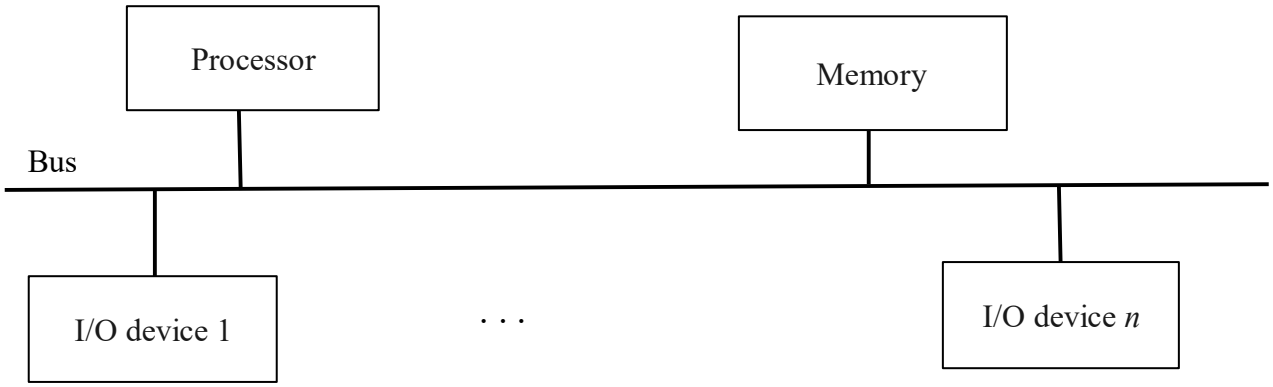
Module 2
Input output organization

A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement. The bus enables all the devices connected to it to exchange information. It consists of three sets of lines used to carry address, data, and control signals. Each I/O device is assigned a unique set of addresses. When the processor places a address on the address line, the device that recognizes this address responds to the commands issued on the control lines. The processor requests either a read or a write operation, and the requested data are transferred over the data lines, when I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.



A single-bus Structure

Memory-mapped I/O:

→ memory and I/O share the same address space.
→ reading and writing are similar to memory read/write. (same instructions are used)
→ uses same memory read and write signals.
→ Most computer systems use memory-mapped I/O.

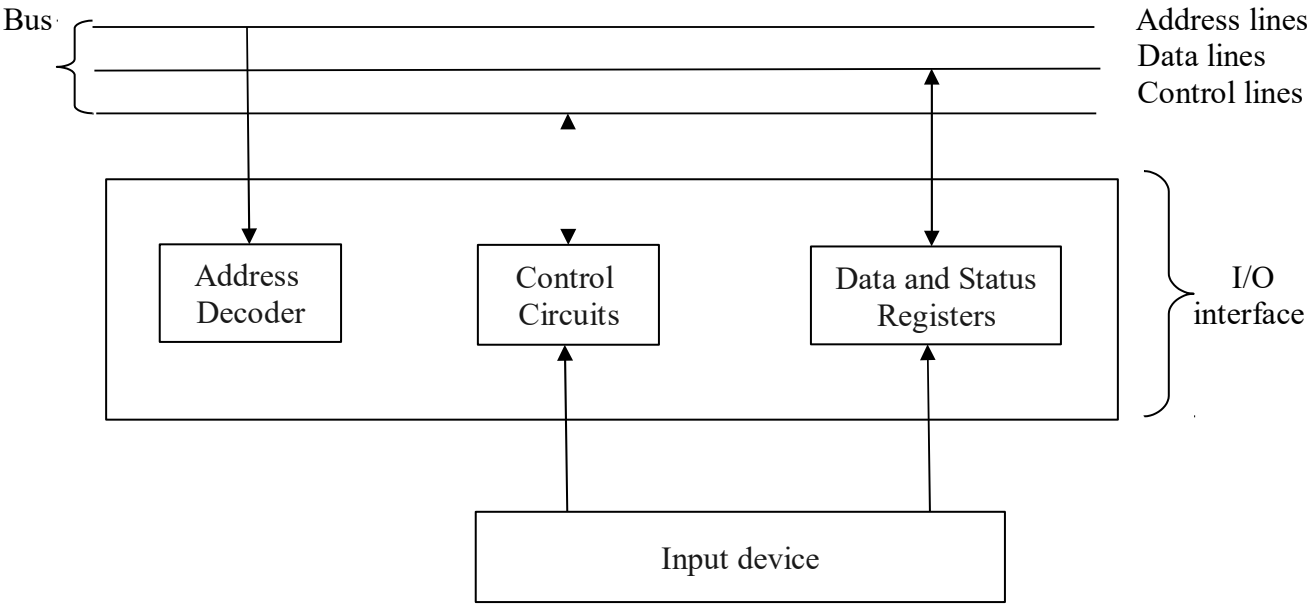
I/O mapped I/O:

→ uses separate address space.
→ less address lines are used for accessing I/O.
→ separate I/O read and write signals are needed.
→ In and Out instructions are used to perform I/O transfers.
→ Pentium supports I/O mapped.

In **program-controlled I/O**, the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. The processor polls the device.

I/O Interface

- The hardware required to connect an I/O device to the bus.
- The address decoder enables the device to recognize its address when this address appears on the address lines.
- The data register holds the data being transferred to or from the processor.
- The status register contains information relevant to the operation of the I/O device. Both the data and status registers are connected to the data bus and assigned unique addresses.
- The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's interface circuit.



Registers in keyboard and display interfaces

DATAIN

DATAOUT

STATUS

				DIRQ	KIRQ	SOUT	SIN

CONTROL

				DEN	KEN		
7	6	5	4	3	2	1	0

Three types of data transfers:

1. Program-controlled
2. Interrupts
3. DMA (Direct Memory Access)

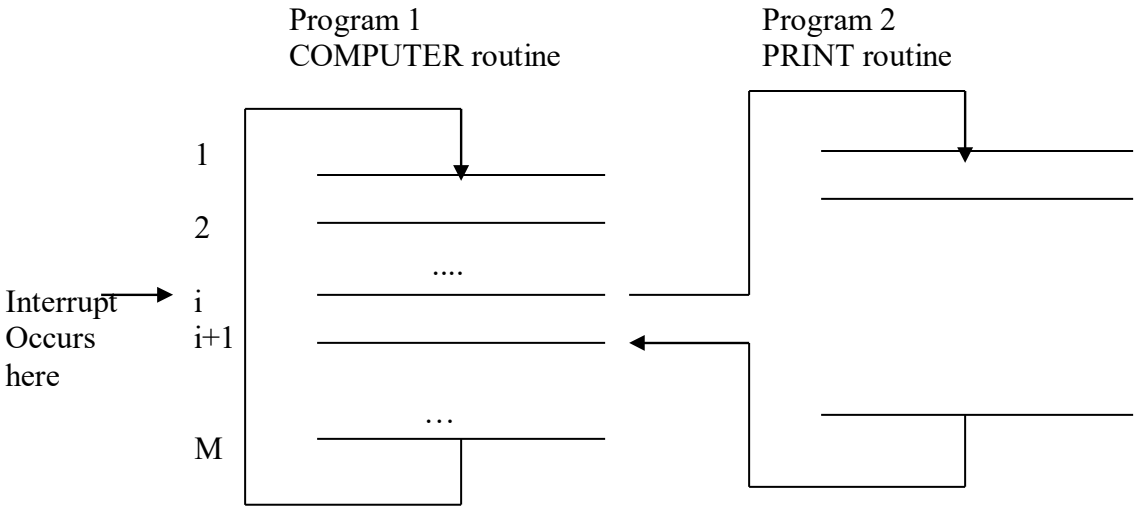
A program that reads one line from keyboard, stores it in memory buffer, and echoes it back to the display. Below program is an example for **program controlled I/O**.

```
        Move      #LINE, R0          Initialize memory pointer.
WAITK   TestBit   #0, STATUS         Test SIN.
        Branch=0  WAITK              wait for character to be entered
        Move      DATAIN, R1         Read character.
WAITD   TestBit   #1, STATUS         Test SOUT.
        Branch=0  WAITD              Wait for display to become ready.
        Move      R1, DATAOUT        Send character to display.
        Move      R1, (R0)+          Store character and advance the pointer.
        Compare   #$0D, R1           Check If Carriage Return.
        Branch=0  WAITK              If not, get another character.
        Move      #$0A, DATAOUT      Otherwise, send Line Feed.
        Call      PROCESS            Call a subroutine to process the input line.
```

Interrupts:

INT: Interrupt is a hardware signal sent by I/O device when it is become ready to alert the processor.
INTR: At least one of the control lines called interrupt request line is usually dedicated for this purpose.
INTA: interrupt-acknowledge signal is a special signal issued by the processor to the device that its request has been recognized so that it may remove its interrupt request signal.
ISR: The routine executed in response to an interrupt request is called the interrupt-service routine.

Assume that an interrupt request arrives during execution of instruction i



Transfer of control using interrupts

The processor first completes execution of instruction i. Then, it loads the program counter with the address of the first instruction of the interrupt-service routine. After execution of the interrupt-service routine, the processor has to come back to instruction i +1. Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction i+1, must be put in temporary storage in a known location. A Return-from-interrupt instruction at the end of the interrupt-service routine reloads the PC from the temporary storage location, causing execution to resume at instruction i +1. In many processors, the return address is saved on the processor stack.

Saving and restoring registers during interrupt

- Before starting execution of the interrupt-service routine, any information that may be altered during the execution of that routine must be saved.
- This information must be restored before execution of the interrupt program is resumed.
- The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.
- The task of saving and restoring information can be done automatically by the processor or by program instructions.
- Saving registers also increase the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. This delay is called **interrupt latency.**
- All registers are saved automatically by the processor hardware at the time an interrupt request is accepted. The data saved are restored to their respective registers as part of the execution of the Return-from interrupt instruction.

INTERRUPT HARDWARE: -

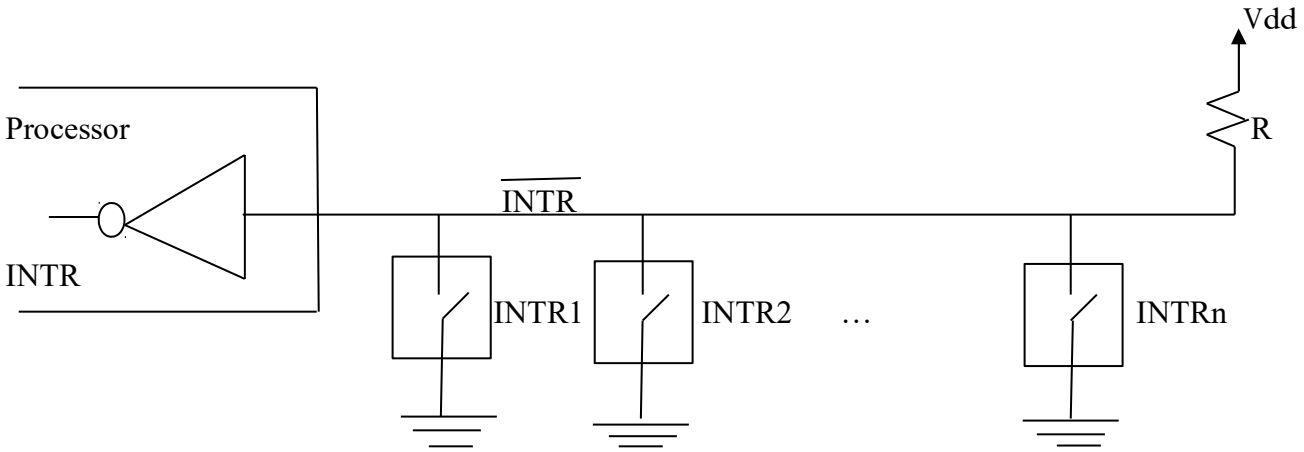
- A single interrupt-request line may be used to serve n devices as depicted.
- All devices are connected to the line via switches to ground.
- To request an interrupt, a device closes its associated switch.
- Thus, if all interrupt-request signals $INTR_1$ to $INTR_n$ are inactive, that is, if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}. This is the inactive state of the line.
- Since the closing of one or more switches will cause the line voltage to drop to 0, the value of INTR is the logical OR of the requests from individual devices, that is,

$$INTR = INTR_1 + \ldots\ldots\ldots + INTR_n$$

It is customary to use the complemented form, $\overline{INTR}$, to name the interrupt-request signal on the common line, because this signal is active when in the low-voltage state.

Open-collector are used to drive the $\overline{INTR}$ line. The output of the open-collector gate is equivalent to the switch to ground that is open when the gate's input is in the 0 state and closed when it is in the state 1.

Resistor R is called a pull-up resistor because it pulls the line voltage up to the high-voltage state when the switches are open.



ENABLING AND DISABLING INTERRUPTS:

When a device activates the interrupt-request signal, it keeps this signal activated until it learns that the processor has accepted its request. This may lead to recursive requests from the same device. This can be handled using 3 techniques, namely:

- Ignore
- Disable
- Use special hardware line

The **first possibility** is to have the processor hardware **ignore** the interrupt-request line until the execution of the first instruction of the interrupt-service routine has been completed. Then, by using an Interrupt-disable instruction as the first instruction in the interrupt-service routine, the programmer can ensure that no further interruptions will occur until an Interrupt-enable instruction is executed. Typically, the Interrupt-enable instruction will be the last instruction in the interrupt-service routine before the Return-from-interrupt instruction.

The **second option**, which is suitable for a simple processor with only one interrupt-request line, is to have the processor automatically disable interrupts before starting the execution of the interrupt- service routine. After saving the contents of the PC and the processor status register (PS) on the stack, the processor performs the equivalent of executing an Interrupt-disable instruction. It is often the case that one bit in the **PS register (PSW- Processor Status Word)** , called Interrupt-enable, indicates whether interrupts are enabled.

In the **third option**, the processor has a special interrupt-request line for which the interrupt- handling circuit responds only to the leading edge of the signal. Such a line is said to be edge-triggered.

HANDLING MULTIPLE DEVICES: -

When a request is received over the common interrupt-request line, additional information is needed to identify the device that activated the line.

The information needed to determine whether a device is requesting an interrupt is available in its status register. When a device raises an interrupt request, it sets IRQ bit to 1, which is in its status register. For example, bits KIRQ and DIRQ are the interrupt request bits for the keyboard and the display, respectively. The simplest way to identify the interrupting device is to have the interrupt-service routine poll all the I/O devices connected to the bus. The first device encountered with its IRQ bit set is the device that should be serviced. An appropriate subroutine is called to provide the requested service.

The polling scheme is easy to implement. Its main disadvantage is the time spent interrogating the IRQ bits of all the devices that may not be requesting any service. An alternative approach is to use vectored interrupts.

Vectored Interrupts: -

- Used to reduce the time involved in the polling process, a device requesting an interrupt may identify itself directly to the processor.
- Then, the processor can immediately start executing the corresponding interrupt-service routine.
- The term vectored interrupts refer to all interrupt-handling schemes based on this approach.
- A device requesting an interrupt can identify itself by sending a special code to the processor over the bus. This enables the processor to identify individual devices even if they share a single interrupt-request line.
- The code supplied by the device may represent the starting address of the interrupt-service routine for that device.
- The code length is typically in the range of 4 to 8 bits.
- The remainder of the address is supplied by the processor based on the area in its memory where the addresses for interrupt-service routines are located.
- This arrangement implies that the interrupt-service routine for a given device must always start at the same location.
- The programmer can gain some flexibility by storing in this location an instruction that causes a branch to the appropriate routine.
- The processor reads this address called the interrupt vector and loads it into the PC

When the device sends an interrupt request the processor may not be ready to receive the vector code immediately. It must first complete the execution of the current instruction, which may require the

use of the bus. When the processor is ready to receive the vector code, it activates the INTA line. The I/O device responds by sending it interrupt vector code and turning of the INTR signal.

Interrupt Nesting: -

- Interrupts should be disabled during the execution of an ISR, to ensure that a request from one device will not cause more than one interruption.
- Execution of a given ISR, once started, always continues to completion before the processor accepts an interrupt request from a second device.
- But an interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device.

To implement this scheme, assign a priority level to the processor that can be changed under program control. The priority level of the processor is the priority of the program that is currently being executed. The processor accepts interrupts only from devices that have priorities higher than its own.

A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt-acknowledge lines for each device, as shown in figure. Each of the interrupt-request lines is assigned a different priority level.

Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor. A request is accepted only if it has a higher priority level than that currently assigned to the processor.

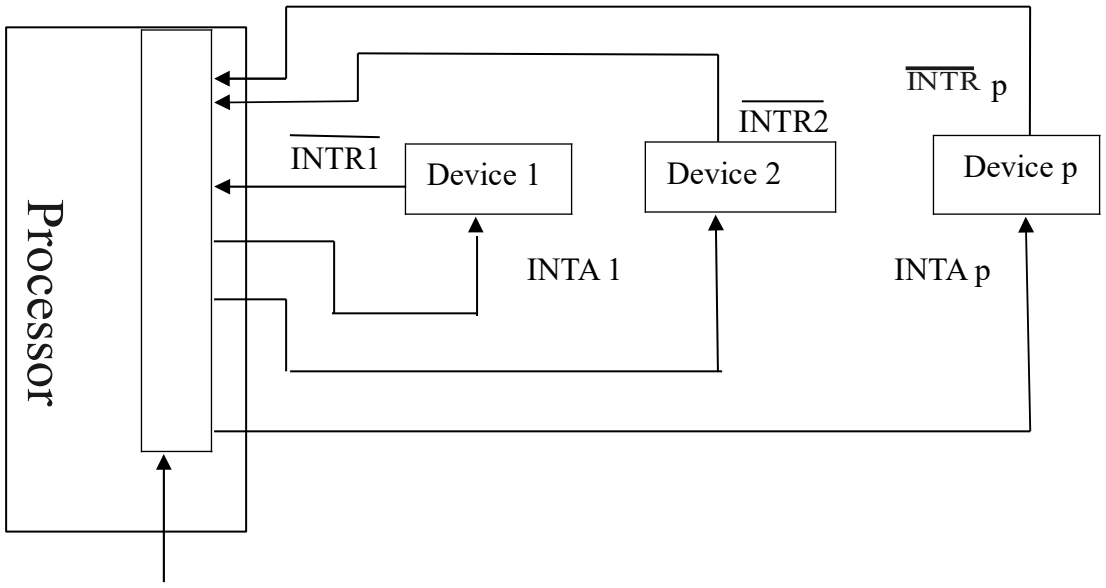


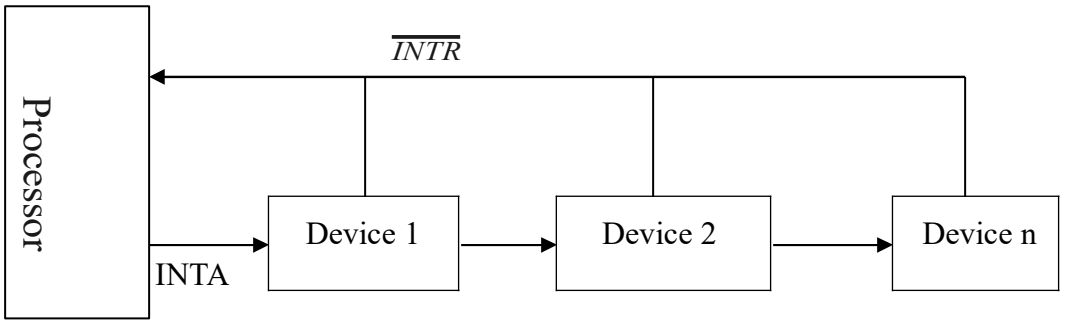
fig 2

Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

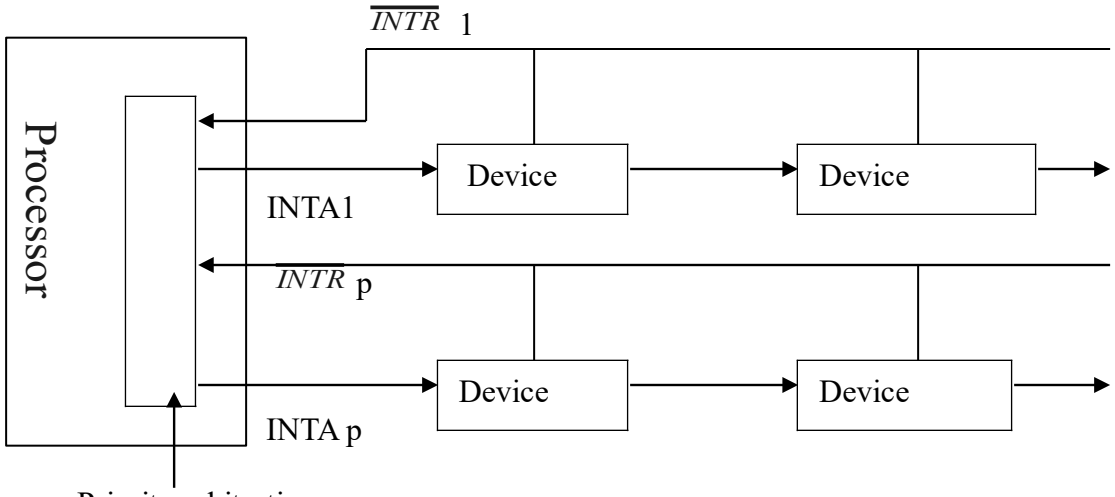
Simultaneous Requests: -

Consider the problem of simultaneous arrivals of interrupt requests from two or more devices. The processor simply accepts the requests having the highest priority.

Polling the status registers of the I/O devices is the simplest such mechanism. In this case, priority is determined by the order in which the devices are polled. When vectored interrupts are used, we must ensure that only one device is selected to send its interrupt vector code. A widely used scheme is to connect the devices to form a **daisy chain,** as shown in figure 3a. The interrupt-request line $\overline{INTR}$ is common to all devices. The interrupt-acknowledge line, INTA, is connected in a daisy-chain fashion, such that the INTA signal propagates serially through the devices.



(3.a) Daisy chain



(3.b) Arrangement of priority groups

When several devices raise an interrupt request and the $\overline{INTR}$ line is activated, the processor responds by setting the INTA line to 1. This signal is received by device 1. Device 1 passes the signal on to device 2 only if it does not require any service. If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines. **Therefore, in the daisy-chain arrangement, the device that is electrically closest to the processor has the highest priority.** The second device along the chain has second highest priority, and so on.

The scheme in figure 3.a requires considerably fewer wires than the individual connections. The main advantage of the scheme in figure 2 is that it allows the processor to accept interrupt requests from some devices but not from others, depending upon their priorities. The two schemes may be combined to produce the more general structure in figure 3b. Devices are organized in groups, and each group is connected at a different priority level. Within a group, devices are connected in a daisy chain. This organization is used in many computer systems.

Exceptions:-

An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin. However, the interrupt mechanism is used in several other situations.

The term exception is often used to refer to any event that causes an interruption. Hence, I/O interrupts are one example of an exception.

Recovery from Errors:-

Computers use a variety of techniques to ensure that all hardware components are operating properly. For example, many computers include an error-checking code in the main memory, which allows detection of errors in the stored data. If errors occur, the control hardware detects it and informs the processor by raising an interrupt.

The processor may also interrupt a program if it detects an error or an unusual condition while executing the instructions of this program. For example, the OP-code field of an instruction may not correspond to any legal instruction, or an arithmetic instruction may attempt a division by zero.

Debugging:-

Another important type of exception is used as an aid in debugging programs. System software usually includes a program called a debugger, which helps the programmer find errors in a program. The debugger uses exceptions to provide two important facilities called **trace and breakpoints.**

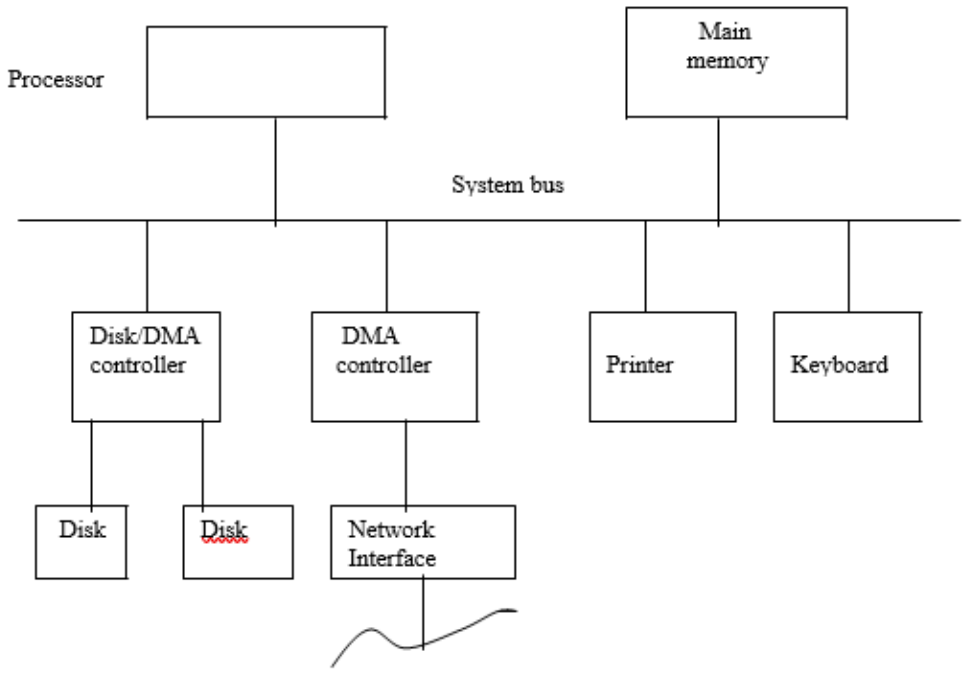
When a processor is operating in the **trace mode**, an exception occurs after execution of every instruction, using the debugging program as the exception-service routine. The debugging program enables the user to examine the contents of registers, memory locations, and so on. On return from the debugging program, the next instruction in the program being debugged is executed, then the debugging program is activated again. The trace exception is disabled during the execution of the debugging program.

Breakpoint provides a similar facility, except that the program being debugged is interrupted only at specific points selected by the user. An instruction called Trap or Software-interrupt is usually provided for this purpose. Execution of this instruction results in the same actions as when a hardware interrupt request is received.

Privilege Exception:

To protect the operating system of a computer from being corrupted by user programs, certain instructions can be executed only while the processor is in supervisor mode. These are called privileged instructions.

DIRECT MEMORY ACCESS:



Use of DMA controllers in a computer system

A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called direct memory access, or DMA.

DMA transfers are performed by a control circuit that is part of the I/O device interface called DMA controller. The DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory.

For each word transferred, it provides the memory address and all the bus signals that control data transfer. Since it must transfer blocks of data, the DMA controller must increment the memory address for successive words and keep track of the number of transfers.

Processor support

- Although a DMA controller can transfer data without intervention by the processor, its operation must be under the control of a program executed by the processor.
- To initiate the transfer, processor sends the starting address, the number of words in the block, and the direction of the transfer.
- On receiving this information, the DMA controller proceeds to perform the requested operation. When the entire block has been transferred, the controller informs the processor by raising an

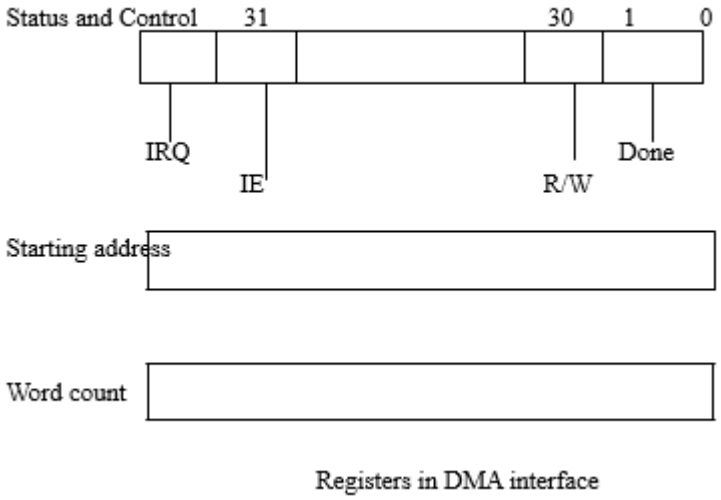
interrupt signal.

- While a DMA transfer is taking place, the program that requested the transfer cannot continue, and the processor can be used to execute another program. After the DMA transfer is completed, the processor can return to the program that requested the transfer.

OS support:

- I/O operations are always performed by the operating system of the computer in response to a request from an application program.
- The OS is also responsible for suspending the execution of one program and starting another.
- Thus, for an I/O operation involving DMA, the OS puts the program that requested the transfer in the Blocked state, initiates the DMA operation, and starts the execution of another program.
- When the transfer is completed, the DMA controller informs the processor by sending an interrupt request. In response, the OS puts the suspended program in the Runnable state so that it can be selected by the scheduler to continue execution.

Registers used in DMA controller



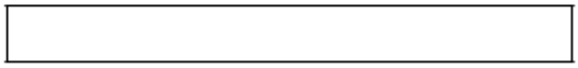
Registers in DMA interface

Figure shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations.

- Two registers are used for storing the Starting address and the word count. The third register contains status and control flags.
- The R/W bit determines the direction of the transfer. When this bit is set to 1 by a program instruction, the controller performs a read operation, that is, it transfers data from the memory to the I/O device. Otherwise, it performs a write operation.
- When the controller has completed transferring a block of data and is ready to receive another command, it sets the Done flag to 1.
- Bit 30 is the Interrupt-enable flag, IE, When this flag is set to 1, it causes the controller to raise

Starting address

Word count



Registers in DMA interface

of data.
requested an interrupt.

An example of a computer system is given in above figure, showing how DMA controllers may be used. A DMA controller connects a high-speed network to the computer bus. The disk controller, which controls two disks, also has DMA capability and provides two DMA channels. It can perform two independent DMA operations, as if each disk had its own DMA controller. The registers needed to store the memory address, the word count, and so on are duplicated, so that one set can be used with each device.

To start a DMA transfer of a block of data from the main memory to one of the disks, a program writes the address and word count information into the registers of the corresponding channel of the disk controller. It also provides the disk controller with information to identify the data for future retrieval. The DMA controller proceeds independently to implement the specified operation. When the DMA transfer is completed. This fact is recorded in the status and control register of the DMA channel by setting the Done bit. At the same time, if the IE bit is set, the controller sends an interrupt request to the processor and sets the IRQ bit. The status register can also be used to record other information, such as whether the transfer took place correctly or errors occurred.

Memory accesses by the processor and the DMA controller are interwoven. Requests by DMA devices for using the bus are always given higher priority than processor requests. Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, or a graphics display device. Since the processor originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the processor. **Hence, the interweaving technique is usually called cycle stealing.** Alternatively, the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. **This is known as block or burst mode.**

Bus Arbitration:

The device that can initiate data transfers on the bus at any given time is called the bus master. When the current master relinquishes control of the bus, another device can acquire this status. Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it.
There are two approaches to bus arbitration: **centralized and distributed.**
In centralized arbitration, a single bus arbiter performs the required arbitration. In distributed arbitration, all devices participate in the selection of the next bus master.

Centralized Arbitration:

- The bus arbiter may be the processor, or a separate unit connected to the bus.
- In this case, the processor is normally the bus master unless it grants bus mastership to one of the DMA controllers.
- A DMA controller indicates that it needs to become the bus master by activating the Bus-Request line, $\overline{BR}$.

- The signal on the Bus-Request line is the logical OR of the bus requests from all the devices connected to it.

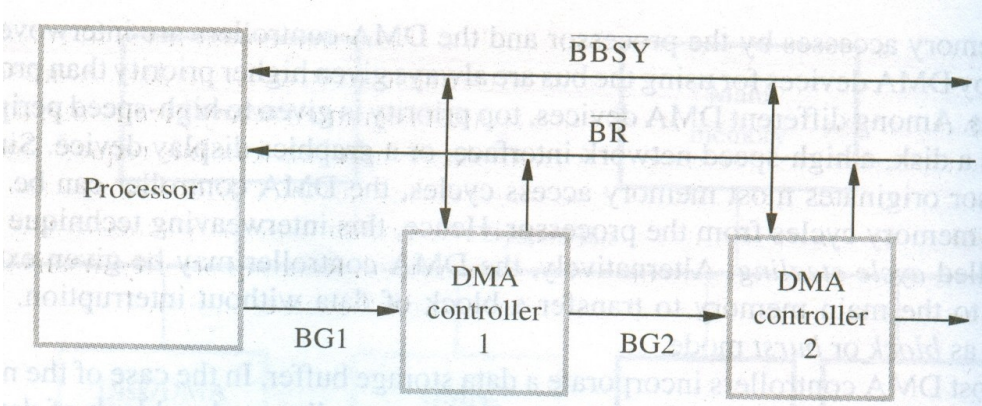


Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

- When Bus-Request is activated, the processor activates the Bus-Grant signal, BG1, indicating to the DMA controllers that they may use the bus when it becomes free.
- This signal is connected to all DMA controllers using a daisy-chain arrangement.
- Thus, if DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to other devices. Otherwise, it passes the grant downstream by asserting BG2.
- The current bus master indicates to all device that it is using the bus by activating another open-controller line called Bus-Busy, *BBSY* .
- Hence, after receiving the Bus-Grant signal, a DMA controller waits for Bus-Busy to become inactive, then assumes mastership of the bus. Currently, it activates Bus-Busy to prevent other devices from using the bus at the same time.

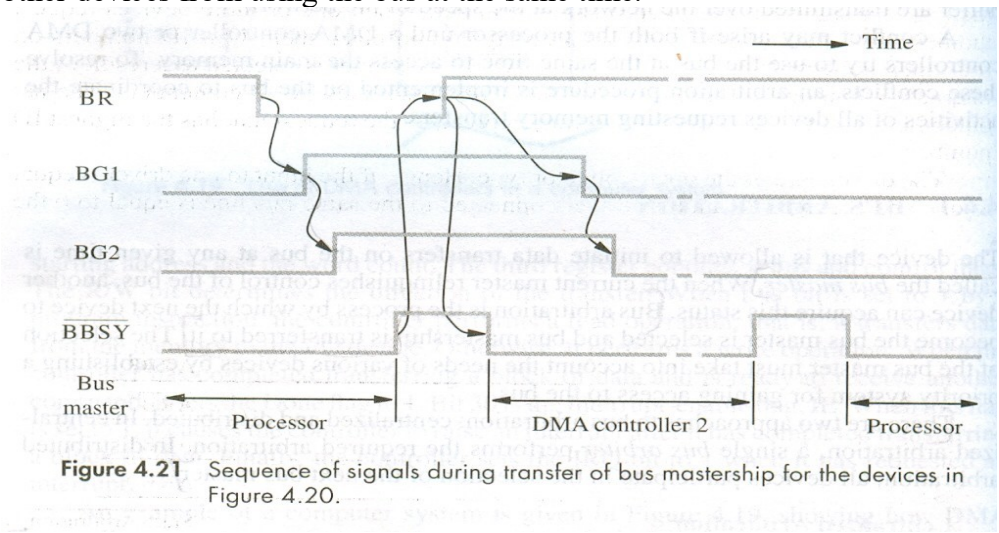


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

- DMA controller 2 requests and acquires bus mastership and later releases the bus. During this

tenure as the bus master, it may perform one or more data transfer operations, depending on whether it is operating in the cycle stealing or block mode.

- After it releases the bus, the processor resumes bus mastership.
- The arbiter circuit ensures that only one request is granted at any given time according to a predefined priority scheme. For example, if there are four bus request lines BR1 through BR4, a fixed priority scheme may be used in which BR1 is given top priority and BR4 is given low priority. Alternatively, a rotating priority scheme may be used to give all devices an equal chance of being serviced.
- Rotating priority means that after a request on line BR1 is granted the priority order becomes 2, 3, 4, and 1.

Distributed Arbitration: -

- Each device on the bus is assigned a four-bit identification number.
- When one or more device requests the bus, they assert the start arbitration signal and place their 4 bit id numbers on 4 open- collector lines, ARB0 through ARB3. a winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders.
- The net outcome is that the code on the four lines represents the request that has the highest id number.

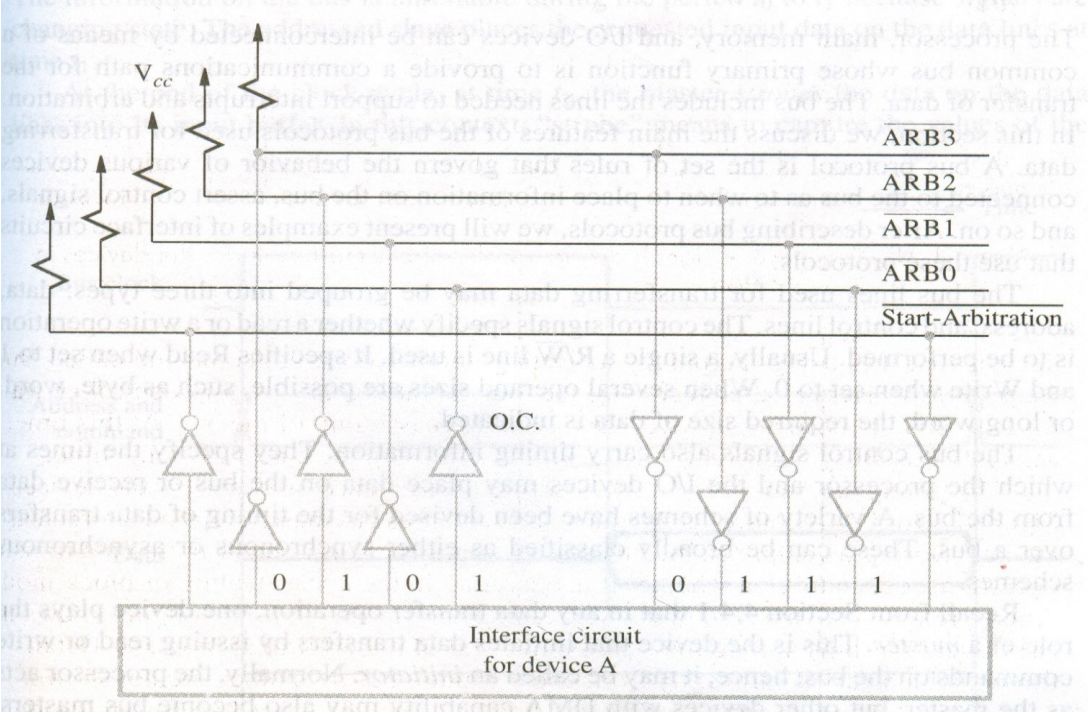


Figure 4.22 A distributed arbitration scheme.

Assume that two devices A and B having ID numbers 5 and 6 respectively are requesting the use of the bus.

Device A transmit the pattern 0110 and device B transmit the pattern 0110.The code seen by both devices is 0110.each device compares the pattern on the arbitration lines to its own ID, starting from the most significant bit. If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower-order bits. It does so by placing a 0 at the input of these drivers. In the case of our example, devices A detects a difference on line ARB1.Hence it disables its drivers on lines ARB1 and ARB0.This causes the pattern on the arbitration lines to change to 0110, which means that B has won the connection. Note that since the code on the priority lines is 0111 for a short period, device B may temporarily disable its driver on line ARB0.However it will enable this driver again once it sees a 0 on line ARB1 resulting from the action by device A.

Decentralized arbitration has the advantage of offering higher reliability, because operation of the bus is not dependent on any single device.

Buses:-

The processor, main memory, and I/O devices can be interconnected by means of a common bus whose primary function is to provide a communication path for the transfer of data. The bus includes the lines needed to support interrupts and arbitration.

A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on.

The bus consists of three lines: address, data & control. The control signal specifies R/W operation & the required size of data (byte, word or long word). The bus control signals also carry timing information. This specifies the times at which the processor & the I/O devices may place/receive data on the bus.

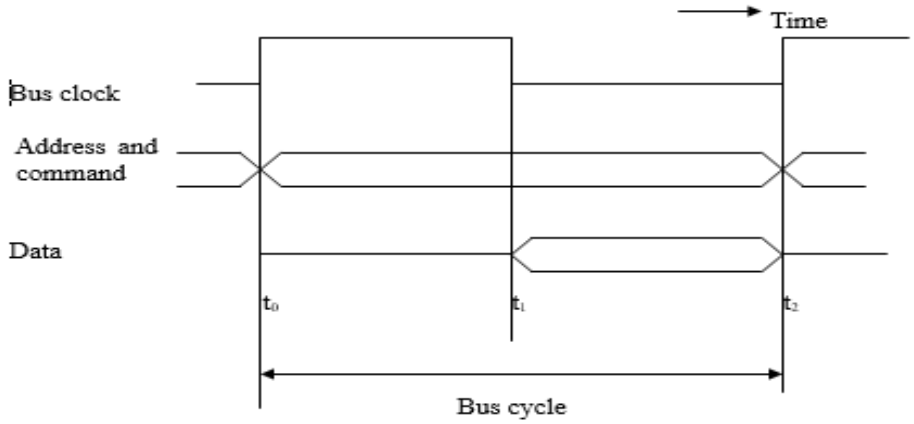
The device that initiates data transfer by issuing read or write commands on the bus is called as bus master or the initiator. Normally, the processor/DMA Controller acts as the master. The device addressed by the master is referred as a slave or target.

The schemes for the timing of data transfer over the bus are classified as synchronous or asynchronous schemes.

Synchronous Bus:-

In a synchronous bus, all devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals. The equal time intervals constitute a bus cycle during which one data transfer can take place.

Timing of an input transfer on a synchronous bus.



- At t0, the master places the device address and command on the respective lines. (command is input operation and specify length of the operand to be read if necessary). Information travels over the bus at a speed determined by its physical and electrical characteristics.

- The clock pulse width t1-t0, must be long enough than the maximum propagation delay between two devices + address and control decode time. So that the addressed device (slave) can respond at t1. The slave places the requested input data on the data lines at t1.

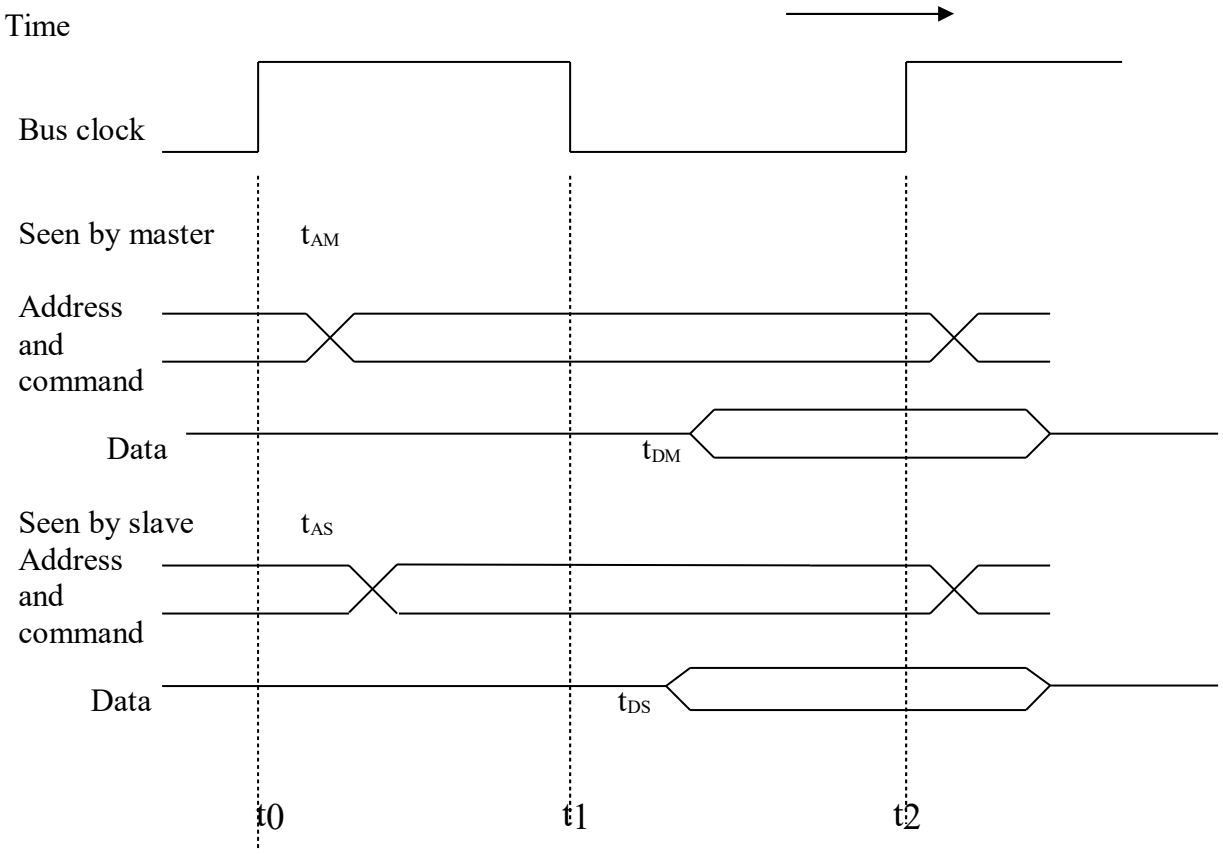
- At the end of bus cycle at t2, the master strobes/captures the data on the data lines and store into its input buffer. Hence, the period t2-t1 must be greater than the maximum propagation time + the setup time of the input buffer register of the master.

- A similar procedure is followed for an output operation. The master places the output data on the data lines when it transmits the address and command information. At time t2, the addressed device strobes the data lines and loads the data into its data buffer.

The exact times at which signals appear on the bus are somewhat different from those shown. Because of propagation delays on the bus wires and in the circuits of the devices as shown.

The master sends the address and command signals on the rising edge at the beginning of clock period 1 (t_0). However, these signals do not actually appear on the bus until f_{AM}, largely due to the delay in the bus driver circuit. A while later, at t_{AS}, the signals reach the slave. The slave decodes the address and at t1 sends the requested data. Here again, the data signals do not appear on the bus until t_{DS}. They travel toward the master and arrive at t_{DM}. At t2, the master loads the data into its input buffer. Hence the period t_2-t_{DM} is the setup time for the master's input buffer. The data must continue to be valid after t_2 for a period equal to the hold time of that buffer.

A detailed timing diagram for the input transfer

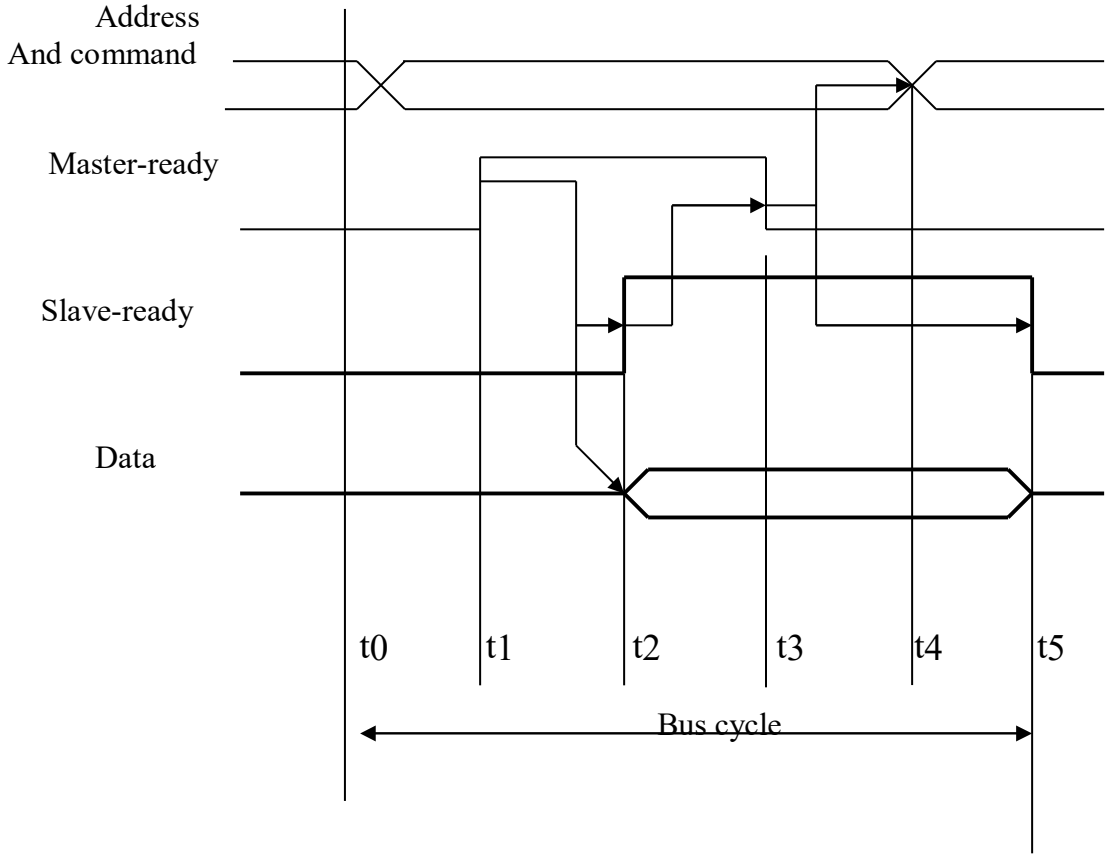


ASYNCHRONOUS BUS: -

An alternative scheme for controlling data transfers on the bus is based on the use of a handshake between the master and the salve. The common clock is replaced by two timing control lines, Master-ready and Slave-ready. The first is asserted by the master to indicate that it is ready for a transaction, and the second is a response from the slave.

The master places the address and command information on the bus. Then it indicates to all devices that it has done so by activating the Master-ready line. This causes all devices on the bus to decode the address. The selected slave performs the required operation and informs the processor it has done so by activating the Slave-ready line. The master waits for Slave-ready to become asserted before it removes its signals from the bus. In the case of a read operation, it also strobes the data into its input buffer.

Handshake control of data transfer during an input operation.



An example of the timing of an input data transfer using the handshake scheme is given in figure, which depicts the following sequence of events.

t0 – the master places the address and command information on the bus, and all devices on the bus begin to decode this information.

t1- the master sets the Master-ready line to 1 to inform the I/O devices that the address and command information is ready.

- The delay t_1-t_0 is intended to allow for any ***skew*** that may occur on the bus.
- ***Skew*** occurs when two signals simultaneously transmitted from one source, arrive at the destination at different times. This happens because different lines of the bus may have different propagation speeds. Thus, to guarantee that the Master-ready signal does not arrive at any device ahead of the address and the command information, the delay t1-t0 should be larger than the maximum possible bus skew(note that, in the synchronous case, bus skew is accounted for as a part of the maximum propagation delay.)
- When the address information arrives at any device, it is decoded by the interface circuitry.

Enough time should be allowed for the interface circuitry to decode the address. The delay needed can be included in the period t1-t0.

t2—the selected slave, having decoded the address and command information, performs the required input operation by placing the data from its data register on the data lines. At the same time, it sets the Slave-ready signal to 1. If extra delays are introduced by the interface circuitry before it places the data on the bus, the slave must delay the Slave-ready signal accordingly. The period t2-t1 depends on the distance between the master and the slave and on the delays introduced by the slave's circuitry. It is this variability that gives the bus its asynchronous nature.

t3—the slave-ready signal arrives at the master, indicating that the input data are available on the bus. However, since it was assumed that the device interface transmits the slave-ready signal at the same time that it places the data on the bus, the master should allow for the bus skew. It must also allow for the setup time needed by its input buffer. After a delay equivalent to the maximum bus skew and the minimum setup time, the master strobes the data into its input buffer. At the same time, it drops the master-ready signal, indicating that it has received the data.

t4—the master removes the address and the command information from the bus. The delay between t3 and t4 is again intended to allow for the bus skew. Erroneous addressing may take place if the address, as seen by some device on the bus, starts to change while the master-ready signal is still equal to 1.

t5—when the device interface receives the 1 to 0 transition of the master-ready signal, it removes the data and the slave-ready signal from the bus.
This completes the input transfer.