CIS11032 Logic Designing & Computer Organization

Lesson 05 Inner Workings of CPU

AR Fathima Shafana
Department of ICT
Faculty of Technology
South Eastern University of Sri Lanka

Learning Outcomes

At the completion of this lesson students should be able to,

- Identify the Inner workings and organization of computer
- Understand the Instruction Cycle
- Understand the Interrupts of Instructions

COURSE OUTLINE

- Computer Functions
- Instruction Cycle
- Interrupts

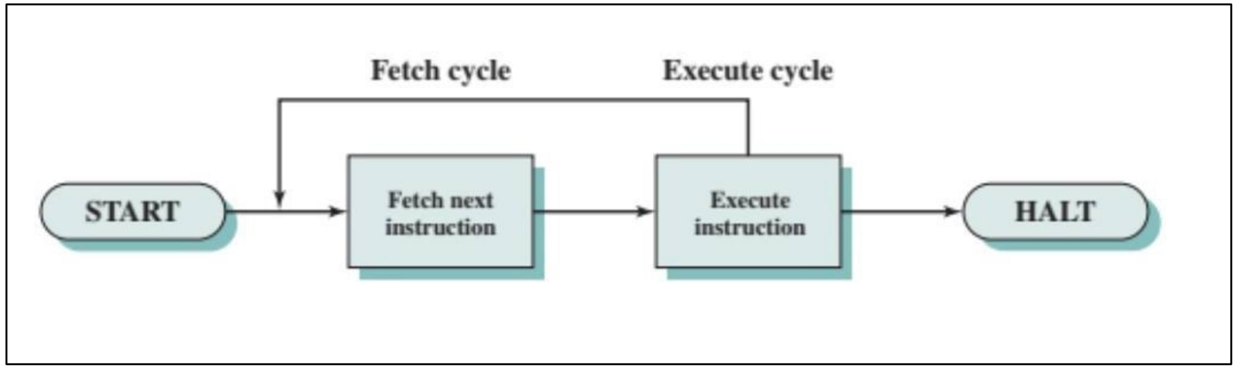
Computer Function

- The basic function of any computer is the execution of programs i.e., execution of instructions stored in a program.
- It is generally accomplished in two steps.
  - *The processor reads* (*Fetches*) *instructions from memory one at a time.*
  - *Executes each instruction.*
- The processing required for a single instruction is called an Instruction cycle

Fetch the instruction → Decode the instruction → Execute the instruction → Repeat

Instruction Cycle

- The basic instruction cycle consists of two basic steps known as **Fetch Cycle** and **Execute Cycle.**



- Program execution halts at the following instances.
  - **The machine is turned off**
  - **Occurrence of unrecoverable errors**
  - **Encounter of a program instruction that halts the computer**

What happens at the Instruction Cycle?

- The instruction cycle begins when the processor fetches an instruction from memory
- A register called the **Program Counter (PC)** holds the address of the instruction to be fetched next
- After each instruction fetch, PC is incremented for the other instruction fetch in the sequence.
- The fetched instruction is loaded into a register in the processor. **(Instruction Register)**
- The instruction contains bits that specify the action the processor is to take.
- The processor interprets the instruction and performs the required action.

Instruction Cycle contd.

Fetch: The Program Counter (PC) holds the address of the next instruction. The CPU fetches the instruction from memory at that address. The PC is then usually incremented to point to the next instruction.

Decode: The Control Unit examines the instruction and determines what operation to perform (e.g., add, load, jump), What operands are needed (registers, memory addresses).

Execute: The CPU performs the operation, If it’s an arithmetic operation, the ALU (Arithmetic Logic Unit) does the math. If it’s a memory operation, the CPU reads or writes data.

Store (sometimes considered part of execute): The result of the operation is stored in a register or memory.

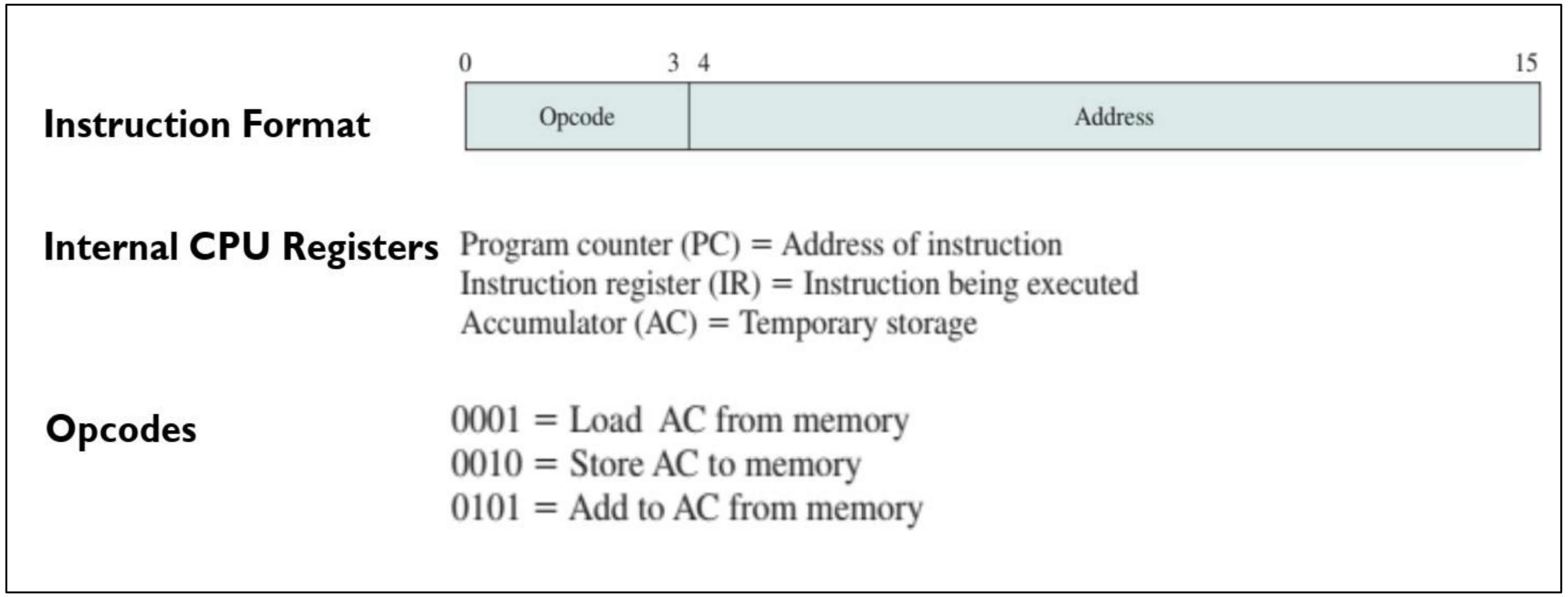
Instruction Cycle contd.

The above actions can fall into 04 categories.

1. Processor-memory: Data may be transferred from processor to memory or from memory to processor.
2. Processor-I/O: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
3. Data processing: The processor may perform some arithmetic or logic operation on data.
4. Control: An instruction may specify that the sequence of execution be altered.

EXAMPLE

- Consider a simple example using a *hypothetical machine* whose processor contains a single data register, called an accumulator (AC) . Both instructions and data are 16 bits long.



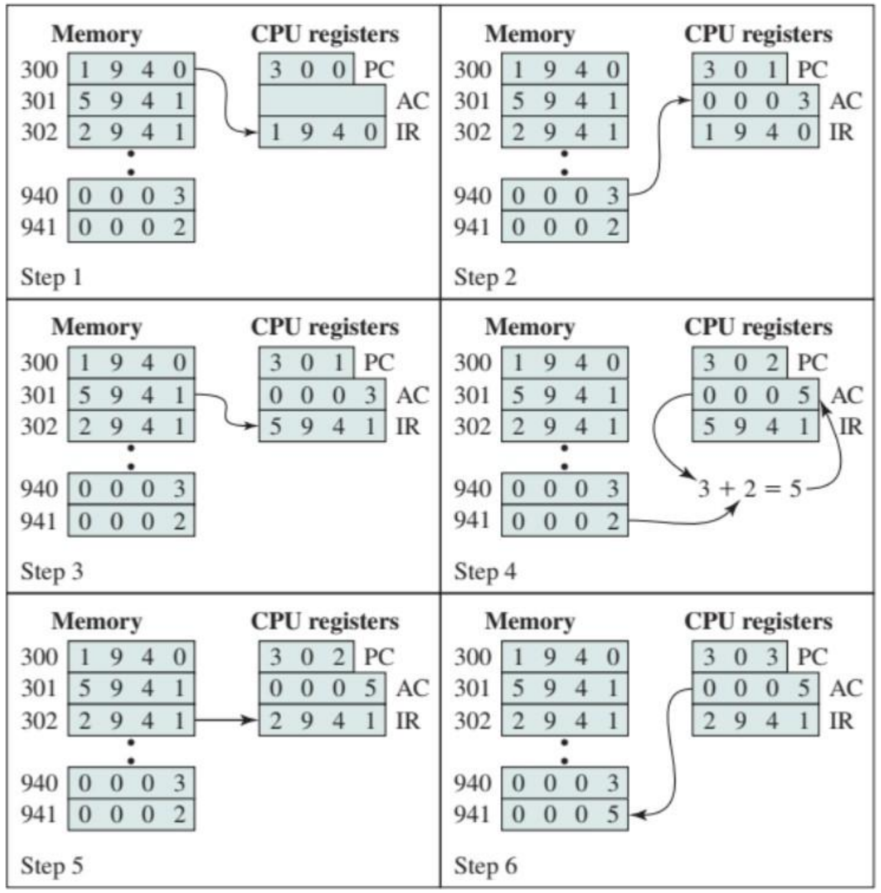
Internal CPU Registers

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

Opcodes

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

EXAMPLE contd.



The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the address 941.

The Process

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IR, and the PC is incremented.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hexadecimal digits) specify the address (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301, and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added, and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302, and the PC is incremented.
6. The contents of the AC are stored in location 941

State Diagram – Instruction Cycle

- The execution cycle for a particular instruction on such processors could involve more than one reference to memory. (E.g. *Older processors included instructions that contain more than one memory address*)
- Considering this, the following state diagram shows the more detailed view of the instruction cycle

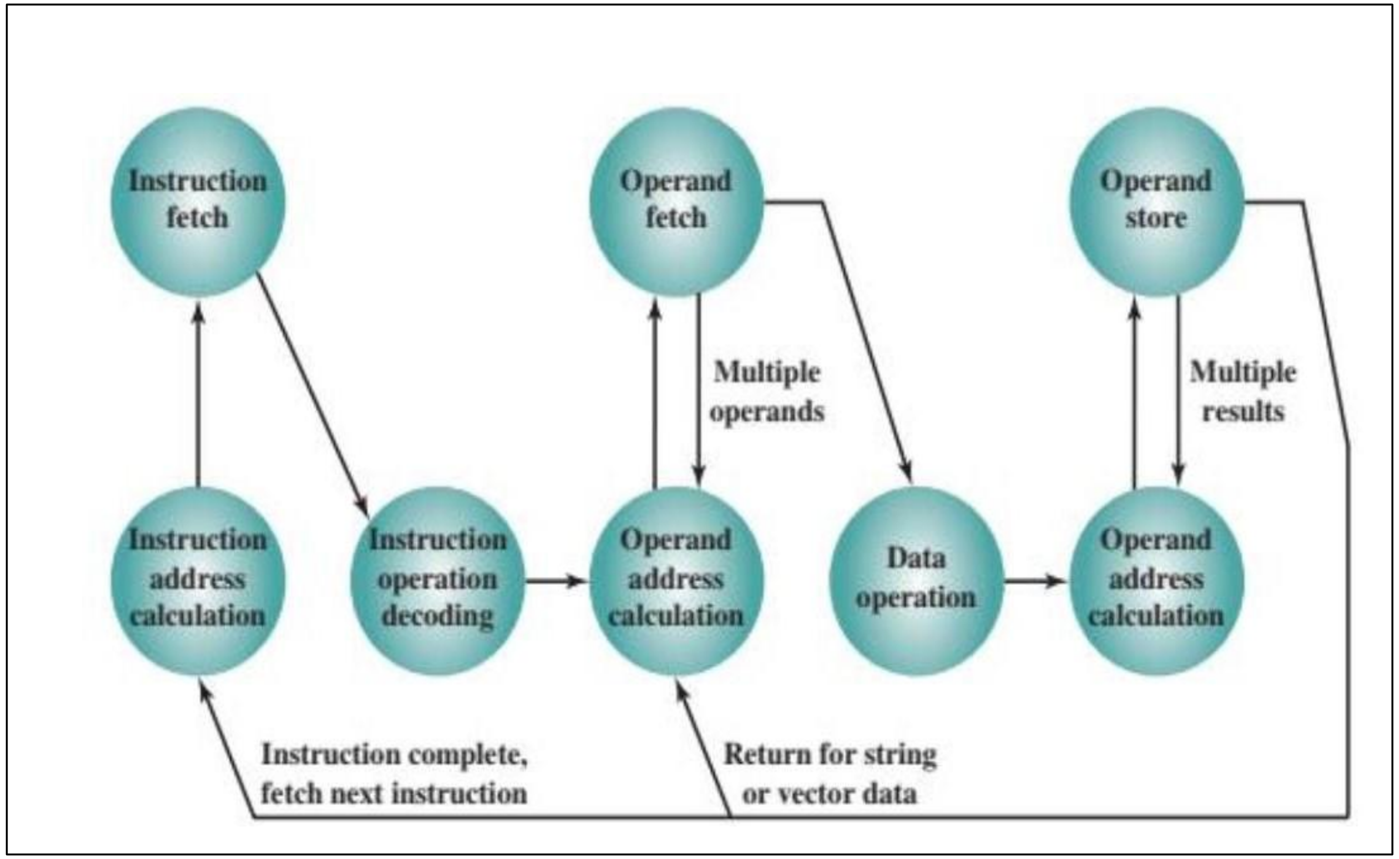
State Diagram – Instruction Cycle contd.

- Instruction address calculation (iac):
  - Determine the address of the next instruction to be executed.
  - Usually, this involves adding a fixed number to the address of the previous instruction
- Instruction fetch
  - Read instruction from its memory location into the processor.
- Instruction operation decoding (iod):
  - Analyze instruction to determine type of operation to be performed and operand(s) to be used.

State Diagram – Instruction Cycle contd.

- Operand address calculation (oac):
  - If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- Operand fetch (of):
  - Fetch the operand from memory or read it in from I/O
- Data operation (do):
  - Perform the operation indicated in the instruction
- Operand store (os):
  - Write the result into memory or out to I/O

State Diagram – Instruction Cycle contd.


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

Interrupts

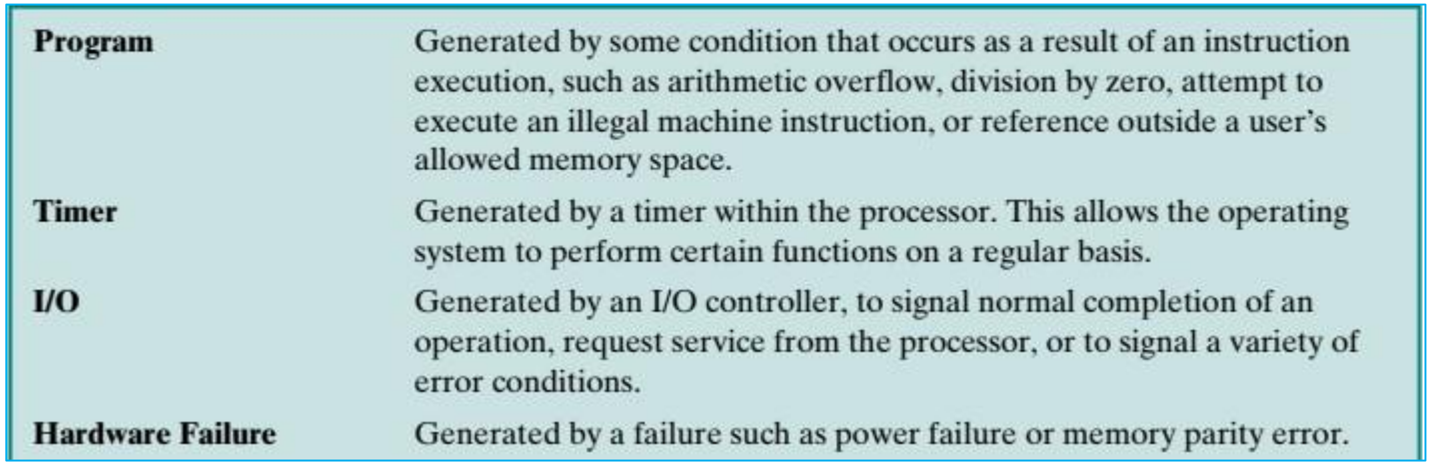
- A mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor.
- Interrupts are provided primarily as a way to improve processing efficiency.

An **interrupt** is a **signal** sent to the CPU that **temporarily stops** the current program and **forces the CPU to deal with an important event** (like input from a keyboard or a hardware error). After handling the event, the CPU **returns** to what it was doing.

Why Interrupts are important?

- They make computers more efficient by letting the CPU respond only when needed, instead of constantly checking devices (called polling).
- They allow the CPU to multitask (handle multiple things at once).

Classes of Interrupts

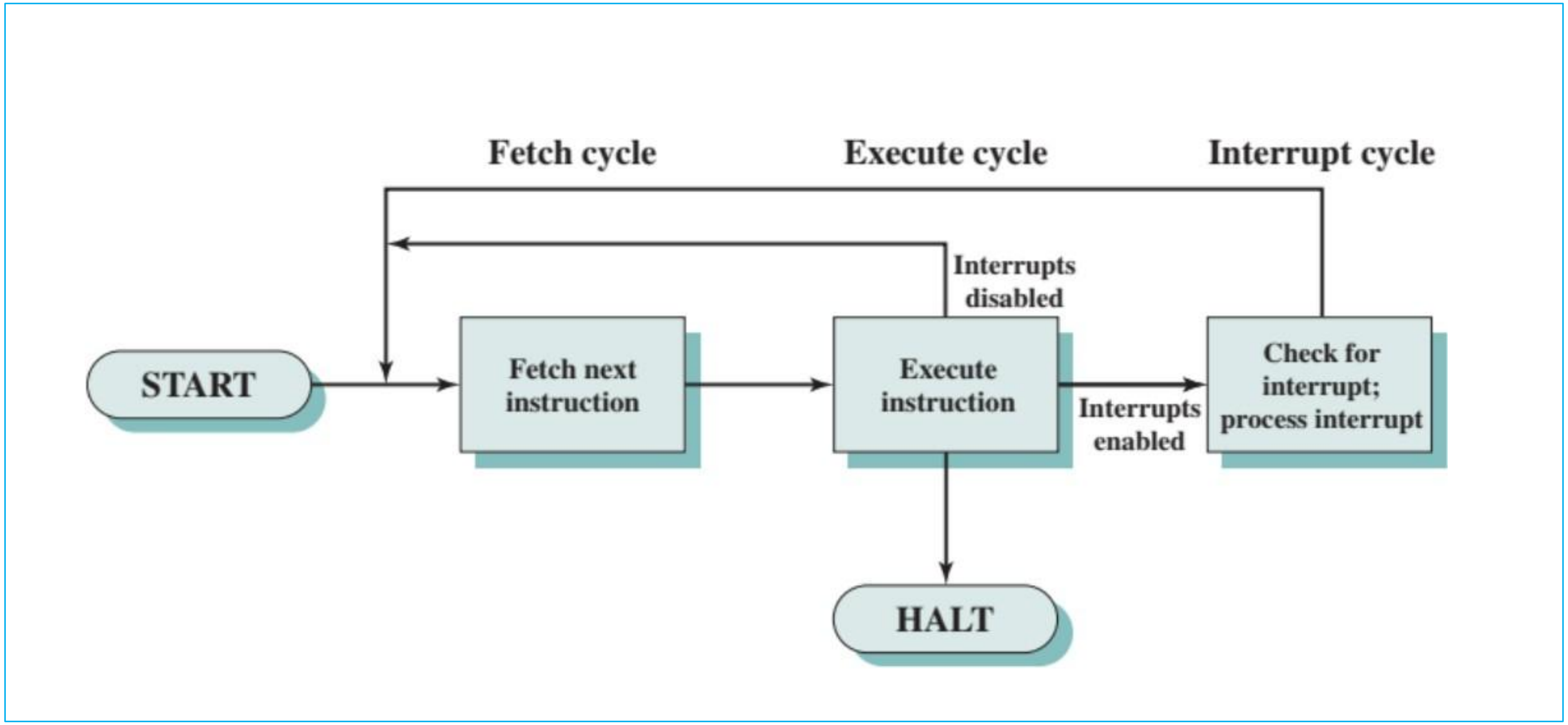


Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware Failure	Generated by a failure such as power failure or memory parity error.

Interrupt Handler

- I/O operation is conducted concurrently with the execution of instructions in the user program
- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress
- When the external device becomes ready to be serviced—that is, when it is ready to accept more data from the processor. The I/O module for that external device sends an interrupt request signal to the processor.
- The processor responds by suspending operation of the current program, branching off to a program to service that particular I/O device, known as an **interrupt handler**, and resuming the original execution after the device is serviced

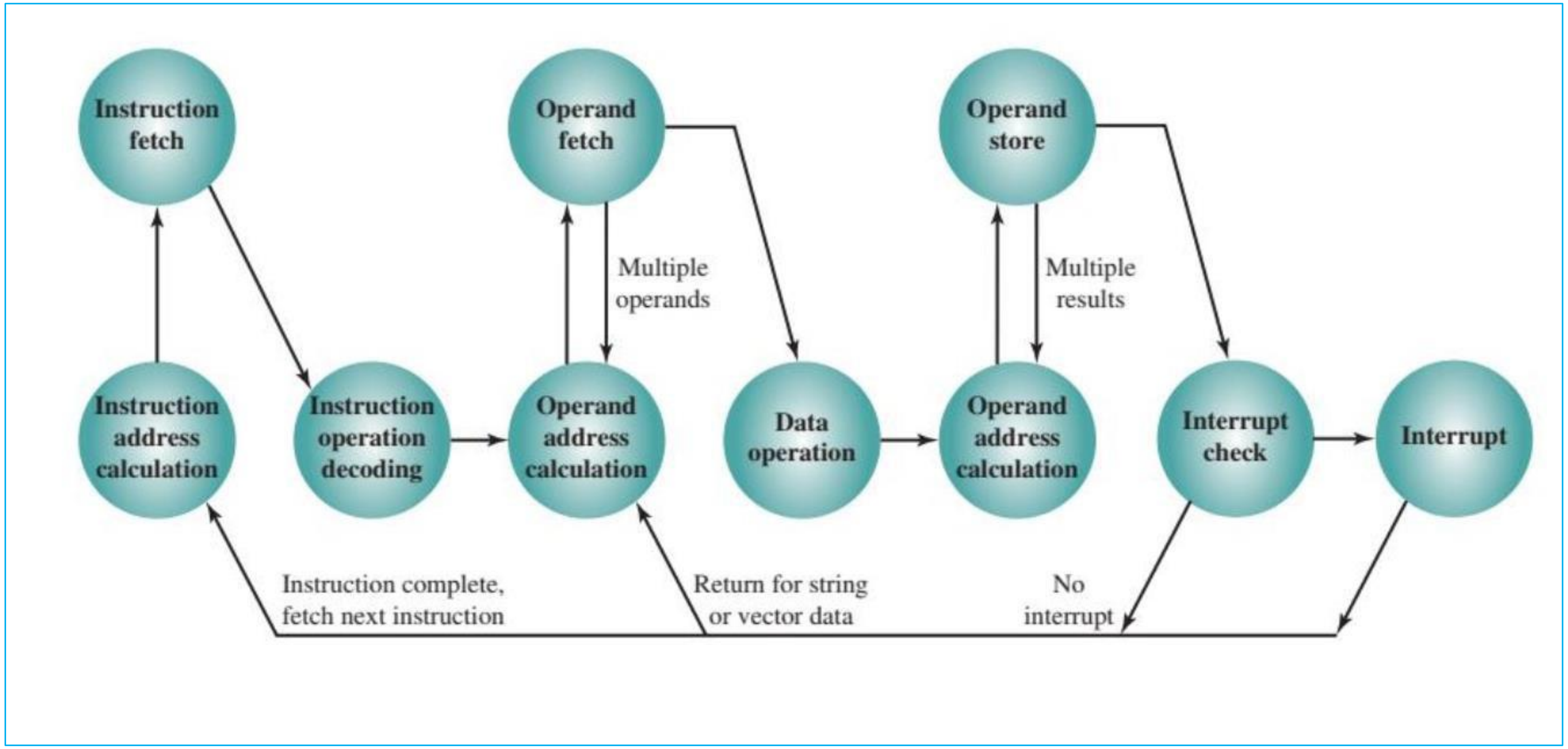
Instruction Cycle with interrupts



Instruction Cycle with interrupts

1. Interrupt Request: A device or software generates an interrupt signal.
2. CPU Saves State: CPU pauses the current work, saving its current status (registers, program counter).
3. Interrupt Handler (ISR): CPU jumps to a special routine called the Interrupt Service Routine to handle the interrupt.
4. Handle the Event: The ISR processes the event (e.g., reads the key pressed).
5. Restore State: CPU restores what it was doing before the interrupt.
6. Resume Normal Execution: CPU continues executing where it left off.

State Diagram- Instruction Cycle with Interrupts


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

Thank you