

Serial / Parallel Communication Adapter

Project Work

SUBMITTED BY

P-1299

S. Krishnakumar

P. M. Mushtaq Amir

V. Richard

A. Sivakumar

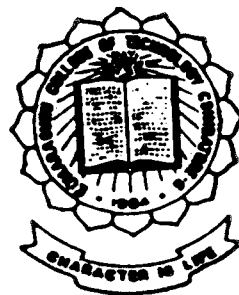
Under the Guidance of

Mr. K. Ramprakash, M. E.

In partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF ENGINEERING

in Electronics and Communication Engineering



Department of Electronics and Communication Engineering
Kumaraguru College of Technology
Coimbatore - 641 006

1994



SOFT LINK SERVICES

67 D, Z-150, 5th AVENUE, ANNA NAGAR, MADRAS-600 040.

Ref.:

Date:

REF:SLS/ECERT/KCT/94
DATE: 14/03/94

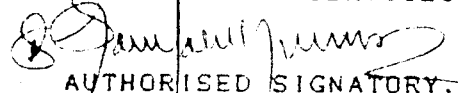
TO WHOMSOEVER IT MAY CONCERN

This to Certify that the following students of the KUMARAGURU COLLEGE OF TECHNOLOGY.COIMBATORE-6.have been doing their project viz:IBM Compatible PC based Serial/Parallel adapter card . in our premises utilising our Computer and other test facilities.Their performance have been good and impressive.

ROLL NUMBER	NAME
-----	-----
1) 90ECE11	S.KRISHNA KUMAR
2) 90ECE17	P.M.MUSTAQ AMEER
3) 90ECE23	V.RICHARD
4) 90ECE29	A.SIVA KUMAR

PERIOD OF EXECUTION:15 FEB'1994 TO 12 MARCH 1994.

for SOFT LINK SERVICES


AUTHORISED SIGNATORY.

ACKNOWLEDGEMENT

First and foremost, we would like to thank our respected principal **Dr.S.Subramanian B.E., M.Sc(Engg)., Ph.D** for giving his consent to do the project.

We are greatly thankful and express our heartfelt gratitude to our beloved Head of the Department **Rtn. Prof. K.Palaniswami M.E., M.I.E.E.E., F.I.E.T.E.**, for his constant encouragement and inspiration throughout our project work.

We are also thankful and consider it a privilege to work under the guidance of our beloved guide **Mr.K.Ramprakash M.E., senior lecturer, who has guided us and had been a source of inspiration.**

We would be failing in our duty if we do not thank our **lab technicians** who had extended their kind co-operation.

AUTHORS

SYNOPSIS

The **project** on **SERIAL/PARALLEL COMMUNICATION ADAPTER** is to enable communication between two computers through a cable. This card is connected to the I/O bus of the system mother board. It converts the parallel data from the microcomputer I/O bus into serial channel. Data can also be received serially which are converted into parallel bits. Another circuit incorporated into this adapter is parallel interface which is exclusively used for printers in IBM PC's.

The heart of the adapter card is the **UART** chip and the single custom IC, designed exclusively for **CENTRONIX INTERFACE**. The former is used for serial communication and latter is used for parallel communication. Asynchronous communication is used in serial data transmission and for which the standard used is EIA RS232-C. In parallel communication centronix interface is used.

A PAL chip has been used to give high performance and has replaced the conventionally used logic gates and flip flops. The PAL device is a programmable AND array driving a fixed OR array. This PAL chip is programmed according to specification.

The software design is a user friendly menu driven utility which has 2 major things:

1:File Transferring.

2:Remote Printing.

C O N T E N T S

Chapter		Page No.
	ACKNOWLEDGEMENT	
	SYNOPSIS	
1	INTRODUCTION	1
2	SYNCHRONOUS / ASYNCHRONOUS COMMUNICATION	4
3	OPERATING MODES OF UART	9
4	SERIAL PARALLEL COMMUNICATION USING UART- 8250B	14
5	SERIAL INTERFACE STANDARDS	23
6	ARCHITECTURE AND ORGANISATION OF IBM PC FAMILIES	28
7	COMMUNICATION SOFTWARE	50
8	APPLICATIONS	126
9	FUTURE DEVELOPMENT	128
10	A SUM UP	129
11	BIBLIOGRAPHY	130
12	APPENDIX	132

INTRODUCTION

Computers have become indispensable and play an important role in our day to day life. They serve many purpose and are recently being used in advanced fields like **communications** and **networking**. Thus in this situation it will be advantageous to design a circuit that can be used for communication between two computers.

The **SERIAL/PARALLEL COMMUNICATION ADAPTER** was developed to serve this purpose. It can enable a parallel computer to communicate with another parallel computer through serial data transfer. This **INTERFACE CARD** has been implemented by using both hardware and software.

It has the hardware feature for transmitting and receiving data between two computers and parallel transmission to the printers. The **SERIAL INTERFACE CIRCUIT** consists of two ports by which simultaneous transmission and reception is possible.

There is no well defined format for synchronous or asynchronous transmission but if timing signal is to coordinate the activity between the device and the interface the transmission would be considered synchronous. If only hand shaking signals are used, transmission is asynchronous.

SERIAL COMMUNICATION

Within a micro-computer data is transferred in parallel which is the fastest method to do. For transmission over long distances parallel transmission requires too many wires. Therefore data to be sent for long distances is usually converted from parallel form to serial form so that it can be sent on a single wire or a pair of wires. Serial data received from a distant source is converted to parallel form so that it can be easily transferred to the computer.

Three terms encountered in serial data systems are **SIMPLX, HALF DUPLEX, FULL DUPLEX.**

1) A simplex data line can transmit data only in one direction.

2) In half duplex, communication takes place in both directions at a time.

3) In full duplex each system can send and receive data at the same time.

There are two basic communication standards namely

- 1) **ASYNCHRONOUS** and
- 2) **SYNCHRONOUS**.

SYNCHRONOUS AND ASYNCHRONOUS COMMUNICATIONS :

In asynchronous each data character has a bit which identifies its start, and 1 or 2 bits which identifies its end. Since the character is identified individually, characters can be sent at any time.

In synchronous communication characters are allowed to be sent back to back. But this method should include special sync characters at the beginning of each message and special idle characters in the data stream to fill up time when no information is being sent. Here they are all sent at a constant rate.

ASYNCHRONOUS COMMUNICATION :

The sending and receiving systems are synchronised using some kind of auxillary signal so that both ends of a connection are always in step. This **synchronous** communication technique is used primarily in mainframe systems.

The alternative is to provide markers in the data train to indicate where every distinguishable block of data begins and ends. The receiving system can then sort out the proper beginning and avoid confusion without any synchronisation. Such systems are described as being **asynchronous** and are the operating basis of personal computer serial ports.

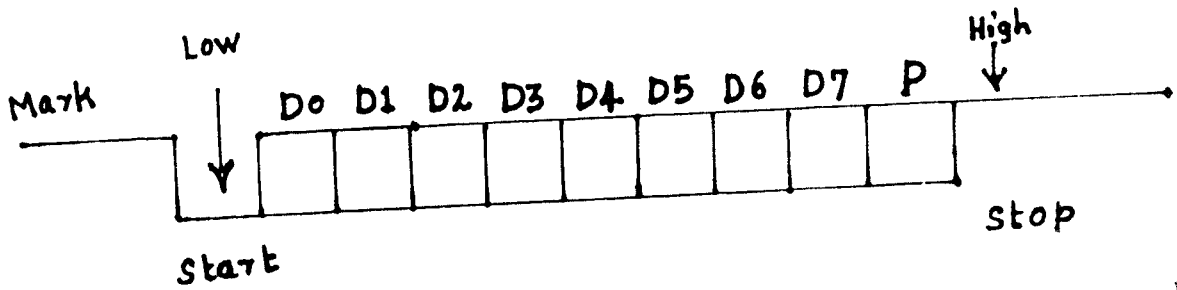
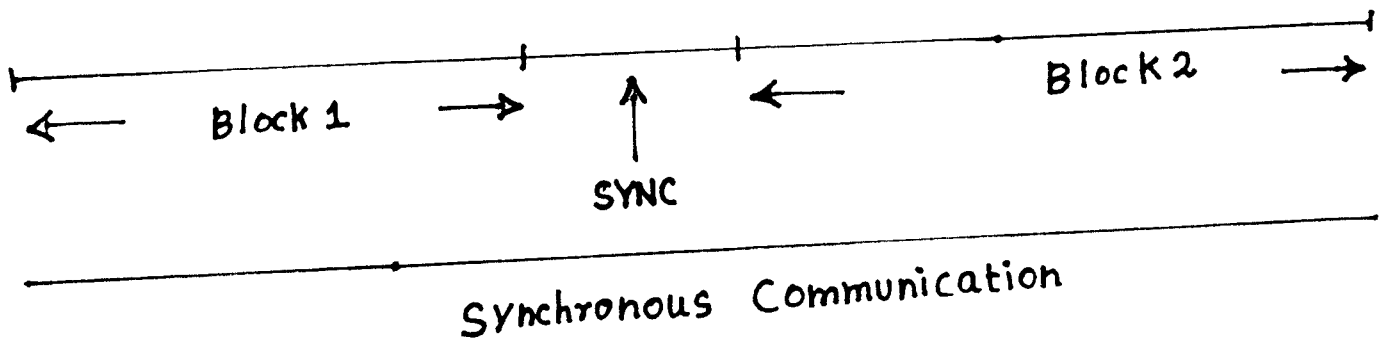
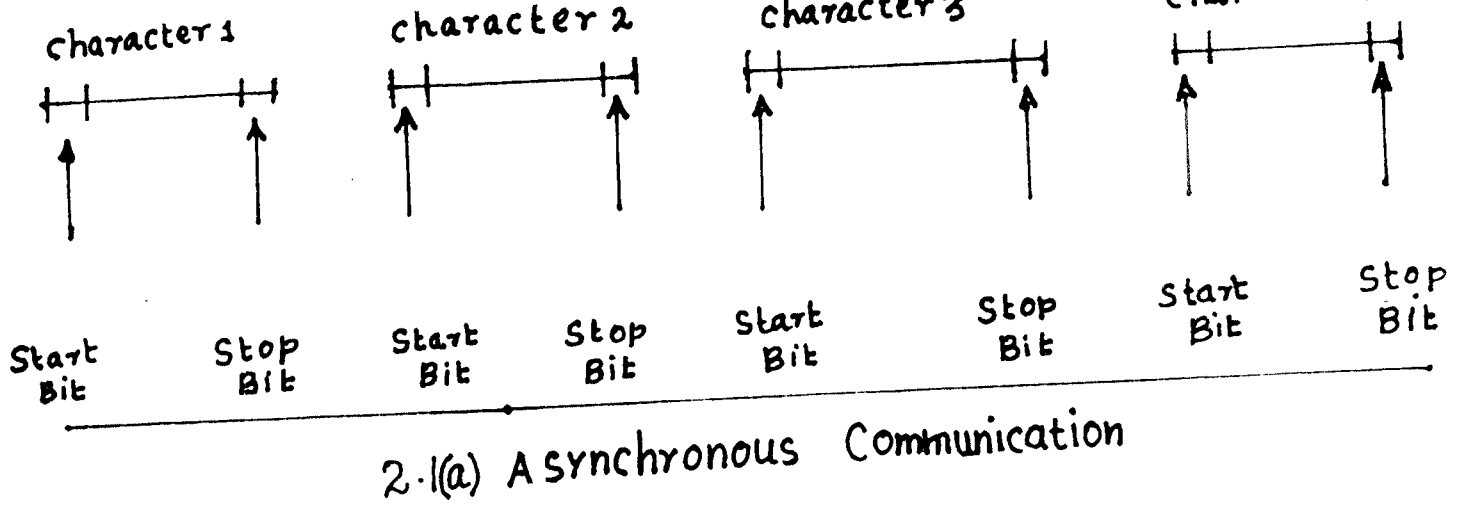
In most asynchronous systems, the data is broken up into small pieces, each roughly-corresponding to one byte. Each of these chunks is called a **word**, and may consist of five to eight data bits. The most widely used word lengths are seven and eight bits, the former because it accommodates all upper and lowercase text characters in ASCII code; the latter because each word corresponds to one data byte.

As serial data, the bits of a word are sent one-at-a-time down the communication channel. As a matter of convention, the least significant bit of the word is sent out first. The rest of the bits follow in order of their increasing significance.

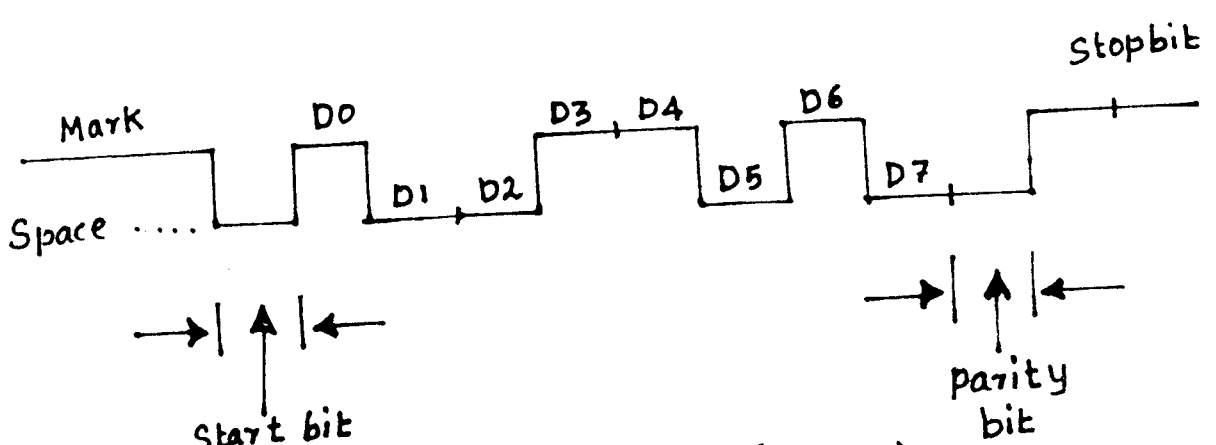
Added to these data bits is a very special double-length pulse called a **start** bit, and it indicates the beginning of a data word. One more bit known as the **stop** bit indicates the end of the word. The arrangement of the start and the stop bit in each of the character is given in fig (2.1a) Between the last bit of the word and the first stop bit a **parity** bit is often inserted as a data integrity check. Together the data bits, the start bit, the parity bit, and the stop bits make up one data **frame**.

Five kinds of parity bits can be used in serial communication, two of which actually offer error detection. This error detection works by counting the number of bits in the data word and determining whether the result is even or odd. In **odd parity**, the parity bit is sent on when the number of bits in the word is odd. **Even parity** switches on the parity bit when the bit total of the word is even. The total bit pattern consisting of all the bits including the start, stop, and parity bits is clearly represented in the fig (2.2a).

In **mark parity** the parity bit is always on, regardless of the bit total of the word. **Space parity** always leaves the parity bit off. **No Parity** doesn't even leave space for a Parity bit.



2.2(a) character format.



(b) character (Hexa A6)

All of these bits are sent down the serial line as **negative-going** pulses superimposed on the normal positive voltage on the data line. That is, the presence of a bit in a serial word will be a negative pulse. Compared to normal logic systems, RS-232 standard data looks upside down. There's no particularly good reason for the inversion except that it's the way things have always been done and, when it comes to communications, things work best when everybody sticks to the same standard.

Serial signals are also described by the nominal rate at which the bits in the serial train are sent. The standard form of measurement is simple-the number of bits per second that are sent-with the standard unit being one bit per second or bps.

For somewhat arbitrary reasons, bit rates are enumerated in a rather odd increment. The usual minimum speed is 300 bps, although slower submultiples of 50, 100, 150 bps are available. Faster standard speeds merely double the preceding rate, so the sequence runs 600, 1200, 2400, 4800, 9600, to 19,200, the fastest personal computers, the PS/2 Models 50 and above.

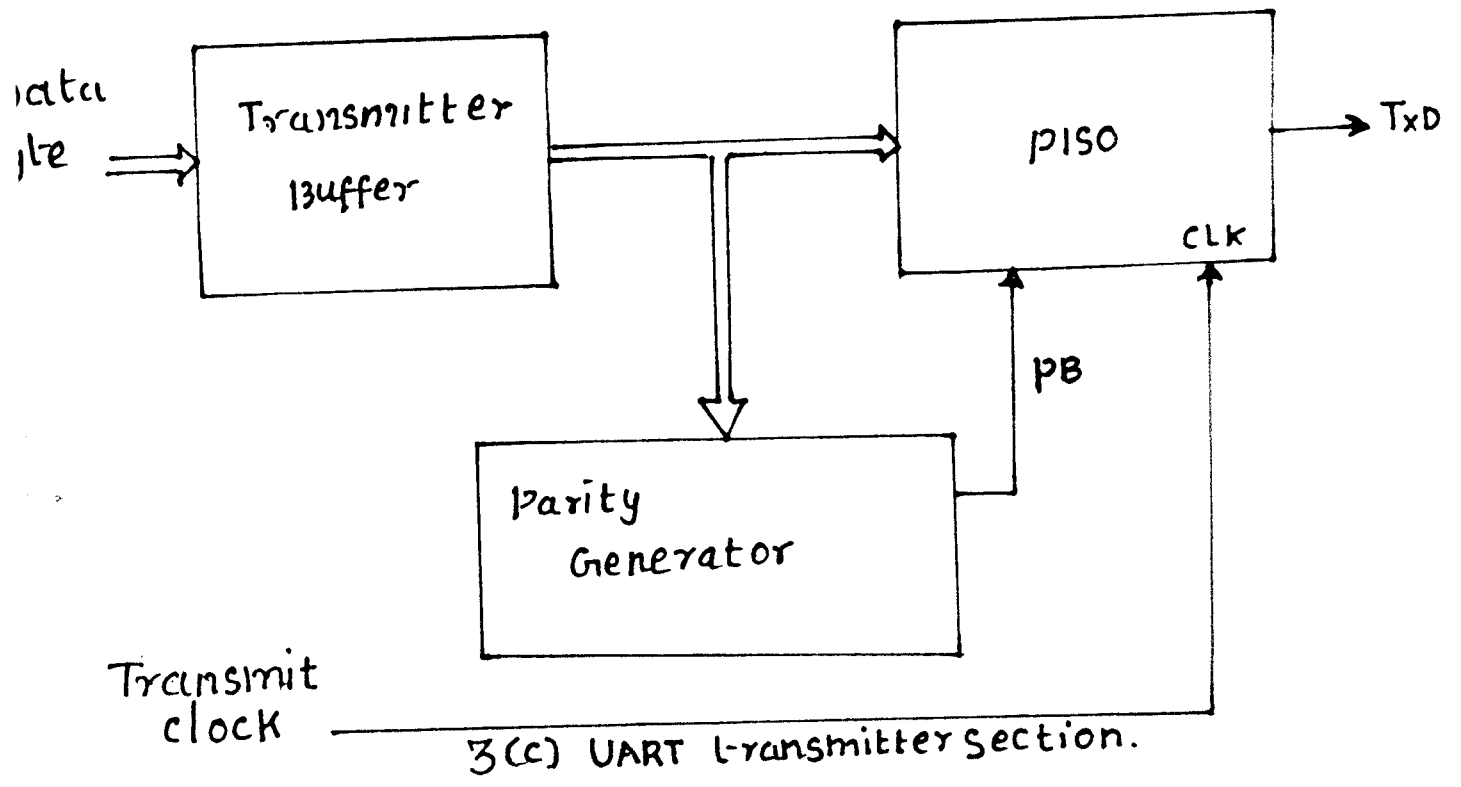
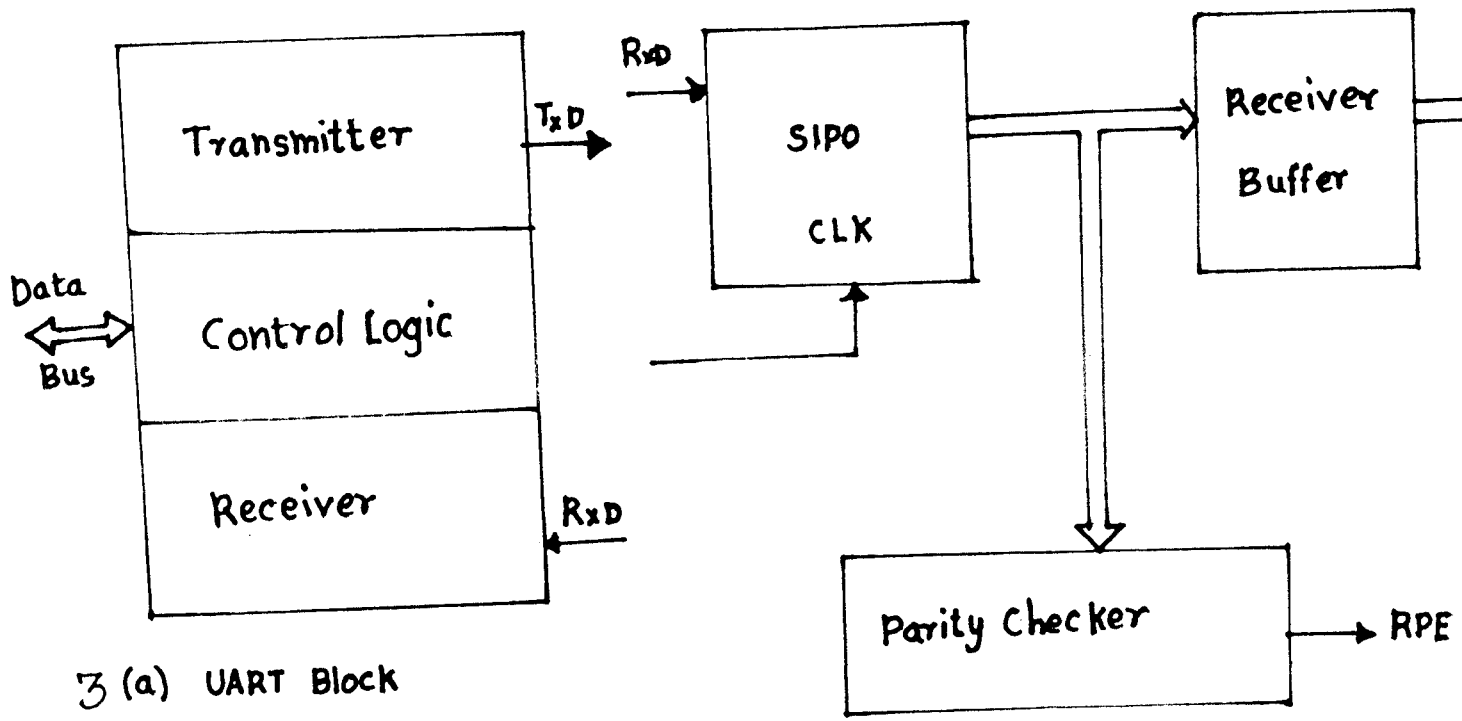
Even the fastest of these speeds are not the limit of serial hardware. In fact, third-party developers offer software that drives IBM hardware to much higher speed-today 115,200 bps appears to be the limit. At the unsupported speeds above 19,200 bps, the doubling increment no longer applies, and most developers allow a 19,200 bps gaps between the higher speeds they support. The 9600 and 19,200 bps limits enforced by IBM represent the maximum operating speed of the parts once the overhead of the serial port BIOS routines are taken into account. Programs that run faster than the IBM limit do so by avoiding these slow BIOS routines.

OPERATING MODES OF UART

IBM relies on a special type of integrated circuit to transform the parallel signals inside the computer into a serial train of pulses called as Universal Asynchronous Receiver/Transmitter or UART, this chip accepts eight lines as a parallel input and provides a serial output. It is designed to work both ways, and can convert serial signals into the parallel signals.

Three different types of UART are used in the IBM family of computer. The original PC and XT used a chip numbered the 8250 that was installed on IBM's Asynchronous Communications Adapter card. Most after market vendors adapted this same chip to their communications and multifunction boards. It is also used in many internal modems.

The Universal Asynchronous Receiver Transmitter (UART) is usually a programmable LSI device having necessary hardware circuits for implementing asynchronous serial communication. Fig 3 shows the essential components of the



receiver section and transmitter section in an UART. The receiver converts serial data bits received on the line into parallel bytes. The transmitter converts the parallel bytes into serial bits to be sent on the line. The frequency of the receiver clock has to match the baud rate of the received data (RXD). The SIPO (Serial In Parallel Out) logic deserialises the serial data bits into a parallel byte. The RPE (Receiver Parity Error) signal is generated by the parity checker if there is a wrong parity in the received character. In the transmitter section, the PISO (Parallel In Serial Out) logic converts the parallel byte into a serial bit stream. It also adds start bit, parity bit and stop bit to the data.

The frequencies of the transmit clock and the receive clock need not be equal. But the baud rate of the sending end transmitter should be equal to the receiving end receiver. If the baud rates are not equal the receiver section will generate an RFE (Receiver Framing Error) signal. When the receiver section finds an invalid data format it issues the RFE signal. The received data is said to be invalid when

(a) The start bit is sensed but no stop bit is found after the data bits and the parity bit time.

(b) The start bit is sensed but its duration is less than a baud period.

If the receiver clock frequency and the data format are proper and still the RFE is generated, it is obvious that there is either noise on the line, or a fault in the line or receiver circuits.

The UART samples the line condition at a fixed frequency which is 16 times the baud rate. To satisfy this, the clock inputs to the UART should be 16 times the desired baud rate.

Besides data transmissions, the UART also creates and reacts to other signals which control its operation. Control is afforded through several registers that are accessed by the computer through I/O ports. For example, to change the speed at which the serial port communicates, the only need is to load the registers with proper number. The conversation control is handled by voltages that appear or are received on the serial port connectors on the rear panel of PC or PS/2.

Flow Control



P-1299

The computers have the problem that they may shovel out data and have it disappear into the ether unused. Even when the connection is good, the receiving equipment may be otherwise engaged and not able to give its attention to the serial information being delivered to it or the serial

data arrive at such a high speed that it exceeds the capacity of the receiving system to do anything with it on the fly-even saving the information for later inspection. Consequently, some means is needed for the receiving system to single the sending system to hold on and wait until it is ready to acquire data. Several techniques for controlling the flow of serial data have evolved, all generally classed as methods of handshaking, called that because it signifies the agreement to the terms of the transmission method.

The easiest solution is to use a special wire as a signal line that the receiving system can use to indicate that's it is actually ready to receive. Because this method used extra hardware-the flow control wire it is termed hardware handshaking. This is the default flow control method used by IBM personal computers.

Some communications channels do not allow the use of an extra signal wire. For instance, the telephone connection used by modems (the prototypical serial communications device) only provide the two wires necessary for carrying data; consequently flow control systems based on characters embedded in the data being transmitted are often used. Because these flow control characters can be added through special programming of the sending system, this is often called as software handshaking.

In most software handshaking methods, the receiving system uses two distinct characters to tell the sending system when it is ready to receive a data transmission and when it can no longer accept more data at least temporarily.

Two methods of software handshaking are commonly used: One, called ETX/ACK, uses the control code represented by the ASCII hexadecimal character 03(hex) to indicate that it requires a pause in data transmission, and the ASCII character 06(hex) to indicate that it's okay to resume. More common among PC products today is XON/XOFF handshaking, which uses the ASCII characters 13(hex), and 11(hex) to ask for pause or resumptions of data flow.

Although most PC peripherals that used a serial connection offer the option of software handshaking, without special driver software, they will not work properly with an IBM personal computer product or a compatible. The result is data overflow and characters are lost from the transmission.

HARDWARE DESCRIPTION

The fig.4.a. shows the circuit diagram of the **Serial/Parallel Communication Adapter** in the null modem configuration using RS 232C standards. This card has two sections,

- 1) a serial port using 8250 UART and
- 2) a printer interface using printer controller IC 82C11.

The serial port consists of a semi custom PAL chip 16L8ACN. A semi custom chip is a chip whose interconnections are incomplete. The user has the facility to program the chip according to his specifications. It is a programmable AND array driving a fixed OR logic. The inputs to the OR gates are fixed permanently and connected to a set of AND gates. The output can be selected by programming the chip. The PAL used has a matrix of 32 by 64 with 10 inputs and 8 outputs.

The address signals are taken from the I/O slot and given to the PAL input pins. The I/O port address of COM1 and COM2 are 03F8 and 02F8 respectively. When the COM1 port is to be selected by using the address 03F8, a pulse is produced at 08 (pin19) of the PAL which selects that port. Similarly for selecting COM2 port, 02F8 is given to the PAL inputs and a pulse is produced at 07 (pin 18) of the PAL.

The I/O slot also gives the address bits A2, A1, A0 which are connected to the respective A2, A1, A0 pins of the UART.

These address signals are used for selecting the 10 internal registers of the UART. This function is implemented by software. The registers are,

1) Receiver Buffer register (RB) or Transmitter Holding register(THR)

During the input operation by CPU the RB is accessed. During output operation THR is accessed.

2) Interrupt Enable register (IER) fig.4.b.1

This is an output register with four bits masking for four different interrupts.

3)Interrupt Identification register (IIR) fig.4.b.2

Indicates two types of information

- a) Interrupt pending status
- b) Interrupt level which has been given priority.

4) Line Control register (LCR) fig.4.c.

An output register, where the format of an asynchronous character is stored by program.

5) MODEM Control register (MCR) fig.4.d.

Controls the output signals to the MODEM and also provides loopback facility.

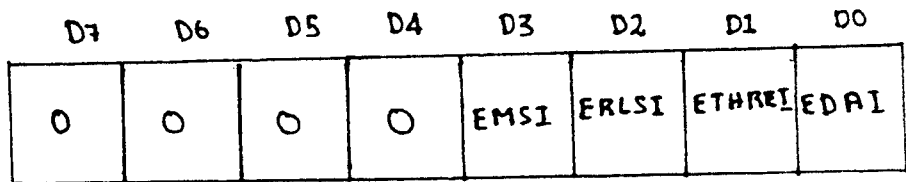


FIG. b.1 INTERRUPT ENABLE REGISTER FORMAT

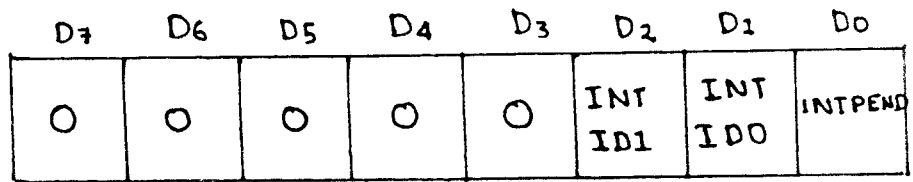


FIG. b.2 INTERRUPT IDENTIFICATION REGISTER FORMAT

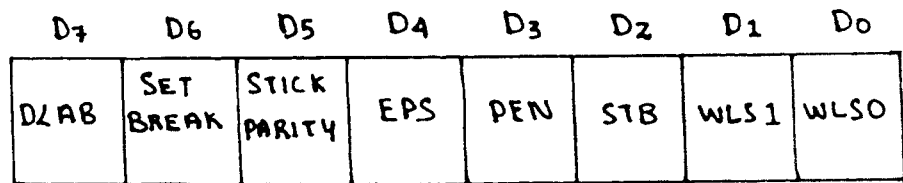


FIG. c LINE CONTROL REGISTER FORMAT

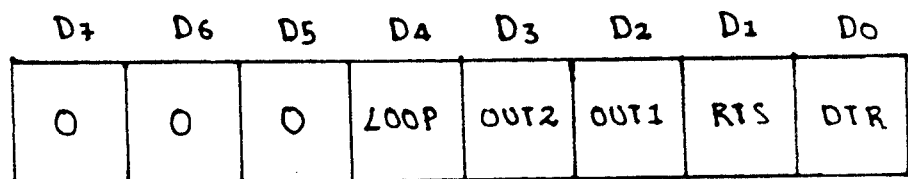


FIG. d. MODEM CONTROL REGISTER FORMAT

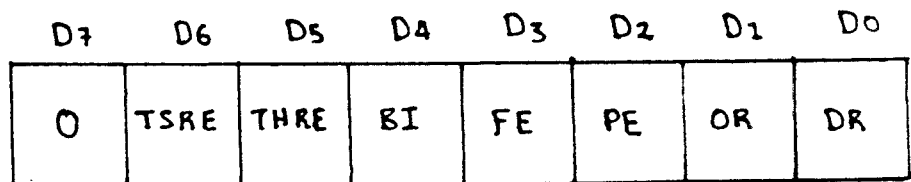


FIG. e. LINE STATUS REGISTER FORMAT

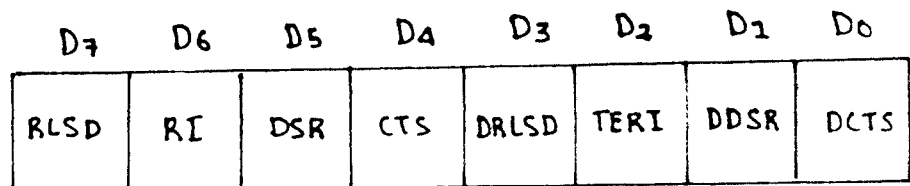


FIG. f. MODEM STATUS REGISTER FORMAT

6) Line Status register (LSR) fig.4.e.

Indicates different status conditions in transmitter and receiver sections of the UART.

7) MODEM Status register (MSR) fig.4.f.

Indicates the status of the signal from the modem.

8) Baud rate divisor (L Byte) and Baud rate divisor (H byte)

The program issues two bytes to these registers.

The 8250 has three chip select signals (CS0, CS1, $\overline{\text{CS2}}$). To select the UART chip $\overline{\text{CS2}}$ is made low while CS0 and CS1 are permanently pulled high.

The UART has two input control signals to enable reading from the chip. The DISTR (pin 22) is an active high input and the $\overline{\text{DISTR}}$ (pin 21) is an active low input. Any of these pins can be used for reading from the UART. Here DISTR is grounded and $\overline{\text{DISTR}}$ is connected to $\overline{\text{IOR}}$ of the I/O slot. Similarly there are two control signals DOSTR (pin 19) and $\overline{\text{DOSTR}}$ (pin 18) to enable writing into the chip. Here the DOSTR is grounded and $\overline{\text{DOSTR}}$ is connected to $\overline{\text{IOW}}$ of the I/O slot. The $\overline{\text{ADS}}$ input (pin 25) is used to latch the chip select and address inputs to the UART. But as the PC works in the CPU bus cycle these pins are permanently grounded. The INTPRT (pin 30) is made high by the UART when it wants to interrupt the CPU. This signal is controlled by software and goes high for the following conditions.

a) a break in transmission

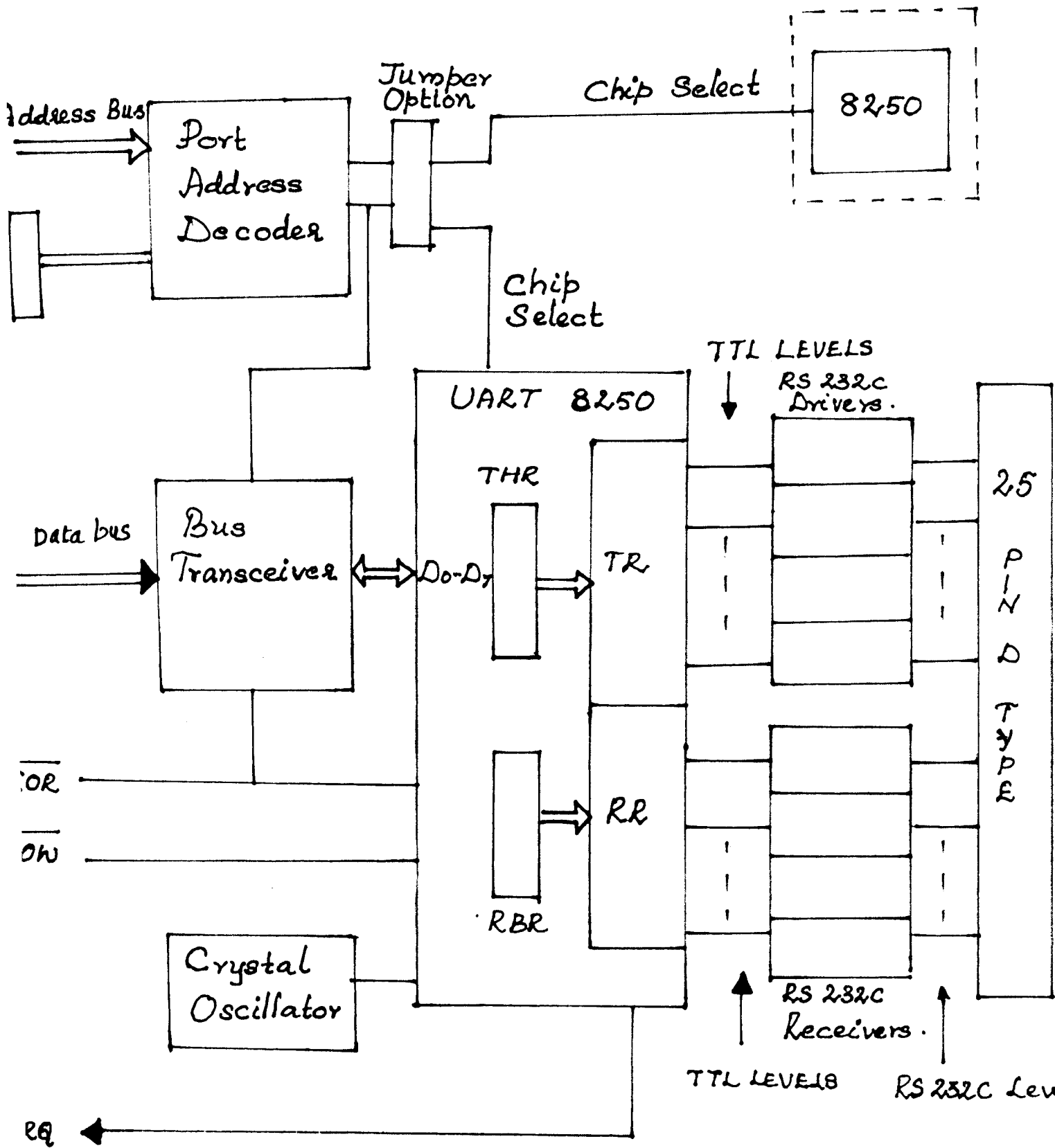


Fig 4(a): BLOCK DIAGRAM.

- b) an error in transmission
- c) after transmission of a file and
- d) after reception of data .

The INTPRT signal goes to the PC as the IRQ signal. Here IRQ3 is reserved for COM2 and IRQ4 is reserved for COM1.

The AEN signal is used for selecting the adapter card when it is low .During DMA cycles the AEN goes high thus disabling the card.

When a data is to be sent through this card the ports are selected using the address for either COM1 or COM2. When COM1 is selected using address 03F8 and if the data is to be transmitted, the \overline{DOSTR} is made active low through the \overline{IOW} signal. The registers of the UART are initialised using software.

The data bits to be transmitted are given to the D0-D7 pins of the UART. These data bits are stored in the THR of the 8250. After a file has been transmitted the UART gives an interrupt.

The parallel data signals are converted into asynchronous serial data whose character length, baud rate, parity bit are determined by software. The serial output data comes from SOUT (pin 11) of the UART and are given as the TXD signals to the RS232C interface chip (MC 1488), converted

to RS 232C standards and transmitted. Similarly when receiving data through RXD line the RS232C signals are converted into 5V signals by the interface chip MC 1489 and given to the SIN (pin 9) of the UART. This data is stored in the RB register of the UART and goes to the PC after given the interrupt.

Handshaking signals are used between the two terminals. The handshaking signals are

- 1) Request To sent (RTS)
- 2) Clear To Sent (CTS)
- 3) Carrier Detect (CD)
- 4) Data Terminal Ready (DTR)
- 5) Data Set Ready (DSR)

The baud rate of the transmitter and receiver sections are determined by software. The $\overline{\text{BAUDOUT}}$ of the 8250 is the signal whose frequency is 16 times the baud rate which is connected to RCLK (pin 9) of the UART. This is to make both the transmitter and receiver sections of the UART to work at the same baud rate.

The next section of the adapter card consists of a centronics printer interface using 82C11. The block diagram of the Controller is shown in Fig.4.g. This chip is selected through the PAL chip using the addresses 0278 and 0378. The printer interface consists of two parts

- 1) Clock generator
- 2) printer interface logic.

The printer controller is a 40 pin LSI chip and is dedicated PC bus compatible.

The clock generator section has an oscillator part to which a crystal of 18.432M frequency is connected at the input pins X1 and X2. The RCLK output is available at pin 9 of the UART.

The interface can be identified by the 25 pin female connector. It has three address ports available which can be configured. There are three address port 03BC, 0378 and 0278.

0378 and 0278 may be configured as I/O port addresses while 03BC is used as data port address. Each address inturn has three internal addresses namely,

03BC --- Data port
03BD --- Status port
03BE --- Command port

The Centronics Interface provides a handshake protocol between a computer and a printer and supports a maximum data transfer speed of about 100 kb/s. The printer side of the interface is a 36 pin connector and the PC side is a 25 pin D type connector. The PC uses 36 pin flat cable in which every alternative wire is for the ground. The signals are TTL level signals and the twisted pair return ground wire for each signal is connected to the signal ground level. To prevent noise effects the twisted pair wires are shielded and

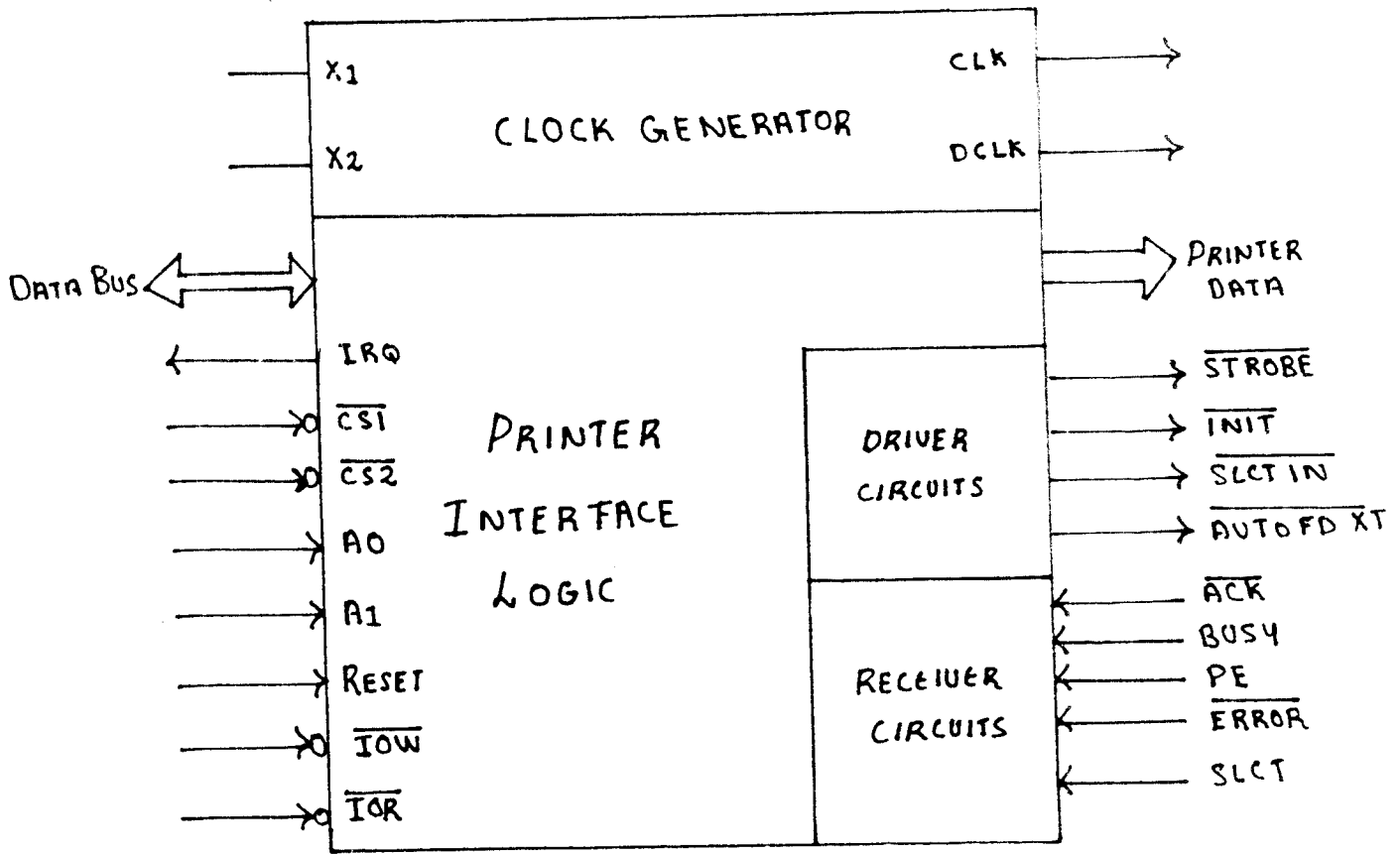
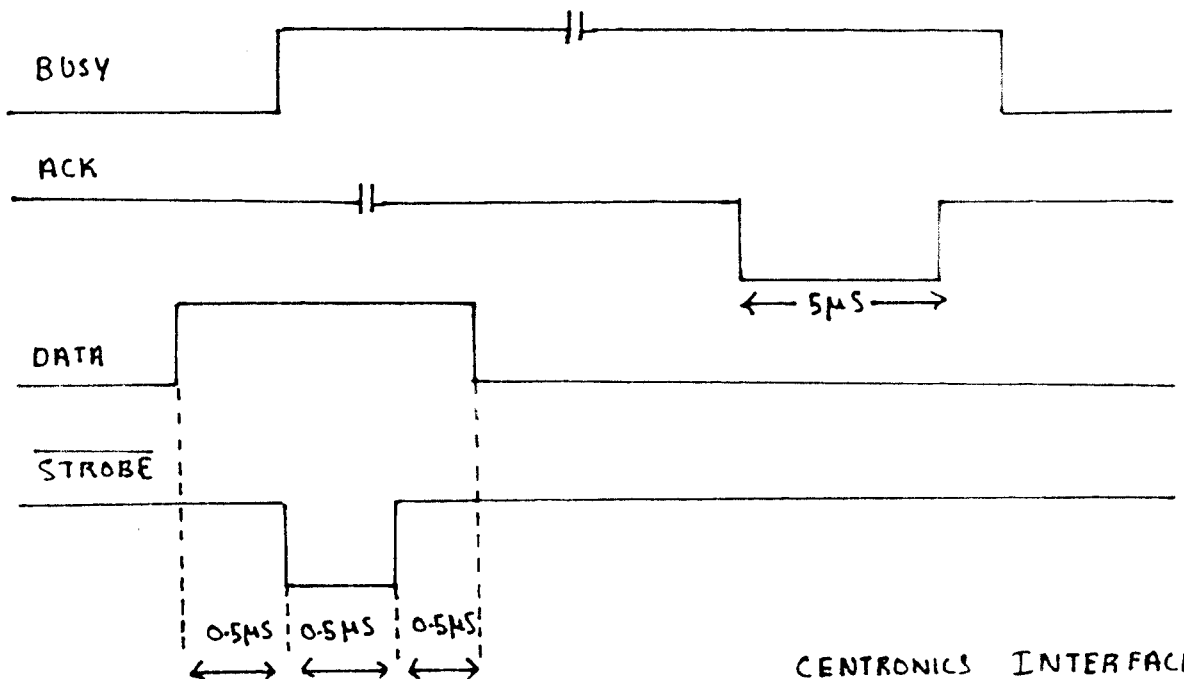


FIG. 4-G 82C11 BLOCK DIAGRAM

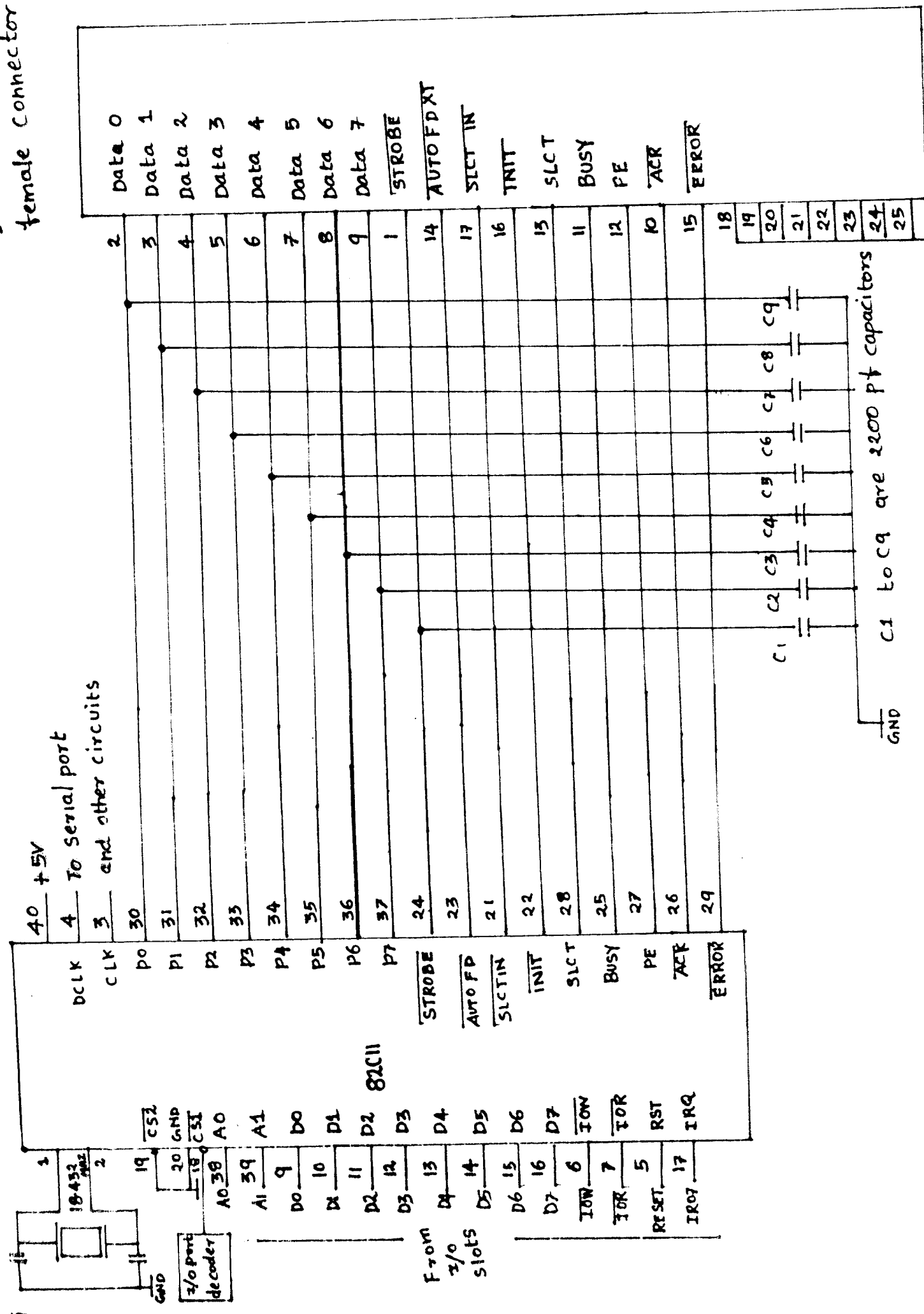


CENTRONICS INTERFACE

FIG. 4-H

TIMING DIAGRAM

D shell 25 pin female Connector



... circuit using 82C11

the shield is connected to the chassis ground in the system box.

SIGNALS FROM PC TO PRINTER:

There are 12 signals from the PC to printer. Out of these eight signals are data bits and four signals are control signals. The control signals are: $\overline{\text{STROBE}}$, $\overline{\text{INIT}}$, $\overline{\text{SLCTIN}}$, $\overline{\text{AUTO FEED XT}}$. All the control signals are low active.

$\overline{\text{STROBE}}$: The printer should take the data when this signal is low.

$\overline{\text{INIT}}$: When $\overline{\text{INIT}}$ is low, the printer resets its electronics logic and clears the printer buffer.

$\overline{\text{SLCTIN}}$: $\overline{\text{SLCTIN}}$ is an interface enable signal. When this signal is low, the printer responds to signals from the controller.

$\overline{\text{AUTO FEED XT}}$: After printing every line, the printer will provide one line feed automatically if this signal is low. This type of line feed is known as hardware line feed.

SIGNALS FROM PRINTER TO PC:

There are five status signals from the printer to the PC. These are $\overline{\text{ACK}}$, BUSY , PE , SLCT , and $\overline{\text{ERROR}}$.

$\overline{\text{ACK}}$: $\overline{\text{ACK}}$ signal is an acknowledgement for STROBE signal from the PC. When active, it indicates that printer has received data sent by the PC and the printer is ready to

accept next data type.

BUSY: When BUSY signal is high, it indicates that the printer is busy and it cannot receive data. This signal becomes high under any of the following four conditions:

1. On receiving $\overline{\text{STROBE}}$ active
2. During printing operation
3. When the printer is in offline state
4. When the printer senses some error condition

PE: When PE signal is high, it indicates that there is no paper in the printer. Either the paper is torn or the paper is over.

SLCT: SLCT signal indicates that the printer is selected and logically connected to the PC.

$\overline{\text{ERROR}}$: $\overline{\text{ERROR}}$ signal indicates that there is some error condition in the printer. The three reasons for this signal to go active are:

1. Mechanical fault or electronic fault in the printer
2. The printer is in offline state
3. There is no paper in the printer, i.e., paper-end state.

The timing diagram of Centronics interface protocol is illustrated in Fig.4.h. The printer controller sends data to the printer. After a minimum gap of 0.5 micro seconds, it makes $\overline{\text{STROBE}}$ low and keeps it low for a minimum duration of 0.5 micro seconds. As soon as $\overline{\text{STROBE}}$ become active low, the

printer makes the BUSY line high. The controller should retain data on the data lines for a minimum interval of 0.5 micro seconds from the trailing edge of $\overline{\text{STROBE}}$. Thus the data should be kept on the data lines for a minimum duration of 1.5 micro seconds. When the printer is ready to receive the next character of data, it makes $\overline{\text{ACK}}$ line low. When $\overline{\text{ACK}}$ is made inactive, the printer also removes BUSY. To perform data transfer with a printer, it is enough if the the printer controller senses either BUSY or $\overline{\text{ACK}}$ signal.

SERIAL INTERFACE STANDARDS

Most of today's sophisticated microcomputers and microprocessors based instrumentation systems come across the term 'RS-232-C' compatible serial interface. In order to understand the serial interface standards various modes of digital data communication are considered. Basically, these interface standards define the interface between a data terminal equipment (DTE) and a data communication equipment (DCE) i.e the links 1 and 3 shown in fig.5(a). The DTE and the DCE are also sometimes referred to as data terminal and data set respectively.

We shall look in to commonly used standards, such as the Electronic Industries Association (EIA) defined as RS 232C, or its approximate the Consultative Committee of International Telephone and Telegraphs (CCITT) defined V24/V28 specifications.

Let us develop the interface logically step by step. The basic data exchange lines have to be present in the most rudimentary interface as shown in fig.5(b). There will be unidirectional transmission and reception data lines along with common ground line between the DTE and the DCE.

It can be noticed, that this is not a complete and sufficient interface. The DTE and DCE do not have any means to 'handshake' with each other; nor do they know

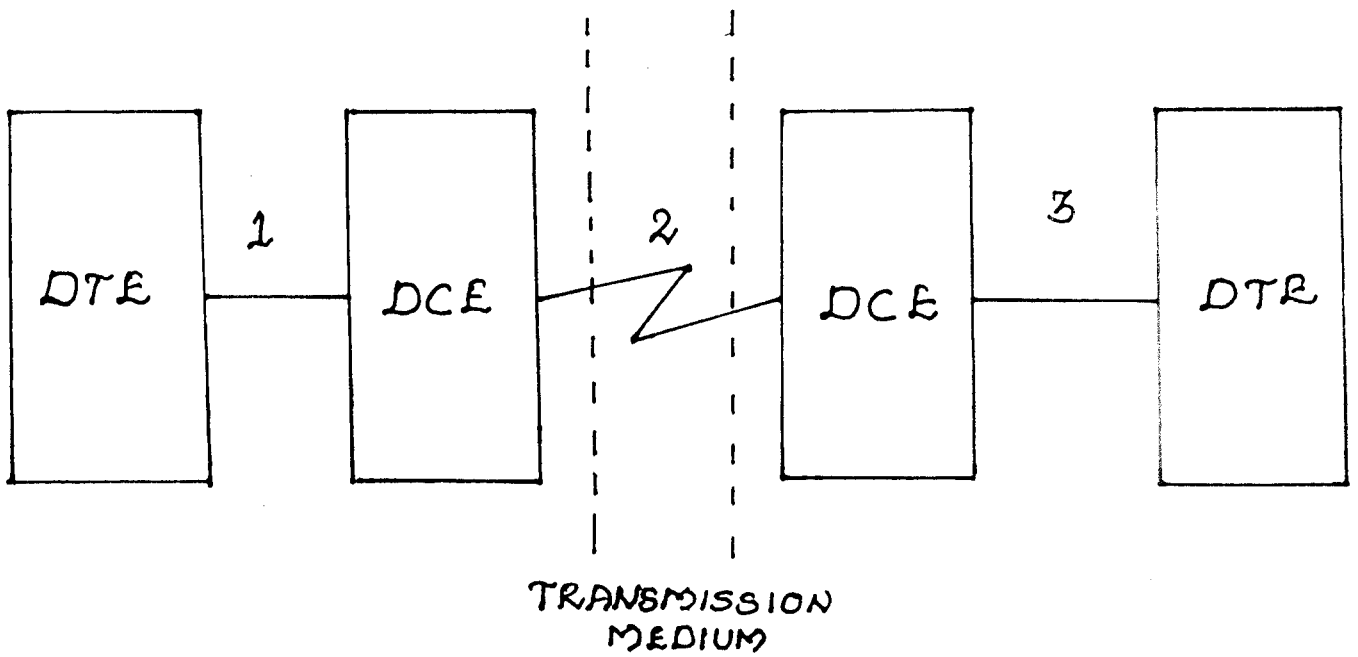


Fig: 5 (a)

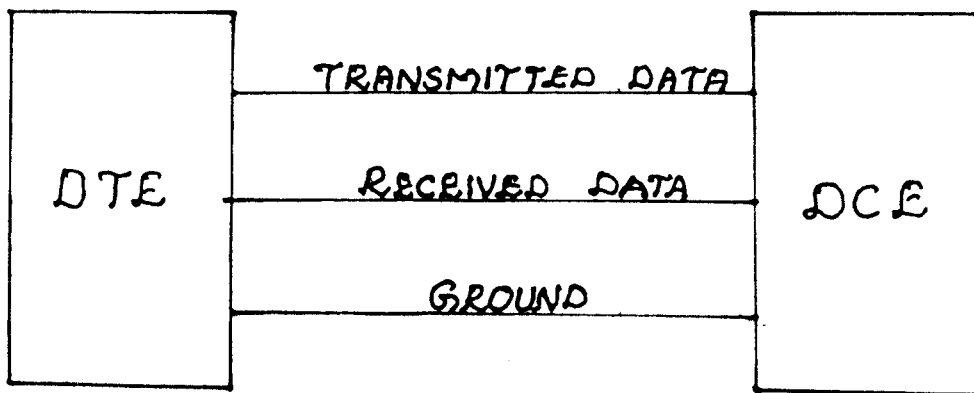


Fig: 5 (b)

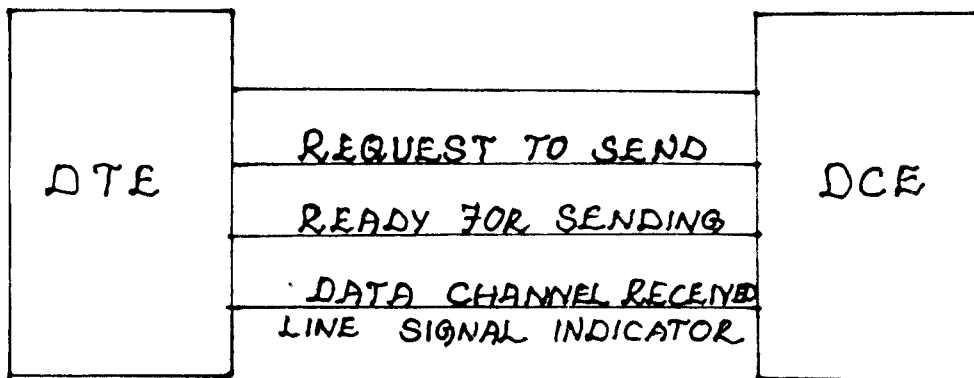


Fig: 5 (c)

when one is going to receive or transmit data. In order to overcome this defect, some lines are included to control the direction of data flow between the DTE and the DCE (see Fig.5(c)). The line 'request to send' is to indicate to DCE that DTE wishes to transmit data. Next line 'ready for sending' is to allow DCE to inform DTE that it is available and ready to transmit. The line 'data channel received line signal indicator' allows DCE to inform DTE that it is in a condition to accept incoming data.

Apart from these lines, some other lines 'enabling lines' should be included in the interface. We may have some status information from DTE and DCE as well. Such an interface is given in Fig.5(d). 'Data set ready' line in Fig.5(d) indicates to DTE that DCE is now operational. 'Data terminal ready' line indicates to DCE that DTE is now operational and it is used to enable the DCE. 'Calling indicator' signal is used by DCE to inform DTE that a calling signal is being received.

Till now the interface has been geared to handle asynchronous data communication efficiently. But in the case of asynchronous data movements, the data lines themselves carry the synchronising bits, namely, the start and stop bits. In order to generalise interface, provisions should be kept for synchronisers. This complete interface is shown in Fig.5(e).

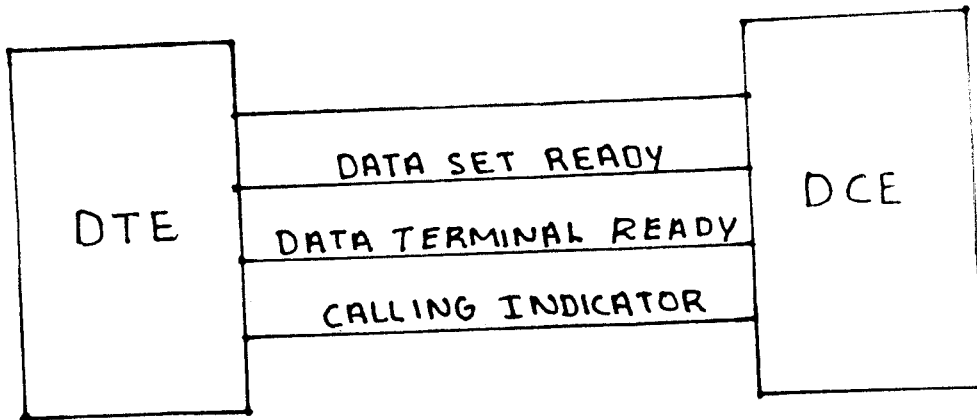


Fig: 5(d)

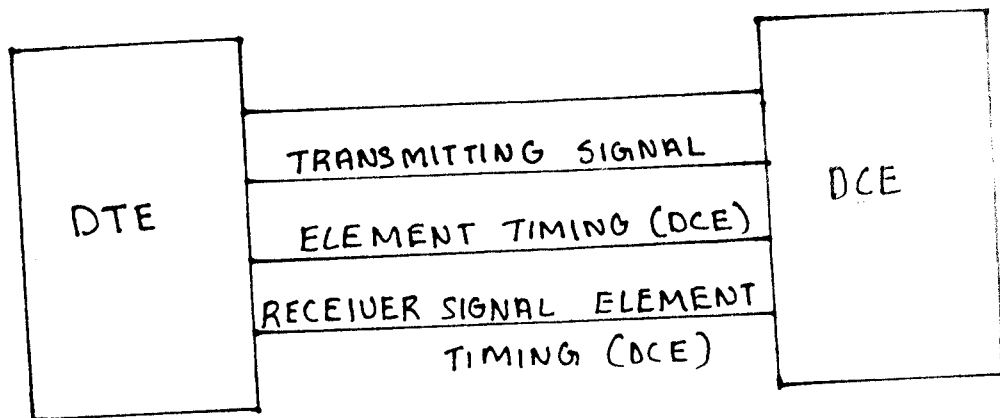


Fig: 5(e)

'On to off' transitions on the line marked 1 indicate the time at which the DCE should sample the 'transmitted data' line. The DTE is responsible for the timing in this case. In a similar way 'off to on' transitions on the line marked 2 indicate to DTE when the next data element for transmission should be presented on the 'transmitted data' line. In this case the DCE is responsible for the timing.

Similarly for other timing element line, 'on to off' transition the line, indicate the time at which DTE should sample the 'received data' line. The DCE is responsible for the timing.

The interface developed above is nothing but the CCITT-V24 interface. The electrical characters of the interface are laid down in the other standard CCITT-V28. The together are often called the CCITT-V24 interface. RS-232C standard is almost equivalent to this interface. RS-232C standard is widely used in American industries and CCITT-V28 is used in European industrial circles. These standards are meant to cover data rates up to 20kbauds (1 baud=1 bit/sec.) and for a maximum distance of about 15 metres. For higher data-rate-distance products, and for DCE's designed to work with high speed data-links, the CCITT-V35 performs the same role as the V24 or V28 for low-data rates. Corresponding DIA standard for high data-rate-product is the RS-423. All these standards relate to an unbalanced mode of transmission.

The other mode of digital data transmission is known as the balanced mode of transmission. This mode has an inherent advantage over the other, in that it has better common-mode rejection (CMR) of noise. This effect is illustrated in Figs. 5(f) and 5(g).

Fig. 5(f) shows an unbalanced driver and receiver connection. Note that the receiver will receive the signal and any noise generated in the line without distinction. In some of the contemporary receiver chips there are threshold-set pins to reject this common-mode noise to some extent.

Fig. 5(g) shows a balanced transmission and reception system. You will notice in this figure that any common-mode noise introduced in the transmission line will be rejected as per the common-mode rejection ratio (CMRR) of the receiver. In this form of transmission the data travels both its direct and inverted forms through a pair of lines in each direction.

There is an EIA standard for balanced transmission at high data rates. It is known as the RS-422 standard.

Some important characteristics of common EIA standards are listed in Table I. Manufacturers like Texas Instruments, Motorola and American Micro Devices produce driver and receiver chips for these EIA standards.

TABLE--1

PARAMETER	RS 232C
MODE OF OPERATION	UN BALANCED
MAXIMUM DISTANCE(ft.)	50
MAXIM DATA RATE (baud)	20K
MAXIMUM COMMON MODE VOLTAGE (volts)	+ 25 -
DRIVER OUTPUT SIGNAL	+ 5V min - + 15Vmax -
DRIVER LOAD(ohms)	3k -- 7k
POWER-OFF DRIVER OUTPUT RESISTANCE (Hi-Z state)	300 ohms
RECEIVER INPUT RESISTANCE (ohms)	3k -- 7k
RECEIVER SENSITIVITY	+3V

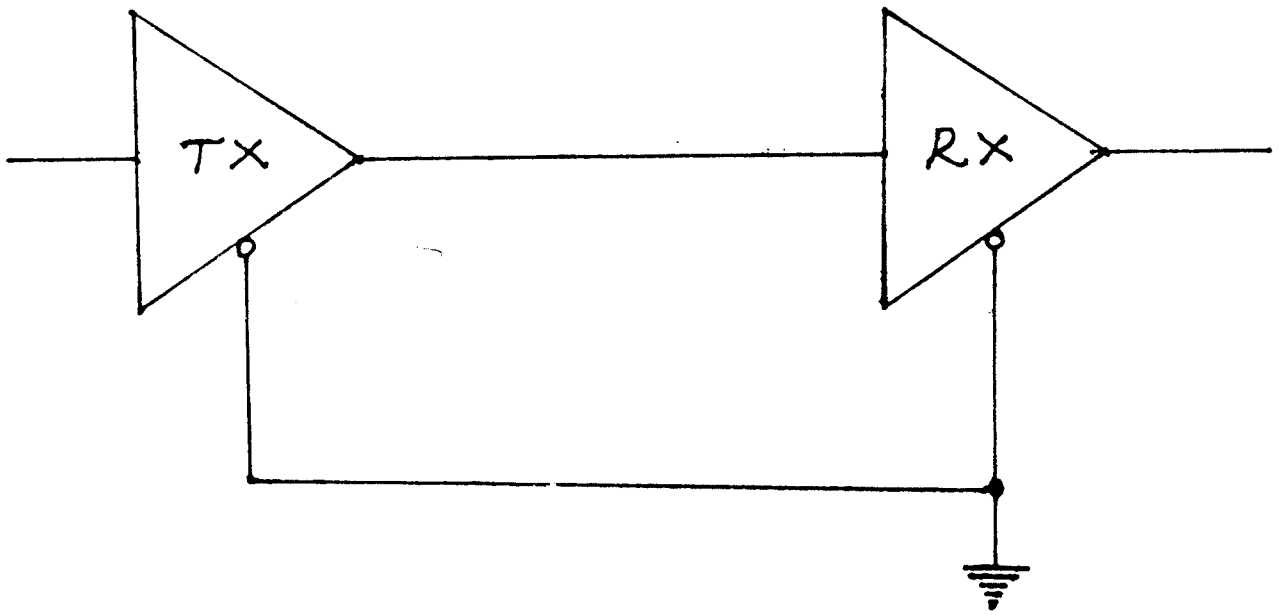


Fig: 5(f)

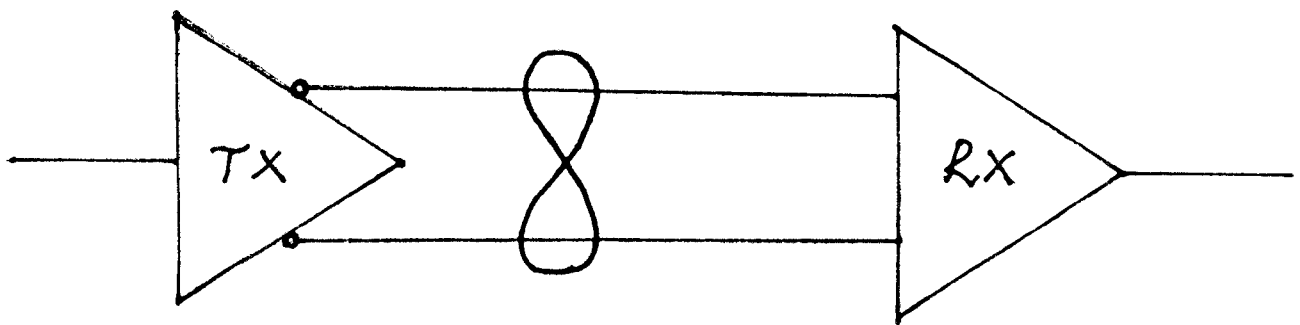


Fig: 5(g)

Sometimes we come across another term '20/40/60 mA current loop.' This current signaling method of digital data transmission is based on telegraphy techniques. The basic circuit of the 20 mA current loop driver and receiver is shown in Fig. 5(h).

In this diagram, transmission is effected by means of energising relay A which causes a current of 20 mA to flow round the loop which in turn, energises relay B and closes contact B of the receiver. In modern circuits, we have opto-couplers and transistor amplifiers at the transmission and reception points.

Finally, while transmitting data, the most important question is when to use the current method, and when to use the voltage method. The 20mA current loop has better performance in electrically noisy environments. When the transmitter and receiver have the same ground level, then only the voltage method can be used successfully.

ARCHITECTURE AND ORGANISATION OF IBM PC FAMILIES

The IBM introduced the IBM personal computer on August 11,1981 and officially withdrew it from market on April 2,1987.During the nearly six year life of PC it dominated the market.

The specifications of PC are as follows:

1. Intel 8088 microprocessor
2. Clock frequency of 4.77 Mhz
3. Five 8 bit I/O expansion slot
4. 40 K ROM with ROM based diagnostics,power on self test (POST)
5. One or two floppy disk drives which can be expanded upto 4 drives
6. Basic interpreter in ROM
7. 256 K of dynamic RAM can be expanded upto 640K through expansion of memory boards
8. 200 nano second memory access time
9. Electrical requirement 104 volts to 127 volts, 50Hz to 60Hz
10. 63.5 watt power supply
11. 83 key
12. Cable length of 6 feet
13. One parallel port along with MDA
14. Socket for 80287 math coprocessor

IBM PC-XT

Introduced March 8, 1983 the PC_xt with built in 10 - M hard disk (original standard) caused a revolution in personal computer configurations. XT stands for extended XTended. IBM choose this name because the IBM PC XT system includes many features with enchanced version.

The specifications are:

1. Intel 8088 16/8 microprocessor working at 4.77Mhz clock speed and having 20 address bits through which it can support 1MB of memory
2. 40K ROM having
 - a. POST(power on self test)
 - b. BIOS(basic input output system)
 - c. Basic interpreter
3. 8 bit I/O expansion bus
4. 256K or 640K of dynamic ROM having 200ns memory access time
5. One 360Kb floppy drive
6. 10M or 20M hard disk drive
7. C.G.A adapter with support TV's
8. One parallel and serial ports
9. Socket for 8087 math coprocessor
10. 135 watt power supply
11. 83 key keyboard and its cable length is 6 feet

IBM PC-AT

IBM introduced the PC-AT (advanced technologies) August 14, 1984. It included many features previously unavailable such as increased performance and advanced microprocessor high capacity floppy disk and hard disk drives larger memory space and an advanced coprocessor. Despite its new design the IBM AT retains compatibility with most existing hardware and software products of earlier system.

The specifications are:

1. Intel 80286 microprocessor having 24 address bits and working at 6 or 8 Mhz clock speed
2. 64K of ROM having
 - a. POST(power on self test)
 - b. BIOS(basic input output systems)
 - c. Basic interpreter
3. The microprocessor has modes
 - a. 8086 compatible real mode (1MB memory only)
 - b. Protected virtual address mode(16MB memory)
4. 512K of dynamic RAM having 150ns memory access time
5. 1.2M double sided or high density floppy drive
6. 20M or 30M hard disk drive
7. Serial/Parallel interface
8. Clock/Calendar and configuration with battery backup

9. Key lock
10. 84 key keyboard having cable length of 9 feet
11. 19 switchable world wide power supply
12. 192 watt variable speed fan temperature controlled power supply
13. Socket for 80287 math coprocessor

Though we are having different kind of PC products but configuration and they are software compatible only. So now we look into PC's whatever going to be specified are all same for starting from PC to PC-AT unless otherwise specifically specified.

1. DMA

In PC, PC-XT and PC-AT DMA data transfer is used between memory and floppy disk. In PC-XT hard disk and memory transfer is also through DMA. But the AT hard disk transfer is through programmed mode of transfer. There are two DMAC in AT, in XT and in ordinary PC there are only one.

They are all mapped for address

DMAC1 = 000 - 00F

DMAC2 = 0C0 - 0DF (only in PC-AT)

2. TIMER

The PC, PC-XT and PC-AT timer is used for three purposes:

- a. Real time clock
- b. DMA refresh
- c. Speaker oscillator

In the above 8253 and 8254 is used and address are mapped between 40 - 4F.

3. INTERRUPT

Interrupts are used for different purposes. They are:

IRQ no.	INT type	Purpose
IRQ0	INT8	Real time clock
IRQ1	INT9	Keyboard
IRQ2	INT0A	Not used
IRQ3	INT0B	COM2
IRQ4	INT0C	COM1
IRQ5	INT0D	Hard disk
IRQ6	INT0E	Floppy disk
IRQ7	INT0F	Printer

The above is true for both PC and PC-XT.

In PC-AT there are two 8259's and hence there uses are:

IRQ no.	INT type	Purpose
IRQ0	INT8	Timer output
IRQ1	INT9	Keyboard
IRQ2	INT0A	Cascaded interrupt controller input

IRQ3	INT0B	COM2
IRQ4	INT0C	COM1
IRQ5	INT0D	LPT2
IRQ6	INT0E	Floppy disk
IRQ7	INT0F	LPT1
IRQ8	INT70	Real time clock
IRQ9	INT71	Redirection to interrupt INT0A H
IRQ10	INT72	Reserved
IRQ11	INT73	Reserved
IRQ12	INT74	Reserved
IRQ13	INT75	Coprocessor
IRQ14	INT76	Hard disk
IRQ15	INT77	Reserved

BASIC INPUT/OUTPUT SYSTEM

Like software the BIOS is a set of instructions to the computers microprocessor. Like hardware, however, the special instructions are not evanescent, rather, they are coded into the silicon of PROM chips. Because the twilight state of programs like the BIOS, existing in the netherworld between hardware and software, such PROM based programs are often termed FIRMWARE. The BIOS of a IBM or compatible computer is very special firmware, comprising routines that test the computer, others that give in its personality more to help other programs more smoothly mesh with the electronics of the system.

The distinct parts of the BIOS work separately and distinctly even though the code for each is contained inside the same silicon chips. It operates like a set of small TERMINATE AND STAY RESIDENT PROGRAMS (like SIDE KICK, or PRO KEY) that are always in memory.

BIOS PURPOSE:

As long as all computers are crafted exactly the same, the same ports used exactly for the same hardware with exactly the same registers, there would be no problem.

However hardware arrangements differ from computer to computer the BIOS routines that worked like the old ones and were indistinguishable from the old ones when used by application software. The addresses inside the routines would be changed, however to match the updated hardware. The same software could thus work with a wide variety of hardware designs.

BIOS DATA AREA:

Once the BIOS code starts executing, it makes use of part of the host systems memory to store parameter values important to its operations. Included among the data that it stores are equipment flags, the base addresses of i/o adapters, keyboard characters, and operating modes. The bios data area comprises 256 bytes of memory, starting at absolute memory location 0000400(Hex).

IMPORTANT BIOS DATA AREA ASSIGNMENTS USED IN OUR SOFTWARE:

ADDRESS	FUNCTION
0400	Base address of first RS232 adapter (COM1)
0402	Base address of second RS232 adapter (COM2)
0404	Base address of third RS232 adapter (COM3)PS/2only
0406	Base address of fourth RS232 adapter(COM4)PS/2 only
0408	Base address of first printer adapter(LPT1)

040A Base address of second printer adapter(LPT2)
040C Base address of third printer adapter (LPT3)
0410 Installed hardware flags
Bit 0 = IPL diskette
Bit 1 = Numeric coprocessor
Bit 2 = Pointing device(except PC,XT,AT,
and convertible)
Bit 4,5 = Video mode
01 = 40*25 color;10 = 80*25color;11 =80*25mono
Bit 6,7 = Number of floppy disk drives
Bit 9,10,11 = Number of serial ports
Bit 13 = Internal model(Convertible only)
Bit 14,15 = Number of printer adapters.

THE APPLICATION ENVIROMENT

When DOS is loaded from a bootable disk, it usually places itself into the lowest memory location available. Device drivers, a special type of DOS program, load next. Finally, COMMAND.COM (or another program specified by the SHELL command in the CONFIG.SYS file) is loaded. The remaining memory is free for user programs.

A large portion of the command interpreter, COMMAND.COM, loads in the highest part of memory. This is the transient portion. (The rest, which loads in low memory, is the resident portion.) The transient portion may be overlaid by a user program requiring additional memory. When this happens, the resident portion of the COMMAND.COM reloads the transient portion after the user program exits. The figure shows system memory map in detail.

It is important to realize that DOS may load a program into any free memory location as long as it is on a segment boundary. In other words, starting addresses offset must be 0000, but its segment can be any value. Location may be different each time the program loads and will almost certainly differ depending on the machine. Different types of programs handle this problem in various ways.

TYPES OF DOS APPLICATIONS

DOS program fall into four categories: .EXE files, .COM files, device drivers and terminate and stay resident (TSR) programs. A .COM, .EXE, or TSR program in memory is sometimes called a PROCESS.

Each type of program has a set of rules that must be followed when writing them, and each works best for certain tasks.

.EXE FILES: .EXE files , the most natural application model for DOS programs, may contain multiple code and data segments. Since a segment can only be 64KB long, this is an important feature when building large programs.

An .EXE file consists of three parts. The first is a header as shown below, that contains information about the program. Among other things the header tells DOS how much memory the program needs, where to start the program, and the location of the data segment.

.EXE HEADER

OFFSET	LENGTH	DESCRIPTION
00H	word	.EXE file magic number (405AH)
02H	word	Length of last sector

04H	word	Size of file, including header(512 byte pages)
06H	word	Number of reallocation table items
08H	word	Size of header (paragraphs)
0AH	word	Minimum# of paragraphs needed above program
0CH	word	Maximum# paragraphs allowed above program
0EH	word	Displacement of stack segment(paragraphs)
10H	word	Contents of SP register on entry
12H	word	Checksum
14H	word	Contents of IP register on entry
16H	word	Displacement of code(paragraphs) relative to start of the program
18H	word	Offset to first relocation table entry (byte)
1AH	word	Overlay number(0 for main program).

The second part of the .EXE file is the relocation table. This table contains segment fix-ups, which point to places in the file that explicitly reference segments. DOS adds two byte number at each fix-up location to the segment address where the .EXE file loads, then copies the resulting number into memory.

The final part of the .EXE file is the data to be loaded into the computer.

.COM FILES: This is the simplest application model. The program all available memory. The data segment is the same as the code segment, which also equals the stack segment. .COM files always load and begin execution at offset 100H in their code segments. DOS does not process a .COM file anyway, so such a file cannot contain explicit references to segments as an .EXE file can.

.COM files have no headers- they only contain the data to load into memory.

.COM files loads faster than .EXE files and are slightly smaller than .EXE files. Its length cannot exceed 64KB.

TSR PROGRAMS:TSRs are not really different kind of programs; they may be .COM files or .EXE files. After an ordinary .COM or .EXE program completes, however, DOS removes it from memory before continuing. A TSR program, on other hand, leaves a portion of itself in memory after exiting. Usually it hooks itself into one or more interrupt routines so that when an interrupt occurs the TSR can process it.

DEVICE DRIVERS: They are special programs that communicate between DOS and one or more devices. DOS provides default device drivers for the standard devices (such as PRN, LPT1, AND CON) and disk drives. It loads device drivers right after itself during the boot sequence. Like TSR's, these drivers remain in memory after exit. Unlike TSR's you cannot remove them nor can you install them without rebooting the system.

There are two types of device drivers: Character and block. Character drivers have names like LPT1 and CON. DOS automatically assigns letters to block drivers.

Each type of driver has a specific format and must contain the routines that DOS prescribes. From that perspective, drivers are easy to write; however, because the routines to interface with some devices are difficult and since some device drivers are very hard to debug most programmers consider them as a challenge.

8088 MICROPROCESSOR AND I/O SLOT

This 8086 and 8088 both are compatible. They are first 16 bit microprocessors, both of them working in a same internal and external architecture. The 8086 microprocessor is a 16 bit microprocessor both internal and external but 8088 is a 16 bit internal and 8 bit external. Since compatible chips like 8237 DMAC, 8255 ppi, 8253 timer, 8259-internet controller were just 8 bit and cost of manufacturing of 16 bit chips were very high, Intel was forced to manufacture those 8088 chips and IBM also implemented their design around 8088 chips.

The 8088 microprocessor is divided internally into two parts BIU and execution unit. The actual reason for this division is to speed up processor operations by reducing CPU idle time. This is done by prefetching of instructions by the BIU into one part of microprocessor while the execution unit executes instructions. So overall efficiency of microprocessor and its speed are decreased because microprocessor need not to fetch an instruction only after execution which eats up too much CPU time though fetching of an instruction is

constant and is comparatively less than EC of an instruction cycle when more number of instructions are executed. This part of an instruction cycle becomes important to note so this division plays an important role in increasing speed of microprocessor.

The specifications and internal structure of this processor:

1. 20 address bits using which max of 1MB can be accessed
2. 8 data bits
3. Working at 5Mhz clock frequency
4. There are two mode minimum and maximum mode
 - a. Minimum mode equal to its predecessors which it will generate all bus control signals
 - b. Maximum mode ,the other microprocessor can be made to work with this microprocessor but bus control signals are to be generated outside the chip using some external hardware components.
5. Made up of HMOS technology(N channel,depletion load, silicon gate technology)and packaged in a 40 pin cer dip package

BIU

1. Bus control unit which generates bus control signals like ALE, DEN, DT/R etc
2. Instruction queue: This is the area where instructions are prefetched into the microprocessor by BIU. It is 4 byte wide in 8088 microprocessor and 6 byte wide in 8086 microprocessor.
3. Instruction pointer: This is a 16 bit pointer which points to the next instruction to be executed. It is just equivalent to program counter in 8085 microprocessor.
4. Segment registers: The 8088/8086 microprocessor views memory as different segments. Each and every segment has its own purpose. The segments are as follows.

SEGMENT	PURPOSE
Code segment	Instruction alone
Data segment	Data alone
Extra segment	Used in addition to data segment
Stack segment	For stack operations

The maximum of each segment is 64KB only, and these segments can be overlapping segments in the sense for eg . code segment is having instructions only up to 10KB ,let us assume that data segment starts immediately from 11KB onwards. There is no rule that it should start from 65KB onwards.

There are four segment pointers pointing the starting address of each segment. They are

CSR	Code segment
DSR	Data segment
SSR	Stack segment
ESR	Extra segment

The above said registers are 16 bit registers only.

5. Address generator : Here for example if you see to the address of an instructions there are two pointers, one code segment register pointing to the starting address of code segment where the instructions are stored. There is also an instruction pointer pointing which points to the next instructions which is to be executed. Both the

above pointers are 16 bit pointers. In order to address a memory location we require a 20 bit address. Hence putting the above two pointers we need to generate what is called as physical address. The formula used is

Segment address * 10H + offset address

Here multiplying a segment address by 10H means shifting segment address by four bits to the left.

EXECUTION UNIT

1. General purpose registers: The general purpose registers ax,bx,cx,dx are 16 bit wide. Though they are 16 bit wide they can be further divided into two 8 bit registers. The allowed pairs are

ax = ah,al

bx = bh,bl

cx = ch,cl and

dx = dh,dl

The above general purpose registers in most of the cases are also used as special purpose registers in some predefined instructions. This means

ax = accumulator
bx = pointer
cx = counter
dx = data register

Their usage can be seen while seeing instructions.

2. Offset registers:

a. Index registers: They are

SI and DI:

The SI is called source index and the DI is called as destination index register. Their usage is as offset register to string operation, one pointing to the source of the string and another the destination of the string. Though they are used as pointers they can also be used as general purpose register, but with one restriction that they cannot be divided as two 8 bit general purpose register.

b. Stack and base pointers:

Designated as SP and BP respectively, both are 16 bit pointers and are used as offset pointers within stack. While SP is used to access data in sequential order from stack and BP in random.

3. FLAG REGISTERS:

Here there are 9 flags which occupy a 16 bit register. The flags are:

bit0 = carry flag

bit1 = x

bit2 = parity flag

bit3 = x

bit4 = auxillary flag

bit5 = x

bit6 = zero flag

bit7 = sign flag

bit8 = Trap flag

bit9 = interrupt flag

bit10 = direction flag

bit11 = overflow flag

bit12 = x

bit13 = x

bit14 = x

bit15 = X

FUNCTIONS OF FLAGS:

CARRY FLAG: After any arithmetic and logic operation generates carry. Carry flag will be set else carry flag will be reset

PARITY FLAG: After any arithmetic or logical operation generates even number of 1's then parity flaf will be set.

AUXILLARY CARRY FLAG: After any arithmetic or logical operation generates carry in lower order nibble and goes to higher order nibble then auxillary carry flag will be set.

ZERO FLAG: After any arithmetic or logical operation result generates zero of this flag will set.

SIGN FLAG: The result of any arithmetic or logical operation makes MSB as 1.This flag will set.

TRAP FLAG: Used to trace the 8088 instructions step by step .

INTERRUPT FLAG: Tthis enables interrupt system of microprocessor

DIRECTION FLAG: Increments or decrements memory by resetting or setting this flag.

OVERFLOW FLAG: Whenever arithmetic operation generates overflow this flag will be set.

SOFTWARE DESCRIPTION.

The main objective of this software is to transfer files from one system and also it incorporate remote printing, a new idea which is incorporated only in recent sophisticated software like NOVELL NETWARE and also it has a ON-LINE message transferring and internal loop back test, for testing the serial/parallel interface card.

The software starts with all needed data declaration contents, public declaration external definitions The main procedure starts with finding all system configuration like PC, or PC,XT or AT, determine peripherals, determining memory etc. Now at this point a full screen box is drawn with the following menu.

1. OPTIONS
2. PORT ADDRESS
3. PARAMETERS
4. DOS FUNCTIONS
5. FILES

Now at this point the procedure named as keyboard is called where the user using exit menu of the main menu "OPTIONS" or by pressing escape. Now the first function within keyboard in "FILEGET" which gets the files(s) and displays at the right corner within a box under option "files".

USAGE OF THE KEYBOARD WITHIN THE MENU:

1. Page up: This key is used only for file display which makes one page of files to another page of files. Each page consists of 13 files and it goes only in upward direction which means that the previous page of the files will be displayed.

2. Page down: It is also used only in file function. It display file from one page to another page in downward direction (i.e) displays next page of 13 files.

3. Space bar: This key is used for selection purposes. (ie) selection of files. This is working as a toggle switch. When you press over one file or port address that file or the port address will be selected using their own markers. The former uses incremental numbers and the latter uses the symbol '*'. .

4. Left Arrow : This moves the created cursor towards left direction, option by option. If it reaches the last option it moves the cursor once again to the first option. If the submenu is opened it moves along with the submenu.

5. Right Arrow: This does the above said operations in the reverse direction.

6. Up Arrow : This activates only if submenu is opened. It moves the cursor to the options of the submenu, step by step in a decremental way.

7. Down Arrow : This does the above said operations in the reverse direction.

8. Escape : The escape key closes the submenu provided the submenu is opened, or closes the main window and comes to DOS prompt while submenu is inactive.

Now let us see the different options elaborately.

1. OPTIONS :

This option has the following menu :

(a) Transfer File : Transfers file from one system to another system. It uses programmed mode of data transfer at a rate specified by the user.

(b) Remote Printing : It prints file through another system. It also uses the programmed mode of data transfer at a rate specified by the user.

(c) Send Message : It is an ON line message transfer between two systems. Both the user can simultaneously transfer information from keyboard directly.

(d) Loop back Test : This loop back test activates the internal loop back of UART.

(e) Exit : Exits from the option.

2. PORT ADDRESS :

It can be selected by pressing the space bar. It acts as a toggle switch. When we press the space bar key it puts ' * ' and selects the port address. If we press it again it de-selects port address, provided space bar is pressed on the selected port address. If the space bar is pressed on a new port address it will deselect selected port address by removing ' * '.

3. GET PARAMETER :

It displays baud rate, character length and parity. It has some default values. The default values are baud rate 9600 bps, parity - odd parity, stop bit - 1, character length - 7. It is possible to change the parameters just by pressing enter key over that parameter. It will create an input window and we can input the data, if there is an error it will prompt the same. For this get parameter file is used.

4. DOS COM :

This uses DOS commands. Here only 3 DOS commands are used. They are

(a) Type File : It is used to display a file on the monitor screen, page by page.

(b) Change Directory : It changes the current directory to any other directory specified. If the directory does not exist it will also prompt error message.

(c) Current Directory : It displays the current directory information. For this one file DOS com is used.

5. FILE FUNCTIONS :

These file uses what are the latest trends available in the file display. By pressing a space bar we can select a file against which numbers will be marked in the ascending order. Whenever directory is changed by change directory command the files corresponding to the directory will be displayed.

The process of communication starts when all the required options are selected by means of the above said operations. First the status of the other terminal is checked by means of sending the handshaking signals. When the system is not ready it requests the first terminal to wait. The first terminal waits until the second terminal gets ready. When the second terminal is ready it sends a status signal to the first terminal that it is ready. Then the required operation is performed.

If the operation is to transmit the data, the

selected file is transmitted to the other terminal through the Transmit Hold Register (THR) of the UART chip. The data that is transmitted is stored in the Receive Hold Register (RHR) of the second terminal. Then an interrupt signal is sent from the UART to the system that it holds some data's to be stored. Then the interrupt is acknowledged and the transmitted file is stored in one particular location.

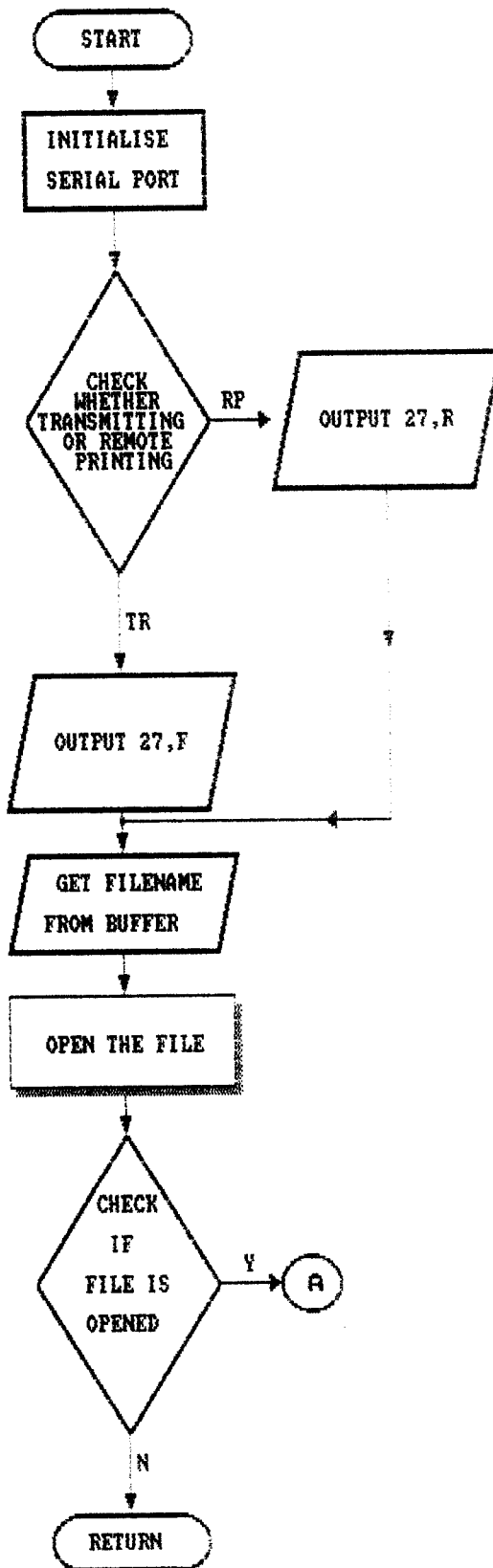
In the case of remote printing the file that is transmitted, instead of storing in the memory location it directly goes to the printer that is connected to the CENTRONIX interface and the file is printed.

In the case of send message, the adapter card acts as a direct link between the two computers. The data that is typed by the first user is displayed on both the first terminal monitor and also in the second terminal monitor. Likewise the data that are typed by the other user is also displayed in the same manner. Thus this proves to be more useful in communicating with two terminals.

The loop back test that is provided is used to check the operation of the adapter card. In this the file that is selected, is transmitted and received in the same monitor, (i.e) the transmission and reception of the file is displayed in the same monitor. This proves to be a good method for testing purposes.

Thus the software written, proves to be a very good software since it can be used not only by technical people but also by a common man.

FLOW CHART



SERIALPRO.ASM

Main program which calls all other functions, written in other files and also incorporates required functions.

```
if1
include mymacro.asm
endif
CODE SEGMENT public 'code'
    ASSUME CS:CODE
    ORG 100H
    extrn baudvalue :word
    extrn parityaddress:word
    extrn stop:word
    extrn charlen:word
    extrn lcrvalue:byte
    extrn getparams:near
    extrn change:byte
    extrn FIND :near
    extrn MON_ADDR :word
    extrn BASE_ADDR :word
    extrn status_port :word
    extrn terminal:near
    extrn doscomm:near
    spc equ 39h
    da equ 50h
    ua equ 48h
    ra equ 04dh
    la equ 04bh
    pu equ 49h
    pd equ 51h
    hm equ 47h
    ed equ 04fh
    cr equ 01ch
    esc_k equ 01h
    STT:JMP MAIN
    star dw ?
    comaddress dw 03f8h
                dw 02f8h
                dw 0
                dw 0
    portvalue dw ?
    starstore dw ?
    countvalue db ?
    setcursor db ?
    value1 db ?
    portaddr dw ?
    filelocation dw ?
    storecount dw ?
    close dw ?
    endaddr dw ?
    startaddr dw ?
    FILETOP DW ?
```

```

FILEBOT DW ?
filerow dw ?
filecol dw ?
dollar db '$'
area db 256*13 dup (?)
count dw ?
value dw ?
savecol dw ?
saverow dw ?
rcount dw 19
lcount dw 50
asciiz db '.*',0
colstart dw ?
colend dw ?
space dw 50 dup (?)
selcount db ?
pcount db ?
public
portaddr,space,area,row,col,stringdisp,display,char,addr,selcount
public countvalue,attr,saverow,savecol,index,portvalue,hexad
string1 db 'OPTIONS$'
STRING2 DB 'PORTADDR$'
STRING3 DB 'Parameters$'
string4 db 'Dos utility$'
string5 db 'Files$'
coll dw ?
      dw ?
      dw ?
      dw ?
row1 dw ?
ROW Dw ?
COL Dw ?
presentrow dw ?
presentcol dw ?
nextaddr dw ?
presentaddr dw ?
CHAR DB ?
ATTR DB ?
INDEX Dw ?
SUBINDEX Dw ?
MAININDEX DB ?
STATUS DB ?
rsize dw ?
csize dw ?
addr dw ?
file dw ?
filecount dw ?
dispcount db ?
colright dw ?
rowbot dw ?
winleft dw ?
wintop dw ?
dollar1 db '$'
options db 'Transmit data$'
          db 'Remote printing$'

```

```

        db 'Send message$'
        db 'Receive$'
        db 'Loop back test$'
        db 'Exit$'
label1 label byte
dollar2 db '$'
portaddr db 'Com 1$'
         db 'Com 2$'
         db 'com 3$'
         db 'com 4$'
label2 label byte
dollar3 db '$'
parameters db 'Baud rate$'
           db 'parity$'
           db 'Stop bits$'
           db 'Char length$'
label3 label byte
dollar4 db '$'
        dosutility db 'type file$'
           db 'change directory$'
           db 'current directory$'
        label4 label byte

submenu dw offset options
        dw offset portaddr
        dw offset parameters
        dw offset dosutility
termination dw offset label1
           dw offset label2
           dw offset label3
           dw offset label4

addrtable dw offset string1
          dw offset string2
          dw offset string3
          dw offset string4
          dw offset string5

address label word
addr1 db '9600$'
addr2 db 'ODD$'
L_middle equ 'L'
r_middle equ '9'
LE TO COR EQU 'I'
HOR EQU 'M'
VER EQU ':'
RI TO COR EQU ';'
LE BO COR EQU 'H'
RI BO COR EQU '<'
MAIN PROC
mov ax,offset addr1
mov baudvalue,ax
mov ax,offset addr2
mov parityaddress,ax
mov stop,1
mov charlen,7
CALL FIND

```

```

MOV ES,MON_ADDR
XOR DI,DI
MOV CX,80*25
MOV AX,0720H
REP STOSW
push es
mov ax,40h
mov es,ax
mov ax,es:60h
mov setcursor,ah
pop es
mov dx,base_addr
mov al,10
out dx,al
inc dx
mov al,20h
out dx,al
mov row,0
mov col,20
mov attr,70h
@string <'PROJECT WORK ON SERIAL PORT$'>
  mov row,2
  mov col,3
  MOV CHAR,LE_TO_COR
  MOV ATTR,07H
11: CALL DISPLAY
  MOV CHAR,hor
  inc col
  cmp col,75
  jb 11
  mov char,ri_to_cor
12: call display
  mov char,ver
  inc row
  cmp row,22
  jb 12
  mov char,ri_bo_cor
13: call display
  mov char,hor
  dec col
  cmp col,3
  ja 13
  mov char,le_bo_cor
14: call display
  mov char,ver
  dec row
  cmp row,2
  ja 14
  mov row,4
  mov char,l_middle
15: call display
  mov char,hor
  inc col
  cmp col,75
  jb 15

```

```

mov char,r_middle
call display
mov cx, offset addrend-offset addrtable
shr cx,1
xor bx,bx
mov col,6
mov row,3
mov si,row
mov row1,si
yet_loop:mov si,col
        mov [bx+col1],si
        mov si,[bx+addrtable]
mov addr,si
CALL STRINGDISP
add col,5
add bx,2
loop yet_loop
mov row,3
mov index,0
mov attr,70h
mov bx,0
mov si,[bx+col1]
mov col,si
mov si,[bx+addrtable]
mov addr,si
mov ah,0
int 16h
call stringdisp
call keyboard
@clrscr
mov dx,base_addr
mov al,10
out dx,al
inc dx
mov al,0
or al,setcursor
out dx,al
ret
main endp
display proc
push ax
push dx
push bx
push cx
push di
push es
mov es,mon_addr
mov dx,status_port
retrace:in al,dx
test al,1
jnz retrace
noretrace:in al,dx
test al,1
jz noretrace
mov di,row

```

```

shl di,1
shl di,1
add di,row
mov cl,4
shl di,cl
add di,col
shl di,1
mov ah,attr
mov al,char
stosw
pop es
pop di
pop cx
pop bx
pop dx
pop ax
ret
display endp
STRINGDISP PROC
PUSH AX
PUSH BX
push si
mov si,addr
CLD
m_s_disp:lodsb
cmp al,'$'
je finish
mov char,al
call display
inc col
jmp m_s_disp
finish:mov colend,si
pop si
pop bx
pop ax
ret
stringdisp endp
keyboard proc
push ax
push si
push bx
mov ax,row
mov filerow,ax
mov ax,col
mov filecol,ax
call fileget
getkey:mov ax,row
mov presentrow,ax
mov ax,col
mov presentcol,ax
mov ax,filerow
mov row,ax
mov ax,filecol
mov col,ax
mov ah,0

```

```

int 16h
cmp ah,pd
jz pl_cont
jmp nopd
pl_cont:push ds
pop es
mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
cmp index,4*2
je y_file
jmp getkey
y_file:
mov ax,[col1+4]
sub ax,5
shl ax,1
add ax,2
mov col,ax
mov ax,filebot
cmp ax,count
jae getkey
MOV FILETOP,AX
MOV row,6
mov attr,07
@delwindow ROW,COL,RSIZE,CSIZE,char
mov si,endaddr
mov startaddr,si
mov colstart,si
lk2:inc filebot
MOV AX,COUNT
CMP FILEBOT,AX
jbe filesr
mov endaddr,si
jmp enddisp
filesr:mov addr,si
mov bx,col
call stringdisp
mov col,bx
add col,15
mov ax,si
mov pcount,0
mov di,offset space
MOV CL,SELCOUNT
CMP CL,0
JnZ lo2
inc col
JMP ncont
lo2:inc pcount
scasw
jz llw1
loop lo2
inc col
jcxz ncont
llw1:mov al,pcount

```

```

mov si,startaddr
mov endaddr,si
lf1:std
dec filetop
cmp filetop,0
jGE y1_more
mov filetop,0
mov filebot,13
jmp anacin
y1_more:sub si,2
reverse:lodsb
cmp al,'$'
je fi_disp
jmp reverse
fi_disp:add si,2
mov addr,si
mov bx,col
call stringdisp
mov col,bx
add col,15
mov ax,si
mov pcount,0
mov di,offset space
MOV CL,SELCOUNT
CMP CL,0
JnZ lo3
inc col
JMP ncont1
lo3:inc pcount
scasw
jz llw2
loop lo3
inc col
jcxz ncont1
llw2:mov al,pcount
aam
or ax,3030h
mov char,ah
call display
inc col
mov char,al
call display
ncont1:sub col,16
dec row
cmp row,6
jb anacin
jmp lf1
anacin:mov ax,filetop
mov filecount,ax
mov startaddr,si
mov colstart,si
mov row,6
mov attr,70h
mov addr,si
mov bx,col

```



```

call stringdisp
mov col,bx
jmp getkey

nupu:cmp ah,1a
jz yla
jmp nola
yla:cmp index,0*2
jnz decindex
mov bx,index
mov close,bx
mov index,4*2
jmp nextw
decindex:mov bx,index
mov close,bx
sub index,2
nextw:mov si,[bx+col1]
mov col,si
mov si,[bx+addrtable]
mov addr,si
mov attr,07h
call stringdisp
mov bx,index
mov si,[bx+col1]
mov col,si

cmp bx,1*2
jne n_col_ind
cmp star,0
je n_col_ind
mov ax,row
push ax
mov ax,starstore
mov row,ax
add col,12
mov attr,07h
mov char,'*'
call display
pop ax
mov row,ax
sub col,12

n_col_ind:
mov si,[bx+addrtable]
mov addr,si
mov attr,70h
call stringdisp
cmp status,1
je ysubme
jmp nosubme
ysubme:
cmp close,4*2
jne yclose
mov si,colstart
mov ax,presentrow

```

```

mov row,ax
mov ax,presentcol
mov col,ax
mov addr,si
mov attr,07
mov bx,col
call stringdisp
mov col,bx
jmp nclose
yclose:mov row,5
mov ax,winleft
mov col,ax
mov attr,07
add rowbot,2
add colright,2
@delwindow row,col,rowbot,colright,char
nclose:mov status,00
mov dispcount,0
jmp y_da
nosubme:jmp getkey

```

```

nola:cmp ah,ra
jz yra
jmp nora
yra:cmp index,4*2
jnz incindex
mov bx,index
mov index,0
jmp next2
incindex:mov bx,index
add index,2
next2:mov close,bx
mov si,[bx+addrtable]
mov addr,si
mov si,[bx+col1]
mov col,si
mov attr,07h
mov bx,col
call stringdisp
mov col,bx
mov bx,index
mov si,[bx+col1]
mov col,si

```

```

cmp index,1*2
jne n_col_index
cmp star,0
je n_col_index
mov ax,row
push ax
mov ax,starstore
mov row,ax
add col,12
mov attr,07h

```

```

mov char,'*'
call display
pop ax
mov row,ax
sub col,12

n_col index:mov si,[bx+addrtable]
mov addr,si
mov attr,70h
mov bx,col
call stringdisp
mov col,bx
cmp status,1
je ysubmenu
jmp nosubmenu
ysubmenu:cmp close,4*2
jne statusok
mov si,colstart
mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
mov addr,si
mov attr,07
mov bx,col
call stringdisp
mov col,bx
mov close,0
jmp eraseno
statusok:mov row,5
mov ax,winleft
mov col,ax
mov attr,07
add rowbot,2
add colright,2
@delwindow row,col,rowbot,colright,char
eraseno:mov status,0
mov dispcount,0
jmp y_da
nosubmenu:jmp getkey

nora:cmp ah,hm
jz yhm
jmp nohm
yhm:mov bx,index
mov close,bx
mov index,0*2
mov si,[bx+addrtable]
mov addr,si
mov si,[bx+col1]
mov col,si
mov attr,07h
call stringdisp
mov bx,index
mov si,[bx+col1]

```

```

mov col,si
mov si,[bx+addrtable]
mov addr,si
mov attr,70h
call stringdisp
cmp status,1
je displayok
jmp nodisplay
displayok:cmp close,4*2
jne closeplease
mov si,colstart
mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
mov addr,si
mov attr,07
mov bx,col
call stringdisp
mov col,bx
jmp closenot
closeplease:mov row,5
mov ax,winleft
mov col,ax
mov attr,07
add rowbot,2
add colright,2
@delwindow row,col,rowbot,colright,char
closenot:mov status,0
mov dispcount,0
jmp option
nodisplay:jmp getkey

```

```

nohm:
cmp ah,ed
jz yed
jmp noed
yed:cmp index,4*2
jne noid
jmp noerase
noid:mov bx,index
mov close,bx
mov index,4*2
mov si,[bx+addrtable]
mov addr,si
mov si,[bx+col1]
mov col,si
mov attr,07h
call stringdisp
mov bx,index
mov si,[bx+col1]
mov col,si
mov si,[bx+addrtable]
mov addr,si

```

```

mov attr,70h
call stringdisp
cmp status,1
je eraseok
jmp noerase
eraseok:
mov row,5
mov ax,winleft
mov col,ax
mov attr,07
add rowbot,2
add colright,2
@delwindow row,col,rowbot,colright,char
mov status,0
mov dispcount,0
jmp eddisp
noerase:mov bx,presentrow
mov row,bx
mov bx,presentcol
mov col,bx
jmp getkey

noed:
cmp ah,da
je y_da
jmp noda
y_da:mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
cmp index,4*2
je filefunc
cmp index,0*2
jne what1
jmp option
what1:cmp index,1*2
jne what2
jmp getport
what2:cmp index,2*2
jne what3
jmp getparam
what3:cmp index,3*2
jne what4
jmp dosfunct
what4:jmp getkey
filefunc:mov ax,count
cmp ax,filecount
jae m1
jmp getkey
m1:
cmp row,18
jb move
jmp getkey
move:mov attr,07
cmp status,1

```

```

je lop
eddisp:mov ax,[col1+4]
sub ax,5
shl ax,1
MOV col,ax
add col,2
MOV row,6
mov status,1
mov attr,70h
mov si,startaddr
mov addr,si
mov bx,col
call stringdisp
mov col,bx
mov colstart,si
jmp getkey
lop:inc filecount
mov ax,filecount
cmp ax,count
jb disp1
dec filecount
jmp getkey
disp1:mov si,colstart
mov addr,si
mov bx,col
call stringdisp
mov col,bx
inc row
cmp row,18
jbe nkeyget
jmp getkey
nkeyget:mov si,colend
mov addr,si
mov colstart,si
mov attr,70h
mov bx,col
call stringdisp
mov col,bx
jmp getkey

option:cmp dispcount,6
jbe ok
jmp getkey
ok:cmp status,1
je open
jmp noopen
open:
inc dispcount
cmp dispcount,6
jbe dok
dec dispcount
jmp getkey
dok:mov si,colstart
mov attr,07
mov addr,si

```

```

mov bx,col
call stringdisp
mov col,bx
mov si,colend
inc row
mov addr,si
mov colstart,si
mov attr,70h
mov bx,col
call stringdisp
mov col,bx
jmp getkey

```

```

noopen:mov status,1
mov dispcount,1
mov row,5
mov bx,index
mov ax,[bx+col1]
mov col,ax
mov ax,col
mov winleft,ax
mov ax,row
mov wintop,ax
add ax,7
mov rowbot,ax
mov ax,col
add ax,20
mov colright,ax
mov attr,07
@drawbox row,col,rowbot,colright,char
add col,2
inc row
mov si,offset options
mov attr,70h
mov colstart,si
more_to_print:mov addr,si
mov bx,col
call stringdisp
mov col,bx
mov si,colend
inc row
inc dispcount
mov attr,07
cmp dispcount,6
jle more_to_print
mov dispcount,1
mov row,6
jmp getkey

```

```

getport:cmp dispcount,4
jbe can
jmp getkey
can:cmp status,1
je port_disp_open
jmp port_disp_nopen

```

```

port_disp_open:inc dispcount
cmp dispcount,4
jbe can1
dec dispcount
jmp getkey
can1:mov si,colstart
mov addr,si
mov attr,07h
mov bx,col
call stringdisp
mov col,bx
inc row
mov si,colend
mov addr,si
mov colstart,si
mov attr,70h
mov bx,col
call stringdisp
mov col,bx
jmp getkey

port_disp_nopen:mov status,1
mov dispcount,1
mov row,5
mov bx,index
mov ax,[bx+col1]
mov col,ax
mov winleft,ax
mov ax,row
mov wintop,ax
add ax,5
mov rowbot,ax
mov ax,col
add ax,15
mov colright,ax
mov attr,07
@drawbox row,col,rowbot,colright,char
add col,2
inc row
mov si,offset portaddr
mov attr,70h
mov colstart,si
more_to_dip:mov addr,si
mov bx,col
call stringdisp
mov col,bx
mov si,colend
inc row
mov attr,07
inc dispcount
cmp dispcount,4
jle more_to_dip
mov dispcount,1
mov row,6
jmp getkey

```



```

getparam:cmp dispcount,4
jle more_display
jmp getkey
more_display:cmp status,01
je param_open
jmp param_nopen
param_open:
inc dispcount
cmp dispcount,4
jle anacin2
dec dispcount
jmp getkey
anacin2:mov si,colstart
mov addr,si
mov attr,07h
mov bx,col
call stringdisp
mov col,bx
inc row
mov si,colend
mov addr,si
mov colstart,si
mov attr,70h
mov bx,col
call stringdisp
mov col,bx
jmp getkey

param_nopen:mov status,1
mov dispcount,1
mov row,5
mov bx,index
mov ax,[bx+col1]
mov col,ax
mov winleft,ax
mov ax,row
mov wintop,ax
add ax,5
mov rowbot,ax
mov ax,col
add ax,19
mov colright,ax
mov attr,07
@drawbox row,col,rowbot,colright,char
add col,2
inc row
mov si,offset parameters
mov attr,70h
mov colstart,si
more_param_disp:mov addr,si
mov bx,col
call stringdisp
mov col,bx
mov si,colend

```

```

inc row
inc dispcount
mov attr,07
cmp dispcount,4
jle more_param_disp
mov attr,87h
mov ax,row
mov presentrow,ax
mov ax,col
mov presentcol,ax
mov row,6
add col,12
MOV AX,BAUDVALUE
MOV ADDR,AX
MOV BX,COL
CALL STRINGDISP
MOV COL,BX
INC ROW
MOV AX,PARITYADDRESS
MOV ADDR,AX
MOV BX,COL
CALL STRINGDISP
MOV COL,BX
INC ROW
MOV AX,30H
ADD AX,STOP
MOV CHAR,AL
CALL DISPLAY
INC ROW
MOV AX,30H
ADD AX,CHARLEN
MOV CHAR,AL
CaLL DISPLAY
MOV AX,PRESENTROW
MOV ROW,AX
MOV AX,PRESENTCOL
MOV COL,AX
mov attr,07h
mov dispcount,1
mov row,6
jmp getkey

dosfunct:cmp dispcount,3
jle more_print
jmp getkey
more_print:cmp status,1
je dos_open
jne dos_nopen
dos_open:
inc dispcount
cmp dispcount,3
jle anacin3
dec dispcount
jmp getkey
anacin3:mov si,colstart

```

```

mov addr,si
mov attr,07h
mov bx,col
call stringdisp
mov col,bx
inc row
mov si,colend
mov addr,si
mov colstart,si
mov attr,70h
mov bx,col
call stringdisp
mov col,bx
jmp getkey

```

```

dos_nopen:mov status,1
mov dispcount,1
mov row,5
mov bx,index
sub bx,2
mov ax,[bx+col1]
mov col,ax
mov winleft,ax
mov ax,row
mov wintop,ax
add ax,5
mov rowbot,ax
mov ax,col
add ax,19
mov colright,ax
mov attr,07
@drawbox row,col,rowbot,colright,char
add col,2
inc row
mov si,offset dosutility
mov attr,70h
mov colstart,si
more dos_disp:mov addr,si
mov bx,col
call stringdisp
mov col,bx
mov si,colend
inc row
inc dispcount
mov attr,07
cmp dispcount,3
jle more_dos_disp
mov row,6
mov dispcount,1
jmp getkey

```

```

noda:cmp ah,ua
jz y_ua
jmp noua
Y-UA:cmp status,1

```

```

je yplease
jmp getkey
yplease:MOV AX,PRESENTROW
MOV ROW,AX
MOV AX,PRESENTCOL
MOV COL,AX
cmp index,4*2
je filecun
jmp normal
filecun:mov ax,count
cmp ax,filecount
jae dodir
jmp getkey
dodir:mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
cmp row,6
ja dora
jmp getkey
dora:mov attr,07
cmp status,1
je lop1
mov ax,[col+4]
sub ax,5
shl ax,1
MOV col,ax
add col,2
MOV row,6
mov status,1
mov attr,70h
mov si,startaddr
mov addr,si
mov bx,col
call stringdisp
mov col,bx
mov colstart,si
jmp getkey
lop1:dec filecount
mov ax,filecount
cmp ax,0
jGe disp2
mov status,0
inc filecount
jmp getkey

disp2:mov si,colstart
mov addr,SI
mov attr,07
mov bx,col
call stringdisp
mov col,bx
dec row
cmp row,6
jae dcont

```

```

jmp getkey

dcont:mov si,colstart
std
sub si,2
nod:lodsb
cmp al,'$'
jnz nod
add SI,2
cld
MOV COLSTART,SI
mov attr,70h
mov addr,si
mov bx,col
call stringdisp
mov col,bx
jmp getkey

normal:cmp dispcount,1
jgE yesprint
jmp getkey
yesprint:dec dispcount
cmp dispcount,1
jge nocz
mov dispcount,1
jmp getkey
nocz:mov si,colstart
mov attr,07
mov addr,si
mov bx,col
call stringdisp
mov col,bx
sub si,2
dec row
std
not$:lodsb
cmp al,'$'
jne not$
add si,2
mov colstart,si
mov addr,si
mov attr,70h
mov bx,col
call stringdisp
mov col,bx
jmp getkey

noua:push ds
pop es
cmp ah,spc
jz y_spc
jmp n_spc
y_spc:mov ax,presentrow
mov row,ax

```

```

mov ax,presentcol
mov col,ax
cmp index,4*2
jne do_addr_test
jmp do_fi_select
do_addr_test:cmp index,1*2
je do_port_addr
jmp getkey
do_port_addr:mov attr,07h
mov char,'*'
cmp star,1
jne nostar
mov ax,starstore
cmp ax,row
jne ystar
add col,10
mov char,20h
call display
mov portaddr,0
mov storecount,0
mov star,0
jmp getcol
ystar:add col,10
mov char,20h
mov ax,row
push ax
mov ax,starstore
mov row,ax
call display
pop ax
mov row,ax
sub col,10
nostar:mov char,'*'
add col,10
mov star,1
call display
xor bh,bh
mov bl,dispcount
dec bx
shl bx,1
mov portvalue,bx
mov ax,[bx+comaddress]
mov portaddr,ax
mov ax,row
mov starstore,ax
getcol:mov ax,presentcol
mov col,ax
jmp getkey
do_fi_select:
xor cx,cx
mov ax,colstart
mov di,offset space
mov cl,selcount
cmp cl,0
jne nze

```

```

jmp yze
nze:repne scasw
jz yfind
jcxz yze
yfind:add col,15
sub di,2
mov [di],word ptr 0
dec selcount
mov attr,07
mov char,20h
call display
mov char,20h
inc col
call display
sub col,16
call reorder
call select
jmp getkey
yze:add col,15
stosw
inc selcount
mov al,selcount
aam
or ax,3030h
mov attr,07
mov char,ah
call display
mov char,al
inc col
call display
sub col,16
jmp getkey

n_spc:cmp ah,cr
jz ycr
jmp nocr
ycr:mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
cmp status,1
je ywindow
jmp getkey
ywindow:cmp index,0*2
jne noprint
cmp dispcount,6
jne noexit
mov status,0
@string <'Exiting Menu $'>
jmp trystatus
noexit:mov al,dispcount
mov countvalue,al
call terminal
jmp nodos
noprint:cmp index,3*2

```

```

jne testparam
mov al,dispcount
mov countvalue,al
call doscomm
cmp change,1
jne nodos2
jmp filefunction
testparam:cmp index,2*2
je yedos
nodos2:jmp getkey
yedos:mov al,dispcount
mov countvalue,al
call getparams
mov attr,87h
mov ax,row
mov presentrow,ax
mov ax,col
mov presentcol,ax
mov row,6
add col,12
MOV AX,BAUDVALUE
MOV ADDR,AX
CALL DELLINE
MOV BX,COL
CALL STRINGDISP
MOV COL,BX
INC ROW
MOV AX,PARITYADDRESS
MOV ADDR,AX
CALL DELLINE
MOV BX,COL
CALL STRINGDISP
MOV COL,BX
INC ROW
MOV AX,30H
ADD AX,STOP
MOV CHAR,AL
CALL DISPLAY
INC ROW
MOV AX,30H
ADD AX,CHARLEN
MOV CHAR,AL
CaLL DISPLAY
MOV AX,PRESENTROW
MOV ROW,AX
MOV AX,PRESENTCOL
MOV COL,AX
mov attr,07
nodos1:jmp getkey
filefunction:mov ax,row
mov presentrow,ax
mov ax,col
mov presentcol,ax
mov si,colstart
push si

```



```

call fileget
pop si
mov colstart,si
mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
nodos:
jmp getkey

nocr:cmp ah,esc_k
jz trystatus
mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
jmp getkey
trystatus:mov ax,presentrow
mov row,ax
mov ax,presentcol
mov col,ax
cmp status,1
je windowopen
jmp noesc
windowopen:cmp index,4*2
jne closewin
jmp getkey
closewin:mov row,5
mov ax,winleft
mov col,ax
mov attr,07
add rowbot,2
add colright,2
@delwindow row,col,rowbot,colright,char
mov status,0
mov dispcount,0
forever:jmp getkey
noesc:pop bx
pop si
pop ax
ret
keyboard endp

```

```

fileget proc
push ax
push si
push di
push cx
push es
push ds
pop es
mov selcount,0
cmp change,1
je chg

```

```

jmp nchg
chg:mov ax,[col1+4]
sub ax,5
shl ax,1
add ax,2
mov col,ax
mov row,6
@delwindow row,col,rsize,csize,char
jmp fichan
nchg:mov ax,row
mov filerow,ax
mov ax,col
mov filecol,ax
fichan:mov count,0
mov change,0
mov filecount,0
mov di,offset area
@findfirst asciiz
domore:inc count
mov si,80h
add si,1eh
more:lodsb
cmp al,0
jz over
stosb
jmp more
over:mov ax,count
mov filecount,ax
mov al,'$'
stosb
@findnext value
cmp value,12h
jnz domore
mov ax,[col1+4]
sub ax,5
shl ax,1
mov col,ax
add ax,20
mov csize,ax
mov rsize,20
mov row,5
@drawbox row,col,rsize,csize
add col,2
mov row,6
mov ax,col
mov di,offset area
mov startaddr,di
mov colstart,di
lk1:mov addr,di
mov ax,col
push ax
call stringdisp
pop ax
mov col,ax
mov al,'$'

```

```

mov cx,13
repne scasb
dec filecount
jle norow
inc row
cmp row,19
jb lk1
jmp lk2
norow:inc row
lk2:mov endaddr,di
mov ax,row
sub ax,6
mov filebot,ax
mov filetop,0
  mov ax,filerow
mov row,ax
mov ax,filecol
mov col,ax
mov filecount,0
pop es
pop cx
pop di
pop si
pop ax
ret
fileget endp

```

```

select proc
push ax
push bx
push cx
push si
push di
mov ax,filetop
mov storecount,ax
mov pcount,0
xor cx,cx
mov ax,presentcol
mov col,ax
mov row,6
mov attr,07
@delwindow row,col,rsize,csize,char
mov si,startaddr
MOREDISP:mov di,offset space
mov addr,si
mov bx,col
call stringdisp
mov col,bx
add col,15
mov ax,si
MOV CL,SELCOUNT
CMP CL,0

```

```

JnZ lo1
inc col
JMP ncount
lo1:inc pcount
scasw
jz llw
loop lo1
inc col
jcxz ncount
llw:mov al,pcount
aam
or ax,3030h
mov char,ah
call display
inc col
mov char,al
call display
ncount:sub col,16
no$:lodsb
cmp al,'$'
jnz no$
MOV PCOUNT,0
inc storecount
mov ax,count
cmp storecount,ax
jb incrow
jmp discomt
incrow:inc row
cmp row,18
jbe moretodisp
jmp discomt
moretodisp:jmp moredisp
discomt:mov ax,presentrow
mov row,ax
MOV SI,COLSTART
MOV ADDR,SI
MOV ATTR,70H
MOV BX,COL
CALL STRINGDISP
MOV COL,BX
MOV ATTR,07
pop di
pop si
pop cx
pop bx
pop ax
ret
select endp

reorder proc
push ax
push si
push di
push cx
mov cl,selcount

```

```

cmp cl,0
je nodata
mov di,offset space
mov si,di
lfor:lodsw
cmp ax,0
je lfor
stosw
nostore:loop lfor
nodata:pop cx
pop di
pop si
pop ax
ret
reorder endp

```

```

DELLINE PROC
PUSH AX
PUSH CX
PUSH ES
MOV ES,MON_ADDR
MOV DI,ROW
SHL DI,1
SHL DI,1
ADD DI,ROW
MOV CL,4
SHL DI,CL
ADD DI,COL
SHL DI,1
MOV AX,0720H
MOV CX,5
REP STOSW
POP ES
POP CX
POP ES
RET
DELLINE ENDP

```

```

hexad proc
    push ax
    push dx
    push cx
    mov dh,ah
    and dh,11110000b
    mov cl,4
    shr dh,cl
    cmp dh,9
    jbe add30
    add dh,07
    add30:add dh,30h

```

```

@dispc dh
mov dh,ah
and dh,00001111b
cmp dh,9
jbe add3
add dh,07
add3:add dh,30h
@dispc dh
mov dl,al
mov cl,04
shr dl,cl
cmp dl,9
jbe add31
add dl,07
add31:add dl,30h
@dispc dl
mov dl,al
and dl,00001111b
cmp dl,9
jbe add32
add dl,07
add32:add dl,30h
@dispc dl
pop cx
pop dx
pop ax
ret

```

```

hexad endp
code ends
end stt

```

MYMACRO.ASM

Consists of all the required macros for the main function and subroutines.

```
@getbuffer macro pram
push ds
push ax
push dx
push cs
pop ds
mov dx,offset pram
mov ah,0ah
int 21h
pop dx
pop ax
pop ds
endm

@buffer macro par1,par2,par3,par4
par1 db par4
par2 db ?
par3 db par4 dup(?)
endm

@mesg macro str1,str2,str3,str4
local msg,jump
jmp jump
msg db str1,str2,str3,str4
jump:push ds
push dx
push ax
push cs
pop ds
mov ah,09H
mov dx,offset msg
int 21h
pop ax
pop dx
pop ds
endm

@dispc macro par
push dx
push ax
mov ah,02
mov dl,par
int 21h
pop ax
pop dx
endm
@crlf macro
push ax
```

```
push dx
mov ah,02
mov dl,10
int 21h
mov dl,13
int 21h
pop dx
pop ax
endm
```

```
@open macro param1,handle
push ds
push ax
push dx
push cs
pop ds
mov al,0
mov ah,03dh
mov dx,offset param1
int 21h
mov handle,ax
pop dx
pop ax
pop ds
endm
```

```
@read macro param3,handle,val
push ds
push ax
push cx
push dx
push bx
push cs
pop ds
mov bx,handle
mov dx,offset param3
mov cx,1
mov ah,03fh
int 21h
mov val,ax
pop bx
pop dx
pop cx
pop ax
pop ds
endm
```

```
@close macro param4
push ds
push ax
push bx
push cs
pop ds
mov ah,03eh
mov bx,param4
```



```
int 21h
pop bx
pop ax
pop ds
endm
```

```
@dispm macro par
push ds
push ax
push dx
push cs
pop ds
mov ah,09h
mov dx,offset par
int 21h
pop dx
pop ax
pop ds
endm
```

```
@mess macro get1,get2
local disp,yet
push ds
push ax
push di
push dx
push bx
push cx
push cs
pop ds
mov di,offset get1
mov bl,get2
yet:dec bl
je disp
mov al,'$'
mov cx,65535
repne scasb
jmp yet
disp:mov dx,di
mov ah,09h
int 21h
pop cx
pop bx
pop dx
pop di
pop ax
pop ds
endm
```

```
@gotoxy macro r,c
push ax
push dx
push bx
mov ah,02
mov dh,r
```

```
mov dl,c
mov bx,0
int 10h
pop bx
pop dx
pop ax
endm
```

```
@cls macro
push ax
push dx
push cx
push bx
mov ah,02
mov bx,0
mov dx,0
int 10h
mov cx,80*25
mov ah,09
mov bh,0
mov bl,07
mov al,20h
int 10h
pop bx
pop cx
pop dx
pop ax
endm
```

```
@dispbios MACRO CH
push ax
push dx
push cx
push bx
mov cx,1
mov ah,09
mov bh,0
mov bl,07
mov al,ch
int 10h
pop bx
pop cx
pop dx
pop ax
endm
```

```
@allocatmem macro para1,addr
push ax
push bx
mov ah,48h
mov bx,para1
int 21h
mov addr,ax
pop bx
pop ax
```

```

endm

@findfirst macro param
push ax
push dx
mov dx,offset param
mov ah,04eh
mov cx,0
int 21h
pop dx
pop ax
endm

@findnext macro value
push ax
mov ah,04fh
int 21h
mov value,ax
pop ax
endm

@delwindow macro rw,co,mr,mc,char
local l11
push ax
mov ax,rw
mov saverow,ax
mov ax,co
mov savecol,ax
mov char,20h
l11:call display
inc co
mov ax,mc
sub ax,2
cmp ax,co
jae l11
mov ax,savecol
mov co,ax
inc rw
mov ax,mr
sub ax,2
cmp ax,rw
jae l11
mov ax,saverow
mov rw,ax
mov ax,savecol
mov co,ax
pop ax
endm

@drawbox macro rw,co,rs,cos
local ls1,next1,ls2,ls3
push ax
push dx
push cx
mov dx,rw

```

```

mov cx,co
mov char,le_to_cor
mov attr,07h
ls1:call display
mov char,hor
mov ax,cos
inc co
cmp co,ax
jb ls1
mov char,ri_to_cor
next1:call display
mov char,ver
mov ax,rs
inc rw
cmp rw,ax
jb next1
mov char,ri_bo_cor
ls2:call display
mov char,hor
dec co
cmp co,cx
ja ls2
mov char,le_bo_cor
ls3:call display
mov char,ver
dec rw
cmp rw,dx
ja ls3
mov co,cx
mov rw,dx
pop cx
pop dx
pop ax
endm

```

```

@getscreen macro location
push es
push ds
push si
push di
mov cx,79*24
push ds
pop es
mov di,offset location
mov si,mon_addr
mov ds,si
xor si,si
rep movsw
pop di
pop si
pop ds
pop es
endm

```

```

@clrscr macro

```

```

push es
push di
push cx
push ax
mov es,mon_addr
xor di,di
mov ax,0720h
mov cx,2000
rep stosw
pop ax
pop cx
pop di
pop es
endm

```

```

@putscreen macro location
push es
push ds
push si
push di
push cs
pop ds
mov cx,79*24
mov es,mon_addr
xor di,di
mov si,offset location
rep movsw
pop di
pop si
pop ds
pop es
endm

```

```

@string macro messm
local jump,message
jmp jump
message db messm
jump:push si
push ds
push cs
pop ds
mov si,offset message
mov addr,si
call stringdisp
pop ds
pop si
endm

```

```

@atoi macro PARA1,PARA2
local ll
PUSH AX
PUSH SI

```

```

PUSH CX
push bx
MOV SI,OFFSET PARA1
XOR CH,CH
mov para2,0
MOV CL,[SI-1]
MOV BX,10
ll:mov ax, para2
MUL BX
mov para2,ax
lods b
and al,0fh
xor ah,ah
add para2,ax
loop ll
pop bx
pop cx
pop si
pop ax
endm

```

```

@strcmp macro pot1,pot2,var
local over,nocomp,comp,ter,test1
push ax
mov var,1
comp:mov al,[di]
mov bl,[si]
@lower bl
@lower al
cmp al,bl
jne nocomp
cmp al,'$'
je test1
inc di
inc si
jmp comp
nocomp:mov var,0
        cmp [si],byte ptr '$'
        je ter
        inc si
        jmp nocomp
ter:    inc si
jmp over
test1:cmp [si],byte ptr '$'
je over
mov var,0
over :pop ax
endm

```

```

@lower macro char
local lower
cmp char,'$'
je lower

```

```
cmp char,'a'  
jae lower  
add char,20h  
lower:nop  
endm
```

TERMINAL.ASM

This incorporates routines for remote printing, transmitting the file and sending messages between the systems.

```
if1
include mymacro.asm
endif

code segment public 'code'
assume cs:code
BUFSIZ EQU 2048
ctrz equ 01ah
picmask equ 21h
piceof equ 20h
extrn countvalue:byte,row:word,col:word,addr:word,portvalue:WORD
extrn attr:byte,char:byte,selcount:word,space:word,area:byte,portaddr:word
extrn stringdisp:near,display:near,mon_addr:word,index:word,hexad:near
public terminal
public errordisp
public getscreen
jmp terminal
coms db 0ch,11101111B
      db 0bh,11110111B
      intrf db ?
      CHAR1 DB ?
value dw ?
oldvect dd ?
buffertop dw ?
bufferend dw ?
comport dw ?
putdata db BUFSIZ dup(?)
prinport dw ?
storefile db 'serial.fil',0
tempfile db 'print.tmp',0
eofcount dw ?
getscreen dw 2000 dup(?)
tracol dw ?
trarow dw ?
rcrow dw ?
rccol dw ?
dispcol dw ?
disprow dw ?
carry db ?
error db ?
localrow dw ?
localcol dw ?
handle dw ?
filename db 12 dup(?)
zero db 0
```


Thr dw ?
Rhr dw ?
Lsr dw ?
Lcr dw ?
bdrh dw ?
bdrl dw ?
mcr dw ?
iir dw ?
ier dw ?
msr dw ?

```
terminal proc
push si
push ax
push di
push bx
push es
push ds
pop es
mov ax,row
mov localrow,ax
mov ax,col
mov localcol,ax
mov dispcol,0
mov disprow,1
@getscreen getscreen
mov ax,portaddr
cmp portaddr,0
je n_port
jmp y_port
n_port:@clrscr
mov attr,07h
@string <'No Port address is found Select different option$'
inc row
mov col,0
@string <'press any key to continue$'>
jmp endpr
y_port:mov thr,ax
mov rhr,ax
mov bdrl,ax
inc ax
mov bdrh,ax
mov ier,ax
inc ax
mov iir,ax
inc ax
mov lcr,ax
inc ax
mov mcr,ax
inc ax
mov lsr,ax
inc ax
mov msr,ax
cmp selcount,0
jne filetransmit
```

```

jmp printend
filetransmit:
cmp countvalue,4
jne noreceiver1
@clrscr
mov row,0
mov col,5
mov attr,07
@string <'THE FILE IS BEING RECEIVED$'>
call serialinit
call receiver
jmp endpr
noreceiver1:mov attr,07h
@clrscr
mov row,0
mov col,5
mov bx,offset space
mov di,offset filename
mov si,[bx]
more:lodsb
cmp al,'$'
jz over
stosb
jmp more
over:mov [di],byte ptr 0
@open filename,handle
jnc noerr
mov error,1
call errordisp
jmp endfile
noerr:cmp countvalue,1
je notest
jmp test1
notest:call serialinit
mov char1,27
call Transmit
mov char1,'F'
call Transmit
JMP PERFORM
test1:cmp countvalue,2
je notest1
jmp test3
notest1:call serialinit
mov char1,27
call transmit
mov char1,'R'
call transmit
jMP PERFORM
test3:cmp countvalue,3
jne test4
call serialinit
call sendmess
jmp endpr
test4:call loopbacktest
jmp endpr

```

```

perform:
@string <'File will be transmitted now $'>
mov ax,offset filename
mov addr,ax
call stringdisp
inc row
printok:@read char1,handle,eofcount
jnc noerr1
mov error,2
call errordisp
jmp endfile
noerr1:cmp char1,ctrz
jne proced
call transmit
jmp endfile
proced:cmp eofcount,0
jne ok
mov char1,ctrz
call transmit
jmp endfile
ok:mov ax,disprow
mov row,ax
mov ax,dispcol
mov col,ax
call displayfile
cmp col,0
jg nocol1
cmp row,24
jb nocol1
push si
push di
push es
push ds
mov si,80*2*2
mov di,1*80*2
mov es,mon_addr
mov ds,mon_addr
mov cx,80*23
rep movsw
mov ax,0720h
mov cx,80
rep stosw
mov row,23
mov col,0
pop ds
pop es
pop di
pop si
nocol1:mov ax,row
mov disprow,ax
mov ax,col
mov dispcol,ax
enddisp0:call transmit
jmp printok
endfile:@close handle

```

```

jnc endpr
mov error,3
call errordisp
endpr:mov ah,0
int 16h
@clrscr
@putscreen getscreen
printend:mov carry,0
mov index,0*2
mov ax,localrow
mov row,ax
mov ax,localcol
mov col,ax
pop es
pop bx
pop di
pop ax
pop si
ret
terminal endp

```

```

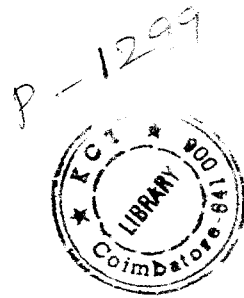
transmit proc
PUSH AX
PUSH DX
PUSH CX
PUSH BX
mov bx,5
ytest:mov cx,65535
mov dx,lsr
in al,dx
test al,20h
jnz ytransmit
ll:loop ll
dec bx
jnz ytest
@string <'Transmission error could not Transmit data $'>
jmp overs
ytransmit:mov al,char1
mov dx,thr
out dx,al
overs: pop bx
pop cx
pop dx
pop ax
ret
transmit endp

```

```

loopbacktest proc
push dx
push ax
push ds
push cs
pop ds
mov row,0
mov col,1

```



```

mov trarow,1
mov tracol,1
@string <'File is being Transmitted $'>
mov row,13
mov col,1
mov rcrow,14
mov rccol,1
@string <'File is being Received $'>
mov dx,1cr
mov al,80h
out dx,al
mov dx,bdr1
mov al,0ch
out dx,al
mov dx,bdrh
mov al,0
out dx,al
mov dx,1cr
mov al,0ah
out dx,al
mov dx,mcr
mov al,13h
out dx,al
mov dx,ier
mov al,0
out dx,al
forever:mov dx,1sr
in al,dx
test al,01h
jz noreceive
call receive
jmp forever
noreceive:test al,1eh
jz noerror
mov row,13
mov col,1
@clrscr
@string <'Loop back reception Fails Press any key to continue $'>
mov ah,0
int 16h
jmp eoffile
noerror:test al,20h
jnz for
jmp forever
for:mov bx,handle
@read char1,handle,eofcount
jnc noerror1
mov error,2
call errordisp
jmp eoffile
noerror1:cmp char1,ctrz
jne noeoffile
jmp eoffile
noeoffile:cmp eofcount,0
je eoffile

```

```

mov ax, trarow
mov row, ax
mov ax, tracol
mov col, ax
call displayfile
cmp col, 0
jg ntraend
cmp row, 12
jb ntraend
push si
push di
push es
push ds
mov si, 80*2*2
mov di, 1*80*2
mov es, mon_addr
mov ds, mon_addr
mov cx, 80*10
rep movsw
mov ax, 0720h
mov cx, 80
rep stosw
mov row, 11
mov col, 0
pop ds
pop es
pop di
pop si
ntraend: mov ax, row
mov trarow, ax
mov ax, col
mov tracol, ax
call transmit
jmp forever
eoffile:
mov cx, 100
eof12: mov dx, lsr
in al, dx
test al, 01h
jnz cllrec
loop eof12
jmp nochar
cllrec: call receive
nochar: pop ds
pop dx
pop ax
ret
loopbacktest endp

```

```

receive proc
mov dx, rhr
in al, dx
and al, 07fh
mov char1, al

```

```

push ax
push ds
push es
push di
push si
mov ax,rcrow
mov row,ax
mov ax,rccol
mov col,ax
call displayfile
cmp col,0
jg nrcend
cmp row,24
jb nrcend
mov si,160*15
mov di,14*2*80
mov es,mon_addr
mov ds,mon_addr
mov cx,80*10
rep movsw
mov ax,0720h
mov cx,80
rep stosw
mov row,23
mov col,0
nrcend:
mov ax,row
mov rcrow,ax
mov ax,col
mov rccol,ax
pop si
pop di
pop es
pop ds
pop ax
ret
receive endp

displayfile proc
cmp char1,0ah
jne nolff
jmp enddisp
nolff:cmp char1,0dh
jne nocr1
mov col,0
jmp rowinc
nocr1:mov al,char1
mov char,al
call display
inc col
cmp col,79
jbe enddisp
mov col,0
rowinc:inc row
enddisp:ret

```

displayfile endp

```
errordisp proc
push ax
mov row,10
mov col,5
cmp error,1
je y_open_err
jmp no_open_err
y_open_err:@clrscr
@string <'Error has occurred while opening file $'>
mov ax,offset filename
mov addr,ax
call stringdisp
inc row
mov col,5
@string <'The present operation cannot be completed $'>
inc row
mov col,5
@string <'press any key to continue $'>
mov ah,0
int 16h
jmp complet
no_open_err:cmp error,2
je y_read_err
jmp no_read_err
y_read_err:@clrscr
@string <'Error has occurred while reading file $'>
mov ax,offset filename
mov addr,ax
call stringdisp
inc row
mov col,5
@string <'The present operation cannot be completed$'>
inc row
mov col,5
@string <'press any key to continue $'>
mov ah,0
int 16h
jmp complet
no_read_err:cmp error,3
je y_close_err
jmp write_err
y_close_err:@clrscr
@string <'Error has occurred while closing file $'>
mov ax,offset filename
mov addr,ax
call stringdisp
inc row
mov col,5
@string <'The present operation cannot be completed$'>
inc row
mov col,5
@string <'press any key to continue $'>
```



```

mov ah,0
int 16h
test write_err:cmp error,4
je write_err
jmp complet
write_err:@string 'Error has occurred while writing $'
mov ax,offset filename
mov addr,ax
call stringdisp
inc row
mov col,5
@string 'prsent operation cannot be completed$'
inc row
mov col,5
@string 'Press any key to continue$'
mov ah,0
int 16h
complet:mov error,0
pop ax
ret
errordisp endp

```

```

serialinit proc
push dx
push ax
push ds
push cs
pop ds
mov dx,lcr
mov al,80h
out dx,al
mov dx,bdr1
mov al,0ch
out dx,al
mov dx,bdrh
mov al,00h
out dx,al
mov dx,lcr
mov al,0ah
out dx,al
mov dx,mcr
mov al,03h
out dx,al
mov dx,ier
mov al,00
out dx,al
pop ds
pop dx
pop ax
ret
serialinit endp

```

```

sendmess proc
@CLRSCR

```

```

@gotoxy 0,0
contever:mov dx,LSR
in al,dx
test al,1eh
jz proced1
mov row,13
mov col,1
@string <'Receiver error has occured Press any key to return to main men
mov ah,0
int 16h
proced1:test al,01h
jnz receive1
test al,20h
jz contever
mov ah,01
int 16h
jz contever
mov ah,0
int 16h
push ax
mov ah,14
mov bx,0
int 10h
pop ax
push ax
cmp al,13
jne nocy1
mov al,10
mov ah,14
int 10h
nocy1:cmp al,27
jne yes_transmit
pop ax
ret
yes_transmit:pop ax
mov dx,THR
out dx,al
jmp contever
receive1:mov dx,RHR
in al,dx
and al,7fh
push ax
mov bx,0
mov ah,14
int 10h
pop ax
push ax
cmp al,13
jne nocr
mov al,10
mov ah,14
int 10h
nocr:pop ax
cmp al,27
je yctrz

```

```
jmp contever
yctrz:ret
sendmess endp
```

```
RECEIVER PROC
PUSH AX
PUSH DX
push bx
call serenb
@creat storefile,handle
back1:call serget
cmp char1,27
je back1
cmp char1,'F'
je testfile
jmp printchar
testfile:
NOFILE:CALL SERGET
@WRITE CHAR1,HANDLE
CMP CHAR1,CTRZ
JnE NOFILE
JMP endreceive
PRINTCHAR:CALL SERGET
mov prinport,03bch
call print
CMP CHAR1,CTRZ
JNE PRINTCHAR
endreceive:@close handle
call serdis
pop dx
pop bx
pop ax
ret
receiver endp
```

```
SERGET proc
PUSH BX
push ax
push cs
pop ds
CLI
MOV BX,BUFFERTOP
CMP BX,BUFFEREND
JNe GETCHAR
STI
POP AX
POP BX
JMP SERGET
GETCHAR :MOV AL,[putdata+BX]
MOV CHAR1,AL
INC BX
CMP BX,BUFSIZ
JC GETCHAR1
xor bx,bx
```

```

GETCHAR1:MOV BUFFERTOP,BX
STI
mov intrf,0
pop ax
POP BX
RET
serget endp

```

```

serint proc
STI
PUSH AX
PUSH BX
PUSH DX
push ds
push cs
pop ds
mov intrf,1
CLI
mov dx,rhr
in al,dx
mov bx,bufferend
mov [putdata+bx],al
inc bx
cmp bx,bufsiz
jc serint1
xor bx,bx
serint1:mov bufferend,bx
mov al,20h
out piceof,al
pop ds
POP DX
pop bx
pop ax
iret
serint endp

```

```

print proc
push dx
push ax
push cx
push bx
mov row,10
mov col,5
mov bx,10
mov dx,prinport
mov al,char1
out dx,al
inc dx
busy:mov cx,65535
in al,dx
and al,10000000b
jz llp
jmp nbusy
llp:loop llp

```

```

dec bx
jnz busy
@clrscr
@string 'Printer is not ready please check printer
        press any key to continue $'
mov carry,1
pop bx
pop cx
pop ax
pop dx
ret
nbusy:
mov al,char1
sub dx,1
out dx,al
mov al,00001101b
add dx,2
out dx,al
mov al,00001100b
out dx,al
pop bx
pop cx
pop dx
pop ax
ret
print endp

```

```

serenb proc
push ax
push bx
push dx
push es
mov bx,portvalue
mov ax,bx
call hexad
mov al,[bx+coms]
MOV AH,[BX+COMS+1]
mov comport,ax
mov ah,35h
int 21h
mov WORD PTR [oldvect+2],es
mov WORD PTR [oldvect],bx
mov dx,offset serint
MOV AX,COMPORT
mov ah,25h
int 21h
mov dx,mcr
mov al,0bh
out dx,al
mov DX,IER
mov al,1
out dx,al
in al,picmask
MOV BX,COMPORT
mov ax,bx

```

```
call hexad
and al,Bh
out picmask,a1
pop es
pop dx
pop bx
pop ax
ret
serenb endp
```

```
serdis proc
push ax
push dx
push ds
IN AL,PICMASK
mov bx,comport
not bh
or al,bh
out picmask,a1
lds dx,oldvect
mov bx,comport
mov al,b1
mov ah,25h
int 21h
pop ds
pop dx
pop ax
ret
serdis endp
code ends
end
```

SELADP.ASM

This finds all type of adapters installed in the host system where the software is running.

```
CODE SEGMENT public 'code'
ASSUME CS:CODE
JMP FIND
printport dw ?
serialport dw ?
totalserial db ?
totalparallel db ?
coms dw ?
      dw ?
      dw ?
      dw ?
prints dw ?
      dw ?
      dw ?
COLOR EQU 0B800H
MEMO EQU 0B000H
MON_ADDR DW ?
coproc db ?
status_port dw ?
base_addr dw ?
gameport db ?
equip dw ?
mode db ?
public mode
public base_addr
public status_port
public mon_addr
public find
FIND PROC
  MOV AX, 40H
  MOV eS, AX
  mov ax,es:63h
  mov base_addr,ax
  add ax,6
  mov status_port,ax
  mov al,es:49h
  mov mode,al
  MOV AX, eS:10H
  AND AX, 30h
  CMP AX, 30h
  JE MOM
  MOV mon_addr,COLOR
  JMP OVER
MOM: MOV mon_addr,MEMO
OVER:mov ax,es:10h
      mov equip,ax
      and ah,11000000b
      xor cx,cx
```

```
mov cl,6
shr ah,cl
mov totalparallel,ah
mov ax,equip
and ah,00001110b
shr ah,1
mov totalserial,ah
mov ax,equip
mov cl,5
and ah,00100000b
shr ah,cl
mov gameport,ah
mov ax,equip
and al,00000010
shr al,1
mov coproc,al
RET
FIND ENDP
CODE ENDS
END
```


DOSCOMM.ASM

This incorporates routines for 3 DOS functions,

- (a) Type a file
- (b) Change directory
- (c) Current directory

```
if1
include mymacro.asm
endif
    code segment public 'code'
        assume cs:code
        stt: jmp doscomm
        extrn
selcount: word, display: near, stringdisp: near, countvalue: byte, errordisp: near
        extrn
row: word, col: word, space: word, addr: word, area: byte, mon_addr: word, char: byte
        extrn
getscreen: word, saverow: word, savecol: word, attr: word, index: word
        ctrz equ 01ah
        L_middle equ 'L'
        r_middle equ '9'
        LE_TO_COR EQU 'I'
        HOR EQU 'M'
        VER EQU ':'
        RI_TO_COR EQU ';'
        LE_BO_COR EQU 'H'
        RI_BO_COR EQU '<'
        count db 22
        direction db ?
        change db ?
        current_dir db 64 dup(?)
        dispcol dw ?
        disprow dw ?
        eofval dw ?
        localcol dw ?
        localrow dw ?
        handle dw ?
        filename db 12 dup(?)
        zero db 0
        dollar db '$'
        carry db ?
        error db ?
        maxrow dw ?
        maxcol dw ?
        public change
        public doscomm
        doscomm proc
            push si
            push ax
            push di
```

```

push bx
push es
push ds
pop es
mov ax,row
mov localrow,ax
mov ax,col
mov localcol,ax
mov dispcol,1
mov disprow,1
cmp countvalue,1
jne testnext
jmp typefile
testnext:cmp countvalue,2
jne moretest
jmp chan_dir
moretest:cmp countvalue,3
jne noaction
jmp curr_dir
noaction:mov direction,0
mov count,22
mov ax,localrow
mov row,ax
mov ax,localcol
mov col,ax
mov index,3*2
pop es
pop bx
pop di
pop ax
pop si
ret
doscomm endp

```

```

typefile:mov attr,07
cmp selcount,0
jne fileprint
jmp endprint
fileprint:@getscreen getscreen
@clrscr
mov row,0
mov col,5
mov bx,offset space
mov di,offset filename
mov si,[bx]
more:lodsb
cmp al,'$'
jz over
stosb
jmp more
over:
mov [di],byte ptr 0
@open filename,handle
jnc noerr
mov error,1

```

```

call errordisp
jmp endfile
noerr:@string <'File will be printed now $'>
mov ax,offset filename
mov addr,ax
call stringdisp
inc row
printok:@read char,handle,eofval
jnc noerr1
mov error,2
call errordisp
jmp endfile
noerr1:cmp char,ctrz
je endfile
noendfile:cmp eofval,0
je endfile
call displayfile
cmp carry,1
je endfile
jmp printok
endfile:@close handle
jnc endpr
mov error,3
call errordisp
endpr:mov ah,0
int 16h
@clrscr
@putscreen getscreen
endprint:mov carry,0
jmp noaction

```

```

displayfile proc
push ax
push cx
push es
push di
push ds
push si
mov ax,dispcol
mov col,ax
mov ax,disprow
mov row,ax
cmp char,0ah
jne nolf
jmp enddisp
nolf:cmp char,0dh
jne nocr
mov dispcol,0
jmp findrow
nocr:call display
inc dispcol
cmp dispcol,79
jbe enddisp
mov dispcol,0
findrow:cmp direction,0

```

```

jne reverse
inc disprow
cmp disprow,24
jb enddisp
mov ah,0
int 16h
mov direction,1
reverse:mov es,mon_addr
mov ds,mon_addr
mov si,80*2*2
mov di,80*2*1
mov cx,80*23
rep movsw
mov cx,80*2
mov ax,0720h
rep stosw
mov disprow,24
mov dispcol,0
dec count
jnz enddisp
mov ah,0
int 16h
mov count,22
enddisp: pop si
pop ds
pop di
pop es
pop cx
pop ax
ret
displayfile endp

```

```

curr_dir:mov row,13
mov col,5
mov maxrow,20
mov maxcol,40
@drawbox row,col,maxrow,maxcol
inc row
inc col
@string <'Current directory is C:\$'>
mov ah,47h
mov dl,0
mov si,offset current_dir
int 21h
mov cx,64
mov al,0
mov di,offset current_dir
repne scasd
mov [di-1],byte ptr '$'
mov si,offset current_dir
mov addr,si
call stringdisp
add row,2
mov col,10
@string <'Press any key to continue $'>

```

```

mov ah,0
int 16h
mov row,13
mov col,5
add maxcol,2
add maxrow,2
mov attr,07
@delwindow row,col,maxrow,maxcol,char
jmp noaction

```

```

chan_dir:mov row,13
mov col,5
mov maxrow,20
mov maxcol,40
@drawbox row,col,maxrow,maxcol
inc row
inc col
@string <'Enter new Directory :$' >
mov [current_dir],byte ptr 64
push es
mov ax,40h
mov es,ax
mov ax,row
mov es:51h,al
mov ax,col
mov es:50h,al
pop es
@getbuffer current_dir
mov di,offset current_dir
xor bh,bh
mov bl,[di+1]
add di,2
mov [di+bx],byte ptr 0
mov [current_dir],byte ptr 0
mov ah,03bh
mov dx,di
int 21h
jnc nocy1
inc row
sub col,10
mov attr,87h
@string <'Error occured !$'>
mov attr,07
nocy1:add row,2
mov col,10
@string <'Press any key to continue$'>
mov ah,0
int 16h
mov row,13
mov col,5
add maxrow,2
add maxcol,2
@delwindow row,col,maxrow,maxcol,char
mov change,1

```

```
jmp noaction  
code ends  
end
```

GETPARAM.ASM

This creates the menu for the following,

- (a) Baud rate
- (b) Parity
- (c) Stop bits
- (d) Character length

```
if1
include mymacro.asm
endif
code segment public 'code'
    assume cs:code
    jmp getparams
    parityaddress dw ?
    baudvalue dw ?
    COUNT DW ?
    baudrate dw ?
    stop dw ?
    charlen dw ?
    localrow dw ?
    localcol dw ?
    parity dw ?
    bdh db ?
    bdl db ?
    carry db ?
    public getparams,baudvalue,parityaddress,stop,charlen,lcrvalue

    baudvalues dw 0900h,0600h,0417h,0300h,0180h,00c0h,0060h,0040h
                dw 003ah,0030h,0020h,0018h,0010h,000ch
    BAUDRATES DW 50,75,110,150,300,600,1200,1800,2000,2400,3600
                DW 4800,7200,9600
    PARITYSTRING DB 'ODD$','EVEN$','MARK$','SPACE$','NONE$'
    PARITYVALUES DB 00001000B,00011000B,00101000B,00111000B,00000000B
    default db '9600$'

    extrn
selcount:word,display:near,stringdisp:near,countvalue:byte
    extrn
row:word,col:word,space:word,addr:word,area:byte,mon_addr:word,char:byte
    extrn
getscreen:word,saverow:word,savecol:word,attr:word,index:word
    L_middle equ 'L'
    r_middle equ '9'
    LE TO COR EQU 'I'
    HOR EQU 'M'
    VER EQU ':'
    RI TO COR EQU ';'
    LE_BO_COR EQU 'H'
    RI_BO_COR EQU '<'
```

```

lcrvalue db ?
maxrow dw ?
maxcol dw ?
disprow dw ?
dispcol dw ?
getparams proc
push si
push ax
push di
push bx
push es
mov ax,row
mov localrow,ax
mov ax,col
mov localcol,ax
mov dispcol,0
mov disprow,0
mov row,13
mov col,5
mov maxrow,20
mov maxcol,40
@drawbox row,col,maxrow,maxcol
inc row
inc col
cmp countvalue,1
jne testnext
jmp getbaud
testnext:cmp countvalue,2
jne moretest
jmp getparity
moretest:cmp countvalue,3
jne yettest
jmp getstop
yetest:cmp countvalue,4
jne noaction
jmp getchar_len
noaction:add row,2
mov col,10
@string <'Press any key to continue $'>
mov ah,0
int 16h
mov row,13
mov col,5
add maxrow,2
add maxcol,2
@delwindow row,col,maxrow,maxcol,char
mov ax,localrow
mov row,ax
mov ax,localcol
mov col,ax
mov index,2*2
pop es
pop bx
pop di
pop ax

```



```

    pop si
    ret
getparams endp

```

```

getbaud:@string <'Enter baud rate :$'>
    jmp getkey
    @buffer max1,act1,val1,5
getkey:CALL GOTOXY
    @getbuffer max1
    @atoi val1,baudrate
    mov di,offset val1
    xor bx,bx
    mov bl,[di-1]
    mov [bx+di],byte ptr '$'
    mov baudvalue,di
    mov si,offset baudrates
    mov cx,15

```

```

morebaud:lodsw
    cmp ax,baudrate
    je y_baud
    inc count
    loop morebaud
    call errordisp
    MOV BDH,00
    MOV BDL,0CH
    mov ax,offset default
    mov baudvalue,ax
    jmp erase1
Y_BAUD: mov BX,COUNT
    MOV AX,[BX+BAUDVALUES]
    mov bdh,AH
    mov bdl,AL
    jmp erase1

```

```

getparity:@string <'Enter parity :$'>
    jmp getkey1
    @buffer max2,act2,val2,06
getkey1:CALL GOTOXY
    mov count,0
    @getbuffer max2
    mov bl,act2
    xor bh,bh
    mov di,offset val2
    mov parityaddress,di
    mov [bx+di],byte ptr '$'
    mov cx,5
    MOV SI,OFFSET PARITYSTRING
more:@strcmp si,di,carry
    cmp carry,1
    je y_found
    inc count
    mov di,offset val2
    loop more
    call errordisp

```

```

MOV PARITY,00001000B
mov ax,offset paritystring
mov parityaddress,ax
jmp erase1
y_found:mov bx,count
mov al,[bx+PARITYvalues]
xor ah,ah
mov parity,ax
erase1:
jmp noaction

getstop:@string <'Enter stop bits    :$'>
        jmp getkey2
        @buffer max3,act3,val3,2
        getkey2:CALL GOTOXY
        @getbuffer max3
        @atoi val3,stop
        cmp stop,2
        ja errent1
        cmp stop,1
        jae noerr
errent1: call errordisp
        mov stop,1
        noerr: jmp erase1

getchar_len:@STRING <'Enter char length    :$'>
        jmp getkey3
        @buffer max4,act4,val4,2
        getkey3:CALL GOTOXY
        @getbuffer max4
        @atoi val4,charlen
        cmp charlen,8
        ja errent
        cmp charlen,5
        jae noeror
errent:  call errordisp
        mov char,7
noeror:  jmp erase1

GOTOXY PROC
push ax
push es
        mov ax,40h
        mov es,ax
        mov ax,row
        mov es:51h,a1
        mov ax,col
        mov es:50h,a1
        pop ax
        pop es
        RET
        GOTOXY ENDP
errordisp proc
INC ROW

```

```
mov col,10
mov attr,87h
@string <'Invalid parameter $' >
mov attr,07
ret
errordisp endp
code ends
end
```

APPLICATIONS

SERIAL/PARALLEL COMMUNICATION ADAPTER is implemented in a wide variety of computer related equipment, such as terminal, printer, mouse, optical scanner, bar code reader, voice synthesiser, OMR (Optical Mark Reader), OCR (Optical Character Reader), process control systems, etc.

This adapter is used in UNIX environment and even in DOS environment for file transferring such as printing. Sophisticated instruments like MODEM can be connected. Data transfer can be done through long distances, which is demonstrated in railway stations by the computerised reservation. The AURLEC computers introduced new tape drive which works in the parallel port mode to which parallel devices like printers can be connected.

A real time processing system is in a parallel time relationship with an ongoing activity and is producing information quickly enough to be useful in controlling this current live and dynamic activity. Thus the words 'realtime' describe a direct access or online processing system with severe time limitation. A real time processing requires immediate transaction input from all input originating terminals. Many stations are directly tied by high speed telecommunications lines into one or more CPU's. Several

stations can operate at the same time. A few examples of real time systems supported by telecommunications are in the reservations systems used by airlines, hotels etc. Military systems where the computers are used to accept , store and constantly update masses of data from world wide radar installations, and in air traffic control systems where millions of aircraft flights are tracked across the nation each year by air traffic controllers which are monitored by computers.

The ability to use telecommunication line to send electronic messages between distant points is not only limited to personal computer users , but also to sent intracompound and intercompany messages.

FUTURE DEVELOPMENTS

The serial/parallel communication adapter can be further developed by designing it to work for four port address. It is also possible to add another printer to the printer adapter. Modem connections can be done, through which telephone communications is also possible. With more modifications the telephone cable can be replaced by fibre optics and faster modes of communication can be achieved. This results in maximum speed of data transfer with a possibility of 20 giga bits/second without amplification upto 68 Km.

CONCLUSION

The **SERIAL/PARALLEL COMMUNICATION ADAPTER** has been successfully designed and fabricated.

The hardware circuitry has been neatly mounted on the PCB manufactured for this purpose. The interfacing of the card with the parallel computer has been performed.

The software has been written in the 8088 assembly language. Provision for checking the circuit by **loop back test** has also been provided. The circuit also has an additional feature for **remote printing**.

Details of the pin out diagrams of the **UART** chip and **CENTRONIX** interface has been provided along with its features. The circuit is working perfectly and is functioning for a distance of 15m.

The report also gives all the necessary data's regarding the basic communication principles and major applications of this communication card.

BIBLIOGRAPHY

TEXTBOOKS

- 1) "Microprocessor Interfacing
Programming and Hardware",
McGraw Hill Company, 1986. - Douglas V.Hall.
- 2) "IBM PC CLONES, Hardware,
Troubleshooting and Maintenance",
McGraw Hill Company, 1991. - Govindarajalu.
- 3) "Hardware Bible", BPB Publications,
1990. - Winn Rosch.
- 4) "Microcomputer Servicing,
Practical Systems and
Troubleshooting", Merrill
Publications. - Straut M. Asser
Vincent J. St
Richard
L. Barenburg.
- 5) "Microcomputer Systems,
The 8086/8088 Family",
Prentice Hall Inc, 1991. - Yu Cheng Liu
Glenn A. Gibson
- 6) "Assembly Language Techniques
for The IBM PC",
BPB Publications, 1988. - Alan R. Miller.
- 7) "Microprocessor and Microcomputer
Based System design",
Universal Book Stall, 1992. - Mohammed
Rafiquzzaman.
- 8) "IBM PC XT/AT Problem Solver". - Robert
Jourdin.

- 9) "MS-DOS Adanvanced Programming". - Ray Dunken.
- 10) "DOS 5:A Developer's Guide Advanced Programming Guide to Dos", BPB Publications1993. - Al Williams.
- 11) "Assembly Language Programming". - Abel Peter.
- 12) "The PC DATA HANDBOOK", BPB Publications,1993. - Stanley Shell.
- 13) "PROGRAMMER'S REFERENCE GUIDE - Peter Norton.

MANUALS

- 1) "Personal computer XT System,TECHNICAL REFERENCE Manual", Port Number-6936832, IBM Corporation, 1984.
- 2) "IBM-PC AT REFERENCE Manual", IBM Corporation.
- 3) "MACRO ASSEMBLER REFERENCE MANUAL", Micro soft corporation, 1981.
- 4) "MICROCOMPUTER SERVICING,Practical Systems And Trouble Shooting", Merrill Publications, 1990.



PAL16R8 Family

20-Pin TTL Programmable Array Logic

DISTINCTIVE CHARACTERISTICS

- As fast as 4.5 ns maximum propagation delay
- Popular 20-pin architectures: 16L8, 16R5, 16R6, 16R4
- Programmable replacement for high-speed TTL logic
- Register preload for testability
- Power-up reset for initialization
- Extensive third-party software and programmer support through FusionPLD partners
- 20-pin DIP and PLCC packages save space
- 28-pin PLCC-4 package provides ultra-clean high-speed signals

GENERAL DESCRIPTION

The PAL16R8 Family (PAL16L8, PAL16R8, PAL16R5, PAL16R4) includes the PAL16R8-5/4 Series which provides the highest speed in the 20-pin TTL PAL device family, making the series ideal for high-performance applications. The PAL16R8 Family is provided with standard 20-pin DIP and PLCC pinouts and a 28-pin PLCC pinout. The 28-pin PLCC pinout contains seven extra ground pins interleaved between the outputs to reduce noise and increase speed.

The family utilizes Advanced Micro Devices' advanced trench-isolated bipolar process and fuse-link technology. The devices provide user-programmable logic for replacing conventional SS/MSI gates and flip-flops at a reduced chip count.

The family allows the systems engineer to implement the design on-chip, by opening fuse links to configure AND and OR gates within the device, according to the desired logic function. Complex interconnections between gates, which previously required time-consuming layout, are lifted from the PC board and placed on silicon, where they can be easily modified during prototyping or production.

The PAL device implements the familiar Boolean logic transfer function, the sum of products. The PAL device

is a programmable AND array driving a fixed OR array. The AND array is programmed to create custom product terms, while the OR array sums selected terms at the outputs.

In addition, the PAL device provides the following options:

- Variable input/output pin ratio
- Programmable three-state outputs
- Registers with feedback

Product terms with all connections opened assume the logical HIGH state, product terms connected to both true and complement of any single input assume the logical LOW state. Registers consist of D-type flip-flops that are loaded on the LOW-to-HIGH transition of the clock. Unused input pins should be tied to Vcc or GND.

The entire PAL device family is supported by the FusionPLD partners. The PAL family is programmed on conventional PAL device programmers with appropriate personality and socket adapter modules. Once the PAL device is programmed and verified, an additional connection may be opened to prevent pattern readout. This feature secures proprietary circuits.

PRODUCT SELECTOR GUIDE

DEVICE	DEDICATED INPUTS	OUTPUTS	PRODUCT TERMS/ OUTPUT	FEEDBACK	ENABLE
PAL16L8	10	6 comb 2 comb	7 7	I/O -	prog prog
PAL16R8	8	8 reg	8	reg.	pin
PAL16R6	8	6 reg 2 comb	6 7	reg I/O	pin prog
PAL16R4	8	4 reg 4 comb	8 7	reg I/O	pin prog

FUNCTIONAL DESCRIPTION

Standard 20-pin PAL Family

The standard bipolar 20-pin PAL family devices have common electrical characteristics and programming procedures. Four different devices are available, including both registered and combinatorial devices. All parts are produced with a fuse link at each input to the AND gate array and connections may be selectively removed by applying appropriate voltages to the circuit. Utilizing an easily-implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Extra test words are pre-programmed during manufacturing to ensure extremely high field programming yields, and provide extra test paths to achieve excellent parametric correlation.

Pinouts

The PAL16R8 Family is available in the standard 20-pin DIP and PLCC pinouts and the PAL16R8-4 Series is available in the new 28-pin PLCC pinout. The 28-pin PLCC pinout gives the designer the cleanest possible signal with only 4.5 ns delay.

The PAL16R8-4 pinout has been designed to minimize the noise that can be generated by high-speed signals. Because of its inherently shorter leads, the PLCC package is the best package for use in high-speed designs. The short leads and multiple ground signals reduce the effective lead inductance, minimizing ground bounce. Placing the ground pins between the outputs optimizes the ground bounce protection, and also isolates the outputs from each other, eliminating cross-talk. This pinout can reduce the effective propagation delay by as much as 20% from a standard DIP pinout. Design files for PAL16R8-4 Series devices are written as if the device had a standard 20-pin DIP pinout for most design software packages.

Variable Input/Output Pin Ratio

The registered devices have eight dedicated input lines, and each combinatorial output is an I/O pin. The PAL16L8 has ten dedicated input lines and six of the eight combinatorial outputs are I/O pins. Buffers for device inputs have complementary outputs to provide user programmable input signal polarity. Unused input pins should be tied to Vcc or GND.

Programmable Three-State Outputs

Each output has a three-state output buffer with three-state control. On combinatorial outputs, a product term controls the buffer, allowing enable and disable to be a function of any product of device inputs or output feedback. The combinatorial output provides a bidirectional I/O pin and may be configured as a dedicated input if the output buffer is always disabled. On registered outputs, an input pin controls the enabling of the three-state outputs.

Registers with Feedback

Registered outputs are provided for data storage and synchronization. Registers are composed of D-type flip-flops that are loaded on the LOW-to-HIGH transition of the clock input.

Register Preload

The register on the PAL16R8 Family can be preloaded from the output pins to facilitate functional testing of complex state machine designs. This feature allows direct loading of arbitrary states, making it unnecessary to cycle through long test vector sequences to reach a desired state. In addition, transitions from illegal states can be verified by loading illegal states and observing proper recovery.

Power-Up Reset

All flip-flops power-up to a logic LOW for predictable system initialization. Outputs of the PAL16R8 Family will be HIGH due to the active low outputs. The Vcc rise must be monotonic and the reset delay time is 1000 ns maximum.

Security Fuse

After programming and verification, a PAL16R8 Family design can be secured by programming the security fuse. Once programmed, this fuse defeats readback of the internal programmed pattern by a device programmer, securing proprietary designs from competitors. When the security fuse is programmed, the array will read as if every fuse is programmed.

Quality and Testability

The PAL16R8 Family offers a very high level of built-in quality. Extra programmable fuses provide a means of verifying performance of all AC and DC parameters. In addition, this verifies complete programmability and functionality of the device to provide the highest programming yields and post-programming functional yields in the industry.

Technology

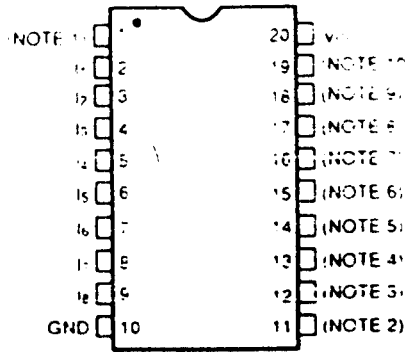
The PAL16R8 Family is fabricated with AMD's advanced trench-isolated bipolar process. This process reduces parasitic capacitances and minimum geometries to provide higher performance. The array connections are formed with proven TiW fuses for reliable operation.



CONNECTION DIAGRAMS

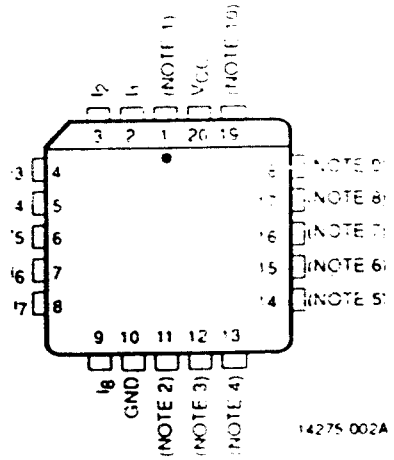
Top View

DIP



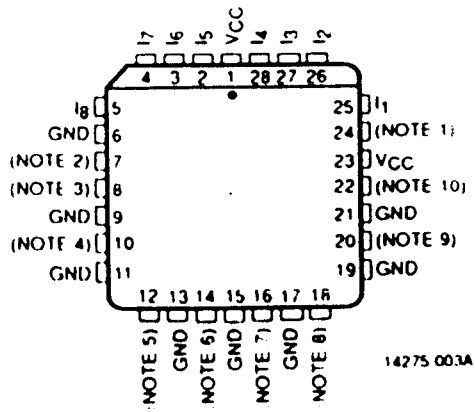
14275-001A

20-Pin PLCC



14275-002A

28-Pin PLCC



14275-003A

PIN DESIGNATIONS

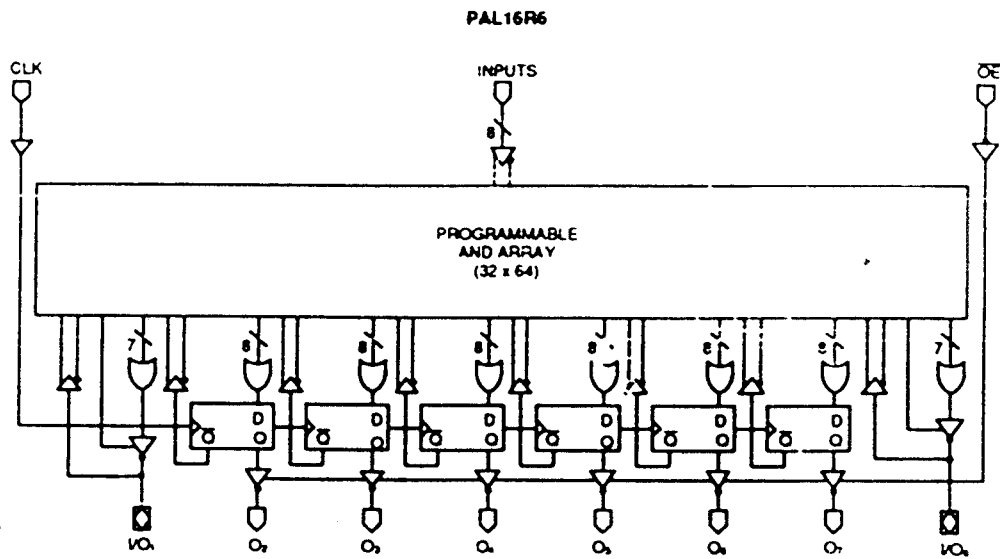
CLK	Clock
GND	Ground
I	Input
I/O	Input/Output
O	Output
\overline{OE}	Output Enable
Vcc	Supply Voltage

Note:

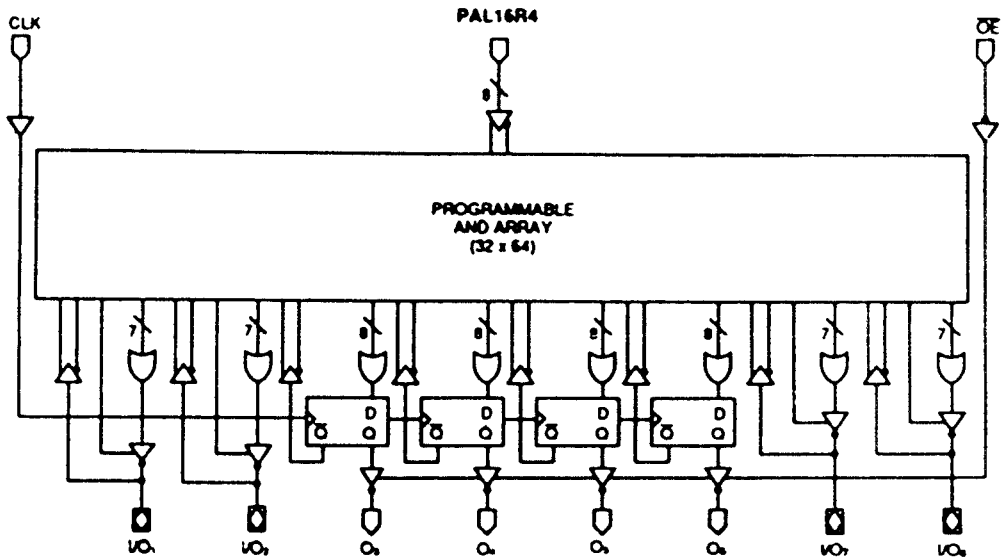
Pin 1 is marked for orientation

Note	16L8	16R8	16R6	16R4
1	I ₀	CLK	CLK	CLK
2	I ₀	\overline{OE}	\overline{OE}	\overline{OE}
3	O ₁	O ₁	I/O ₁	I/O ₁
4	I/O ₂	O ₂	O ₂	I/O ₂
5	I/O ₃	O ₃	O ₃	O ₃
6	I/O ₄	O ₄	O ₄	O ₄
7	I/O ₅	O ₅	O ₅	O ₅
8	I/O ₆	O ₆	O ₆	O ₆
9	I/O ₇	O ₇	O ₇	I/O ₇
10	O ₈	O ₈	I/O ₈	I/O ₈

BLOCK DIAGRAMS



12468-002A

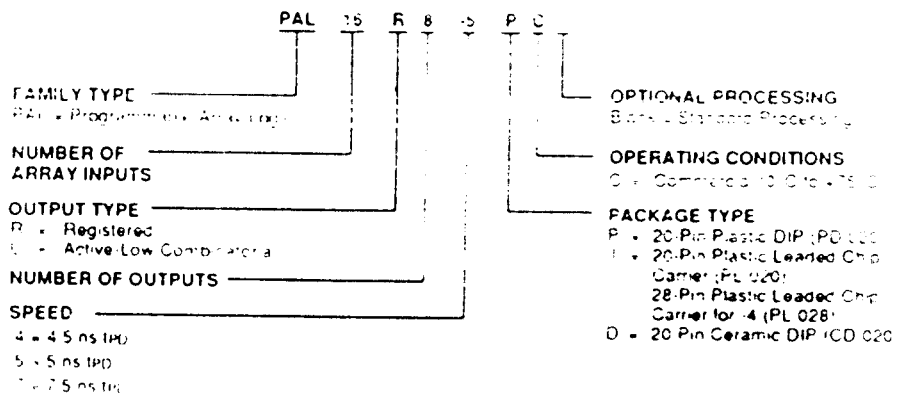


12468-003A

ORDERING INFORMATION

Commercial Products

AMD's programmable logic products for commercial applications are available with several ordering options. The order number for a combination is formed by a combination of:



Valid Combinations	
PAL16L8	5PC, 5JC, 4JC
PAL16R8	
PAL16R6	
PAL16R4	
PAL16L8 7	PC, JC, DC
PAL16R8 7	
PAL16R6 7	
PAL16R4 7	

Valid Combinations

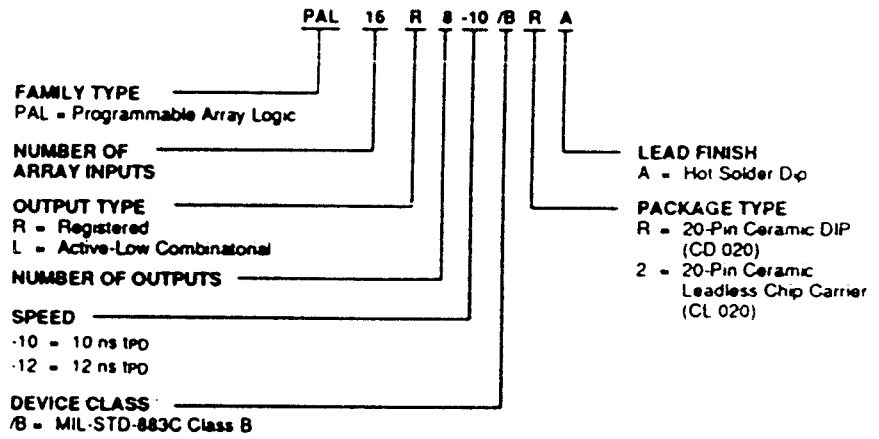
The Valid Combinations table lists configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, and to check on newly released combinations.

Note: Marked with AMD logo

ORDERING INFORMATION

APL Products

AMD programmable logic products for Aerospace and Defense applications are available with several ordering options. APL (Approved Products List) products are fully compliant with MIL-STD-883 requirements. The order number (Valid Combination) is formed by a combination of



Valid Combinations		
PAL16L8		
PAL16R8	-10, -12	/BRA, /B2A
PAL16R6		
PAL16R4		

Valid Combinations

The Valid Combinations table lists configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

Note: Marked with AMD logo

Group A Tests

Group A Tests consist of Subgroups: 1, 2, 3, 7, 8, 9, 10, 11

Military Burn-in

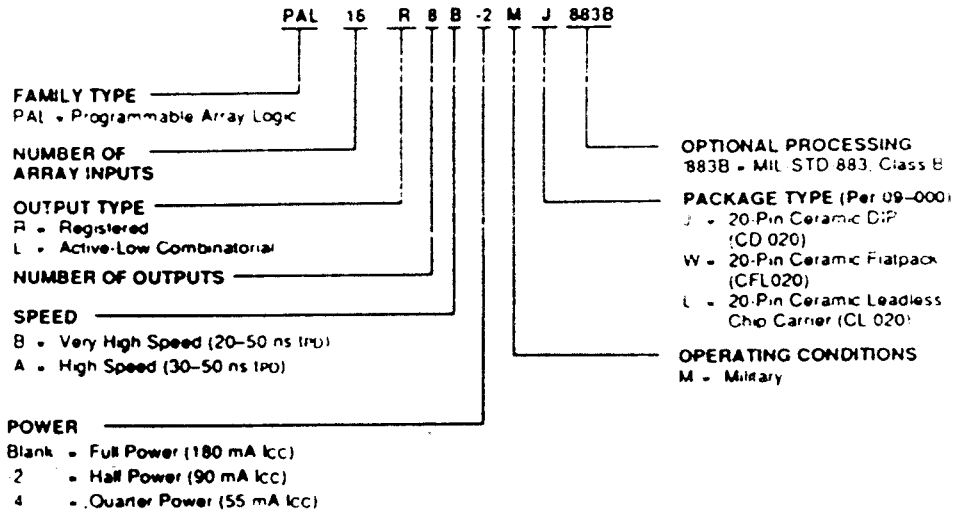
Military burn-in is in accordance with the current revision of MIL-STD-883, Test Methods 1015, Conditions A through E. Test conditions are selected at AMD's option.



ORDERING INFORMATION

APL Products (MMI Marking Only)

AMD programmable logic products for Aerospace and Defense applications are available with several ordering options. APL (Approved Products List) products are fully compliant with MIL-STD-883 requirements. The order number (Valid Combination) is formed by a combination of:



Valid Combinations		
PAL16L8	B, B-2	MJ/883B
PAL16R8	A, B-4	MW/883B
PAL16R6		ML/883B
PAL16R4		

Valid Combinations

The Valid Combinations table lists configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations and to obtain additional information on AMD's Standard Military grade products.

Note: Marked with MMI logo

Group A Tests

Group A Tests consist of Subgroups 1, 2, 3, 7, 8, 9, 10, 11

Military Burn-In

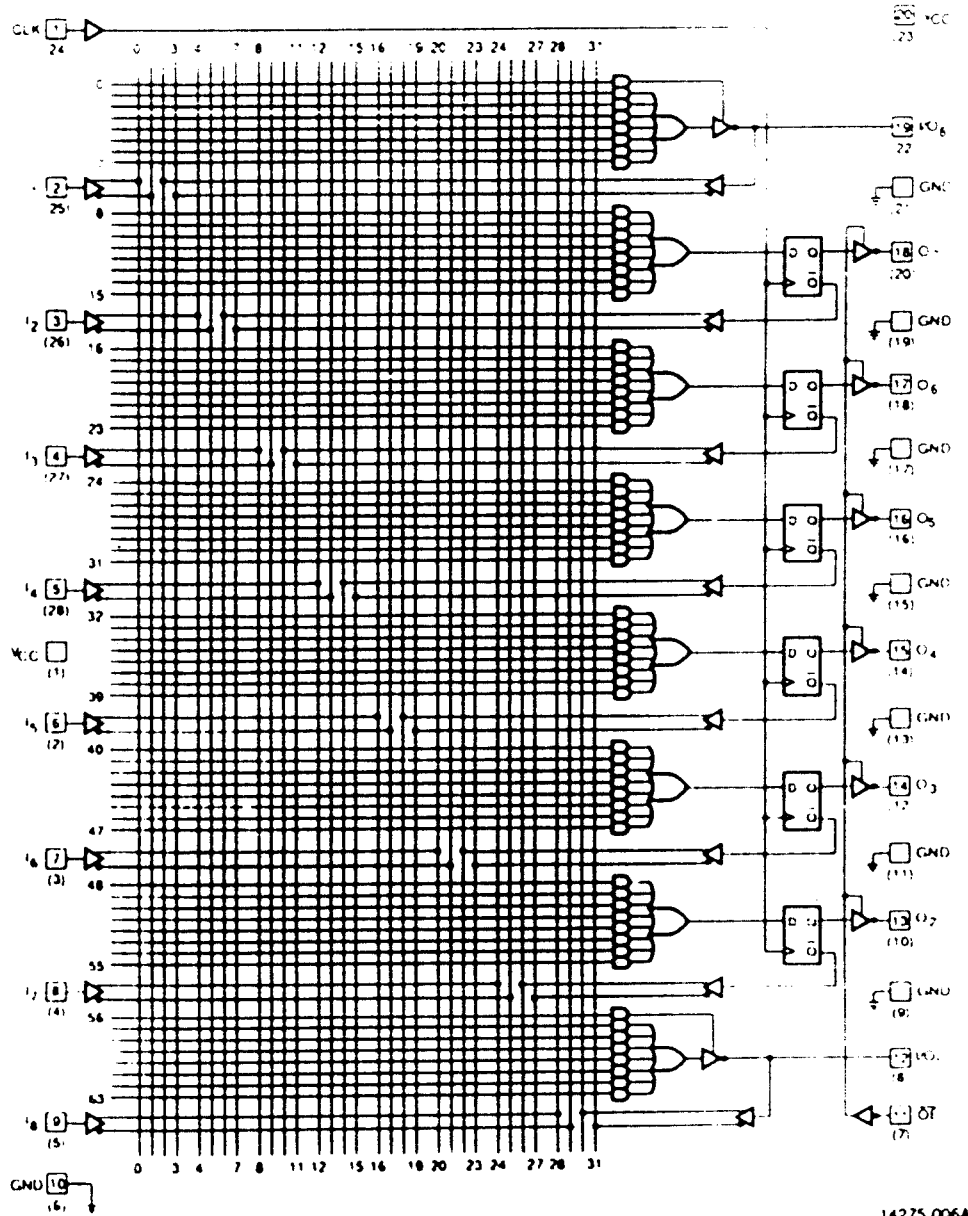
Military burn-in is in accordance with the current revision of MIL STD 883, Test Methods 1015, Conditions A through E. Test conditions are selected at AMD's option.



LOGIC DIAGRAM

DIP and 20-Pin PLCC (28-Pin PLCC) Pinouts

16R6 (-4)



14275 006A

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature with Power Applied	-65°C to +150°C
Storage Temperature	-55°C to +125°C
Supply Voltage with Respect to Ground	-0.5 V to +7.0 V
DC Input Voltage	-1.2 V to $V_{CC} + 0.5$ V
DC Input Current	-30 mA to +5 mA
DC Output or I/O Pin Voltage	-0.5 V to $V_{CC} + 0.5$ V
Static Discharge Voltage	2001 V

OPERATING RANGES

Ambient Temperature (T_A) Operating in Free Air	0°C to +75°C
Supply Voltage (V_{CC}) with Respect to Ground	+4.75 V to +5.25 V

Operating ranges define those limits between which the functionality of the device is guaranteed

Stresses above those listed under Absolute Maximum Ratings may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to Absolute Maximum Ratings for extended periods may affect device reliability. Programming conditions may differ.

DC CHARACTERISTICS over COMMERCIAL operating ranges unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Unit
V_{OH}	Output HIGH Voltage	$I_{OH} = -3.2$ mA $V_{IN} = V_{IH}$ or V_{IL} $V_{CC} = \text{Min}$	2.4		V
V_{OL}	Output LOW Voltage	$I_{OL} = 24$ mA $V_{IN} = V_{IH}$ or V_{IL} $V_{CC} = \text{Min}$		0.5	V
V_{IH}	Input HIGH Voltage	Guaranteed Input Logical HIGH Voltage for all inputs (Note 1)	2.0		V
V_{IL}	Input LOW Voltage	Guaranteed Input Logical LOW Voltage for all inputs (Note 1)		0.8	V
V_I	Input Clamp Voltage	$I_{IN} = -18$ mA, $V_{CC} = \text{Min}$.		-1.2	V
I_{IH}	Input HIGH Current	$V_{IN} = 2.7$ V, $V_{CC} = \text{Max}$. (Note 2)		25	μ A
I_{IL}	Input LOW Current	$V_{IN} = 0.4$ V, $V_{CC} = \text{Max}$. (Note 2)		-250	μ A
I_I	Maximum Input Current	$V_{IN} = 5.5$ V, $V_{CC} = \text{Max}$.		1	mA
I_{OZH}	Off-State Output Leakage Current HIGH	$V_{OUT} = 2.7$ V, $V_{CC} = \text{Max}$ $V_{IN} = V_{IH}$ or V_{IL} (Note 2)		100	μ A
I_{OZL}	Off-State Output Leakage Current LOW	$V_{OUT} = 0.4$ V, $V_{CC} = \text{Max}$ $V_{IN} = V_{IH}$ or V_{IL} (Note 2)		-100	μ A
I_{SC}	Output Short-Circuit Current	$V_{OUT} = 0.5$ V, $V_{CC} = \text{Max}$. (Note 3)	-30	-130	mA
I_{CC}	Supply Current	$V_{IN} = 0$ V, Outputs Open ($I_{OUT} = 0$ mA) $V_{CC} = \text{Max}$.		210	mA

Notes:

1. These are absolute values with respect to device ground and all overshoots due to system and/or tester noise are included.
2. I/O pin leakage is the worst case of I_{IL} and I_{OZL} (or I_{IH} and I_{OZH}).
3. Not more than one output should be tested at a time. Duration of the short-circuit should not exceed one second. $V_{OUT} = 0.5$ V has been chosen to avoid test problems caused by tester ground degradation.

CAPACITANCE (Note 1)

Parameter Symbol	Parameter Description	Test Conditions	Typ.	Unit
C _{IN}	Input Capacitance	CLK, \overline{OE}	8	pF
		I ₁ -I ₀		
C _{OUT}	Output Capacitance	V _{IN} = 2.0 V	5	
		V _{OUT} = 2.0 V		

Note:

1. These parameters are not 100% tested, but are evaluated at initial characterization, and at any time the design is modified where capacitance may be affected.

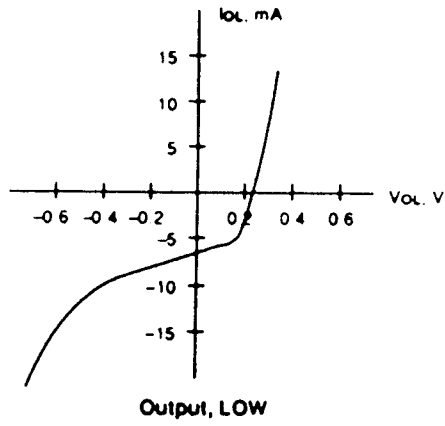
SWITCHING CHARACTERISTICS over COMMERCIAL operating ranges (Note 2)

Parameter Symbol	Parameter Description		-5		-4		Unit
			Min. (Note 3)	Max.	Min. (Note 3)	Max.	
t _{PD}	Input or Feedback to Combinatorial Output		1	5	1	4.5	ns
t _S	Setup Time from Input or Feedback to Clock		4.5		4.5		ns
t _H	Hold Time		0		0		ns
t _{CO}	Clock to Output		1	4.0	1	3.5	ns
t _{SKT WRT}	Skew Between Registered Outputs (Note 4)			1		0.5	ns
t _W	Clock Width	LOW	4		4		ns
t _{WH}		HIGH	4		4		ns
f _{MAX}	Maximum Frequency (Note 5)	External Feedback	1/(t _S + t _{CO})		125		MHz
		Internal Feedback (ICNT)		125	125		MHz
		No Feedback	1/(t _{WH} + t _{WL})	125		125	
t _{PZ1}	\overline{OE} to Output Enable		1	6.5	1	6.5	ns
t _{PZ2}	\overline{OE} to Output Disable		1	5	1	5	ns
t _{EA}	Input to Output Enable Using Product Term Control		2	6.5	2	6.5	ns
t _{ED}	Input to Output Disable Using Product Term Control		2	5	2	5	ns

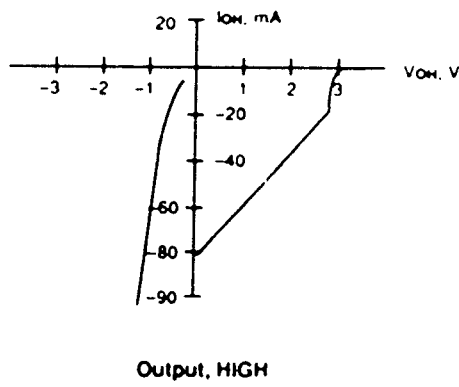
Notes:

2. See Switching Test Circuit for test conditions.
3. Delay minimums for t_{PD}, t_{CO}, t_{PZ1}, t_{PZ2}, t_{EA}, and t_{ED} are chosen based on two considerations: they must allow for the large number of variables that define "best case" conditions, and they must attempt to anticipate possible future process enhancements that may increase performance. It is possible that such process improvements may someday push the minimum delays beyond what was originally anticipated, therefore minimums should be used with care, and are recommended primarily for simulation.
4. Skew testing takes into account pattern and switching direction differences between outputs.
5. These parameters are not 100% tested, but are calculated at initial characterization and at any time the design is modified where the frequency may be affected.

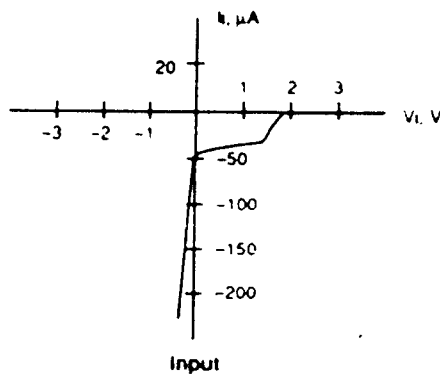
CURRENT VS. VOLTAGE (I-V) CHARACTERISTICS for the PAL16R8-4/5
 $V_{CC} = 5.0 \text{ V}$, $T_A = 25^\circ\text{C}$



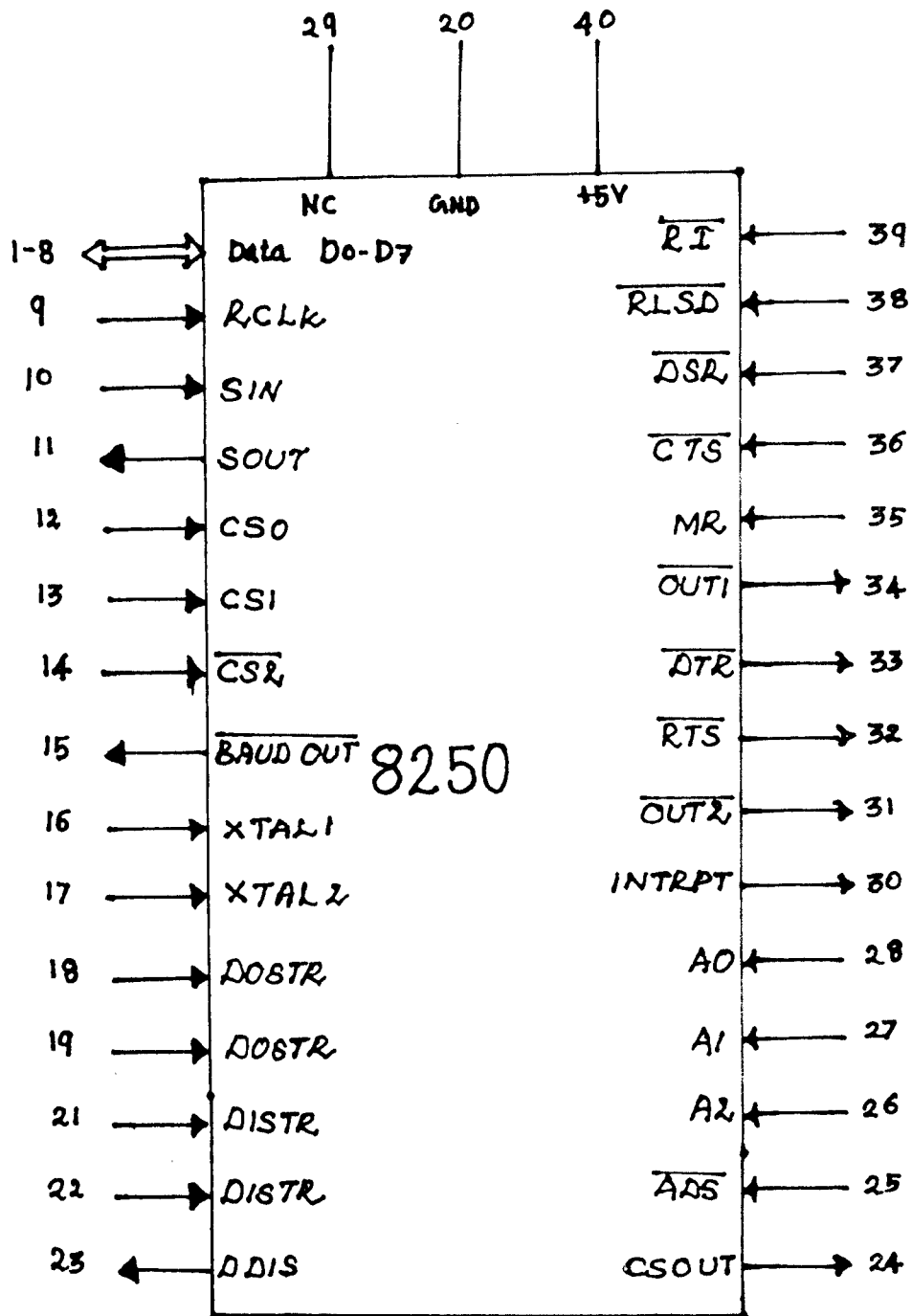
10240-0038



10240-0046



10240-005A



8250 Pinout Diagram