电子科技大学
格拉斯哥学院
Glasgow College, UESTC

UESTC4019: Real-Time Computer Systems and Architecture

Lecture 5

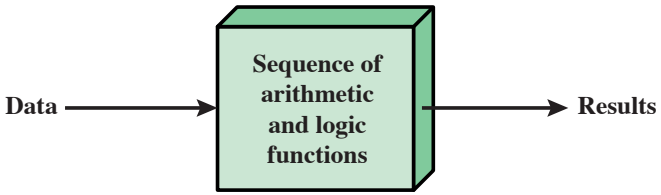
Computer Function and Interconnection (Part-1)

Computer Components

- Virtually all Contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton
- Such a design is referred to as the von Neumann architecture and is based on three key concepts:
  - Data and instructions are stored in a single read-write memory
  - The contents of this memory are addressable by location, without regard to the type of data contained there
  - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next
- Hardwired program
  - The result of the process of connecting the various components in the desired configuration

Hardware and Software Approaches

- In the original case of customized hardware, the system accepts data and produces results, Fig. (a). With general-purpose hardware, the system accepts data and control signals and produces results.
- The entire program is actually a sequence of steps. At each step, some arithmetic or logical operation is performed on some data. For each step, a new set of control signals is needed.
- Let us provide a unique code for each possible set of control signals, and let us add to the general-purpose hardware a segment that can accept a code and generate control signals, Fig. (b).



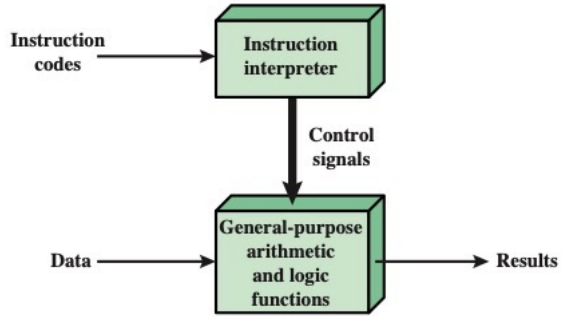
(a) Programming in hardware

Software

- Changing or controlling through programming is much easier compare to rewiring the hardware
- Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes
- Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals
- To distinguish this new method of programming, a sequence of codes or instructions is called software

Programming in Software

- Fig. (b) indicates two major components of the system:
  - an instruction interpreter and
  - a module of general-purpose arithmetic and logic functions
- These two components constitute the CPU



(b) Programming in software

- Several other components are needed to yield a functioning computer
- Data and instructions must be put into the system. For this we need some sort of input module. This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system
- A means of reporting results is needed, and this is in the form of an output module. Taken together, these are referred to as I/O components

Major Components

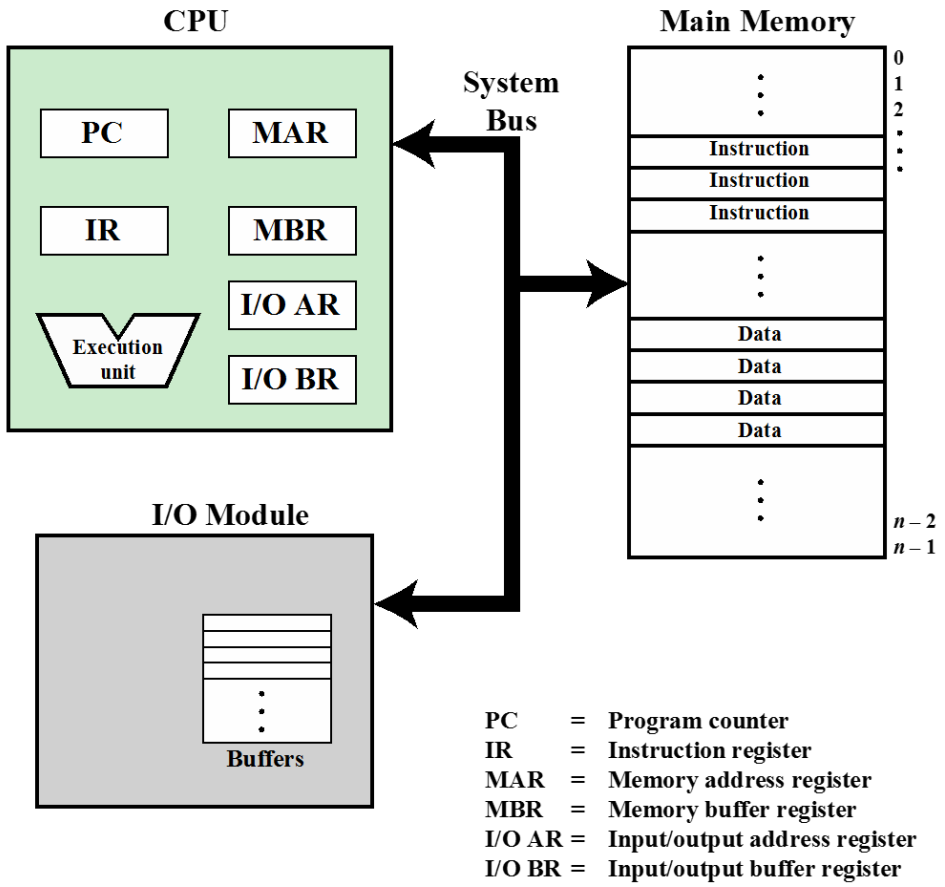
- CPU
  - Instruction interpreter
  - Module of general-purpose arithmetic and logic functions
- I/O Components
  - Input module
    - Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
  - Output module
    - Means of reporting results

Memory

- Memory or Main Memory is a place to store temporarily both instructions and data. Von Neumann pointed out that the same memory could be used to store both instructions and data.
- Memory address register (MAR)
  - Specifies the address in memory for the next read or write
- Memory buffer register (MBR)
  - Contains the data to be written into memory or receives the data read from memory
- I/O address register (I/O AR)
  - Specifies a particular I/O device
- I/O buffer register (I/O BR)
  - Used for the exchange of data between an I/O module and the CPU

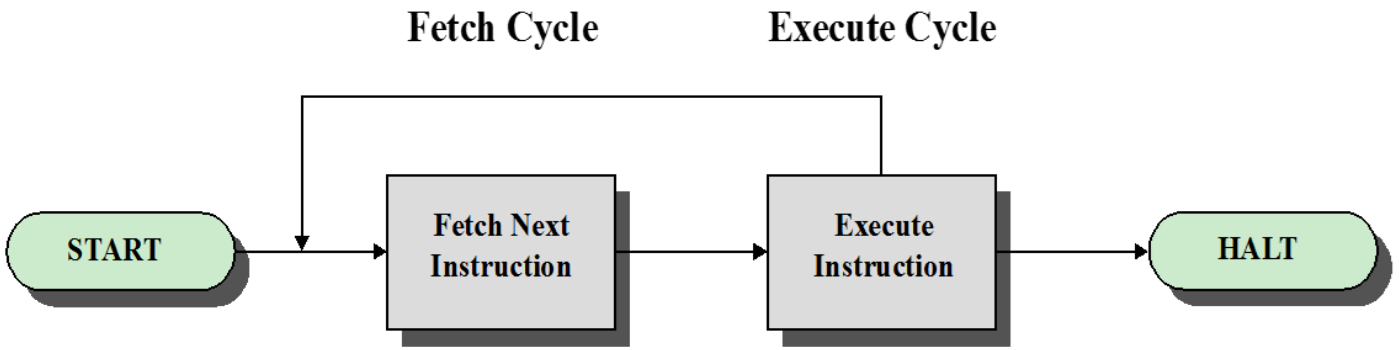
Computer Components :Top Level View

- A memory module consists of a set of locations, defined by sequentially numbered addresses
- Each location contains a binary number that can be interpreted as either an instruction or data
- An I/O module transfers data from external devices to CPU and memory, and vice versa
- It contains internal buffers for temporarily holding these data until they can be sent on



Basic Instruction Cycle

- The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory
- The processor does the actual work by executing instructions specified in the program
- Instruction processing consists of two steps: The processor reads (fetches) instructions from memory one at a time and executes each instruction



Fetch Cycle

- At the beginning of each instruction cycle the processor fetches an instruction from memory
  - The program counter (PC) holds the address of the instruction to be fetched next
  - The processor increments the PC after each instruction fetch so that it will fetch the next instruction in sequence

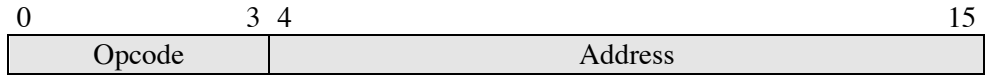
- The fetched instruction is loaded into the instruction register (IR)
  - The processor interprets the instruction and performs the required action

Action Categories

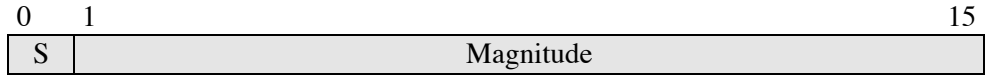
- **Processor-memory**
  - Data transferred from processor to memory or from memory to processor
- **Processor-I/O**
  - Data transferred to or from a peripheral device by transferring between the processor and an I/O module
- **Data processing**
  - The processor may perform some arithmetic or logic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

Characteristics of a Hypothetical Machine

- The processor contains a single data register, called an accumulator (AC). Both instructions and data are 16 bits long. Thus, it is convenient to organize memory using 16-bit words
- The instruction format provides 4 bits for the opcode, so that there can be as many as $2^4 = 16$ different opcodes, and up to $2^{12} = 4096$ (4K) words of memory can be directly addressed.



(a) Instruction format



(b) Integer format

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

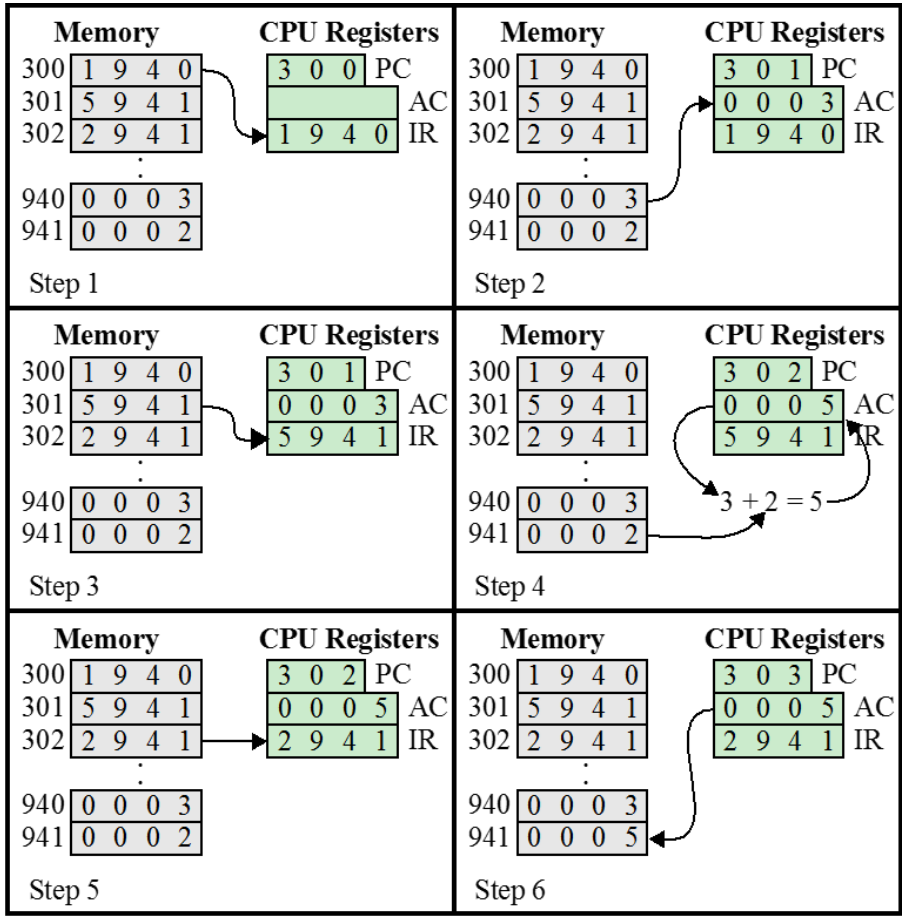
(c) Internal CPU registers

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

(d) Partial list of opcodes

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

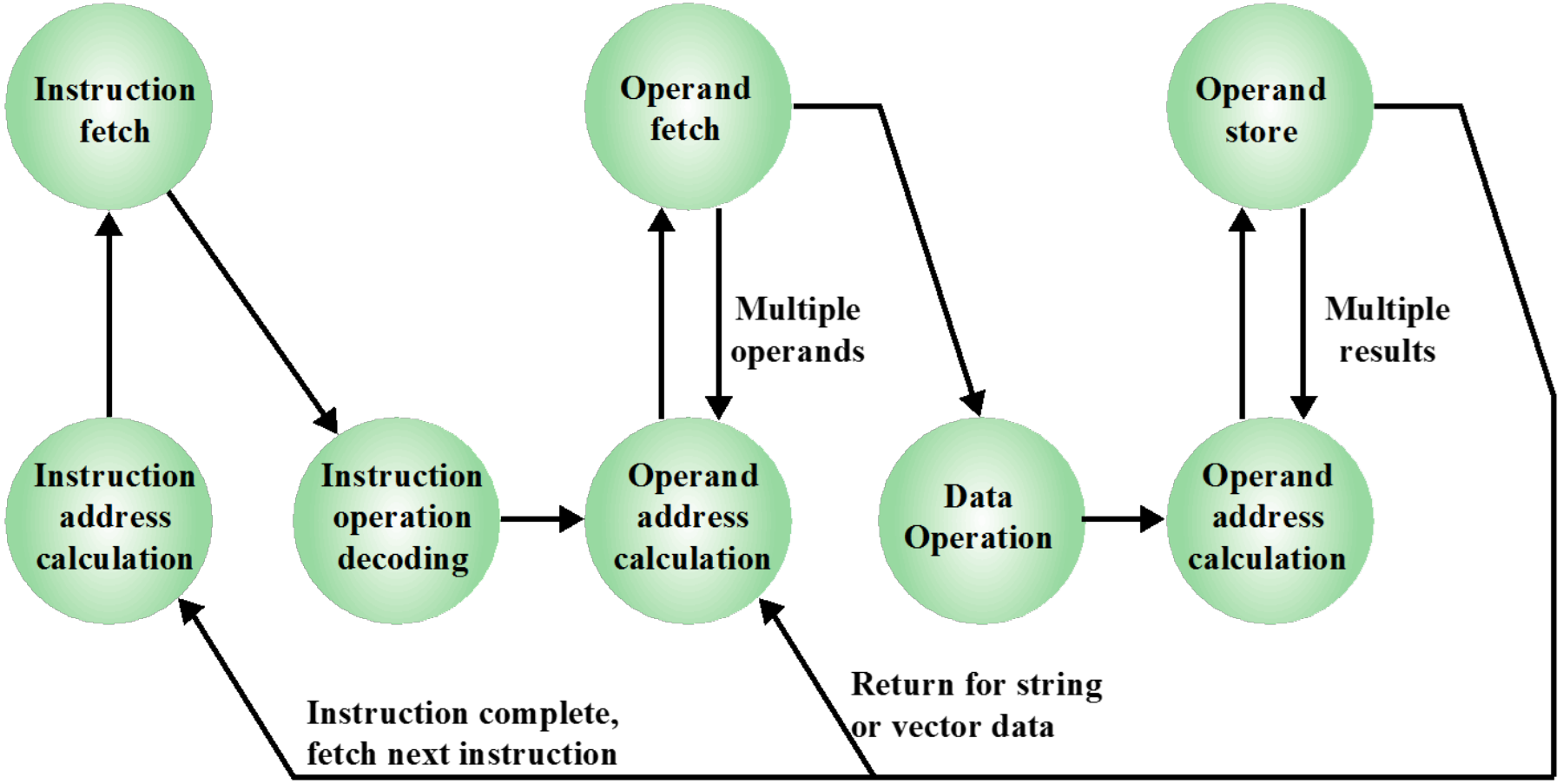
- The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location.
- Three instructions, which can be described as three fetch and three execute cycles, are required.



Example of Program Execution

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. Note that this process involves the use of a memory address register and a memory buffer register. For simplicity, these intermediate registers are ignored.
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

Instruction Cycle State Diagram (1 of 2)



Instruction Cycle State Diagram (2 of 2)

For any given instruction cycle, some states may be null and others may be visited more than once. The states can be described as follows:

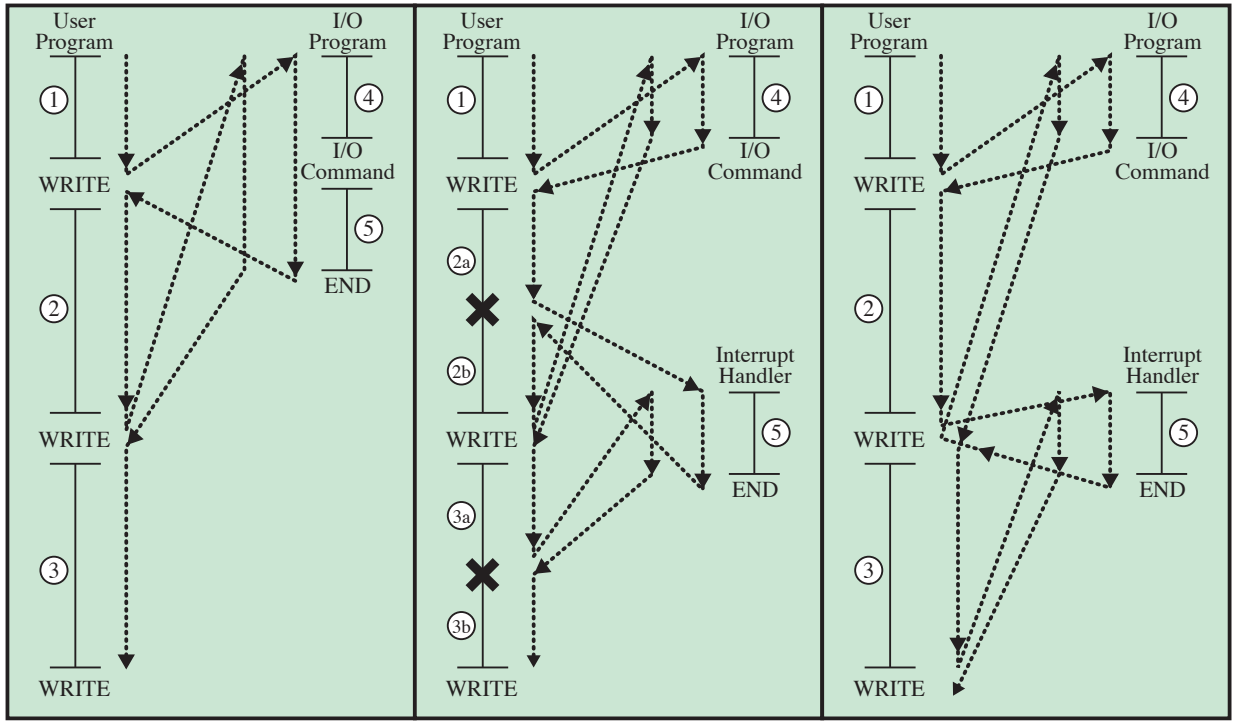
- **Instruction address calculation (iac):** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction
- **Instruction fetch (if):** Read instruction from its memory location into the processor
- **Instruction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

Class of Interrupts

- Virtually all computers provide a mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor
- Table below lists the most common classes of interrupts. The specific nature of these interrupts is examined later in the course

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.

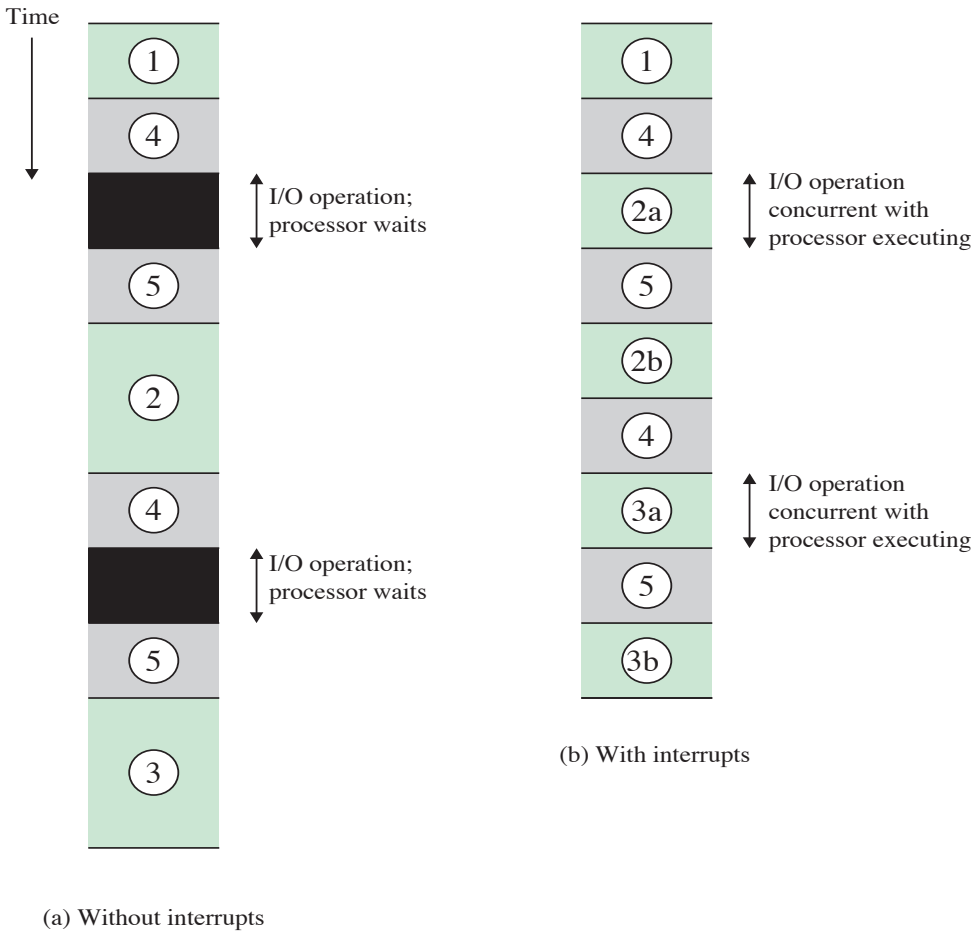
Program Flow of Control With and Without Interrupts



(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

X = interrupt occurs during the course of execution of user program

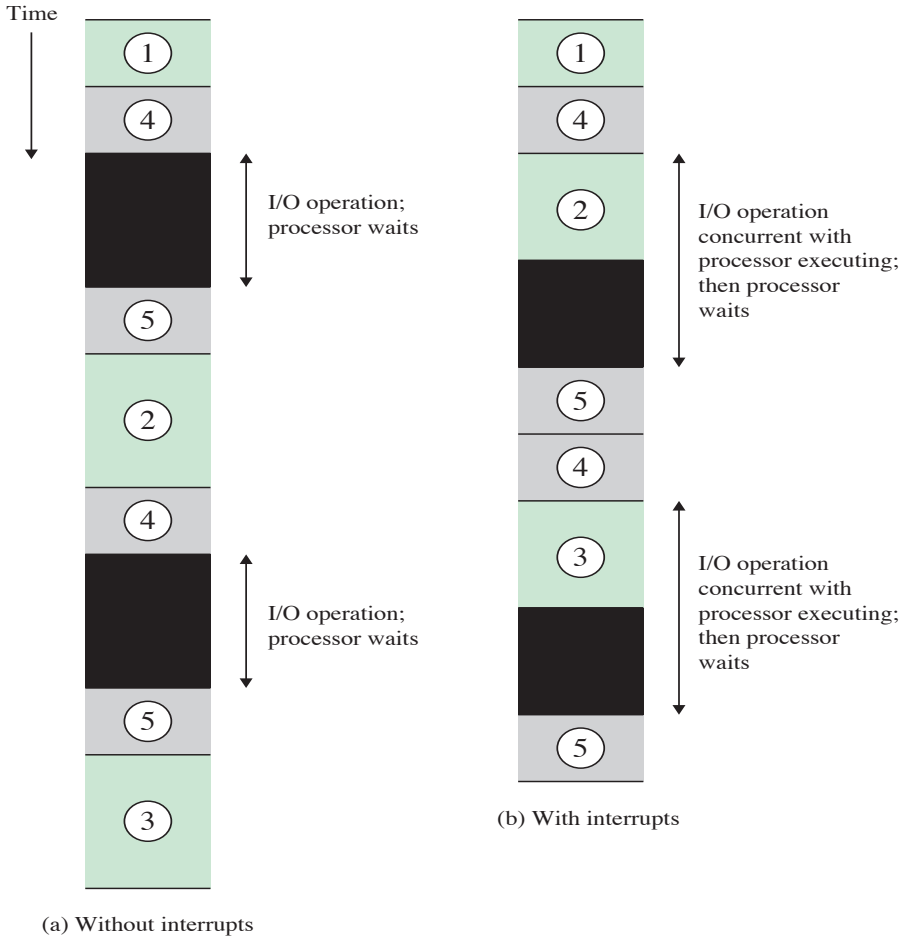
Program Timing: Short I/O Wait



(a) Without interrupts

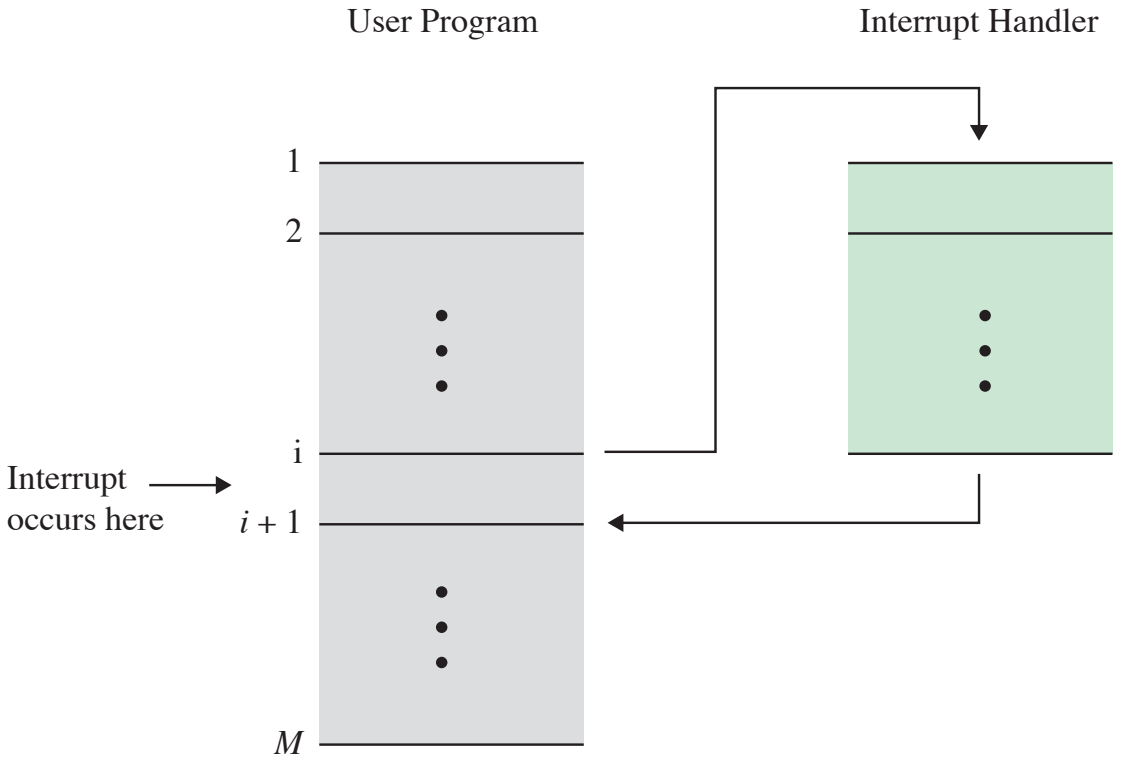
(b) With interrupts

Program Timing: Long I/O Wait

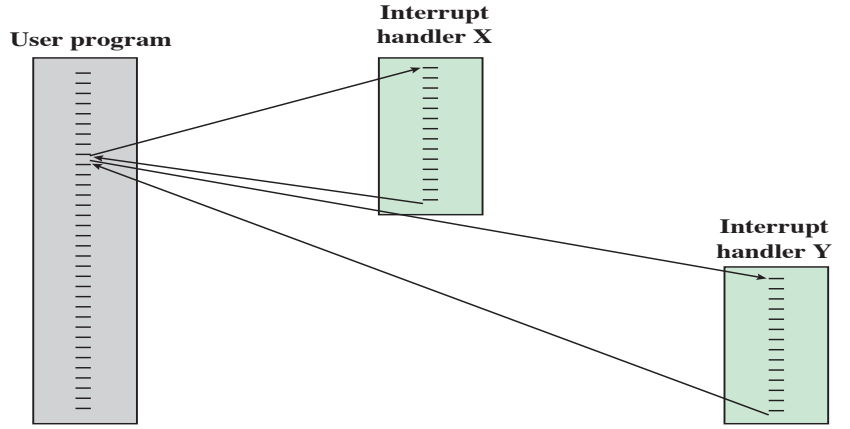


Transfer of Control via Interrupts

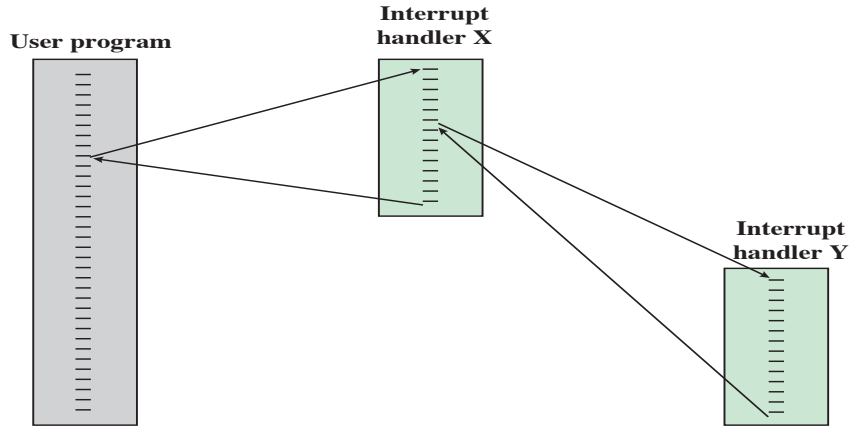
- From the point of view of the user program, an interrupt is just a normal sequence of execution
- When the interrupt processing is completed, execution resumes
- Thus, the user program does not have to contain any special code to accommodate interrupts; the processor and the operating system are responsible for suspending the user program and then resuming it at the same point



Transfer of Control With Multiple Interrupts

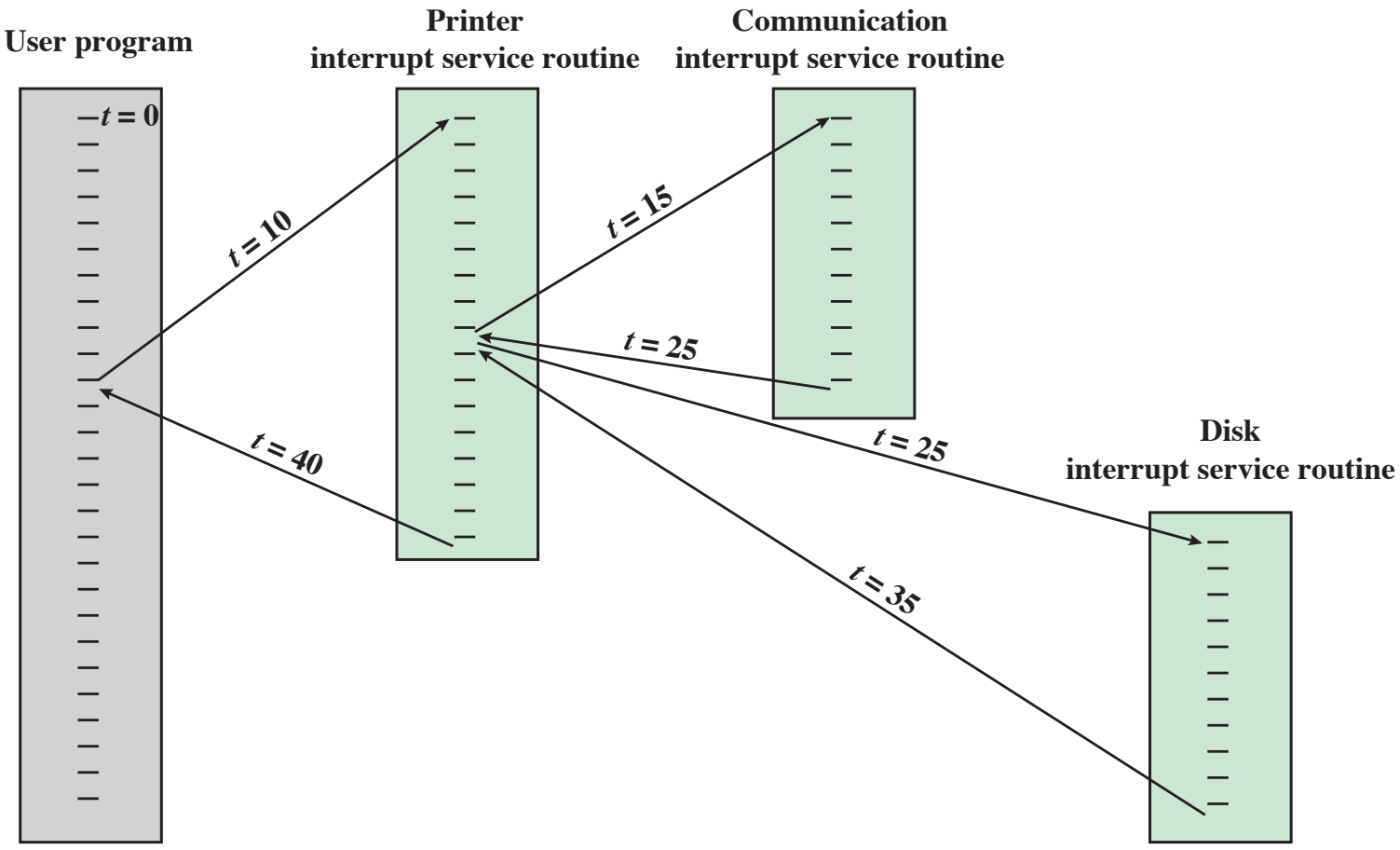


(a) Sequential interrupt processing

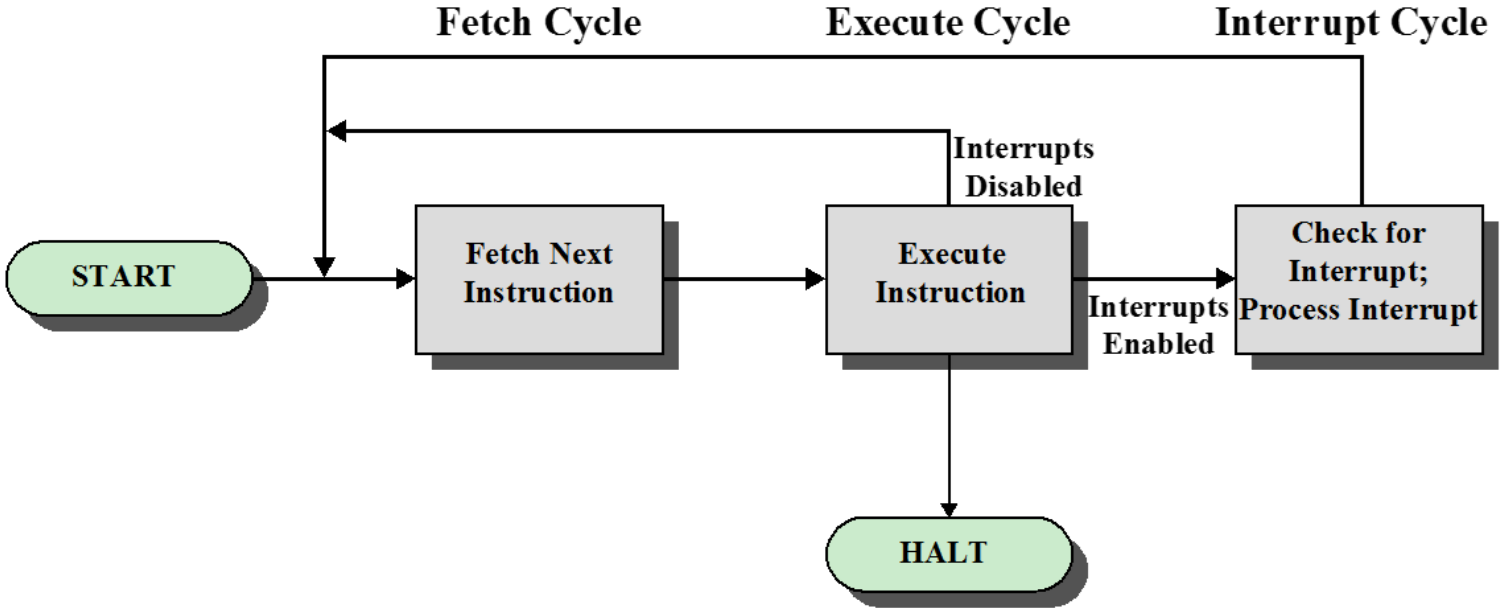


(b) Nested interrupt processing

Example Time Sequence of Multiple Interrupts

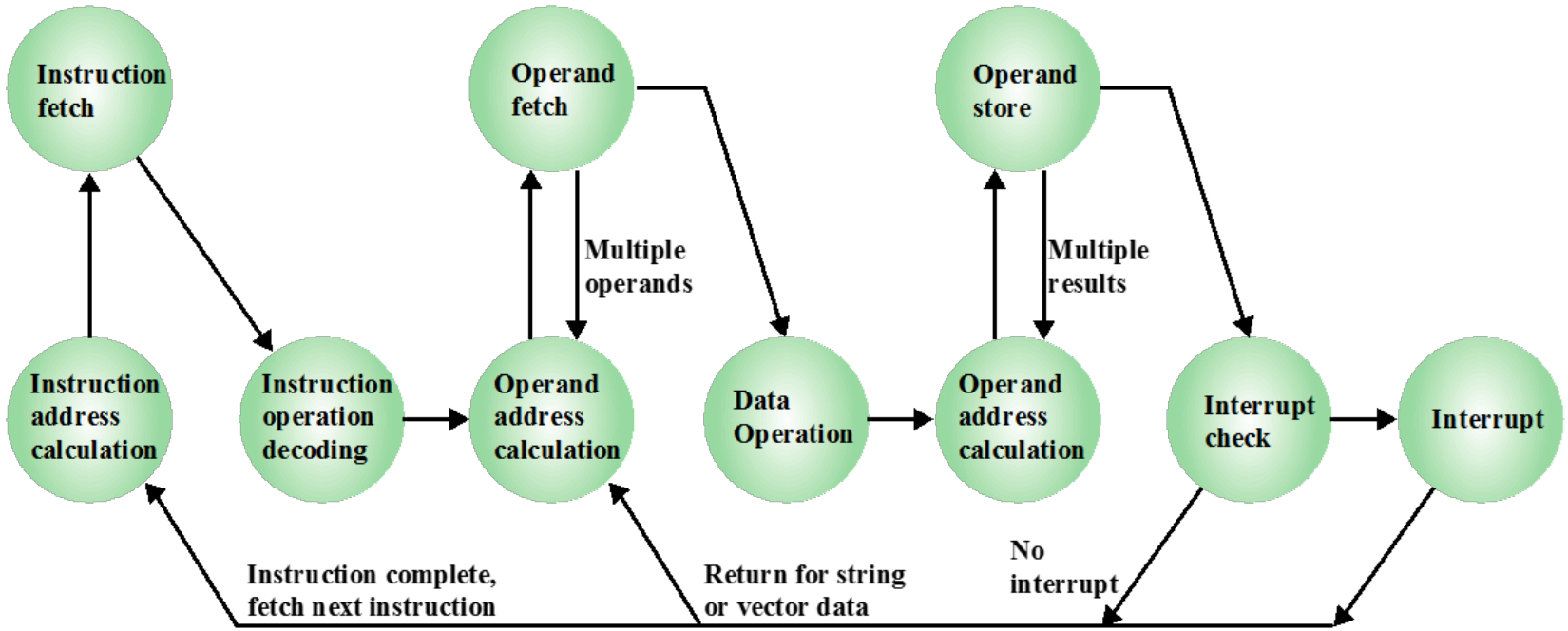


Instruction Cycle with Interrupts

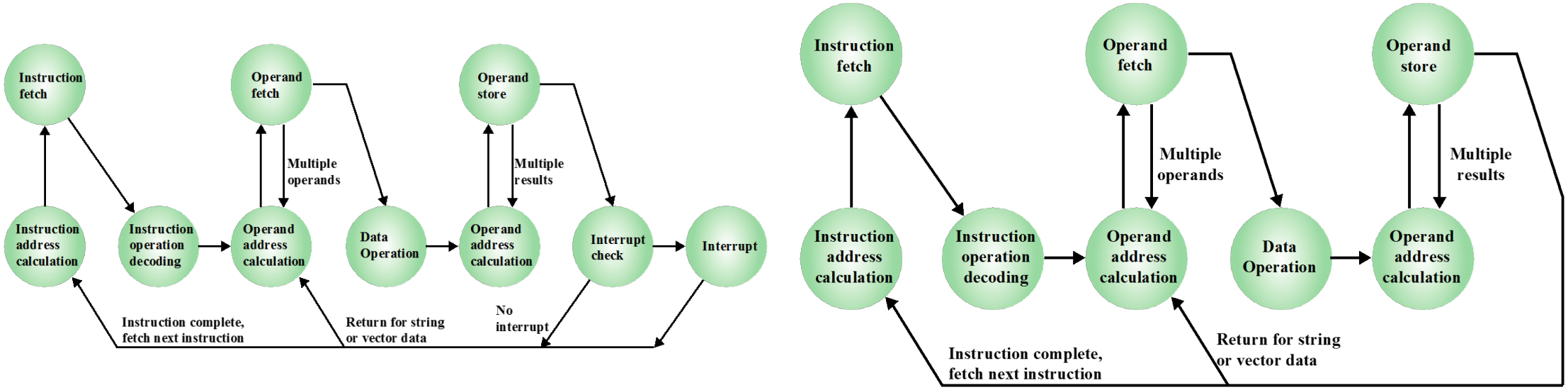


- To accommodate interrupts, an *interrupt cycle* is added to the instruction cycle, as shown in Figure. In the interrupt cycle, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal. If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program

Instruction Cycle State Diagram, With Interrupts



Comparison


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
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

Example Questions (1 of 2)

- What general categories of functions are specified by computer instructions?

Example Questions (1 of 2)

- What general categories of functions are specified by computer instructions?
  - **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
  - **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
  - **Data processing**: The processor may perform some arithmetic or logic operation on data.
  - **Control:** An instruction may specify that the sequence of execution be altered.

Example Questions (2 of 2)

- What types of transfers must a computer's interconnection structure (e.g., bus) support?

Example Questions (2 of 2)

- What types of transfers must a computer's interconnection structure (e.g., bus) support?
  - **Memory to processor:** The processor reads an instruction or a unit of data from memory
  - **Processor to memory:** The processor writes a unit of data to memory
  - **I/O to processor:** The processor reads data from an I/O device via an I/O module
  - **Processor to I/O:** The processor sends data to the I/O device
  - **I/O to or from memory:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA)