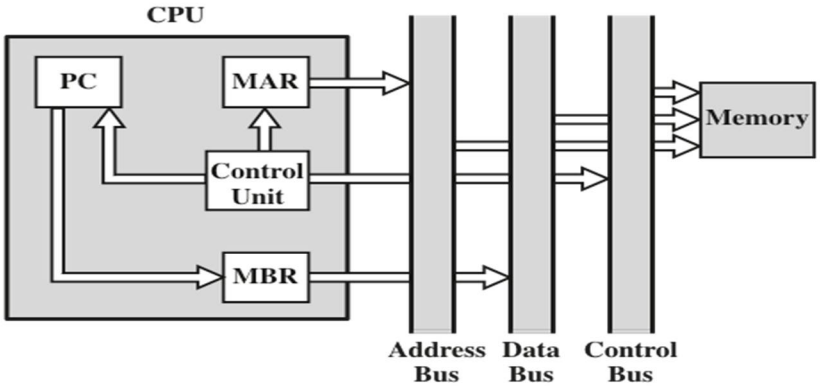

Figure 14.8 Data Flow, Interrupt Cycle

Pipelining Strategy:

- Similar to the use of an assembly line in a manufacturing plant
- New inputs are accepted at one end before previously accepted inputs appear as outputs at the other end
- To apply this concept to instruction execution we must recognize that an instruction has a number of stages

Two-Stage Instruction Pipeline:

- The pipeline has two independent stages:
  - ✓ The **first stage** fetches an instruction and buffers it.
  - ✓ When the **second stage** is free, the first stage passes it the buffered instruction.
  - ✓ While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction.
  - ✓ This is called instruction prefetch or fetch **overlap**.
  - ✓ Pipelining requires registers to store data between stages.

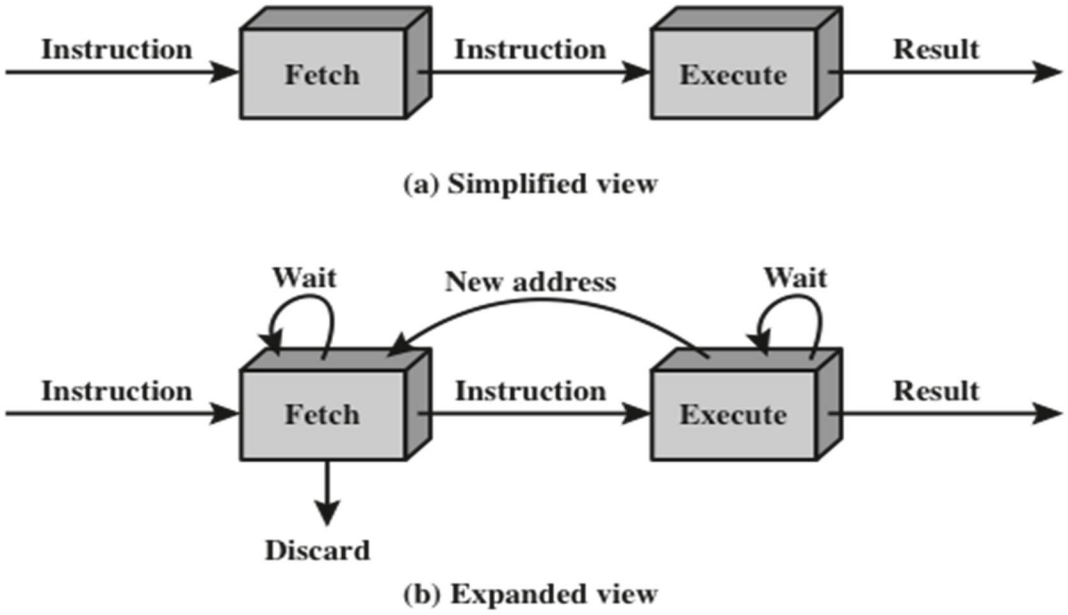

Figure 14.9 Two-Stage Instruction Pipeline

The decomposition of the instruction processing:

- **Fetch instruction (FI):** Read the next expected instruction into a buffer
- **Decode instruction (DI):** Determine the opcode and the operand specifiers
- **Calculate operands (CO):**
  - ✓ Calculate the effective address of each source operand
  - ✓ This may involve displacement, register indirect, indirect, or other forms of address calculation
- **Fetch operands (FO):**
  - ✓ Fetch each operand from memory
  - ✓ Operands in registers need not be fetched
- **Execute instruction (EI):**
  - ✓ Perform the indicated operation and store the result, if any, in the specified destination operand location
- **Write operand (WO):** Store the result in memory

Example :Performance:

- # of instructions n=9
- # of stages k=6
- The total time required to execute instructions without pipeline
  = k*n=6*9 =54 cycle
- The total time required to execute instructions with pipeline
  = k+(n-1)=6+8=14 cycle
- speed up= 54/14=3.86 = 4

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

Figure 14.10 Timing Diagram for Instruction Pipeline Operation

- Assume that instruction 3 is a conditional branch to instruction 15.
- Until the instruction is executed, there is no way of knowing which instruction will come next.

- The pipeline, in this example, simply loads the next instruction in sequence (instruction 4) and proceeds.

- During time unit 8, instruction 15 enters the pipeline. No instructions complete during time units 9 through 12; this is the performance penalty incurred because we could not anticipate the branch.

- **The Effect of a Conditional Branch on Instruction Pipeline Operation**

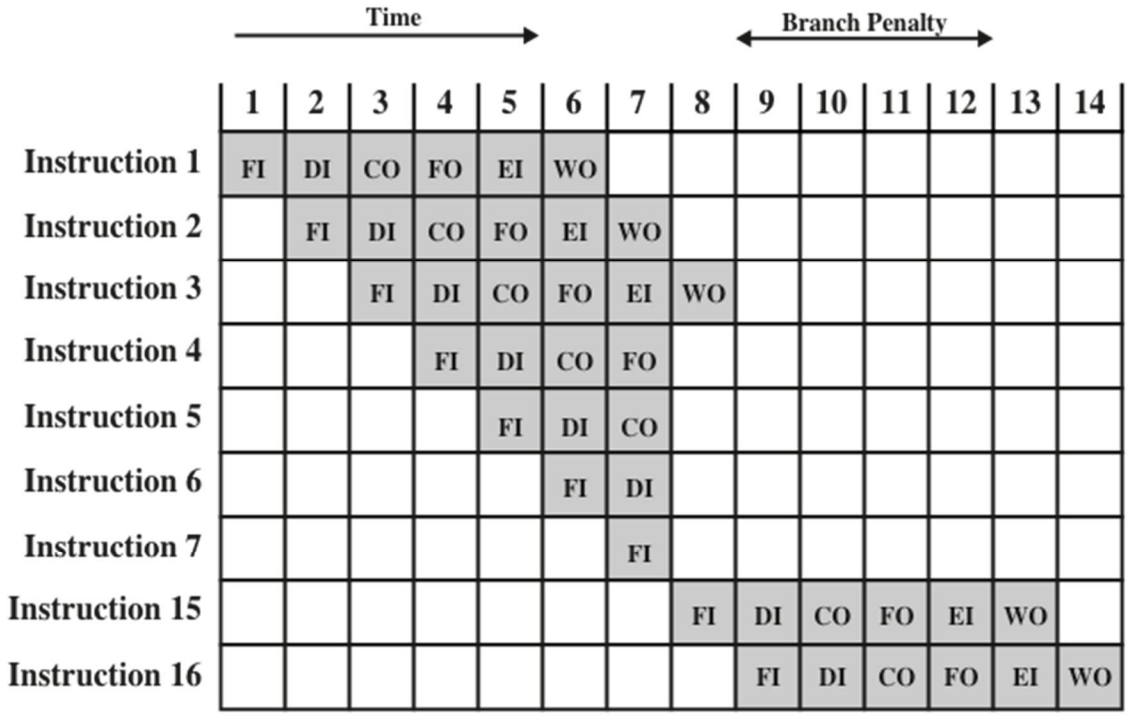

Figure 14.11 The Effect of a Conditional Branch on Instruction Pipeline Operation

❖ Six Stage Instruction Pipeline:

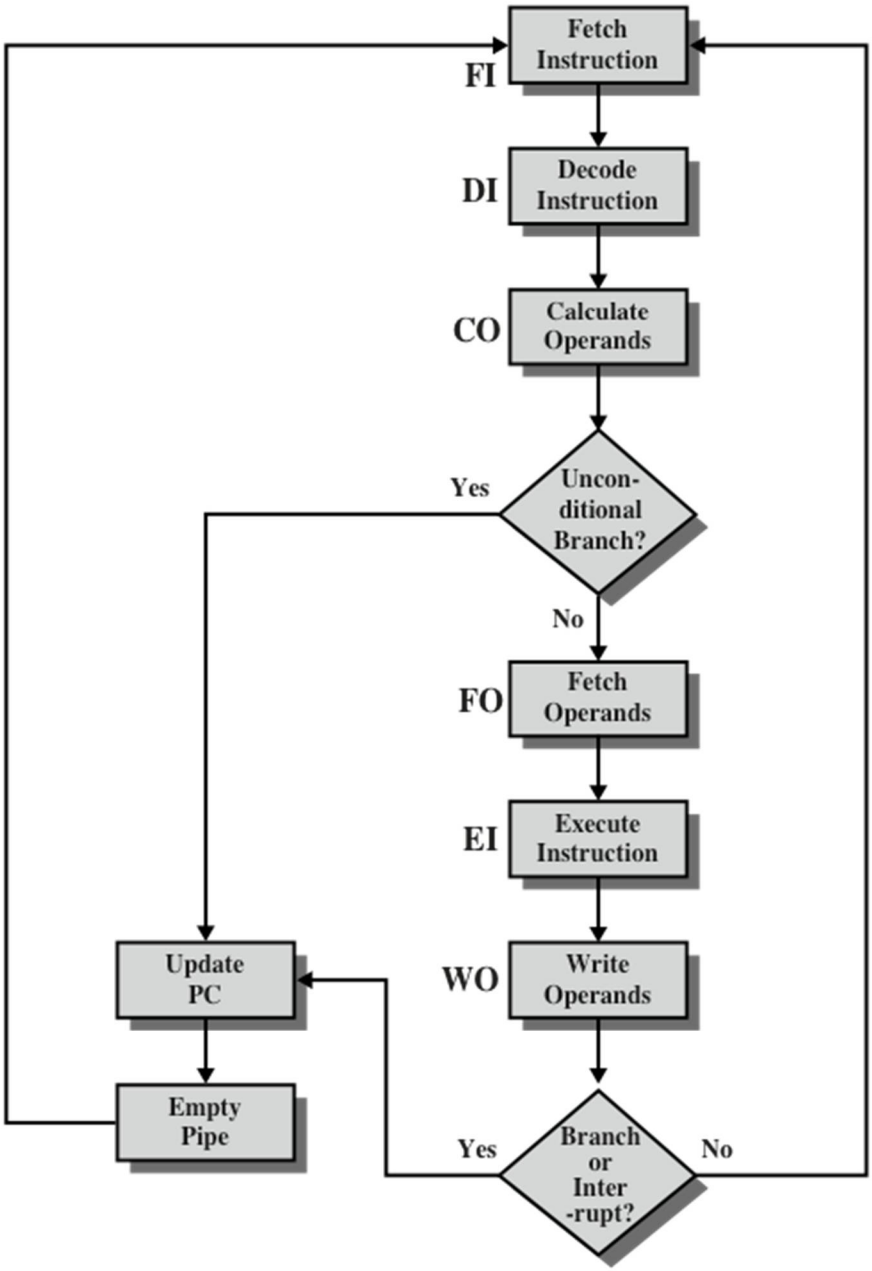

Figure 14.12 Six-Stage Instruction Pipeline