# INPUT/OUTPUT ORGANIZATION

UNIT-II

**A. M. Chandrashekhar**
**Asst. Prof. CS & E Dept.**
**S. J. C. E - Mysore**

# UNIT-2.

# Input/Output Organization (10 Hours)

## Topics To Be Covered:

Basic Input/Output Operations
Accessing I/O Devices,
Interrupts,
Bus Structure,
Arbitration,
Interface Circuits

# Basic I/O Operations

- 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.

# Need of Synchronization of Data Transfer

- I/O devices is slower than the speed of the processor.
- 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

- Task: Read in character input from a keyboard and produce character output on a display screen.
- Rate of data transfer (keyboard, display, processor)
- Need handle Difference of speed in processor and I/O device
- 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

Assumption – the initial state of SIN is 0 and the initial state of SOUT is 1.

SIN=1 : processor reads

SOUT= 1 processor sends data

# Ways for 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 : MOV
  - Simpler software.
- I/O devices and the memory may have different address spaces: (I/O Mapped I/O)
  - Special instructions to transfer data to and from I/O devices : IN & OUT
  - 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.

# Program-Controlled I/O .....

## Implementation 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

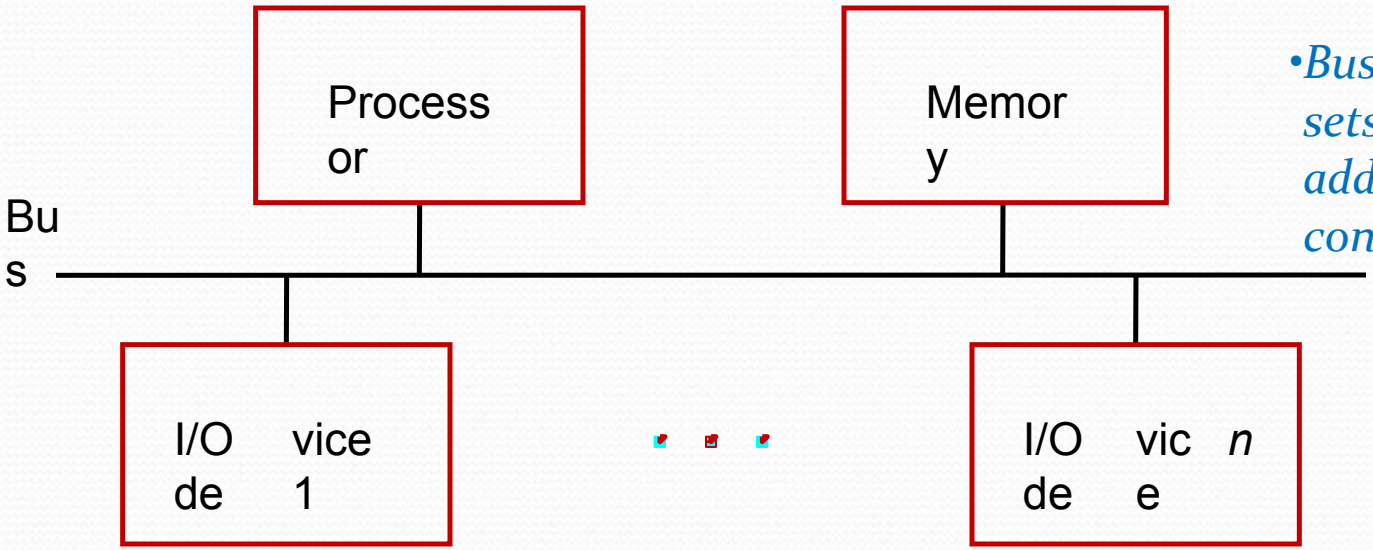
WRITEWAIT Branch to WRITEWAIT if SOUT = 0
Output from R1 to DATAOUT

# Memory Mapped I/O

- Memory-Mapped I/O – some memory address values are used to refer to peripheral device buffer registers.
- No special instructions are needed.
- Also use device status registers.

```
READWAIT  Testbit   #3, INSTATUS
          Branch=0  READWAIT
          MoveByte  DATAIN, R1
```

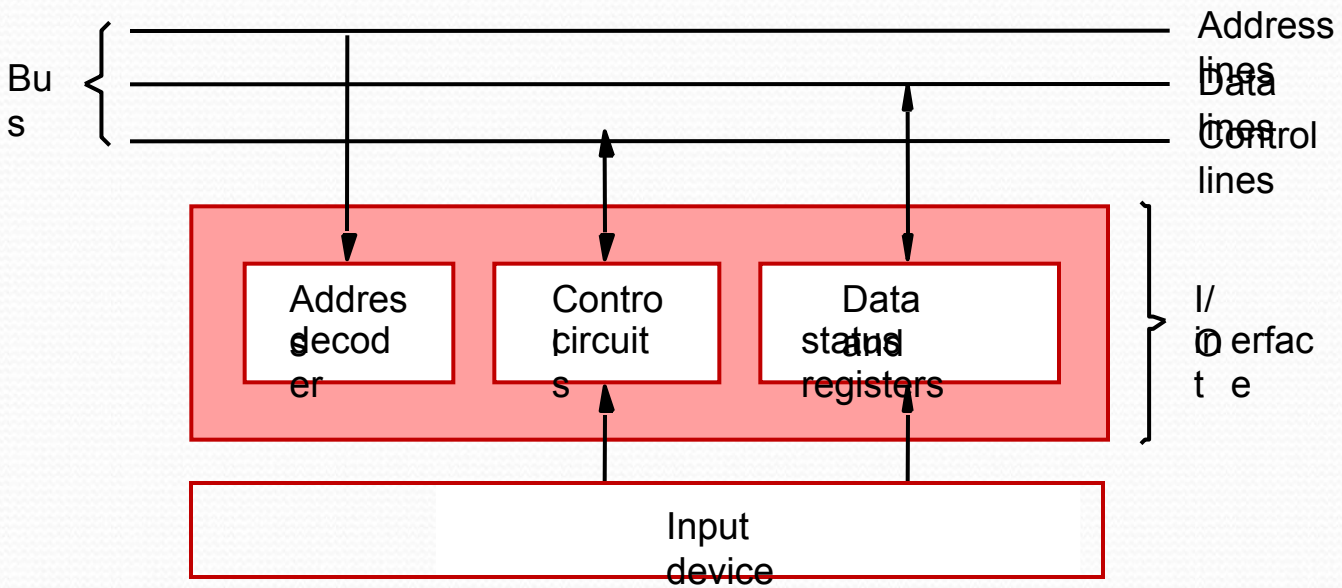
# Accessing I/O devices




- *Bus consists of three sets of lines to carry address, data and control signals.*

- *Multiple I/O devices may be connected to the processor and the memory via a bus.*
- *Each I/O device is assigned an unique address.*
- *To access an I/O device, the processor places the address on the 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, and data and status registers.*
  - *I/O interface circuit coordinates I/O transfers.*

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

- *Data and status registers are connected to the data lines, and have unique addresses.*

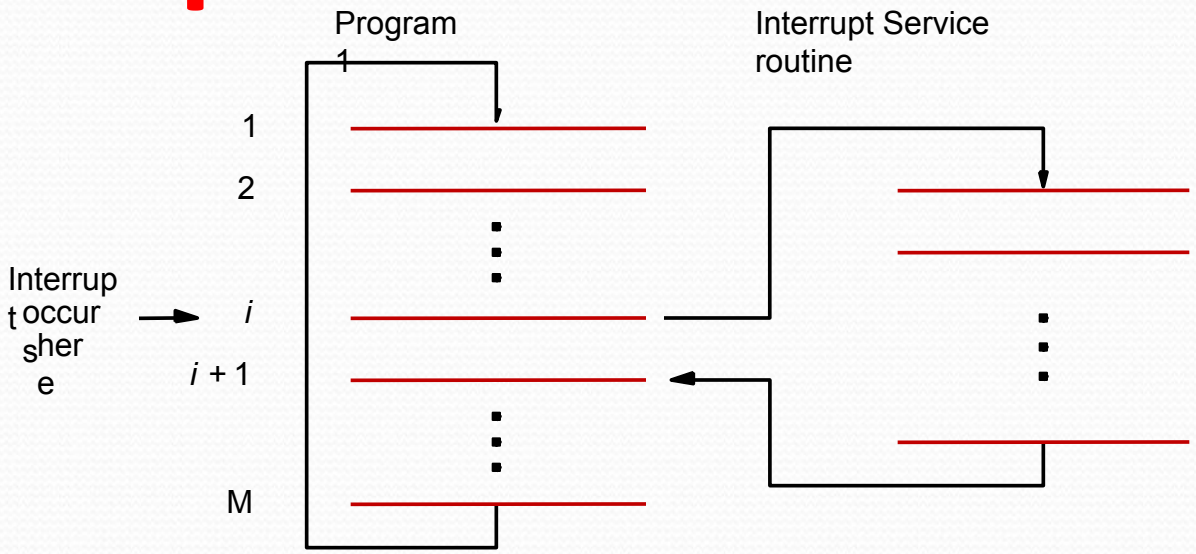
# Interrupts

- 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.

# Interrupt Service Routine (ISR)




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

# Interrupts v/s Subroutine

- 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.

# Interrupt Handling

- 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 by sending INTA control signal
- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:

# Interrupt Handling (contd..)

- 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.

# Handling Multiple Interrupts

- 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.

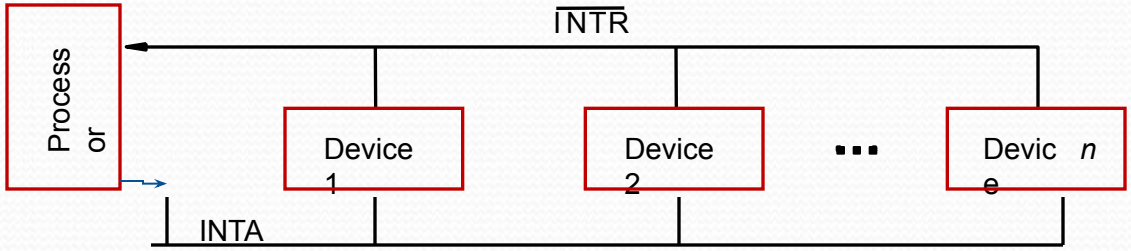
**Following Question Arises**

- 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?

# Vectored Interrupt scheme

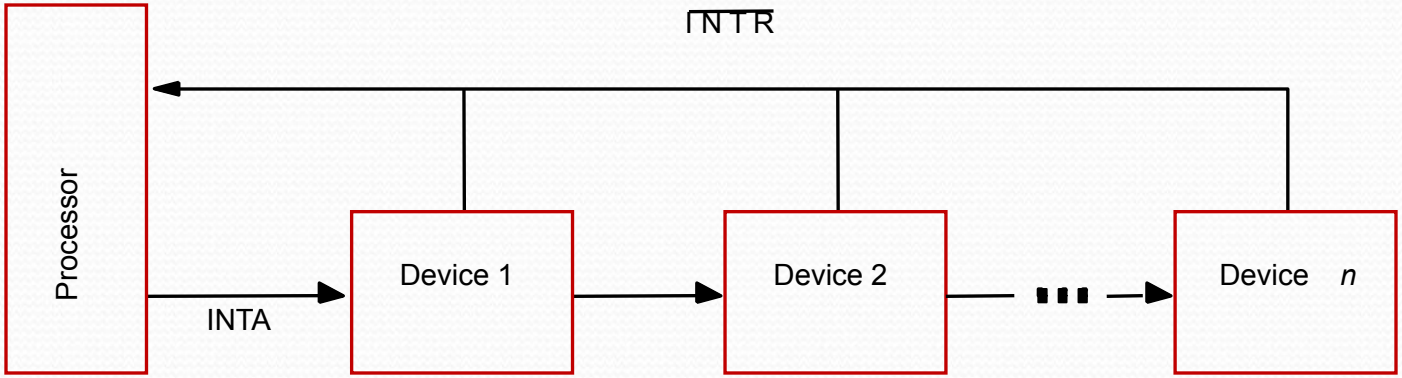
- The device requesting an interrupt may identify itself directly to the processor.
  - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
  - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
  - The remainder 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.

# Polling scheme in Interrupts




- 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.

# Daisy chain Scheme




- *Devices are 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.*
- *INTA is 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.*

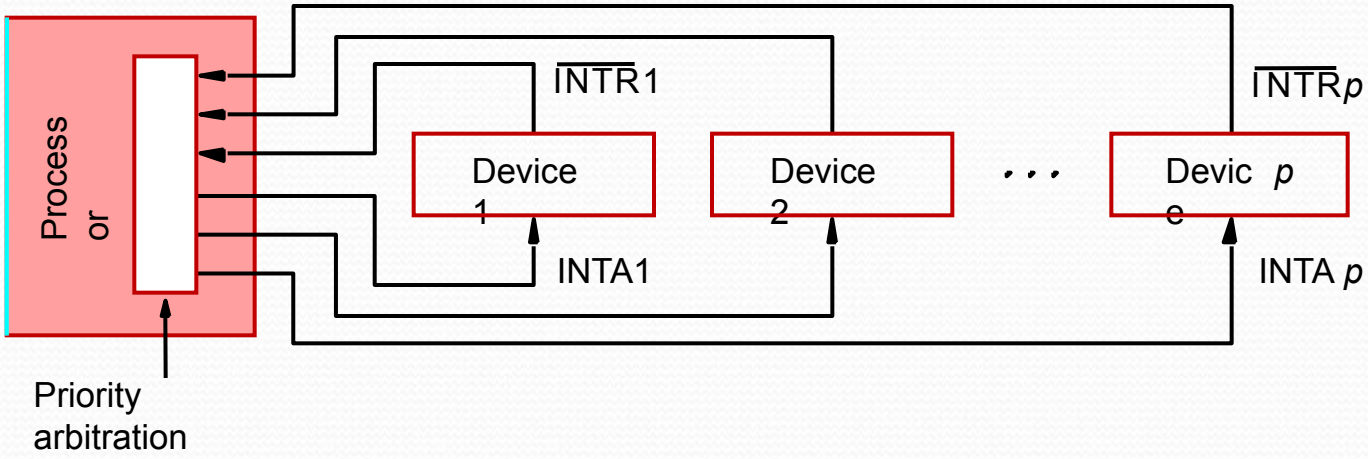
# Priority Structure Scheme

- 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.

**Processor's priority is encoded in a few bits of the processor status register.**

# Priority Structure Scheme




- *Each device has a 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.*

# Handling Simultaneous Requests

- 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.
- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

*In Polling scheme:*

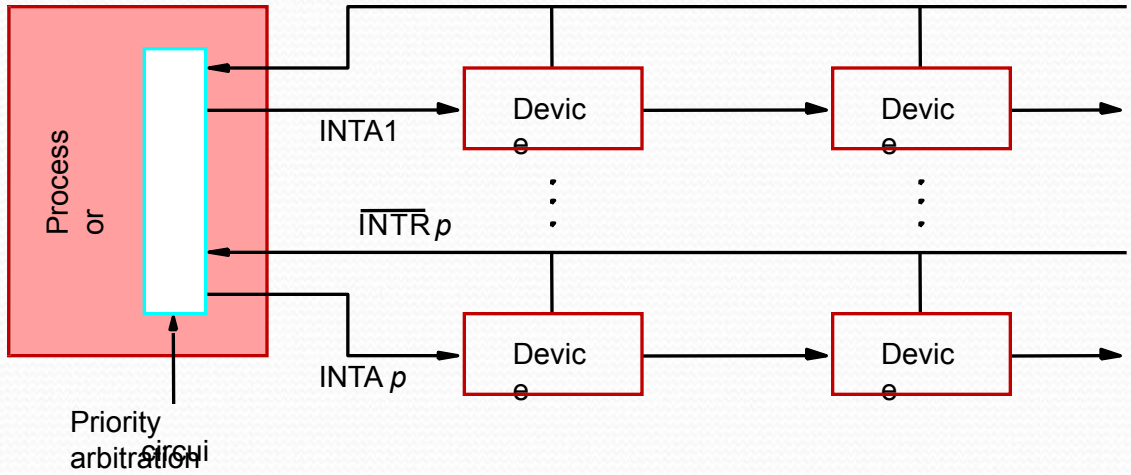
- *In this case the priority is determined by the order in which the devices are polled.*
- *The first device with status bit set to 1 is the device whose interrupt request is accepted.*

*In Daisy chain scheme:*

- *Device that is electrically closest to the processor has the highest priority.*

# Combination of schemes

- *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.*
- *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.*

# Exceptions

- Interrupts caused by interrupt-requests sent by I/O devices.
- 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
- 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.

# Exceptions (contd..)

- 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.
- Debugger uses exceptions to provide important features: like Trace and Breakpoints.
- Trace mode:
  - Exception occurs after the execution of every instruction.
- Breakpoints:
  - Exception occurs only at specific points selected by the user.

**Certain instructions can be executed only when the processor is in the supervisor mode. These are called privileged instructions.**

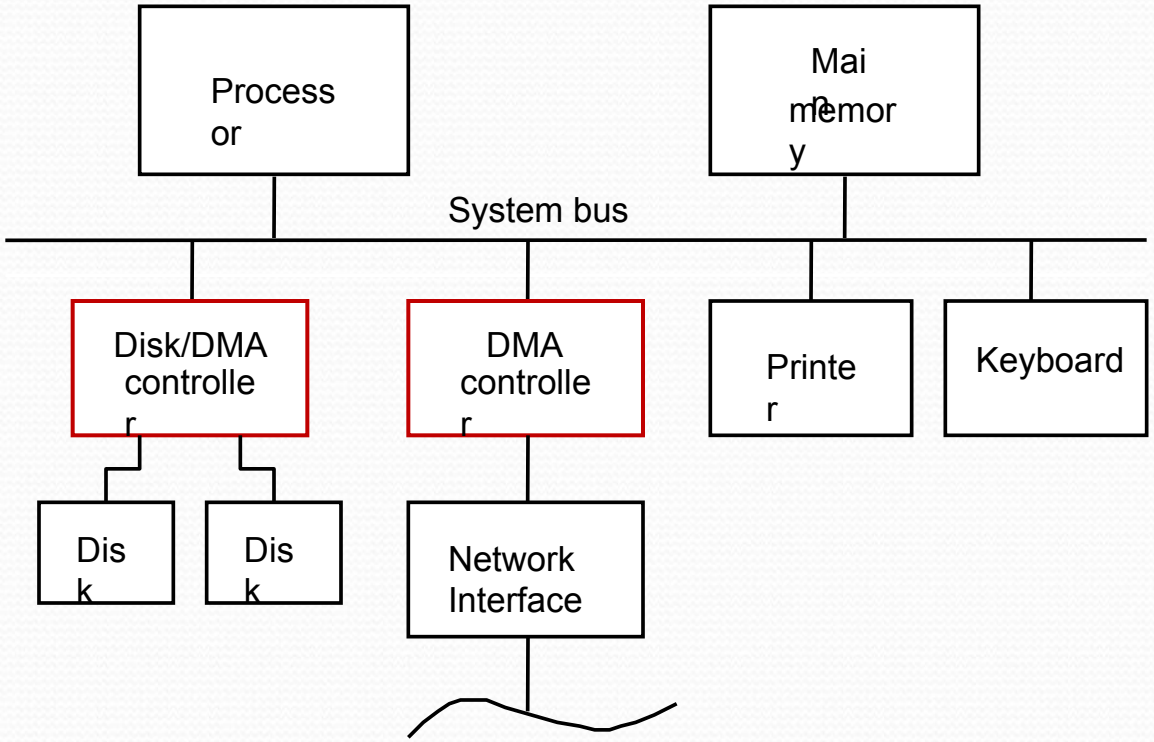
# Direct Memory Access

- 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: co-processor
- DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor. (processor must initiate the DMA transfer.)

# Direct Memory Access (contd..)

- 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 (Cont…)




- *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.*

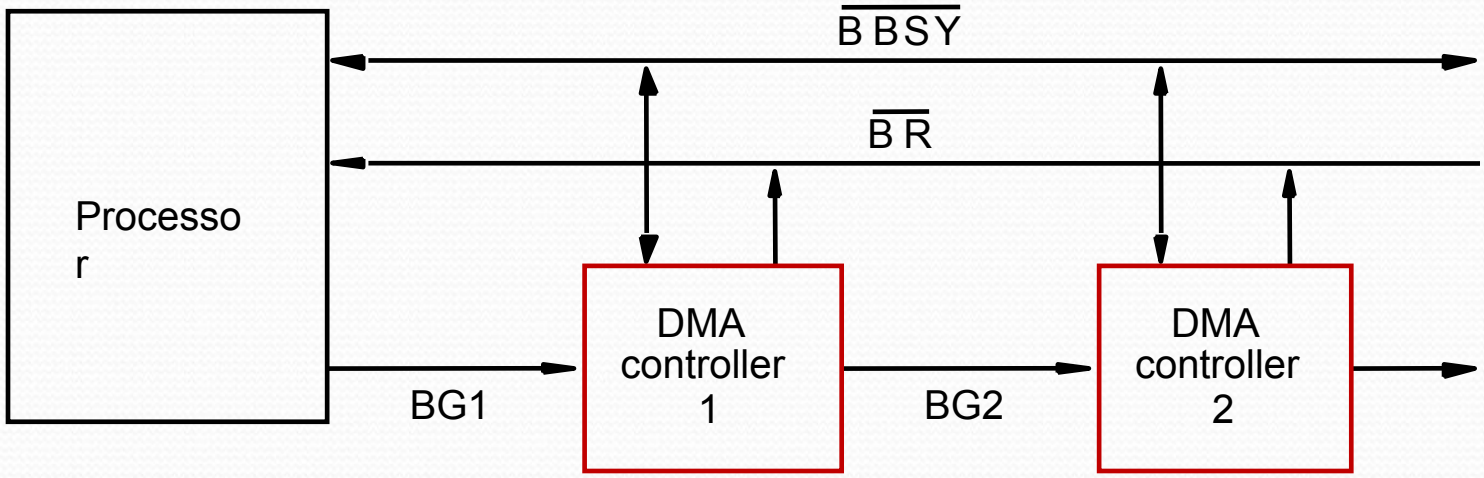
# Direct Memory Access (contd..)

- 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

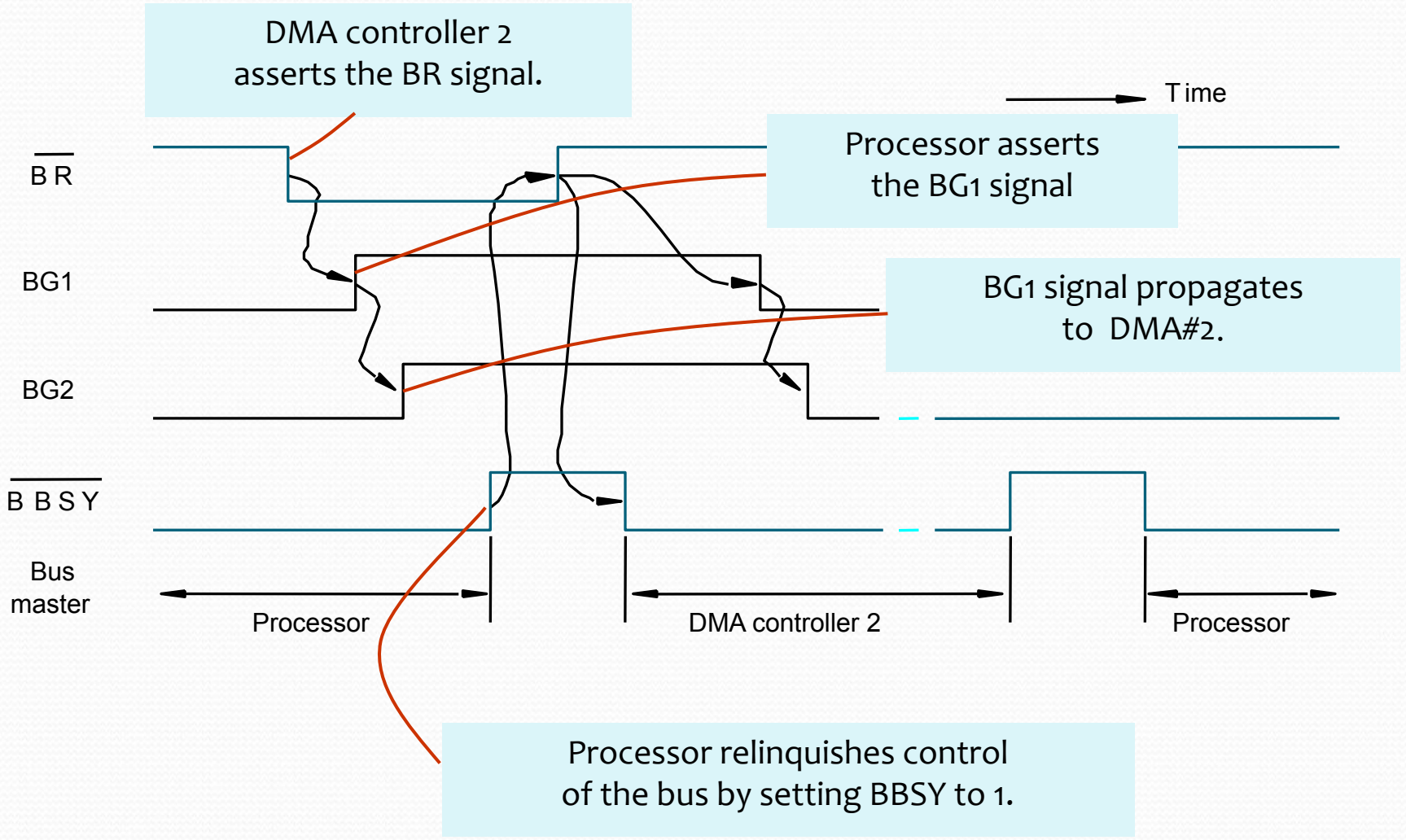
- Processor and DMA controllers both need to initiate data transfers on the bus and access main memory.
- The device that is allowed to initiate transfers on the bus at any given time is called the bus master.
- *Normally, the processor is the bus master, unless it grants bus membership to one of the DMA controllers.*
- 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




- **Bus arbiter may be the processor or a separate unit connected to bus.**
- **DMA controller requests the control of the bus by asserting the Bus Request (BR) line. In response, the processor activates the Bus-Granti (BG1) line, indicating that the controller may use the bus if it is free.**
- **BG1 signal is connected to all DMA controllers in 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.**

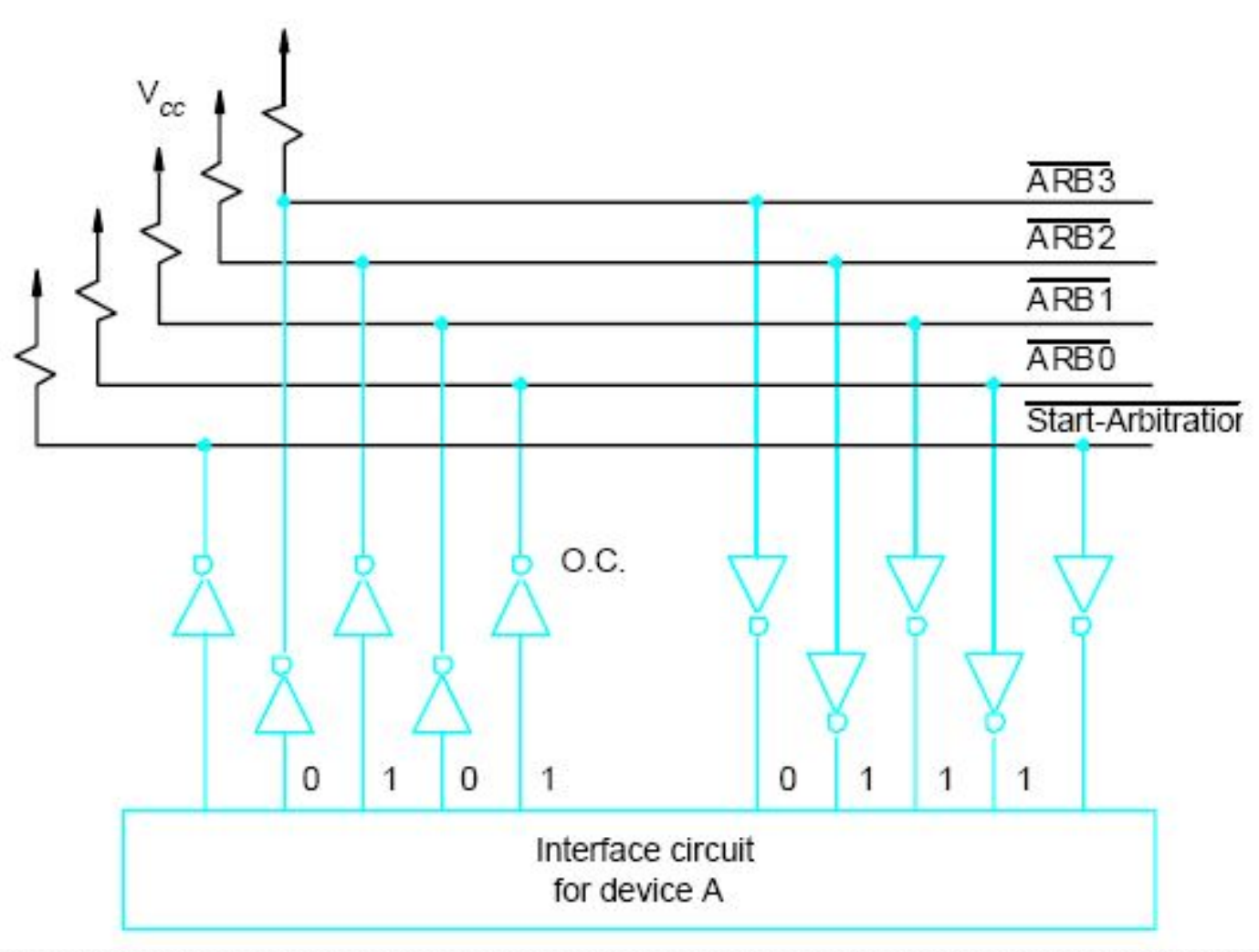
# Centralized arbitration (contd..)



# 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 os 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..)

- *Device A has the ID 5 and wants to request the bus:*
  - *Transmits the pattern 0101 on the arbitration lines.*
- *Device B has the ID 6 and wants to request the bus:*
  - *Transmits the pattern 0110 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*

*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.*
- *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*

# Buses

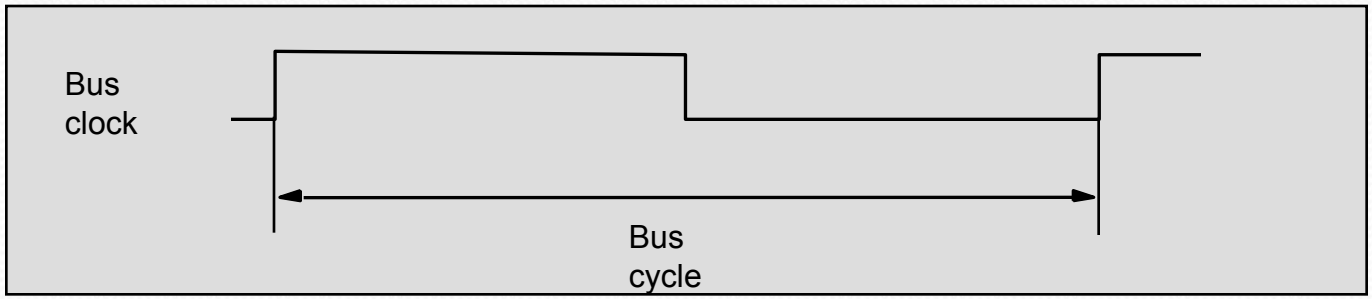
# Buses

- 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.
- 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.

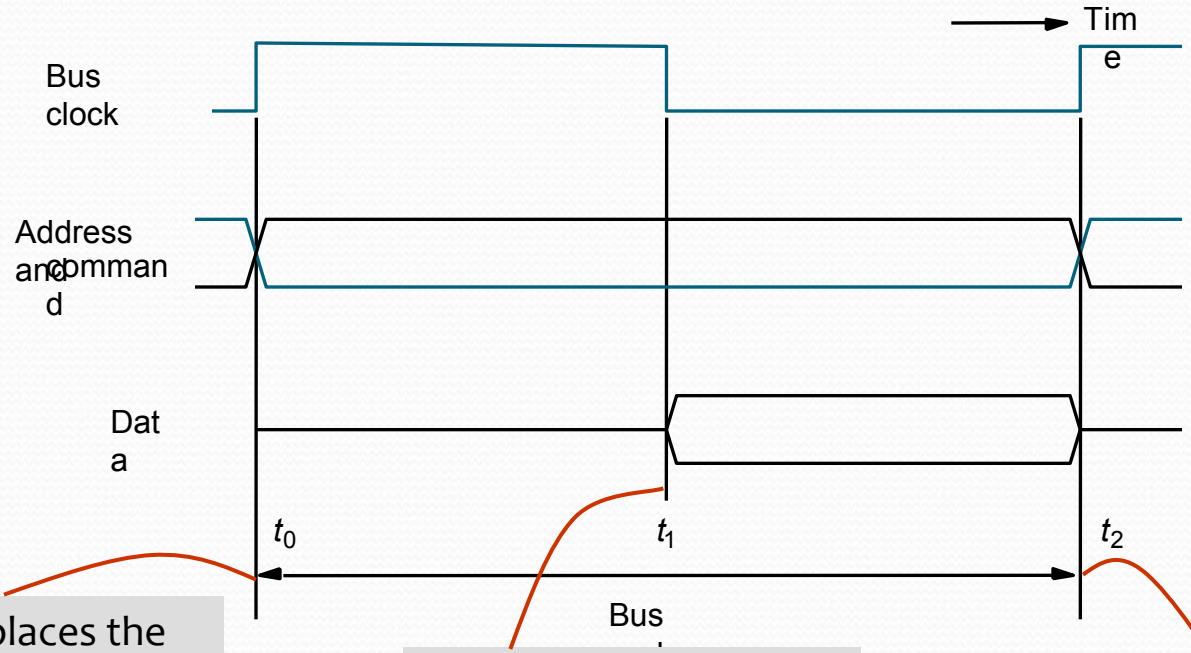
# Buses (contd..)

- 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.

- *In case of a Write operation, the master places the data on the bus along with the address and commands at time $t_o$.*
- *The slave strobes the data into its input buffer at time $t_2$.*

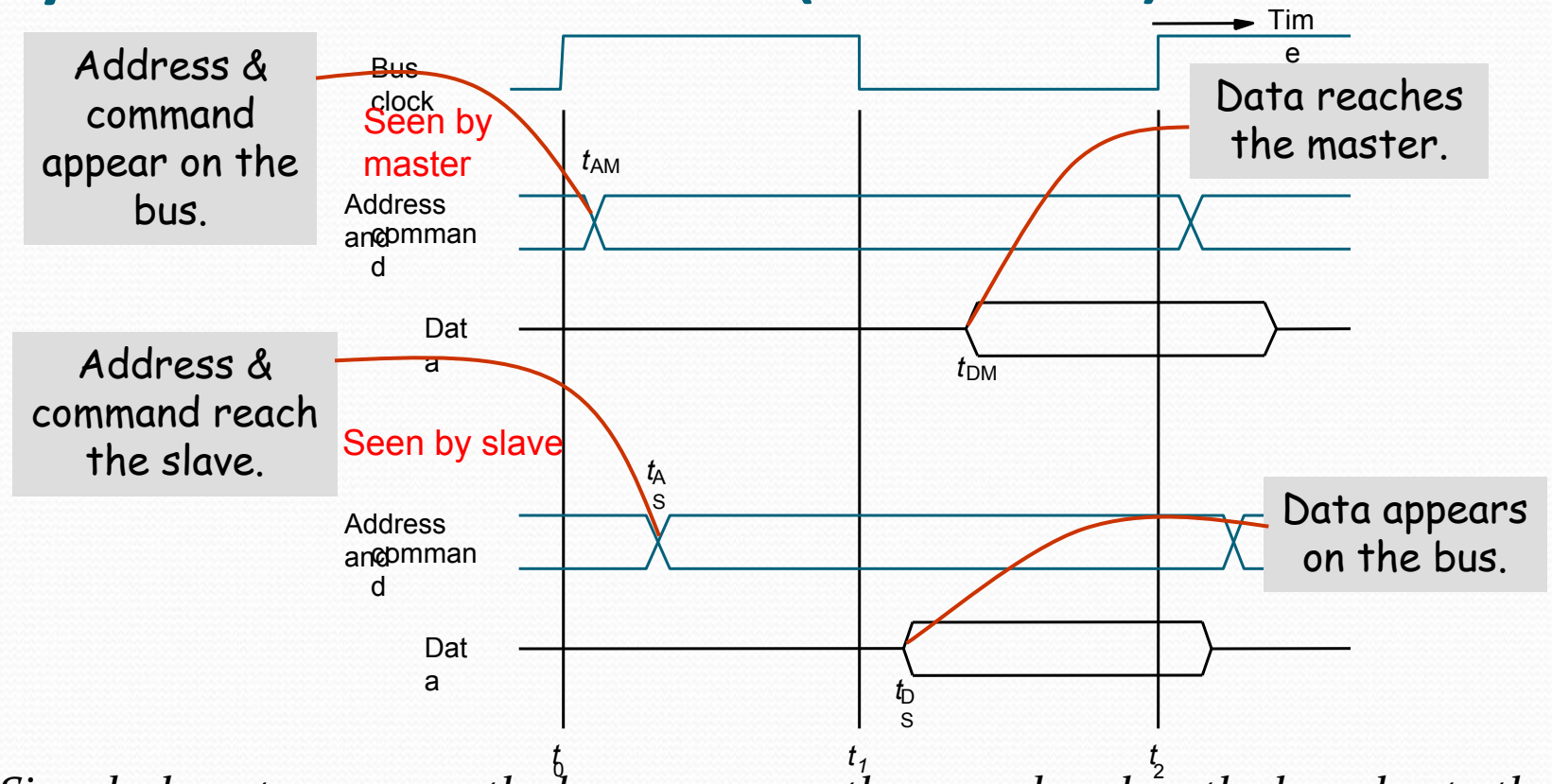
# Synchronous bus (contd..)

- 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..)




- *Signals do not appear on the bus as soon as they are placed on the bus, due to the propagation delay in the interface circuits.*
- *Signals reach the devices after a propagation delay which depends on the characteristics of the bus.*
- *Data must remain on the bus for some time after $t_2$ equal to the hold time of the buffer.*

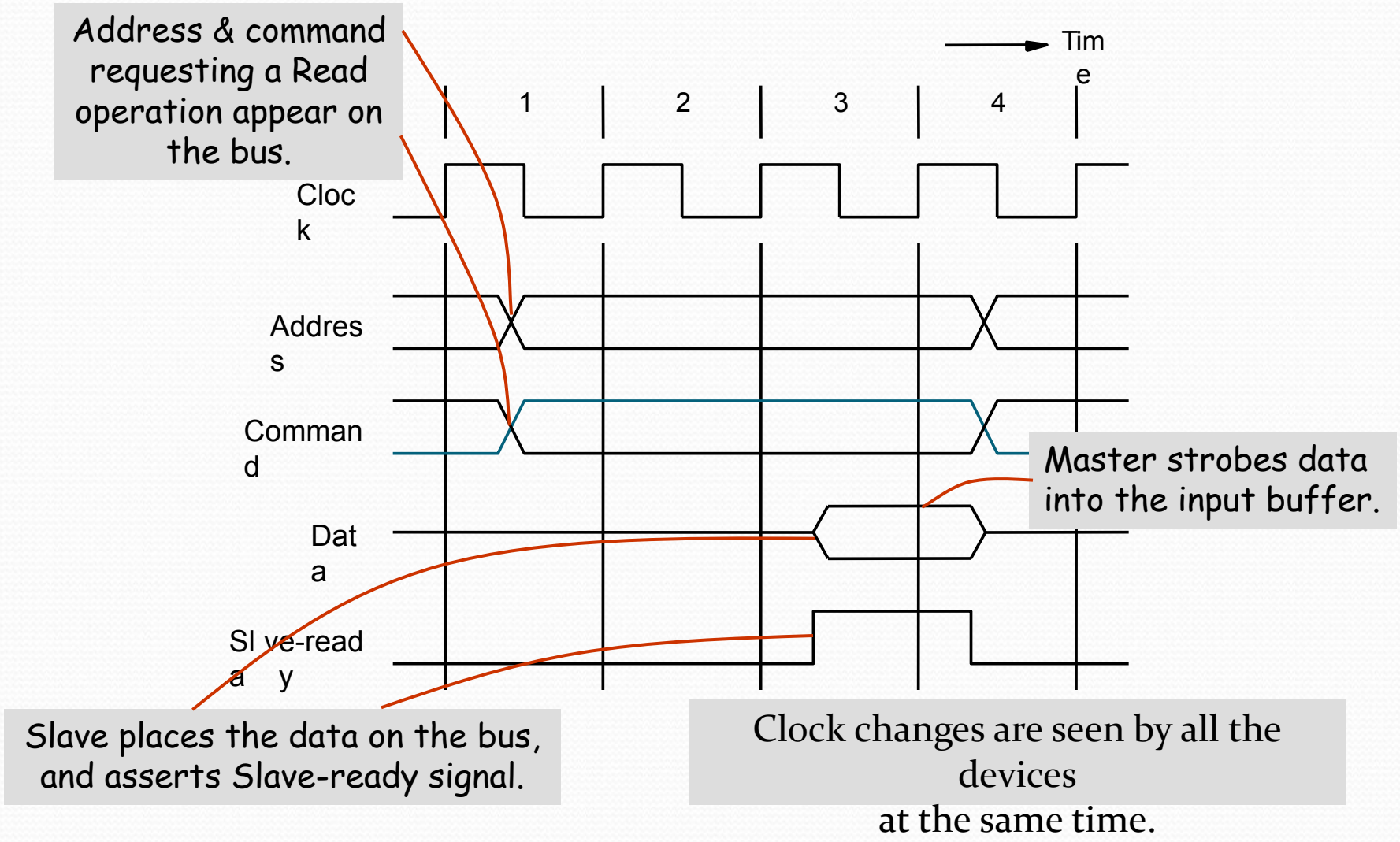
# Synchronous bus (contd..)

- Data transfer has to be completed within one clock cycle.
  - Clock period t2 - to 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..)



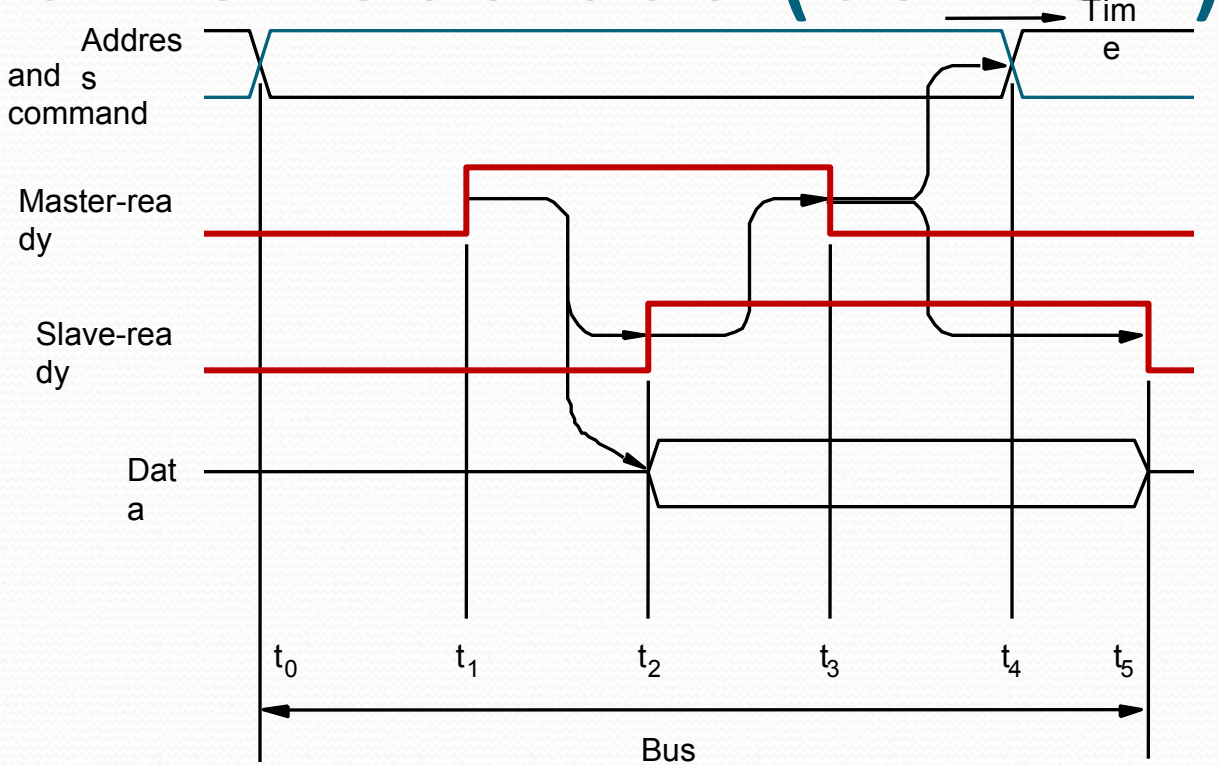
# Asynchronous bus

- 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 the declaration of master to indicate to the slave that it is ready to participate in a data transfer.
- Slave-ready signal is the declaration of slave in response to the master-ready and it indicates 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.

# Interface Circuits

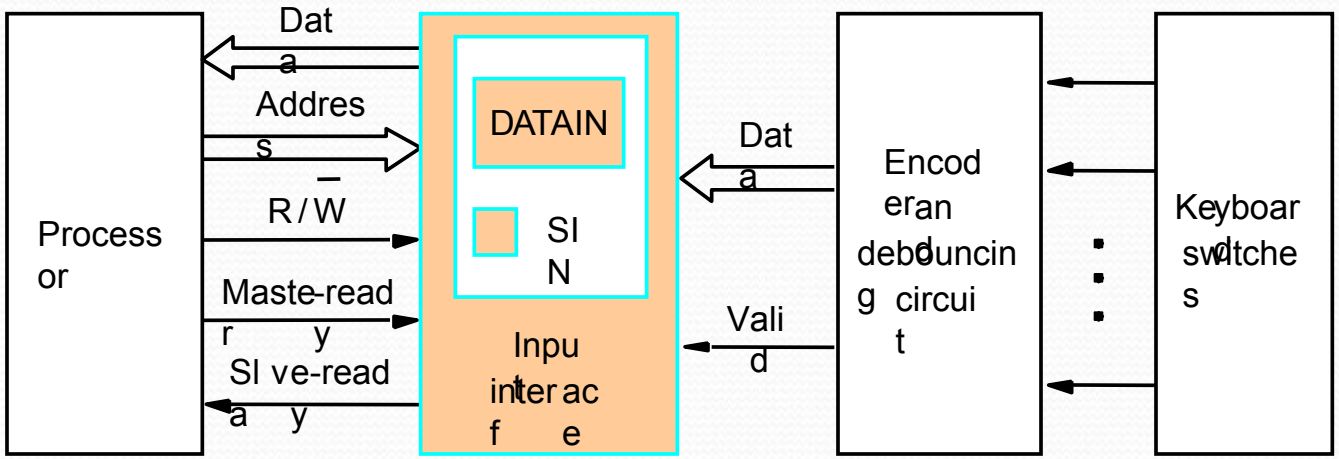
# Interface circuits

- I/O interface consists of the circuitry required to connect an I/O device to a computer bus.
- Side of the interface which connects to the computer has bus signals for:
  - Address,
  - Data
  - Control
- Side of the interface which connects to the I/O device has:
  - Datapath and associated controls to transfer data between the interface and the I/O device.
  - This side is called as a "port".
- Ports can be classified into two:
  - Parallel port,
  - Serial port.

# Interface circuits (contd..)

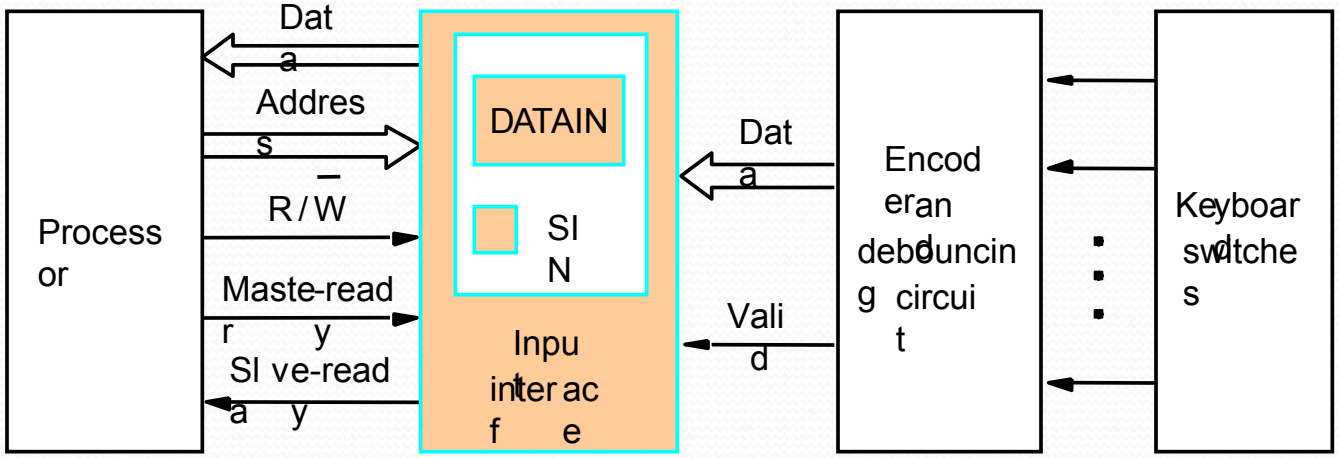
- Parallel port transfers data in the form of a number of bits, normally 8 or 16 to or from the device.
- Serial port transfers and receives data one bit at a time.
- Processor communicates with the bus in the same way, whether it is a parallel port or a serial port.
  - Conversion from the parallel to serial and vice versa takes place inside the interface circuit.

# Parallel port



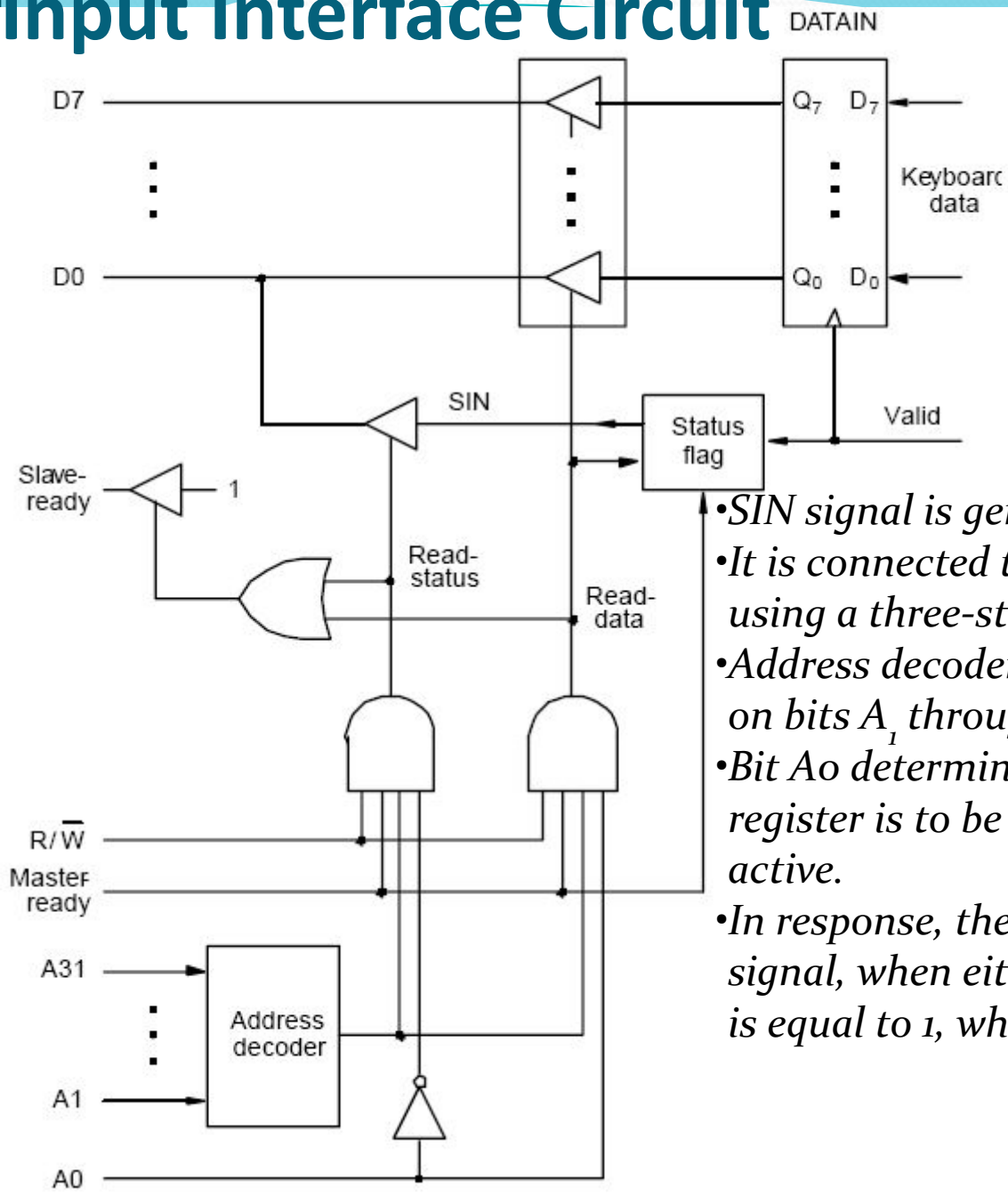

- *Keyboard is connected to a processor using a parallel port.*
- *Processor is 32-bits and uses memory-mapped I/O and the asynchronous bus protocol.*
- *On the processor side of the interface we have:*
  - *Data lines.*
  - *Address lines*
  - *Control or R/W line.*
  - *Master-ready signal and*
  - *Slave-ready signal.*

# Parallel port (contd..)




- *On the keyboard side of the interface:*
  - *Encoder circuit which generates a code for the key pressed.*
  - *Debouncing circuit which eliminates the effect of a key bounce (a single key stroke may appear as multiple events to a processor).*
  - *Data lines contain the code for the key.*
  - *Valid line changes from 0 to 1 when the key is pressed. This causes the code to be loaded into DATAIN and SIN to be set to 1.*

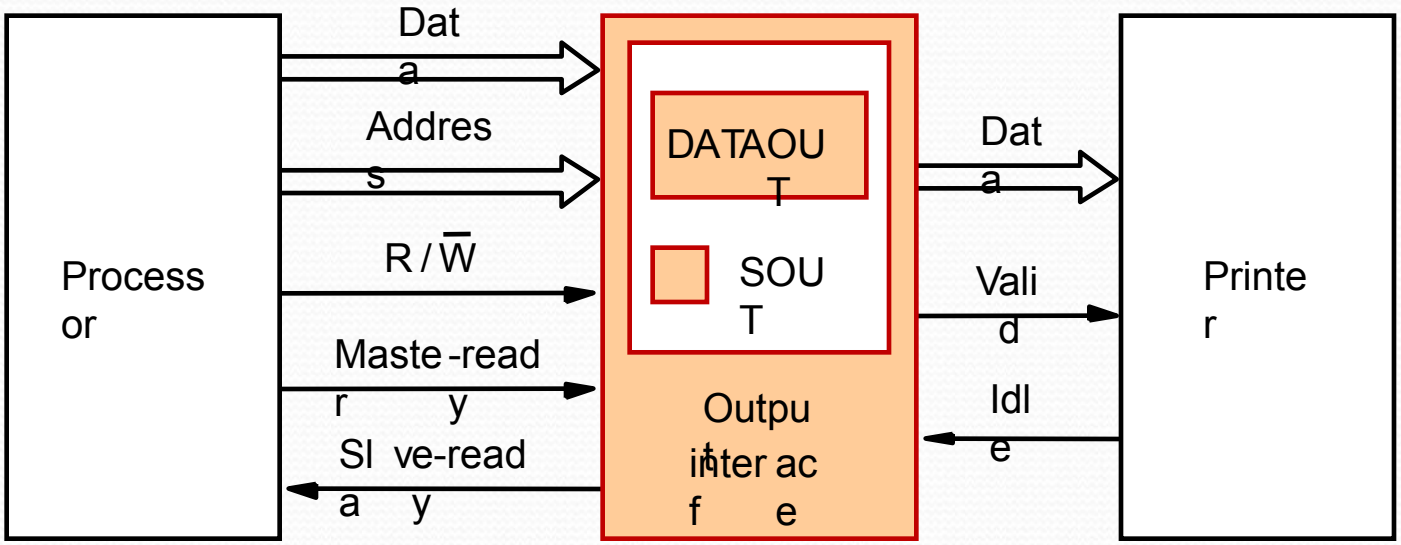
# Input Interface Circuit




- *Output lines of DATAIN are are connected to the data lines of the bus by means of 3 state drivers*
- *Drivers are turned on when the processor issues a read signal and the address selects this register.*

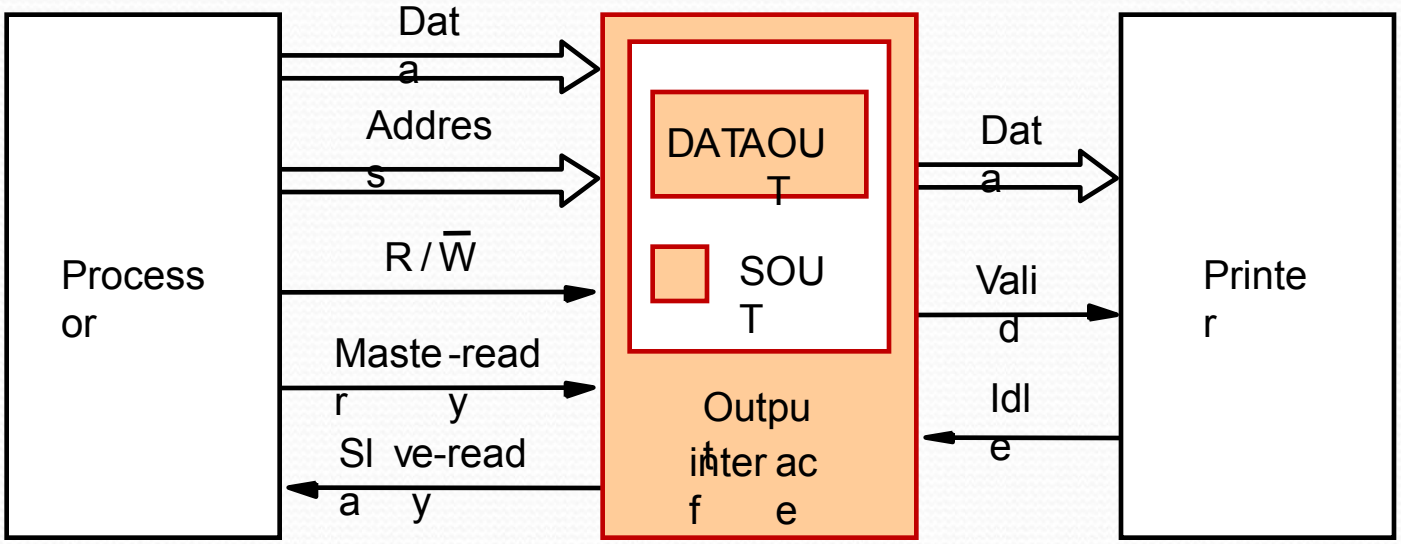
- *SIN signal is generated using a status flag circuit.*
- *It is connected to line $D_0$ of the processor bus using a three-state driver.*
- *Address decoder selects the input interface based on bits $A_1$ through $A_{31}$.*
- *Bit Ao determines whether the status or data register is to be read, when Master-ready is active.*
- *In response, the processor activates the Slave-ready signal, when either the Read-status or Read-data is equal to 1, which depends on line $A_0$.*

# Parallel port (contd..)



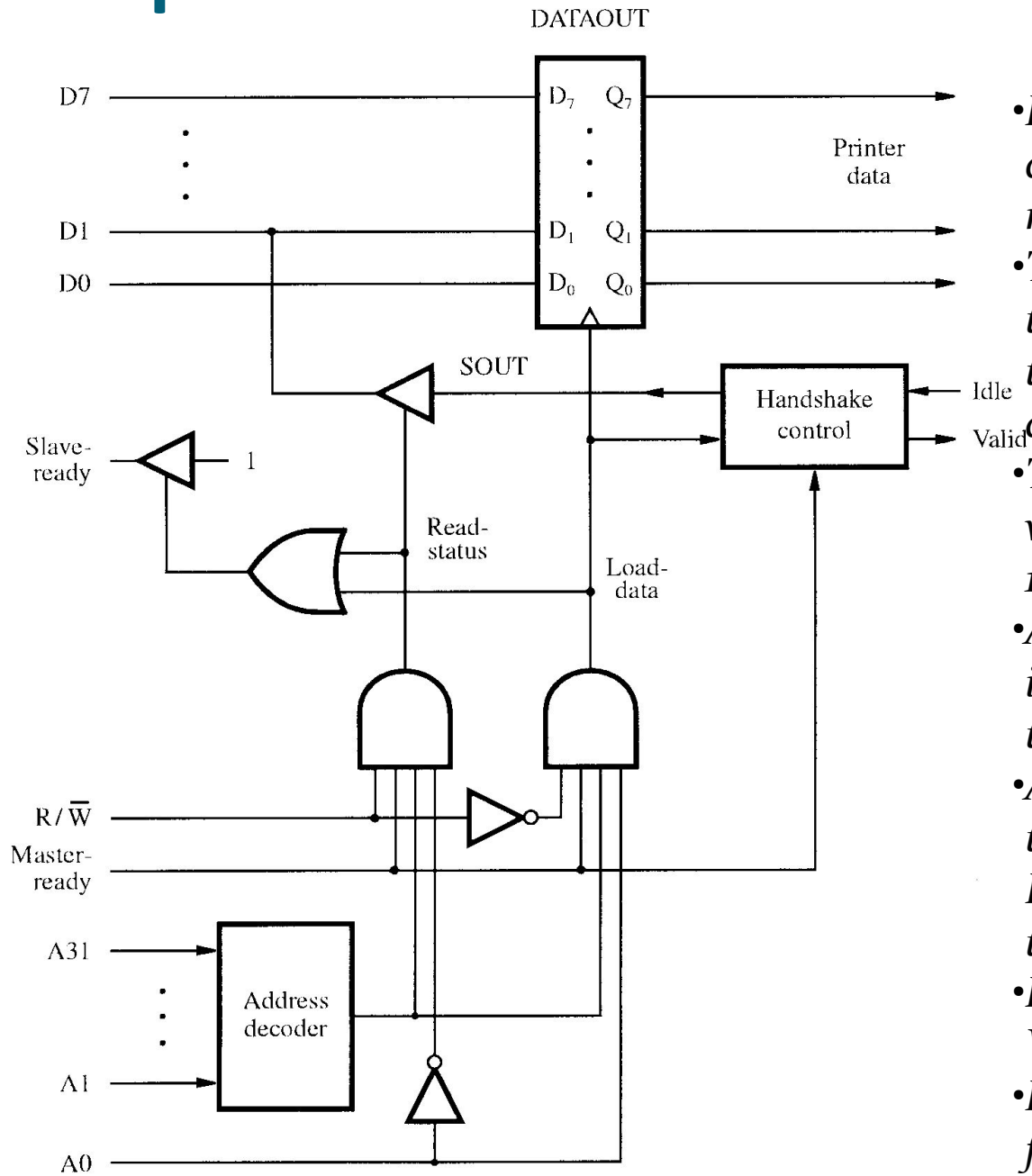

- *Printer is connected to a processor using a parallel port.*
- *Processor is 32 bits, uses memory-mapped I/O and asynchronous bus protocol.*
- *On the processor side:*
  - *Data lines.*
  - *Address lines*
  - *Control or R/W line.*
  - *Master-ready signal and*
  - *Slave-ready signal.*

# Parallel port (contd..)

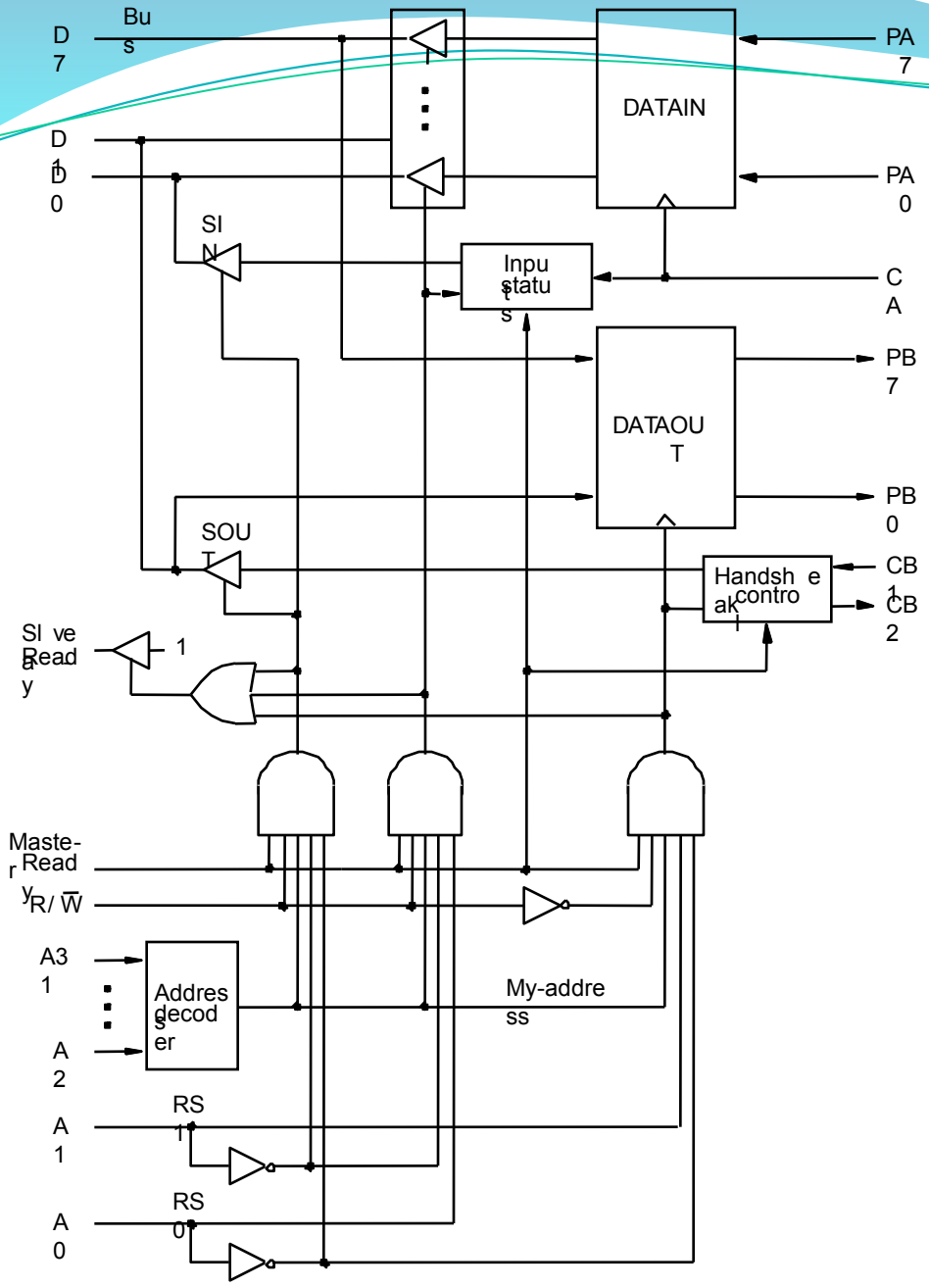



- *On the printer side:*
  - *Idle signal line which the printer asserts when it is ready to accept a character.*
  - *This causes the SOUT flag to be set to 1.*
  - *Processor places a new character into a DATAOUT register.*
  - *Valid signal, asserted by the interface circuit when it places a new character on the data lines.*

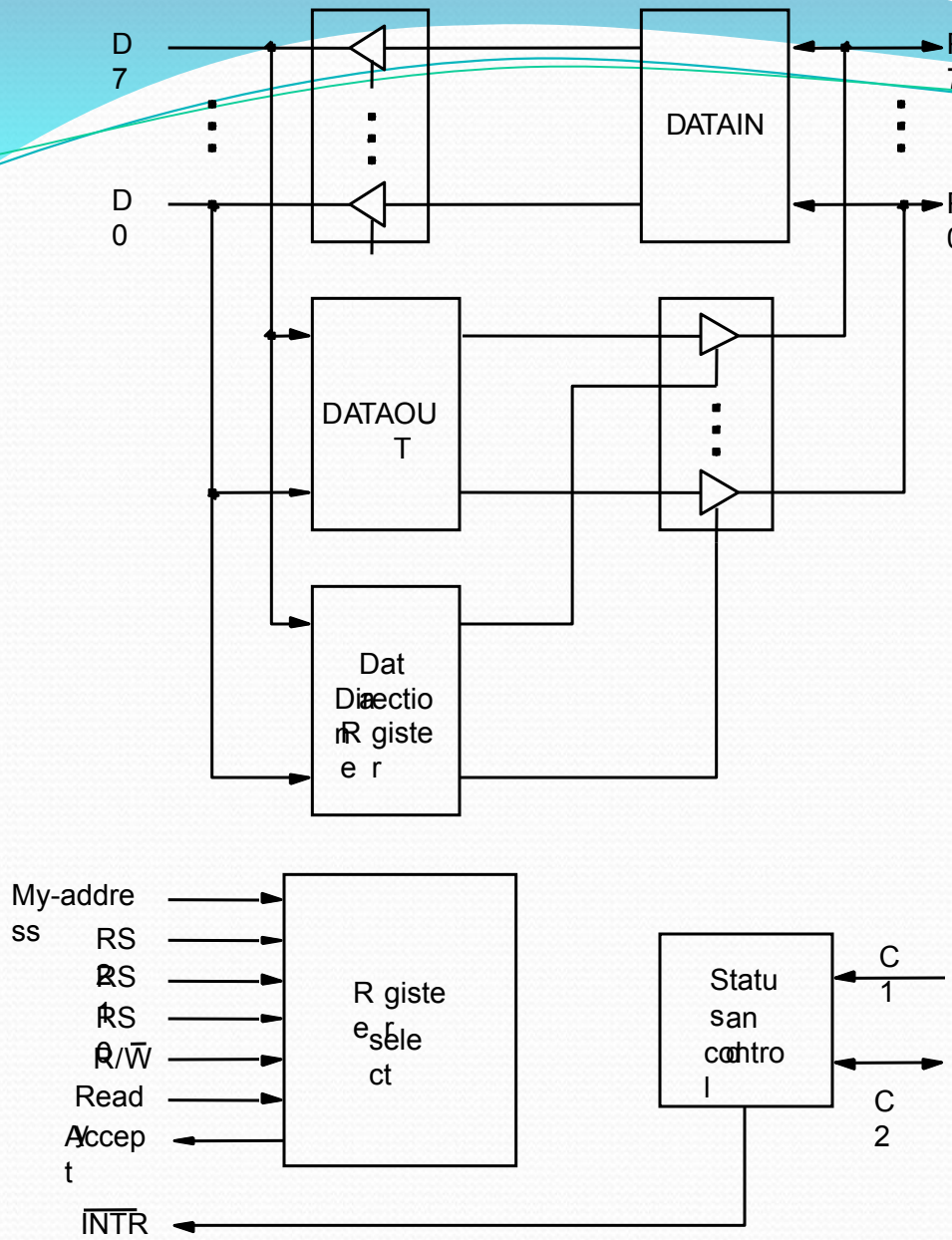
# Output Interface Circuit




- *Data lines of the processor bus are connected to the DATAOUT register of the interface.*
- *The status flag SOUT is connected to the data line D1 using a three-state driver.*
- *The three-state driver is turned on, when the control Read-status line is 1.*
- *Address decoder selects the output interface using address lines A1 through A31.*
- *Address line A0 determines whether the data is to be loaded into the DATAOUT register or status flag is to be read.*
- *If the Load-data line is 1, then the Valid line is set to 1.*
- *If the Idle line is 1, then the status flag SOUT is set to 1.*



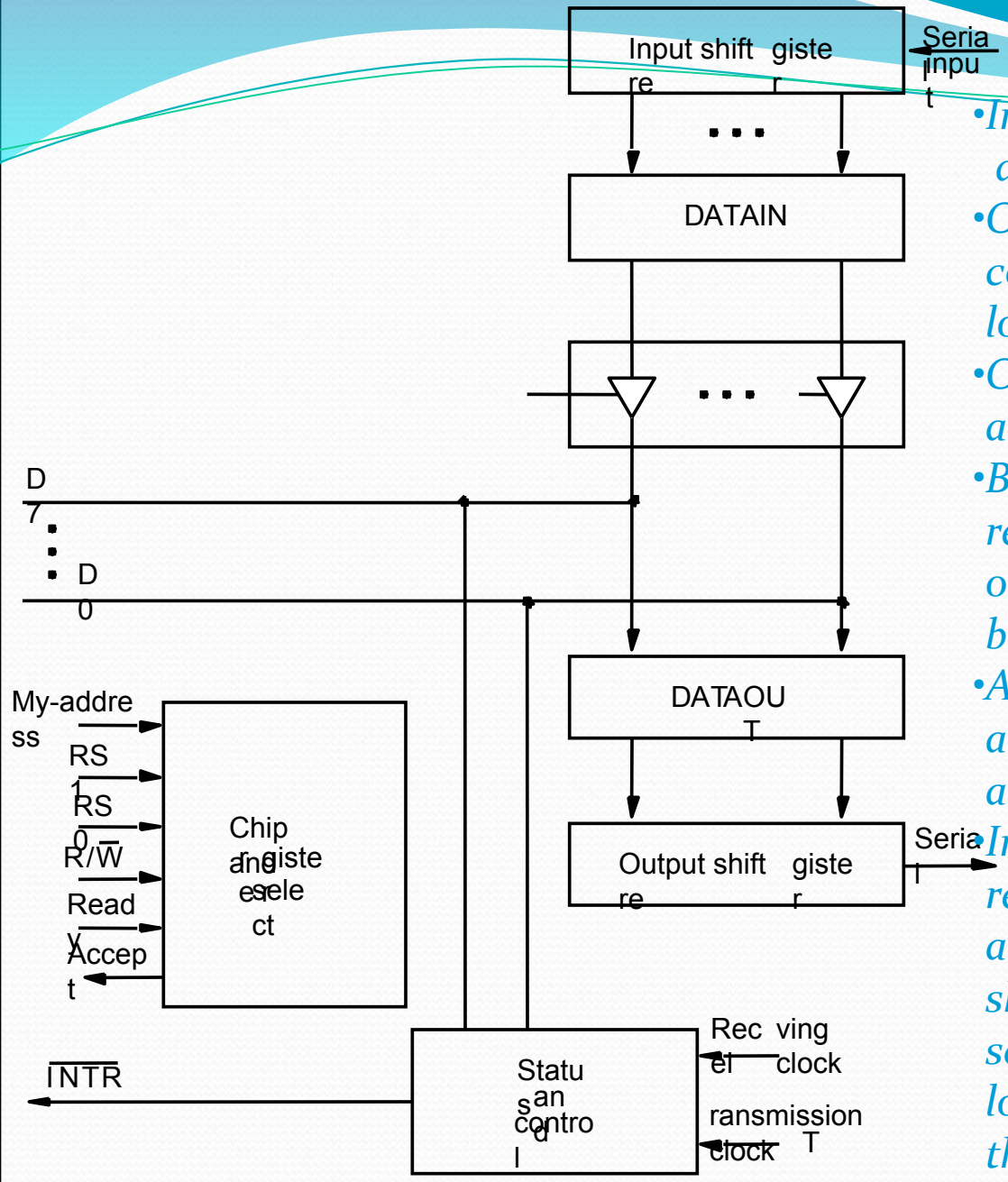

- *Combined I/O interface circuit.*
- *Address bits A2 through A31, that is 30 bits are used to select the overall interface.*
- *Address bits A1 through A0, that is, 2 bits select one of the three registers, namely, DATAIN, DATAOUT, and the status register.*
- *Status register contains the flags SIN and SOUT in bits 0 and 1.*
- *Data lines PA0 through PA7 connect the input device to the DATAIN register.*
- *DATAOUT register connects the data lines on the processor bus to lines PB0 through PB7 which connect to the output device.*
- *Separate input and output data lines for connection to an I/O device.*




- *Data lines to I/O device are bidirectional.*
- *Data lines P7 through Po can be used for both input, and output.*
- *In fact, some lines can be used for input & some for output depending on the pattern in the Data Direction Register (DDR).*
- *Processor places an 8-bit pattern into a DDR.*
- *If a given bit position in the DDR is 1, the corresponding data line acts as an output line, otherwise it acts as an input line.*
- *C1 and C2 control the interaction between the interface circuit and the I/O devices.*
- *Ready and Accept lines are the handshake control lines on the processor bus side, and are connected to Master-ready & Slave-ready.*
- *Input signal My-address is connected to the output of an address decoder.*
- *Three register select lines that allow up to 8 registers to be selected.*

# Serial port

- Serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
- Serial port communicates in a bit-serial fashion on the device side and bit parallel fashion on the bus side.
  - Transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.




- *Input shift register accepts input one bit at a time from the I/O device.*
- *Once all the 8 bits are received, the contents of the input shift register are loaded in parallel into DATAIN register.*
- *Output data in the DATAOUT register are loaded into the output shift register.*
- *Bits are shifted out of the output shift register and sent out to the I/O device one bit at a time.*
- *As soon as data from the input shift reg. are loaded into DATAIN, it can start accepting another 8 bits of data.*
- *Input shift register and DATAIN registers are both used at input so that the input shift register can start receiving another set of 8 bits from the input device after loading the contents to DATAIN, before the processor reads the contents of DATAIN. This is called as double-buffering.*

# Serial port (contd..)

- Serial interfaces require fewer wires, and hence serial transmission is convenient for connecting devices that are physically distant from the computer.
- Speed of transmission of the data over a serial interface is known as the "bit rate".
  - Bit rate depends on the nature of the devices connected.
- In order to accommodate devices with a range of speeds, a serial interface must be able to use a range of clock speeds.
- Several standard serial interfaces have been developed:
  - Universal Asynchronous Receiver Transmitter (UART) for low-speed serial devices.
  - RS-232-C for connection to communication links.