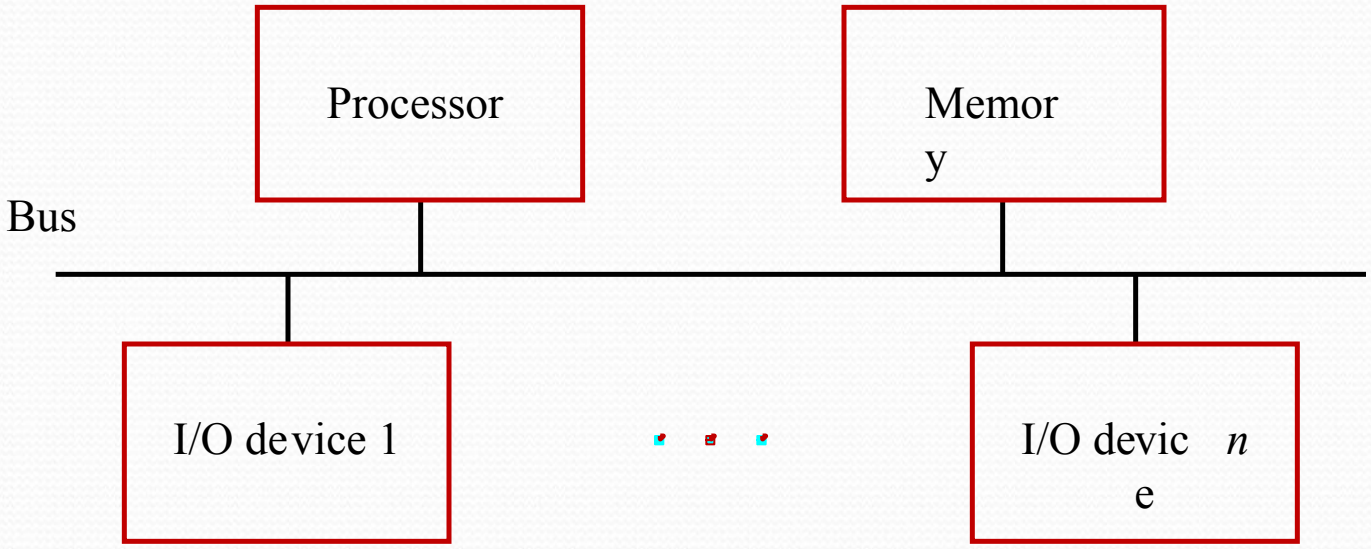
INPUT/OUTPUT ORGANIZATION

Accessing I/O Devices

I/O

- The data on which the instructions operate are not necessarily already stored in memory.
- Data need to be transferred between processor and outside world (disk, keyboard, etc.)
- I/O operations are essential, the way they are performed can have a significant effect on the performance of the computer.

Accessing I/O devices

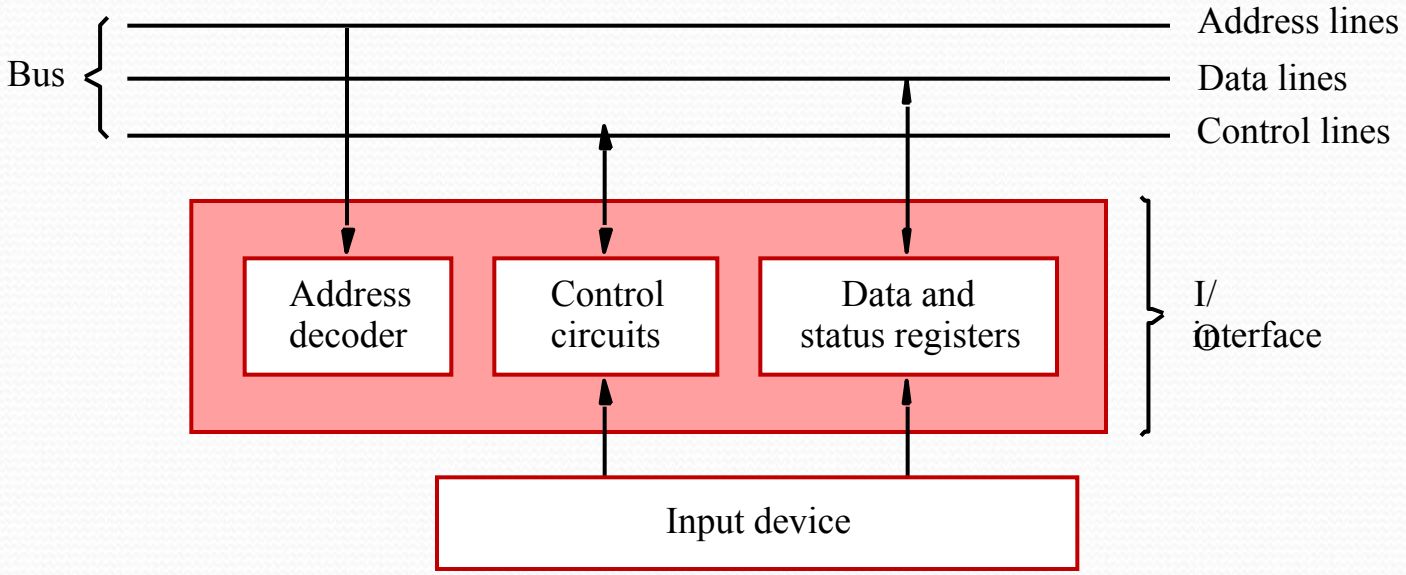


- *Multiple I/O devices may be connected to cpu and the memory via a bus.*
- *Bus consists of 3 sets of lines to carry addr, data and control signals.*
- *Each I/O device is assigned an unique address.*
- *To access an I/O device, the processor places the addr on addr lines.*
- *The device recognizes the address, and responds to the control signals.*

Accessing I/O devices (contd..)

- I/O devices and the memory may share the same address space:
  - Memory-mapped I/O.
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
  - Simpler software.
- I/O devices and the memory may have different address spaces:
  - Special instructions to transfer data to and from I/O devices.
  - I/O devices may have to deal with fewer address lines.
  - I/O address lines need not be physically separate from memory address lines.
  - In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

Accessing I/O devices (contd..)



- *I/O device connected to bus using I/O interface circuit which has:*
  - *Address decoder, control circuit, and data and status registers.*
- *Addr decoder decodes addr placed on the address lines thus enabling the device to recognize its addr.*
- *Data register holds the data being transferred to or from the cpu.*
- *Status register holds info necessary for the operation of the I/O device.*
- *Data and status registers are connected to the data lines, and have unique addresses.*
- *I/O interface circuit coordinates I/O transfers.*

Accessing I/O devices (contd..)

- Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor. This creates the need for mechanisms to synchronize data transfers between them.
- Program-controlled I/O:
  - Processor repeatedly monitors a status flag to achieve the necessary synchronization.
  - Processor polls the I/O device.
- Two other mechanisms used for synchronizing data transfers between the processor and memory:
  - Interrupts.
  - Direct Memory Access.

Program-Controlled I/O Example

- Read in character input from a keyboard and produce character output on a display screen.
  - Rate of data transfer (keyboard, display, processor)
  - Difference in speed between processor and I/O device creates the need for mechanisms to synchronize the transfer of data.
  - A solution: on output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character. Input is sent from the keyboard in a similar way.

Program-Controlled I/O Example

- Registers
- Flags
- Device interface

Program-Controlled I/O Example

- Machine instructions that can check the state of the status flags and transfer data:

```
READWAIT  Branch to READWAIT if SIN = 0
          Input from DATAIN to R1

WRITEWAIT Branch to WRITEWAIT if SOUT = 0
          Output from R1 to DATAOUT
```

Program-Controlled I/O Ex.

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

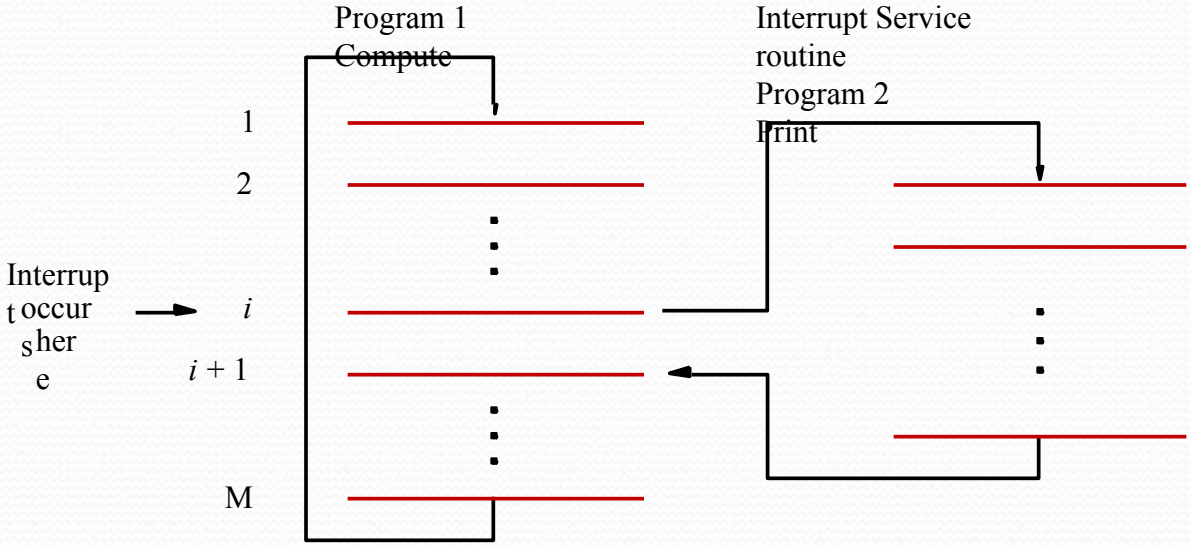
instructions that can check the state of the status flags and transfer data:

Interrupts

Interrupts

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.
- Processor can perform other useful tasks while it is waiting for the device to be ready.

Interrupts (contd..)



- *Processor is executing the instruction located at addr i when an interrupt occurs.*
- *Routine executed in response to inter request is called the interrupt-service routine.*
- *When an inter occurs, control must be transferred to the interrupt service routine.*
- *But before transferring control, the current contents of the PC (i+1), must be saved in a known location.*
- *This will enable the return-from-interrupt instruction to resume execution at i+1.*
- *Return address, or the contents of the PC are usually stored on the processor stack.*

Interrupts (contd..)

- Treatment of an interrupt-service routine is very similar to that of a subroutine.
- However there are significant differences:
  - A subroutine performs a task that is required by the calling program.
  - Interrupt-service routine may not have anything in common with the program it interrupts.
  - Interrupt-service routine and the program that it interrupts may belong to different users.
  - As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
  - This will enable the interrupted program to resume execution upon return from interrupt service routine.

Interrupts (contd..)

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.
- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - Increases 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.
- In order to reduce the interrupt latency, most processors save only the minimal amount of information:
  - This minimal amount of information includes Program Counter and processor status registers.
- Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

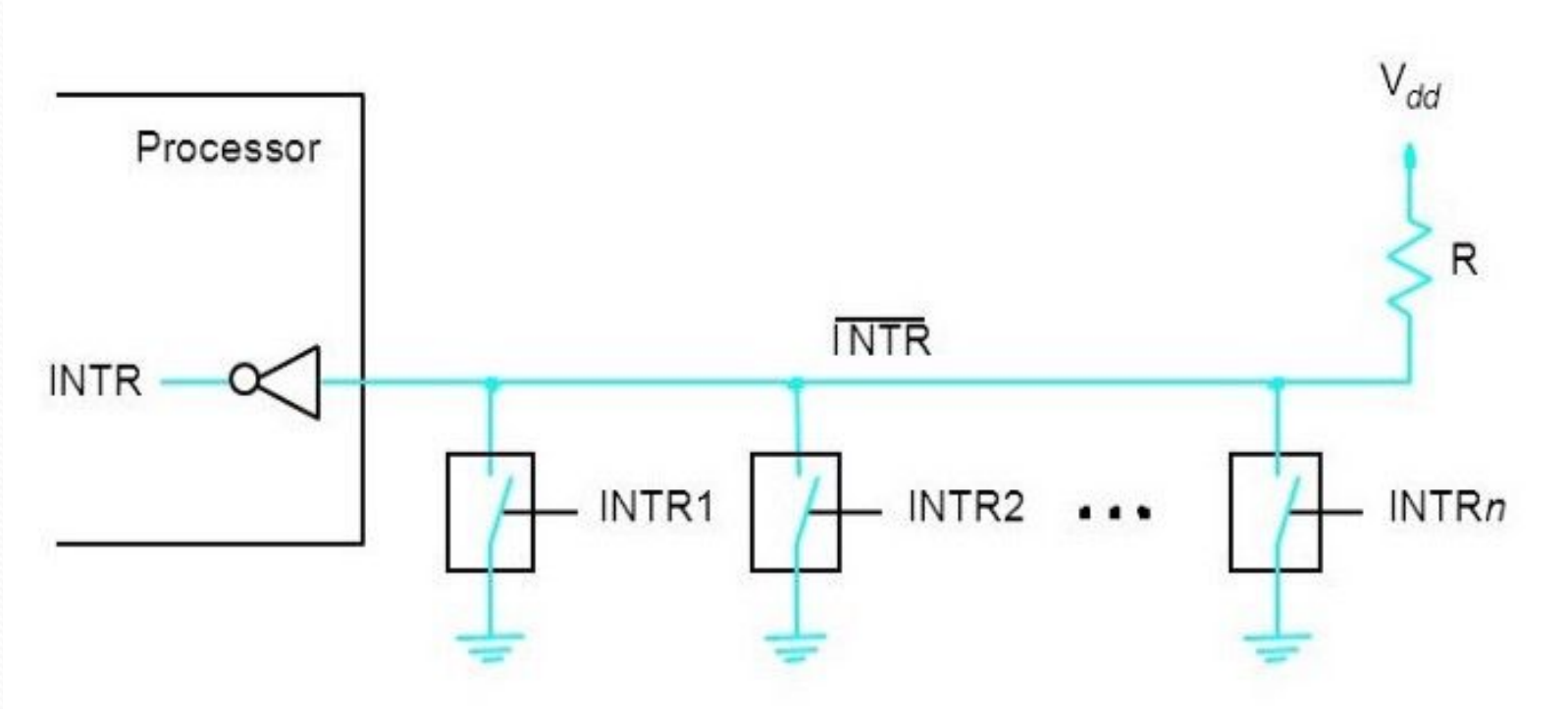
Interrupts (contd..)

- When a processor receives an interrupt-request, it must branch to the interrupt service routine.
- It must also inform the device that it has recognized the interrupt request.
- This can be accomplished in two ways:
  - Some processors have an explicit interrupt-acknowledge control signal for this purpose.
  - In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

Interrupts (contd..)

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be Interrupt-disable.
  - Last instruction of an interrupt service routine can be Interrupt-enable.

Interrupt Hardware



INTR=INTR1+INTR2+…+INTRn

Interrupt Hardware

Common interrupt line for all N I/O devices:

When no i/o device interrupts all switches are open and entire voltage from Vdd is flown through single INTR line and reaches the processor. i.e. processor gets 1V.

When the interrupt is issued by the i/o devices then switch associated with i/o device is closed.

Entire current now passes via the switches which means h/w line reaching the processes i.e INTR line gets 0V, indicating interrupt has occurred and processor needs to find which i/o device has triggered the interrupt.

The value of INTR is a logical OR of the requests from individual devices.The resistor R is called as a pull up resistor because it pulls the line voltage to high voltage state when all switches are open(no interrupt state).

Interrupts (contd..)

- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
  - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

Interrupts (contd..)

- Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.
- When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?
- This information is available in the status register of the device requesting an interrupt:
  - The status register of each device has an *IRQ* bit which it sets to 1 when it requests an interrupt.
- Interrupt service routine can poll the I/O devices connected to the bus. The first device with *IRQ* equal to 1 is the one that is serviced.
- Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

Interrupts (contd..)

- Vectored Interrupts : The device requesting an interrupt may identify itself directly to the processor.
  - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
  - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
  - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.
- Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine.

Interrupts (contd..)

- I/O devices are organized in a priority structure:
  - An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.
- A priority level is assigned to a processor that can be changed under program control.
  - Priority level of a processor is the priority of the program that is currently being executed.
  - When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.
  - If the device sending an interrupt request has a higher priority than the processor, the processor accepts the interrupt request.

Interrupts (contd..)

- Previously, before the processor started executing the interrupt service routine for a device, it disabled the interrupts from the device.
- In general, same arrangement is used when multiple devices can send interrupt requests to the processor.
  - During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device.
  - Since the interrupt service routines are usually short, the delay that this causes is generally acceptable.
- However, for certain devices this delay may not be acceptable.
  - Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

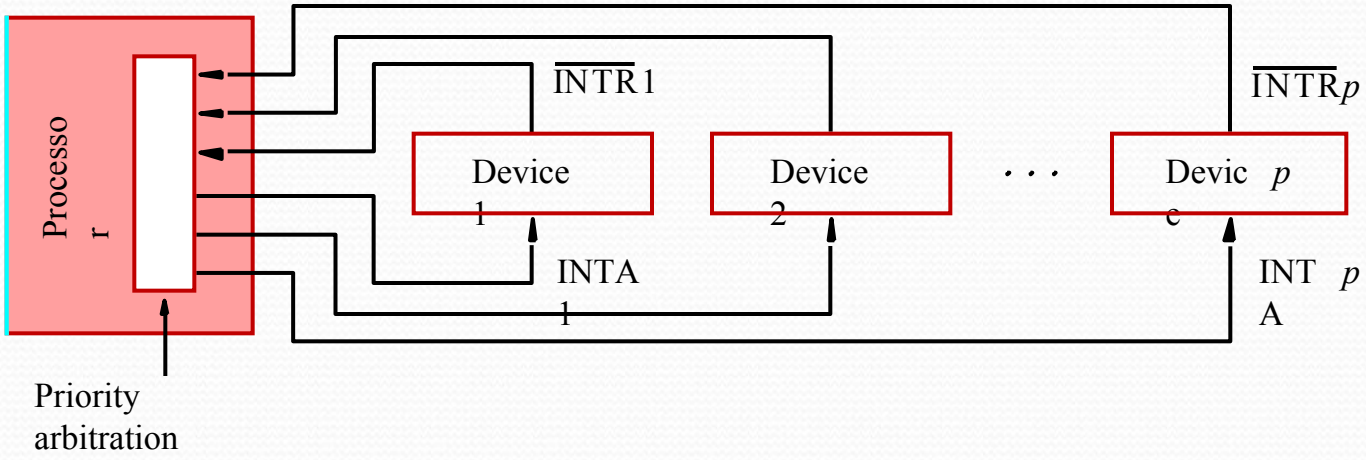
Interrupts (contd..)

- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
  - Each device has its own interrupt request and interrupt acknowledge line.
  - A different priority level is assigned to the interrupt request line of each device.
- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

Interrupts (contd..)

- Processor's priority is encoded in a few bits of the processor status register.
  - Priority can be changed by instructions that write into the processor status register.
  - Usually, these are privileged instructions, or instructions that can be executed only in the supervisor mode.
  - Privileged instructions cannot be executed in the user mode.
  - Prevents a user program from accidentally or intentionally changing the priority of the processor.
- If there is an attempt to execute a privileged instruction in the user mode, it causes a special type of interrupt called as privilege exception.

Interrupts (contd..)



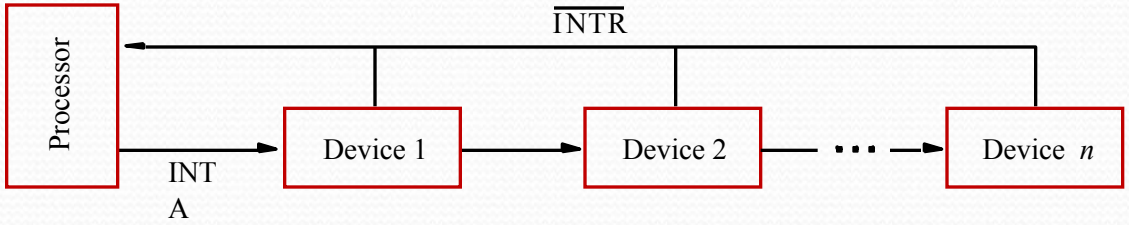
- *Each device has separate interrupt-request and interrupt-acknowledge line.*
- *Each interrupt-request line is assigned a different priority level.*
- *Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.*
- *If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.*

Interrupts (contd..)

Polling scheme:

- *If the CPU uses polling mechanism to poll the status registers of I/O devices to determine which device is requesting an interrupt.*
- *In this case the priority is determined by the order in which the devices are polled.*
- *First dev with status bit 1 is the dev whose intr request is accepted.*

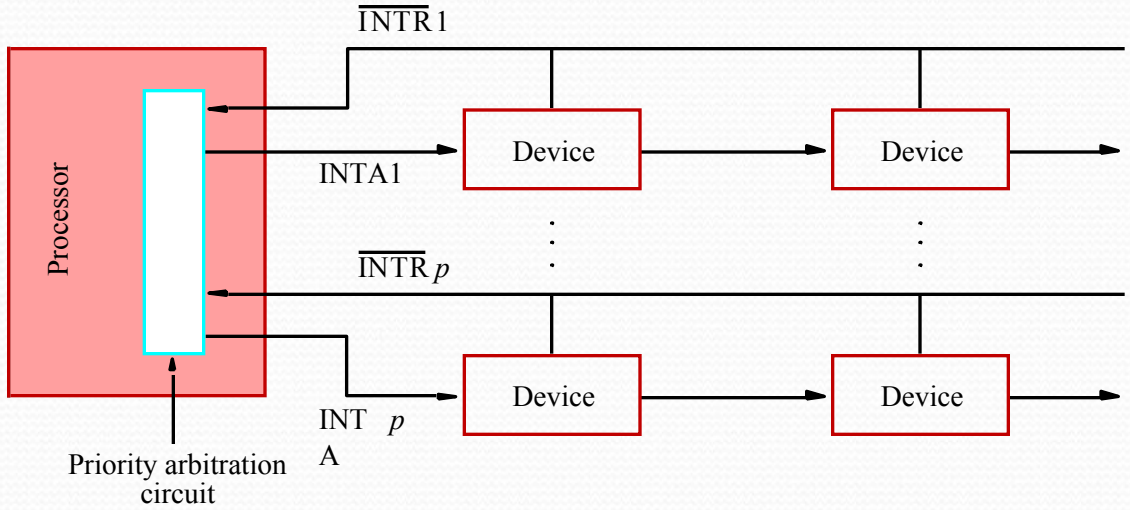
Daisy chain scheme:



- *Devices are connected to form a daisy chain.*
- *Devices share inter-req line, and inter-acknowledge line is connected to form a daisy chain.*
- *When devices raise an inter req, the inter-req line is activated.*
- *The processor in response activates inter-acknowledge.*
- *Received by device 1, if device 1 does not need service, it passes the signal to device 2.*
- *Device that is electrically closest to the processor has the highest priority.*

Interrupts (contd..)

- *When I/O devices were organized into a priority structure, each device had its own interrupt-request and interrupt-acknowledge line.*
- *When I/O devices were organized in daisy chain fashion, devices shared inter-req line, and inter-acknowledge propagated through the devices.*
- *A combination of priority structure and daisy chain scheme can also used.*



- *Devices are organized into groups.*
- *Each group is assigned a different priority level.*
- *All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain.*

Interrupts (contd..)

- Only those devices that are being used in a program should be allowed to generate interrupt requests.
- To control which devices are allowed to generate interrupt requests, the interface circuit of each I/O device has an interrupt-enable bit.
  - If the interrupt-enable bit in the device interface is set to 1, then the device is allowed to generate an interrupt-request.
- Interrupt-enable bit in the device's interface circuit determines whether the device is allowed to generate an interrupt request.
- Interrupt-enable bit in the processor status register or the priority structure of the interrupts determines whether a given interrupt will be accepted.

Exceptions

- Interrupts caused by interrupt-requests sent by I/O devices.
- Interrupts could be used in many other situations where the execution of one program needs to be suspended and execution of another program needs to be started.
- In general, the term exception is used to refer to any event that causes an interruption.
  - Interrupt-requests from I/O devices is one type of an exception.
- Other types of exceptions are:
  - Recovery from errors
  - Debugging
  - Privilege exception

Exceptions (contd..)

- Many sources of errors in a processor. For example:
  - Error in the data stored.
  - Error during the execution of an instruction.
- When such errors are detected, exception processing is initiated.
  - Processor takes the same steps as in the case of I/O interrupt-request.
  - It suspends the execution of the current program, and starts executing an exception-service routine.
- Difference between handling I/O interrupt-request and handling exceptions due to errors:
  - In case of I/O interrupt-request, the processor usually completes the execution of an instruction in progress before branching to the interrupt-service routine.
  - In case of exception processing however, the execution of an instruction in progress usually cannot be completed.

Exceptions (contd..)

- Debugger uses exceptions to provide important features:
  - Trace,
  - Breakpoints.
- Trace mode:
  - Exception occurs after the execution of every instruction.
  - Debugging program is used as the exception-service routine.
- Breakpoints:
  - Exception occurs only at specific points selected by the user.
  - Debugging program is used as the exception-service routine.

Exceptions (contd..)

- Certain instructions can be executed only when the processor is in the supervisor mode. These are called privileged instructions.
- If an attempt is made to execute a privileged instruction in the user mode, a privilege exception occurs.
- Privilege exception causes:
  - Processor to switch to the supervisor mode,
  - Execution of an appropriate exception-servicing routine.

Direct Memory Access

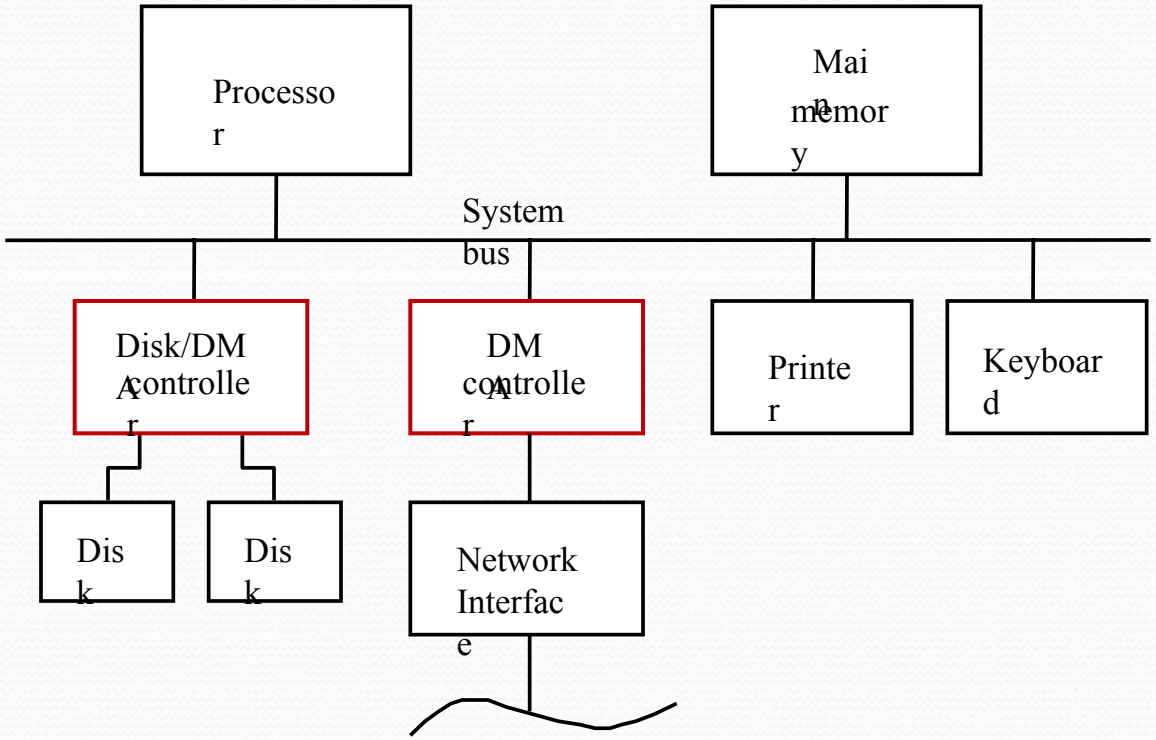
Direct Memory Access

- DMA: A special control unit may be provided to transfer a block of data directly between an I/O device and the main memory, without continuous intervention by the processor.
- Control unit which performs these transfers is a part of the I/O device's interface circuit. This control unit is called as a DMA controller.
- DMA controller performs functions that would be normally carried out by the processor:
  - For each word, it provides the memory address and all the control signals.
  - To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

Direct Memory Access (contd..)

- DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor.
  - However, the operation of the DMA controller must be under the control of a program executed by the processor. That is, the processor must initiate the DMA transfer.
- To initiate the DMA transfer, the processor informs the DMA controller of:
  - Starting address,
  - Number of words in the block.
  - Direction of transfer (I/O device to the memory, or memory to the I/O device).
- Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

Direct Memory Access



- *DMA controller connects a high-speed network to the computer bus.*
- *Disk controller, which controls two disks also has DMA capability. Provides two DMA channels.*
- *Can perform two independent DMA operations, as if each disk has its own DMA controller. Registers to store the memory address, word count and status and control information are duplicated.*

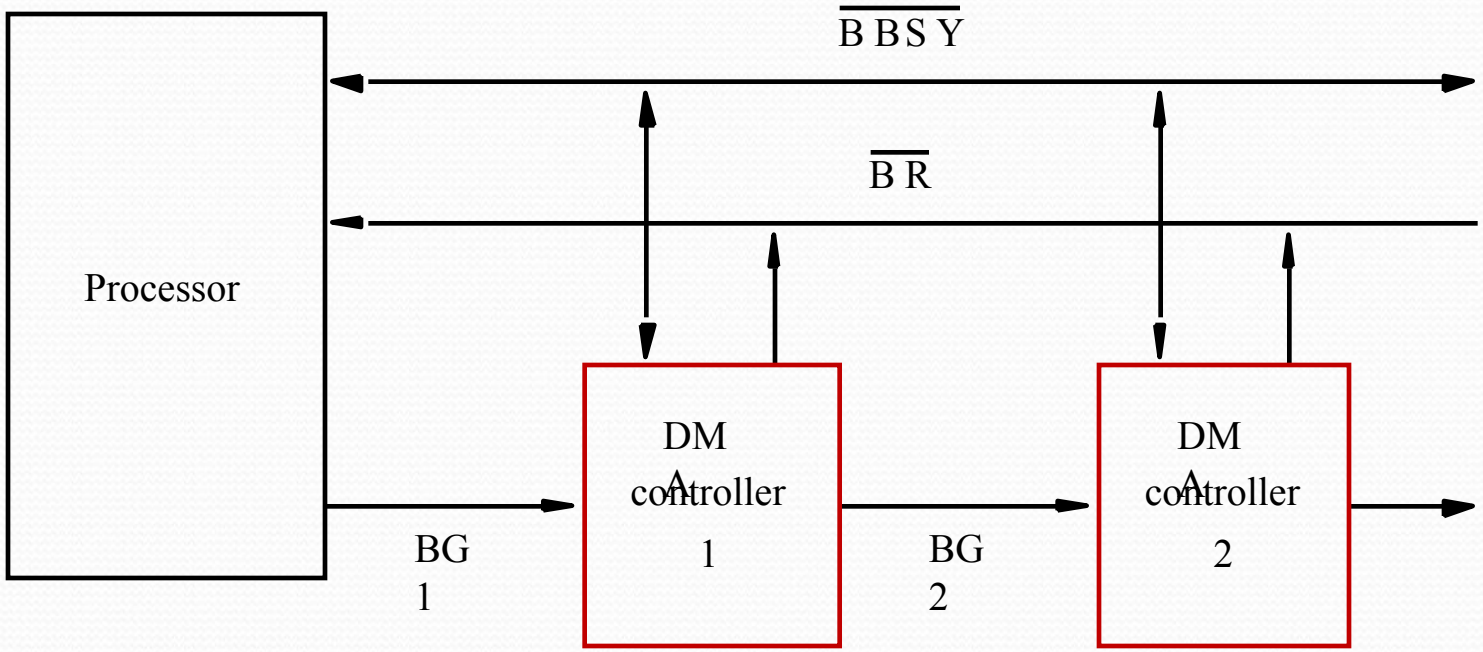
Direct Memory Access (contd..)

- Processor and DMA controllers have to use the bus in an interwoven fashion to access the memory.
  - DMA devices are given higher priority than the processor to access the bus.
  - Among different DMA devices, high priority is given to high-speed peripherals such as a disk or a graphics display device.
- Processor originates most memory access cycles on the bus.
  - DMA controller can be said to "steal" memory access cycles from the bus. This interweaving technique is called as "cycle stealing".
- An alternate approach is the provide a DMA controller an exclusive capability to initiate transfers on the bus, and hence exclusive access to the main memory. This is known as the block or burst mode.

Bus arbitration

- Processor and DMA controllers both need to initiate data transfers on the bus and access main memory.
- The device that is allowed to initiate transfers on the bus at any given time is called the bus master.
- When the current bus master relinquishes its status as the bus master, another device can acquire this status.
  - The process by which the next device to become the bus master is selected and bus mastership is transferred to it is called bus arbitration.
- Centralized arbitration:
  - A single bus arbiter performs the arbitration.
- Distributed arbitration:
  - All devices participate in the selection of the next bus master.

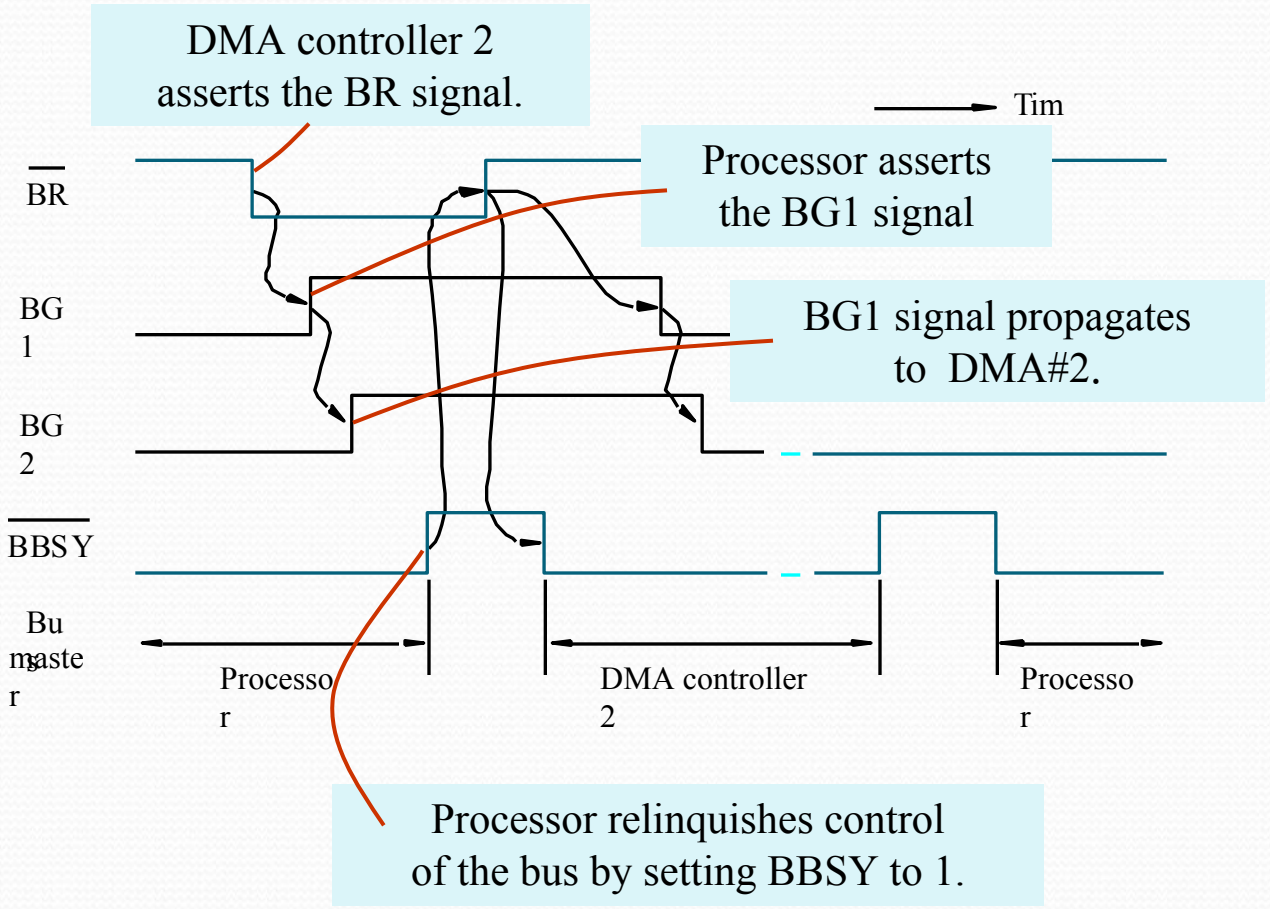
Centralized Bus Arbitration



Centralized Bus Arbitration(cont.,)

- *Bus arbiter may be the processor or a separate unit connected to the bus.*
- *Normally, the processor is the bus master, unless it grants bus membership to one of the DMA controllers.*
- *DMA controller requests the control of the bus by asserting the Bus Request (BR) line.*
- *In response, the processor activates the Bus-Grant1 (BG1) line, indicating that the controller may use the bus when it is free.*
- *BG1 signal is connected to all DMA controllers in a daisy chain fashion.*
- *BBSY signal is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller which asserted BR can acquire control of the bus.*

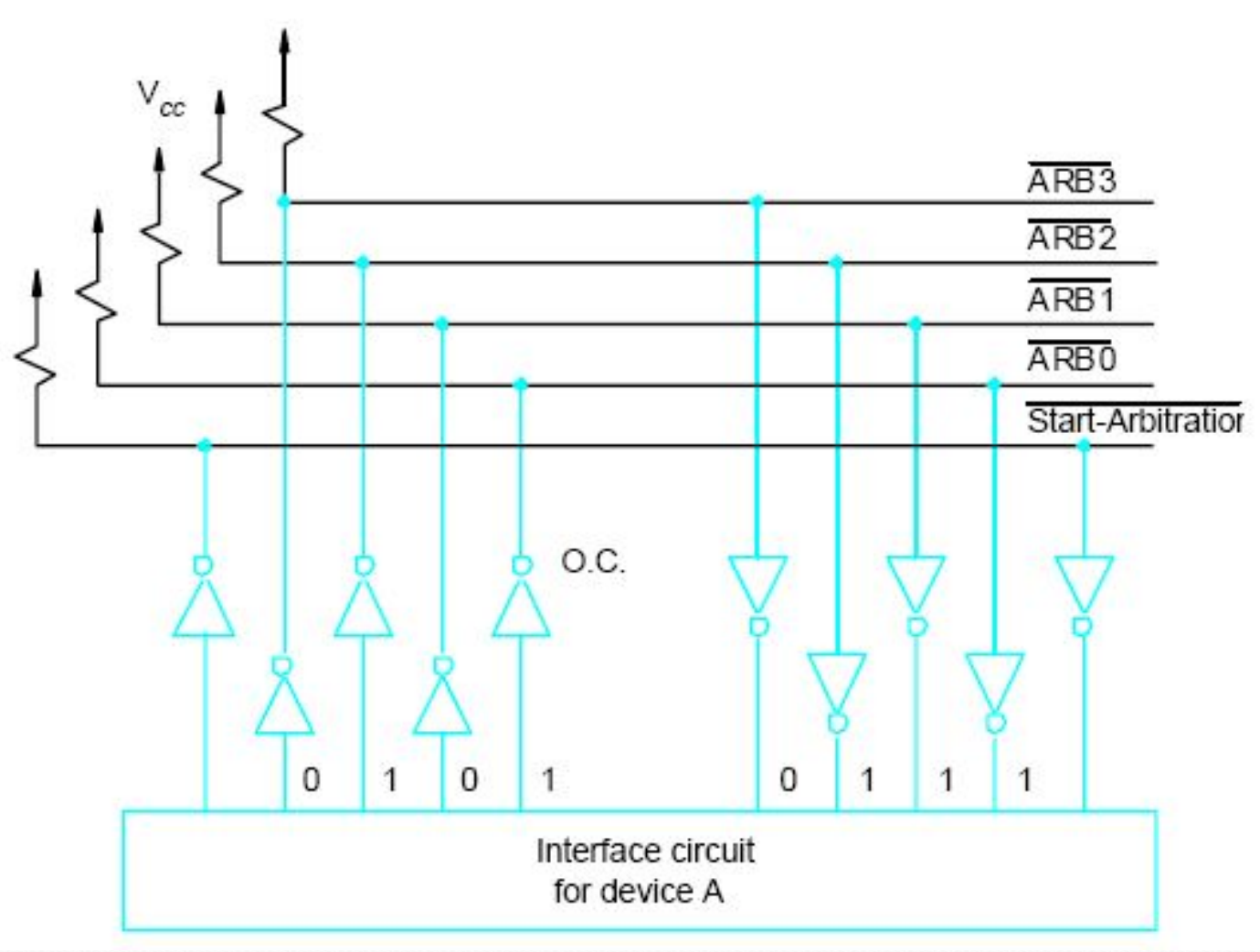
Centralized arbitration (contd..)



Distributed arbitration

- All devices waiting to use the bus share the responsibility of carrying out the arbitration process.
  - Arbitration process does not depend on a central arbiter and hence distributed arbitration has higher reliability.
- Each device is assigned a 4-bit ID number.
- All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start-Arbitration signal.
- To request the bus a device:
  - Asserts the Start-Arbitration signal.
  - Places its 4-bit ID number on the arbitration lines.
- The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.

Distributed arbitration



Distributed arbitration(Contd.,)

- *Arbitration process:*
  - *Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.*
  - *If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions.*
  - *The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.*

Distributed arbitration (contd..)

evice A has the ID 5 and wants to request the bus:

- Transmits the pattern 0101 on the arbitration lines.

evice B has the ID 6 and wants to request the bus:

- Transmits the pattern 0110 on the arbitration lines.

attern that appears on the arbitration lines is the logical OR of the patterns:

- Pattern 0111 appears on the arbitration lines.

tration process:

h device compares the pattern that appears on the arbitration lines to its own starting with MSB.

detects a difference, it transmits 0s on the arbitration lines for that and all lower positions.

ice A compares its ID 5 with a pattern 0101 to pattern 0111.

tects a difference at bit position 0, as a result, it transmits a pattern 0100 on the itration lines.

pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, ch is 0110.

pattern is the same as the device ID of B, and hence B has won the arbitration.

Buses

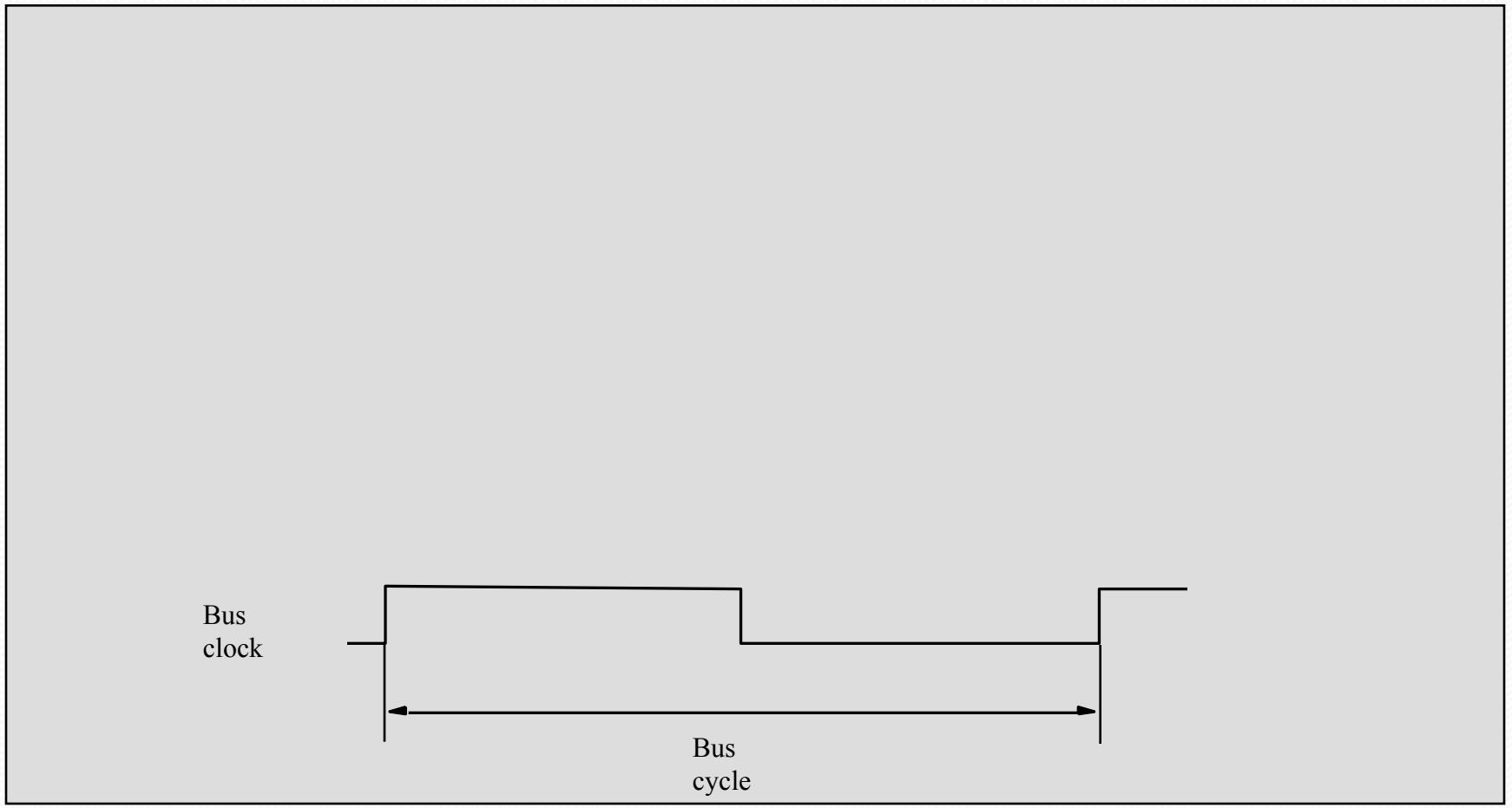
Buses

- Processor, main memory, and I/O devices are interconnected by means of a bus.
- Bus provides a communication path for the transfer of data.
  - Bus also includes lines 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, when to assert control signals, etc.

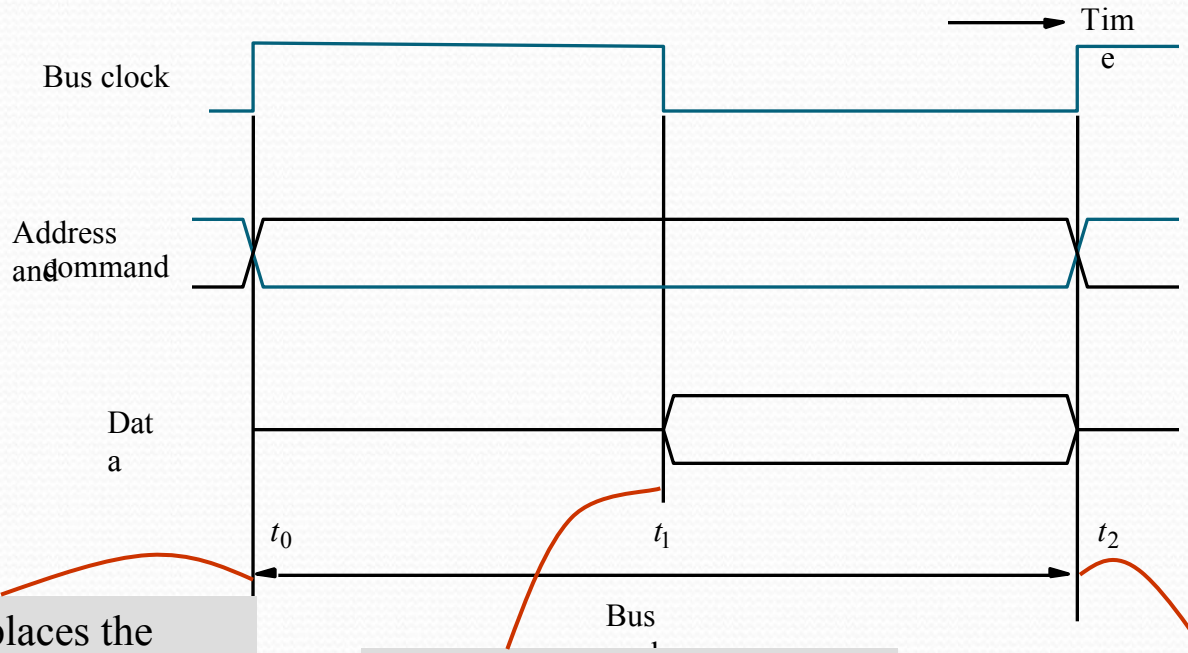
Buses (contd..)

- Bus lines may be grouped into three types:
  - Data
  - Address
  - Control
- Control signals specify:
  - Whether it is a read or a write operation.
  - Required size of the data, when several operand sizes (byte, word, long word) are possible.
  - Timing information to indicate when the processor and I/O devices may place data or receive data from the bus.
- Schemes for timing of data transfers over a bus can be classified into:
  - Synchronous,
  - Asynchronous.

Synchronous bus



Synchronous bus (contd..)



Master places the device address and command on the bus, and indicates that it is a Read operation.

Addressed slave places data on the data lines

Master "strobes" the data on the data lines into its input buffer, for a Read operation.

ase of a Write operation, the master places the data on the bus along with the dress and commands at time t_0.

slave strobes the data into its input buffer at time t_2.

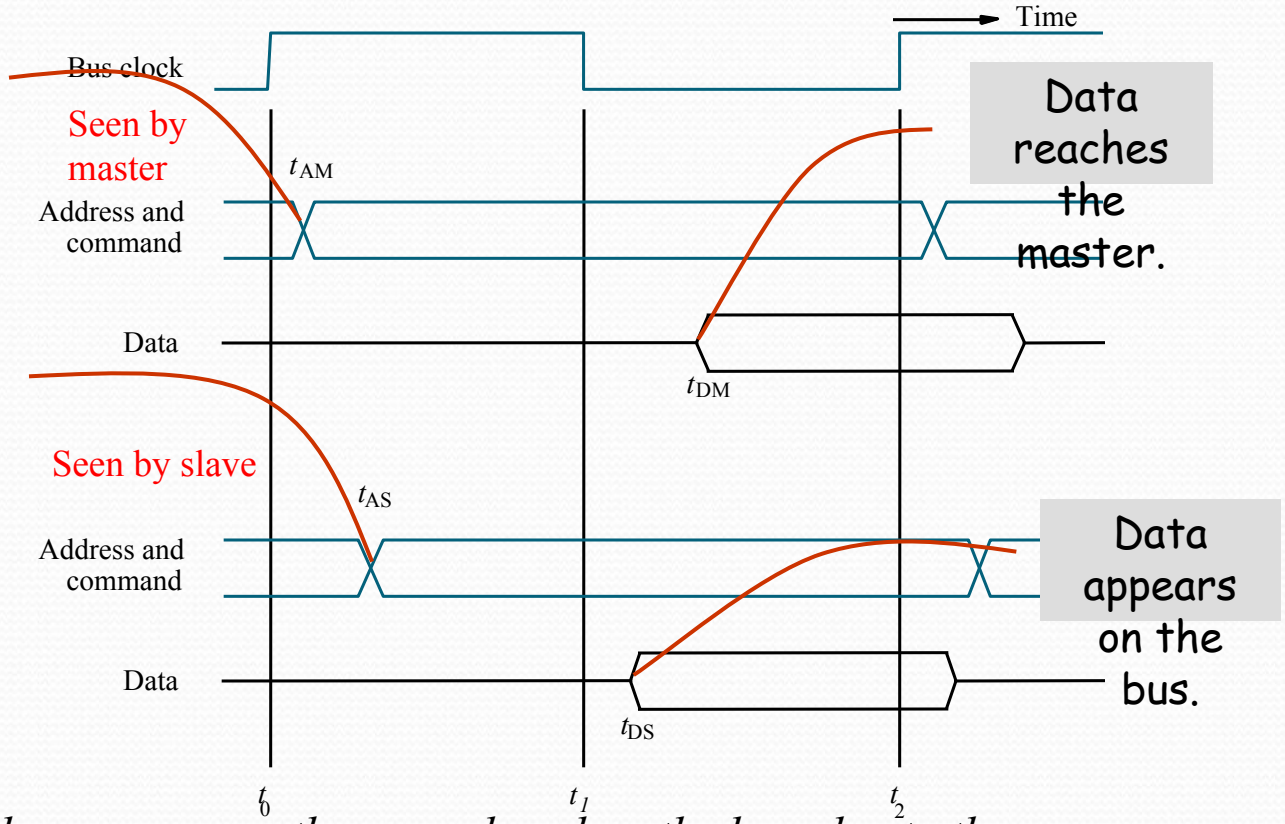
Synchronous bus (contd..)

- Once the master places the device address and command on the bus, it takes time for this information to propagate to the devices:
  - This time depends on the physical and electrical characteristics of the bus.
- Also, all the devices have to be given enough time to decode the address and control signals, so that the addressed slave can place data on the bus.
- Width of the pulse t_1 - t_0 depends on:
  - Maximum propagation delay between two devices connected to the bus.
  - Time taken by all the devices to decode the address and control signals, so that the addressed slave can respond at time t_1.

Synchronous bus (contd..)

- At the end of the clock cycle, at time t_2, the master strobes the data on the data lines into its input buffer if it's a Read operation.
  - "Strobe" means to capture the values of the data and store them into a buffer.
- When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.
- Width of the pulse t_2 - t_1 should be longer than:
  - Maximum propagation time of the bus plus
  - Set up time of the input buffer register of the master.

Synchronous bus (contd..)



Address & command appear on the bus.

Address & command reach the slave.

Data reaches the master.

Data appears on the bus.

s do not appear on the bus as soon as they are placed on the bus, due to the gation delay in the interface circuits.

s reach the devices after a propagation delay which depends on the cteristics of the bus.

nust remain on the bus for some time after t_2 equal to the hold time of the buffer.

Synchronous bus (contd..)

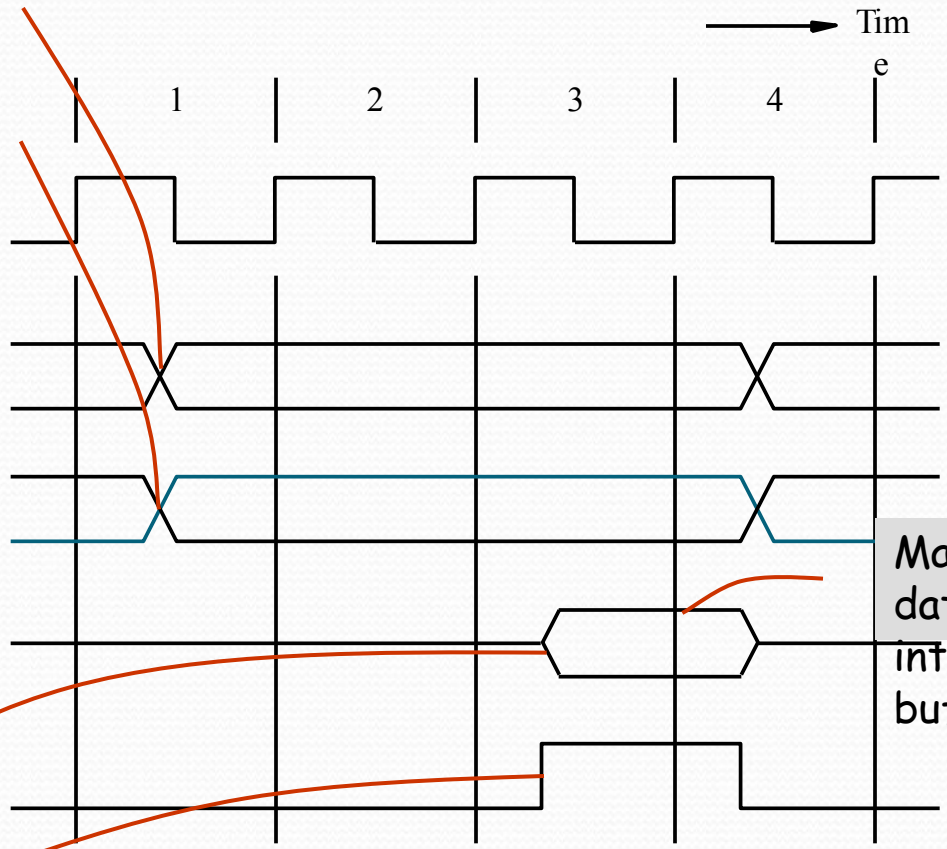
- Data transfer has to be completed within one clock cycle.
  - Clock period t2 - t0 must be such that the longest propagation delay on the bus and the slowest device interface must be accommodated.
  - Forces all the devices to operate at the speed of the slowest device.
- Processor just assumes that the data are available at t2 in case of a Read operation, or are read by the device in case of a Write operation.
  - What if the device is actually failed, and never really responded?

Synchronous bus (contd..)

- Most buses have control signals to represent a response from the slave.
- Control signals serve two purposes:
  - Inform the master that the slave has recognized the address, and is ready to participate in a data transfer operation.
  - Enable to adjust the duration of the data transfer operation based on the speed of the participating slaves.
- High-frequency bus clock is used:
  - Data transfer spans several clock cycles instead of just one clock cycle as in the earlier case.

Synchronous bus (contd..)

Address & command requesting a Read operation appear on the bus.



Master strobes data into the input buffer.

Slave places the data on the bus, and asserts Slave-ready signal.

Clock changes are seen by all the devices at the same time.

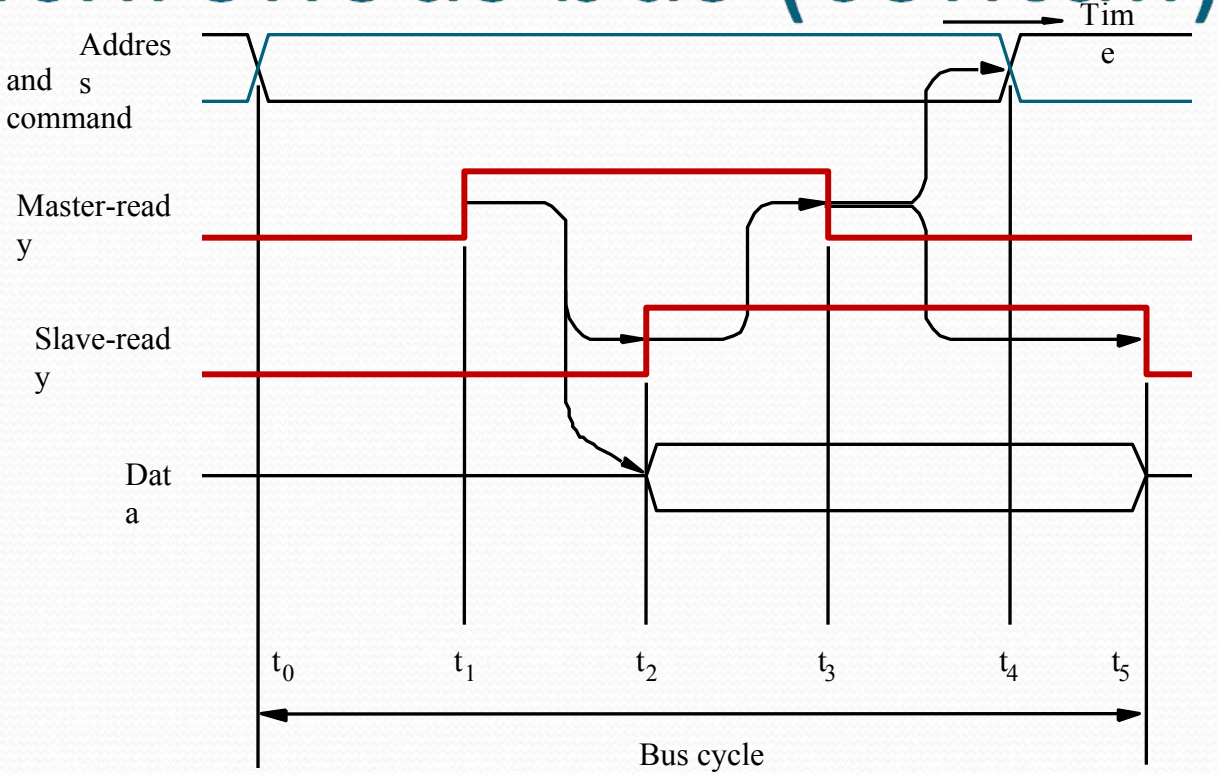
Asynchronous bus

- Data transfers on the bus is controlled by a handshake between the master and the slave.
- Common clock in the synchronous bus case is replaced by two timing control lines:
  - Master-ready,
  - Slave-ready.
- Master-ready signal is asserted by the master to indicate to the slave that it is ready to participate in a data transfer.
- Slave-ready signal is asserted by the slave in response to the master-ready from the master, and it indicates to the master that the slave is ready to participate in a data transfer.

Asynchronous bus (contd..)

- Data transfer using the handshake protocol:
  - Master places the address and command information on the bus.
  - Asserts the Master-ready signal to indicate to the slaves that the address and command information has been placed on the bus.
  - All devices on the bus decode the address.
  - Address slave performs the required operation, and informs the processor it has done so by asserting the Slave-ready signal.
  - Master removes all the signals from the bus, once Slave-ready is asserted.
  - If the operation is a Read operation, Master also strobes the data into its input buffer.

Asynchronous bus (contd..)



t_0 - Master places the address and command information on the bus.

t_1 - Master asserts the Master-ready signal. Master-ready signal is asserted at t_1 instead of t_0

t_2 - Addressed slave places the data on the bus and asserts the Slave-ready signal.

t_3 - Slave-ready signal arrives at the master.

t_4 - Master removes the address and command information.

t_5 - Slave receives the transition of the Master-ready signal from 1 to 0. It removes the data and the Slave-ready signal from the bus.

Asynchronous vs. Synchronous bus

- Advantages of asynchronous bus:
  - Eliminates the need for synchronization between the sender and the receiver.
  - Can accommodate varying delays automatically, using the Slave-ready signal.
- Disadvantages of asynchronous bus:
  - Data transfer rate with full handshake is limited by two-round trip delays.
  - Data transfers using a synchronous bus involves only one round trip delay, and hence a synchronous bus can achieve faster rates.