# Computer Architecture

**Course Code: CSE 360**

**Presented by**

**Dr. Md. Nawab Yousuf Ali**

**Professor, Dept. of CSE**

# Computer Architecture

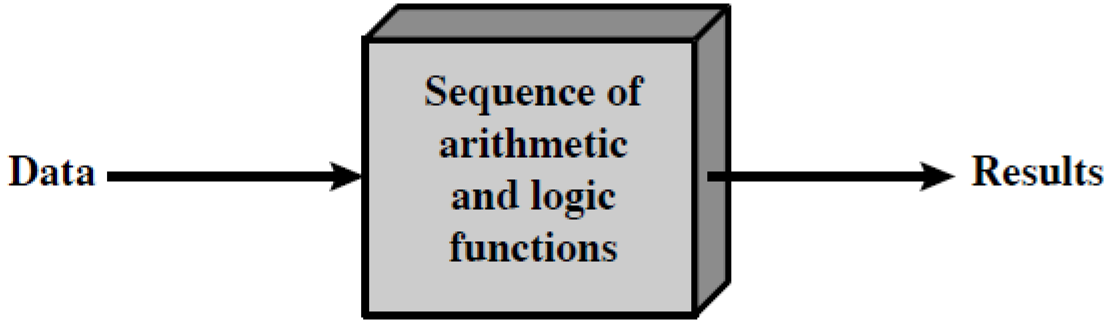
Course Code: CSE360

## Lecture - 3

## System Buses

## Program Concept: Programming in Hardware

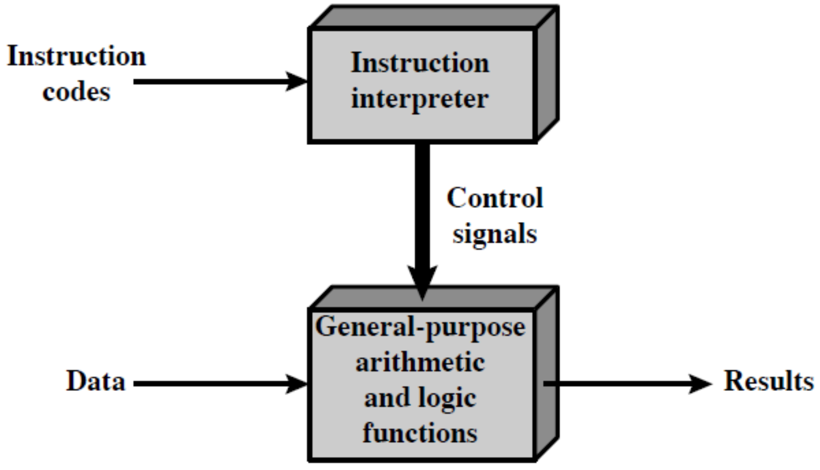
- We construct a general purpose configuration of arithmetic and logic functions. **This set of hardware will perform various functions depending on control signals applied to the hardware.**
- With general-purpose hardware, the system accepts data and control signal and produces results [See (a)]
- Thus, instead of re-wiring the hardware for each new program, **supply a new set of control signals** (as programming is now much easier)



(a) Programming in hardware

## Program Concept: Programming in Software

- How control signals be supplied?
- The entire program is a sequence of codes/instructions (also called *software*)
- For each step, an arithmetic or logical operation is performed on same data
- For each step a new set of control signals is needed
- **Provide a unique code for each possible set of control signals and add to the general-purpose hardware a segment that can accept a code and generate control signals.** [See (b)]



(b) Programming in software

## Function of Control Unit

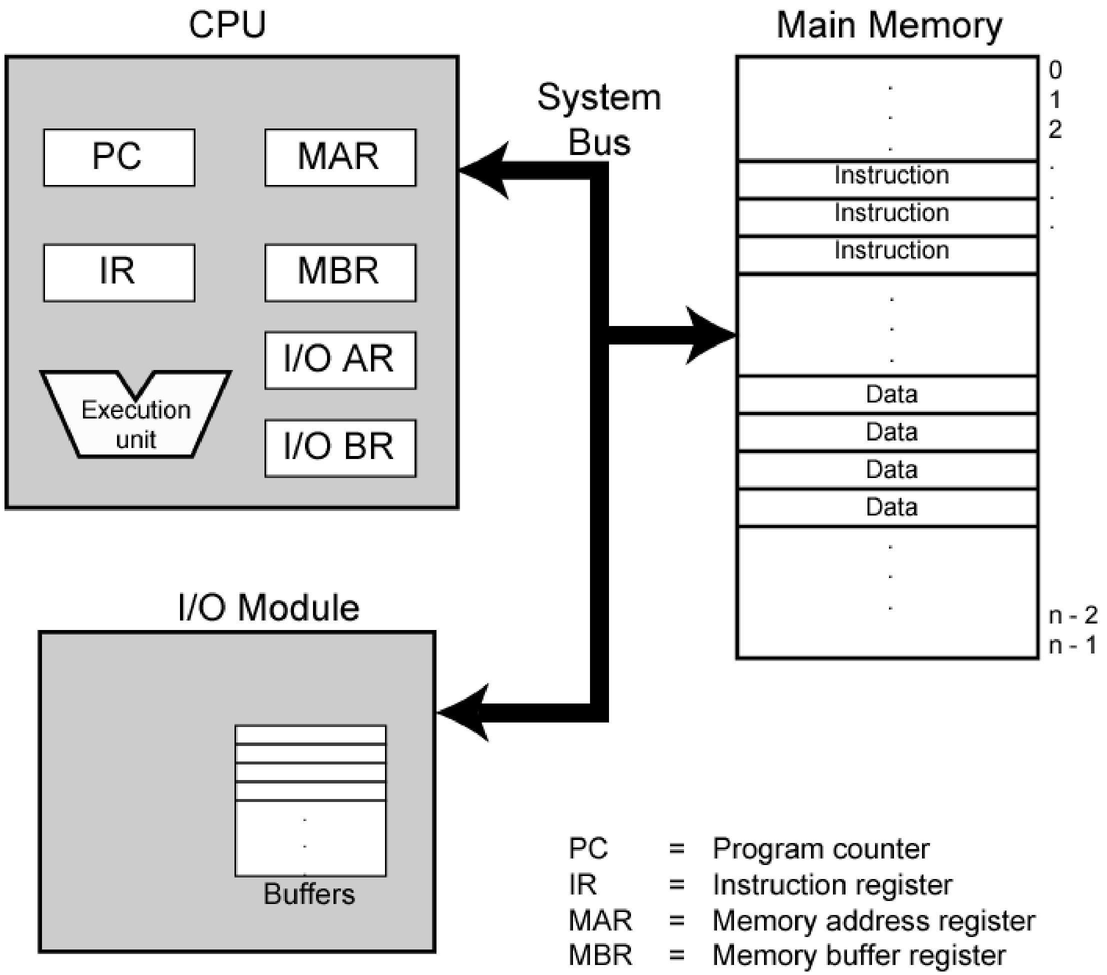
- Figure (b) indicates two major components of the system: an **instruction interpreter** and general-purpose **arithmetic and logic functions**.
- These two constitute the **CPU**
- All the computer resources are managed by the Control Unit
- Control Unit tells the rest of the computer system how to execute operations
- Control unit controls the movement of electronic signals between ALU & MM and MM and I/O devices

## Computer Components

- The **Control Unit** and the **Arithmetic and Logic Unit** constitute the **Central Processing Unit**
- Data and instructions must be put into the system and results out
    - —Input/output
- Temporary storage of code and results is needed
    - —Main memory

# Computer Components: Top Level View

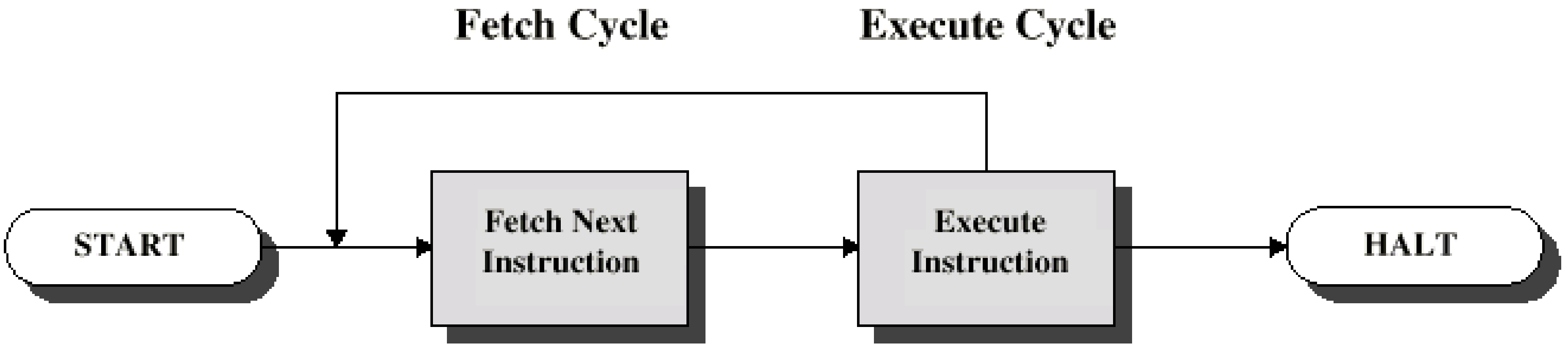
- The CPU exchanges data with memory.
- (i) memory address register (MAR) specifies the address in memory for next read/write (ii) memory buffer register (MBR) contains the data to be written into memory or receives the data read from memory
- I/O address register (I/O AR) specifies a particular I/O device.
- I/O buffer (I/O BR) register is used for the exchange of data between an I/O and the CPU.
- A memory consists of a set of locations; each location contains a binary number that can be interpreted as an instruction or data
- I/O module transfers data from external devices to CPU and memory, and vice versa



| | | |
|---|---|---|
| PC | = | Program counter |
| IR | = | Instruction register |
| MAR | = | Memory address register |
| MBR | = | Memory buffer register |
| I/O AR | = | Input/output address register |
| I/O BR | = | Input/output buffer register |

## Instruction Cycle

- The processor reads (fetches) instructions from memory one at a time and executes each instruction
- The processing required for a single instruction is called an ***instruction Cycle***
- Two steps:
  - **Instruction Fetch**
  - **Instruction Execution**



## Fetch Cycle

- Program Counter (PC) register holds address of next instruction to fetch
- **Processor (CPU) fetches instruction from memory location pointed by PC**
- Processor increments the PC after each instruction fetch so that it will fetch the next instruction in  sequence
- **The fetched instruction is loaded into Instruction Register (IR)**
- Processor interprets the instruction and performs the required actions

## Execute Cycle

- Processor-memory
  - data transfer between CPU and main memory
- Processor-I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Processor may perform some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
- An instruction’s execution may involve a combination of above

## CPU Registers and Opcodes

**Internal CPU registers**

Program counter (PC) = address of next instruction

Instruction Register (IR) = Instruction being executed

Accumulator (AC) =  Temporary storage

**List of Opcodes**

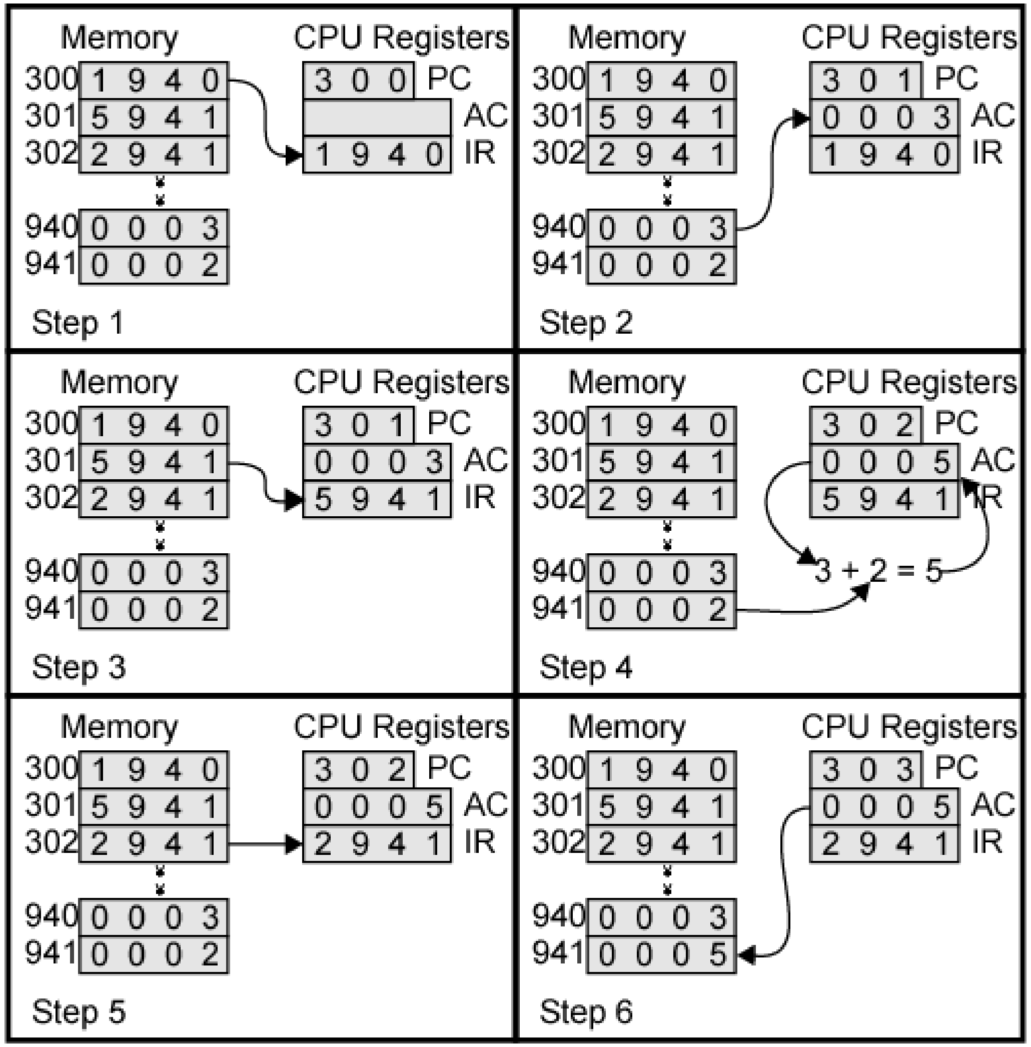
0001 = Load AC from memory

0010 = Store AC to memory

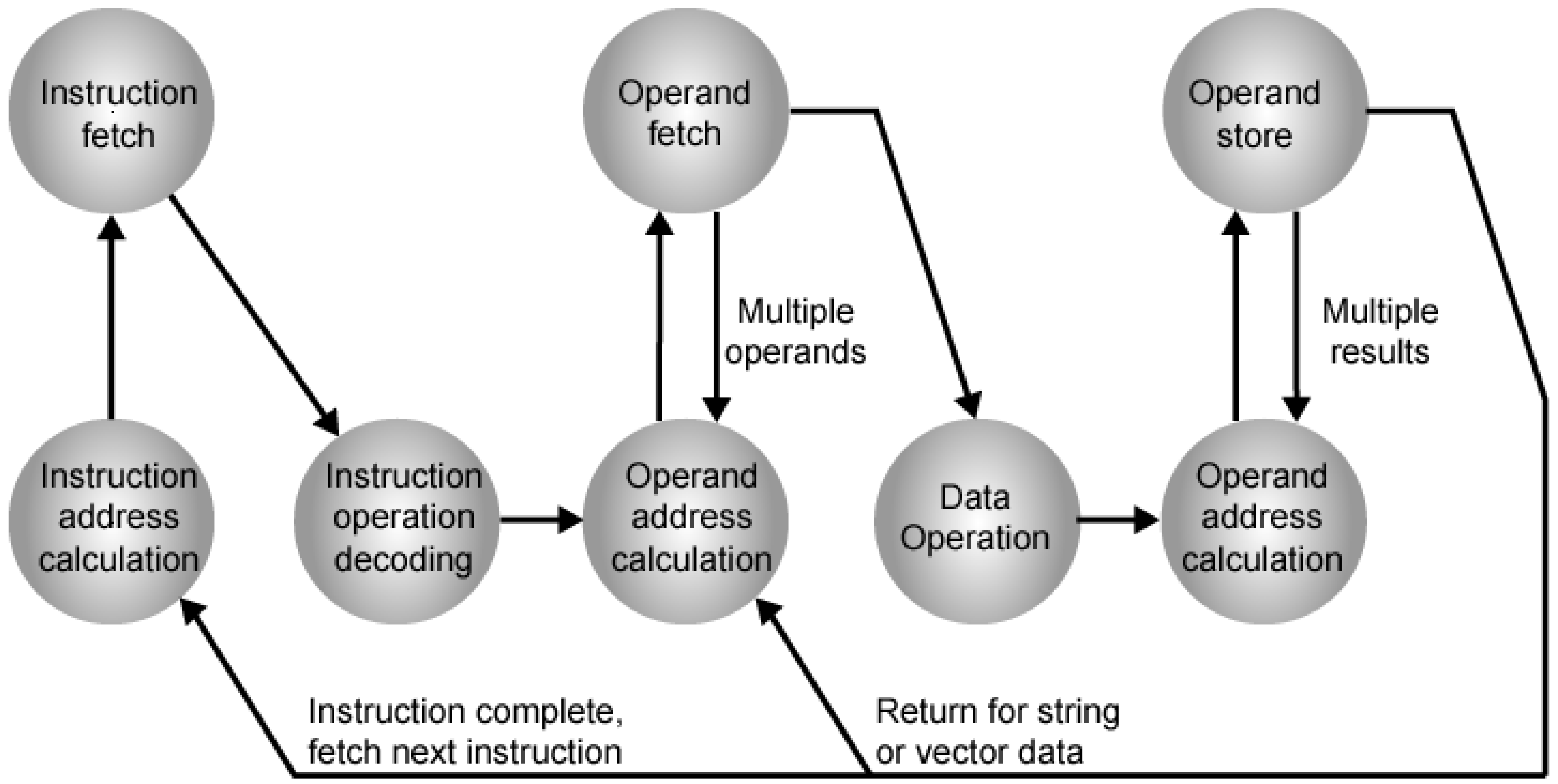
0101 = Add to AC from memory

# Example of Program Execution

- PC contains 300, the address of the first instruction. The instruction is loaded into the instruction register IR and PC is incremented.
- The first 4 bits in the IR indicated that AC is to be loaded; the remaining 12 bits specify the address (940) from which data are to be loaded.
- The next instruction (5941) fetches from location 301 and PC is incremented.
- The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
- Next instruction (2941) is fetched from location 302 and PC is incremented.
- The contents of the AC are stored in location 941.



## Instruction Cycle State Diagram

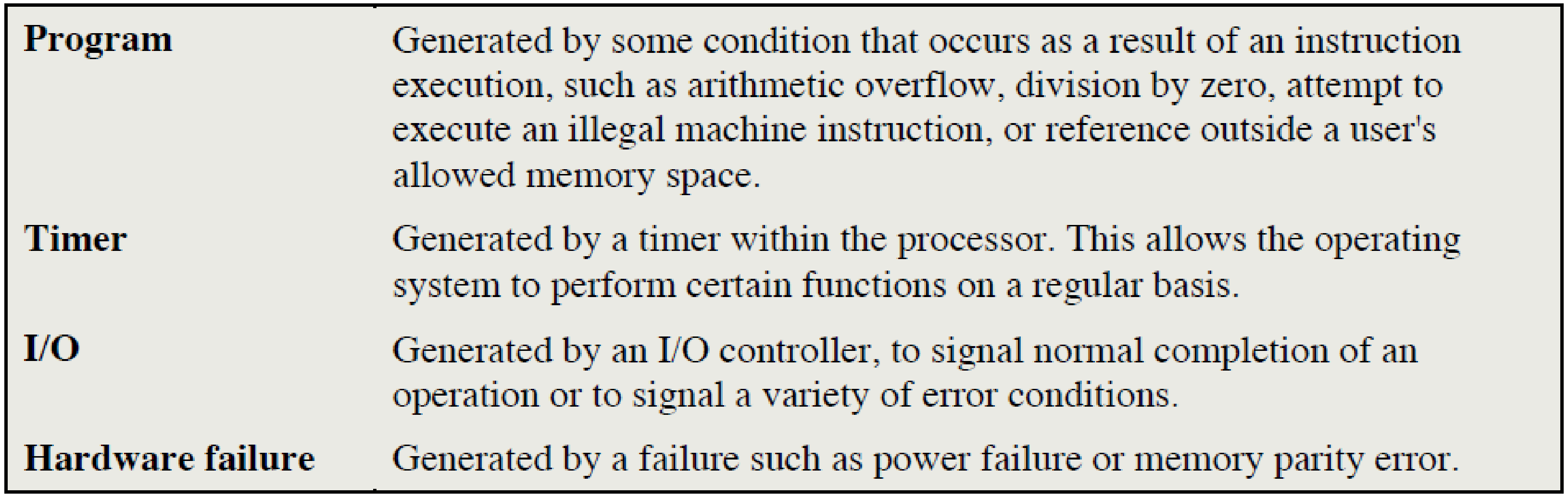

Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

## Instruction Cycle State

- **Instruction address calculation:** Determine the address of the next instruction to be executed.
- **Instruction fetch:** Read instruction from its memory location into the processor.
- **Instruction operation decoding:** Analyze instruction to determine type of operation to be performed and operand to be used
- **Operand address calculation:** If the operation involves reference to an operand in memory or via I/O, then determine the address of the operand.
- **Operand fetch:** Fetch the operand from memory or read it in from I/O.
- **Data operation:** Perform the operation indicated in the instruction
- **Operand store:** Write the result into memory or out to I/O.

## Interrupts

- **Mechanism by which other modules** (e.g. I/O, memory) **may interrupt the normal processing of the processor**.
- Interrupts are provided **to improve processing efficiency**.
- Most external devices are much slower than the processor. Suppose, the processor is transferring data. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be of many hundred or thousands of instruction cycles, which is very wasteful use of the processor. Classes of interrupts are given in the table



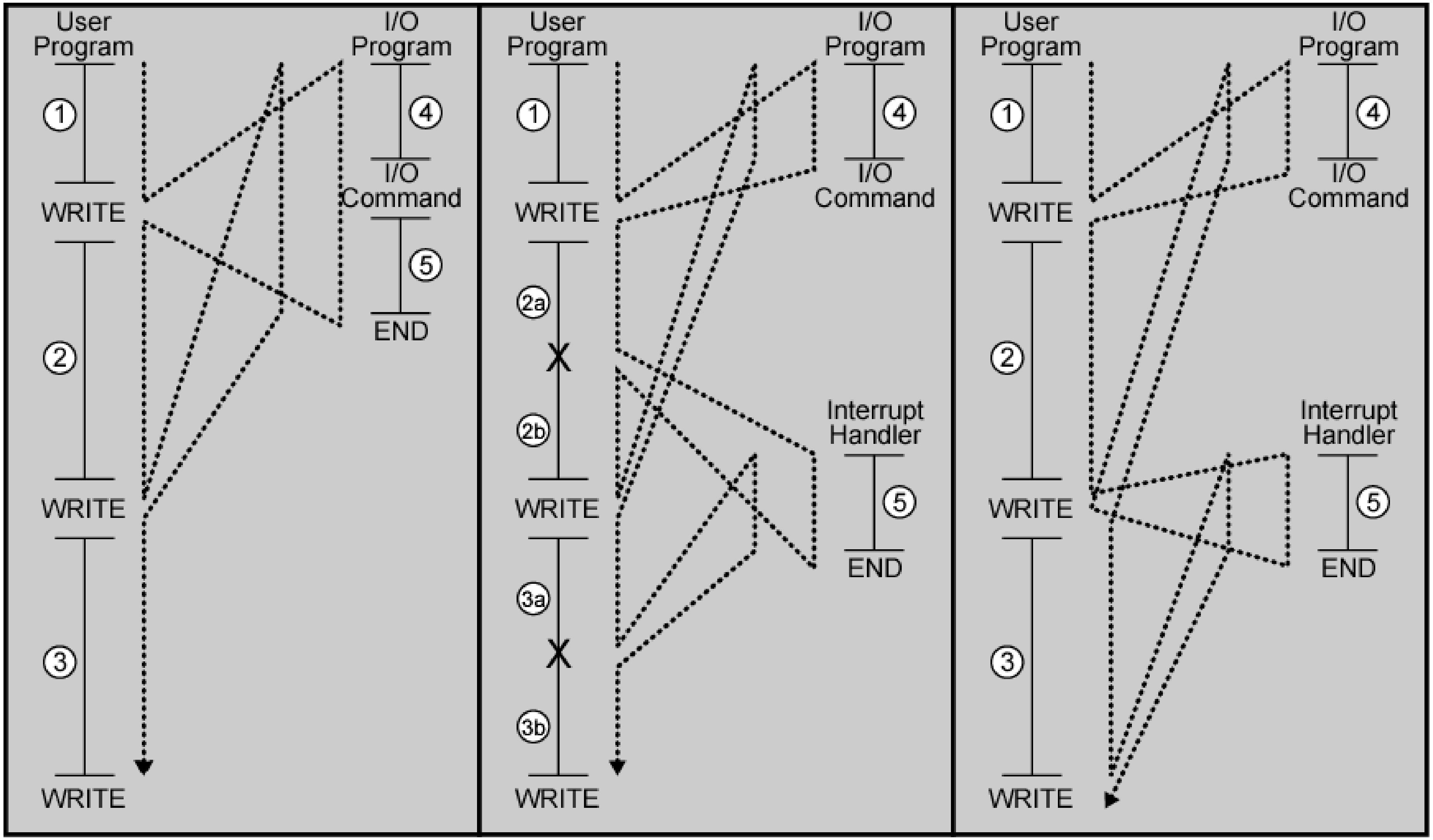
| | |
|---|---|
| **Program** | Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space. |
| **Timer** | Generated by a timer within 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 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

- The user program performs a series of **WRITE** calls. Code segments **1,2, and 3** refer to sequences of instructions that do not involve I/O.
- The I/O program consists of three sections:
  - **A sequence of instructions**, labeled 4, to prepare actual I/O operation.
  - **The actual I/O command**. Without the use of interrupts, once this command issued, the program must wait for the I/O device to perform the requested function. The program might wait by repeatedly performing a test operation to determine if the I/O operation is done.
  - **A sequence of instructions**, labeled 5, to complete the operation. This includes setting a flag indicating success or failure of the operation.

  **I/O operation takes a relatively long time to complete, the I/O program is hung up waiting for the operation to complete; hence user program is stopped at the point of the WRITE call for some considerable period of time.**

# Program Flow Control



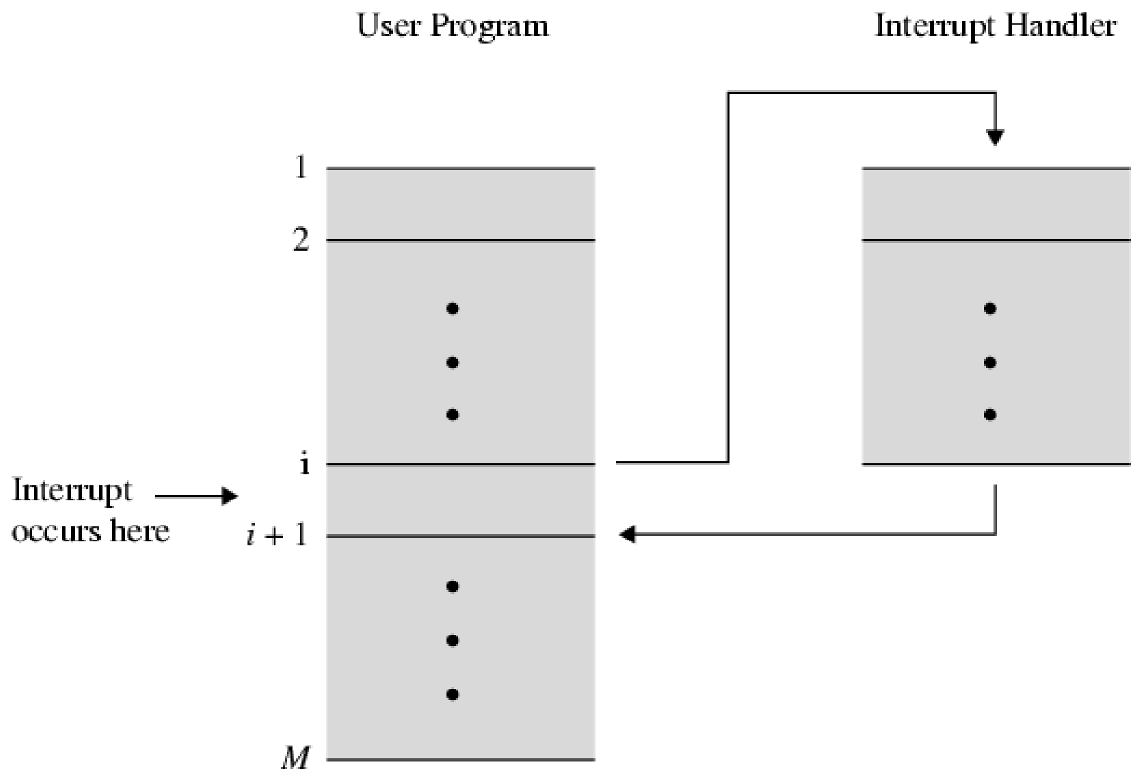
(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

## Interrupts and the Instruction Cycle

- With interrupt the processor can be engaged in executing other instructions while an I/O operation is in progress.
- The user program reaches a point at which **it makes a system call in the form of a WRITE call**.
- The **I/O program is invoked** consists of **preparation code** and actual **I/O command**. **After these few instructions, control returns to the user program**. Meanwhile, the external device is busy accepting data from computer memory and printing it.
- **When the external device becomes ready to be serviced**, i.e. when it is ready to accept more data from the processor, the I/O module for the external device sends an ***interrupt request*** signal to the processor.

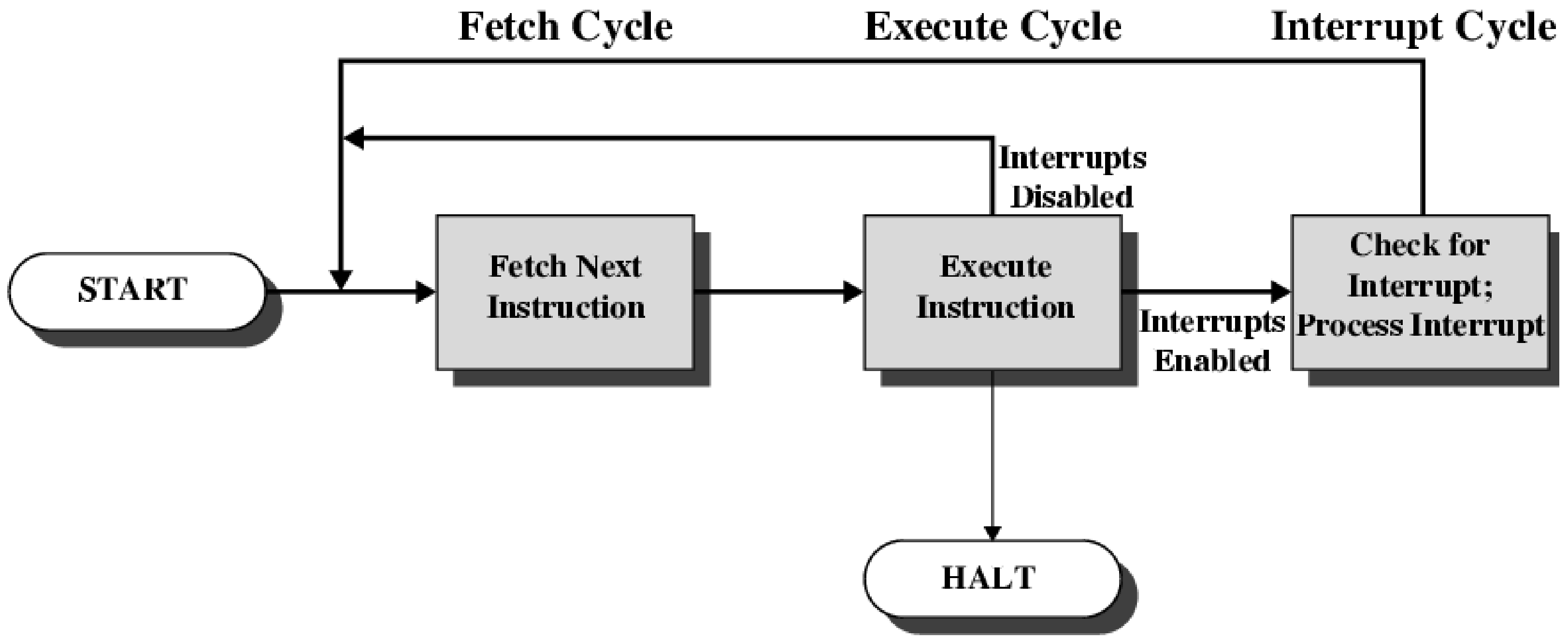
- **The processor responds by suspending operation of the current program**, branching off to a program to service that particular I/O device, known as an ***interrupt handler***.

## Transfer of Control via Interrupts



- **Interrupt is just an interruption of the normal sequence of program execution.**
- **When interrupt processing is completed, execution resumes.**
- **The processor and operating system are responsible for suspending the user program and resuming it at the same point.**

## Instruction Cycle with Interrupts

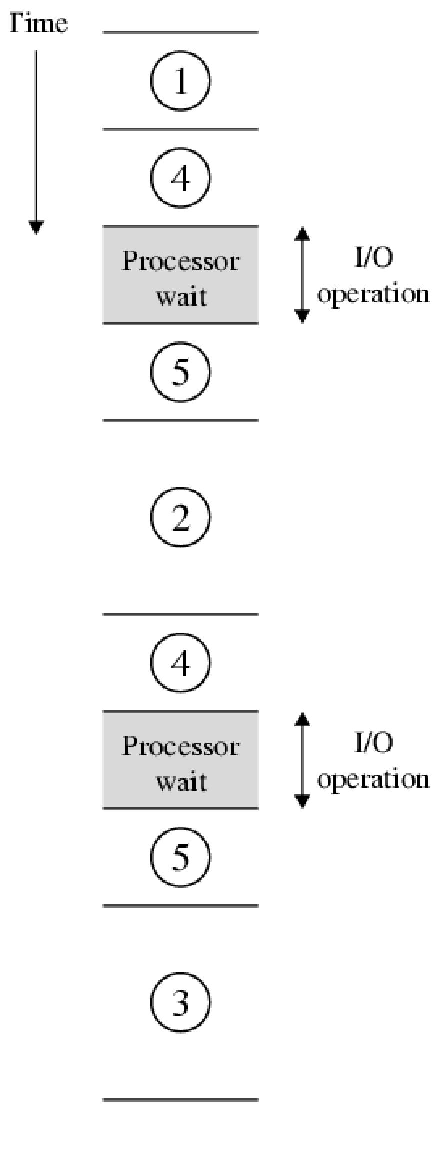


- An ***interrupt cycle*** is added to the instruction cycle shown in above Figure.
- The processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction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

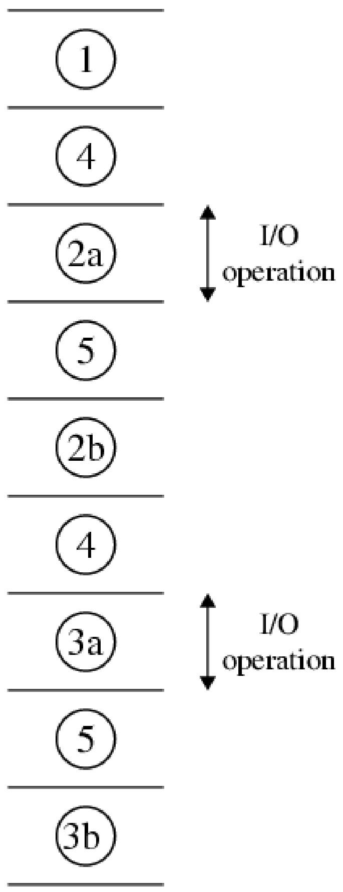
## Interrupt Cycle

- If interrupt pending the program does the following:
  - **Processor suspend execution of current program**
  - Save its context (this means **saving the address of the next instruction to be executed**)
  - Now **sets the program counter to the starting address of an *interrupt handler* routine**
  - **Process interrupt**
  - **When the *interrupt handler* is completed, the processor resume execution of the user program at the point of interruption**.

# Program Timing
# Short I/O Wait



(a) Without interrupts

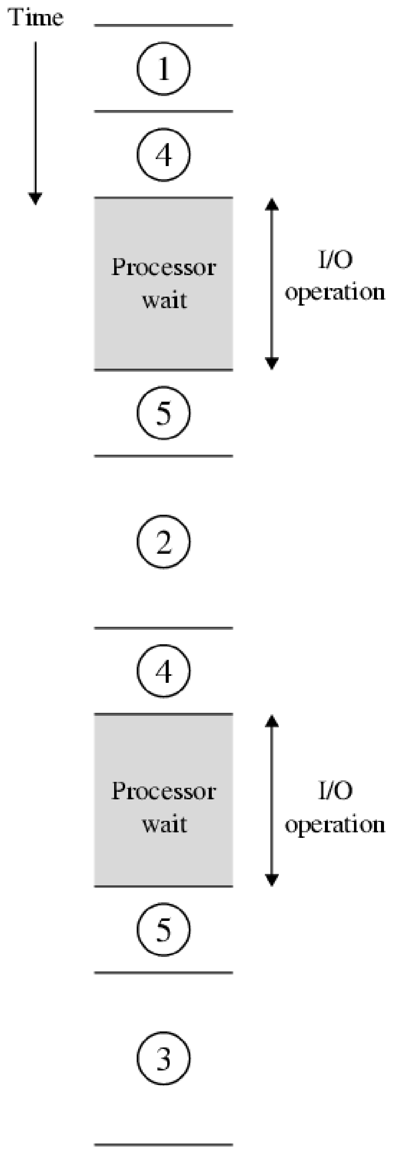


(b) With interrupts

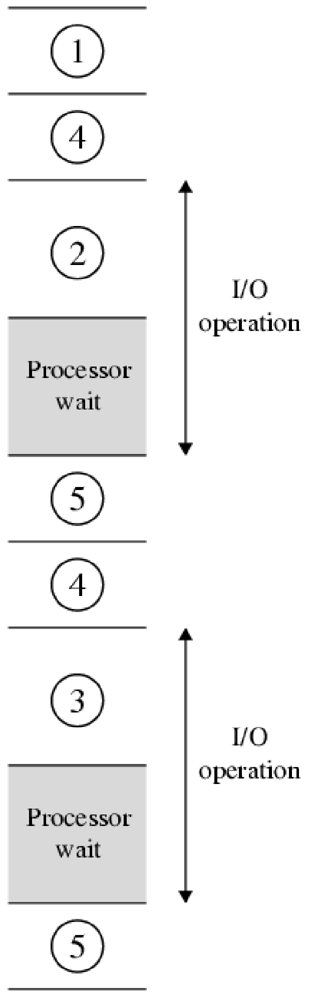
- **The time required for the I/O operation is relatively short** (**less than the time to complete the execution** of the instructions **between write operations in the user program**).

- A slow device such as printer will take much more time than executing a sequence of user instructions.

## Program Timing
## Long I/O Wait



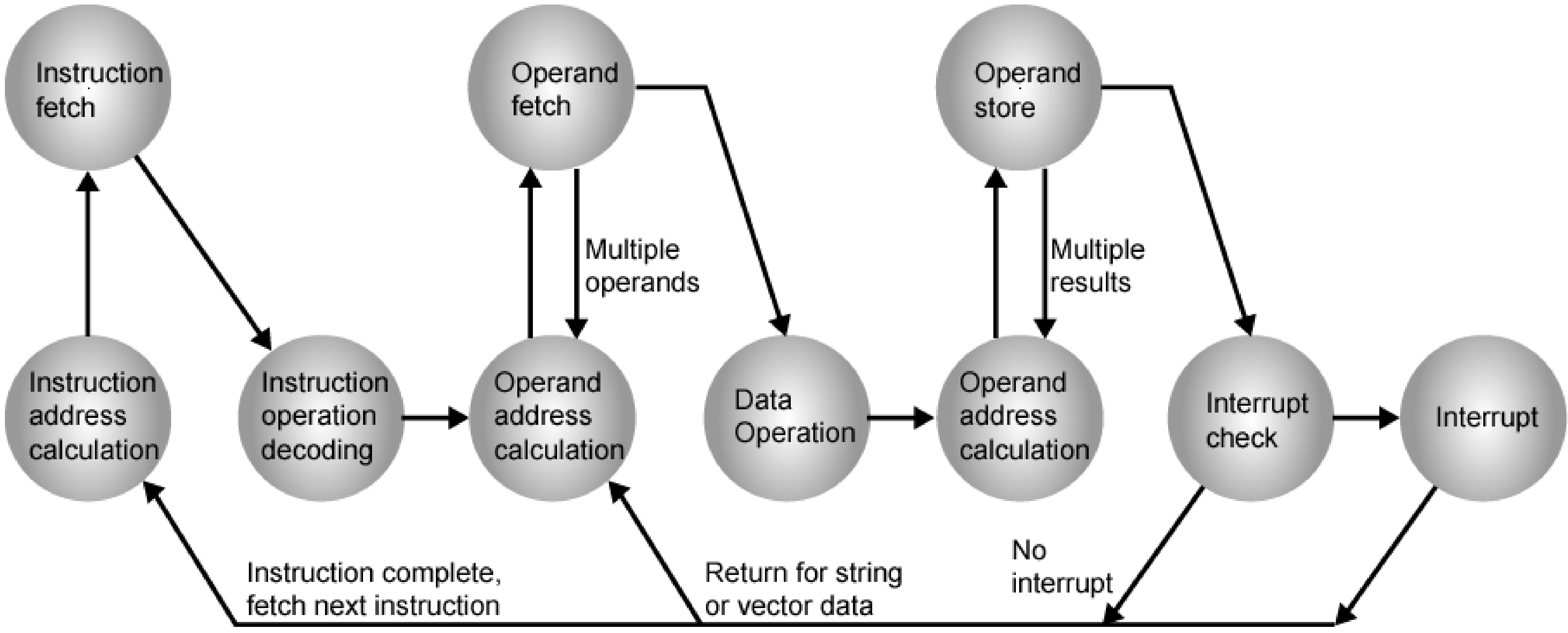
(a) Without interrupts



(b) With interrupts

- **The user program reaches the second WRITE call before the I/O initiated by the first call is complete**.
- The result is that the **user program is hung up at that point**.
- **When the preceding I/O operation is completed, this new WRITE call may be processed**, and a **new I/O operation may be started**.

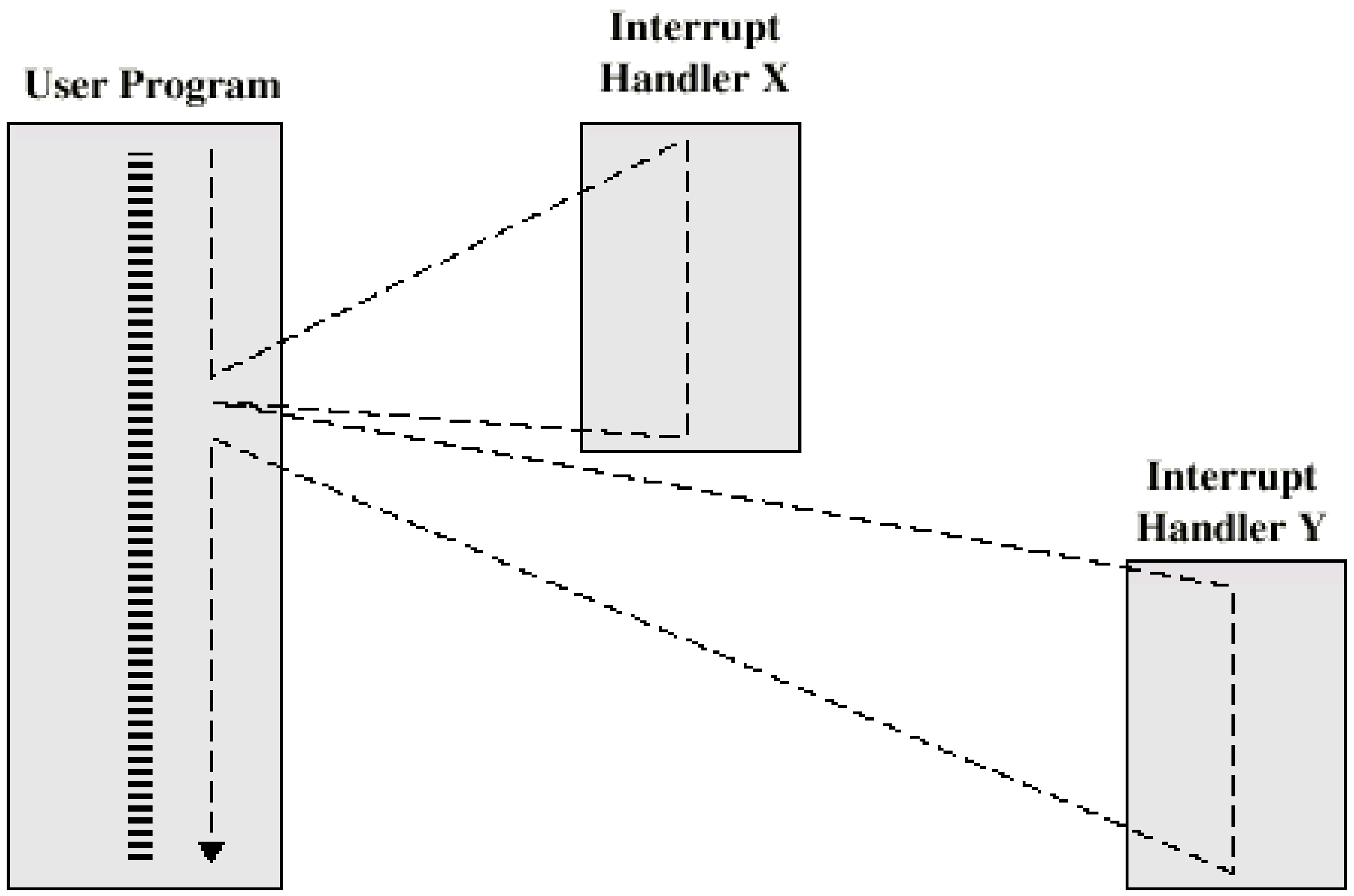
## Instruction Cycle (with Interrupts) - State Diagram


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

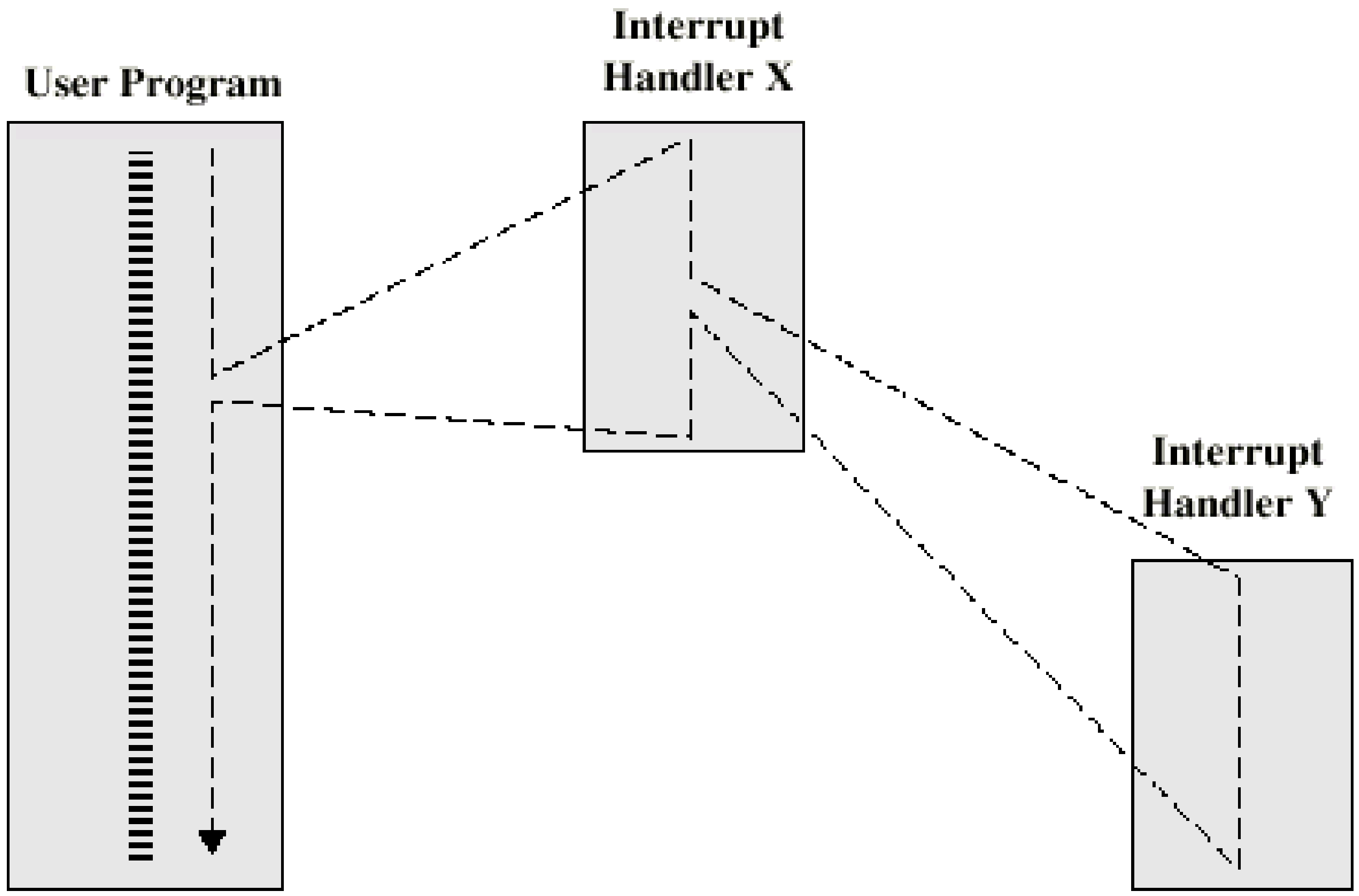
## Multiple Interrupts

- **Multiple interrupts can occur**, e.g. a program may be receiving data from a communications line and printing results. The printer will generate an interrupt for **every item it completes a print operation** and the communication line controlller will generate an interrupt **every time a unit of data arrives**.
- Disable interrupt
  - **Processor will ignore the  interrupt request signal while an** interrupt is being processed.
  - **Interrupts remain pending** and are checked after first interrupt has been processed
  - **Interrupts are handled in strict sequential order**
- Define priorities
  - **Low priority interrupts can be interrupted by higher priority interrupts**
  - **When higher priority interrupt has been processed, processor returns to previous interrupt**

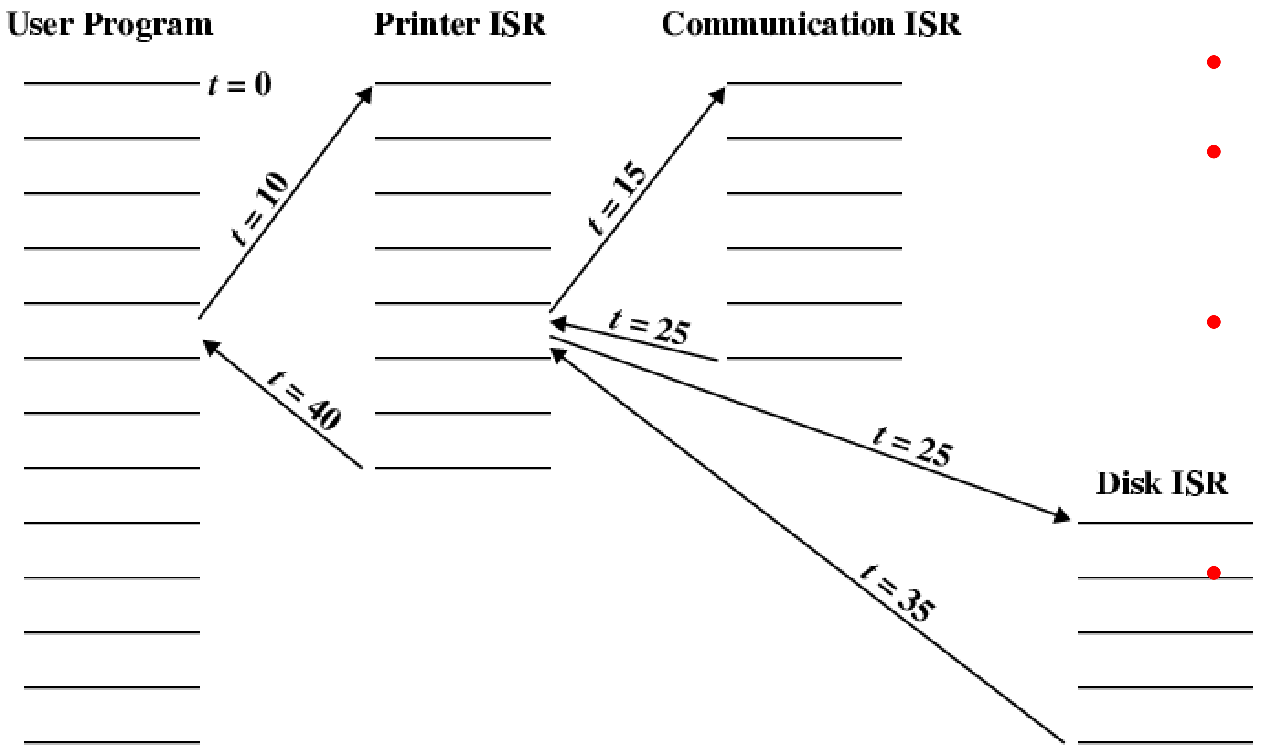
## Multiple Interrupts - Sequential



# Multiple Interrupts – Nested



# Time Sequence of Multiple Interrupts



- A user program begins at (*t=0*)
- At *t=10,* a printer interrupt occurs.
- While this routine is still executing, at *t=15* communication interrupts occurs.
- While this routine is executing, a disk interrupt occurs (*t=20* ); lower priority and communication ISR runs to completion.
- When the communications ISR is complete (*t=25*), the previous processor state is restored and back to disk ISR.
- Only when that routine is complete (*t=35*) is the printer ISR resumed.
- When that routine completes (*t=40*), control finally returns to the user program.

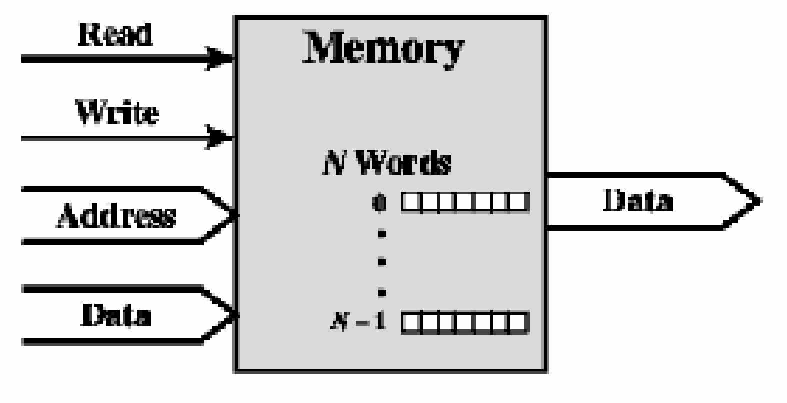
## Interconnection

- A computer consists of a set of components/modules of three basic types that communicate with each other.
  - Memory
  - Input/Output
  - CPU

- In effect, a computer is a network of basic modules. Thus, there must be path for connecting the modules.
- The collection of paths connecting the various modules is called the *interconnection structure*.

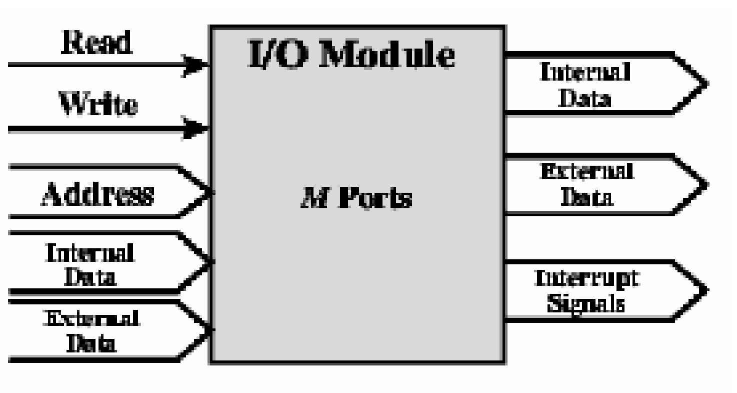
## Memory Connection

- A memory module consists of *N* words of equal length.
- Each word is assigned a unique numerical address (0,1,…, *N*-1).
- A word of **data** can be read from or written into the memory.
- The nature of the operation is indicated by **read** and **write** control signals.
- The location for the operation is specified by an **address**.
- A memory address is a number that indicate the location in a memory chip



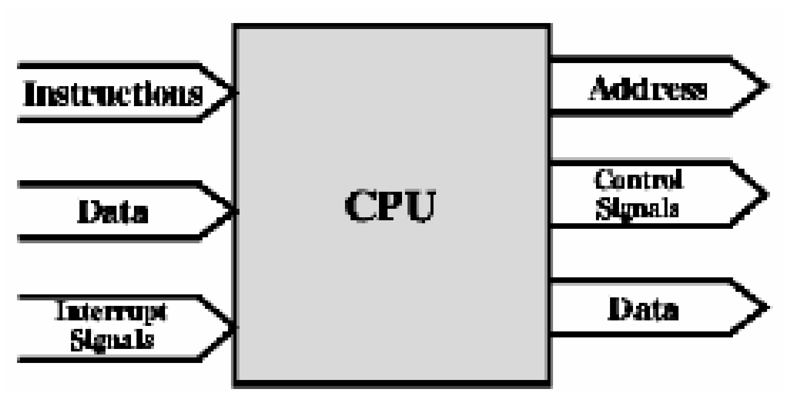
## Input/Output Connection

- Similar to memory from computer's viewpoint
- Output
  - Receive data from computer
  - Send data to peripheral
- Input
  - Receive data from peripheral
  - Send data to computer
- Receive control signals from computer
- Send control signals to peripherals
- Receive addresses from computer
  - e.g. port number to identify peripheral
- Send interrupt signals (control)



## CPU Connection

- Reads instruction and data
- Writes out data (after processing)
- Sends control signals to other units
- Receives (& acts on) interrupt signals



## Types of Data Transfers

- **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 both cases, I/O module is allowed to exchange data directly between I/O to or from memory without going through the processor.

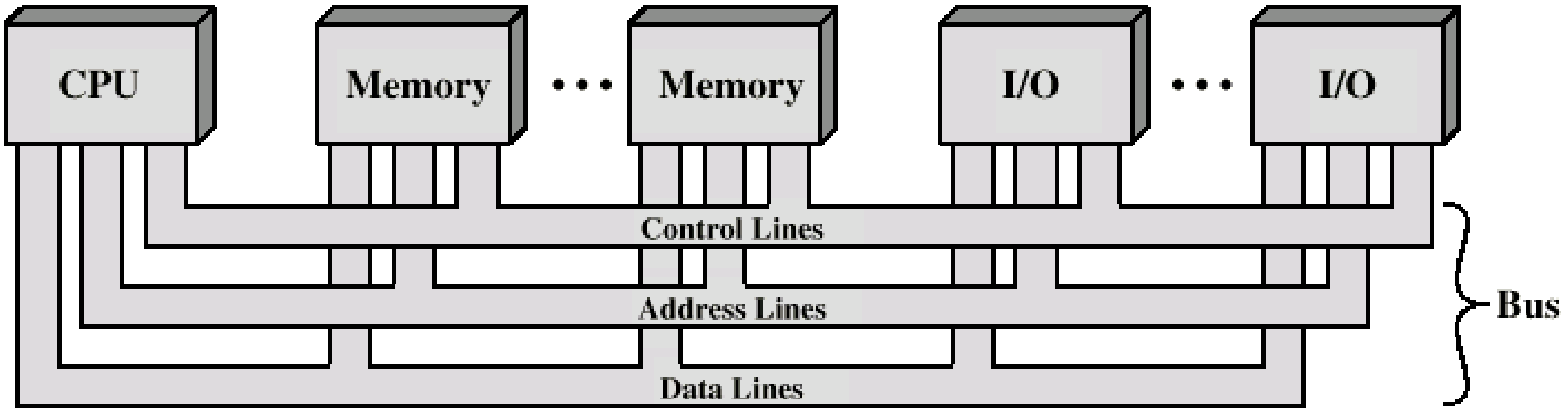
## Buses

- There are a number of possible interconnection systems
- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)
- e.g. Unibus (DEC-PDP)

## Bus Interconnection

- A bus is a communication pathway connecting two or more devices.
- Bus is a shared transmission medium.
- Multiple devices are connected to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, signals will overlap and become garbled. Only one device at a time can successfully transmit.
- A bus consists of multiple communication pathways, or lines, each of which transmit signals representing binary "1" or "0".
- Several lines of a bus can be used to transmit binary digits simultaneously (in parallel); e.g. an 8-bit data can be transferred over eight bus lines
- Computer system contains **a number of different buses** that provide pathways between components.
- A bus that connects major computer components (processor, memory, I/O) is called a ***system bus***.

## Bus Interconnection Scheme



- A system bus consists of about 50 to hundreds of separate lines.
- Each line is assigned a particular meaning or function.
- On any bus the lines can be classified into three: data, address, and control lines.

## Data Bus

- The data lines provide a path for moving data between system modules; these data lines are collectively called **the data bus**.
- The data bus may consist of 32 to hundreds of separate lines, the number of lines being referred to as the **width of the data bus**.
- Each line can carry 1 bit at a time, the number of lines determines how many bits can be transferred at a time.
- If the data bus is 8 bits wide and each instruction is 16 bits long, then processor must access the memory module twice during each instruction cycle.

## Address bus

- Identify the **source** or **destination** of data on data bus
- e.g. if the processor wishes to read a word (8, 16, or 32 bits) of data from main memory, it puts **the address of the desired word on the address line**.
- The width of the address bus determines maximum memory capacity of system
  - e.g. 8080 has 16 bit address bus giving 64k address space

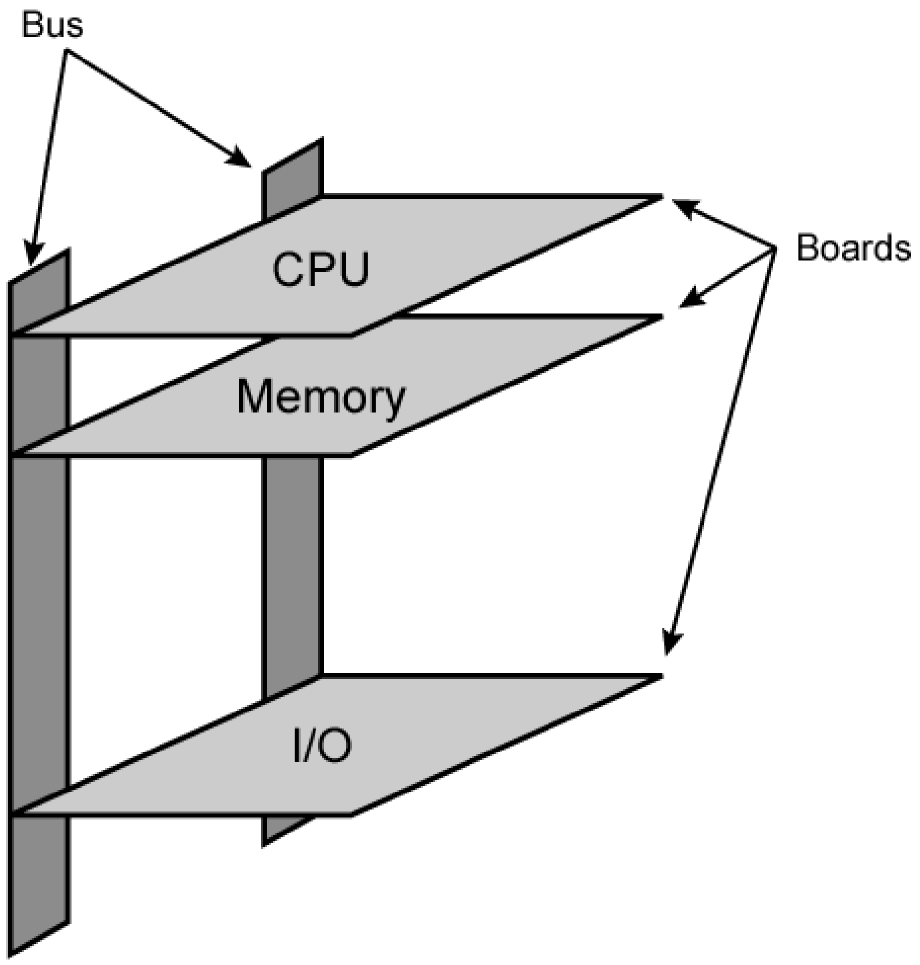
## Control Bus

- Because the data and address lines are shared by all components, **there must be a means of controlling their use**.
- Control signals transmit both **command and timing information** between system modules.
- Timing signals indicate the **validly of data and address information.**
- Command signals specify **operations to be performed**.

## Control Bus (contd...)

- Memory read: Causes data from the addressed location to be placed on the bus.
- Memory write: Causes data on the bus to be written into the addressed location.
- I/O read: Causes data from the addressed I/O port to be placed on the bus.
- I/O write: Causes data on the bus to be output to the addressed I/O port.
- Transfer ACK: Indicates data have been accepted or placed on the bus.
- Bus request: Indicates that a module needs to gain control of the bus.
- Bus grant: Indicates that request has been granted.
- Interrupt request: Indicates that an interrupt is pending.
- Interrupt ACK: Acknowledges that the pending interrupt has been recognized.
- Clock: Used to synchronize operations.
- Reset: Initializes all modules.

## Physical Realization of Bus Architecture

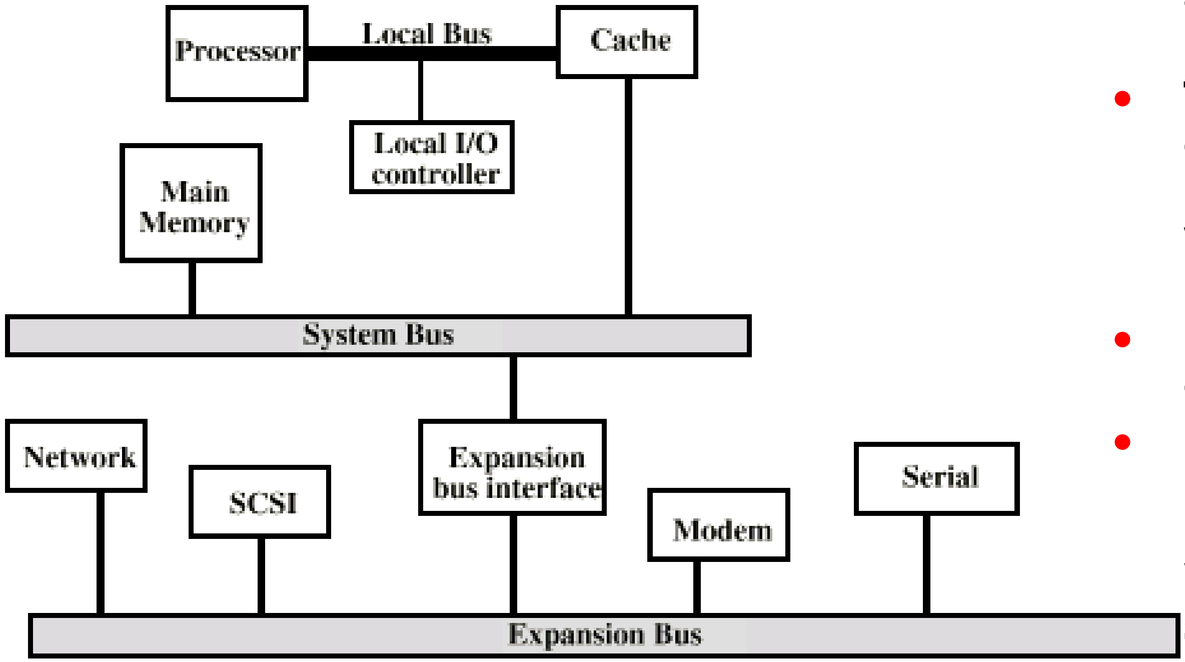


- The bus (big and yellow) is a number of parallel electric conductors.
- These conductors are metal lines etched in a printed circuit board.
- e.g. the bus consists of two vertical columns of conductors.
- At regular intervals along the columns, there are attached points in the form of slots that horizontally support a printed circuit board.
- Each of the major system components occupies one or more boards and plugs into the bus at these slots.

## Single Bus Problems

- Lots of devices on one bus leads to:
  - Propagation delays (the greater the bus length, the greater the propagation delay)
    - This delay determines the time it takes for devices to co-ordinate the use of the bus. When control of the bus passes from one device to another frequently, **these propagation delay can adversely affect performance**
    - **This problem can solved** to some extent **by increasing the data rate that the bus can carry** and **by using wider data buses** (from 32 to 64 bits)
- Most systems use multiple buses to overcome these problems as data rates generated by **graphics and video** controllers are growing rapidly.

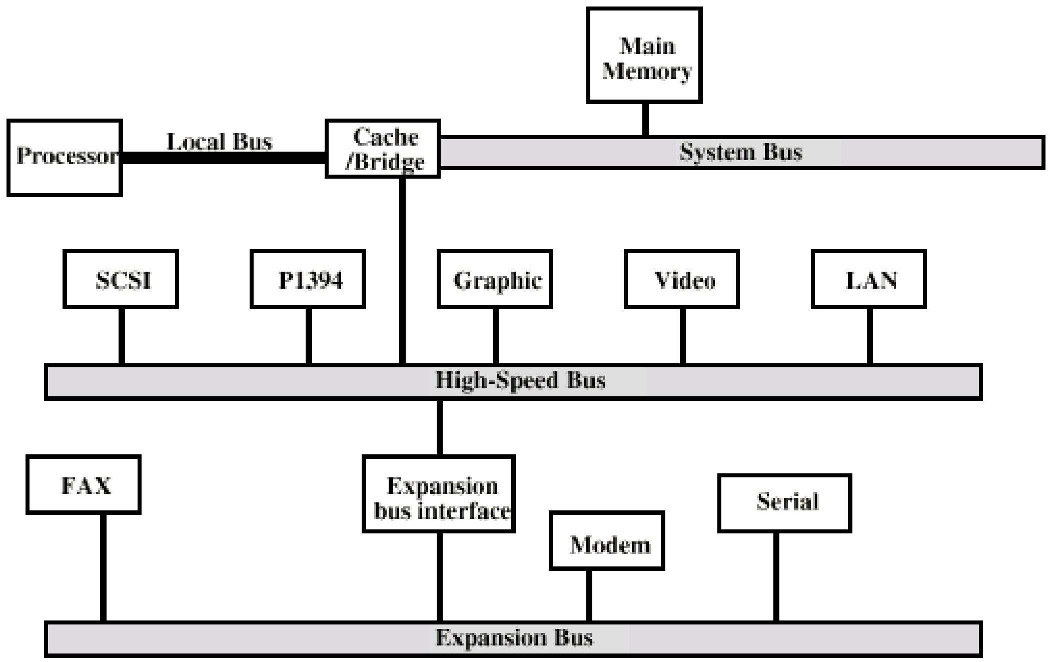
# Traditional (ISA) (with cache)



- A local bus connects the processor to a cache memory and may support one or more devices.
- The cache memory controller connects the cache to local bus and to a system bus where main memory modules are attached.
- I/O devices attached to the expansion bus.
- Network connection includes LANs such as 10-Mbps Ethernet connections to WANs such as packet-switching network. SCSI (small computer system interface) is itself a type of bus used to support local disk drives and other peripherals. A serial port used to support a printer/scanner.

- In this way, I/O transfers to and from the main memory across the system but don't interfere processor's activity.

# High Performance Bus



- The advantage of this arrangement is that the high speed bus brings high demand devices into closer integration with the processor and <u>at the same time is independent of the processor.</u>

- A local bus that connects the processor to a cache controller, which is in turn connected to the system bus that supports main memory.
- The cache controller is integrated into a bridge, that connects to the high-speed bus.
- This bus supports connections to high-speed LANs, such as Fast Ethernet at 100 Mbps, video and graphics workstation controller, as well as interface controllers to local peripheral buses, including SCSI and FireWire.
- High-speed bus designed to support high capacity I/O devices.
- Low-speed devices are still supported off an expansion bus

## Bus Types

Dedicated

- Separate dedicated address & data lines

- Multiplexed
  - Address and data information may be transmitted over the same set of lines using an Address Valid control line. At the beginning of data transfer, address is placed on the bus and Address Valid line is activated. At this point, each module has a specific time period to copy the address and determine if it is in the address module. The address then removed from the bus, and same bus connections are used for subsequent read/write data transfer.
  - Advantage - fewer lines which saves space and cost
  - Disadvantages
    - More complex circuitry is needed within each module
    - Potential reduction in performance because certain events that share the same lines cannot take place in parallel.

## Bus Arbitration

- More than one module controlling the bus
- e.g. I/O module may need to read or write directly to memory, without sending the data to the processor.
- Because only one unit at a time can successfully transmit over the bus, some method of arbitration is needed.
- Arbitration may be centralised or distributed

## Centralised or Distributed Arbitration

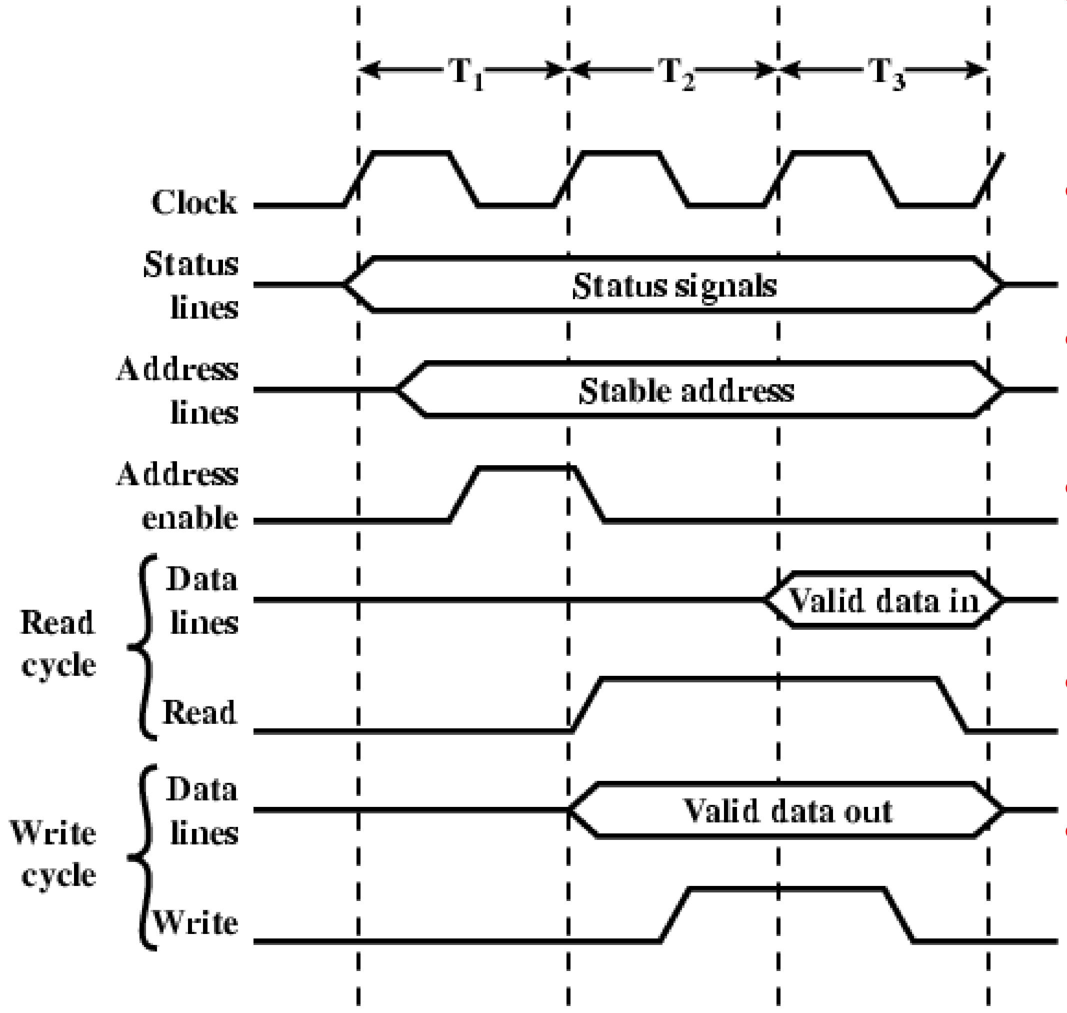
- Centralised
  - Single hardware device controlling bus access (responsible for allocating time on the bus)
    - Bus Controller
    - Arbiter
  - May be part of CPU or separate
- Distributed
  - There is no central controller
  - Each module contains access control logic and the modules act together to share the bus.

➢ **With both methods of arbitration, the purpose is to designate one device, either the processor or an I/O module, as master. The master then initiate a data transfer (read/write) with some other devices that acts as slave for this particular exchange.**

## Timing

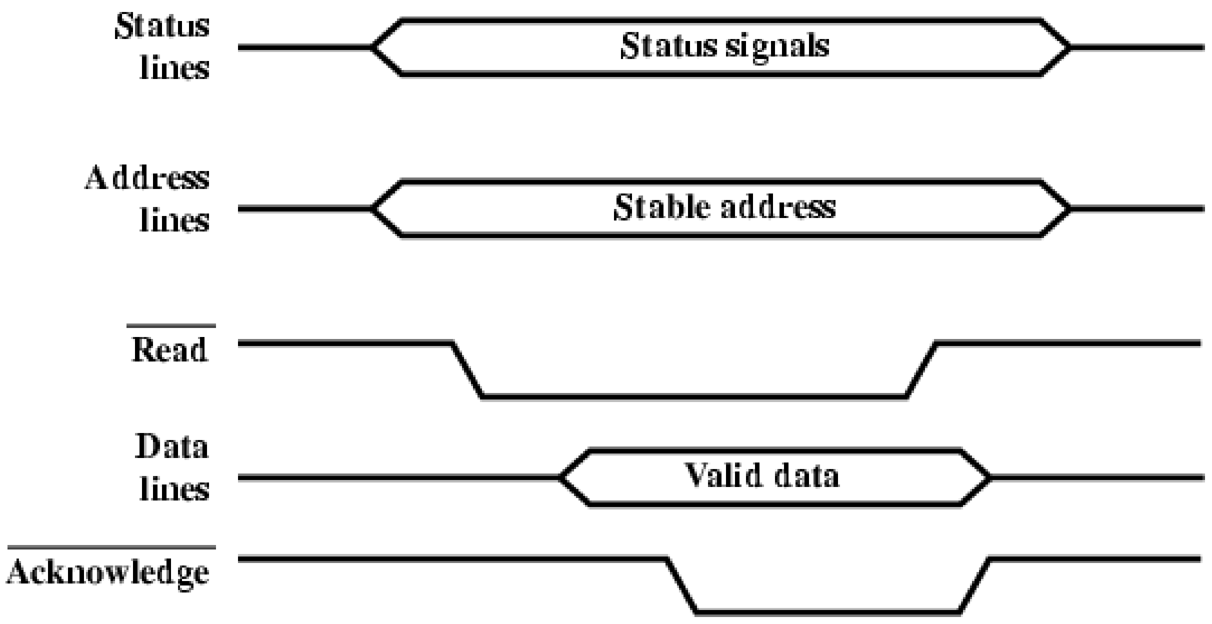
- Timing refers to co-ordination of events on bus. Buses use synchronous timing and asynchronous timing.
- Synchronous
  - The occurrence of events on the bus is determined by a clock. **The clock defines equal width time slot.**
  - The bus includes clock line upon which a clock transmits a regular sequence of 1s and 0s of equal duration.
  - A single 1-0 transmission is referred to as *bus cycle* or *clock cycle*
  - All devices on the bus can read clock line and all events start at the beginning of clock cycle.

# Synchronous Timing Diagram



- The processor places a memory address on the address lines during the first clock cycle and may assert various status lines
- Once the address lines have stabilized, the processor issues an address enable signal.
- For a read operation, the processor issues a read command at the start of the second cycle.
- A memory module recognizes the address and after a delay of one cycle, places the data on the data lines.
- The processor reads the data from the data lines and drops the read signal.
- For a write operation, the processor puts the data on the data lines at the start of the second cycle, and issues a write command after the data lines have stabilized.
- The memory modules copies information from the data lines during third clock cycle.

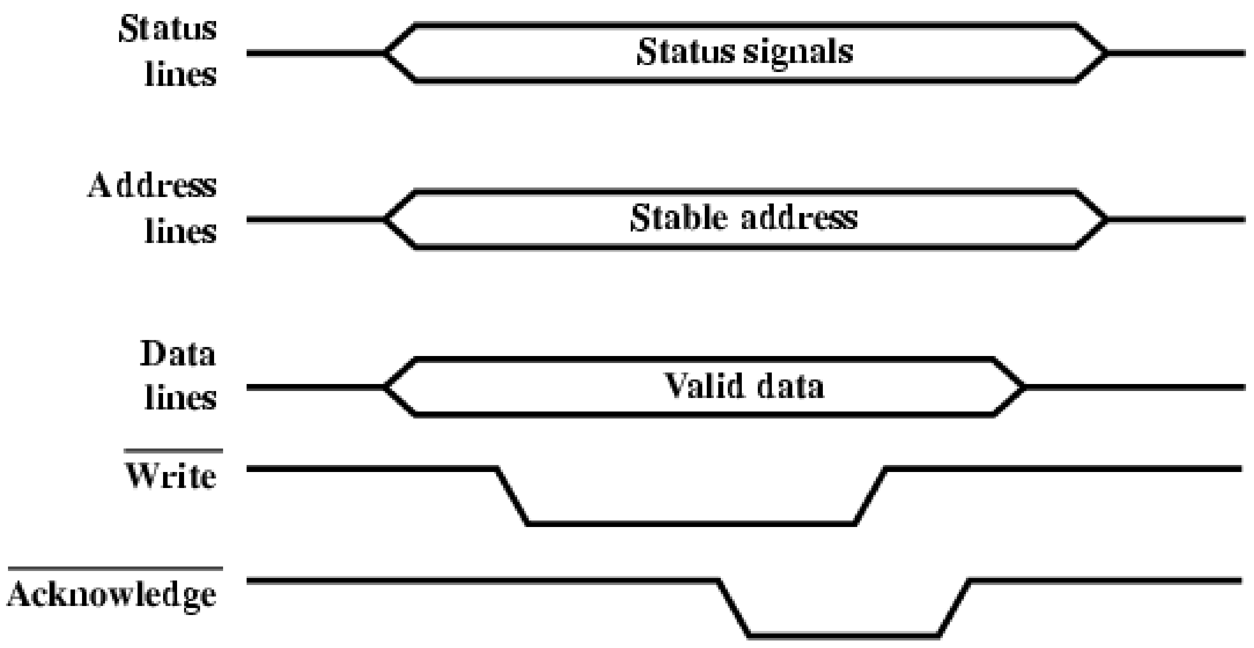
# Asynchronous Timing – Read Diagram



- With asynchronous timing, the occurrence of one event on a bus follows and depends on the occurrence of a previous event.
- The processor places <u>address</u> and <u>status signals on the bus</u>.
- After pausing for these signals to stabilize, it issues a read command, indicating the presence of valid address and control signals.
- The appropriate memory decodes the address and responds by placing the data on the data line.
- Once the data lines have stabilized, the memory module asserts the acknowledge line to signal the processor that the data are available.

- Once the master has read the data from the data lines, it deasserts the read signal.
- This causes the memory module to drop the data and acknowledge lines.
- Finally, once the acknowledge line is dropped, the master removes the address information.

# Asynchronous Timing – Write Diagram



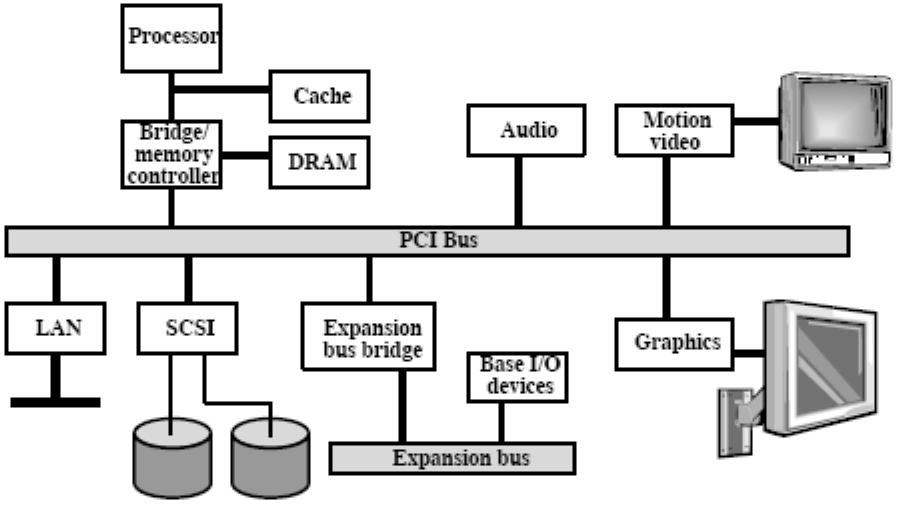
- The master places the data on the data line at the same time, that is, put signal on the status and address lines.
- The memory module responds to the write command by coping the data from the data lines and then asserting the acknowledge line.
- The master then drops the write signals and memory module drops the acknowledge signal.

- Synchronous timing is simpler to implement and test. It is less flexible than asynchronous timing, because all devices are tied to a fixed clock rate.
- With asynchronous timing, a mixture of slow and fast devices, using older and newer technology, can share a bus.

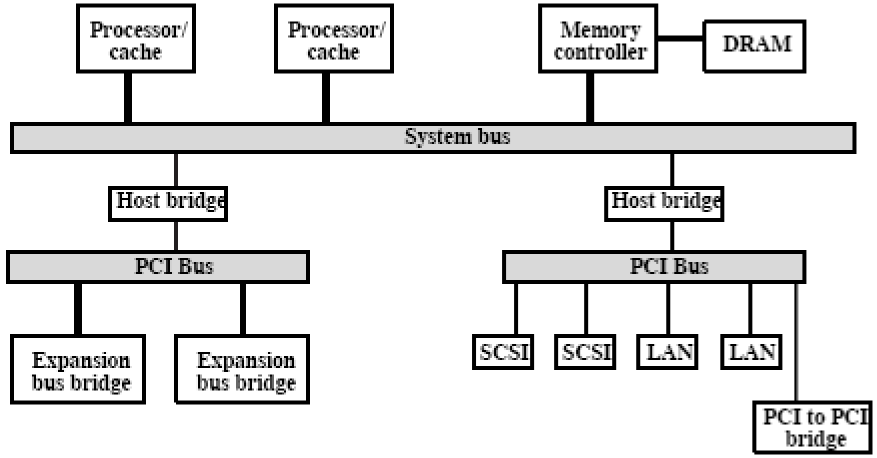
## PCI Bus

- Peripheral Component Interconnection (PCI) is a popular **high-bandwidth**, **processor independent bus** that can function as **peripheral bus**.
- Compared with other common bus specifications, **PCI delivers better system performance for high speed I/O subsystems** (graphics display adapters, network interface controllers, disk controllers, and so on.)
- The current standard allows the use of up to 64 data lines at 66 MHz, for a raw transfer rate of 528 MByte/s, or 4.224 Gbps.
- PCI is specifically designed to meet economically the I/O requirements of modern system; requires very few chips to implement and supports other buses attached to the PCI bus.

# PCI Configuration



(a) Typical desktop system



(b) Typical server system

- (a) A combined DRAM controller and bridge to the PCI bus provides tight coupling with the processor and ability to data at high speed. The bridge acts as a data buffer so that the speed of PCI may differ from that of the processor’s I/O capability.
- (b) In a multiprocessor system, one or more PCI configurations may be connected by bridges to the processor’s system bus. The system bus supports only the processor/cache units, main memory, and PCI bridges. The use of bridges keeps PCI independent of the processor speed yet provides the capability to receive and deliver data rapidly.

## PCI Bus Lines (required)

- Systems pins
  - Include the clock and reset pins
- Address and data pins
  - Include 32 lines that are time multiplexed for addresses/data
  - The other lines are used to interrupt and validate lines that carry addresses and data.
- Interface control pins
  - Control the timing of transactions and provide coordination among initiators and targets.
- Arbitration pins
  - There are not shared lines
  - Direct connection to PCI bus arbiter
- Error lines
  - Used to report parity and other errors.

## PCI Bus Lines (Optional)

- Interrupt pines
  - These are not shared lines. Each PCI device has its own interrupt line or lines to an interrupt controller.
- Cache support

  - These pins are needed to support a memory on PCI that can be cached in the processor or another device.
- 64-bit Bus Extension
  - Include 32 lines that are time multiplexed for addresses and data.
  - Other lines are used to interpret and validate the signal lines that carry the addresses and data.
  - 2 lines to enable PIC devices to agree to the use of the 64-bit transfer
- JTAG/Boundary Scan
  - For testing procedures