INPUT/OUTPUT ORGANIZATION

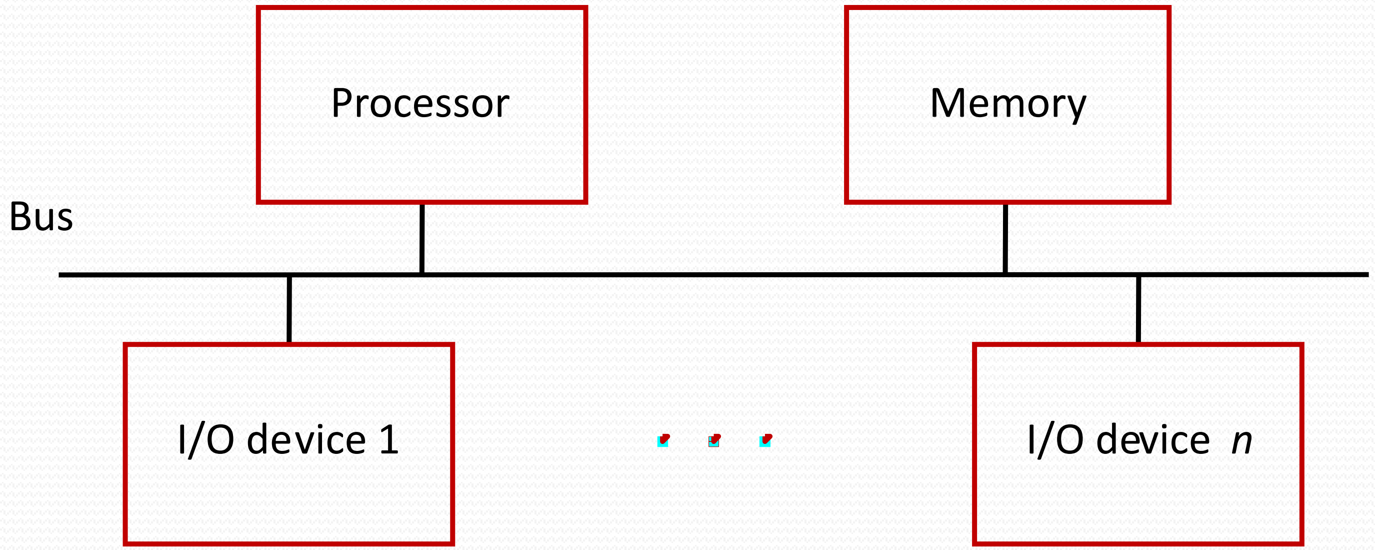
Chapter 4

****Don't depend on slides only.**

*** Self studies are important. You will have questions from self studies. Please feel free to contact me if you have any problem regarding self studies.**

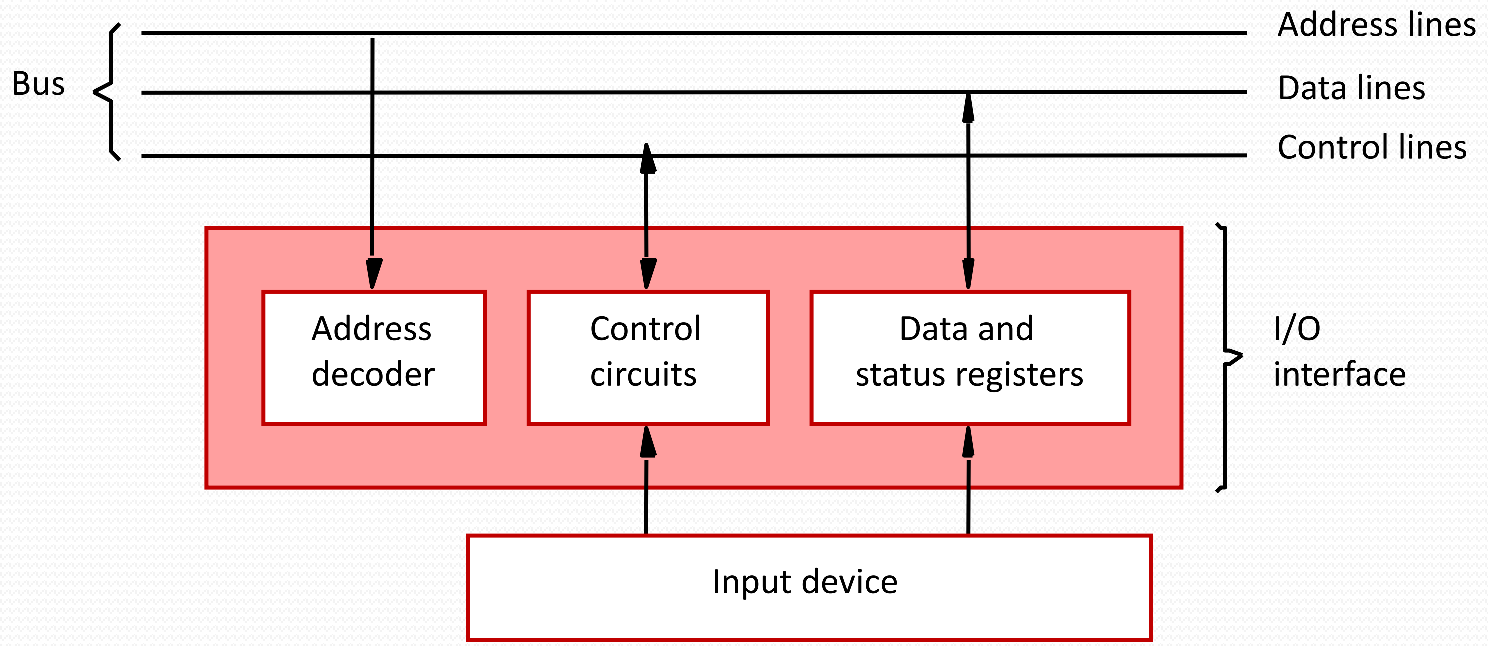
Accessing I/O Devices

Accessing I/O devices



•*Multiple I/O devices may be connected to the processor and the memory via a bus.*
•*Bus consists of three sets of lines/wires to carry address, data and control signals.* (e.g., 32 address lines, 32 data bus lines, 20 control lines)
•*Each I/O device is assigned an* ***unique address****.*
•*To access an I/O device, processor* ***places the address*** *on address lines.*
•*The device recognizes the address, and responds to the control signals.*

Accessing I/O devices (contd..)



•I/O device is connected to the bus using an I/O interface circuit which has:

- Address decoder, control circuit, data registers and status registers.

•Data and status registers have unique addresses & connected to the data lines

•Data register holds the data being transferred to or from the processor.

•Status register holds information necessary for the operation of the I/O device.

•Address decoder decodes the address placed on the address lines thus enabling the device to recognize its address.

•I/O interface circuit coordinates I/O transfers.

Accessing I/O devices …

- 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 memory may have different address spaces:
  - Special instructions to transfer data to and from I/O devices (e.g., IN, OUT)
  - Advantage: I/O devices usually slower. So, possible to treat them specially!
  - Besides, 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 I/O address.

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.
- Three paradigms of techniques to access I/O devices:

1. **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:

2. **Interrupts.**
3. **Direct Memory Access.**

Relative advantages/ disadvantages of the three paradigms of techniques to handle I/O activities

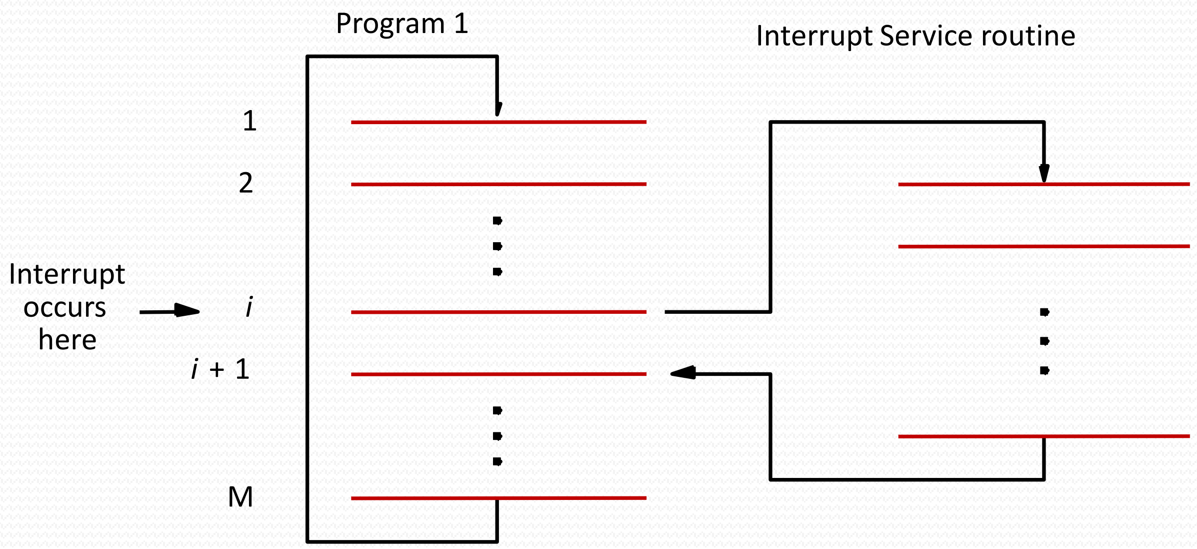
- **Program controlled I/O:**
  - **Advantages:** Very simple to implement. No special hardwire (such as IRQ or ACK control signal/bus lines) is required.
  - **Disadvantages:** The processor has to repeatedly (in a loop) monitor the status registers of the I/O devices in order to detect any I/O activity. This is wasteful, because the processor can't do any other meaningful task in the meantime.
- **Interrupts:**
  - **Advantages:** Does not need continuous attention/monitoring by the Processor. So, the processor can perform other useful tasks.
  - **Disadvantages:** (a) requires special bus lines, such as: IRQ (for interrupt requests) and ACK (for Interrupt Acknowledgements). (b) There may be some delay (interrupt latency). Also, interrupt is not as fast/efficient as the DMA mode.
- **Direct Memory Access (DMA):**
  - **Advantages:** DMA is the most efficient and fastest I/O transfer mode. Can work both 'Burst' and 'cycle stealing' mode, without drawing any attention from the processor. So, the processor is free to do other meaningful tasks in parallel.
  - **Disadvantages:** Most complex to implement. Requires special hardwire (e.g., DMA controller chip, some control bus lines, bus arbitration algorithms) to implement

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 ***special hardware signal*** called an interrupt to the processor.
  - Sent by/from an I/O device to the CPU, to inform the CPU that some I/O activity has taken place at its end.
  - At least one of the bus control lines, called an ***interrupt-request (IRQ)*** line is dedicated for this purpose.
- Processor can perform other useful tasks while it is waiting for the device to be ready.

Interrupts … Interrupt Service Routine



•*Routine (sub-program) executed in response to an interrupt request is called the interrupt-service routine* **(ISR)**. *It is subroutine, defined and designed by either the operating system or the manufacturer of the device that request the interrupt.*

•*Processor is executing the instruction located at address i when an interrupt occurs*

•*When an interrupt occurs, control must be transferred to the interrupt service routine.*

•*But before that, the current contents of the PC (i.e., the value i+1), must be saved in a known location (e.g., some part of the memory maintained as STACK).*

•*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.. Self study)

- 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 … Interrupt Latency

- 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… Interrupt Acknowledgement

- 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.
  - A special control bus line (ACK or IACK) is used for this.
  - In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

Interrupts (contd..) ... IE and ID

- 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 same or any other device while executing an 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 (IE)*** and ***Interrupt-disable (ID)*** 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 (ID).**
  - **Last instruction** of an interrupt service routine can be **Interrupt-enable (IE).**

Interrupts (contd..) ... IE and ID

- 4.2.2 Enabling & disabling interrupts (Self study)

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?

which device Interrupt? … **Polling by ISR**

- 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. (When multiple interrupting devices, the **Priority** of a device set by its **Polling order**)
- Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

which device Interrupt?Self-Identification(Vectored interrupts)

- **Self-Identification:** The device requesting an interrupt may identify itself directly to the processor.
  - Device can do so by sending it own, **unique ID** (a special code - may be just 4 to 8 bits code) to the processor over the **data 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..)

- 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..)

- 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..)

- 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.

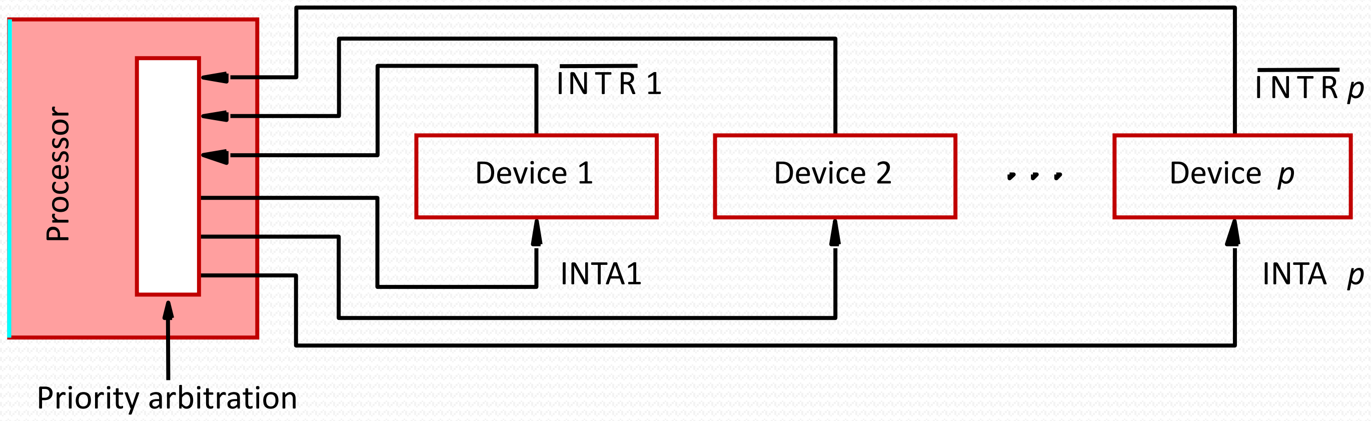
Interrupts (contd..)   Self-study

- 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.

How to Handle Simultaneous Interrupt Requests from I/O devices?

- If more than one device interrupt simultaneously, the device with **Highest Priority** gets the service (i.e., its Interrupt Service Routine is executed)!
- Four Schemes to handle simultaneous interrupt requests:
  - Priority Structured Scheme
  - Polling Scheme
  - Daisy Chain Scheme
  - Prioritized Daisy Chain Scheme

Priority Structured Scheme



•*Each device has a separate interrupt-request and interrupt-acknowledge line.*

•***Each interrupt-request line is assigned a different priority level*** **(but, usually it can be customized by Operating System)**

•*Interrupt requests received over these lines are sent to **<u>a priority arbitration circuit</u>** in the processor.*

•*If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.*

Handling Simultaneous Interrupt Requests … (contd..)

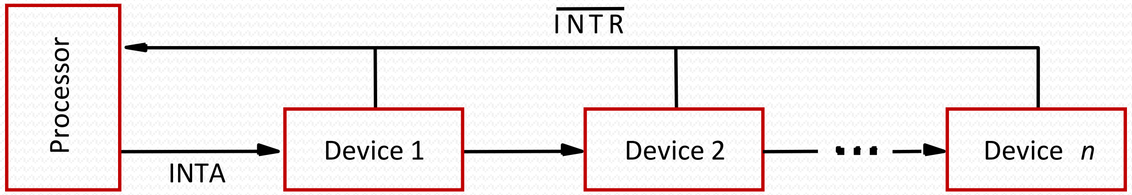
- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
  - Each device has its own interrupt 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? **Three Techniques:**
  - Polling,
  - Daisy Chain
  - Prioritized Daisy Chain

Handling Simultaneous Interrupt Requests in a System with Shared IRQ Lines

Polling scheme:

- *If the processor uses a 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.***
- *The first device with status bit set to 1 is the device whose interrupt request is accepted.*

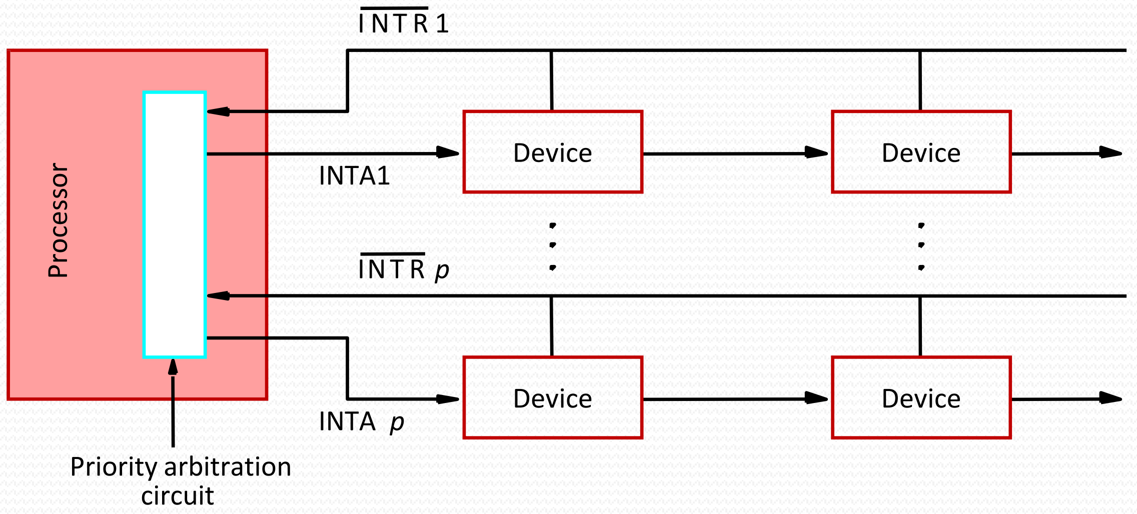
Daisy chain scheme:



- *Devices are connected to form a daisy chain.*
- *Devices share the interrupt-request line, and interrupt-acknowledge line is connected to form a daisy chain.*
- *When devices raise an interrupt request, the interrupt-request line is activated.*
- *The processor in response activates interrupt-acknowledge.*
- *Received by device 1, if device 1 does not need service, it passes the signal to device 2.*
- ***Priority: Device that is electrically closest to the processor has the highest priority.*** **(Here, Priority = Electronic Proximity of the Device to the Processor)**

Handling Simultaneous Interrupt Requests in a System with Shared IRQ Lines … (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 a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.*
- ***Prioritized Daisy Chain Scheme:** A combination of priority structure and daisy chain schemes*


$\overline{\text{INTR}}\,1$
Processor
Device
Device
INTA1
$\overline{\text{INTR}}\,p$
Device
Device
INTA p
Priority arbitration circuit

- *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.* ***Inside a Group, priority is determined by electronic proximity to the Processor.***

Interrupts (contd…) … IE bit

- **Only those devices that are being used in a program should be allowed to generate interrupt requests.**
- To control which devices allowed to generate interrupt requests, the interface circuit of **each I/O device has an interrupt-enable bit.**
  - If the interrupt-enable (***IE***) bit in the I/O device interface is set to 1, then the device is allowed to generate an interrupt-request.
- Interrupt-enable (***IE***) bit in the device's interface circuit determines whether the device is allowed to generate an interrupt request.
- Interrupt-enable (***IE***) bit in the processor status register determines whether interrupts allowed or not. (This bit is used by the *ID* and *IE* instructions of ISR. ID resets the IE bit of Processor, but IE sets the IE bit)

Exceptions Self-Study

- 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..) Self-Study

- 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..) Self-Study

- 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..) Self-Study

- 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

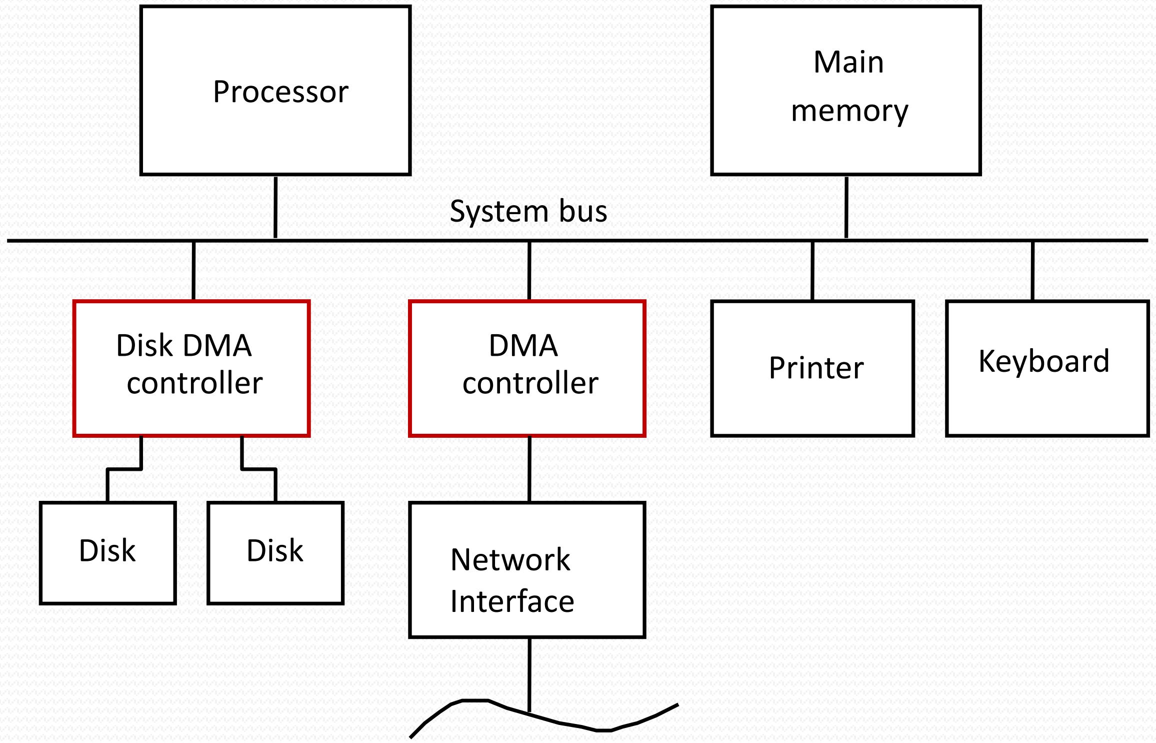
Direct Memory Access (contd..)

- 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 memory, 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 (of both source and destination)
  - 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. It provides two DMA channels.*
- *It can perform two independent DMA operations, as if each disk has its own DMA controller. The registers to store the memory address, word count and status and control information are duplicated.*

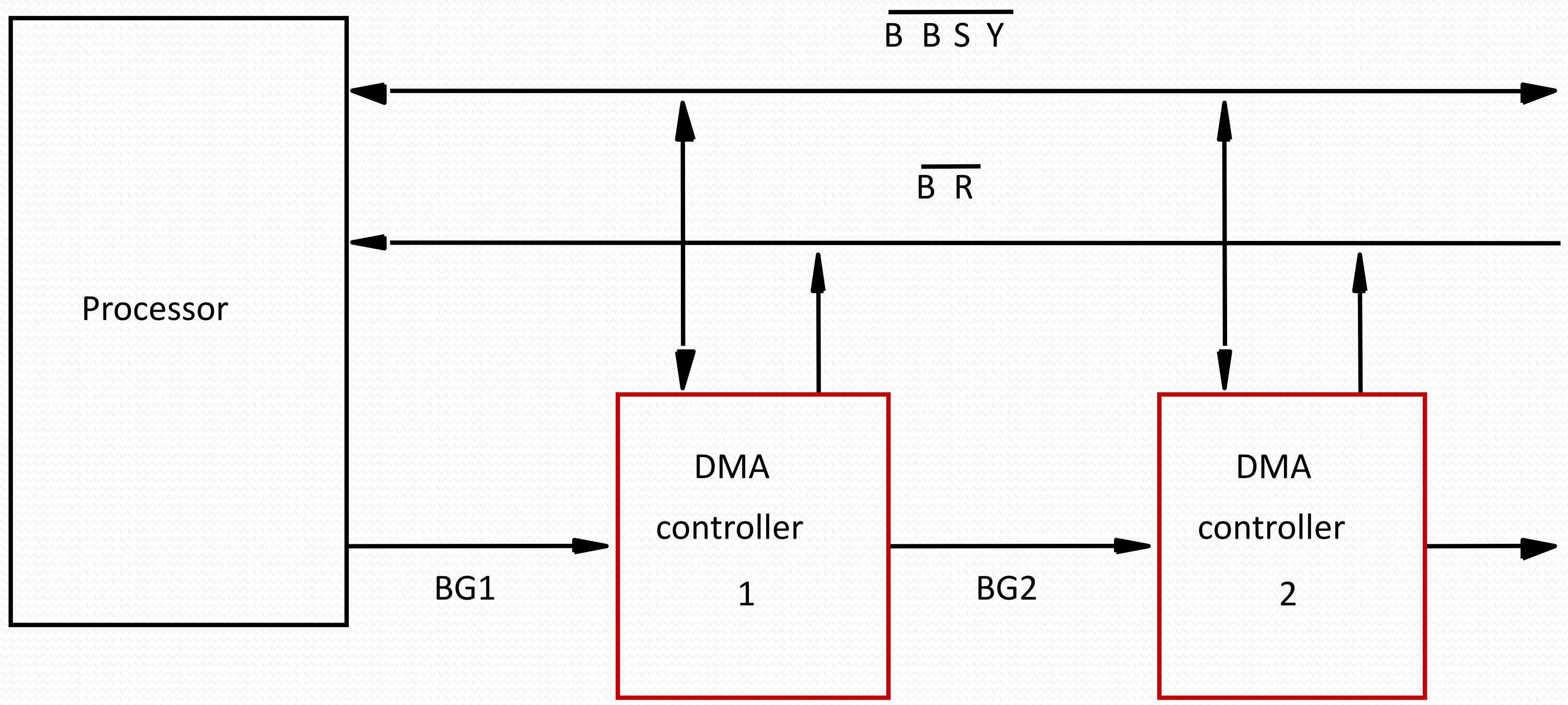
Direct Memory Access (contd..)

- Processor and DMA controllers have to use the bus in an interwoven fashion to access the memory.
  - DMA devices are given higher priority than processor to access bus.
  - Among different DMA devices, high priority is given to high-speed peripherals such as a disk or a graphics display device.
- Processor starts 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 DMA mode. To do this, the DMA Controller (instead of the Processor) has to be the Bus Master during the transfer. section 4.4 (Self Study)**

Bus arbitration (self study)

- 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 bus arbitration:**
  - A single bus arbiter performs the arbitration.
- **Distributed bus arbitration:**
  - All devices participate in the selection of the next bus master.

Centralized Bus Arbitration Procedure/Algorithm



Centralized Bus Arbitration …

- *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.*
- ***A DMA controller*** **(e.g., DMA Controller 2)** ***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 => BG2 reaches DMA Controller 2 to make it Bus Master*
- ***BBSY signal is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller 2, which asserted BR, can acquire control of the bus.***

Distributed Bus arbitration Procedure

- 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 Unique 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 ...

- *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 0's on the arbitration lines for that & all lower bit positions* (denoted by '*'-s in the following example)**
  - *The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.*

$$\begin{array}{l} \text{Device A ID=5 => } \mathbf{0101} \\ \text{Device B ID=6 => } \mathbf{0110} \end{array} \rightarrow \mathbf{0111} \rightarrow \begin{array}{c} \mathbf{01}\overset{*}{\mathbf{0}}\overset{*}{\mathbf{0}} \\ \mathbf{011}\underset{*}{\mathbf{0}} \end{array} \rightarrow \mathbf{0110} \rightarrow \textbf{B wins}$$

Distributed arbitration ...

- *Device A has the ID 5 and wants to request the bus:*
  - *Transmits the pattern **<u>0101</u>** on the arbitration lines.*
- *Device B has the ID 6 and wants to request the bus:*
  - *Transmits the pattern **<u>0110</u>** on the arbitration lines.*
- *Pattern that appears on the arbitration lines is the logical OR of the patterns:*
  - *Pattern 0111 appears on the arbitration lines.*

<u>Arbitration process:</u>

- *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.*
- *Device A compares its ID 5 with a pattern 0101 to pattern 0111.*
- *It detects a difference at bit position 0, as a result, it transmits a pattern 0100 on the arbitration lines.*
- *The pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, which is 0110.*
- *This pattern is the same as the device ID of B, and hence B has won the arbitration procedure and becomes the Bus Master.*