2.1 Computer Components

In the original case of customized hardware, the system accepts data and produces results (Figure 2.1a), the system accepts data and control signals and produces results. Thus, instead of rewiring the hardware for each new program

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 (Figure 2.1b).

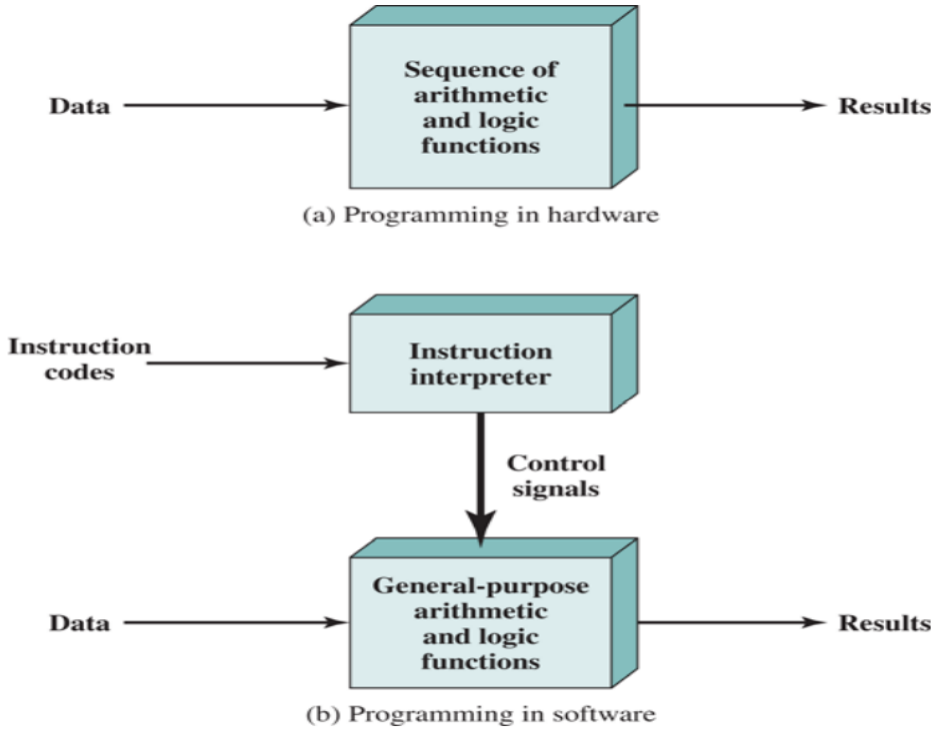


Figure 2.1 Hardware and Software Approaches

Figure 2.2 illustrates these top-level components and suggests the interactions among them. The CPU exchanges data with memory. For this purpose, it typically makes use of

- two internal (to the CPU) registers:
  1- a memory address register (MAR) , which specifies the address in memory for the next read or write,
  2- and a memory buffer register (MBR) , which contains the data to be written into memory or receives the data read from memory.
- Similarly, an I/O
  1- address register (I/O AR) specifies a particular I/O device.

2- An I/O buffer register (I/O BR) is used for the exchange of data between an I/O module and the CPU.

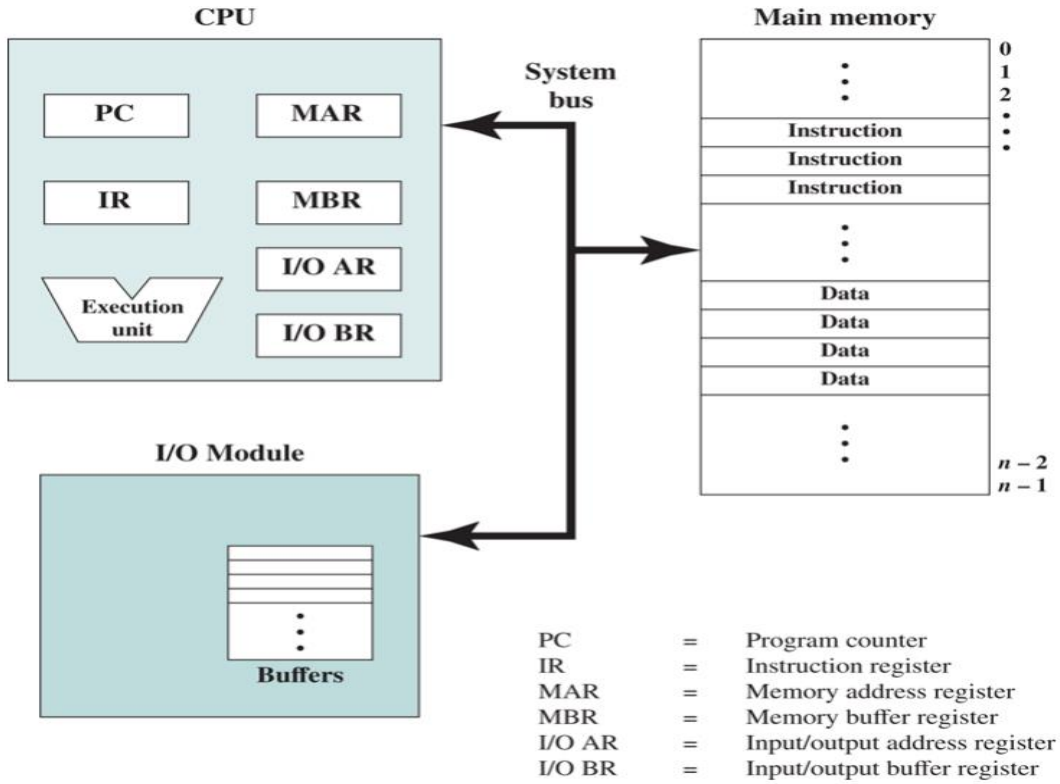


Figure 2.2 Computer Components: Top-Level View

2.2 Computer Function

The processing required for a single instruction is called an instruction cycle . Using the simplified two-step description given previously, the instruction cycle is depicted in Figure 2.3. The two steps are referred to as

- the fetch cycle and
- the execute cycle .

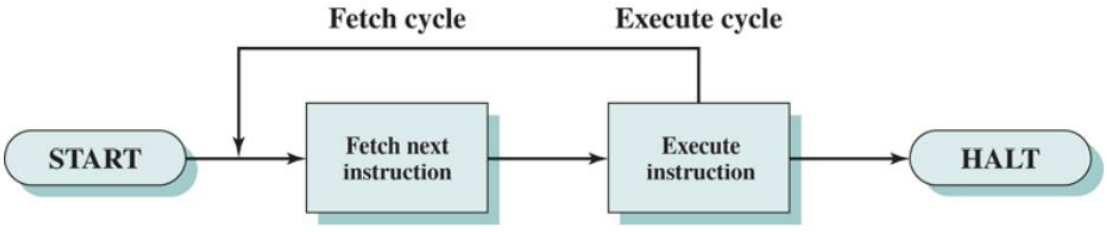


Figure 2.3 Basic Instruction Cycle

Instruction Fetch and Execute: At the beginning of each instruction cycle, the processor fetches an instruction from memory.

1- a register called the program counter (PC) holds the address of the instruction to be fetched next
2- the processor always increments the PC after each instruction fetch so that it will fetch the next instruction

3- The fetched instruction is loaded into a register in the processor known as the instruction register (IR).
4- . The processor interprets the instruction and performs the required action. In general, these actions fall into four categories.

Consider a simple example using a hypothetical machine that includes the characteristics listed in Figure 2.4. For And the organize memory using 16-bit words.

- The instruction format provides 4 bits for the opcode,, so 2^4=16 different opcodes
- and up to 2^{12}=4096 (4K) words of memory can be directly addressed

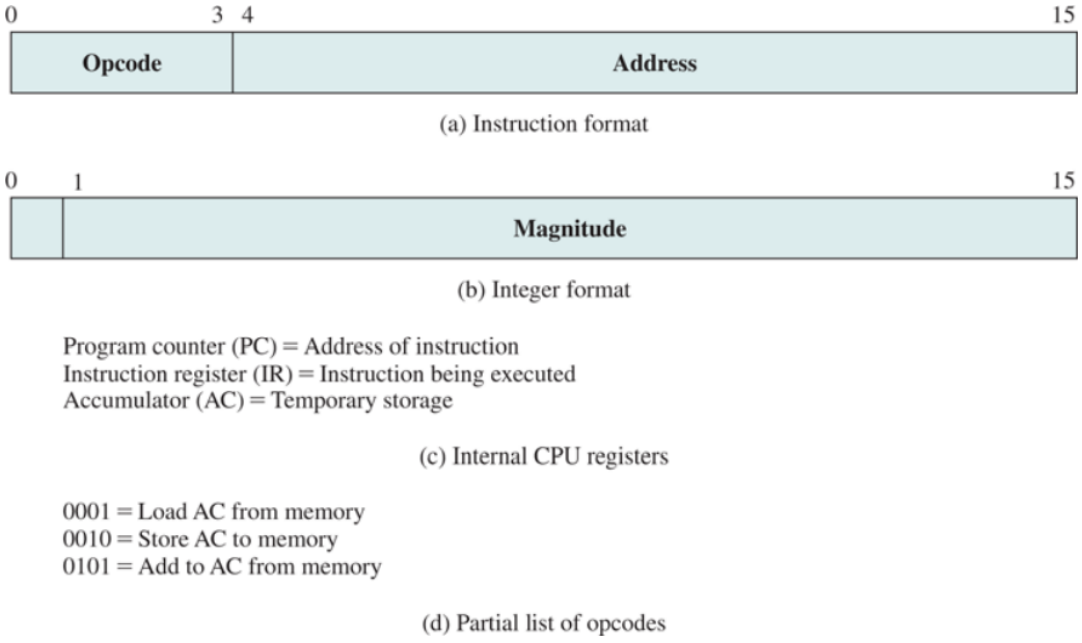


Figure 2.4 Characteristics of a Hypothetical Machine

A single instruction cycle with the following steps occurs:

1- Fetch the ADD instruction.
2- Read the contents of memory location A into the processor.
3- Read the contents of memory location B into the processor.
4- Add the two values.
5- Write the result from the processor to memory location A

In Figure 2.5 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):
- Instruction fetch (if):
- Instruction operation decoding (iod):
- Operand address calculation (oac):.
- Operand fetch (of):.
- Data operation (do):
- Operand store (os):

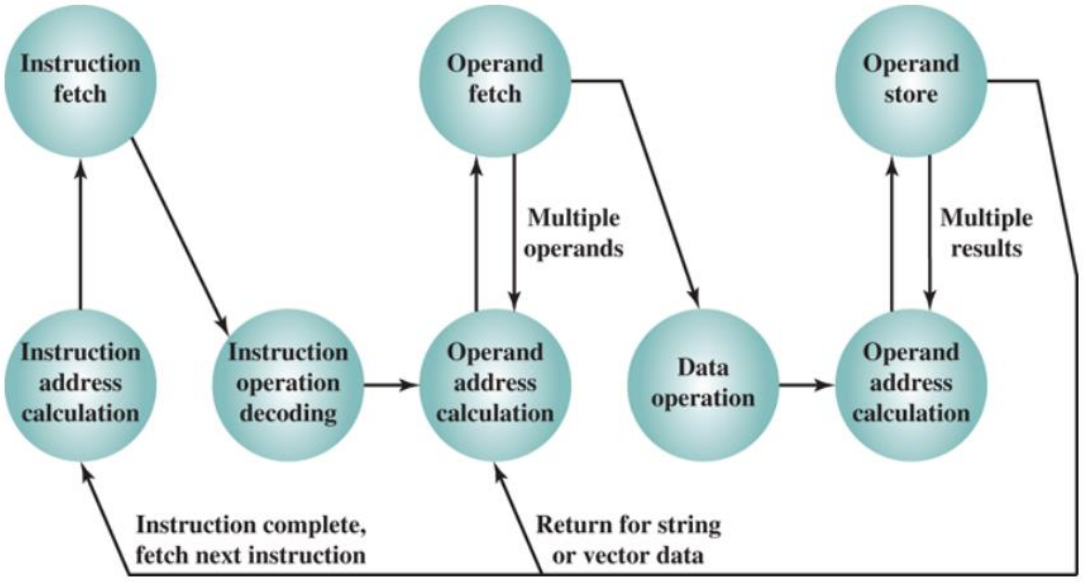


Figure 2.5 Instruction Cycle State Diagram

Interrupts

Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor. Table 2.1 lists the most common classes of

Table 2.1 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.

Let us try to clarify what is happening in *Figure 2.6* ,We have a user program that contains two *WRITE* commands. There is a segment of code at the beginning, then one *WRITE* command, then a second segment of code, then a second *WRITE* command, then a third and final segment of code. The *WRITE* command invokes the I/O program provided by the OS. Similarly, the I/O program consists of a segment of code, followed by an I/O command, followed by another segment of code. The I/O command invokes a hardware I/O operation.

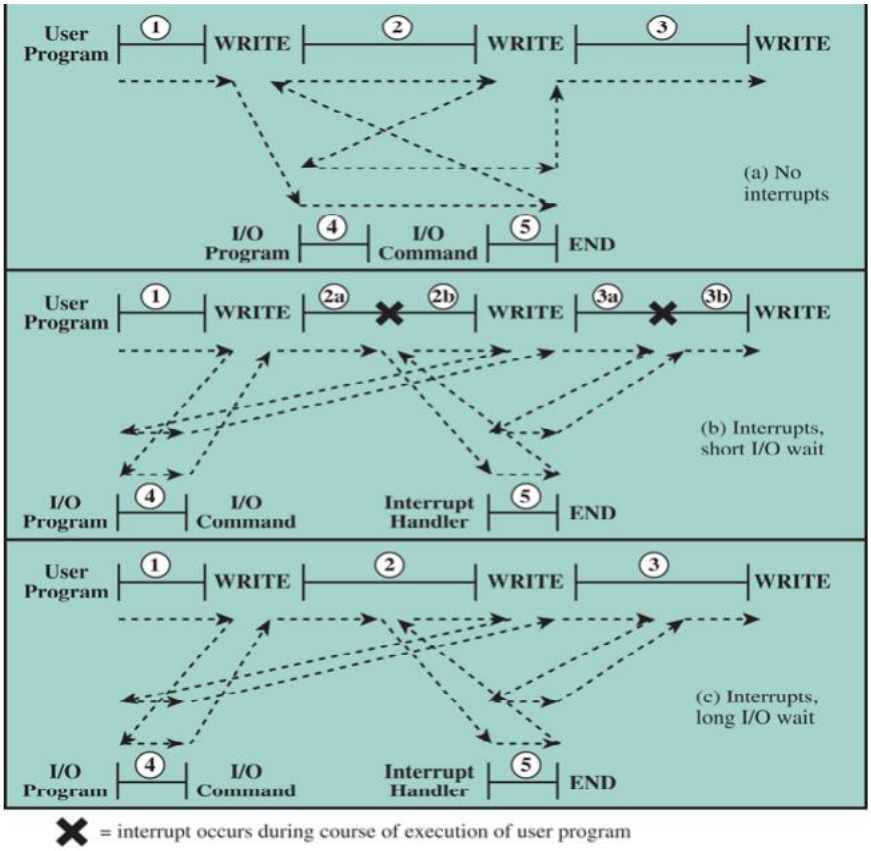


Figure 2.6 Program Flow of Control without and with Interrupts

When the interrupt processing is completed, execution resumes *(Figure 2.7*). To accommodate interrupts, an interrupt cycle is added to the instruction cycle, as shown in Figure 2.8:

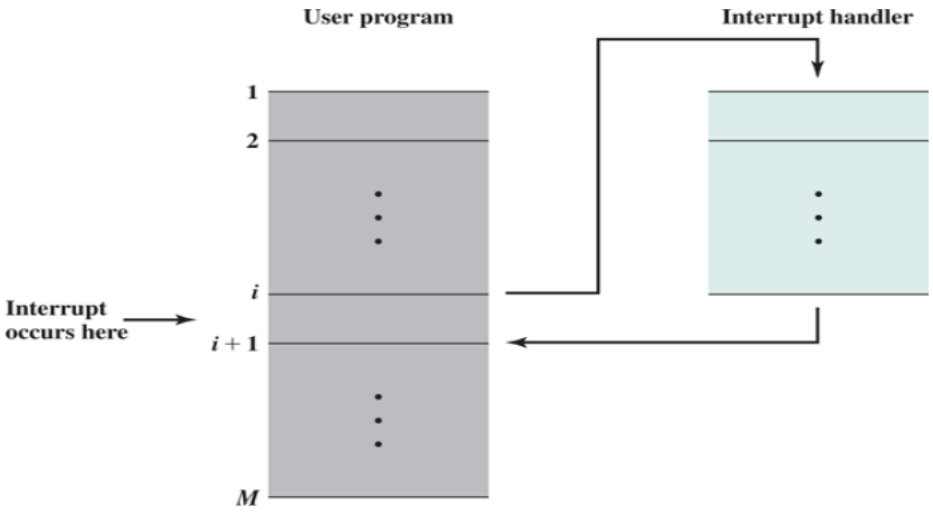


Figure 2.7 Transfer of Control via Interrupts

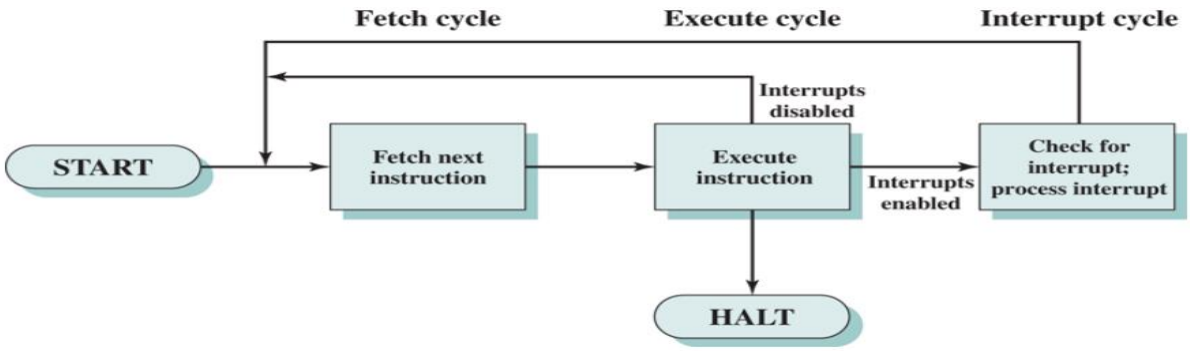
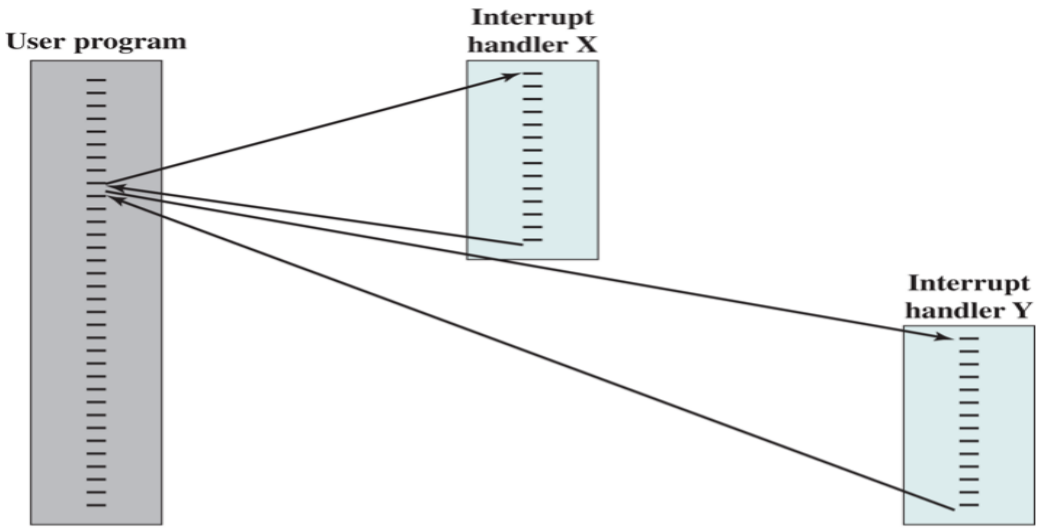


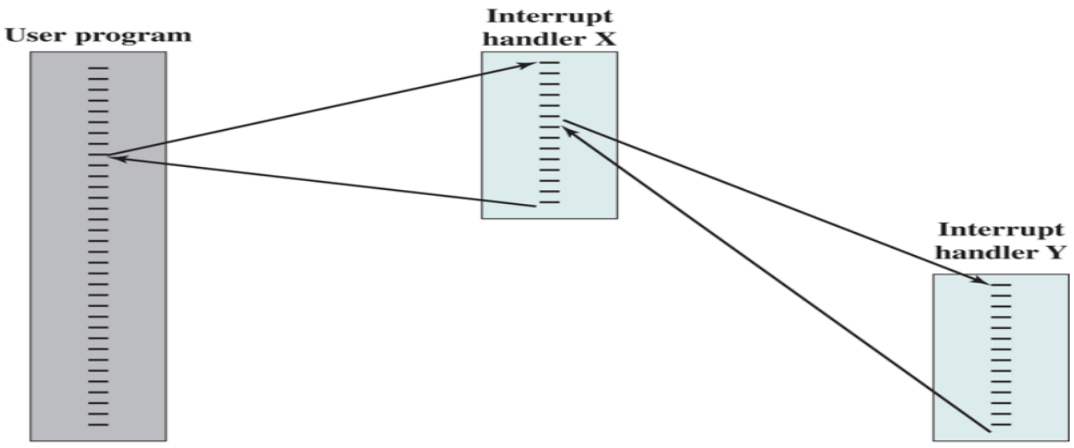
Figure 2.8 Instruction Cycle with Interrupts

MULTIPLE INTERRUPTS. Two approaches can be taken to dealing with multiple interrupts:

- The first is to disable interrupts while an interrupt is being processed. This approach is nice and simple, as interrupts are handled in strict sequential order (Figure 2.9a)
- A second approach is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to itself be interrupted (Figure 3.9b).



(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 2.9 Transfer of Control with Multiple Interrupts

Example: of this second approach, consider a system with three I/O devices: a printer, a disk, and a communications line, with increasing priorities of 2, 4, and 5, respectively. Figure 3.10 illustrates a possible sequence

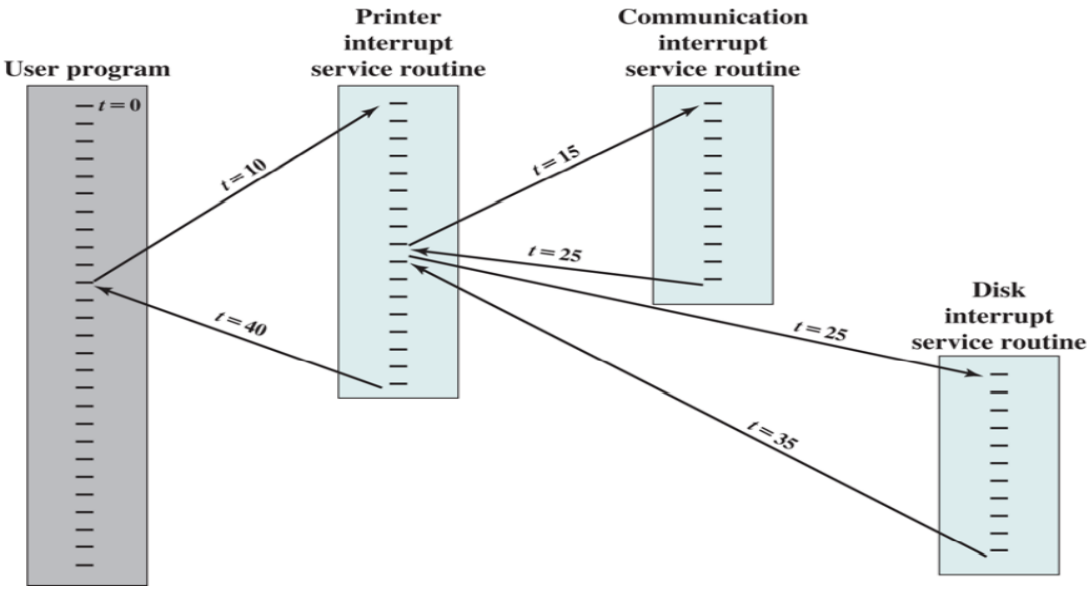


Figure 2.10 Example Time Sequence of Multiple Interrupts

2.3 Interconnection Structures

A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other. Thus, there must be paths for connecting the modules. The collection of paths connecting the various modules is called the interconnection structure. Figure 2.11 suggests the types of exchanges that are needed by indicating the major forms of input and output for each module type

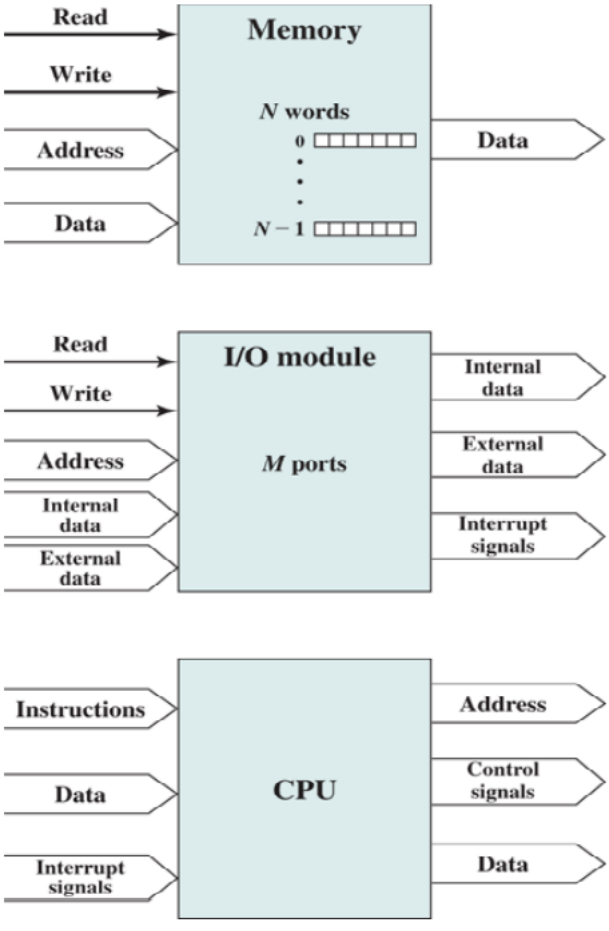


Figure 2.11 Computer Modules

The interconnection structure must support the following types of transfers:

- Memory to processor:
- Processor to memory:
- I/O to processor:
- Processor to I/O:
- I/O to or from memory:

Over the years, a number of interconnection structures have been tried. By far the most common are :
(1) the bus and various multiple-bus structures,
(2) point-to-point interconnection structures with packetized data transfer.

3.3.1 Bus Interconnection

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy. A bus that connects major computer components (processor, memory, I/O) is called a system bus .

Although there are many different bus designs, on any bus the lines can be classified into three functional groups (Figure 2.12): data, address, and control lines.

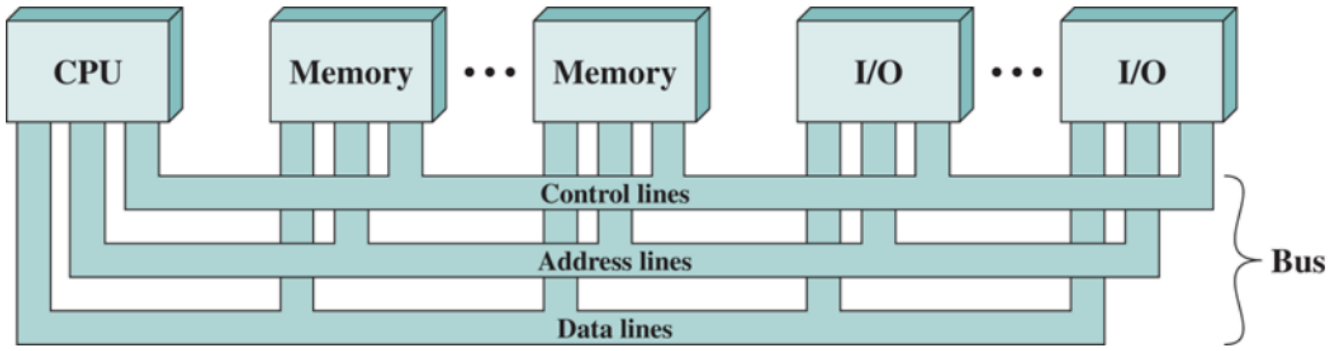


Figure 2.12 Bus Interconnection Scheme

The data lines =data bus . The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the width of the data bus.
The address lines = data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly, the width of the address bus determines the maximum possible memory capacity of the system.
The control bus are used to control the access to and the use of the data and address lines. Typical control lines include:

- Memory write.
- Memory read
- I/O write.
- I/O read.
- Transfer ACK
- Bus request
- Bus grant

- Interrupt request
- Interrupt ACK
- Clock
- Reset

2.3.1 Point-to-Point Interconnect

contemporary systems increasingly rely on point-to-point interconnection rather than shared buses. The principal reason driving the change from bus to point-to-point interconnect was:

1- the electrical constraints encountered with increasing the frequency of wide synchronous buses.

2- with multiple processors and significant memory on a single chip, it was found that the use of a conventional shared bus on the same chip magnified the difficulties of increasing bus data rate and reducing bus latency to keep up with the processors.

example of the point-to-point interconnect approach: Intel's QuickPath Interconnect (QPI), which was introduced in 2008**.** The following are significant characteristics of QPI:

- Multiple direct connections
- Layered protocol architecture.
- Packetized data transfer

Figure 2.13 illustrates a typical use of QPI on a multicore computer. If core A in Figure 2.13 needs to access the memory controller in core D then

1- it sends its request through either cores B or C,

2- which must in turn forward that request on to the memory controller in core D.

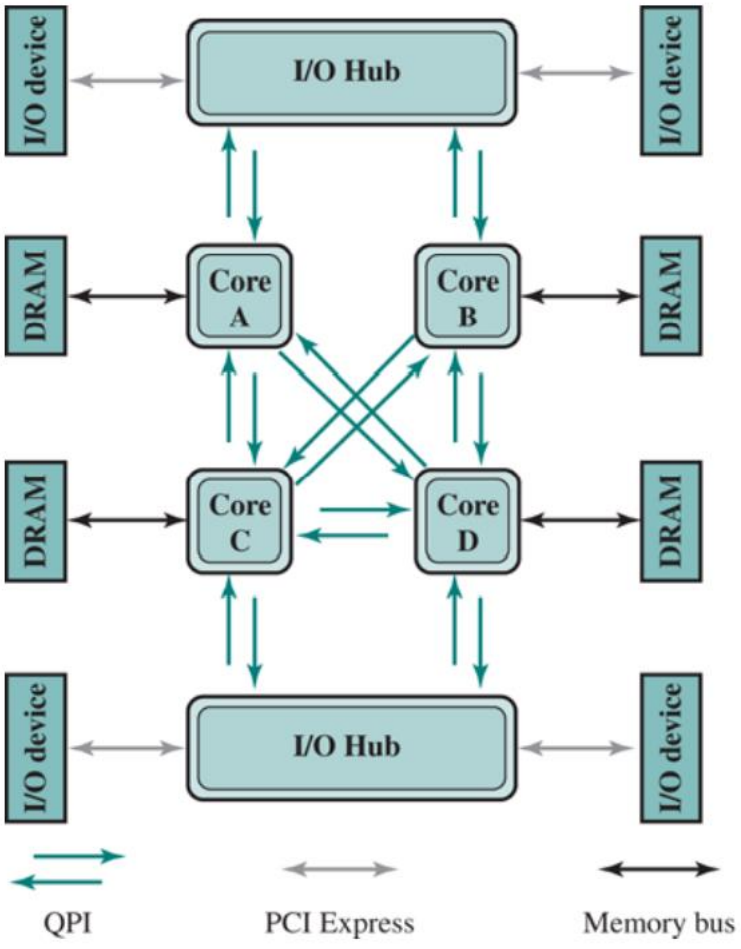


Figure 2.13 Multicore Configuration Using QPI

I/O hub (IOH) acts as a switch directing traffic to and from I/O devices.

QPI is defined as a four-layer protocol architecture, encompassing the following layers (Figure 3.14)

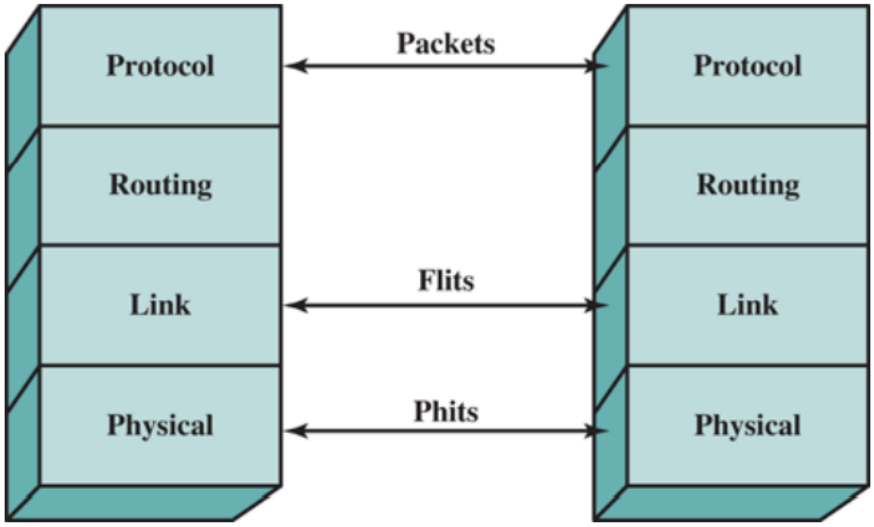


Figure 2.14 QPI Layers

- Physical:. The unit of transfer at the Physical layer is 20 bits, which is called a Phit (physical unit).
- Link: Responsible for reliable transmission and flow control. The Link layer's unit of transfer is an 80-bit Flit (flow control unit).
- Routing: Provides the framework for directing packets through the fabric.
- Protocol: The high-level set of rules for exchanging packets of data between devices.