Military Institute of Science and Technology
Department of Computer Science and Engineering

Course Introduction

Course Name: Computer Interfacing

Course Code: CSE-405

Credit Hour: 3.00

Course Instructors

- **Assoc. Prof Dr. Nusrat Sharmin**
  Dept. of CSE, MIST

- **Lecturer Fairooz Tasnia**
  Dept. of CSE , MIST

Course Objective

1. To enable the students familiar to interface external components (peripherals, sensors, PPIs -Programmable Peripheral Interface, PICs - Programmable Interrupt Controller etc.) with computer systems.

2. To enhance the knowledge on basic working principle and different applications of basic microcomputer and microcontroller.

3. To enable the students capable of designing and constructing simple control system incorporating input/output to and from external devices.

Course Outcome

CO1: Classify, identify and analyse that how the interface different types of external components work and communicate (Peripherals, sensors, PPIs, PICs etc.) with computer system

CO2: Apply and implement the external components in real life application and improve the results based on statistical analysis

CO3: Analyze and evaluate abstract problems and apply hardware and software components to address the problem.

Reference Books

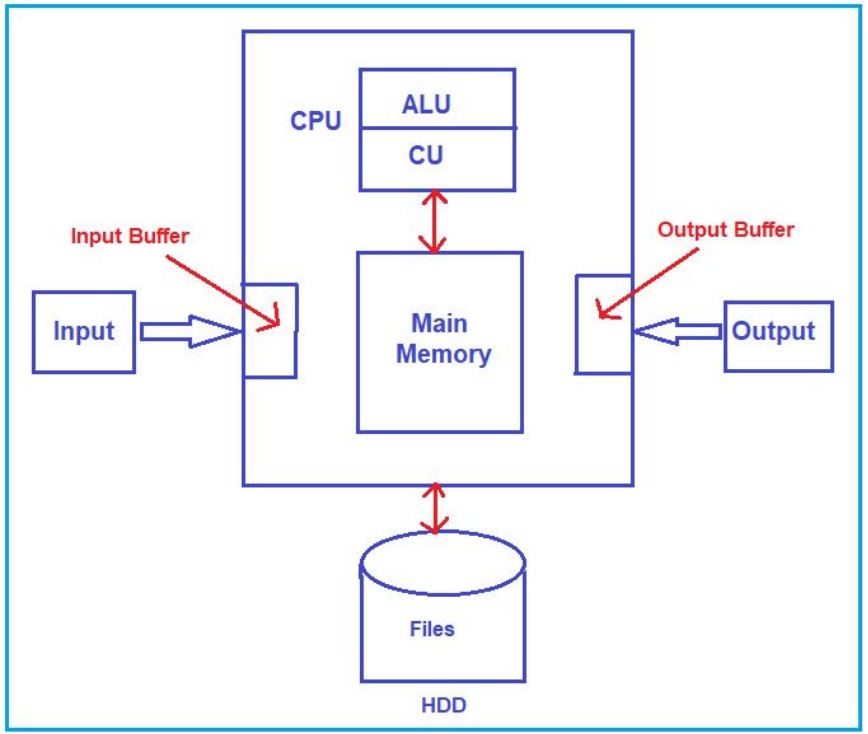
1. Microprocessors and Interfacing (2nd Edition) - Douglas V Hall; McGraw Hill (2005)
2. Computer Peripherals (3rd Edition) - Cook and White; Butterworth-Heinemann (1995)
3. The Intel Microprocessors (8th Edition) - Barry B Brey; Pearson (2008)

Assessment

Component	Percentage
Class Test	20%
Class Participation and Attendance	10%
MID Term	10%
Final Exam	60%

Computer Interfacing

Computer Architecture



Computer Interfacing

Back of computer case and each

Peripheral and Peripheral Devices

- A peripheral is a device that is connected to a host computer but not part of it. It expands the host's capabilities but does not form part of the core computer architecture. It is often, partially or completely, dependent on the host.
- A peripheral device is ancillary device used **to put information into and get information out** of the computer
- Three categories of peripheral devices exist based on their relationship with the computer
  - Input Device
  - Output Device
  - Input/Output Device

Categories and Peripheral Devices

- Input Device- sends data or instruction to the computer
  - Such as mouse, keyboard, scanner, barcode reader, microphone , webcam, game controller,

- Output Device- provides output from computer.
  - Monitor, projector, printer, headphones, speaker

- Input/Output Device- performs both input and output functions
  - Data storage device (disk drive, USB, flash drive, memory card)

What is Computer Interfacing

- An interface is a concept that refers to a point of interaction between objects or components and is applicable at the level of both hardware and software
- It is a kind of interaction between processor and external or peripheral devices
- In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information.
- The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these (Wiki).
- To interface physically, a component or **a mediator** between I/O devices and processor is used which is called **I/O modules**

Can't we connect/interface the I/O device to the processor directly?

NO!!!!!

Why Computer Interfacing

- I/O devices are much slower than the CPU; direct connection would **waste CPU time**

- CPU and I/O devices often use different data format and word lengths.

Digital Interfacing

Book Reference

Microprocessor and Interfacing-
Douglas V. Hall (Chapter 9)

Serial & Parallel Data Transfer

Serial and Parallel Data Transfer

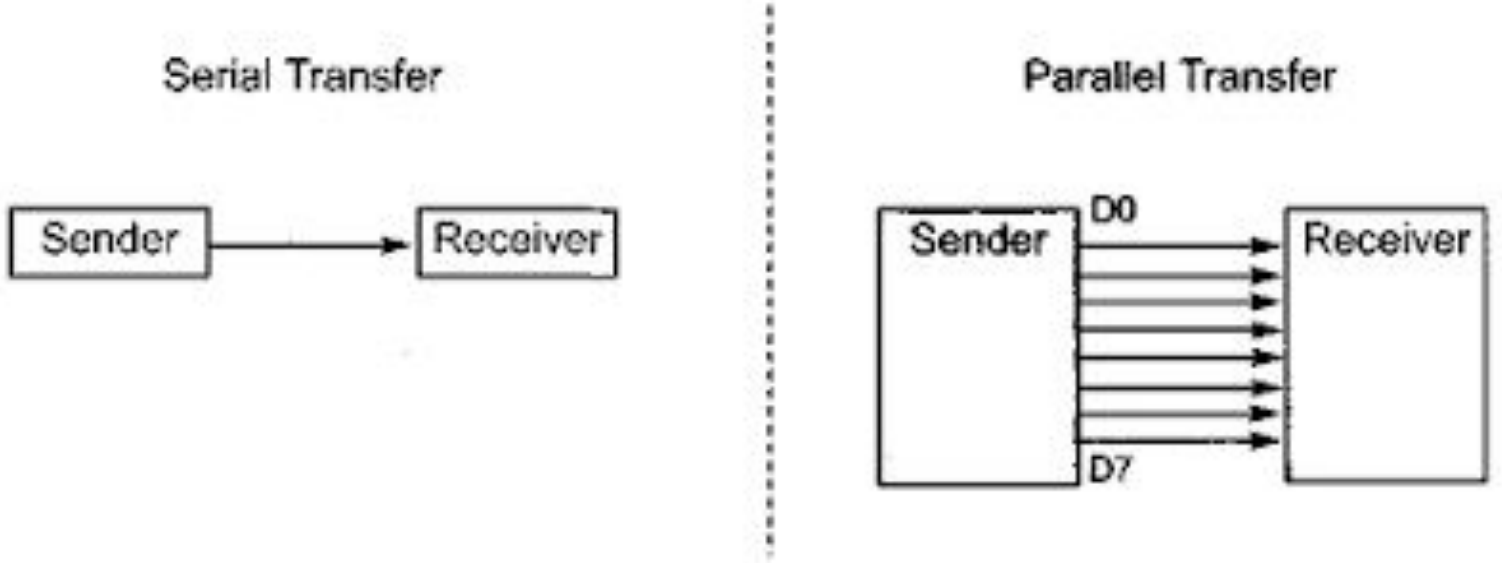
Serial Data Transfer

- One bit at a time over a single wire or channel
- Computer to computer, USB

Parallel Data Transfer

- More that one bit same time using multiple wire
- Usually 8 bit (1 byte) so whole byte can be sent at once
- Computer to a printer

Serial and Parallel Data Transfer



Differences between Serial & Parallel Data Transfer

Serial Transmission	Parallel Transmission
Data flows in two directions, bit by bit	Data flows in multiple directions, 8 bits at a time
Cost is economical	Cost is expensive
Number of bits transferred per clock pulse is 1 bit	Number of bits transferred per clock pulse is 8 bits.
Speed is slower	Speed is faster
Used for long distance communication	Used for short distance communication
Computer to computer	Computer to printer

Parallel Data Transfer

Basic Input and Output Interfaces

- The basic input device is a set of three-state buffers.
- The basic output device is a set of data latches.
- The term IN refers to moving data *from* the I/O device *into* the microprocessor
  Ex: IN AL, P8 [data of port P8 will read to the accumulator reg AL]
- The term OUT refers to moving data *out* of the microprocessor *to* the I/O device.
  Ex: OUT P8, AL [data of AL reg will transfer to the port P8]

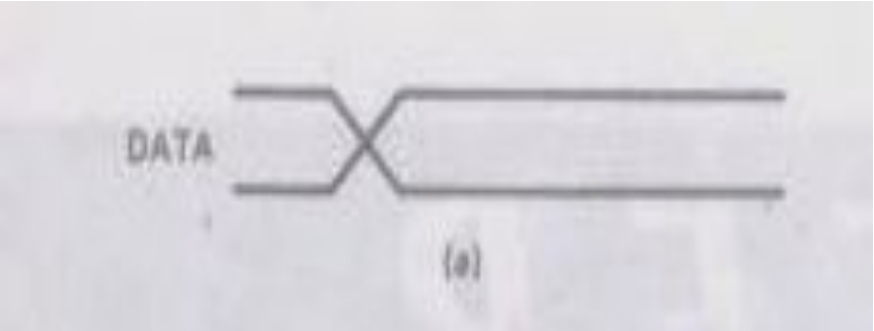
** Every I/O device has a specific 8 bit port address
** IN and OUT instructions are used with **accumulator reg only** (AL, AX, EAX)

Methods of Parallel Data Transfer

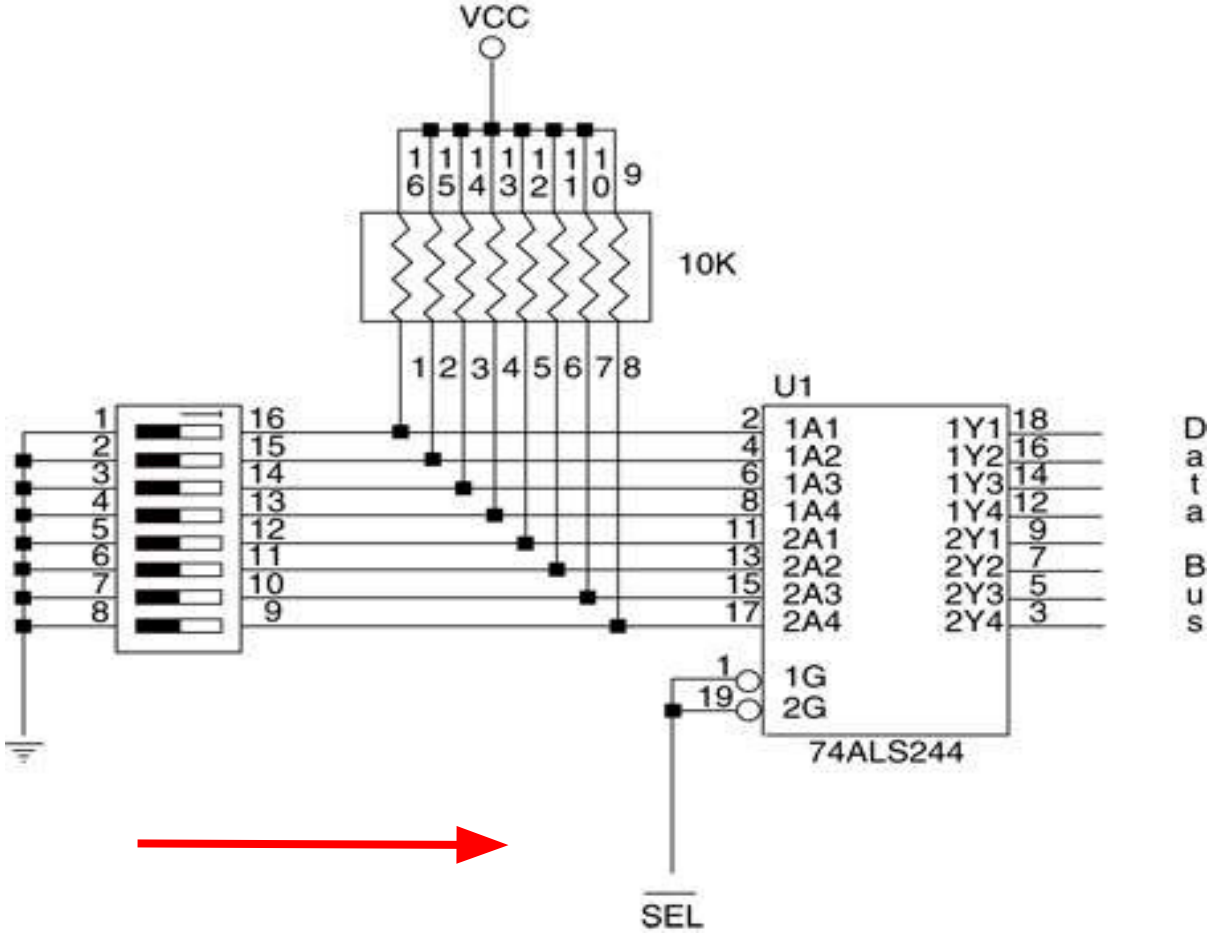
a) Simple I/O
b) Simple Strobe I/O
c) Single Handshake I/O
d) Double Handshake I/O

Simple I/O

- To get digital data from a simple peripheral device **(switch)** into a microprocessor, connect the device to **an input port** line and read the port.
- The data is always ready in the device, so can read it any time
- For output data to a device **(LED**), connect the LED buffer to an **output port** and output the logic data to turn on the light

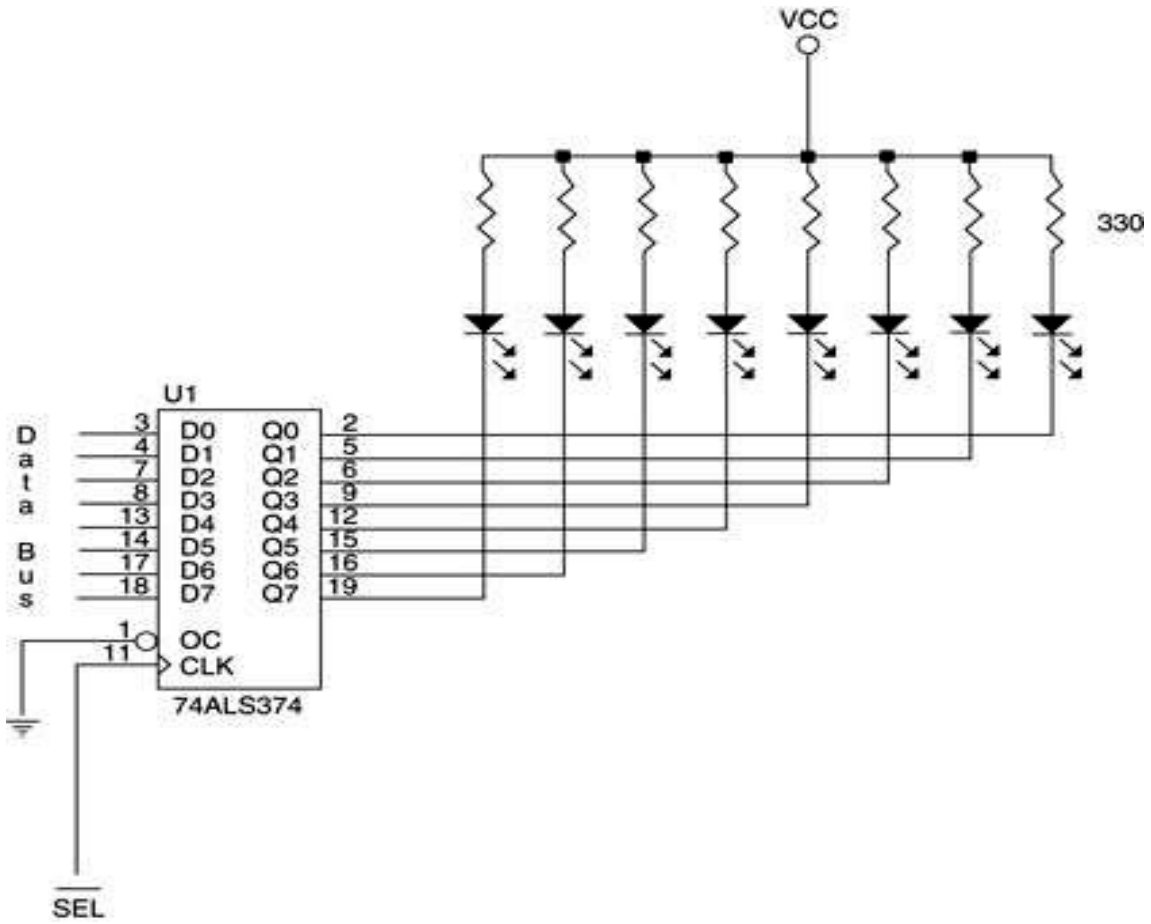


The basic input interface



- Illustrates the connection of eight switches.
- the 74ALS244 is a three-state buffer
- Three State: High, low, High impedance
- controls the application of the switch data to the data bus

The basic Output Interface



*The basic output interface (latches) connected to a set of LED displays.

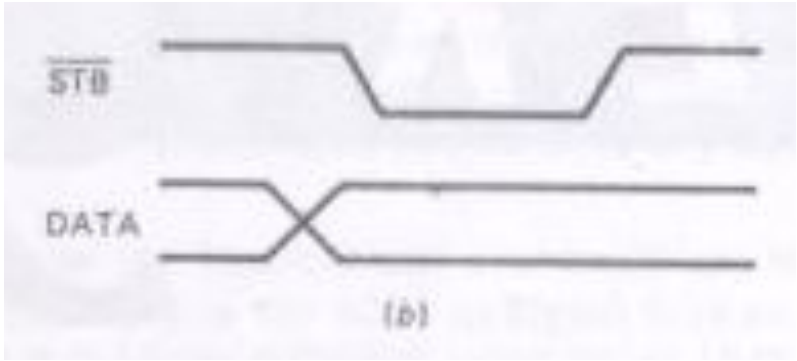
* SEL (active low): for activate the latch

*Receives data from the processor and usually must hold it for some external device.

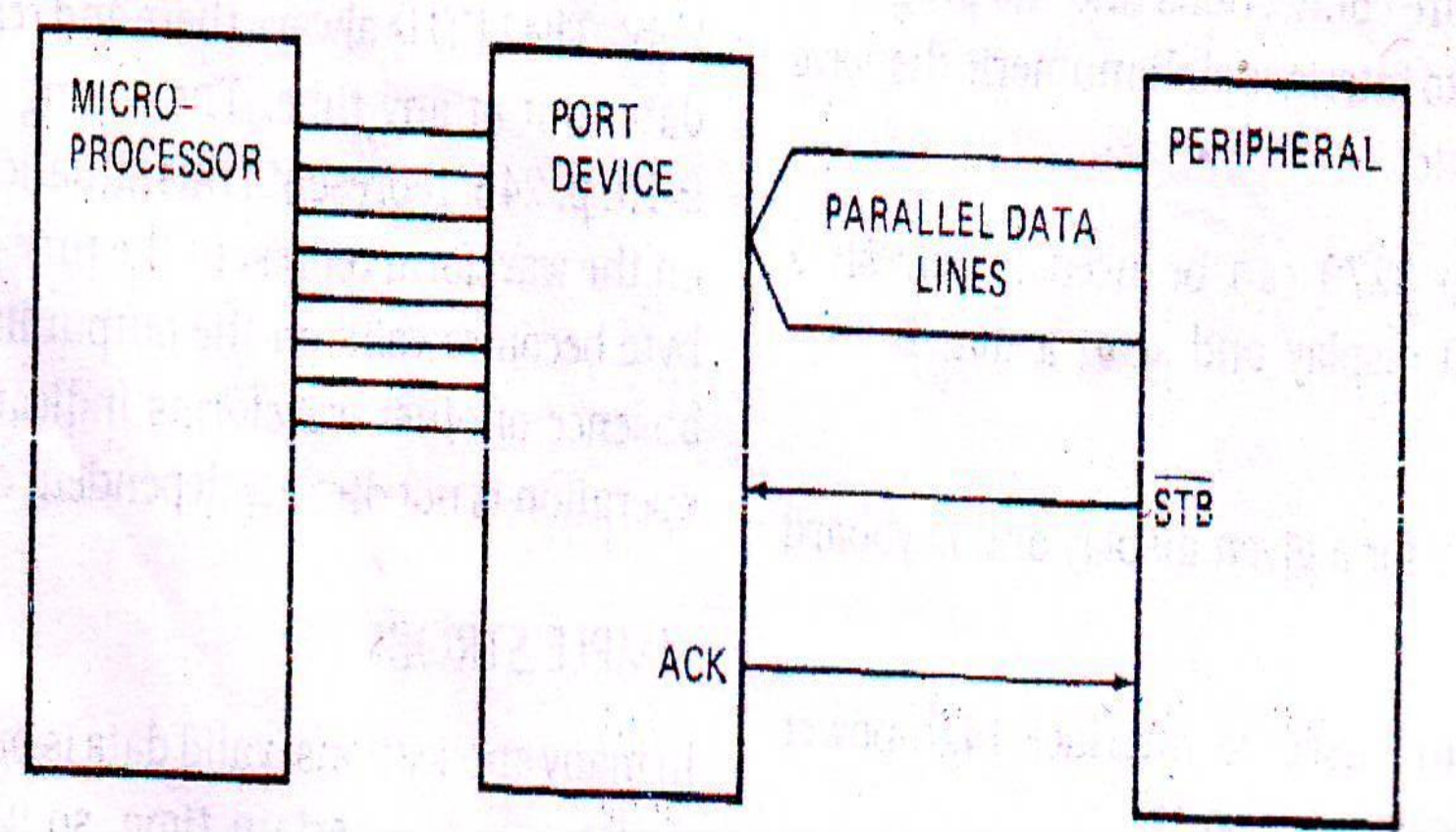
* when output ‘0’, led in forward bias, hence on

Simple Strobe I/O

- On key press, keyboard sends out ASCII code on eight parallel data line
- sends a **strobe signal** on another line indicates that valid data is present on data line
- Strobe = a control signal used to indicate that data is ready or to trigger data transfer
- Ex: Keyboard

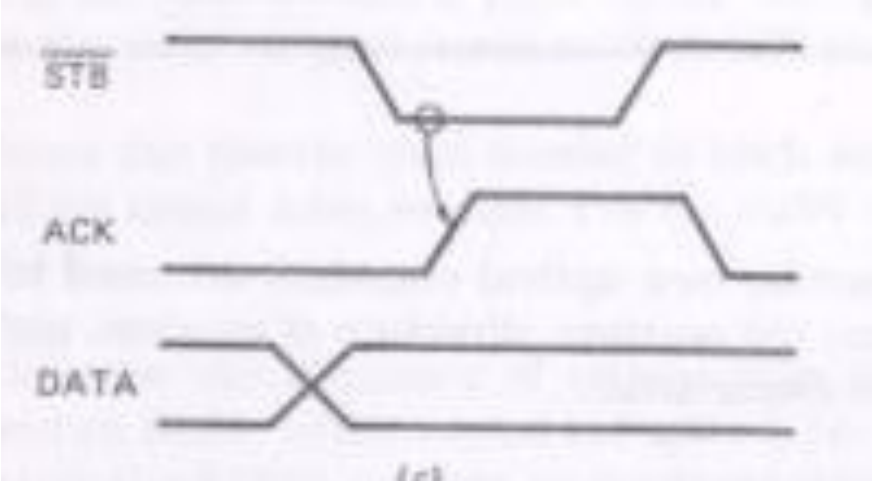


Simple Handshake I/O



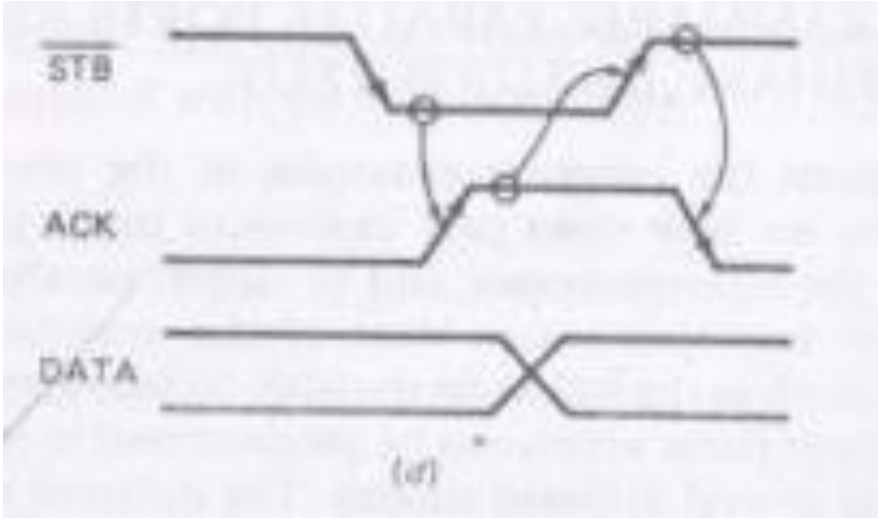
Simple Handshake I/O

- Device sends data and STB signal to microprocessor
- Microprocessor detect STB signal
- Then Microprocessor reads the data and sends acknowledgement signal (ACK) to device
- ACK indicates data has been read and ready for next data
- E.g- Parallel printer



Double-Handshake Data transfer

- Sending device asserts its STB line low to ask : Are you ready?
- Receiving device sends ACK line high to say: I am ready
- Sending device sends the data and raise STB to high, indicates: here some valid data for you.
- Read Data and receiving device drops the ACK low to say: got the data, waiting for the next data.
- E.g.- speech synthesized device



Parallel Printer Interface

- Common printers
  - IBM PC printers
  - Epson dot-matrix printers
  - Panasonic dot-matrix printers
  - Brother laser printer

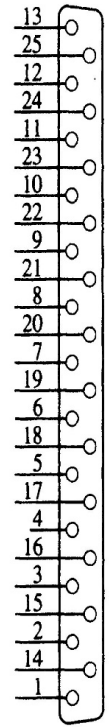
PC printer

Dot- matrix printer

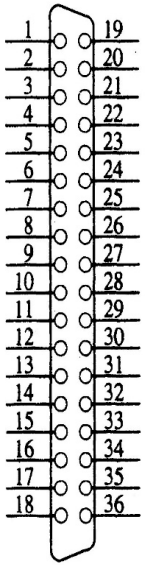
Laser printer

An Example: Parallel Printer Interface

- An example of handshaking is a parallel printer that prints a few hundred characters per second (CPS).
- The processor can send data much faster.
  - a way to slow the microprocessor down to match speeds with the printer must be developed
- Fig 11–5 illustrates typical input and output connections found on a printer.



Connector DB25



Connector CENT36

DB25 Pin number	CENT36 Pin number	Function	DB25 Pin number	CENT36 Pin number	Function
1	1	$\overline{\text{Data Strobe}}$	12	12	Paper empty
2	2	Data 0 (D0)	13	13	Select
3	3	Data 1 (D1)	14	14	Afd
4	4	Data 2 (D2)	15	32	$\overline{\text{Error}}$
5	5	Data 3 (D3)	16	—	$\overline{\text{RESET}}$
6	6	Data 4 (D4)	17	31	Select in
7	7	Data 5 (D5)	18—25	19—30	Ground
8	8	Data 6 (D6)	—	17	Frame ground
9	9	Data 7 (D7)	—	16	Ground
10	10	$\overline{\text{Ack}}$	—	33	Ground
11	11	Busy			

FIGURE 11-5 The DB25 connector found on computers and the Centronics 36-pin connector found on printers for the Centronics parallel printer interface.

Centronics Parallel Interface

- 36 pin connections
- Data and signal line has own individual ground return line – noise reduce
- Major Control Signals
  - $\overline{\text{INIT}}$ - Pin No 31, tell to perform internal initialization
  - $\overline{\text{STROBE}}$ - Pin 1, low: Tells the printer- a character is sent
  - $\overline{\text{ACKNLG}}$- Pin : 10 , Low: data character has been accepted and printer is ready for next character (5 μs pulse)
  - $\overline{\text{BUSY}}$- Pin 11, high- printer busy
  - PE – Pin No: 12 , high: out of paper
  - $\overline{\text{ERROR}}$ – Pin No: 32

Centronics Parallel Interface

Figure 9-11

SIGNAL PIN NO.	RETURN PIN NO.	SIGNAL	DIRECTION	DESCRIPTION
1	19	$\overline{\text{STROBE}}$	IN	STROBE pulse to read data in. Pulse width must be more than 0.5 μs at receiving terminal. The signal level is normally "high"; read-in of data is performed at the "low" level of this signal.
2	20	DATA 1	IN	These signals represent information of the 1st to 8th bits of parallel data respectively. Each signal is at "high" level when data is logical "1" and "low" when logical "0."
3	21	DATA 2	IN	
4	22	DATA 3	IN	
5	23	DATA 4	IN	
6	24	DATA 5	IN	
7	25	DATA 6	IN	
8	26	DATA 7	IN	
9	27	DATA 8	IN	
10	28	$\overline{\text{ACKNLG}}$	OUT	Approximately 5 μs pulse; "low" indicates that data has been received and the printer is ready to accept other data.
11	29	BUSY	OUT	A "high" signal indicates that the printer cannot receive data. The signal becomes "high" in the following cases: 1. During data entry. 2. During printing operation. 3. In "offline" state. 4. During printer error status.
12	30	PE	OUT	A "high" signal indicates that the printer is out of paper.
13	—	SLCT	OUT	This signal indicates that the printer is in the selected state.
14	—	$\overline{\text{AUTO FEED XT}}$	IN	With this signal being at "low" level, the paper is automatically fed one line after printing. (The signal level can be fixed to "low" with DIP SW pin 2-3 provided on the control circuit board.)
15	—	NC		Not used.
16	—	OV		Logic GND level.
17	—	CHASIS-GND	—	Printer chasis GND. In the printer, the chassis GND and the logic GND are isolated from each other.
18	—	NC	—	Not used.
19-30	—	GND	—	"Twisted-Pair Return" signal; GND level.
31	—	$\overline{\text{INIT}}$	IN	When the level of this signal becomes "low" the printer controller is reset to its initial state and the print buffer is cleared. This signal is normally at "high" level, and its pulse width must be more than 50 μs at the receiving terminal.
32		$\overline{\text{ERROR}}$	OUT	The level of this signal becomes "low" when the printer is in "Paper End" state, "Offline" state and "Error" state.
33	—	GND	—	Same as with pin numbers 19 to 30.
34	—	NC	—	Not used.
35				Pulled up to +5 Vdc through 4.7 k-ohms resistance.
36	—	$\overline{\text{SLCT IN}}$	IN	Data entry to the printer is possible only when the level of this signal is "low." (Internal fixing can be carried out with DIP SW 1-8. The condition at the time of shipment is set "low" for this signal.)

Parallel Printer Interface (Contd.)

- 8 bit ASCII data are placed on D_7–D_0
- A pulse is applied to the STB connection.
- STB is a clock pulse used to send data to printer
- As the printer receives data, it places logic 1 on the BUSY pin, indicating it is printing data
- BUSY= 1 indicates the printer is busy
- The Microprocessor repeatedly tests the BUSY pin to decide whether the printer is busy.
  - If the printer is busy, the processor waits
  - if not, the next ASCII character goes to the printer

Timing Waveforms

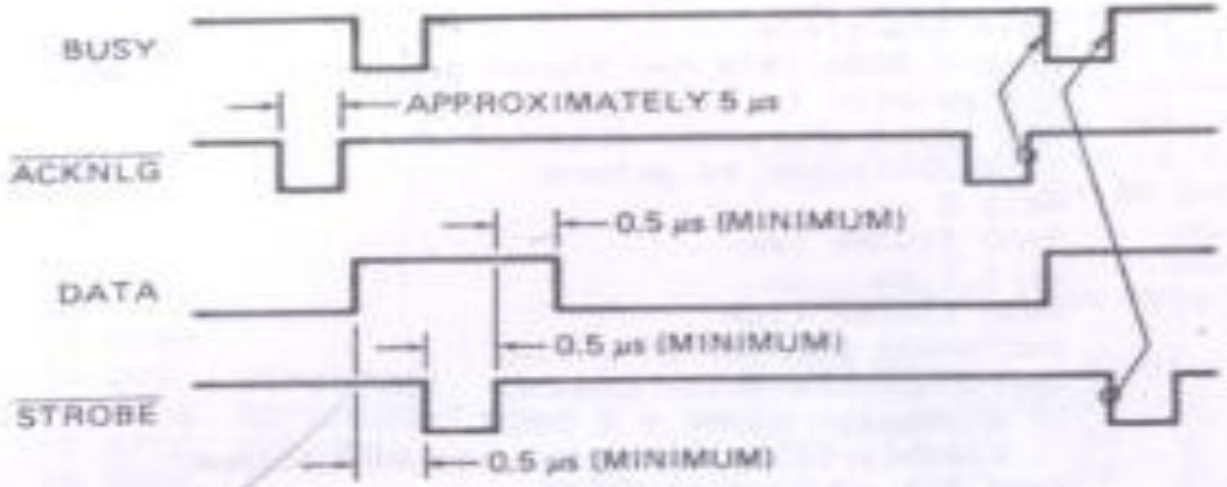


FIGURE 9-13 Timing waveforms for transfer of a data character to a Centronics-type parallel printer such as the IBM-PC or Epson printer. (*IBM Corporation*)

Timing Waveforms

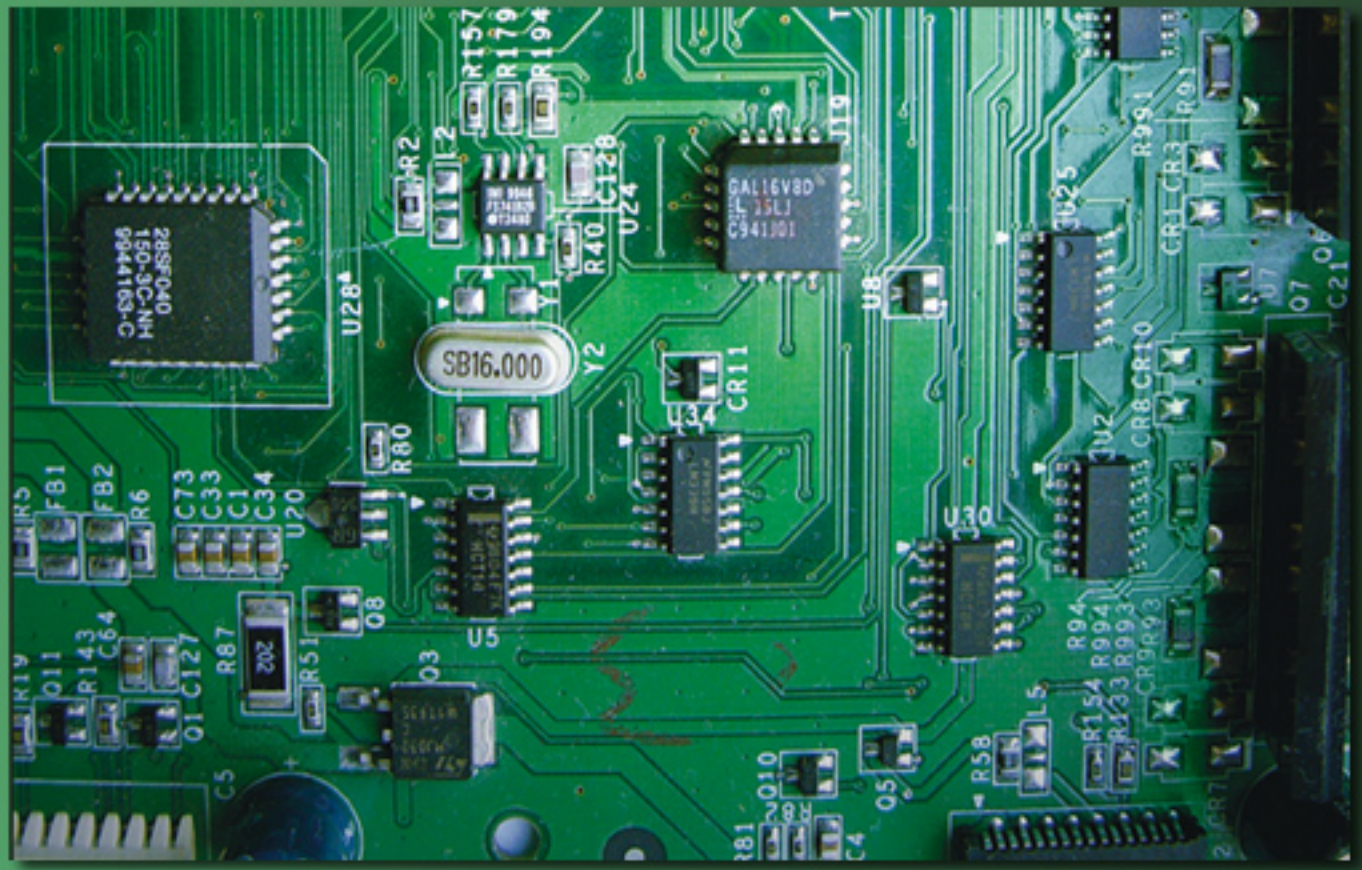
- The BUSY signal is checked to see if the printer is ready
- If BUSY=0 , printer is ready
- An ASCII char is sent on 8 parallel data lines
- After atleast 0.5 microsec STROBE signal is asserted low to indicate to tell the printer that char has been sent
- The STROBE signal going low causes the busy signal going high
- After apprx. 0.5 microsec the STROBE signal can be raised high
- When the printer is ready to receive the next char , it asserts its ACKNLG signal low for 5 microsec
- The rising edge of ACKNLG tells the m/p that it can send the next char
- It also reset the BUSY Pin to LOW

Thank You

The Intel Microprocessors

8086/8088, 80186/80188, 80286, 80386, 80486 Pentium, Pentium Pro Processor, Pentium II, Pentium 4, and Core2 with 64-bit Extensions

Architecture, Programming, and Interfacing



EIGHTH EDITION

Barry B. Brey

PEARSON

Chapter 11: Basic I/O Interface

11–3 THE PROGRAMMABLE PERIPHERAL Interface

- 82C55 **programmable peripheral interface** (**PPI**) is a popular, low-cost interface component found in many applications.
- The PPI has 24 pins for I/O, programmable in groups of 12 pins and groups that operate in three distinct modes of operation.
- 82C55 can interface any I/O device to the microprocessor.

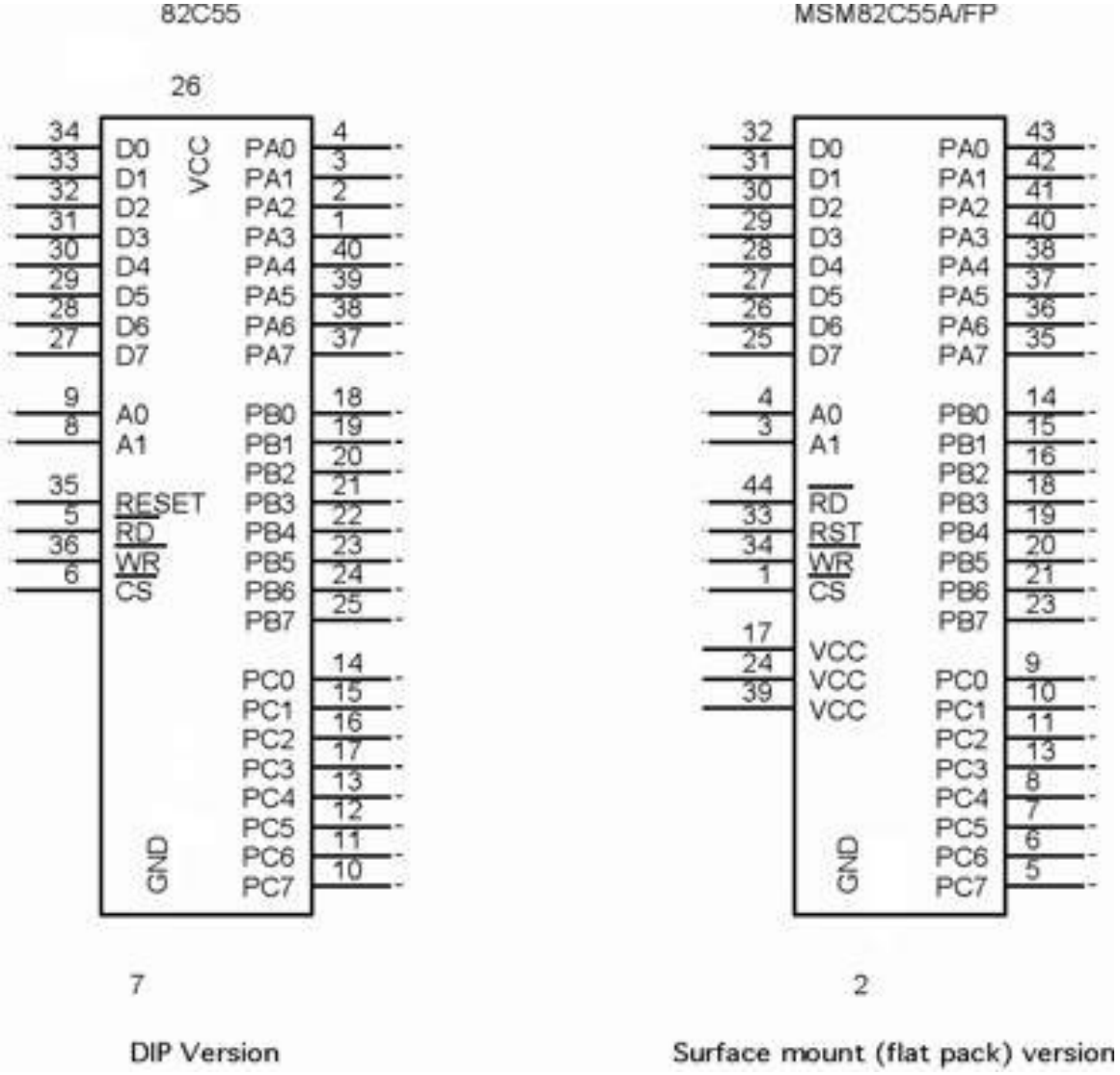
- The 82C55 still finds application even in the latest Core2-based computer system.

- 82C55 is used for interface to the keyboard and parallel printer port in many PCs.
  - found as a function within an interfacing chip set
  - also controls the timer and reads data from the keyboard interface
- The 8255 is programmed in either assembly language or Visual C++ through drivers available with the board.

Basic Description of the 82C55

- Fig 11–18 shows pin-outs of the 82C55
- The three I/O ports (labeled A, B, and C) are programmed as groups.
  - group A connections consist of port A (PA_7–PA_0) and the upper half of port C (PC_7–PC_4)
  - group B consists of port B (PB_7–PB_0) and the lower half of port C (PC_3–PC_0)
- 82C55 is selected by its CS pin for programming and reading/writing to a port.

Figure 11–18 The pin-out of the 82C55 peripheral interface adapter (PPI).



- Register selection is accomplished through the A1 and A0 input pins that select an internal register for programming
- Table 11–2 shows I/O port assignments used for programming and access to the I/O ports.

TABLE 11–2 I/O port assignments for the 82C55.

A_1	A_0	*Function*
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Command register

- In the PC, 82C55s are decoded at I/O ports 60H–63H and also at ports 378H–37BH.
- The 82C55 is a fairly simple device to interface to the microprocessor and program.
- For 82C55 to be read or written, the CS input must be logic 0 and the correct I/O address must be applied to the A_1 and A_0 pins.
- Remaining port address pins are *don't cares*.

- 82C55 has 3 modes:
- MODE 0: Basic I/O operation
- MODE 1: STROBE operation for group B connections, where data are transferred through port B and handshaking signals are provided by port C.
- MODE 2: Bidirectional mode of operation for Group A (can send and receive data at a time)

- Fig 11–19 shows an 82C55 connected to the 80386SX so it functions at 8-bit addresses C0H (port A), C2H (port B), C4H (port C), and C6H (command register).
- All 82C55 pins are direct connections to the 80386SX, except the CS pin. The pin is decoded/selected by a $\overline{74}$ALS138 decoder.
- A RESET to 82C55 sets up all ports as simple input ports using mode 0 operation.
  - initializes the device when the processor is reset

Figure 11–19 The 82C55 interfaced to the 80386SX microprocessor.

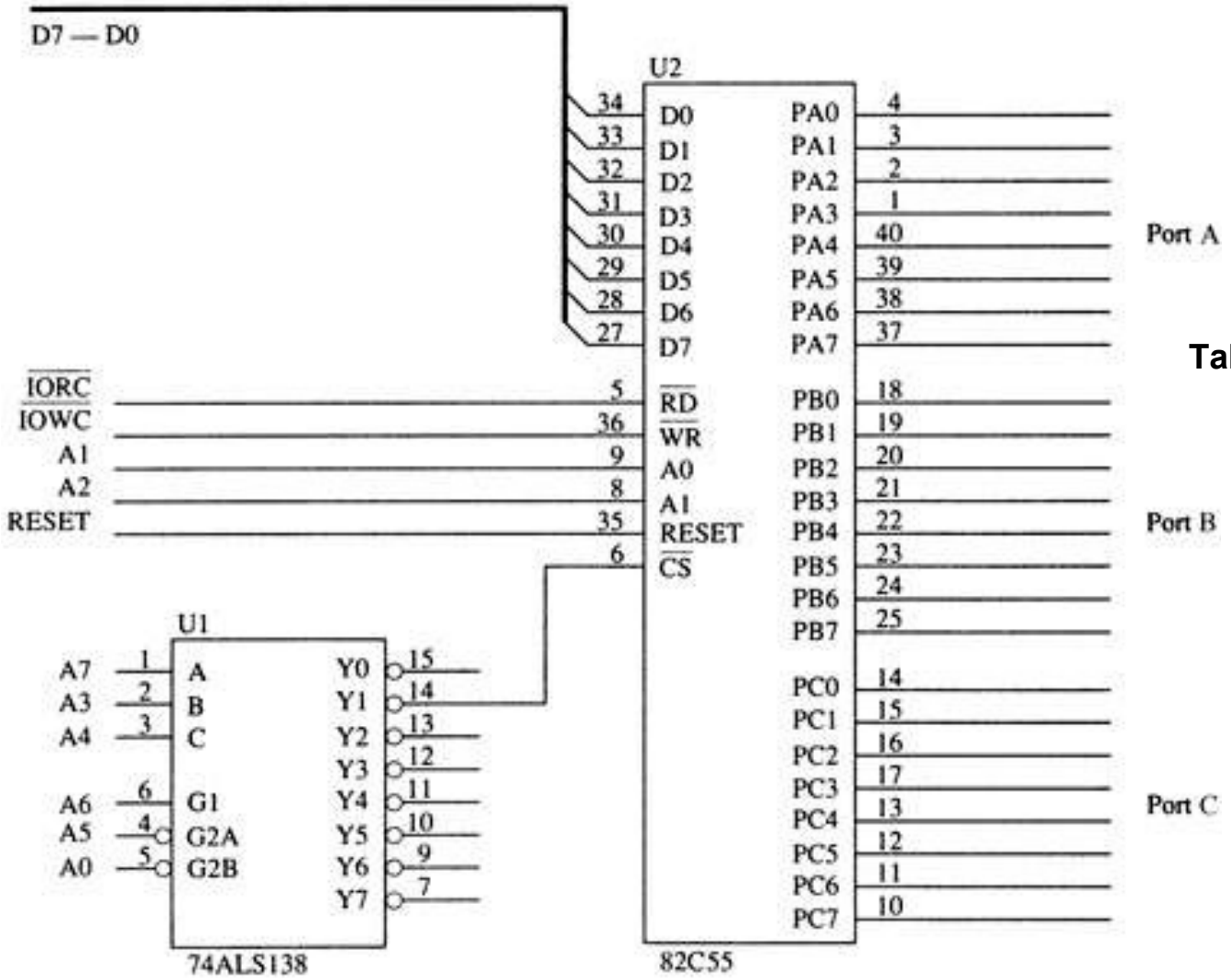


Table 11–2: I/O port assignments

A_1	A_0	*Function*
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Command Register

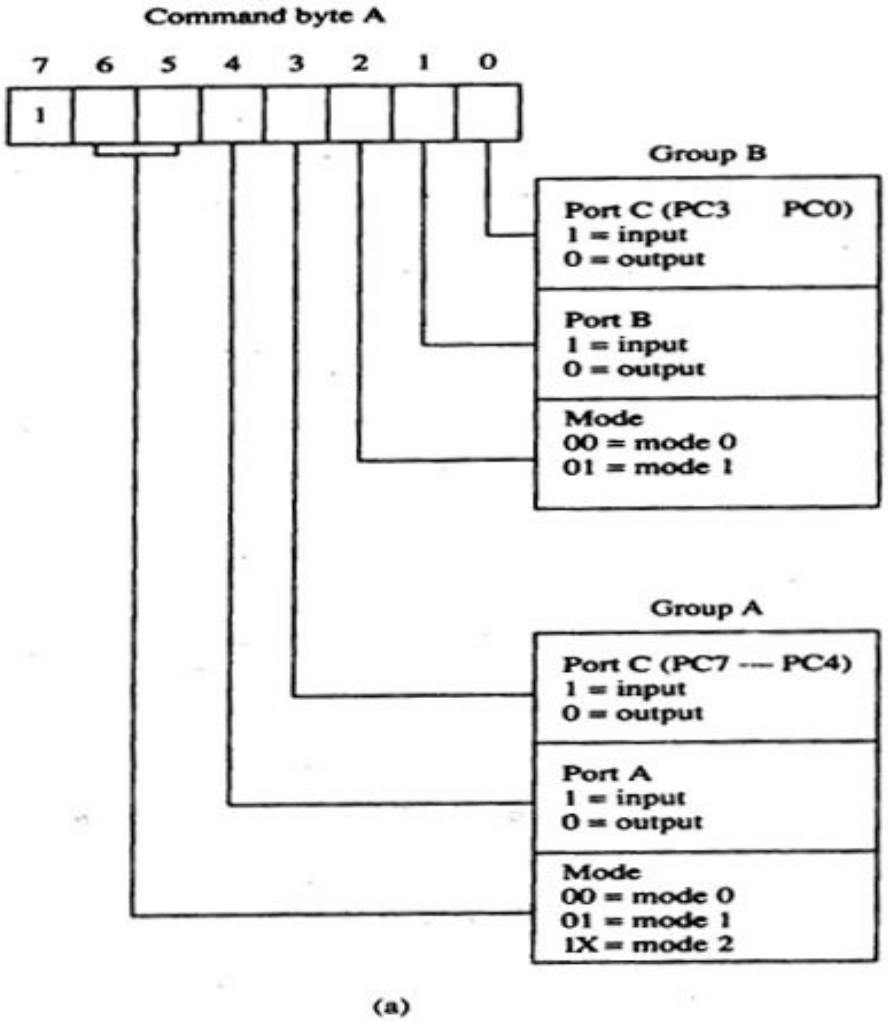
A7	A6	A5	A4	A3	A2	A1	A0	
1	1	0	0	0	0	0	0	C0H
1	1	0	0	0	0	1	0	C2H
1	1	0	0	0	1	0	0	C4H
1	1	0	0	0	1	1	0	C6H

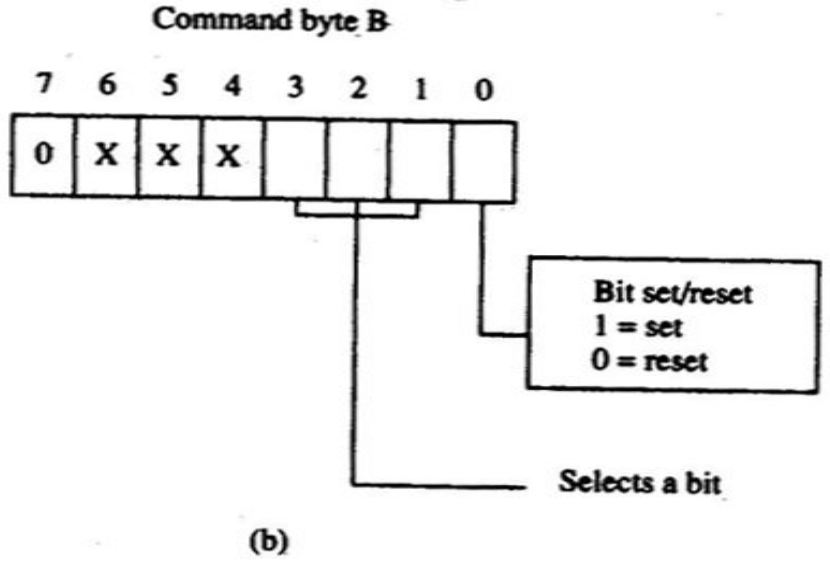
- 82C55 is interfaced to the PC at port addresses 60H–63H for keyboard control.
  - also for controlling the speaker, timer, and other internal devices such as memory expansion
- It is also used for the parallel printer port at I/O ports 378H–37BH.

Programming the 82C55

- 82C55 is programmed through two internal command registers shown in Figure 11–20.
- Bit position 7 selects either command byte A (for 1) or command byte B (for 0).
  - command byte A programs functions of group A and B
  - Command byte B sets (1) or resets (0) bits of port C only if the 82C55 is programmed in **mode 1 or 2**
- Group A and B are programmed as input or output pins.

Figure 11–20 The command byte of the command register in the 82C55. (a) Programs ports A, B, and C. (b) Sets or resets the bit indicated in the select a bit field of port C.





- group B operates in mode 0 or mode 1
- Mode 0 is basic input/output mode that allows the pins of group B to be programmed as simple input and latched output connections
- Mode 1 operation is the strobed operation for group B connections
- Here data are transferred through port B
- handshaking signals are provided by port C

- Group A (port A and the upper part of port C) are programmed as input or output pins.
- Group A can operate in modes 0, 1, and 2.
  - mode 2 operation is a bidirectional mode of operation for port A
- If a 0 is placed in bit position 7 of the command byte, command byte B is selected
- This allows any bit of port C to be set (1) or reset (0), if the 82C55 is operated in either mode 1 or 2.
  - otherwise, this byte is not used for programming

Example: In a smart home automation system, an 82C55 Programmable Peripheral Interface (PPI) is used to manage various input and output devices efficiently. The system connects eight temperature sensors to **Port A to continuously monitor the room's temperature**. To provide manual control over certain appliances, **four switches are connected to the lower half of Port C (PC0–PC3)**. Meanwhile, **Port B is used to drive eight LEDs that indicate statuses such as door security or gas leaks**. **The upper half of Port C (PC4–PC7) controls four relays that switch lights, fans, or other devices on or off**. Since the system needs straightforward data reading and control without complex handshaking, both **Group A and Group B of the 82C55 are programmed in** the basic input/output mode. Design the command byte A to configure the 82C55 accordingly

Example: In a smart home automation system, an 82C55 Programmable Peripheral Interface (PPI) is used to manage various input and output devices efficiently. The system connects eight temperature sensors to **Port A to continuously monitor the room's temperature**. To provide manual control over certain appliances, **four switches are connected to the lower half of Port C (PC0–PC3)**. Meanwhile, **Port B is used to drive eight LEDs that indicate statuses such as door security or gas leaks**. **The upper half of Port C (PC4–PC7) controls four relays that switch lights, fans, or other devices on or off**. Since the system needs straightforward data reading and control without complex handshaking, both **Group A and Group B of the 82C55 are programmed in** the basic input/output mode. Design the command byte A to configure the 82C55 accordingly.

10010001

Mode 0 Operation

- Mode 0 operation causes 82C55 to function:
  - as a buffered input device
  - as a latched output device
- Fig 11–21 shows 82C55 connected to a set of eight seven-segment LED displays.
- These are standard LEDs.

Interfacing to Alphanumeric Displays

- Microprocessor controlled machine needs to display alphabets and number- to give direction or data values to users

- CRT (Cathode Ray Tube)- display large amount of data
- LED and LCD - small amount of data
- LCD – low power, portable device, battery-powered instrument
  - Do not emit their own light- reflect the available lights
-
- LED formats
  - 18 Segment display - numbers, entire alphabet
  - 5 by 7 dot matrix-numbers, entire alphabet
  - 7 Segment displays –numbers, Hexadecimal letters

    least expensive, most commonly used, easiest to interface

- 7 segment data will be given in port A
- Port B will be used to select 1 LED at a time

Figure 11–21 An 8-digit LED display interfaced to the 8088 microprocessor in PORT address 0700H-0703H through an 82C55 PIA .

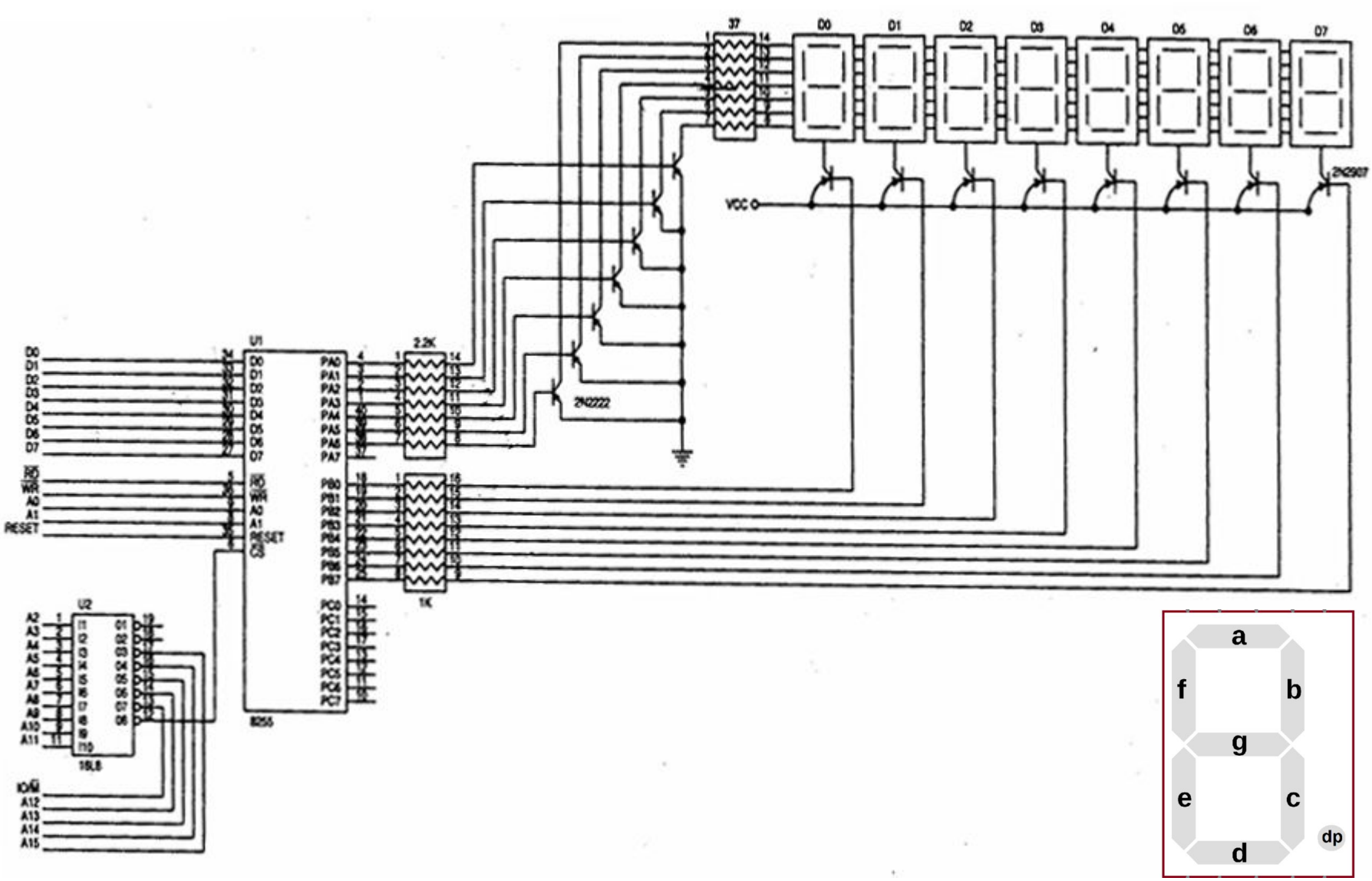
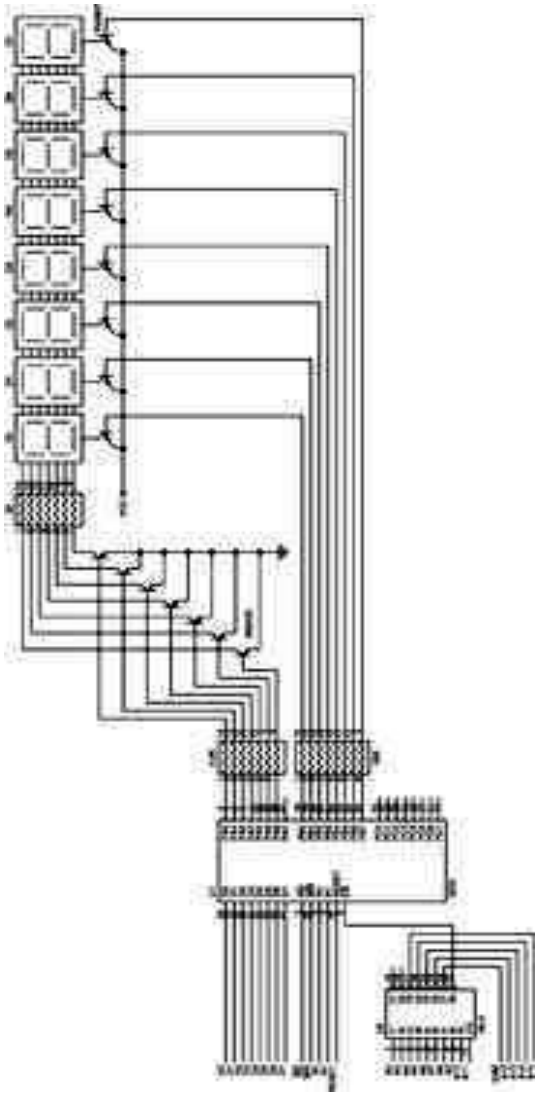


Figure 11–21 An 8-digit LED display interfaced to the 8088 microprocessor through an 82C55 PIA.



- ports A & B are programmed as (mode 0) simple latched output ports
- port A provides segment data inputs port B provides a means of selecting one display position at a time for multiplexing the displays
- the 82C55 functions at I/O port numbers 0700H–0703H

Key Matrix Interface

- Keyboards come in a variety of sizes, from standard 101-key QWERTY keyboards to special keyboards that contain 4 to 16 keys.
- Fig 11–25 is a key matrix with 16 switches interfaced to ports A and B of an 82C55.
  - the switches are formed into a 4 × 4 matrix, but any matrix could be used, such as a 2 × 8
- The keys are organized into four rows and columns: (ROW_0–ROW_3) (COL_0–COL_3)

Figure 11–25 A 4 × 4 keyboard matrix connected to an 8088 microprocessor through the 82C55 PIA.

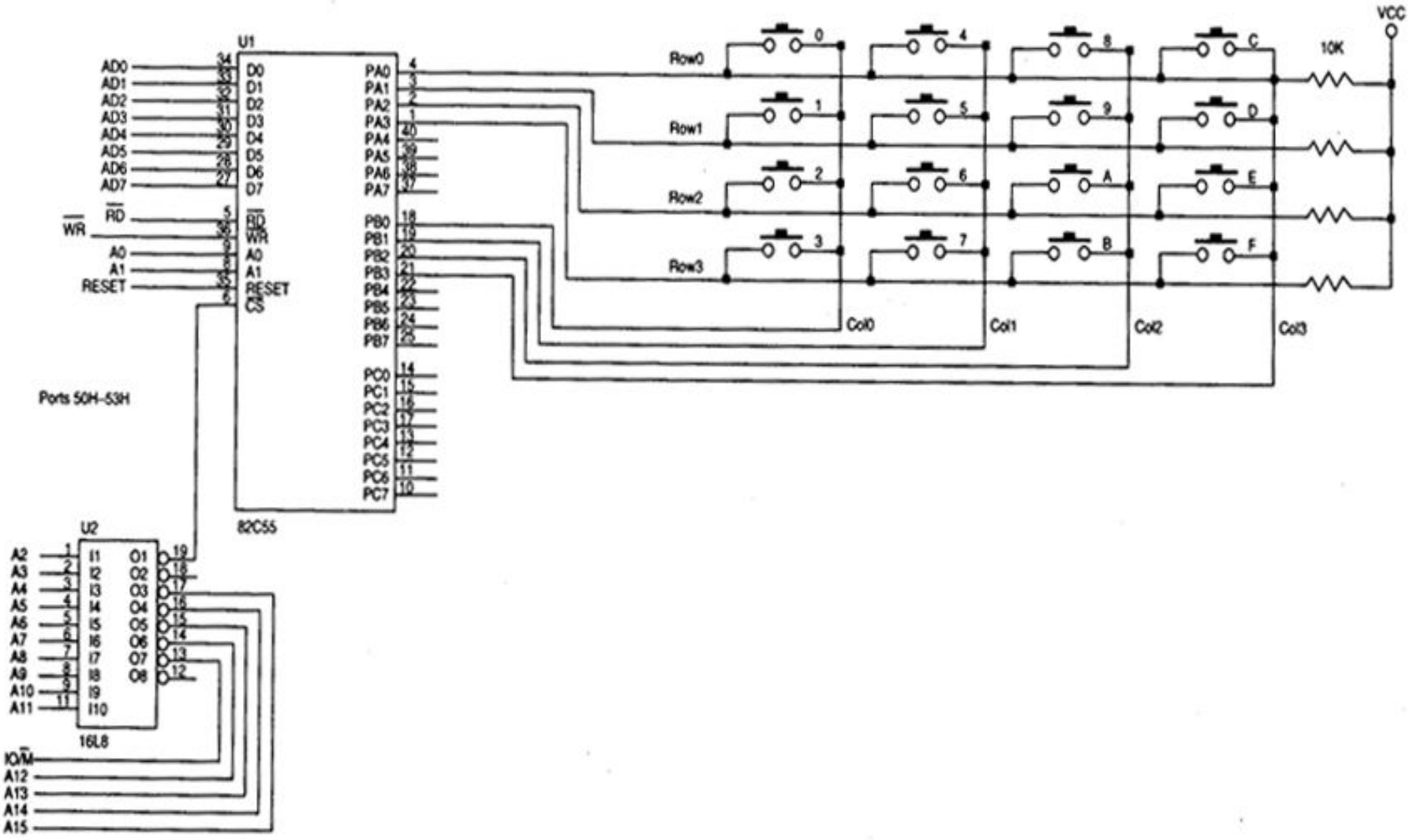
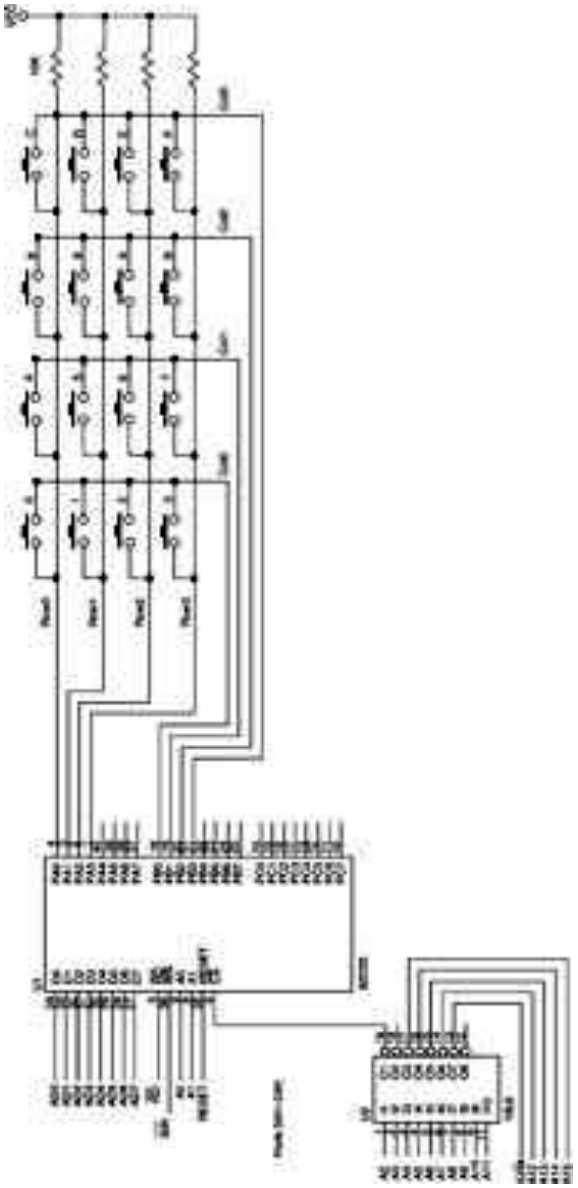
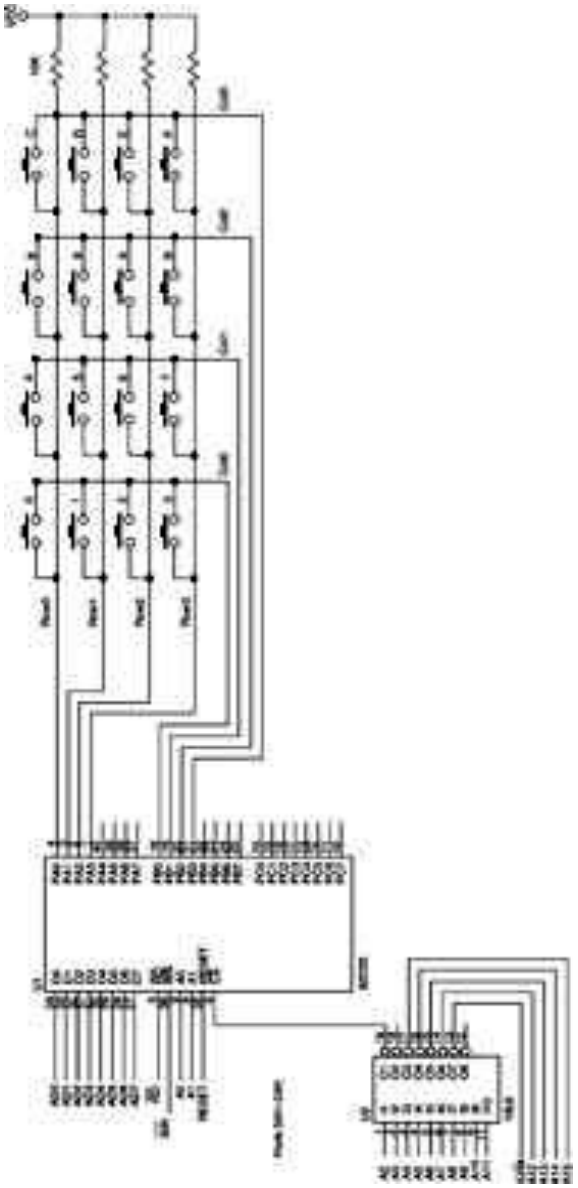


Figure 11–25 A 4 × 4 keyboard matrix connected to an 8088 microprocessor through the 82C55 PIA.



- the 82C55 is decoded at I/O ports 50H–53H for an 8088
- port A is programmed as an input port to read the rows
- port B is programmed as an output port to select a column

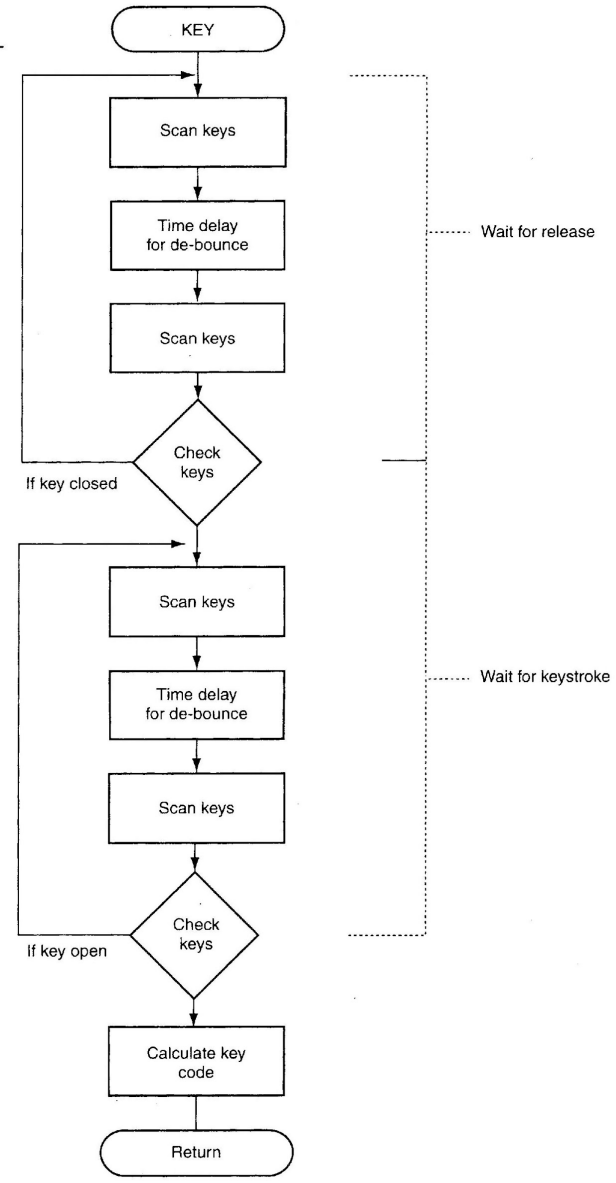
Figure 11–25 A 4 × 4 keyboard matrix connected to an 8088 microprocessor through the 82C55 PIA.



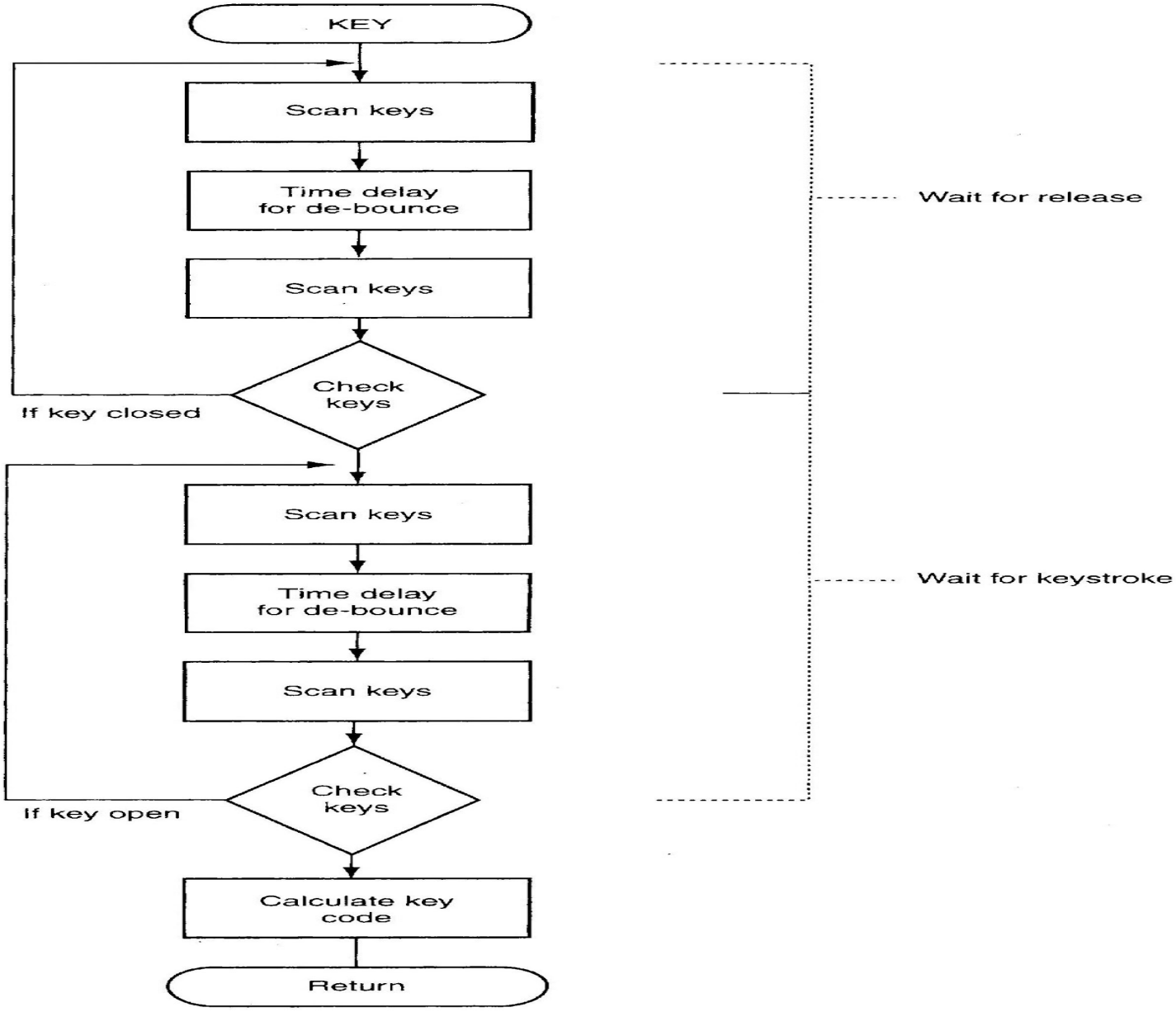
- Each row is connected to 5.0 V through a 10 KΩ pull-up resistor to ensure that the row is pulled high when no push -button switch is closed
- if 1110 is output to port B pins PB3–PB0, the four keys in column 0 are selected.
- With a logic 0 on PB0, the only switches that can be selected onto port A are switches 0–3
- if 1101 is output to port B, switches 4–7 are selected

- Key-switches are connected in a matrix of rows and columns
- Getting meaningful data from a keyboard requires three major steps-
    1. Detect a keypress
    2. Debounce the keypress
    3. Encode the keypress

Figure 11–26 The flowchart of a keyboard-scanning procedure.



- keys must be debounced, normally with a time delay of 10–20 ms
- the software uses a procedure called SCAN to scan the keys and another called DELAY10 to waste 10 ms of time for debouncing
- the main keyboard procedure is called KEY
- the KEY procedure is generic, and can handle any configuration from a 1 × 1 matrix to an 8 × 8 matrix.



Military Institute of Science and Technology
Department of Computer Science and Engineering

Peripheral/Computer Connections & Interrupt

Course Name: Computer Interfacing
Course Code: CSE-405

Book Reference

Computer Peripherals (3rd Edition) - Cook and White; Butterworth-Heinemann (1995)

Introduction

- Computer instructions are taken from memory and executed in the CPU
- It involves requests for data from memory and sending results back to memory.
- Part of the computer's memory consist of Read Only Memory (ROM) where a program is available when the machine is first switched ON.
- This program reads other programs into RAM, known as bootstrapping.
- Special instructions are available for accessing the peripheral interfaces.

Component of a Digital Computer

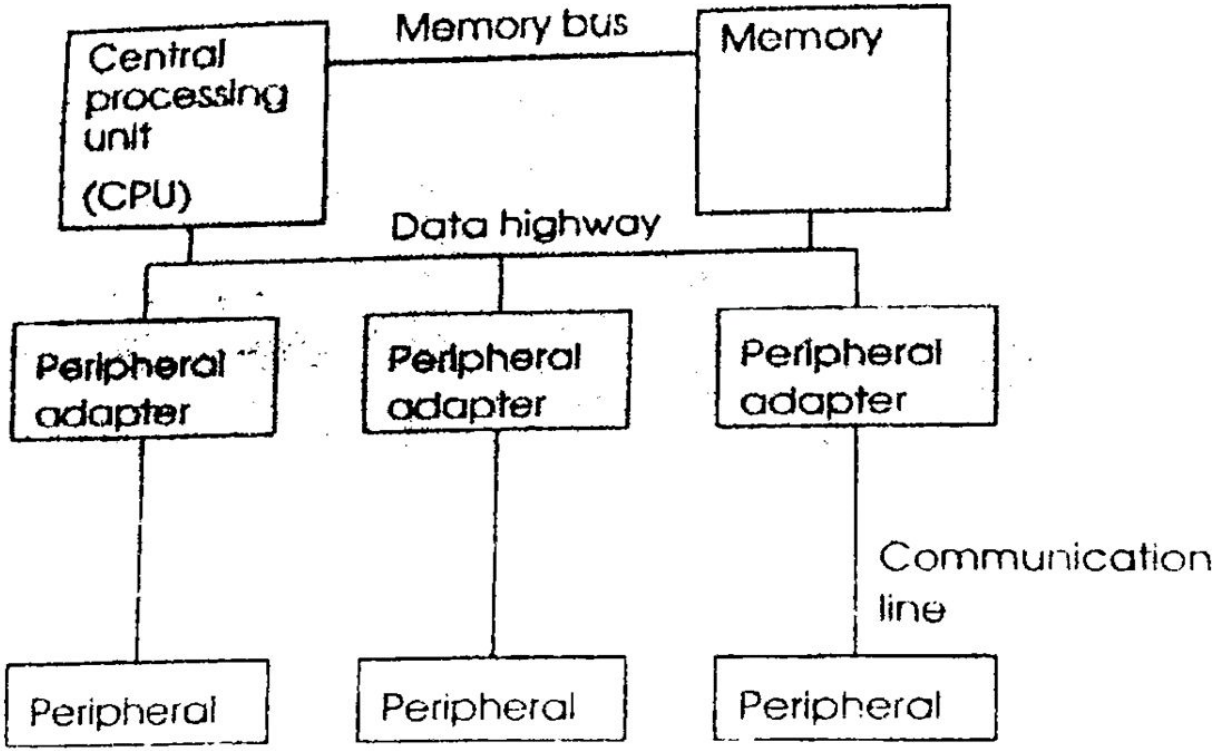


Figure 2.1 *Components of a digital computer*

Data Highway

Data Highway

- Data (including instructions) are moved around the computer on a set of wires forming a data highway (Bus).
  - Data Bus
  - Address Bus
  - Control Bus

Data Highway: Data Bus

- It contains the information or program instructions required by the CPU.
- The speed of operation of a computer depend on the rate at which data can be made available
- A good way to improve the speed is to add more data lines to transfer more data at a time.
- Small systems use 8-bit data bus. They are relatively slow but of low cost.
- 8086 uses 16-bit and 80386 uses 32-bit bus.

Data Highway: Address Bus

- It contains address information.
- The number of lines available for this determines the maximum amount of physical memory that can be addressed.
- If there are n lines , then 2^n locations can be addressed.
- Small system have 16 address line - 2^{16}= 65536 locations
- 8086 has 20 address lines and thus it can address 2^{20}=1048576 (1 Mega bytes) memory locations

Data Highway: Control Bus

- It contains control signals.
- Control signals are required to maintain orderly flow of data along the bus.
- It is necessary to indicate whether to **read** or **write** data during a transfer.
- It also indicates how much data to transfer, timing signals, status lines etc.
- It also indicates whether a transfer is to **memory** or a **peripheral adaptor**

Peripheral Adapter Register

Peripheral Adapter Register

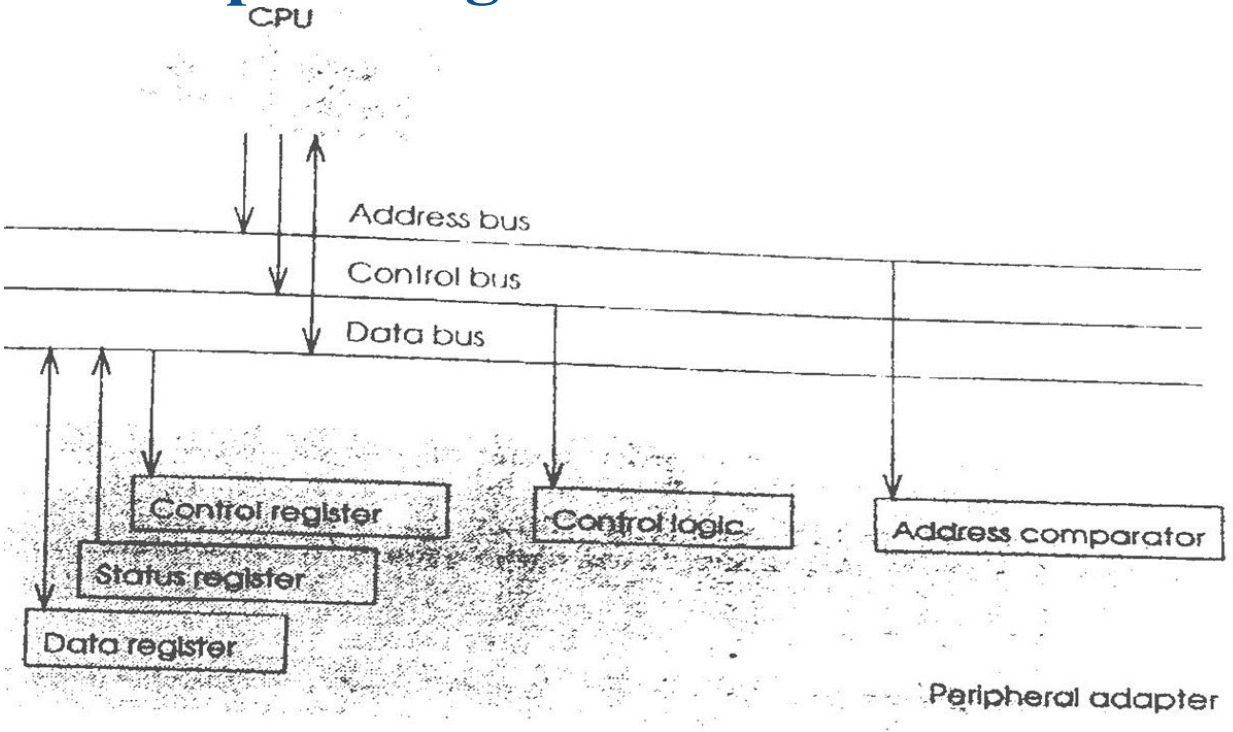


Figure 2.2 *Connection of peripheral adapter to highway*

Peripheral Adapter Register

Peripheral adaptors are directly connected to the buses.

PA contains:

- Address Comparator.
- Control register
- Status register
- Data register
- Control Logic

Peripheral Adapter Register

❖ **Address Comparator**

- It is necessary to distinguish between adapters that are used for different peripherals.
- Address lines are compared with a values allocated to the adaptor
- An adaptor may recognize a single address or a small group of address.
- It compares the address on the address bus with the peripheral adapter's address.

❖ **Control logic**

- If the address on the bus matches the adapter address then the control lines are interpreted by the ***control logic*** to perform the required function.
- It is connected to a data bus typically to read from or write to a ***register*** (may be data register)

Peripheral Adapter Register

- **Control Register**
  - Stores values written to it to control the operation of the adaptor (Receive, transmit, interrupt, mode).
- **Status Register**
  - Can be read by the CPU to determine the status of the device (e.g. w**hether it is ready for use, busy, error, buffer overflow, transmission complete, receive complete etc.**)
  - Each piece of information stored in the registers usually needs only a single bit
  - Several such bits are stored in each register, each bit is often known as a *flag*.
- **Data Register**
  - Used to **hold temporarily a value** to be transferred to or from the peripheral
  - So that it is not necessary to synchronize the computer with the peripheral.

Computer Input/Output Operation: Programmed I/O

Programmed I/0

- The programmed I/O was the simplest type of I/O technique for the exchanges of data or any types of communication **between the processor and the external devices.**
- With programmed I/O, data are exchanged between the processor and the I/O module.
- The instruction set for the CPU typically contains instructions such as
  - **Input data to the CPU from the peripheral**
  - **Output data from the CPU to the peripheral**
  - **Set individual bits in the peripheral adapter's control register**
  - **Test individual bits in the peripheral adapter's status register**

Programmed I/0 : Output

- Output of the data item is straightforward
- The CPU addresses the adapter and tests status register.
- If the relevant bit in the register shows that the device is ready then CPU puts data into the peripherals.
- If the peripheral is not ready, then the program must wait and try again.

Programmed I/0 : Input

- For data input, the status of the input device may be tested and data taken from the input data register if it is available.
- If a number of peripherals are active at the same time, it is necessary to test each one for readiness.
- The technique of testing a number of peripherals in turn is known as ***software polling.***

Programmed I/0 : Example

- Processor instructions for communicating with peripherals are often given obvious mnemonics such as "IN" and "OUT"

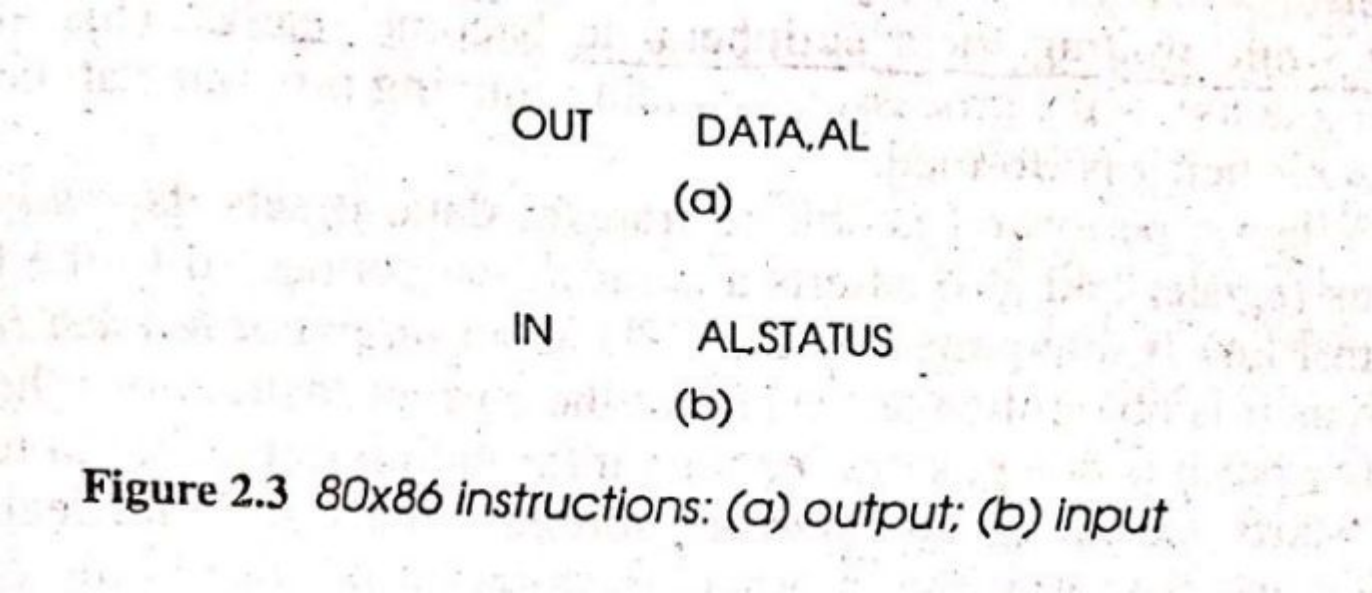
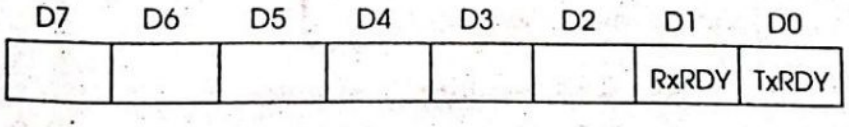


Figure 2.3 *80x86 instructions: (a) output; (b) input*

Programmed I/0 : Example

- Processor instructions for communicating with peripherals are often given obvious mnemonics such as "IN" and "OUT"



(a)

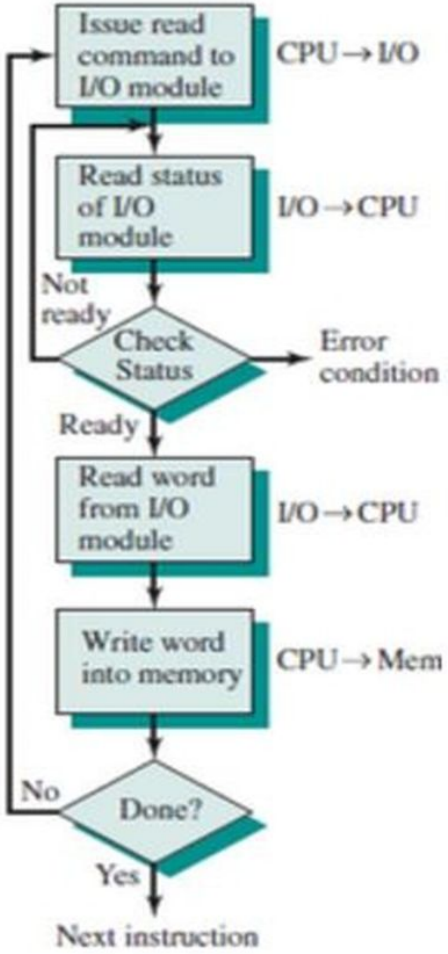
```
STATUS   EQU   0FFH          ; Address of status register
DATA     EQU   0FEH          ; Address of data register

CHECK:   IN    AL,STATUS     ; Get status register
         TEST  AL,1B         ; Test TxRDY bit
         JZ    CHECK         ; Repeat until ready
         MOV   AL,CHARACTER  ; Get the character to send
         OUT   DATA,AL       ; Output to the data register
                             ;  for transmission
```

(b)

Figure 2.4 *Transmitting a character via an 8251; (a) schematic view of the status register; (b) program used*

Programmed I/0 : Data Transfer

- In this case, the I/O device does not have direct access to the memory unit.
- A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory.
- In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer.
- This is a time consuming process since it needlessly keeps the CPU busy.



Computer Input/Output Operation: Interrupt

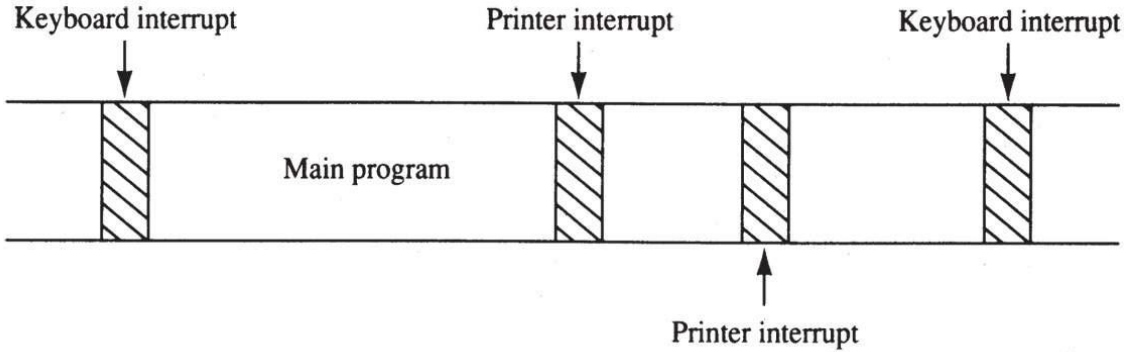
Interrupt

- An interrupt is a hardware or software generated change of flow within the system.
- The main purpose of the interrupt processing are:
  - Interrupts are useful when interfacing I/O devices at relatively low data transfer rates, such as keyboard inputs
  - It reduces the waste of time
  - It allows simultaneous execution of softwares

Interrupt : Example

- Interrupt processing allows the processor to execute other software while the keyboard operator is thinking about what to type next.
- When a key is pressed, the keyboard encoder debouncing a switch and puts out one pulse that interrupts the microprocessor.

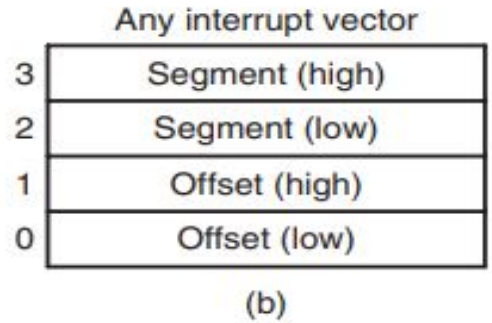
Figure 12–1 A time line that indicates interrupt usage in a typical system.



- a timeline shows typing on a keyboard, a printer removing data from memory, and a program executing
- the keyboard interrupt service procedure, called by the keyboard interrupt, and the printer interrupt service procedure is called by Printer interrupt
- each take little time to execute

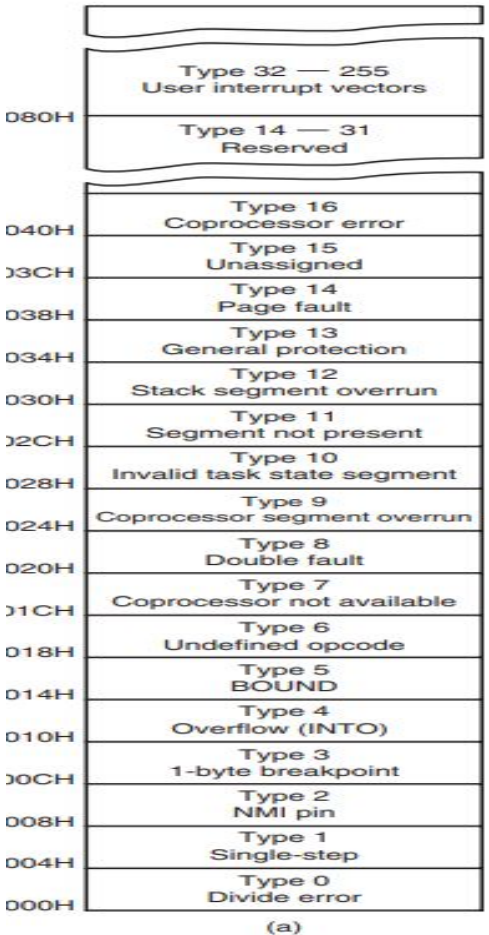
Interrupt Vectors

- An interrupt vector contains the address (segment and offset) of the interrupt service procedure.



- The **interrupt vector table** is located in the first 1024 bytes of memory at addresses 000000H–0003FFH.
- contains 256 different four-byte interrupt vectors

Interrupt Vectors



(a)

- Intel reserves the first 32 interrupt vectors
- the last 224 vectors are user-available
- each is four bytes long in real mode and contains the starting address of the interrupt service procedure.

Intel Dedicated Interrupts

- **Type 0**
  The **divide error** whenever the result from a division overflows or an attempt is made to divide by zero.
- **Type 1**
  Single-step or trap occurs after execution of each instruction if the trap (TF) flag bit is set.

- **Type 2**
  The **non-maskable interrupt** occurs when a logic 1 is placed on the NMI input pin to the microprocessor.
  – non-maskable—it cannot be disabled
- **Type 3**
  A special one-byte instruction (INT 3) used to store a breakpoint in a program for debugging

- **Type 4**
  Overflow is a special vector used with the INTO instruction. The INTO instruction interrupts the program if an overflow condition exists.
  - as reflected by the overflow flag (OF)

Type 5

- The **BOUND** instruction compares a register with boundaries stored in the memory.
- If the contents of the register are greater than or equal to the first word in memory and less than or equal to the second word, no interrupt occurs because the contents of the register are within bounds. if the contents of the register are out of bounds, a type 5 interrupt ensues

- For example, if the instruction BOUND AX,DATA is executed, AX is compared with the contents of DATA and DATA+1 and also with DATA+2 and DATA+3. If AX is less than the contents of DATA and DATA+1, a type 5 interrupt occurs. If AX is greater than the contents of DATA+2 and DATA+3, also type 5 interrupt occurs.

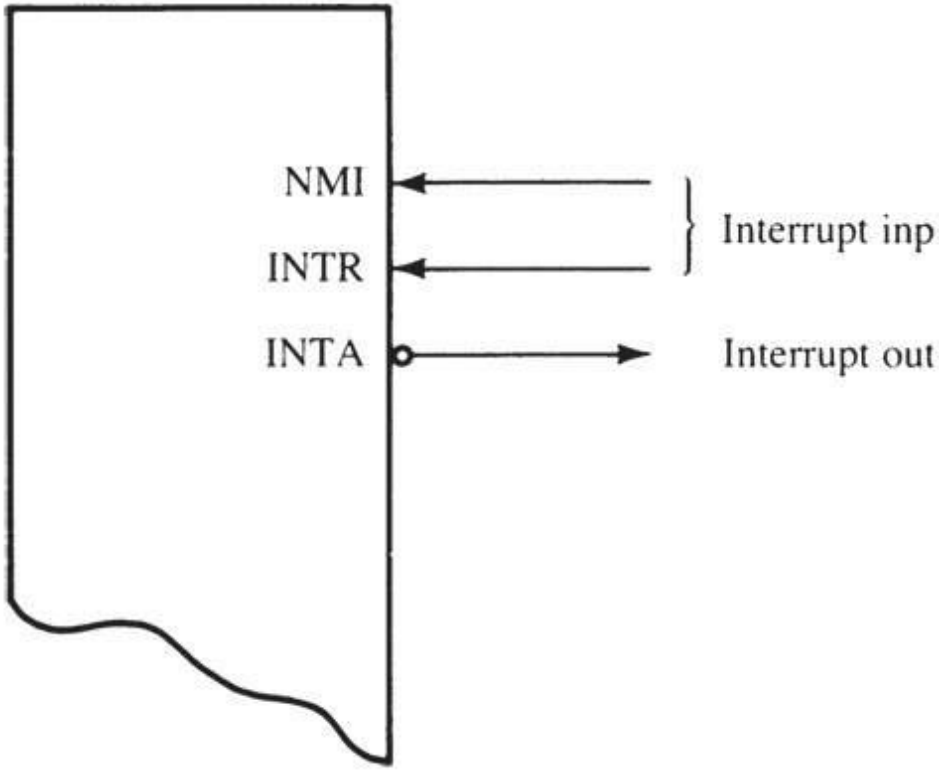
Interrupt Instructions: BOUND, INTO, INT, INT 3, and IRET

- **BOUND** : Check array against boundary
- **INTO:** The INTO instruction checks or tests the overflow flag (O). If O = 1, the INTO instruction calls the procedure whose address is stored in interrupt vector type number 4.
- **INT n:** INT n instruction calls the interrupt service procedure that begins at the address represented in vector number n. For example, INT 5 = 4 × 5 or 20 (14H). The vector for INT 5 begins at address 0014H and continues to 0017H
- **INT 3: Interrupt 3**
- **IRET:** used to return for both software and hardware interrupts.

12–2 HARDWARE INTERRUPTS

- The two processor hardware interrupt inputs:
  - non-maskable interrupt (NMI)
  - interrupt request (INTR)
- One interrupt output:
  - INTA

Figure 12–5 The interrupt pins on all versions of the Intel microprocessor.



The **non-maskable interrupt** (NMI):

- It is an input pin that requests an interrupt.
- The NMI pin must remain logic 1 until recognized by the microprocessor.
- This type of interrupts can not be avoided.
- The NMI input is often used for parity errors and other major faults, such as power failures.

- **Interrupt request input (INTR)**: It is used for requesting an interrupt. It must be held at a logic 1 level until it is recognized. INTR is set by an external event and cleared inside the interrupt service procedure .INTR is automatically disabled once accepted and re-enabled by IRET at the end of the interrupt service procedure
- **INTA:** The processor responds to INTR by pulsing INTA output. It is used to apply interrupt vector type number on data bus connections D_7–D_0.

Operation of a Real Mode Interrupt

- When the processor completes executing the current instruction, it determines whether an interrupt is active by checking:
  - (1) instruction executions
  - (2) single-step
  - (3) NMI
  - (4) coprocessor segment overrun
  - (5) INTR
  - (6) INT instructions in the order presented

Interrupt Handeling Steps

- External device or PIC asserts INTR = 1.
- CPU finishes the current instruction and samples INTR; if IF = 1, it accepts the interrupt.
- CPU drives $\overline{\text{INTA}}$ low (first acknowledge).
- CPU drives $\overline{\text{INTA}}$ low again (second acknowledge); device/PIC places the 8-bit interrupt type on D7..D0 and CPU latches it.
- CPU computes IVT_entry = type × 4.
- CPU reads new IP and new CS from the IVT.
- CPU pushes FLAGS, CS, IP onto the stack and clears IF and TF.
- CPU loads CS:IP and begins executing the ISR.
- ISR ends with IRET; CPU pops IP, CS, FLAGS and resumes execution.

Interrupt Flag Bits

- IF: when IF is set, it *allows* the INTR pin to cause an interrupt. When IF is cleared, it *prevents* the INTR pin from causing an interrupt
-
- TF: when TF = 1, it causes a trap interrupt (type 1) to occur after each instruction executes .Trap is often called a *single-step*. when TF = 0, normal program execution occurs

– the interrupt flag is set and cleared by the STI and CLI instructions, respectively
– the contents of the flag register and the location of IF and TF are shown here

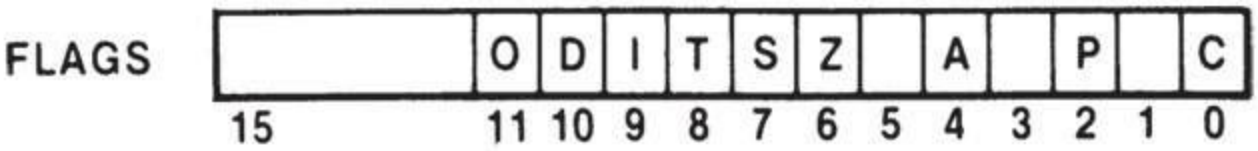


Figure 12–4 The flag register. (Courtesy of Intel Corporation.)

Priority Interrupt

- Whenever a peripheral is ready to transfer data , it is necessary to service its interrupt within a reasonable time.
  - Otherwise subsequent data may be lost
- **Faster peripheral require faster servicing**
- If two or more peripherals are ready at the same time, it is better to **service the faster one first** since the slow one can be made to wait a little while
- Multiple interrupt requests can be resolved by using a priority interrupt scheme
- A **hardware priority encoder** arbitrates between requests and sends a single value- that **represents the highest priority device**- to the CPU
- The interrupt request is formed by performing a **logical OR** of the peripherals IREQ lines.

Priority Interrupt

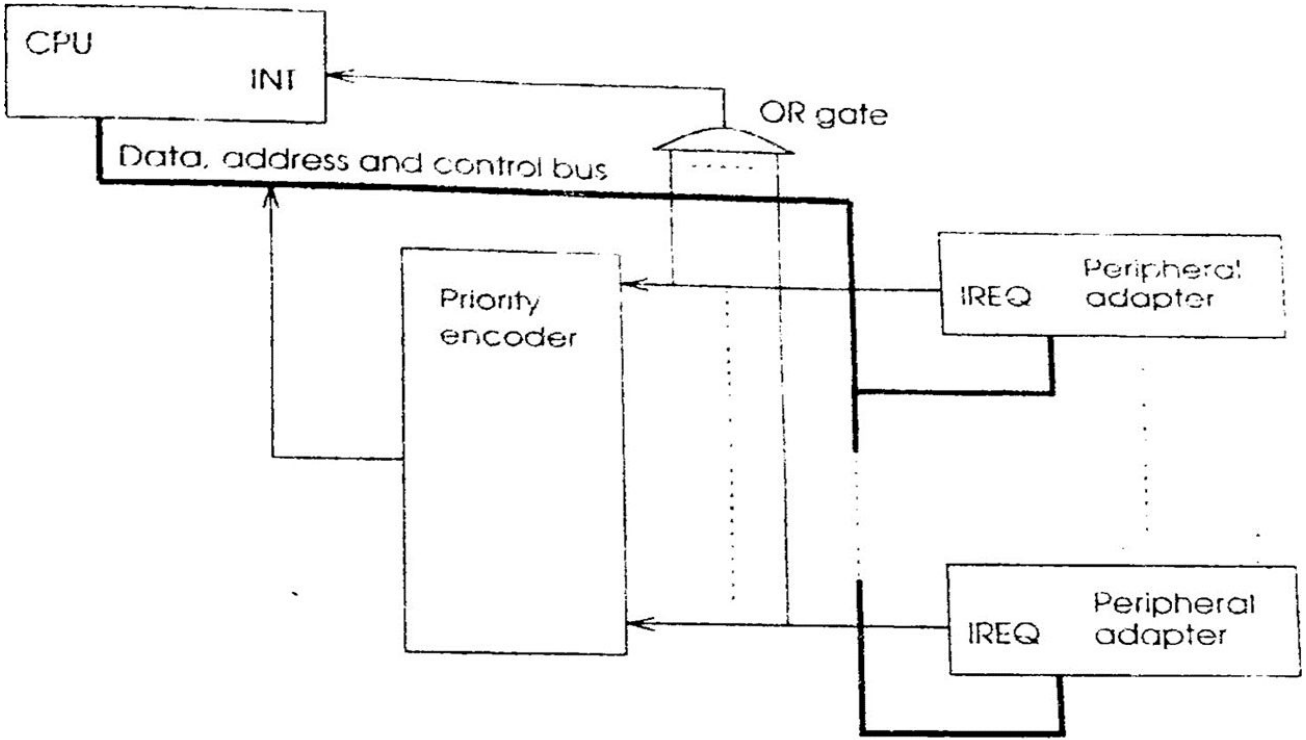


Figure 2.6 *Priority interrupts using a priority encoder*

Priority Interrupt Problem & Solution

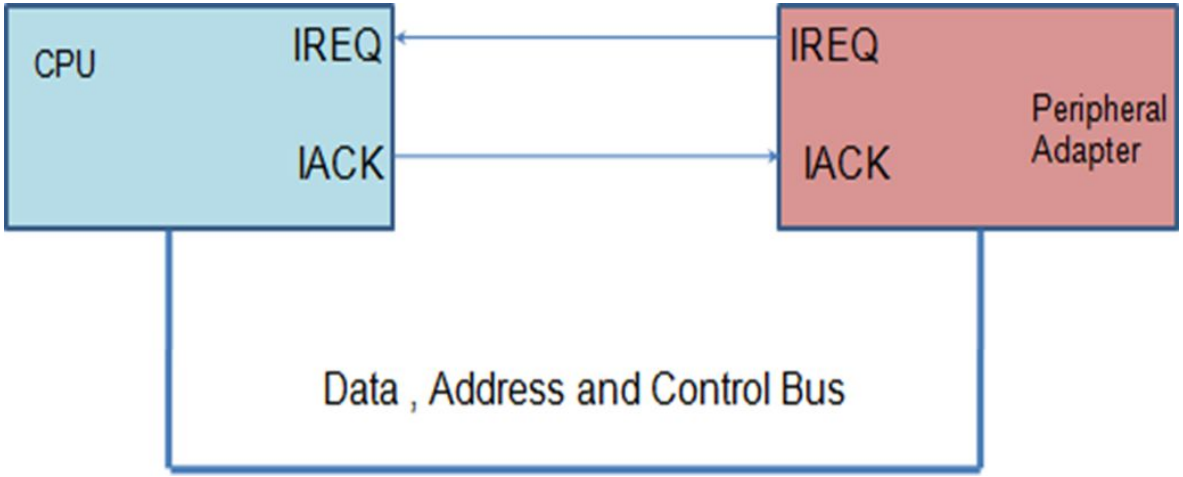
Problem

- Interrupt requests are assumed to **remain asserted until reset by instructions** in the service routine.
- It is not possible for another request to be seen until a request is de-asserted
- Data from a fast peripheral being lost while a service routine is getting around to clearing a low priority interrupt

Solution

- Most of the computers have a signal generated by the CPU that is returned to the peripheral as soon as the interrupt is detected- ***Interrupt acknowledge signal*** **(IACK).**
- This clears the interrupt request from that device and allows other devices to use the interrupt line.

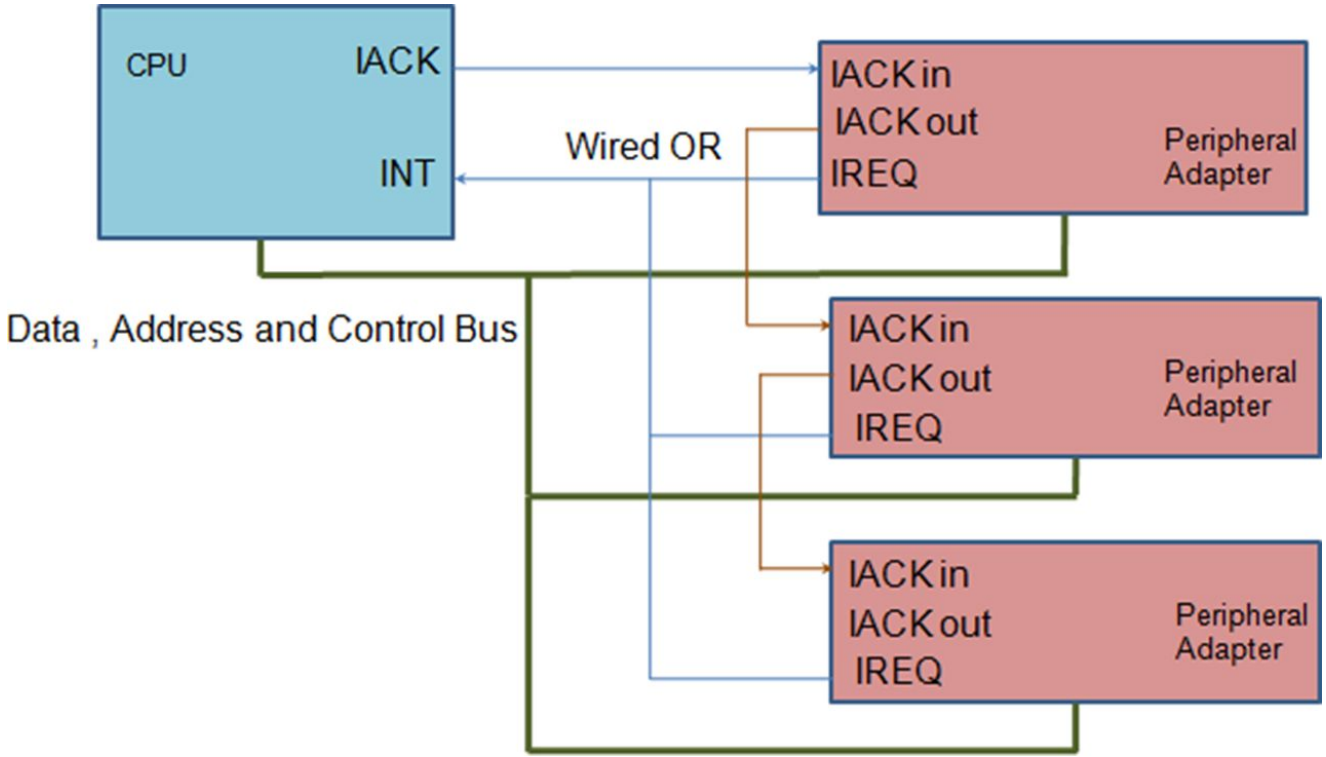
Interrupt with Acknowledgement



Priority Interrupt Using Daisy Chain

- Using an interrupt acknowledge enables the construction of priority interrupt scheme.
- CPU is able to **determine priority** not from the interrupt request but **by which device the acknowledge is sent to**.
- All the interrupt request lines are OR together.
- The **CPU IACK is connected directly to the highest priority device** so that if more than one request has been made the highest priority device sees it first.
- If it has not made a request, **it passes the IACK along to the next device**
- This continues down to the lowest priority device.
    - This receives an acknowledgement only if no other device has made a request.

Priority Interrupt Using Daisy Chain

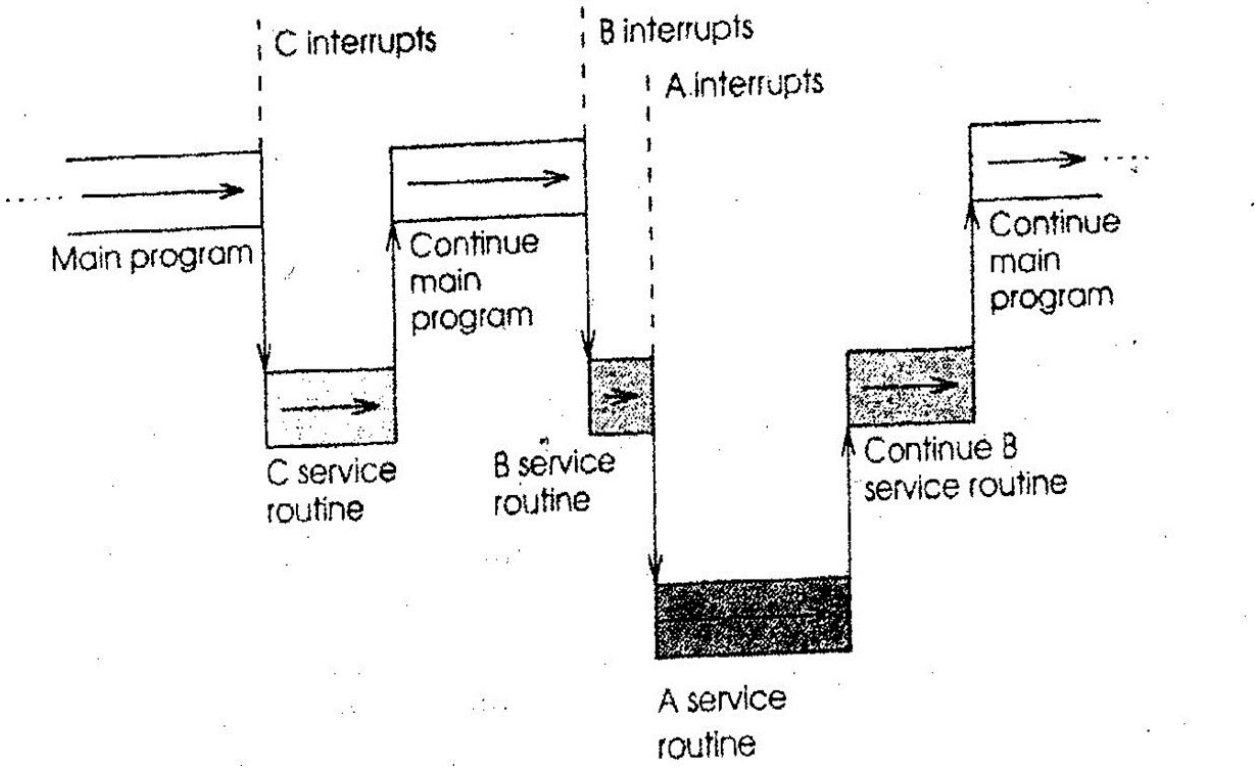


Nested Interrupt

- In order to ensure a fast response to a high priority interrupt it is possible to allow interrupts to interrupt interrupts.

  .

Nested Interrupt



Nested Interrupt

- Device C interrupts the main program, it’s service procedure runs to the completion, returning to the main program.
- Device B similarly interrupt the main program but is,. In turn, interrupted by a higher priority device, A.
- Execution of the service procedure for A is nested within that for B
- When the service routine for device A complete the CPU returns to perform the service routine for B
- When it completes the main program continues.

.

Thank You

Military Institute of Science and Technology
Department of Computer Science and Engineering

Peripheral/Computer Connections (Part 2)

Course Name: Computer Interfacing
Course Code: CSE-405

Book Reference

- Computer Peripherals (3rd Edition) - Cook and White; Butterworth- Heinemann (1995)

Block Data Transfer

- Processors have instructions for moving **blocks of data** which may be used for I/O.
- **CPU has to synchronize** its block moves with the peripheral and is unable to performs any other functions while the block moves is in progress.
- **Only suited for fast transfers** where the CPU and the peripheral speeds are reasonably well matched.
- It is not commonly used.

Direct Memory Access (DMA)

- It is better to avoid using the CPU completely for I/O transfer.
- A special hardware called **DMA controller** can be used to transfer data between memory and I/O devices.
- When a peripheral indicates that it is ready for data transfer
  - The **DMA unit gains the control of the Bus**.
  - Places **appropriate address and control signals** on it to make the transfer.
  - **Releases the bus**.
- This action of taking over the bus for a period and **executing a memory access cycle** instead of the CPU doing so is known as *cycle stealing.*

Parallel Interface: SCSI

Parallel Interface: SCSI

- **SCSI (Small Computer System Interface)** is a **set of standards** for physically **connecting and transferring data between computers and peripheral devices especially storage devices.**
- Used to connect computers to disk drives and other peripherals (generally those needing a high data rate).
- A complete communication protocol and bus standard
- **8 devices can be supported on the SCSI bus**
  - Each device can have 8 logical units
  - Each logical unit contains a unique Logical Unit Number (LUN)
  - Each logical unit can have 256 logical sub units
    - Thus a very large total number of possible devices

Parallel Interface: SCSI

- Each device can be either an **initiator**, a **target** or both.

Initiator:

- A SCSI device that requests an operation to be performed **by another SCSI device**.
- Initiate a SCSI session.
- Does not provide any Logical Unit Number.

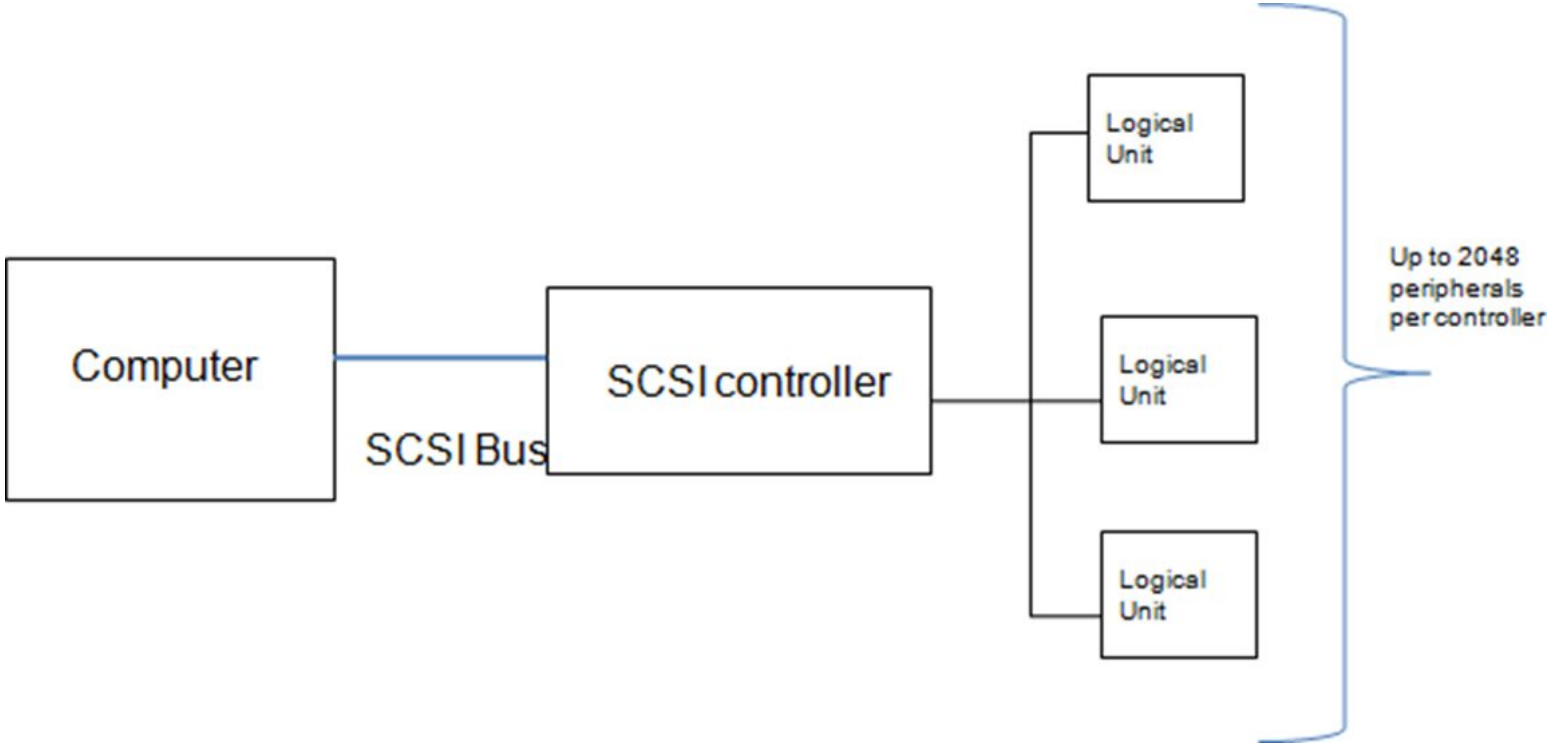
Target:

- A SCSI device that performs an operation as requested by an initiator.
- Waits for initiators' commands and provides required input/output data transfers.
- Provide one or more Logical Unit Number.

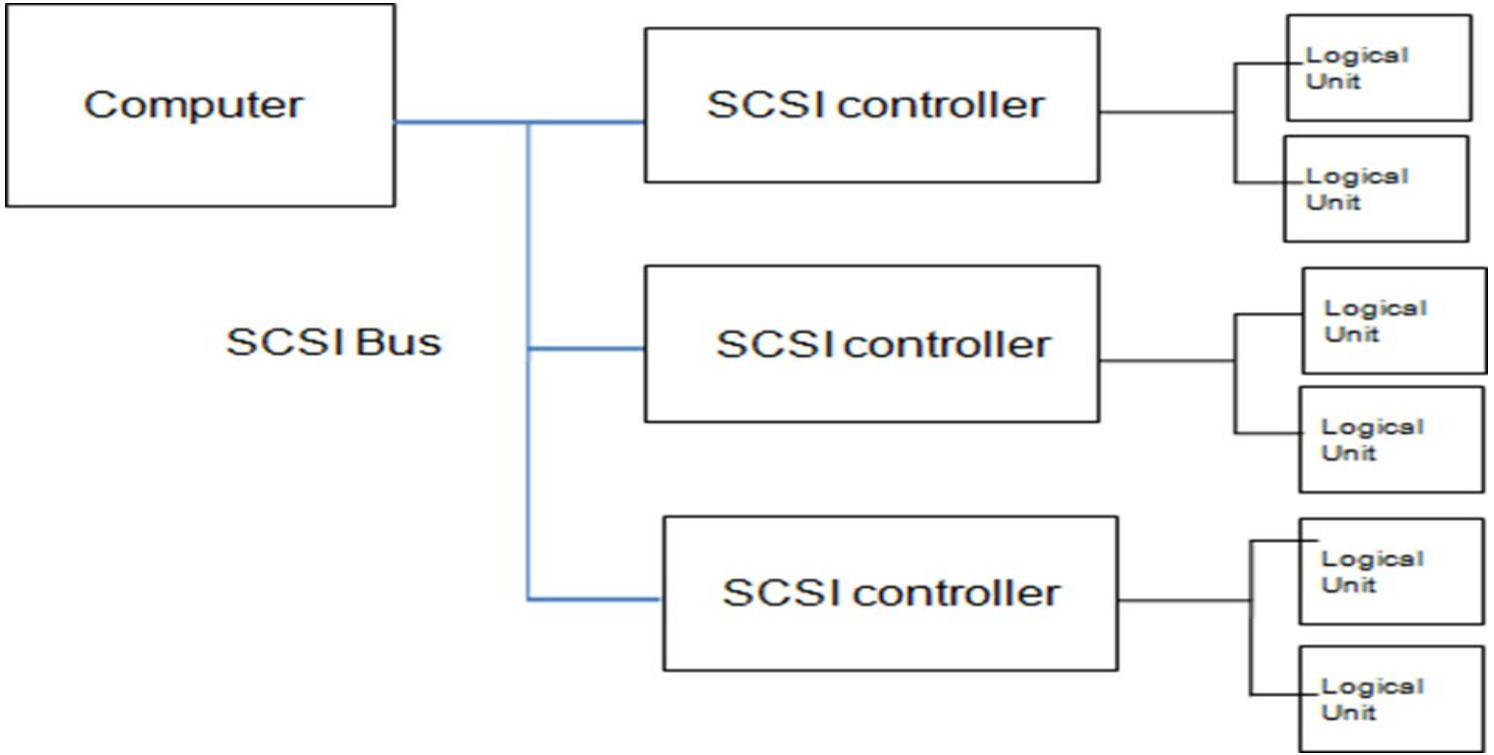
Parallel Interface: SCSI

- To be useful SCSI bus **must have at least one initiator and one target**
  - Can have multiple initiator and/or target
- Data is transferred over an **8 bit bidirectional bus**
  - **Optional parity line to check the error** of transmission
- **Data and control lines are implemented** using **one wire each together with common ground in single-ended SCSI**
  - With pair of wires for each line in differential SCSI
- When an **end user sends a request**, the **operating system sends the SCSI commands to the SCSI controller**, which then sends it to the storage device.
- **SCSI controller** (embedded on motherboard) is a **host bus adapter** or a chip that allows a SCSI storage device to communicate with operating system across a SCSI bus.

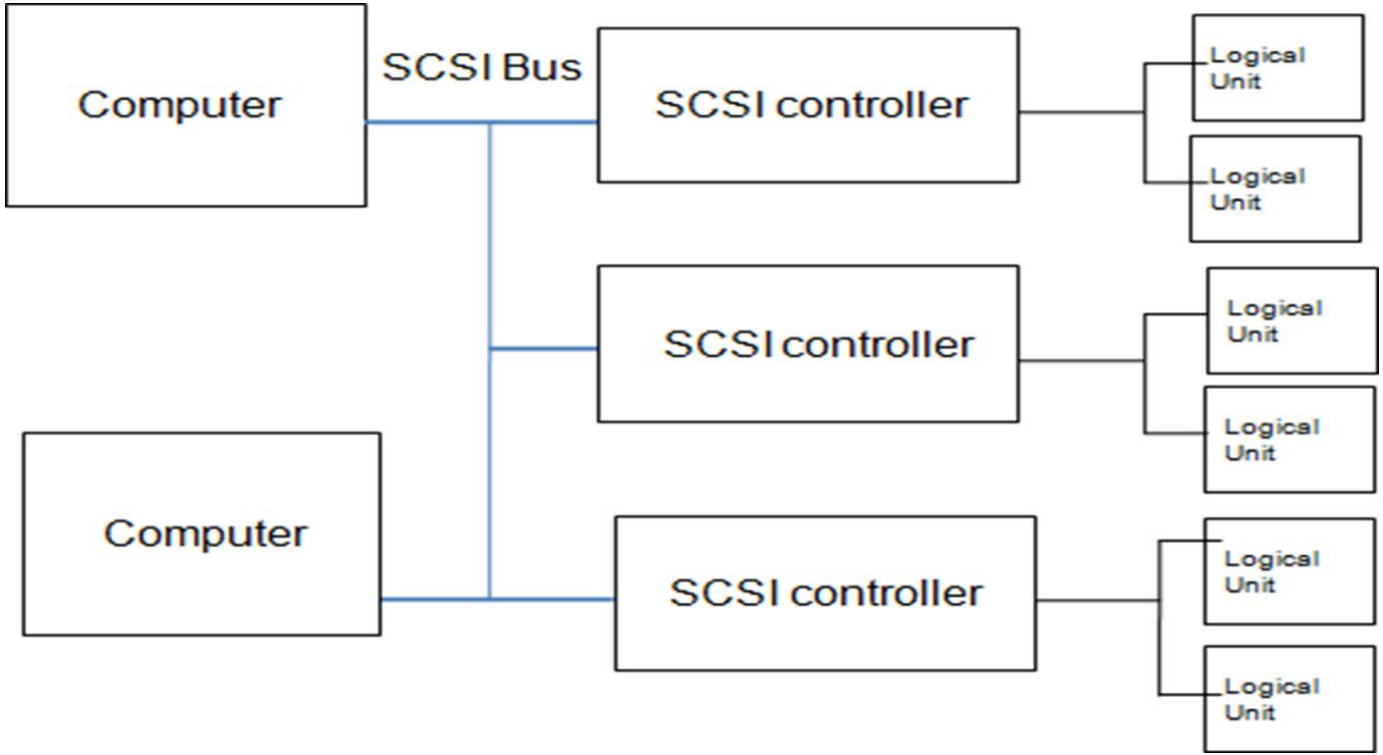
One Initiator, One Target



One Initiator, Multiple Targets



Multiple Initiators, Multiple Targets



Protocol for Using SCSI Bus

- Phase 1 : Arbitration
- Phase 2 : Selection
- Phase 3 : Information Transfer
- Phase 4 : Bus free
- Phase 5 : Reselection

➢ Several commands can be received before the first is completed

➢ Devices can queue requests

Protocol for Using SCSI Bus

Phase 1: Arbitration

- Allows one SCSI device (here initiator) to gain control of the SCSI bus so that it can initiate or resume an I/O process.
- The **SCSI device shall first wait for the BUS FREE phase** to occur

Phase 2: Selection

- The SELECTION phase **allows an initiator to select a target** for the purpose of initiating some target function.

Phase 3: Information transfer

- The COMMAND, DATA, STATUS, and MESSAGE phases are combined to transfer data or control information.

Protocol for Using SCSI Bus

Phase 4: Bus Free

- BUS FREE phase is entered when a **target release the BUSY signal**
- The BUS FREE phase indicates that **there is no current I/O process and the SCSI bus is available for a new connection.**

Phase 5: Reselection

- When the target is ready to respond to the given command then it can re-establish contact with the initiator in Reselection phase.
- Connection that started by the initiator but was **suspended by the target**.

SCSI: Mathematical Problem-1

Initiator	Target	Arbitration time of initiator (sec)	Task time of target given by initiator (sec)
2	1	4	8.4
1	3	5.1	5
1	2	6.4	5.8
1	3	8	3.5
2	2	8.2	4.9

Suppose, 2 computers are sharing a common SCSI bus among them. 3 SCSI controllers are connected with the same bus. Now solve the following scenario and find out the value of t at which all the controllers and bus will become free after executing all the tasks given by the all the computers.

Consider ,data transfer rate between all components = 1.7 sec

And arbitration +selection/reselection time= 2 sec

SCSI: Mathematical Problem-2

Initiator	Target	Arbitration time of initiator (sec)	Task time of target given by initiator (sec)
2	3	1	10
1	2	3	8
1	3	6	4
3	1	8	20
3	3	12	9

Suppose, 3 computers are sharing a common SCSI bus among them. 3 SCSI controllers are connected with the same bus. Now solve the following scenario and find out the value of t at which all the controllers and bus will become free after executing all the tasks given by the all the computers.

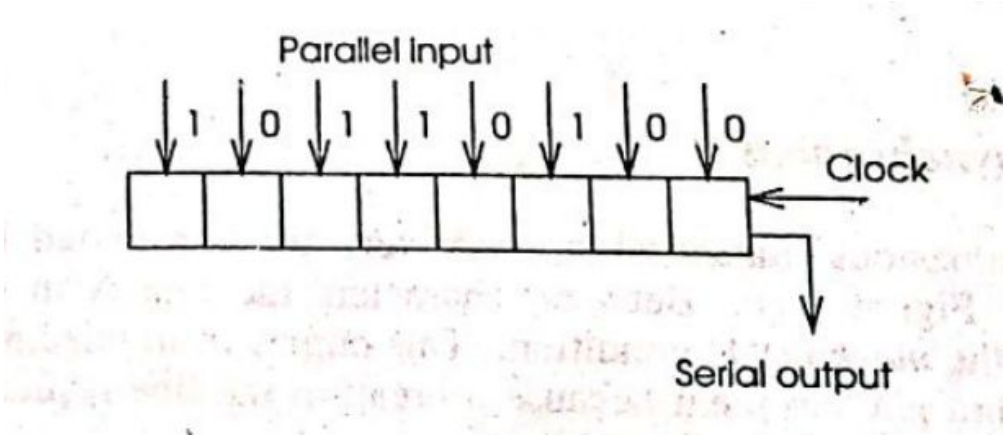
Consider ,data transfer rate between all components = 2 sec

And arbitration +selection/reselection time= 1 sec

Serial Interface

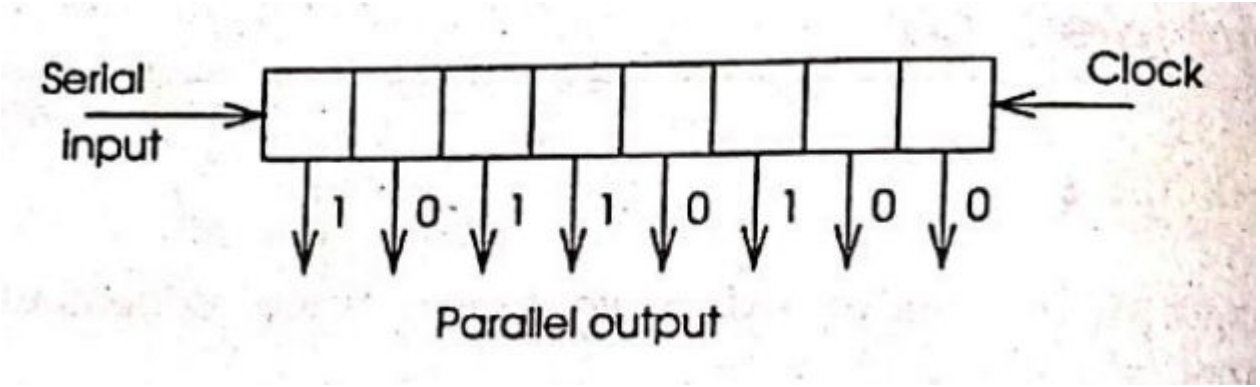
Parallel to Serial Interface

- Simplest convertor is a **shift register**
- Parallel data word is loaded to into a shift register
- A pulse on the clock input shift the data by 1 bit. And output 1 bit.



Serial to Parallel Interface

- Sequence of n clock pulse caused the input to propagate along the shift register
  - Until it is all available in parallel
  - First bit shifted all the way through shift register and appears at the right end.
- **The clock in the receiver must operates with the same timing as the transmitter.**



Asynchronous Transmission

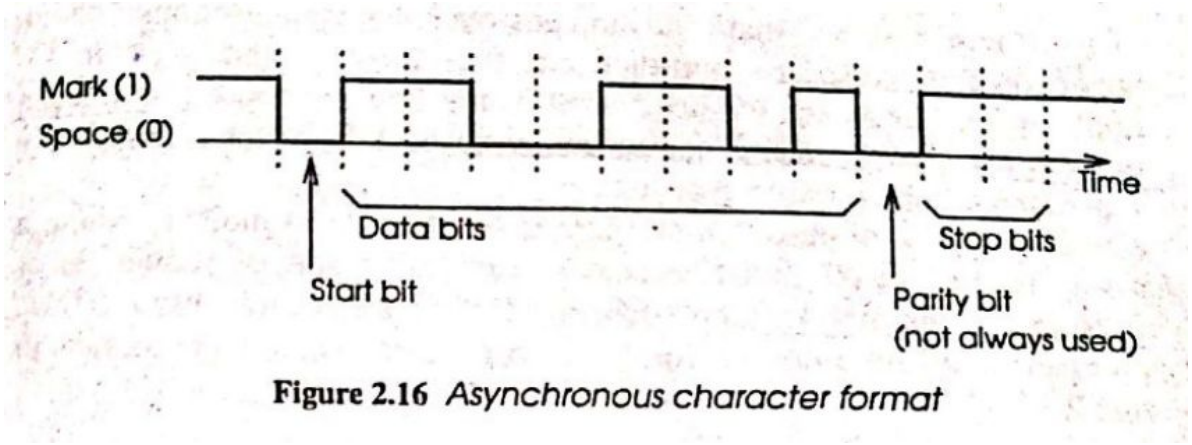


Figure 2.16 *Asynchronous character format*

Asynchronous Transmission

- Between characters the lines are is in the **idle state** which is the **‘mark’ or ‘1’** condition
- The first bit of the character, the **‘start bit’** is in the **‘space’ or ‘0’ condition**
- The 1 to 0 transition is used to start the receiver clock and indicates that a character is arriving
- The **first bit of data can not be the start bit**
- Because if t**he first bit is 1 it will be indistinguishable from the idle condition**
- For the data part of the character 5,6,7 or 8 data bits are transmitted with least significant bit first
- There may be a p**arity bit to allow transmission error** to be detected
- Finally, the line returns to a idle state fo**r 1, or 2 stop bits which allow the receiver to** settle ready for the next character
- Also known as ‘stop-start’ transmission

Asynchronous Transmission

- Irregular intervals may occur between character
- The s**ender and the receiver are not synchronized at the character level**
- Hence known as asynchronous transmission

Synchronous Transmission

- Supply a **clock signal** and **a data signal**
- Transmission may be along a **pair of line, one for clock signal and one for data**
- Clock may be merged with data stream into a single line
- Two advantages:
  1. **No time wasted** for non-data bits
  2. **Less margin of error** is required to ensure correct recovery of data- increase acceptable **transmission speed**
- **Use a special sequence of data known as protocols**
- One protocol: HDLC (High Level Data Link Control)

Synchronous Transmission : HDLC format

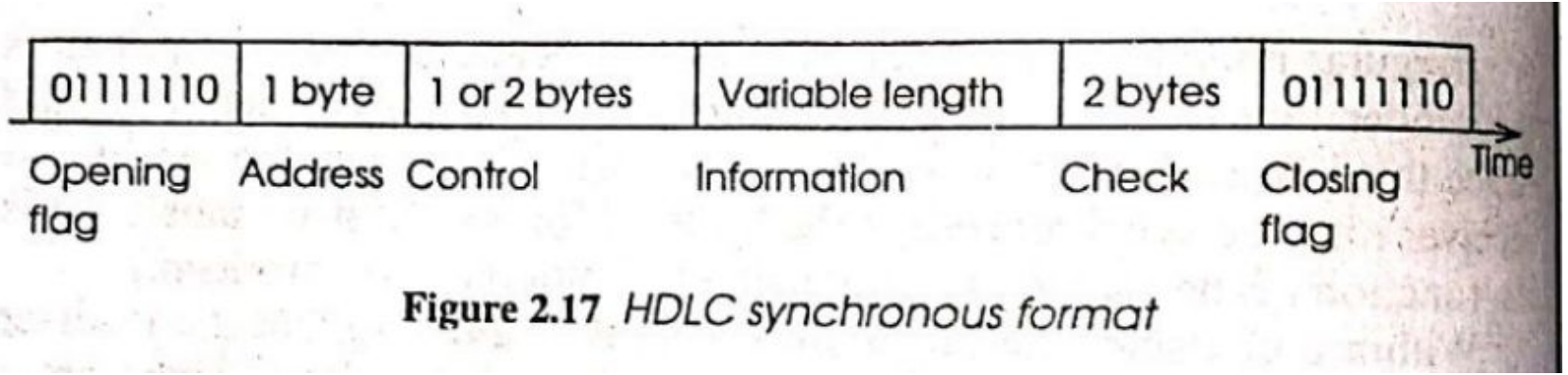


Figure 2.17 *HDLC synchronous format*

Synchronous Transmission: Bit stuffing

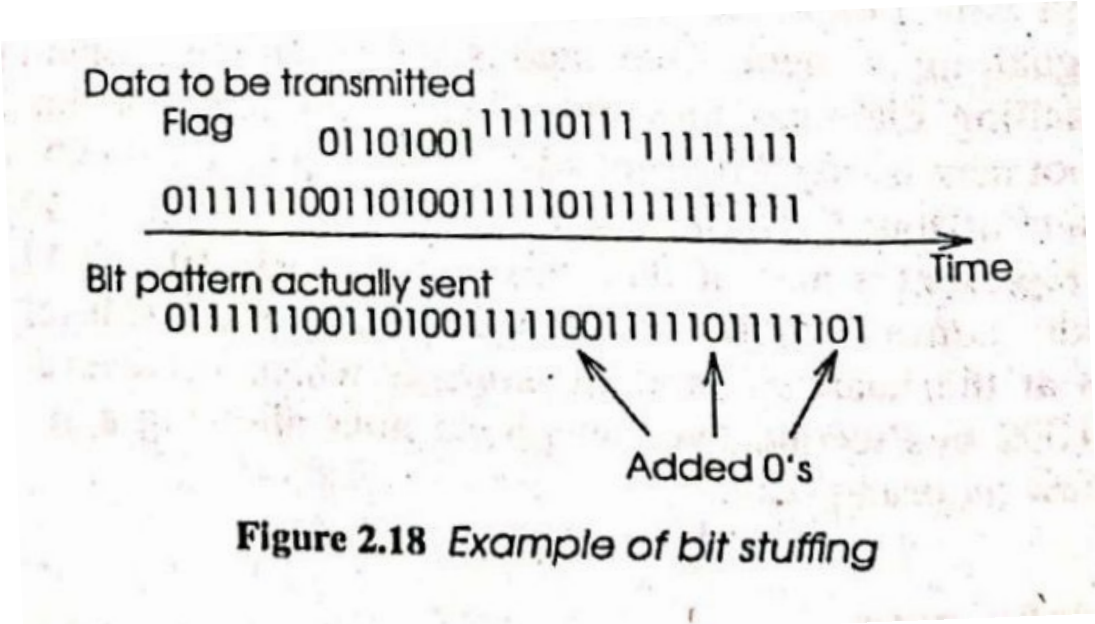


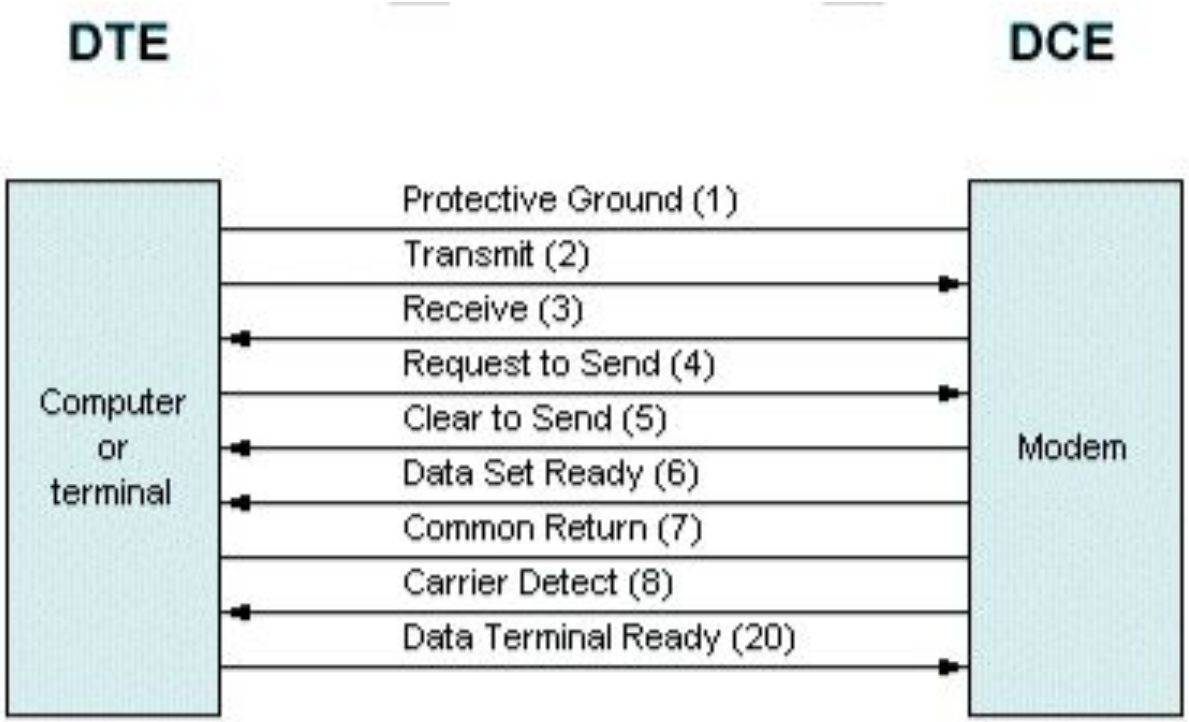
Figure 2.18 *Example of bit stuffing*

The RS232 Standard

- A standard used for serial data transmission.
- Communication is assumed to be between **Data Terminal Equipment (DTE) and Data Communication Equipment (DCE).**
- DTE can be a computer or peripheral, DCE is another device such as a modem **that connects the DTE to a communication network**.
- **Standard telephone** line can be used due to its availability.

The RS232 Standard Protocol

- When the terminal is powered on, it performs **a self-checking and asserts DTR (Data Terminal Ready) signal** to tell the modem that it is ready.
- When modem is ready, **it sends DSR (Data Set Ready) signal to the terminal**.
- Then the modem dials the computer.
- When terminal is ready to send, **it asserts RTS (Request to send) signal**.
- Modem asserts **CD (Carrier Detect) signal** to indicate that **it has established contact with computer**.
- When modem is ready, it asserts **CTS (Clear to Send) signal**.
- Terminal sends data.
- When all data has been sent, **terminal raises**, **RTS signal high**.
- Modem raises **CTS signal high**.


DTE
DCE
Computer or terminal
Modem
Protective Ground (1)
Transmit (2)
Receive (3)
Request to Send (4)
Clear to Send (5)
Data Set Ready (6)
Common Return (7)
Carrier Detect (8)
Data Terminal Ready (20)

Thank You

Military Institute of Science and Technology
Department of Computer Science and Engineering

16550 (UART) PROGRAMMABLE COMMUNICATIONS INTERFACE

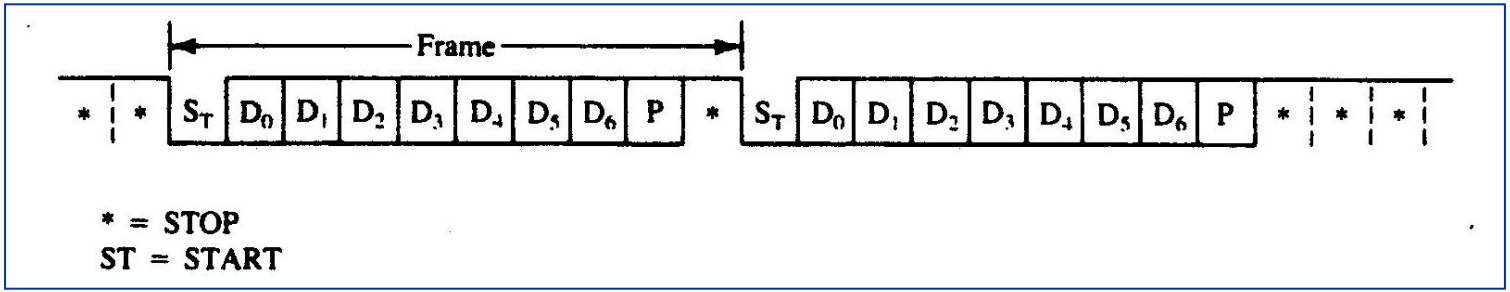
Course Name: Computer Interfacing
Course Code: CSE-405

16550: PROGRAMMABLE COMMUNICATIONS INTERFACE

- The National Semiconductor Corporation's **PC16550D** is a **programmable communications interface** designed to connect to virtually any type of serial interface.
- The 16550 is a **Universal Asynchronous Receiver/Transmitter (UART)** that is fully **compatible** with the **Intel microprocessors**.
- The 16550 is capable of operating at **0-1.5 M Baud**.
- The 16550 also includes a **programmable Baud rate generator** and s**eparate FIFO for input and output data** to ease the load on the microprocessor.
- Each **FIFO contains 16 bytes of storage**. This is the most common communications interface found in modem microprocessor-based equipment, including the personal computer and many modems

Asynchronous Serial Data

- Asynchronous serial data are transmitted and received **without a clock** or timing signal.



- Figure illustrates two frames of asynchronous serial data. **Each frame** contains **a start bit, seven data bits, parity, and one stop bit**.
- In this figure, a frame, which contains one ASCII character, has 10-bits.
- Most dial-up **communications systems, such as CompuServe, Prodigy, and America Online**, use 10-bits for asynchronous serial data with **even parity**.

16550 Functional Description

- Figure illustrates the pin-out of the 16550 UART.
- This device is available as a **40-pin DIP** (Dual In-line Package) or as a **44-pin PLCC** (Plastic Lead-less Chip Carrier).
- The 16550 **can control a Modem**.
- Six pins on the 16550 are devoted to modem control:

  DSR □Dataset ready

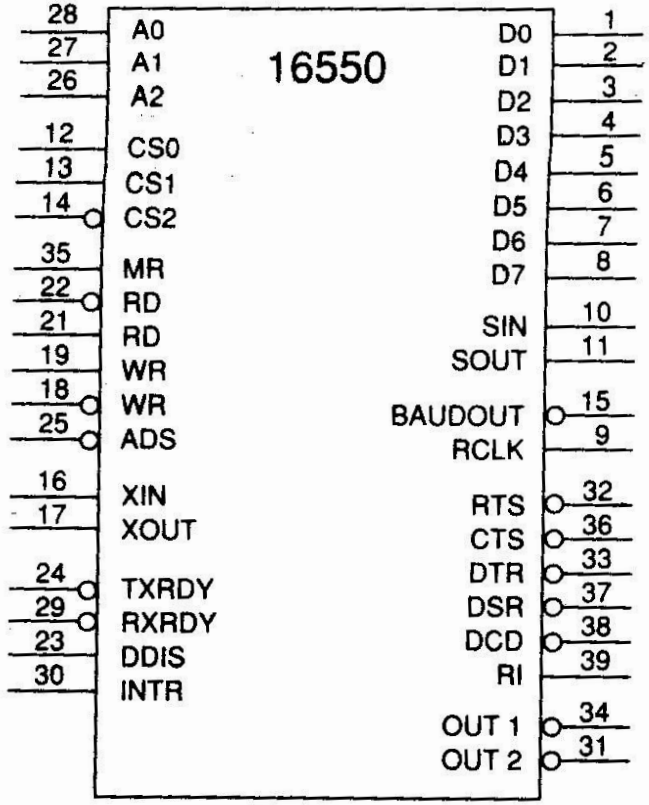
  DTR □Data terminal ready

  CTS □Clear to send

  RTS □Request to send

  RI □Ring Indicator

  DCD □Data carrier detect



DIGITAL DATA TRANSMISSION USING MODEMS AND STANDARD PHONE LINE

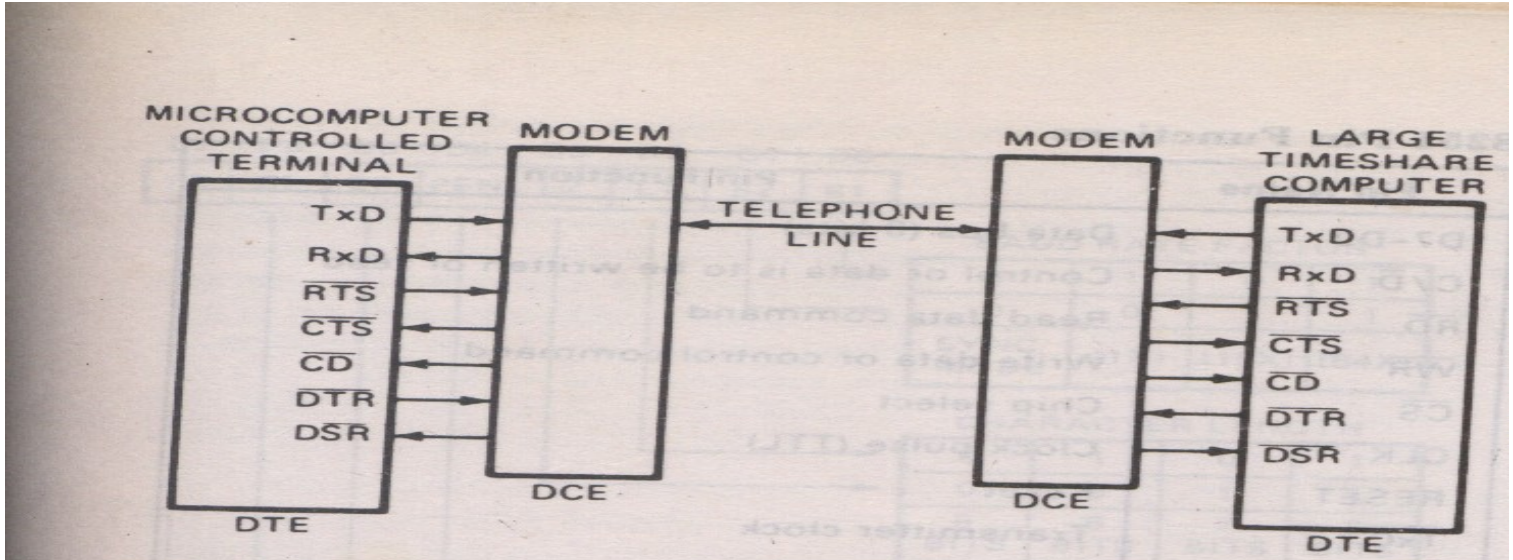


FIGURE 14-2 Digital data transmission using modems and standard phone lines.

16550 Functional Description

- **Two completely separate sections** are responsible for data communications: **the receiver and the transmitter.**
- Because each of these sections is independent of the other, the 16550 is able to **function in simplex, half-duplex, or full-duplex modes**.
- One of the main features of the 16550 is its i**nternal receiver and transmitter FIFO** (first-in, first-out) memories.
- Because each is **16 bytes deep**, the UART only requires attention from the microprocessor after receiving 16 bytes of data.

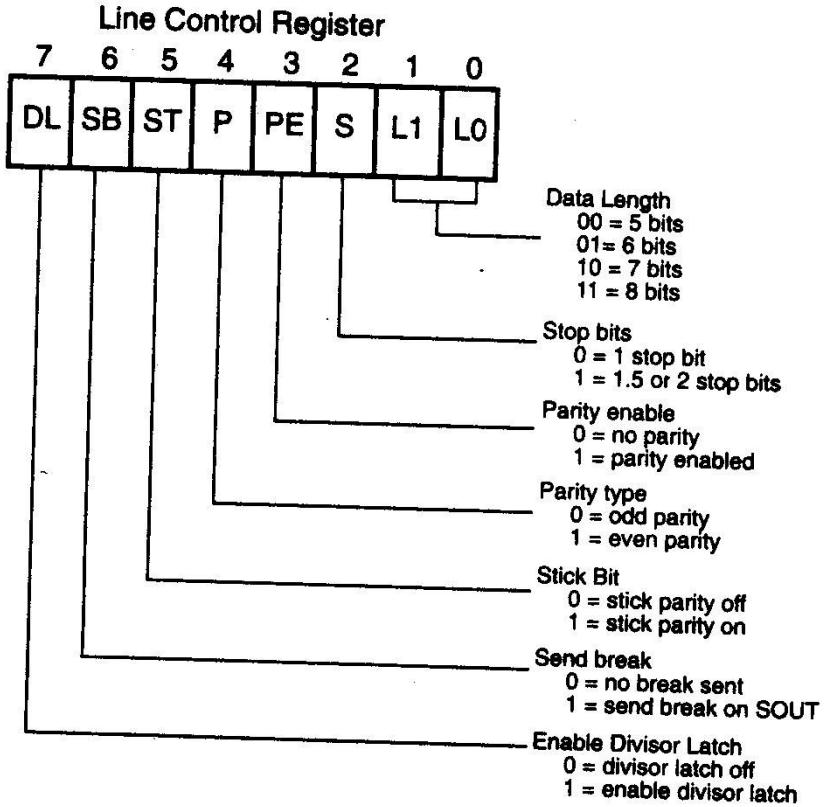
Programming the 16550

- **Initializing the 16550:** Initialization dialog, which occurs after a hardware or software reset
- **Consists of two parts**: **programming the line control register** and **the Baud rate generator.**

- The **line control register** selects the **number of data bits, number of stop bits, and parity (whether it is even or odd or if parity is sent as a one or a zero)**.

- The **Baud rate generator** is programmed with **a divisor** that determines the **Baud rate of the transmitter section**.

Line Control Register



ST	P	PE	Function
0	0	0	No parity
0	0	1	Odd parity
0	1	0	No parity
0	1	1	Even parity
1	0	0	Undefined
1	0	1	Send/receive 1
1	1	0	Undefined
1	1	1	Send/receive 0

Programming the Baud Rate

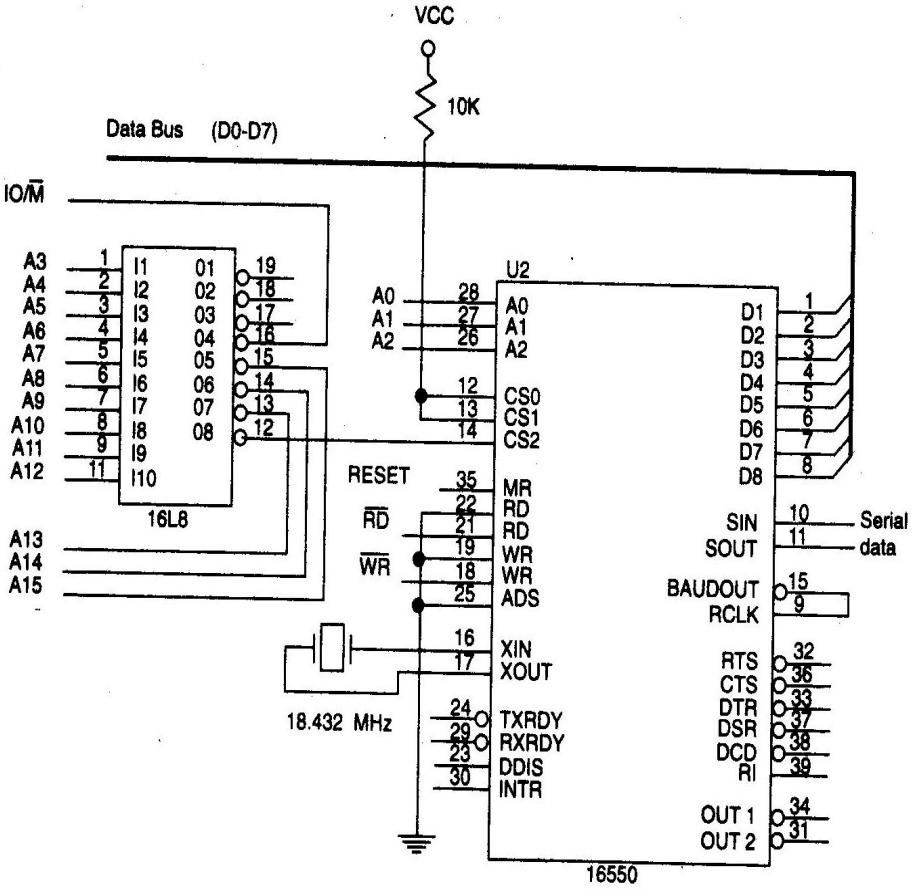
- The Baud rate generator is programmed **at I/O addresses 000 and 001** (A2, A1, A0).
- **Port 000 is used to hold the least-significant part of the 16-bit divisor**, and port **001 holds the most-significant part**.
- The value used for the divisor depends on the **external clock or crystal frequency.**
- Table 11-7 illustrates common Baud rates obtainable if a **18.432 MHz crystal is used as a timing source.**
- For example, if **240** is programmed into the Baud rate **divisor**, the Baud rate is 18.432MHz/(16 *240)=4800 Baud.

Baud rate	*Divisor value*
110	10,473
300	3,840
1,200	920
2,400	480
4,800	240
9,600	120
19,200	60
38,400	30
57,600	20
115,200	10

Programming the Baud Rate

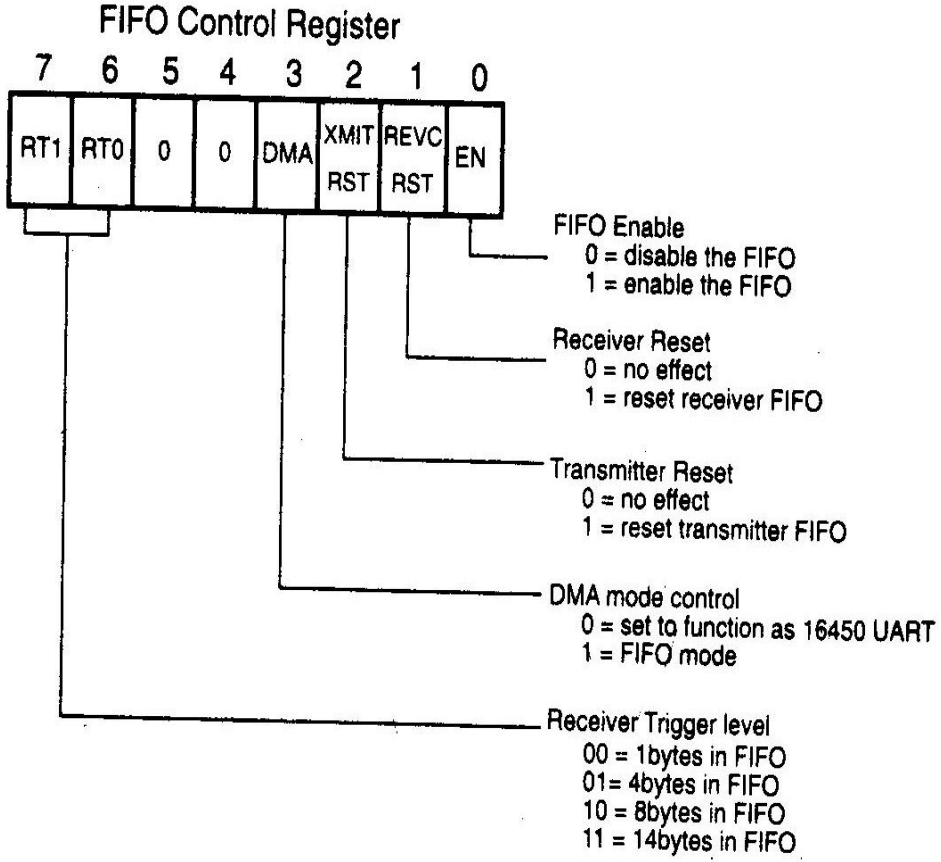
- Figure 11-45 shows the interface to the 8088 microprocessor using a PAL16L8 to decode the 8-bit port addresses **F0H through F7H**.

- Here port F3H accesses the line control register and FOH and F1H access **the Baud rate divisor registers**.



FIFO Control Register

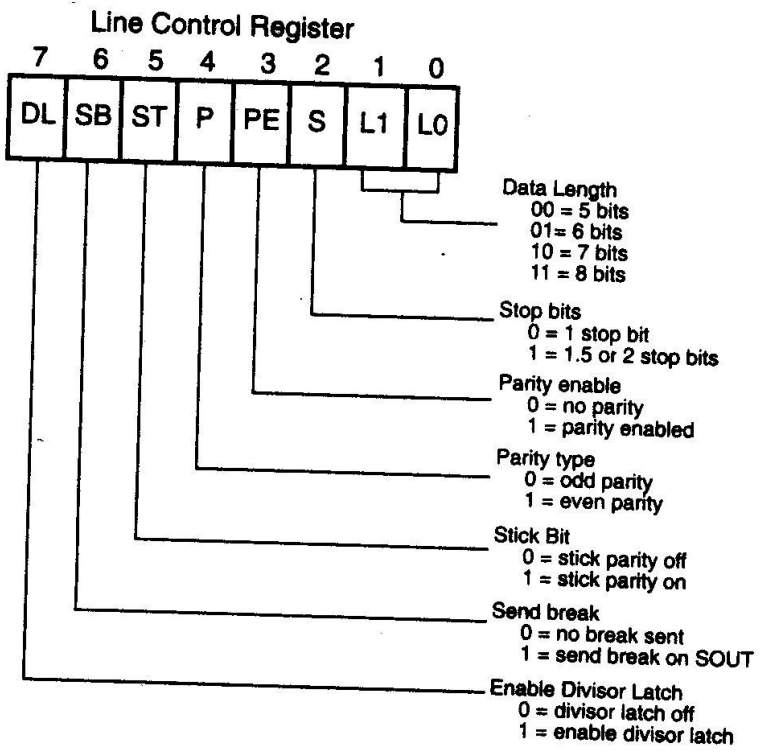
- This register enables the transmitter and receiver (bit 0 = 1) and clears the transmitter and receiver FIFOs.
- It also provides control for the 16550 interrupts
- This enables the transmitter and receiver and clears both FIFOs.



Sample Initialization

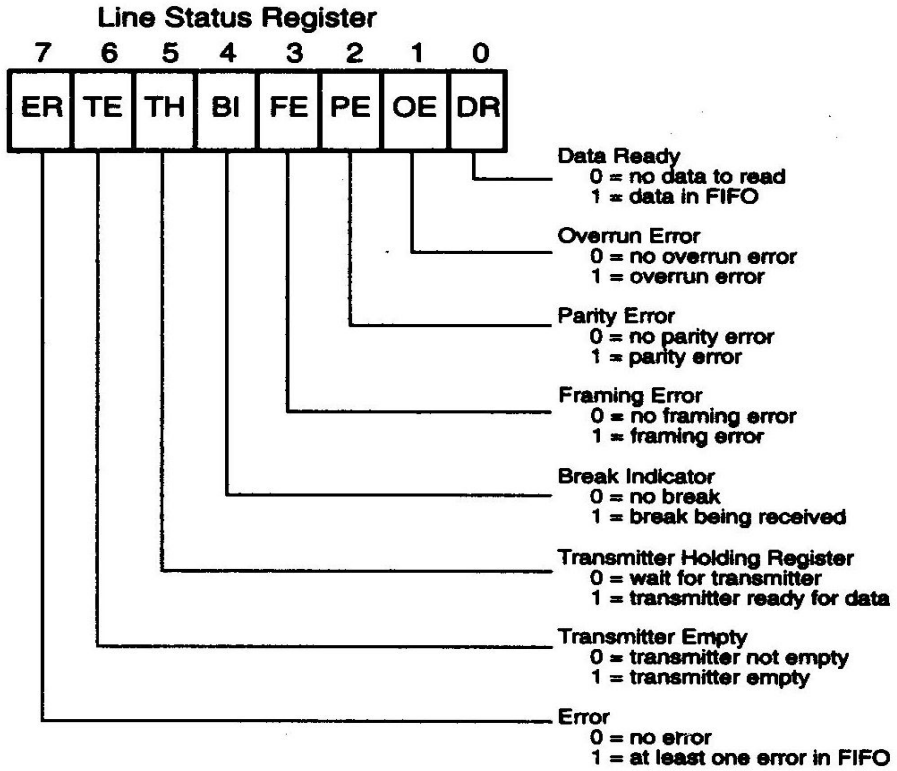
Suppose that an asynchronous system requires **seven data bits, odd parity, a Baud rate of 9600, and one stop bit**.

```
LINE EQU 0F3H
FIFO  EQU 0F2H
LSB   EQU 0F0H
MSB   EQU 0F1H
INIT PROC NEAR
        MOV AL, 10001010B
        OUT  LINE, AL
        MOV AL,120
        OUT  LSB, AL
        MOV AL,0
        OUT  MSB, AL
        MOV AL, 00001010B
        OUT  LINE, AL
        MOV AL,00000111B
        OUT  FIFO, AL
        RET
INIT ENDP
```



Sending Serial Data

- Before serial data can be sent or received through the 16550, we need to know the function of the **line status register**.
- The line status register contains information about error conditions and the state of the transmitter and receiver.
- This register is tested before a byte is transmitted or can be received.



Sending Serial Data

You are given the following 8086 assembly procedure to send a character to a UART (e.g., the 16650) using programmed I/O. In this routine, the data to be transmitted is stored in the AH register before calling the procedure.

```
LSTAT   EQU     OF5H
DATA    EQU     OFOH

SEND    PROC    NEAR USES AX
  .REPEAT
     IN   AL, LSTAT
     TEST    AL, 20H        ;test TH bit
  .UNTIL !ZERO

  MOV   AL, AH
  OUT   DATA, AL
  RET
SEND    ENDP
```

Receive Serial Data

A procedure that tests the DR bit to decide if the 16550 has received any data. Upon reception of data, the procedure tests for Errors.

```
LSTAT     EQU  OF5H
DATA      EQU  OFOH
SEND      PROC      NEAR USES AX
          .REPEAT
                IN     AL, LSTAT
                TEST  AL,  1
          .UNTIL  !ZERO?

          TEST AL, 0EH
          .IF  ZERO?
                IN     AL,DATA
          .ELSE
                MOV AL, '?'
          .ENDIF
          RET
REVC      ENDP
```

UART Errors

The types of errors detected by the 16550 are parity error, framing error, and overrun error:

- A **parity error** indicates that the received data contain the wrong parity.
- A **framing error** indicates that the start and stop bits are not in their proper places.
- An **overrun error** indicates that data have overrun the internal receiver FIFO buffer.

These errors should not occur during normal operation.

- A **parity error** indicates that **noise was encountered during reception**.
- A **farming error** occurs if the receiver is receiving data at an **incorrect Baud rate**.
- An **overrun error** occurs only if the software fails to read the data from the UART before the receiver FIFO is full.

Thank You

Military Institute of Science and Technology
Department of Computer Science and Engineering

Interfacing Microcomputer Ports to high power devices

Course Name: Computer Interfacing
Course Code: CSE-405

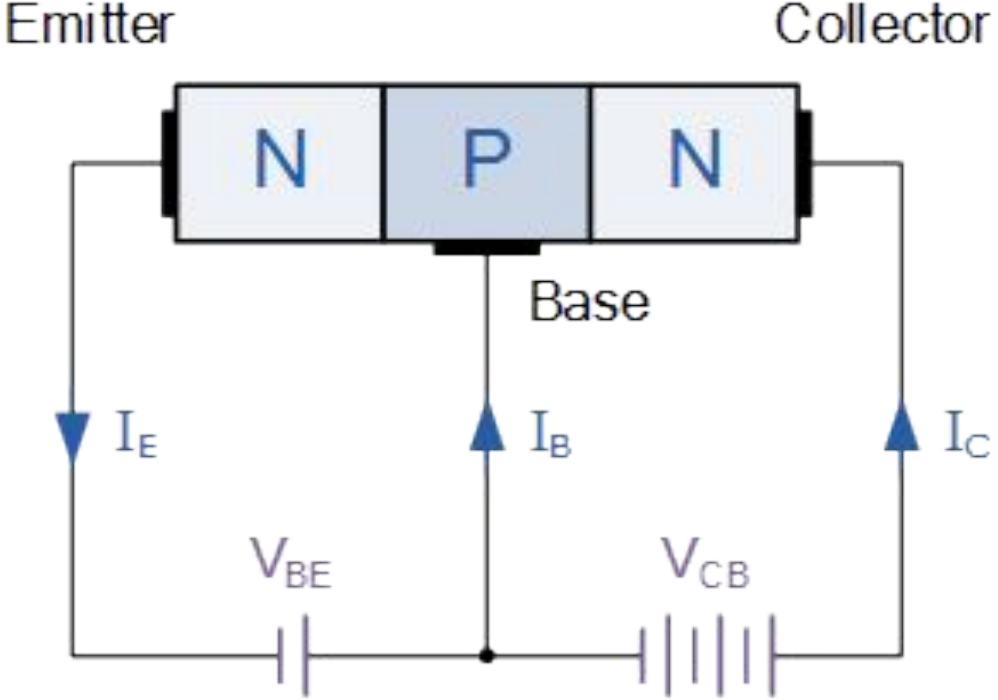
Book Reference

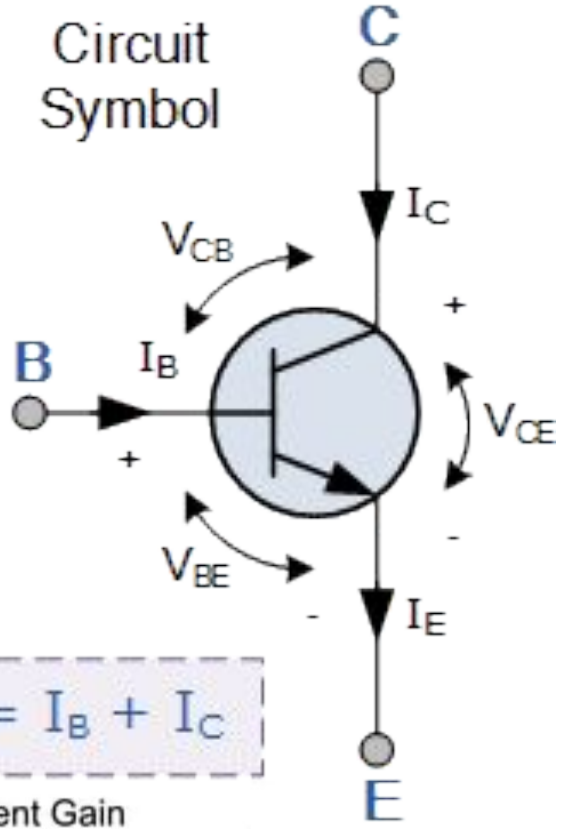
- Microprocessor and Interfacing- Douglas V. Hall (Chapter 9)

Introduction

- High power devices – lights, motors, solenoids, heaters etc
- Programmable port devices can source only a few tenths of a milliampere from the +5V supply and sink **only 1 or 2 mA** to ground
- We need to use interface devices between the port pin and the high power device
- One of the interfaces is **Transistor Buffer**

Transistor Buffer





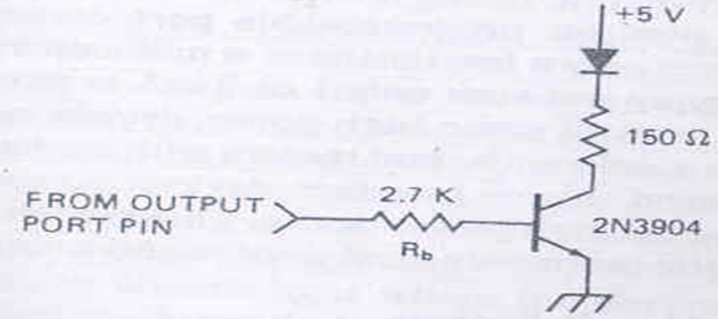
$$I_E = I_B + I_C$$

- Current Gain

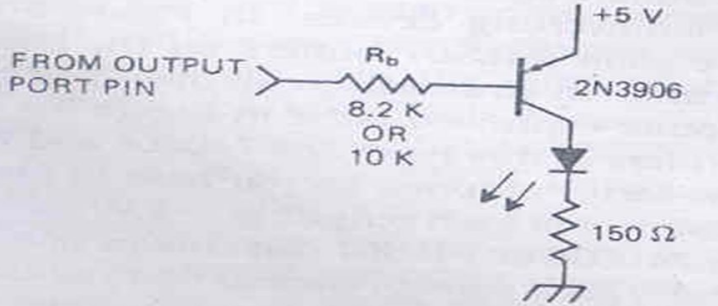
$$\beta = \frac{I_C}{I_B}$$

Transistor Buffer

- Transistor buffer **works as an amplifier**.
- Single Transistor circuits can be connected to microprocessor port lines.
- First of all need to determine whether we want a logic high or a logic low on the output port pin to turn on the device.
  - **Logic High- NPN circuits**
  - **Logic low- PNP circuit**
- Determine how much current we need to flow through the LED, Lamp or other device
- Current Gain of transistor is need to produce the required current.



(a)



(b)

FIGURE 9-35 Transistor buffer circuits for driving LED from 8255A port pin. (a) NPN. (b) PNP.

Math Problem

- Let we want to flow 20mA to flow through an NPN transistor to LED. If current gain is 50 (β) then what will be the base current?

- If the base voltage is 2V & base emitter drop is 0.7V, then what is the base resistor value ?

Darlington Pair

Darlington Pair

- If we need to switch currents **larger than about 50mA** on and off with a output port line, a single transistor does not have enough current gain to do this dependably.
- Solution: to **connect two transistors in a Darlington Configuration**
- Usually used to drive small solenoid valve
- **Output port pin** supplies the base current to **transistor Q1**
- This base current produces a collector current β times as large in Q1
- Emitter current of Q1– base current of Q2 and is amplified by the current gain of Q2
- Device acts like a single transistor-
- –Current gain= β Q1 * β Q2

Darlington Pair

$I_{c1}=\beta I_{b1}$

$I_{E1}=I_{c1}+I_{b1}$

$=\beta I_{b1}+I_{b1}$

Again,

$I_{b2}=I_{E1}$

$=\beta I_{b1}+I_{b1}$

$I_{c2}=\beta I_{b2}$

$=\beta(\beta I_{b1}+I_{b1})$

$I_{E2}=I_{c2}+I_{b2}$

$=\beta(\beta I_{b1}+I_{b1})+(\beta I_{b1}+I_{b1})$

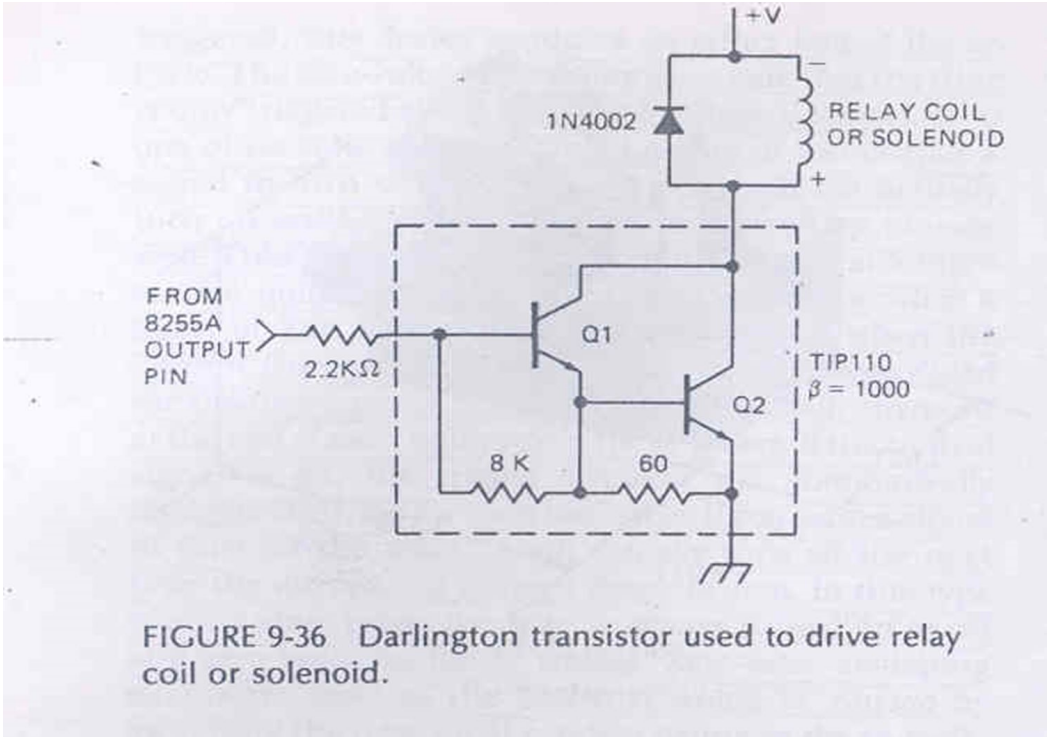


FIGURE 9-36 Darlington transistor used to drive relay coil or solenoid.

Inductive Kick

- A reverse biased diode is connected across the solenoid coil
- **Basic principle** of an inductor – it fights a change in the current through it
- When we apply a voltage to the coil by turning on the transistor- it takes a while for the current to start flowing (not a major problem)
- When we turn off the transistor – the collapsing magnetic field in the inductor keeps the current flowing for a while
- Cannot flow through the transistor(off)
- **Current develop a large reverse voltage across the inductor - called "inductive kick"**
    - Usually large enough to break down the transistor
- **When the coil is conducting , diode is reverse-biased**
- When the induced voltage reaches 0.7v – diode turns on & supplies a return path for the induced current
    - So the transistor is saved

Interfacing to AC Power Devices

Interfacing to AC Power Devices

- To turn 110/220/440 V ac devices on/off- used **mechanical or solid state relay**.
- **Relay control circuits** are electrically isolated from the actual switch – bakes ICs

Mechanical Relay

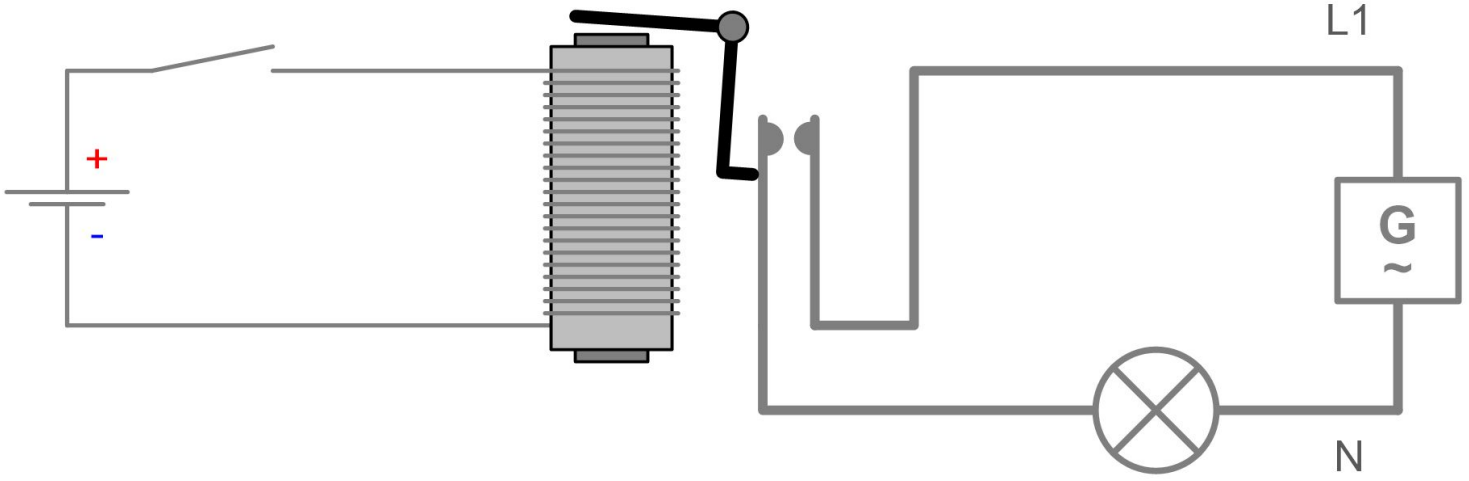
- Open and close contacts
- When current passed through coil, switch arm is pulled down- open top contacts and closes bottom contacts.

Disadvantages

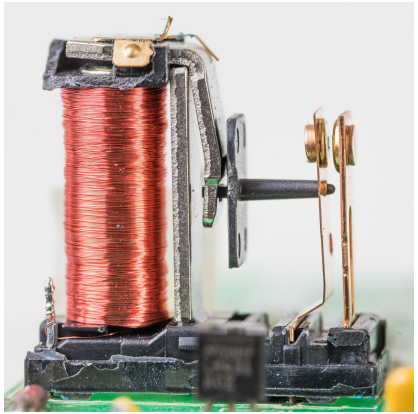
- Arcing, oxidized, higher resistance, may get hot enough to melt.
- Switch on/off at any point in the AC cycle – cause large noise – EMI(Electromagnetic Interference)

Solution -Solid state relay

Mechanical Relay



Mechanical Relay

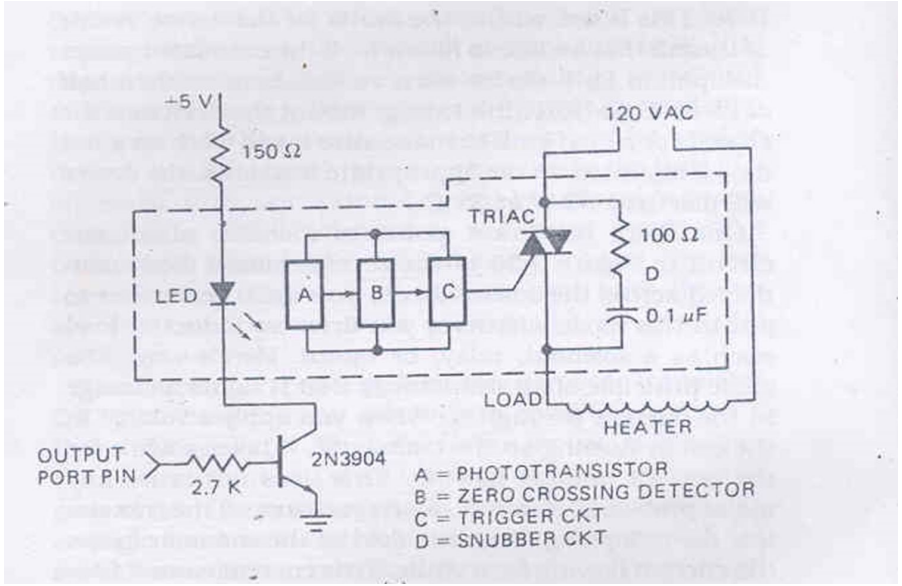


Internal Circuit of Solid State Relay

- Input circuit of **solid-state relay is just an LED**
- Required simple **NPN transistor** buffer and a current **limiting resistor to interface to microcomputer output port pin**
- Apply high on the port pin - turns on the transistor and pulls required current through the internal LED
- The LED light **focused on a phototransistor (A)** connected to the actual output-control circuitry.
- Actual switch of **SSR is triac**- conducts on either half of ac cycle when triggered.
- **Zero voltage detector** makes sure that the **triac is only triggered** when the ac line voltage is close to **zero-voltage crossing points**.

Internal Circuit of Solid State Relay

- Relay will not automatically turn on until the next time ac line voltage crosses zero
- Triac automatically **turn off** at **the end of each half cycle** when current drops below holding current (small value)
- In SSR, triac is turned on and off at zero point on the ac voltage.



https://commons.wikimedia.org/wiki/File:Triac_funktion.gif

Solid State Relay

Advantages

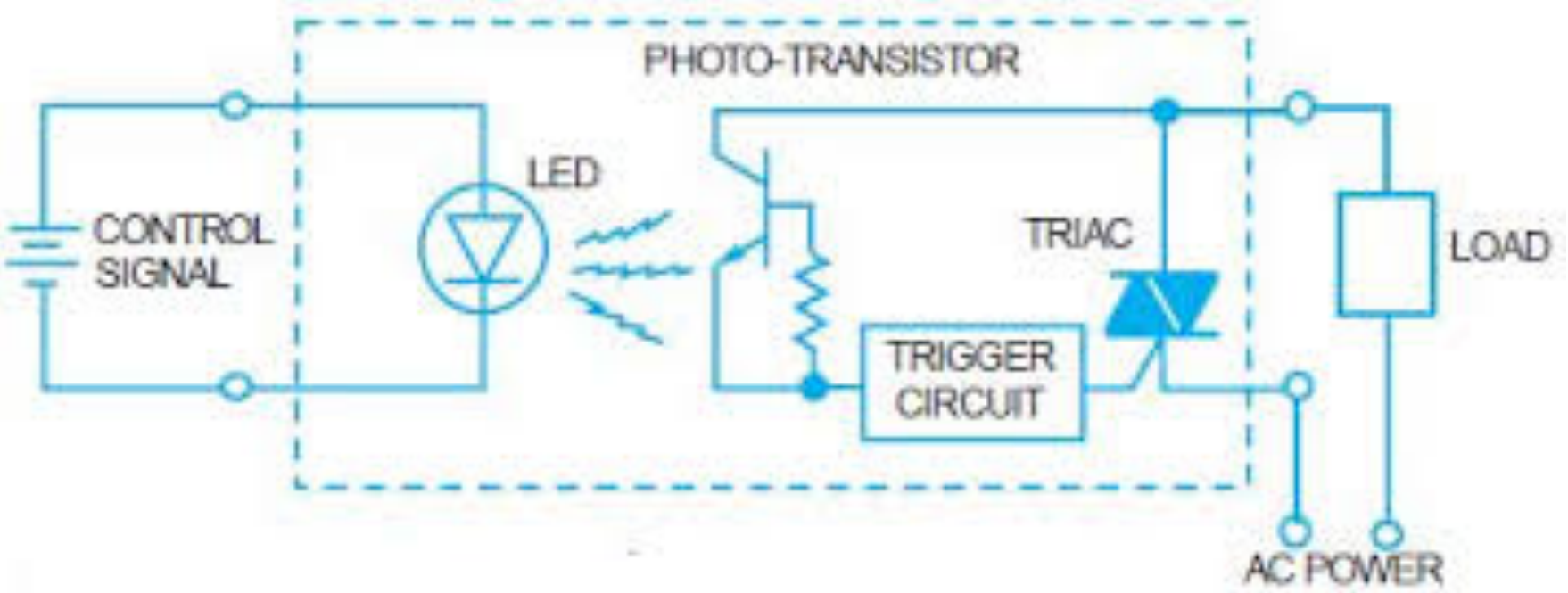
- Produce less EMI
- No mechanical contact-no arc
- Easily driven from microcomputer ports

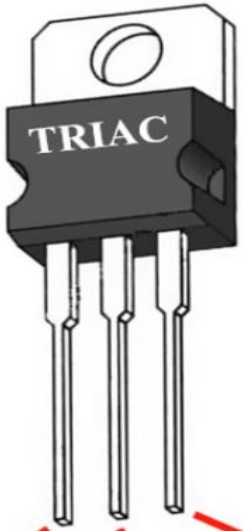
Disadvantages

- Expensive
- Significant Voltage drop across the triac.
- Jump to several voltage across triac for large inductive load when turn off triac.
- This large voltage may turn on the triac

RC snubber circuit (To keep the voltage and load current in phase)

Solid State Relay

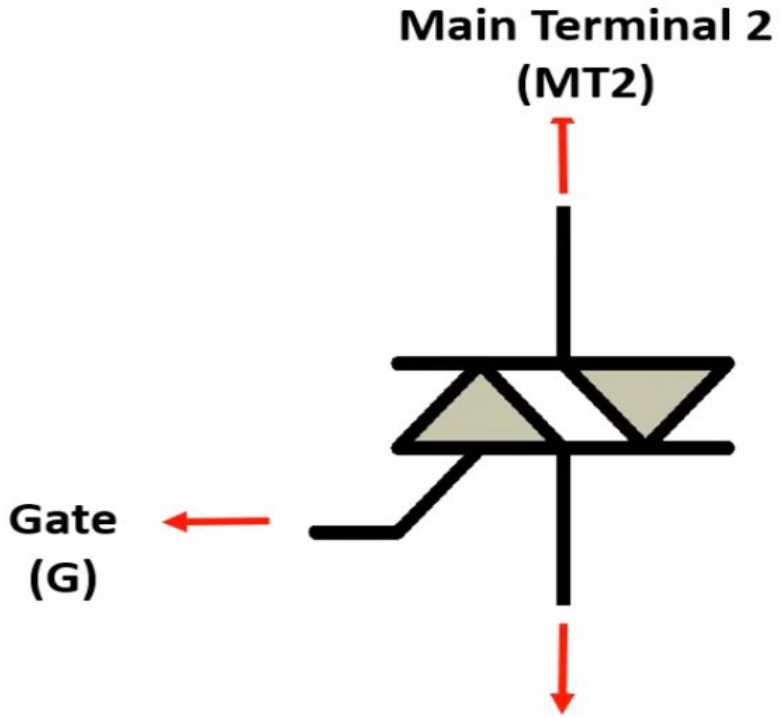


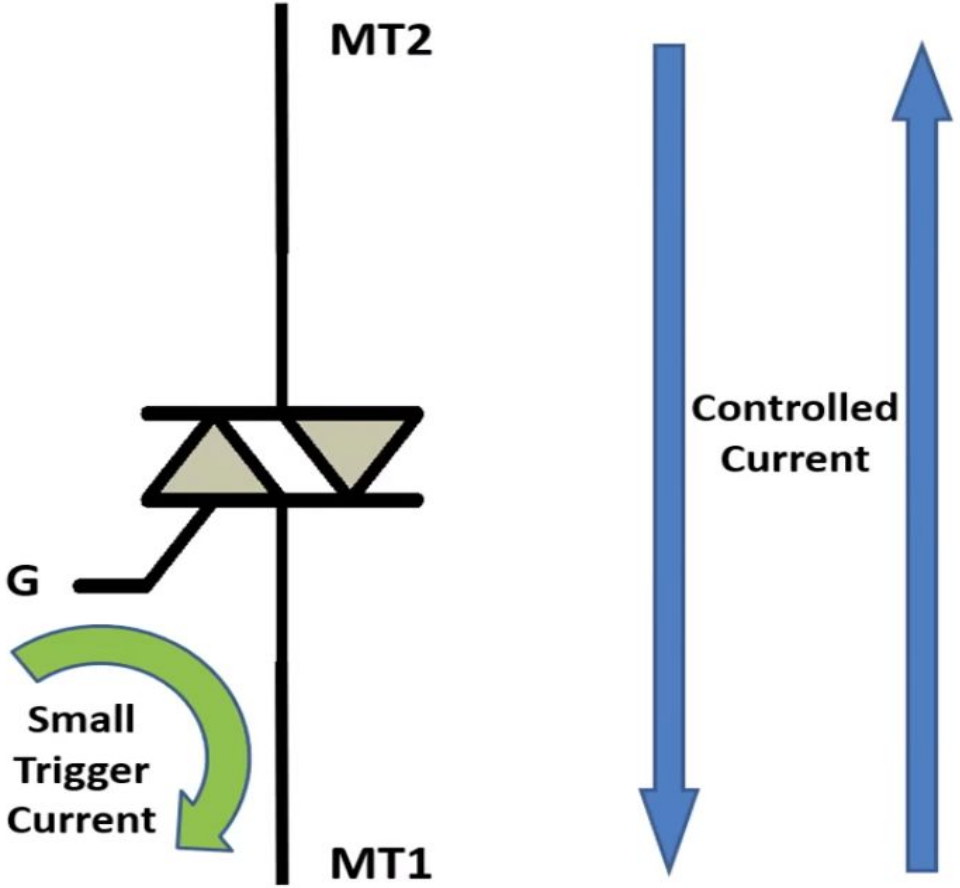


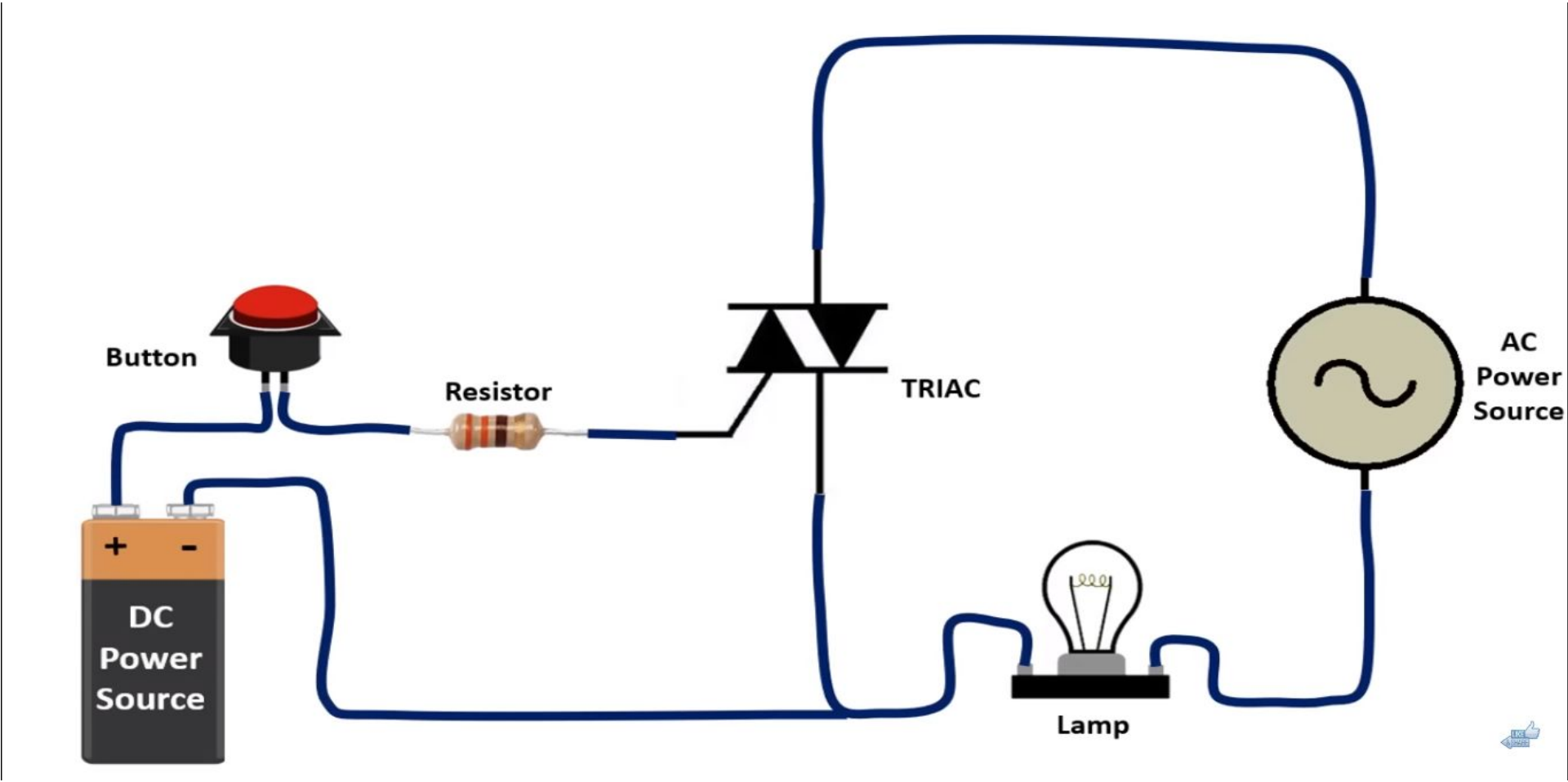
Main Terminal 1 (MT1)

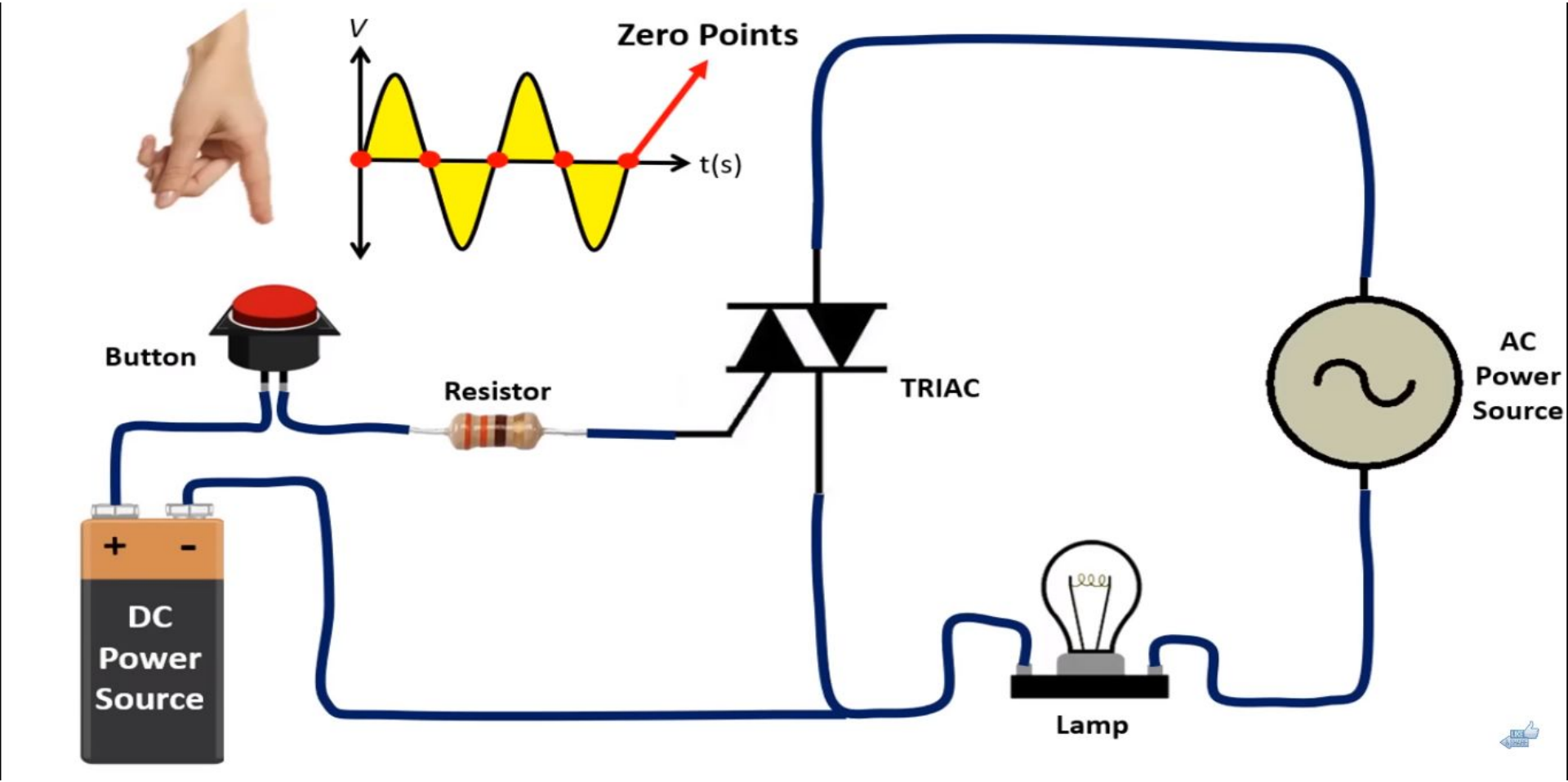
Main Terminal 2 (MT2)

Gate (G)




MT2
G
Small
Trigger
Current
MT1
Controlled
Current


Button
Resistor
TRIAC
AC
Power
Source
+
-
DC
Power
Source
Lamp


V
Zero Points
t(s)
Button
Resistor
TRIAC
AC
Power
Source
+
-
DC
Power
Source
Lamp

Interfacing to Stepper Motor

Interfacing to Stepper Motor

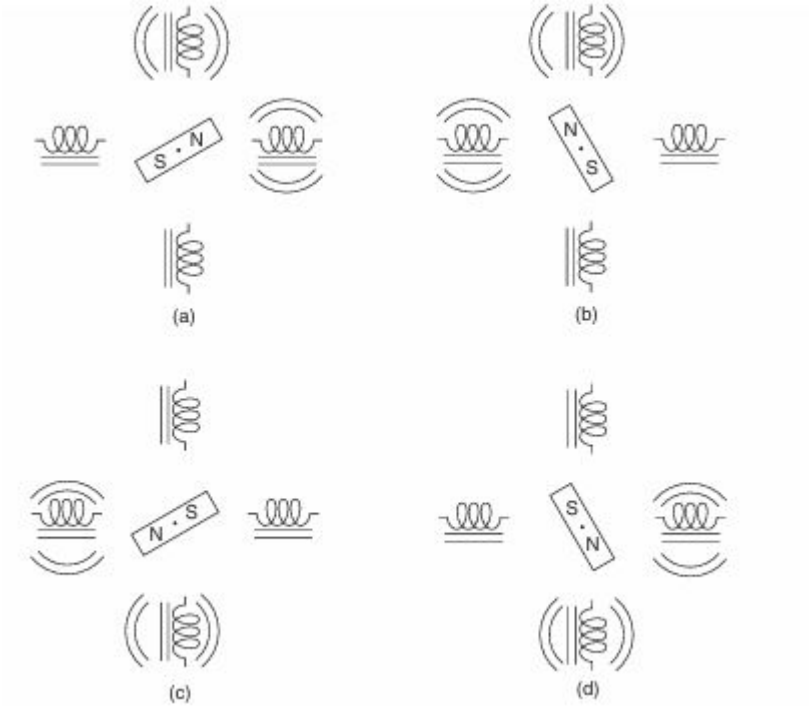
Stepper Motor:

- A stepper motor is a digital motor because it is moved in discrete steps as it traverses through 360°
- A common stepper motor is geared to **move perhaps 15°** per step in an inexpensive stepper motor
- **1° per step** in a more costly, high-precision stepper motor.
- **A four-coil stepper** motor uses **an armature** with a single pole.
- The stepper motor is **driven by using NPN Darlington amplifier** pairs to provide a large current to each coil.

Interfacing to Stepper Motor

Full Step:

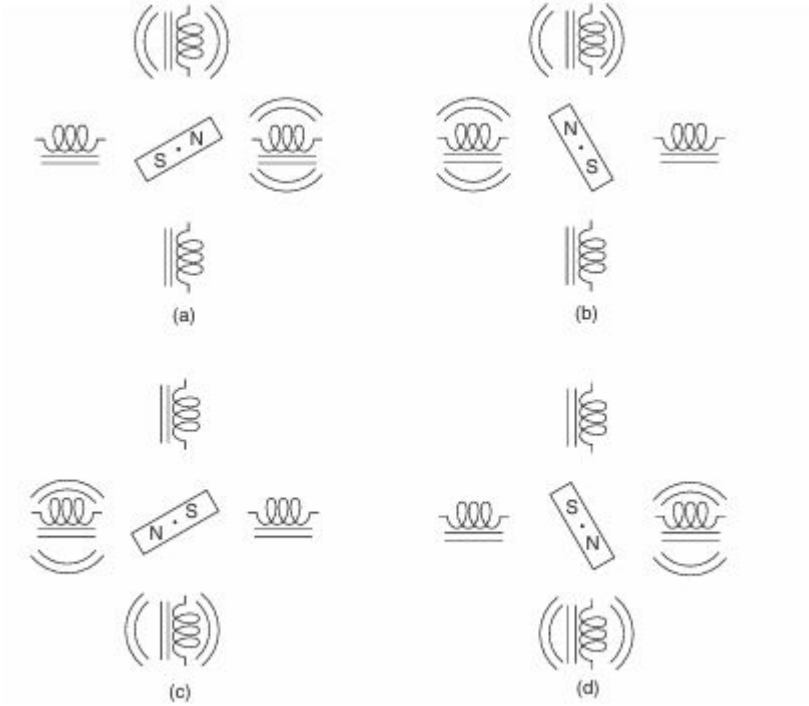
- The motor to step at 45°, 135°, 225°, and 315°
- Two coils are energized at a time

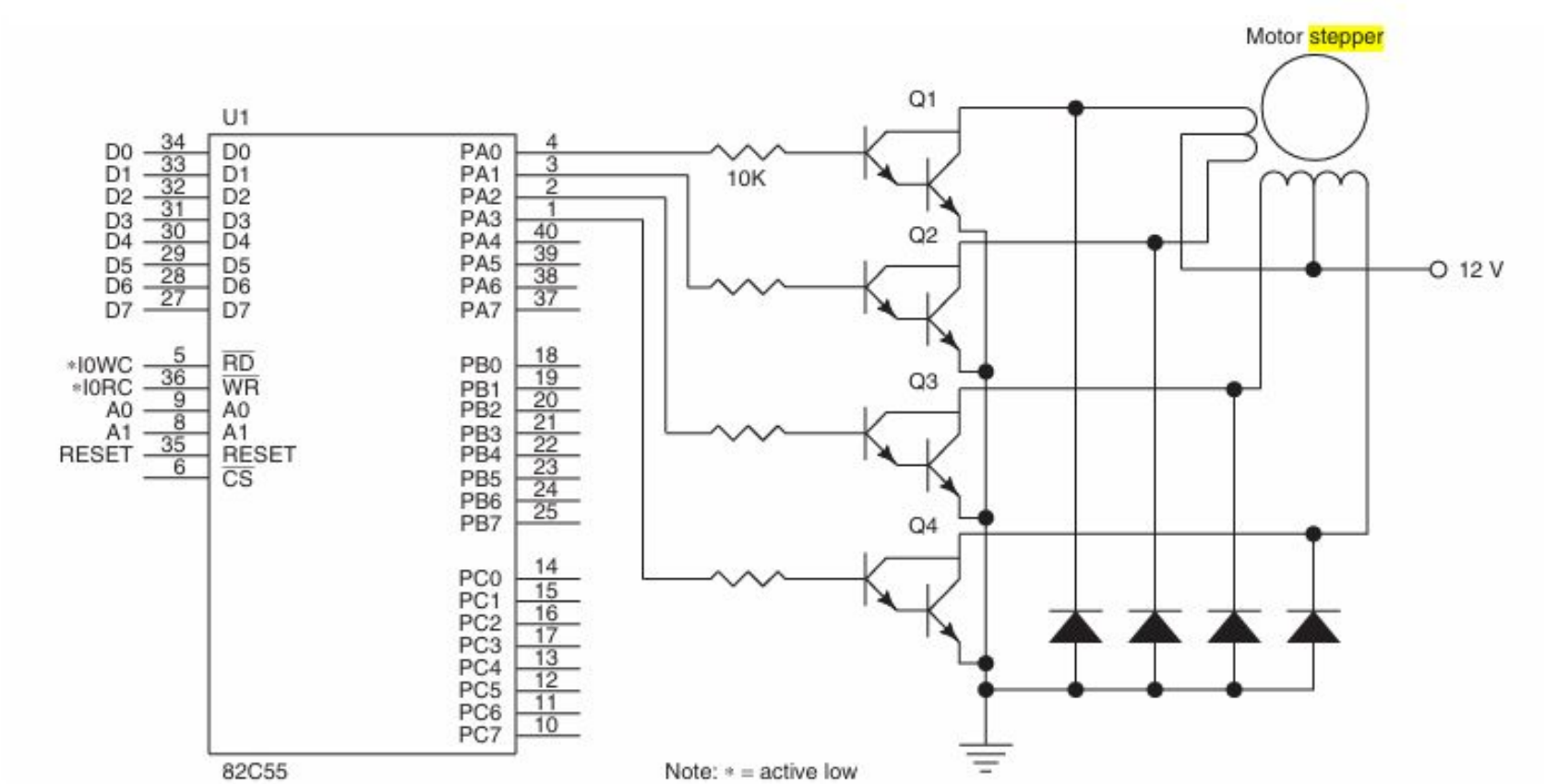


Interfacing to Stepper Motor

Half Step:

- Allows eight steps per sequence.
- This is accomplished by using the full-step sequence described with a half step obtained by **energizing one coil** interspersed between the full steps.
- Half-stepping allows the armature to be positioned at **0°, 90°, 180°, and 270°**.
- Total 8 steps:
  0, 45,90,135,180,225,270,315,360





https://content.instructables.com/ORIG/F2D/4UWB/IB22QT5X/F2D4UWBIB22QT5X.gif?format=mp4&fbclid=IwAR0_X5iPfXjiSbkri6P97vWlv89ZS8JAzkvf18VP3dt_UhgqIJfBUfycdck

STEP	SWITCH			
	SW4	SW3	SW2	SW1
1	0	0	1	1
2	1	0	0	1
3	1	1	0	0
4	0	1	1	0
1	0	0	1	1

1 = SWITCH ON

EIGHT-STEP INPUT SEQUENCE
(HALF-STEP MODE)

STEP	SW4	SW3	SW2	SW1
1	OFF	OFF	ON	ON
2	OFF	OFF	OFF	ON
3	ON	OFF	OFF	ON
4	ON	OFF	OFF	OFF
5	ON	ON	OFF	OFF
6	OFF	ON	OFF	OFF
7	OFF	ON	ON	OFF
8	OFF	OFF	ON	OFF
1	OFF	OFF	ON	ON

Interfacing to Stepper Motor

- A simple procedure that drives the motor (assuming that port A is programmed in mode 0 as an output device)
- CX holding the number of steps and direction of the rotation.
- **If MSB of CX is 1**, the motor spins in the r**ight-hand direction (CW)**
- If **MSB of CX is 0**, the motor spins in the **left-hand direction (CCW)**
- The remaining 15 bits contain the number of steps
- The procedure uses **a time delay** (not illustrated) that causes **a l ms time delay**
- Timing of damping command must be determined experimentally
- This time delay is required to **allow the stepper-motor armature time** to move to its next position..

Interfacing to Stepper Motor

- The current position is **stored in memory location POS**
- Initialized with 33H, 66H, 0CCH, or 99H
- This allows a simple **ROR (step right) or ROL (step left) instruction** to rotate the **binary bit pattern for the next step.**
- The half-step position codes are 11H, 22H, 44H, and 88H. A complete sequence of eight steps would be as follows: 11H, 33H, 22H, 66H, 44H, 0CCH, 88H, and 99H.

Interfacing to Stepper Motor

EXAMPLE 11–16

```
        PORT   EQU   40H

;An assembly language procedure that controls the stepper motor

STEP    PROC   NEAR USES CX AX

        MOV   AL,POS                        ;get position
        OR    CX,CX                         ;set flag bits
        IF !ZERO?
              .IF !SIGN?                    ;if no sign
                    .REPEAT
                         ROL   AL,1         ;rotate step left
                         OUT   PORT,AL
                         CALL DELAY         ;wait 1 ms
                    .UNTILCXZ
              .ELSE
                    AND CX,7FFFH            ;make CX positive
                    .REPEAT
                         ROR   AL,1         ;rotate step right
                         OUT   PORT,AL
                         CALL DELAY         ;wait 1 ms
                    .UNTILCXZ
              .ENDIF
        .ENDIF
        MOV POS,AL
        RET

STEP    ENDP
```

Thank You

CSE 405
(Computer Interfacing)

USB

(Microprocessors and Interfacing by Hall)

Introduction

- The **Universal Serial Bus (USB)** is a serial I/O interface between peripherals and computers.
- Usage: personal computers, consumer electronics, mobile products.
- Connects devices like a **mouse, keyboard, PDA, gamepad, joystick, scanner, camera, printer, hard disk drive, flash drive** etc. to a computer.
- Initially supported a maximum speed of **12 Mbps**. Currently supports **4 different speeds**:
  - **A low speed of 1.5Mbps** for low bandwidth devices like keyboards, mouse, and joystick.
  - **The full speed of 12 Mbps.**
  - **A hispeed of 480 Mbps.**
  - **A superspeed of 4.8Gbps.**

Advantages

1. Simple Connectivity
2. No need for switch setting
3. Supports variety of data, from slow mouse inputs to digitized audio and compressed data.
4. Permit **hot swapping**; devices can be plugged on or unplugged without powering off the system.
5. Support a large number of devices, up to **127 devices** can be used on a USB port
6. The **OTG features** allow **cell phones and digital cameras** to be directly connected to USB peripherals.

USB Versions

USB1.0:

- allows transfer at **12 Mbps**.
- known as **full-speed USB**
- supports a wide range of devices

USB1.1:

- supports **two modes:** full speed mode of **12 Mbps** and low speed mode of **1.5 Mbps**.

USB Versions

USB2.0:

- supports **three modes**: **hispeed of 480 Mbps**, **full speed mode** of 12 Mbps and **low speed mode** of 1.5 Mbps.
- **supports** both low bandwidth devices like **mouse, keyboard,** joystick; and **high bandwidth devices** like high resolution **web camera, printer, scanner.**
- known as hi speed USB
- The **USB On-The-Go (OTG)**, a supplement of USB2.0, supports **dual role devices** that **can perform as peripherals** and **hosts**. So, **two USB devices can communicate directly**.

USB OTG

- USB OTG (On-The-Go) is a specification that allows a USB device, such as a **smartphone or tablet, to act as a host**
- Enables it to connect to other **USB devices like keyboards, mice, flash drives**, or even other smartphones.
- Typically, these mobile devices act as USB peripherals when connected to a computer.
- With USB OTG, they can also become a host, controlling other USB devices.
- To use USB OTG, **a special OTG cable or adapter is required**.

USB Versions

- **USB3.0**:
  - supports data transfer rates of 4.8 Gbps.
  - known as superspeed USB
- **Wireless USB**:
  - uses ultra-wideband wireless technology for data rates up to 480 Mbps.

USB principles of operation

- **The host** (USB master) **manages** all communications.
- **Peripheral devices** respond to commands received from the host.
- USB uses **7-bit addresses**.
- Supports up to **127 devices** except all zero code.
- Each **physical USB device** can consist of multiple **logical USB devices** identified as device functions.
- Each **logical device is allotted a unique address by host**.
- For example, **a webcam** with a **built-in microphone** has **two functions** with distinct addresses.

Device Classes

- USB classifies the devices and assigns class codes.
- The class code denotes the class identity for proper driver usage.

Table 13.9 Use typical device classes

Class (Hexadecimal)	*Function*	*Description*	*Examples*
01	Interface	Audio	Speaker, microphone, sound card
02	Both interface and device	Communications and CDC Control	Ethernet adapter, modem, serial port adapter
03	Interface	Human Interface Device (HID)	Keyboard, mouse, joystick
05	Interface	Physical Interface Device (PID)	Force feedback joystick
06	Interface	Image	Webcam, scanner
07	Interface	Printer	Laser printer, inkjet printer, CNC machine
08	Interface	Mass Storage	USB flash drive, memory card reader, digital audio player, digital camera, external drive
09	Device	USB hub	Full speed hub, hi-speed hub
0A	Interface	CDC-Data	This class and class 02 (Communications and CDC Control) are used together.
0B	Interface	Smart Card	USB smart card reader
0E	Interface	Video	Webcam
E0	Interface	Wireless Controller	Wi-Fi adapter, Bluetooth adapter

USB system Topology

- The USB system consists of **a host**, **USB hubs**, and **peripheral devices**
- USB devices are connected through hubs.
- The **host controller and the root hub** are part of the **computer hardware**.
- A **USB host can have multiple host controllers** and **each host controller can have multiple ports.**

USB Hubs:

- Routes data from the host to its correct destination
- Detect the topology changes when connecting or or removing any peripherals.
- Source the power to the USB network.

Host controllers:

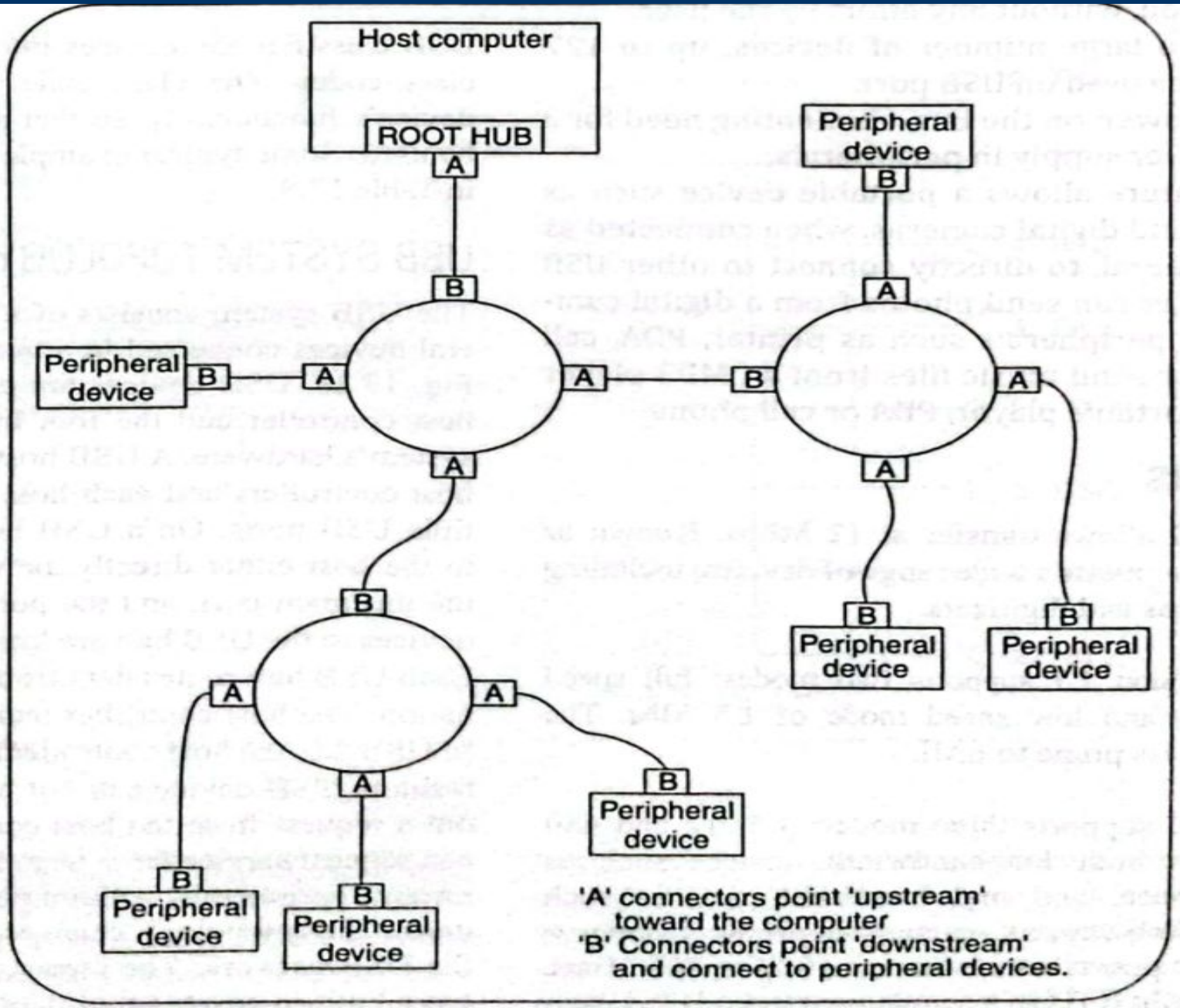
- Manages **traffic flow to devices**
- USB devices can not transfer data without a request from host controllers.

USB system Topology

Two types of ports

- **Upstream Port: On a USB hub**, the port connected to the host either directly or via another hub.
- **Downstream Port: The ports used for connecting other devices** to the USB hub.

USB 'Tiered Star' Topology



USB Packet Format

ADDR Field: address of the destination device.

Sync Field:

- synchronizes the clock of the receiver with the transmitter.
- 8 bit long (low and fullspeed)
- 64 bit long (Hispeed)

PID Field:

- identifies the **type of packet**
- 4-bit PID is used.
- **bitwise complement is added then**
- Total 8-bits are used.
- Redundancy helps **error detection.**

USB Packet and Format

ENDP Field:

- denotes endpoint
- 4-bit long
- possible 16 devices.

CRC Field:

- Cyclic Redundancy Check
- 5 bit long (Token Packet)
- 16 bit long (Data Packet)

USB Packet Type

Types of packet:

- Handshake packet
- Token Packet
- Data Packet
- PRE Packet
- Start of Frame Packet

Handshake Packet

- A data related packet
- Sent by both **host and peripheral, whichever is the receiver**
- Sent **in response to a data packet**
- **Three basic types** (typically), two additional in USB 2.0

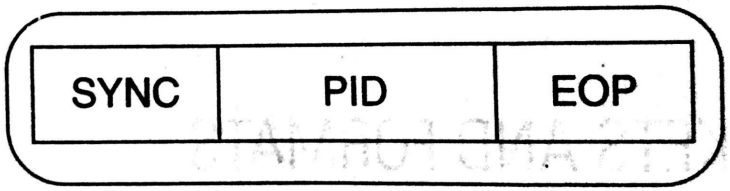


Fig. 13.29 Handshake packet.

Handshake Packet

Table 13.10 Handshake Packets

Sl. No	*Handshake Packet type*	*Meaning*
1	ACK	Data packet successfully received.
2	NAK	Data packet not received, and should be retransmitted.
3	STALL	Error in device; corrective action is required.
4	NYET	A split transaction is not yet complete.
5	ERR	A split transaction failed.

Split Transaction

- A large data transfer is broken down into smaller segments.
- This allows the USB system to handle data that exceeds the capacity of a single transaction
- **Three stages of Split Transactions:**
  - **Transaction Stage:** This is where the initial request for data transfer is made.
  - **Split Stage:** This involves **breaking down the data into smaller chunks** and **scheduling these chunks for transfer**.
  - **Completion Stage:** The completion stage ensures that all data chunks are successfully transmitted and assembled in the correct order.

Token Packet

- **8 bit PID**, 5 bit CRC
- **tokens sent by HOST** (not device)
- **Three types;** USB2.0 added more

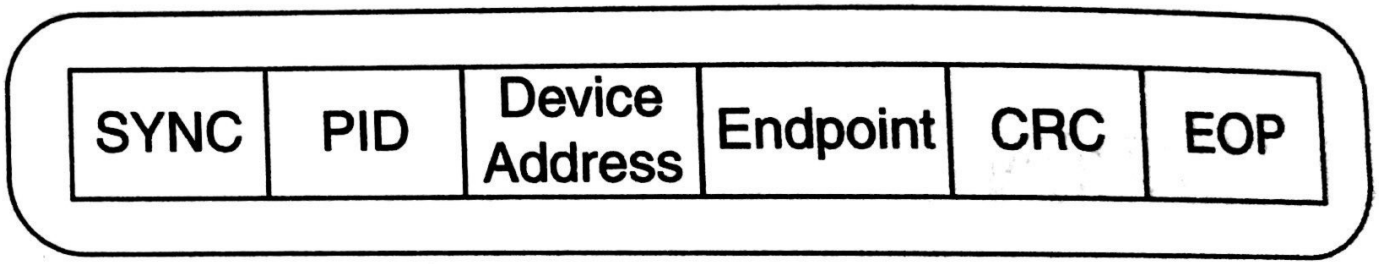


Fig. 13.30 Token packet.

Token Packet

Table 13.11 Token packets

Sl. no.	Token Packet type	Meaning/Function
1	In token	Informs the USB device that the host wishes to read information
2	Out token	Informs the USB device that the host wishes to send information
3	Setup token	Used to begin control transfers.
4	PING token.	Asks if device is ready to receive.
5	SPLIT token	Similar to OUT token; used for initial device setup.

Token Packet

In token:

- Sends when direction of data transfer from device to host
- Expects response from device
- Response can be **NAK, STALL or a DATA packet**
- If response is a dataframe, then **host issues an ACK handshake** if appropriate.

OUT token:

- Sends when direction of **data transfer from host to device**.
- **OUT token is immediately followed by a data frame**.
- The device responds with **ACK, NAK or STALL**

SPLIT token:

- Used to perform split transactions when communicating with **slow devices**
- A SPLIT token is given to the nearest high speed hub.
- **It then transfers data at full or low speed to the device**.

Data Packet

- PID, **0-1023 bytes of data payload**, 16 bit CRC
- Data can be up to 1024 byte in Hispeed, maximum 8 byte low speed.
- **Two basic types**; USB2.0 added more two

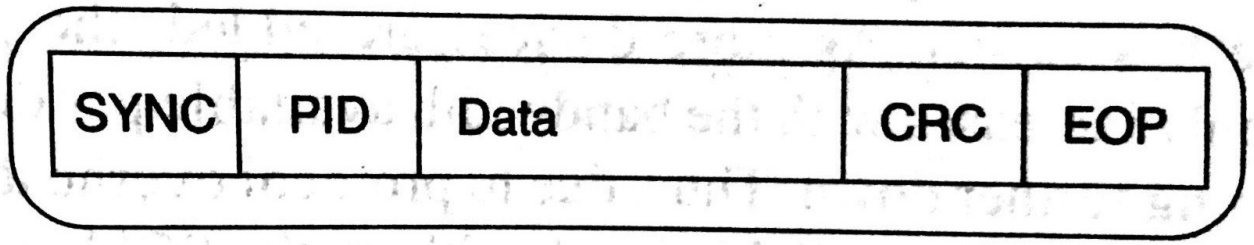


Fig. 13.31 Data packet.

Data Packet

Table 13.12 Data packets

Type	*Name*	*Description*
Data	DATA0	Even-numbered data packet
	DATA1	Odd-numbered data packet
	DATA2	Data packet for high-speed isochronous transfer (USB 2.0)
	MDATA	Data packet for high-speed isochronous transfer (USB 2.0)

If USB host does not receive any ACK:

- Lost data frame
- Lost ACK frame

Data Packet

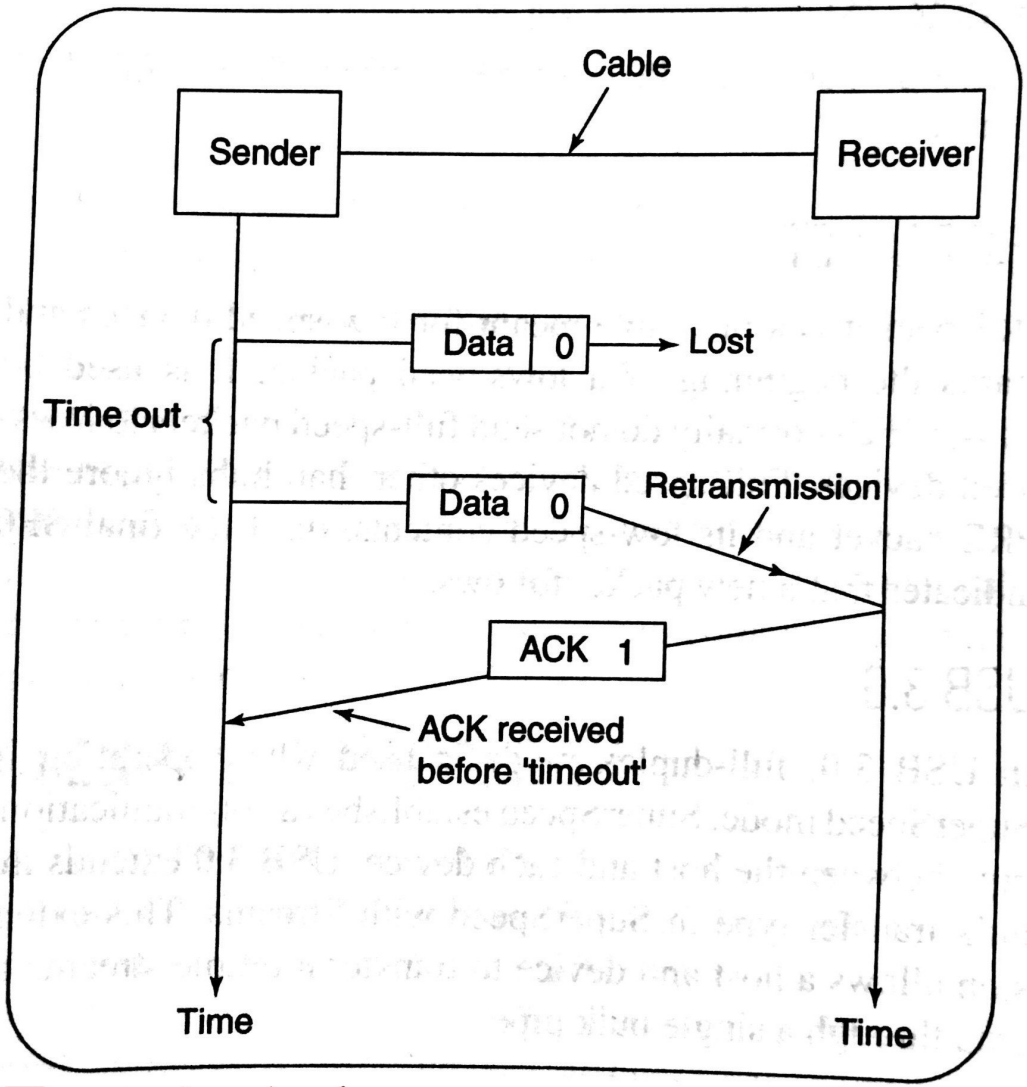


Fig. 13.32 Lost data frame.

Data Packet

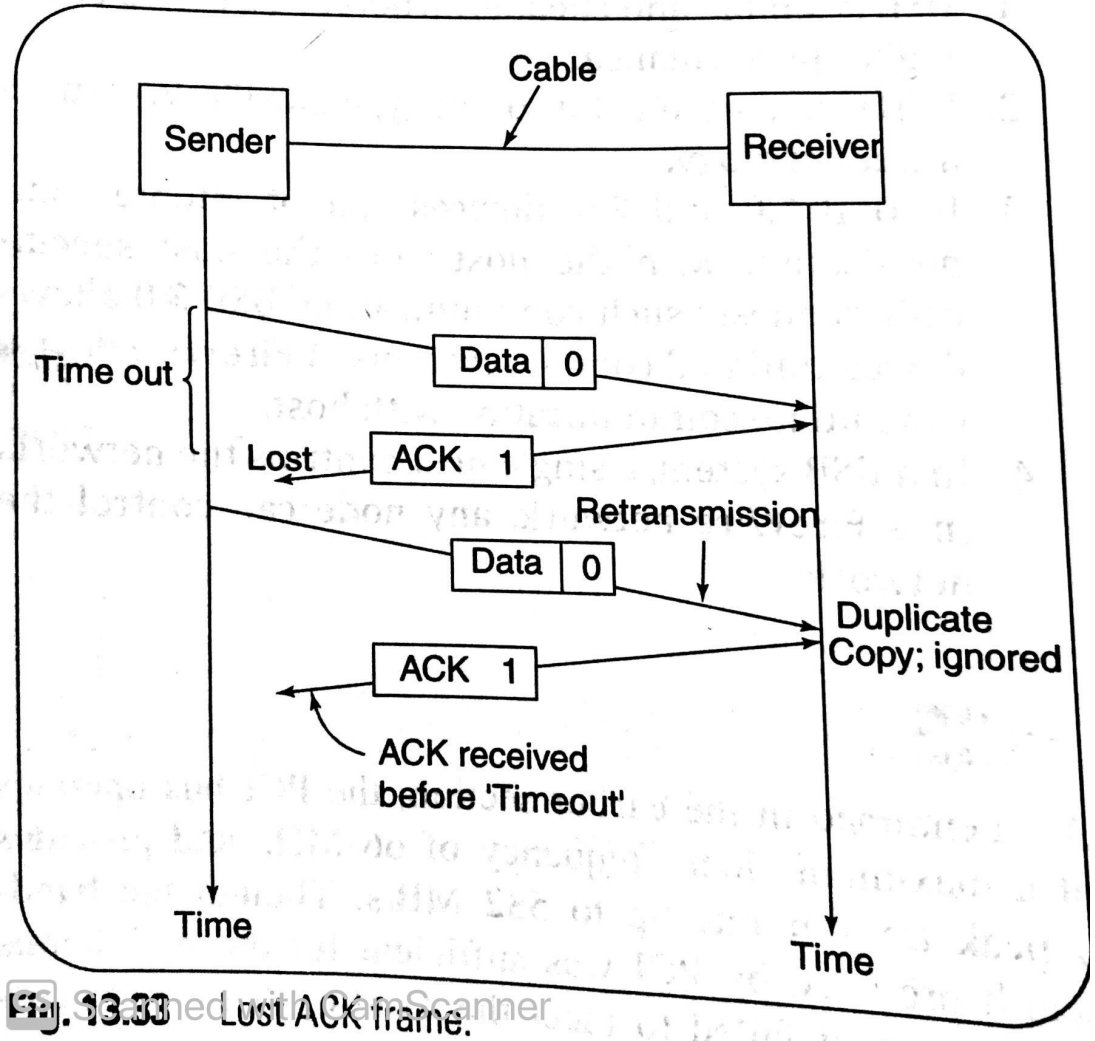


Fig. 13.33 Lost ACK frame.

Start of Frame Packet

- USB host transmits a special SOF token at every 1 ms.
- 11 bit incrementing frame number
- Synchronizes **isosynchronized** data flow.

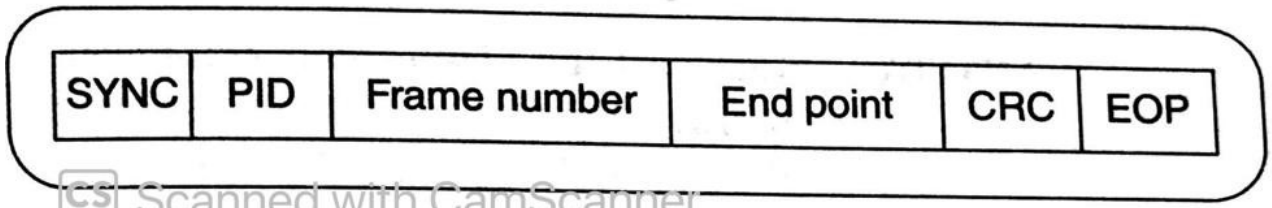


Fig. 13.34 Start of frame packet.

PRE Packet

- Preamble meant for low speed devices and marks the beginning of a low speed packet.
- Used by hubs
- Full speed devices other than hubs ignore PRE packets

End of Slides