700

Inter PC Communication

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements for
the award of the Degree of
MASTER OF TECHNOLOGY

V. KISHAN

School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi
January 1988

Inter PC Communication

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements for
the award of the Degree of
MASTER OF TECHNOLOGY

V. KISHAN

School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi
January 1988

CERTIFICATE

This work, embodied in the dissertation titled,

INTER PC COMMUNICATION

has been carried out by Mr.V.Kishan, bonafide student of school of computer and systems sciences, Jawaharlal Nehru University, New Delhi.

This work is original and has not been submitted for any degree or diploma in any other university or institute.

Dr.S.Balasundaram
Asst.Professor
School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi

Prof Karmeshu
Dean, School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi

S Y N O P S I S

If an institute or organization has more than one computer system,it is very much essential that these computers to be interconnected,so that they can exchange information.My aim in this project is to -

Interconnect two PCs with RS232-C interface and then provide facilities for file transfer between the PCs and other utilities like mail and phone. The file transfer between the pcs is carried out in the background by implementing multitasking.Resource sharing is incorporated wherein a printer connected to one of the pcs can be accessed by the other PC as well.

The PC to PC connection can be improved with some more facilities and some more PCs can be connected to the existing two node network.

ACKNOWLEDGEMENTS

My sincere thanks are due to Dr A.K.Dua, Systems Manager,CMC Ltd,New Delhi who initiated me to this innovative project.

I am very much indebted to my guide, Dr.S.Balasundaram, Asst.Professor, who has been extreemly helpful and encouraging throughout the project,without which it would have been very difficult to complete the project.

Mr.Sanjiv Aggarwal, Systems Engineer, CMC Ltd,New Delhi played a very significant role by giving me timely and usefull suggestions and sparing his valuable time for discussions with me.

Mr.Katpalia of DCM data products gave very useful suggestions regarding the implementation of multitasking. I am thankful to him.

I express my heartfelt gratitude to Prof.K.K.Nambiar, former dean of our school,for providing the required facilities and for his unfailing interest he has shown in this project,without which it would not have materialized.

I am thankful to our dean Prof.Karmeshu,who has shown special interest in my work.

CONTENTS

1. Introduction	1
2. IBM PC and RS232-C Architecture	3
2.1 8088 Architecture	3
2.2 Interrupts and interrupt service routins	8
2.3 Serial asynchronous communication	13
2.4 RS-232C serial data transfers	17
3. PC to PC communication	18
3.1 Introduction	18
3.2 UART intialization	20
3.3 Implementation	22
3.3.1 The RESPC module	23
3.3.2 The SYSINT module	29
3.3.3 The TIME_INT module	31
3.3.4 The PCNET module	33
3.3.5 The FACILITY module	37
3.3.6 The PHONE facility	48
3.3.7 The MAIL facility	48
3.3.8 The file transfer utility	51
3.3.9 Resource sharing with DOS utility	51
4. Instructions for use	54
5. Future extensions and modifications	55
Appendix A 8088 instruction set	
Appendix B Programming 8250 UART	
Program Listings	

INTRODUCTION

As computers have become smaller,cheaper and more numorous,people have become more and more interested in connecting them together to form networks and distributed systems.Advanced computer and communication technology has been the key to survival of a many institutions and organizations.The exciting tools and techniques of this high technology are used in high technology base,for arriving at general solutions and for applications support.These approaches to the implementation of computer metworks are revolutionizing communications, business systems and manufacturing and technology. When different computers can communicate with each other and are interconnected into a network,we have many advantages like -

- greater reliability
- sharing common resourses
- better support facilities
- faster response time
- internetworking capabilities
- flexibility in application programs and so on.

Most of the terminals that connect office desks to mainframes are dumb.In contrast,the personal computer is fast developing into an intelligent user-programmable terminal.It is a monotask but multi processor, low cost,high capacity device.There is a significant trend toward multifunction work station as opposed to single function terminals.Interconnecting personal computers into a local area network and networking these with a main frame system offers many advantages.

MOTIVATION FOR THIS PROJECT :

We,at JNU have very good computing facilities.The systems include a VAX 11/780,HP1000 and six DCM TANDY1000 pcs.So far these computers are isolated and there is no way a user working on one system can look into his files on the other machine.My basic aim is to provide this facility.Since networking all these computers in not a task that can be completed with in a semester of six months,i started with a subset of it.

I want to connect all the pcs into a local area network.As a first step towards this,i wanted to interconnect two pcs through RS-232C,so that thay can exchange information.This can be extended to interconnect all the pcs into a token ring network.By adding some more software,the pcs can also be connected to the VAX. Collision detection has to implemanted when more than two pcs are interconnected.Ideally the master pc should be a PC/XT or a PC/AT.

II.IBM PC AND RS232-C ARCHITECTURE

The brain of the personal computer is the 8088 microprocessor.This chapter gives an introduction to the architecture and programming aspects of the INTEL 8088 microprocessor and it's communication aspects.

2.1 8088 Architecture

Fig 2.1 shows the internal architecture of 8088 microprocessor. The control unit and working registers are divided into three groups according to their functions. They are -

i. The data group ,which is essentially the set of arithmetic registers,

ii.The pointer group ,which includes base and index registers, but also contains the program counter and stack pointer,

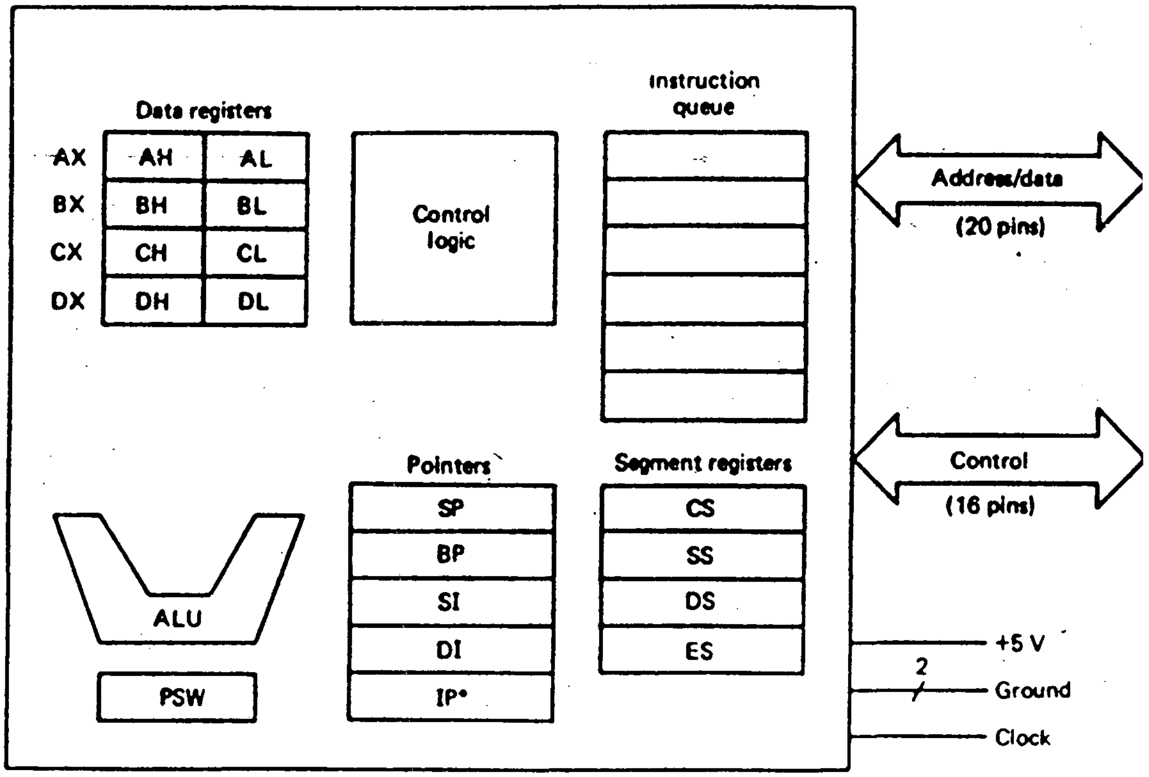
iii.The segment group which is a set of special purpose base registers.

All the registers are 16 bit wide.

The data group consists of AX,BX,CX and DX registers. These registers can be used to store both operands and results and each of them can be accessed as a whole,or lower and upper bytes can be accessed separately.

In addition to serving as arithmetic registers, the BX,CX and DX registers play special addressing,counting and I/O roles.

BX may be used as a base register in address



*For the 8086 the program counter is called the instruction pointer (IP).

FIG 2.1 8088 BLOCK DIAGRAM

calculations.

CX is used as an implied counter by certain instructions.

DX is used to hold the I/O address during certain I/O operations.

The pointer and index group consists of the IP,SP,BP,SI and DI registers.The instruction pointer (IP) and SP registers are essentially the program counter and stack pointer registers,but the complete instruction and stack addresses are formed by adding the contents of these registers to the four bit left shifted contents of the code segment(CS) and stack segment(SS) registers. BP is a base register for accessing the stack and may be used with other registers and/or a displacement,that is a part of instruction.The SI and DI registers are for indexing. Although, they may be used by themselves,they are often used with the BX or BP registers and/or a displacement. Except for the IP,a pointer can be used to hold an operand,but must be accessed as a whole.

To provide flexible base addressing and indexing, a data address may be formed by adding together a combination of the BX or BP register contents, SI or DI register contents and a displacement. The result of such computation is called an effective address(EA) or offset. The final data address,however is determined by adding the EA to the four bit left shifted contents of the appropriate data segment,extra segment or stack segment registers. This enables the proceesor to generate a 20 bit address .

The segment group consists of the CS,SS,DS and ES registers. The utilization of the segment registers essentially devides the memory space into overlapping segments,with each segment being 64k bytes long and beginning at a 16 byte paragraph boundary , i.e beginning at an address that is divisible by 16. So the contents of the segment register is the segment address and the segment address multiplied by 16 is the beginning physical segment address.

The advantages of using segment registers are to

1. Allow the memory capacity to be one magabyte even though the addresses associated with the individual instructions are only 16 bits wide.

2. Allow the instruction,data or the stack portion of a program to be more than 64k bytes long by using more than one code,data or stack segment.

3. Facilitate the use of separate memory areas for a program, it's data and the stack.

4. Permit a program and/or it's data to be put into different areas of memory each time the program is executed.

FLAGS : The 8088's Program status word(PSW) contains 16 bits,but seven of them are not used.Each bit in the PSW is called a flag.The flags are divided into the conditional flags, which reflect the result of the previous operation involving the ALU, and control flags which control the execution of special functions.

The flags are summarized below.The lower byte in the PSW corresponds to the eight bit PSW in the 8085 and contains all of the condition flags,except the overflow flag(OF).

The condition flags are -

SF (sign flag) is set if the result is negative,reset if positive.

ZF (zero flag) is set if the result is zero and reset if the result is nonzero.

PF (parity flag) is set if the lower order eight bits of the result contain an even number of ones,otherwise it is cleared.

CF (carry flag) - an addition or subtraction causes this flag to be set if a carry in MSB or a borrow is needed.

AF (auxiliary carry flag) is set if there is a carry out of bit 3 during an addition or a borrow by bit 3 during a subtraction.This is used exclusively for BCD arithmetic.

OF (overflow flag) is set if an overflow occurs.

					DF	IF	TF	SF	ZF		AF		PF		CF

DF (direction flag) - used by string manipulation insructions. If clear,the string is processed from it's beginning with the first element having the lowest address.Otherwise the string is processed from the high address towards the low address.

IF (interrupt enable flag) - If set,a certain type of interrupt (a maskable interrupt) can be recognized by the CPU,otherwise these interrupts are ignored.

TF (trap flag) if set, a trap is executed after the current instruction.

The 8088 provides various addressing modes,for details see Microcomputer Systems: The 8086/8088 family by YU-CHENG LIU and GLENN A.GIBSON.See appendix A for the instruction set of 8088.

2.2. Interrupts and interrupt service routines

It is sometimes necessary to have a computer automatically execute one of a collection of special routines,whenever certain conditions exist within a program or the computer system.The action that prompts the execution of one of these routines is called an interrupt and the routine that is executed is called an interrupt service routine. There are two general classes of interrupts and associated routines. They are the interanl interrupts that are initiated by the state of the CPU or by an instruction and the external interrupts that are caused by a signal being sent to the CPU from elsewhere in the computer system.Typical internal interrupts are those caused by division by a zero or a special instruction like INT and typical external interrupts are caused by the need of an I/O device to be served by the CPU.

In general interrupts can be recognized in two ways.

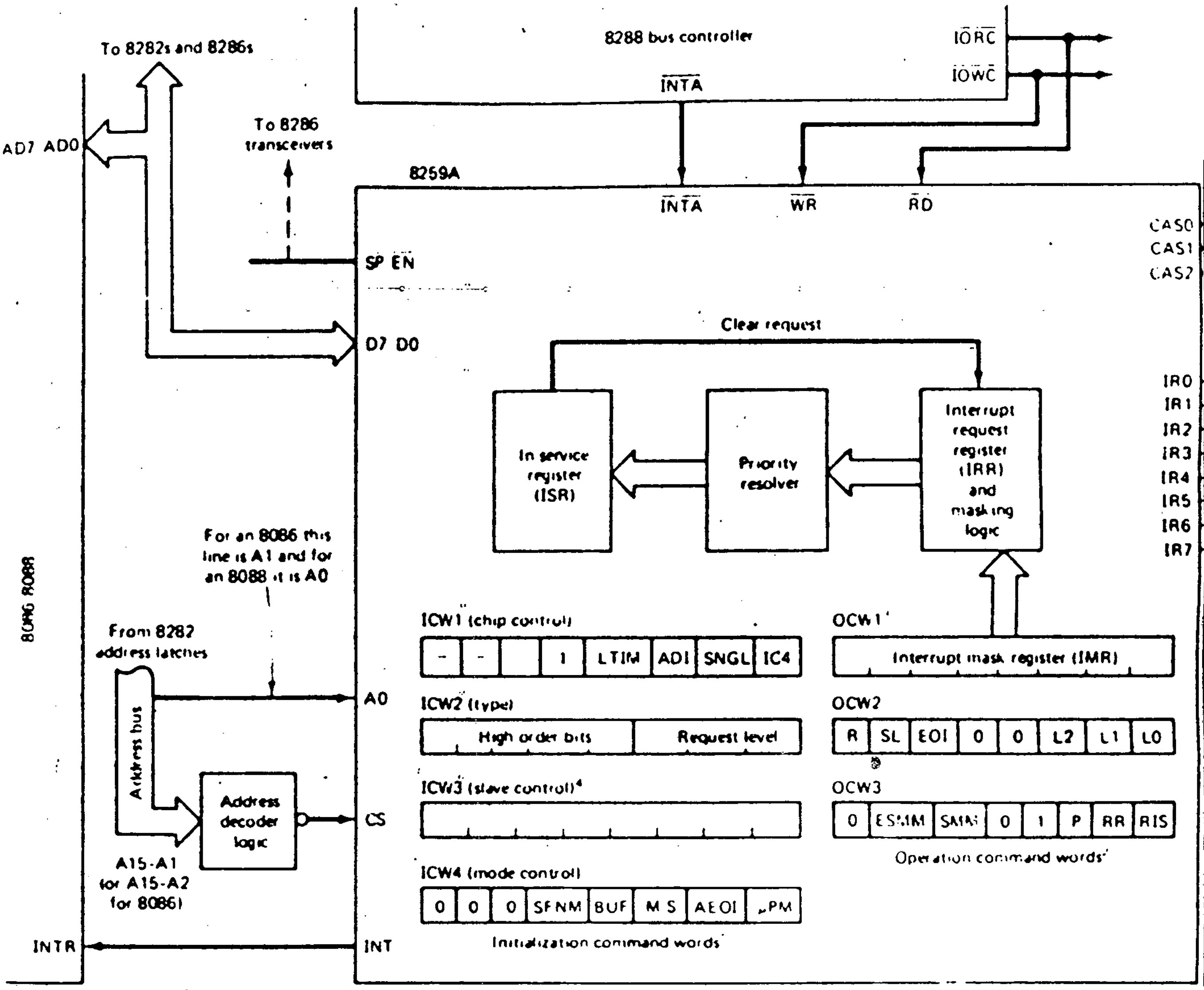
a. By polling and b.Interrupt basis. In polling, the CPU regularly checks the I/O ports for any pending interrupts. The disadvantage with polling is that the CPU time will be wasted, since the CPU has to regularly check the I/O devices.Not only that,data can be lost at the I/O port if there is considerable delay in successive pollings. In the other mode, i.e.,interrupt basis the CPU recognizes the interrupt only when the I/O device sends an interrupt.

An interrupt service routine is similar to a procedure,in that it may be branched to ,from any other program and return branch is made to that program after the interrupt service routine is executed.The interrupt service routine must be so written that,except for the lapse in time,the interrupted program will proceed just as if nothing had happened.This means that the PSW and the registers used by the routine must be saved and restored and the return must be made to the instruction following the last instruction executed before the interrupt. An interrupt service routine is unlike a procedure in that, instead of being linked to a particular program,it is sometimes put in a fixed place in memory.Because it is not linked to other segments,it can use only common areas that are absolutely located to communicate with other programs.Because some kinds of interrupts are initiated by external events,they occour at random points in the interrupted program.For such interrupts no parameter addresses can be passed to the interrupt routine.Instead, data communication can be made through variables that are directly accessible by both routines.

Regardless of the type of the interrupt,the action that results from an interrupt are the same and are known as the interrupt sequense. Some kind of interrupts are controlled by the IF and TF flags and in those cases,these flags must be properly set or else the interrupt action is blocked. If the conditions for an interrupt are met and the necessary flags are set,the instruction that is currently executing is completed and the interrupt sequence proceeds by pushing the current contents of the PSW,CS and IP on to the stack,inputting the new contents of IP and CS from a double word whose address is determined by the type of interrupt and clearing the IF and TF flags.The new contents of the IP and CS determine the beginning address of the interrupt service routine to be executed.After the interrupt has been executed,the return is made to the interrupted program by an instruction called IRET which pops the IP,CS and PSW from the stack.

The double word containing the new contents of IP and CS is called the interrupt pointer.Each interrupt type will be given a number between 0 and 255 inclusive and the address of the interrupt pointer is found by multiplying the type by 4. These addresses are loaded by the operating system when the system is booted.

I/O operations that take place between I/O devices and CPU on an interrupt basis are called interrupt I/O.Since there is only one interrupt input to an 8088,in order to support more than one device, programmable interrupt priority management circuit (8259) is connected to INTR and INTA pins of 8088.See Fig 2.2 for a block diagram



[1] A0 = 0 for addressing the first word (ICW1) and 1 for addressing the succeeding words.

[2] A0 = 1 for addressing the first word and 0 for addressing the succeeding words.

[3] Bits correspond to IR inputs. Bit = 1 means IR is masked and Bit = 0 means it is not masked.

[4] If 8259A is a master, Bit = 1 indicates that the corresponding IR input is connected to a slave. For a slave, bits 3-7 are 0 and bits 0-2 identify the slave.

FIG 2.2.

8259 INTERRUPT PRIORITY MANAGEMENT BLOCK DIAGRAM of 8259 interrupt controller.I/O devices are connected to the different levels of priority management circuit.Each level is assigned a unique interrupt vector. When an interrupt comes from a device on a particular level,priority management circuit checks for the priority.If any higher priority interrupt is in progress ,it keeps it in pending,otherwise it interrupts the CPU on behalf of the I/O device and sends the interrupt vector number which enables the CPU to respond to the interrupt.

The interrupt priority management circuit contains the logic needed to assign priorities to the incoming requests.For example, the highest priority could be given to IR0, the next priority to IR1 and so on. When an interrupt request is recognized by the priority logic as having the highest priority,then the three least significant bits of the type register are set to the number of the request line, a bit is set in the inservice register and an interrupt is sent to the CPU. If IF flag is set then the CPU returns an acknowledgement signal and the management circuit sends the CPU the type. All the requests having lower priority are blocked untill the bit in the inservice register is cleared,an action which is normally done by the routine.Therefore when IF is reenabled by an STI instruction,higher priority requests may interrupt the currently executing routine, but the lower priority requests will be blocked by the priority logic until the bit that was set in the in service register is cleared. This allows the lower priority interrupts to proceed. The priority management circuit is programmable.

For details of programming the 8259 refer INTEL manual.

In addition to the built in priority,a one byte mask register is provided to allow the masking of individual requests.Bit n in this register is for masking IRn.

2.3 Serial asynchronous communication

For two computers to exchange information,there should be proper interface between them.This is provided through a communication link,which facilitates the data transfer.

Within the computer,data is transferred in parallel,because that is the fastest way to do it. For transferring data over long distances ,however parallel data transfer requires too many wires,which is not feasible when the computers are located far apart.Therefore data to be sent to long distances is usually converted from parallel form to serial form,so 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 bus.

Serial data can be sent synchronously or asynchronously. For synchronous transmission,data is sent in blocks at a constant rate. The start and end of block are identified with specific bytes or bit patterns.For asynchronous transmission,each data character has a bit which identifies it's start and one or two bits which identifies it's end. Since each character is individually identified,characters can be sent at any time.

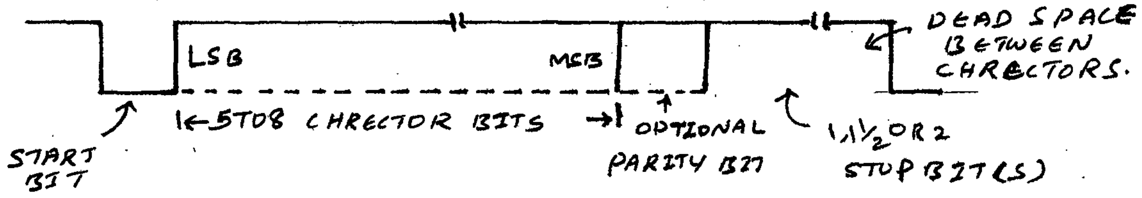


Fig 2.3 Asynchronous communication format

Fig 2.3 shows the bit format often used for transmitting asynchronous data.When no data is being sent,the single line is in a constant high or a marking state.The beginning of a data character is indicated by the line going low for one bit time.This bit is called a start bit.The data bits are then sent out on the line one after the other.The least significant bit is sent out first.Depending on the system,the data word may consist of 5,6,7 or 8 bits. Following the data bits, a parity bit is used to check for the errors in the received data. Some systems do not insert or look for a parity bit. After the data bits and parity bit ,the signal line is returned high for at least one bit time to identify the end of the character.This always high bit,is referred to as a stop bit.Some systems use 2 stop bits.

The term baud rate is used to indicate the rate at which serial data is transferred.Commonly used baud rates are 110,300,1200,2400,4800,9600 and 19200.

To interface a computer with serial data lines,the data must be converted to and from serial form.A parrallel in,serial out shift register and a serial in,parallel out shift register can be used to do this.A hand shaking circuitry is needed to ensure that the transmitter does not send data faster than it can be read in by the receiving system.There are available several programmable LSI devices which contain most of the circuitry needed for serial communication.A device such as the INS 8250 which can do asynchronous communication is referred to as a Universal Asynchronous Receiver Transmitter or UART.

Fig 2.4 shows the block diagram of 8250. The status register would contain error and other information concerning the state of the current transmission,and the control register is for holding the information that determines the operating mode of the interface.The data in buffer is paired with data in shift register.During an input operation,the bits are brought into the shift register one at a time and after a character has been received,the information is transferred to the data in buffer register,where it waits to be taken by the CPU.Similarly the data out buffer is associated with a parallel output shift register.An output is performed by sending data to the data out buffer,transferring it to the shift register and then shifting it to the serial output line.

Although there are several ways in which the four port registers can be addressed,it has been assumed that the status register can only be read from and control register can only be written into.Therefore an active signal on the read line would indicate either the status or data in buffer

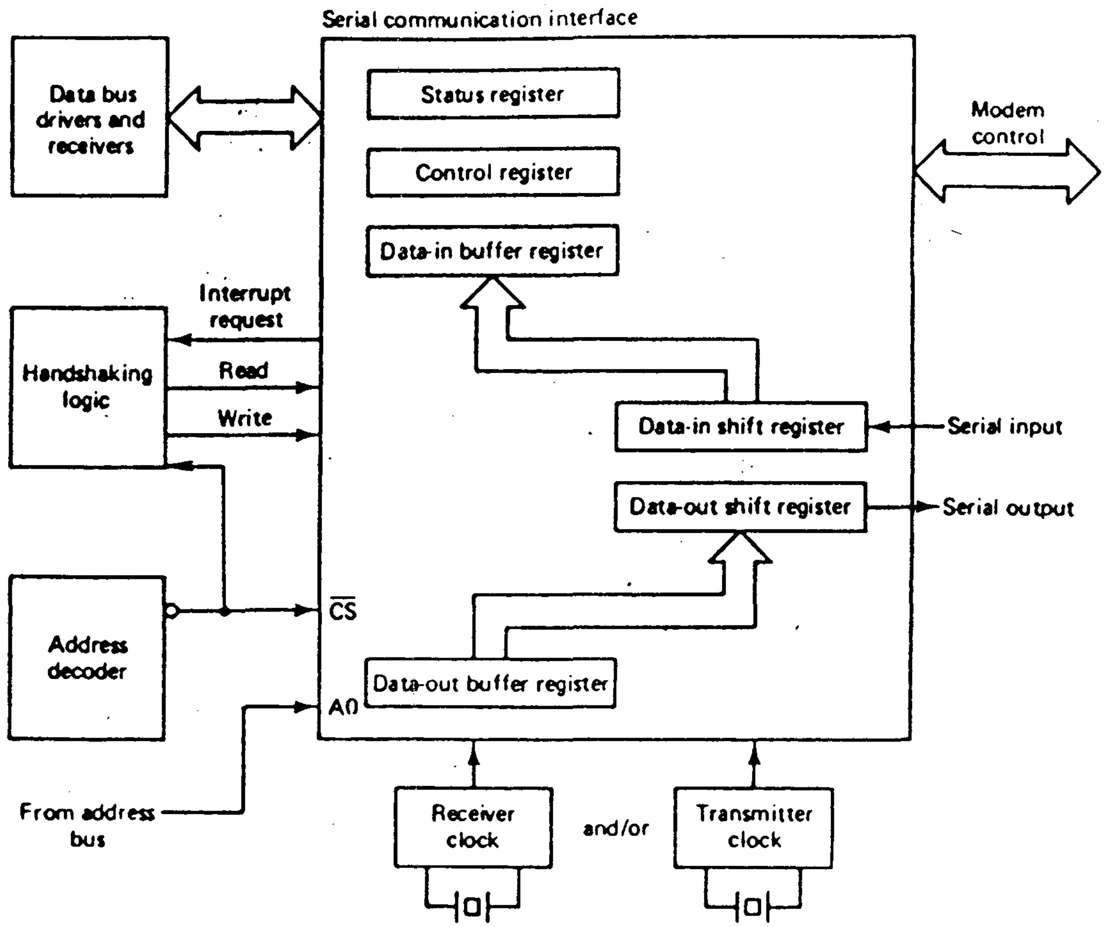


FIG 2.4

8250 UART BLOCK DIAGRAM

register.The interface has separate lines for sending and receiving information.So it can be used as a full duplex channel.

The information can be read from data_in register either by polling or on an interrupt basis.In our implementation,the characters are received on an interrupt basis.Accordingly the 8250 is programmed to interrupt whenever there is a character in data_in register.It is also programmed to the appropriate baud rate,number of stop bits,number of data bits,and the parity.

For details of programming the 8250, see appendix B.

2.4 RS-232C serial data transfer standards

Modems and other devices used to send serial data are often referred to as data communication equipment(DCE).The terminals or computers that are sending or receiving the data are referred as data terminal equipment(DTE).In response to the need for signal and hand shake standards between DCE and DTE the Electronics Industries Association (EIA) developed EIA standard RS-232C. This standard describes the configuration and function of 25 singnal and handshake pins for serial data transfer.It also describes the voltage level,impedence level,rise and fall times,maximum bit rate and maximum capacitance for these signal lines.RS-232C specifies 25 signal pins and it specifies that the DTE connector should be a male and,the DCE connector should be a female.A specific connector is not given,but the most commonly used connectors are the DB25-P male and the DB25-S female.It is important to note the order in which the pins are numbered.See appendix B for RS-232C pin configuration.

The voltage levels for all RS-232C signals are as follows-A logic high or mark is a voltage between -3V and -15V under load. A logic low is a voltage between +3V and +15V under load.Voltages such as +/-12V are commonly used.

III . PC TO PC COMMUNICATION

3.1 INTRODUCTION

For a terminal to communicate with a nearby computer,a simple RS-232C connection is sufficient.If the computer is distant,then a modem is required.

As another example of computer communication, suppose that we have several computers in one building or a complex of buildings,that need to communicate with each other.What is needed in this case is a high speed network,commonly called a local area network or LAN, connecting the computers together.In this part of the project,we are connecting two PCs via RS-232C,which will communicate at a baud rate of 9600. Since only two PCs are connected, no bus arbitration is required, However if more PCs are connected, then collisions have to be taken care of.

The facilities provided in this project are -

a. File transfer between the pcs .

b. Other utilities like mail and phone.

c. Resource sharing.

The file transfer utility runs in the background.When a request for file transfer is made from pc1 to pc2, a resident program on pc2 responds to the request and transfers the file in background.The user can continue his session as usual,and for the larger part,is unaware of the file transfer.A printer connected to one of the PCs can be accessed by both the PCs.

Our aim in this work is to get the maximum throughput of the pcs and share the resourses like printer. When we are using RS-232C,the data transfer rate is always slow.RS-232C can support only upto 9600 baud rate,while the CPU execution speed is much higher. It has to wait till each byte is transferred.Not only this,we are keeping the user idle.A user requsting for a file can wait,but it is not reasonable to keep a file sender idle.To overcome this problem,we are doing the serving job in the background.

When there is a request from pc1,the process residing in pc2 is initiated and requests the user for his permission for the file transfer.If the request is granted,it continues the job,otherwise simply returns.

LANGUAGES CHOSEN

For writing interrupt service routines and adjusting the interrupt vectors,assembly language is the natural choice and we chose the same for our RESPC program.

The rest of the module is developed in TURBO pascal.Pascal,as such is a good procedural language and it is much easier to debug a program written in pascal.Compiling and debugging with TURBO Pascal is very easy because of it's speed and inbuilt editor.TURBO also provides excellent and very useful features like interface to assembly language programs,executing MSDOS interrupt service routines,windowing,direct memory access,direct port addressing,efficient file handling and enabling and disabling I/O errors.

The assembly language interface is used in calling GETKEY and GETBUFF assembly functions.Many of the procedures like POSCUR and so on utilize the software interrupt service routine execution facility.Windowing has a direct usage with minor modifications for cursor positioning.Direct port addressing capability is utilized in addressing the 8250 communication port.

3.2 UART INTIALIZATION

The theory and programming aspects of UART were discussed in chapter2.The PC has two communication ports COM1 and COM2.Each of them can be independently programmed.For PC to PC communication,COM1 is used.The interrupt output of this device is connected to the IR4 interrupt of the 8259A priority interrupt controller in the PC mother board.The 8259A itself is mostly initialized by BIOS when the system is booted.However,since the UART is connected to IR4 of the 8259A,that input has to be unmasked.To do this,the current contents of the 8259A interrupt mask register are read in from address 21H.The bit corresponding to IR4 (bit 4) is then ANDed with a 0 to unmask the interrupt and the result put back in the register.

In this communication,only four wires are used (See Fig4.1). RXD (Receive Data), TXD (Transmit Data), protective ground and signal ground;and 8250 is programmed accordingly.

First the divisor latch register is programmed for the appropriate baud rate.To program the baud rate,the devisor latch address bit(DLAB) of line control register has to be set.So 80H is output to line control register.

Next, the divisor latch register 03F8H and 03F9H are programmed with the appropriate baud rate.For a baud rate of 9600,the values to be output are 00 to 03F9H and 0C to 03F8H.Since the communication parameters can be changed with in the session using the setup option,this baud rate is programmable and can be changed at any time.

Next, the line control register is programmed with the default parameters. For our communication,the parameters are 8 bit data,one stop bit and no parity.Hence 03 is output to the line control register.Like baud rate,this is also programmable and is taken care in setup.

Since characters are received on an interrupt basis,the enable data available interrupt bit (bit 0) in interrupt enable register is set.So 01 is output to interrupt enable register.

In this implementation,characters are received on an interrupt basis and buffered.These characters are later read from another program and processed.Let us consider a simple program where characters are received by polling the 8250 and displayed,and input from the keyboard is sent to another PC.

Initialize 8250

repeat

if keypressed,then read key and send it

if UART has a character,then read the character and display it

forever.

The above program works well at 300bd or 600bd. However for a baud rate of 1200 and above,the first character of each line of characters received from the host will be lost. After a carriage return is sent to the CRT,the display on the screen is scrolled up one line.Not only this,the input from the keyboard has to be processed and the received characters have to processed for escape and control sequences,which takes considerable time.To avoid loss of characters during this time,the characters are received on an interrupt basis and stored in a circular buffer.

3.3 IMPLEMENTATION:

This package consists of assembly programs and pascal programs,in which pascal programs are loaded and executed using assembly routines and assembly programs are called from pascal programs.

These programs are -

1. RESPC.ASM
2. PCNET.ASM
3. FACILITY.PAS
4. PHONE.PAS
5. MAIL.PAS

6. ASKFILE.PAS

7. DOS.PAS

3.3.1 RESPC:

See Fig 3.1 for a flowchart of RESPC.This program is written in assembly language.It stores the characters in a circular buffer and another function GETBUFF(which is in another module) reads characters from this buffer.Since both these functions share certain parameters,there should be a way to access these common parameters. In this implementation,the Data Segment of RESPC is stored in 0000:0184H. GETBUFF later loads the DS with the data in 0000:0184H and accesses different parameters as offsets with in the data segment.

Since communication port is connected to IR4 of 8259A,the 8259A will send interrupt vector 0C to the processor.So the starting address of the communication interrupt service routine is stored at vector 60H,using DOS function call 25H,later it will be stored at 0C by a routine TEMPCOM .

The communication interrupt service routine,which is resident all the time in memory, receives characters from PC2 and stores them in circular buffer.The flow chart is given in fig 3.2

Since the interrupt can occour at any time,it is important to save the DS register and load the DS with DATA-HERE.

The buffer used here here is a circular buffer.One

RESPC

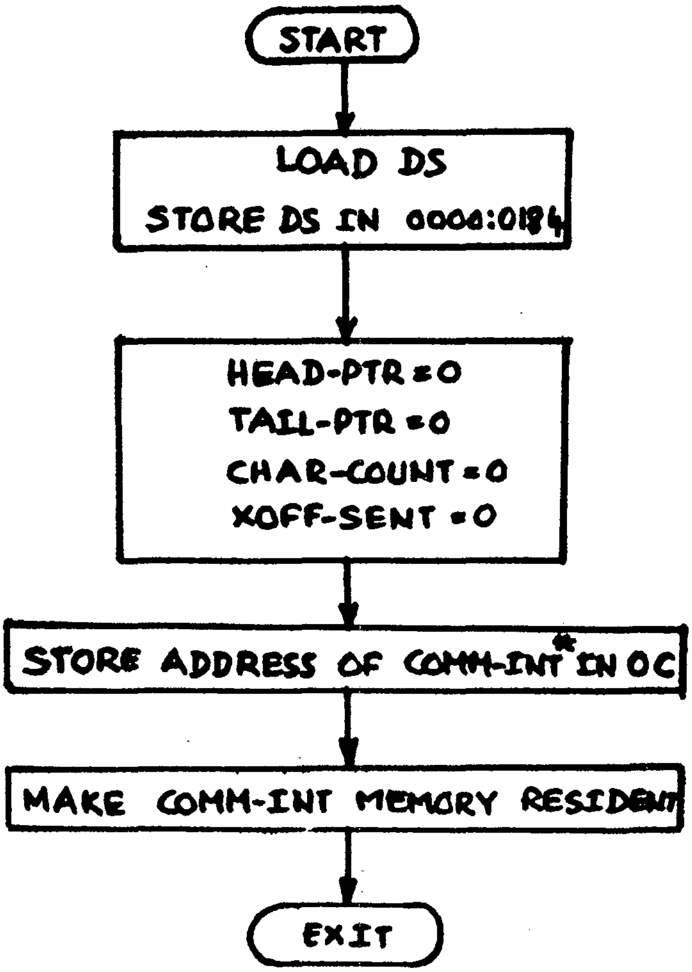

START
LOAD DS
STORE DS IN 0000:0184
HEAD-PTR = 0
TAIL-PTR = 0
CHAR-COUNT = 0
XOFF-SENT = 0
STORE ADDRESS OF COMM-INT* IN 0C
MAKE COMM-INT MEMORY RESIDENT
EXIT

FIG 3.1

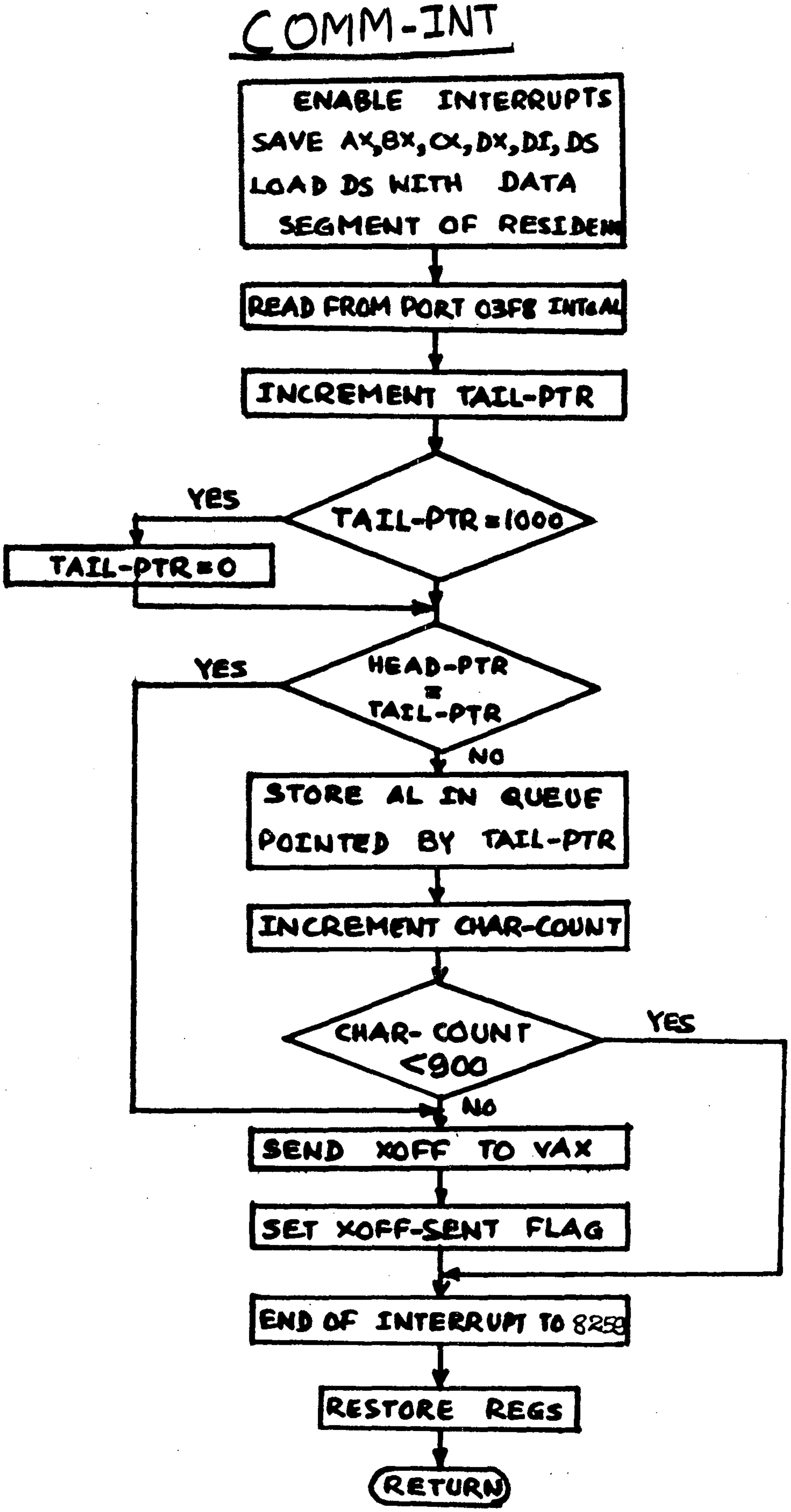

COMM-INT
ENABLE INTERRUPTS
SAVE AX, BX, CX, DX, DI, DS
LOAD DS WITH DATA SEGMENT OF RESIDENT
READ FROM PORT 03F8 INTO AL
INCREMENT TAIL-PTR
TAIL-PTR = 1000
YES
TAIL-PTR = 0
HEAD-PTR = TAIL-PTR
YES
NO
STORE AL IN QUEUE POINTED BY TAIL-PTR
INCREMENT CHAR-COUNT
CHAR-COUNT <900
YES
NO
SEND XOFF TO VAX
SET XOFF-SENT FLAG
END OF INTERRUPT TO 8259
RESTORE REGS
RETURN

FIG 3.2

pointer called the tail-pointer is used to keep track of where the next byte is written into buffer.Another pointer called the head-pointer is used to keep track of where the next character is to be read from the buffer.The buffer is circular because,when the tail-ptr reads the highest location in the memory space set aside for the buffer,it is wrapped around to the beginning of the buffer again.The head-ptr follows the tail-ptr around the circle as characters are read from the buffer.The checks are made on the tail-ptr before a character is written into buffer.

First the tal-ptr is brought into a register and incremented.This incremented value is then compared with the maximum number of bytes the buffer can load.If the values are equal,the pointer is at the highest address in the buffer.So the register is reset to zero,after current character is put into the buffer.The value will be loaded into the tail-ptr to wrap around to the lowest address in the buffer.

Secondly,a check is made to see if the incremented value of the tail-ptr is equal to the head-ptr.If the two are equal,it means that the current byte can be written,but for the next byte the buffer would be full.If this happens,an XOFF character is sent to PC2 to stop it from sending more characters and the xoff-sent flag is set.But. some characters may be sent by PC2 before we send XOFF.To avoaid this,every time a charecter is stored in buffer, a variable char-count is incremented.This char-count is compared with 950 and if they are equal,an XOFF is sent and xoff-sent flag is set.This way the host is restrained from sending more characters before the buffer gets filled up.

The other procedure which reads characters from this buffer(GETBUFF) checks the xoff-sent flag after every read.If this flag is set,it checks the char-count to see if there is enough space in the buffer.If the char-count is less than 750,it sends an XON and resets xoff-sent flag.This assures that there is a buffer space of 250 characters and RESPC can ressume buffering.

Finally before returning,an end of interrupt command must be sent to the 8259A to reset bit4 of the interrupt mask register.

PCNET is an assembly program,consisting of SYSINT,TIME_INT and TEMPCOM interrupt routines.Before proceeding to describe these routines,it should be born in mind that MSDOS is a single user operating system and it's code is not reentrant.In our program,the file transfer is carried in multitasking.SYSINT and TIME_INT serve this purpose.

For all I/O functions and certain special functions,every program has to request the operating system,with the proper parameters.The operating system does the specified task and gives control back to the called process.IBM pc provides some firmware routines for certain basic functions and MSDOS provides variety of routines under interupt 21H with different function calls.(See DOS technical reference manual for detailes).Since MSDOS is a single user operating system,we can run only one process at a time and only one function request is made at a time.The process requests for system services one after the other.Since MSDOS routines are not reentrant,in the multitasking,when a process enters the system routine,other process should not be allowed to enter.If this is allowed the system will crash.We can implement multitasking,by executing each process till it's time slice expires.This works very well if both processes are not requesting for system services at the same time.

But imagine a case,where multitasking is implemented and a process called a system function ,and it's time slice is over when it is halfway through in the system call.If control is passed on to the other process,and if that process also requests for the same system function,there is no way MSDOS can know that the request has come from the second process and it is in the way of serving it.Hence the register variables of the first process will be reinitialized ,which will lead to system crash.One solution to this is to execute the process,though it's time slice has expired.But this may lead to another problem,where the system routine may be indefinitely waiting for the input.For example it may indefinitely wait for an input from the keyboard.The user may take his own time in giving the input.During this time,the process is simply waiting for the input from the keyboard and the second process can not be served.Since MSDOD is serving one process,we should keep the other process's request in pending.Another solution to this problem is not to allow MSDOS to respond to keyboard I/O until a key has been pressed.This method is implemented in the following SYSINT routine.

3.3.2 SYSINT : When a system call is made by a process,it puts the appropriate values into the registers and executes the corresponding interrupt.Then control branches to the appropriate address and the routine is executed. When we run PCNET,it takes the address of the actual system routine and places it in vector 64H.It stores the starting address of SYSINT at vector 21H. So whenever a system call is made with vector 21h,The control is retained by SYSINT.

This SYSINT checks the int flag.If it is set,it gives control to the actual system routine.This is necessary because,when a process reads a key through MSDOS,it returns the ASCII value in AL register.If the key is an extended key,AL contains zero and another call must be made to get the extended code.When this happens,the next request must be served to the same process.For this,SYSINT sets the intflag, when the process is leaving the SYSINT in this particular case.

In the next step SYSINT checks whether the request is for keyboard I/O.If it is ,it simply loops until a key or keys are pressed.Then it sets the key flag and the int flag and gives control to the system routine.If

SYS-INT

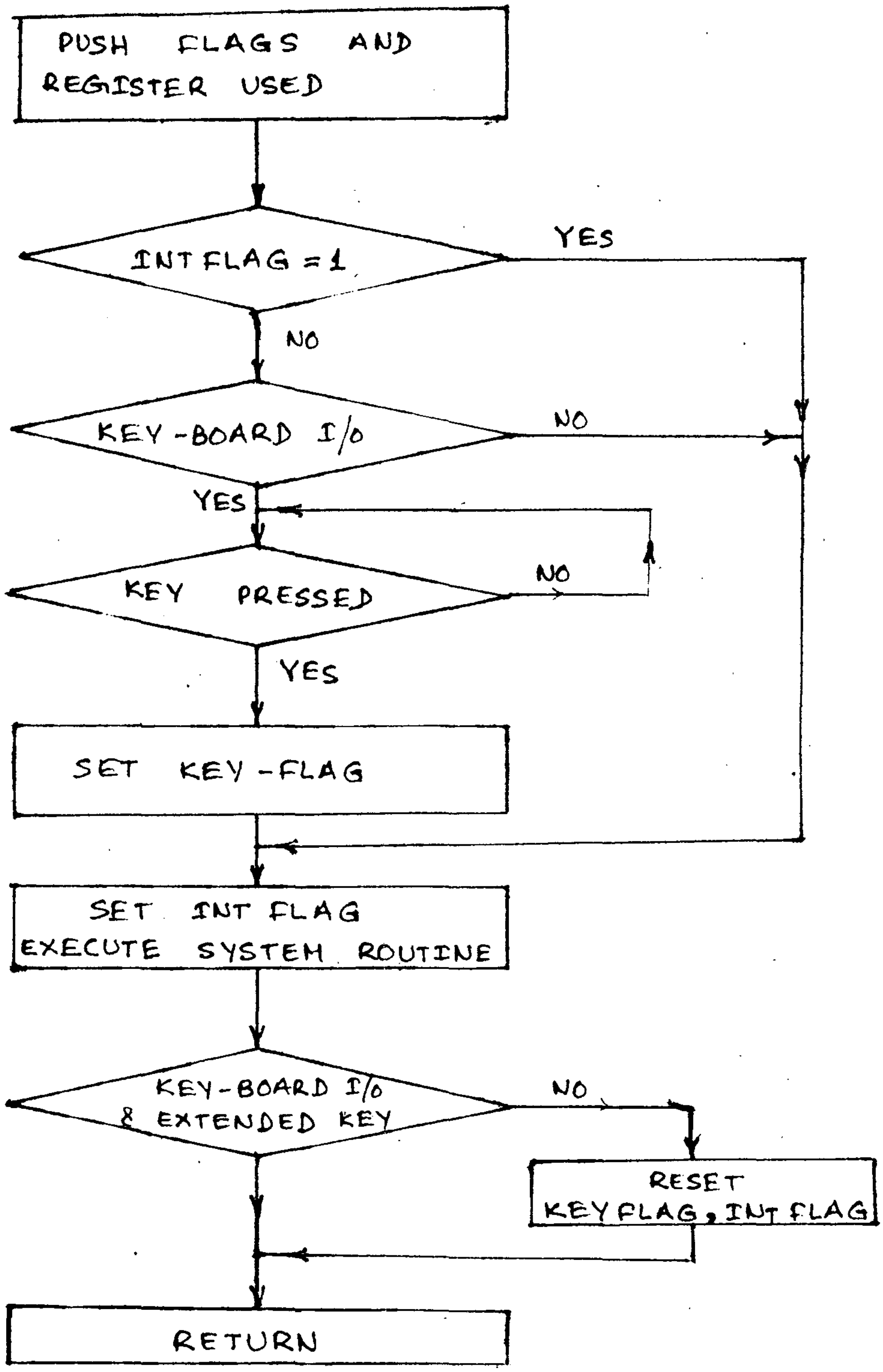


FIG 3.3

the request is not for keyboard I/O,it simply gives control to the system routine.Since it is looping for a keyboard input in SYSINT, TIME_INT can give control to the other process.This int flag serves as an indicator to the TIME_INT that a process is getting system service from MSDOS.

After executing the system service routine, control is returned to SYS_INT.Then it resets the intflag.If the returned value is that of an extended key,then it sets the int flag ,resets the keyflag and control is returned to the requested process.The flowchart is given in Fig 3.3

3.3.3 TIME_INT : IBM PC has 8253-5 timer chip,which has three timers in it.One is connected to the CPU through 8259 interrupt priority controller,the second one is connected to DMA and the third is connected to the speaker.When the system is booted,MSDOS programs the first timer to interrupt the processor periodically,so that the timer routine does the time keeping.Timer has the highest priority interrupt.It is connected to IRO of the 8259 interrupt priority controller. In our implementation,Multitasking is accomplished, using this timer. Whenever there is a timer interrupt,control is retained by our TIME_INT routine.This routine first does the system time keeping,then it pops the instruction pointer,code segment,and PSW of the interrupted process from the stack.It checks whether the interrupted process is getting served by MSDOS by checking the intflag or the code segment of the interrupted process.If the intflag is set or the code segment is equal to the segment

TIME-INT

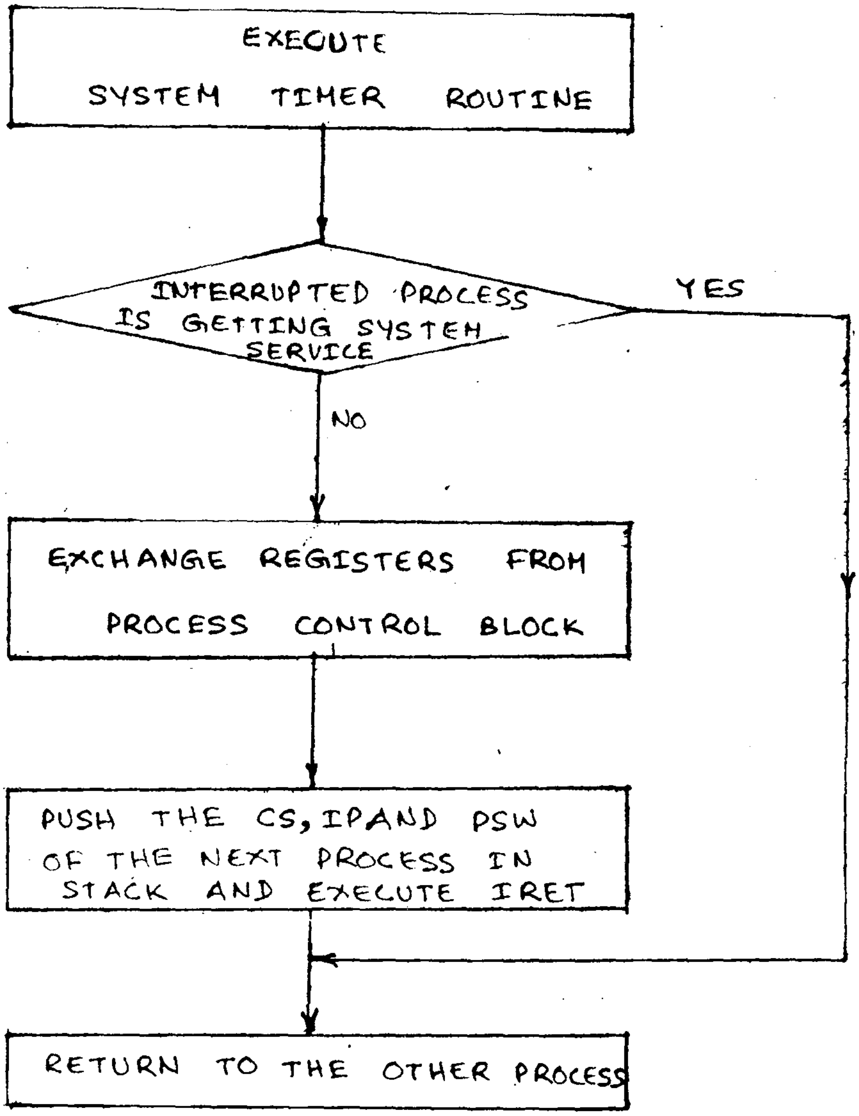


FIG 3.4

of the MSDOS system routine,it restores the stack and control is returned back to the interupted process.Otherwise,it stores all registers the procees control block and loads the registers with the values of the next process to be served, from the process control block of the other process .The PSW,CS and IP of the process to be given control are pushed onto the stack and control is given back to that process by executing IRET instruction. The flowchart is given in Fig 3.4

3.3.4 PCNET : This program initializes the 8250 such that,whenever it receives a character,it should interrupt the processor.It's interrupt level .on 8259 priority controller is IR4 and it's vector is 0Ch. PCNET stores the address of the TEMPCOM interrupt service routine at 0CH and 67H.This routine stores all the registers of the interrupted process,masks IR4 bit of the 8259 priority controller,so that another interupt is not recognized during the execution of this routine.Whenever there is an interrupt from 8250,it reads the characters from UART and checks if the caharacter is an escape character.If it is,then it gives control to a process,where it checks for the request and serves it.If the received character is not an escape character,then TEMPCOM unkmasks IR4,sends an end of interrupt to 8259 and calls the disable function.This disable function places the starting address of the TEMPCOM at vector 0Ch and loads all the registers of the interrupted process from the process control block and gives control to it.

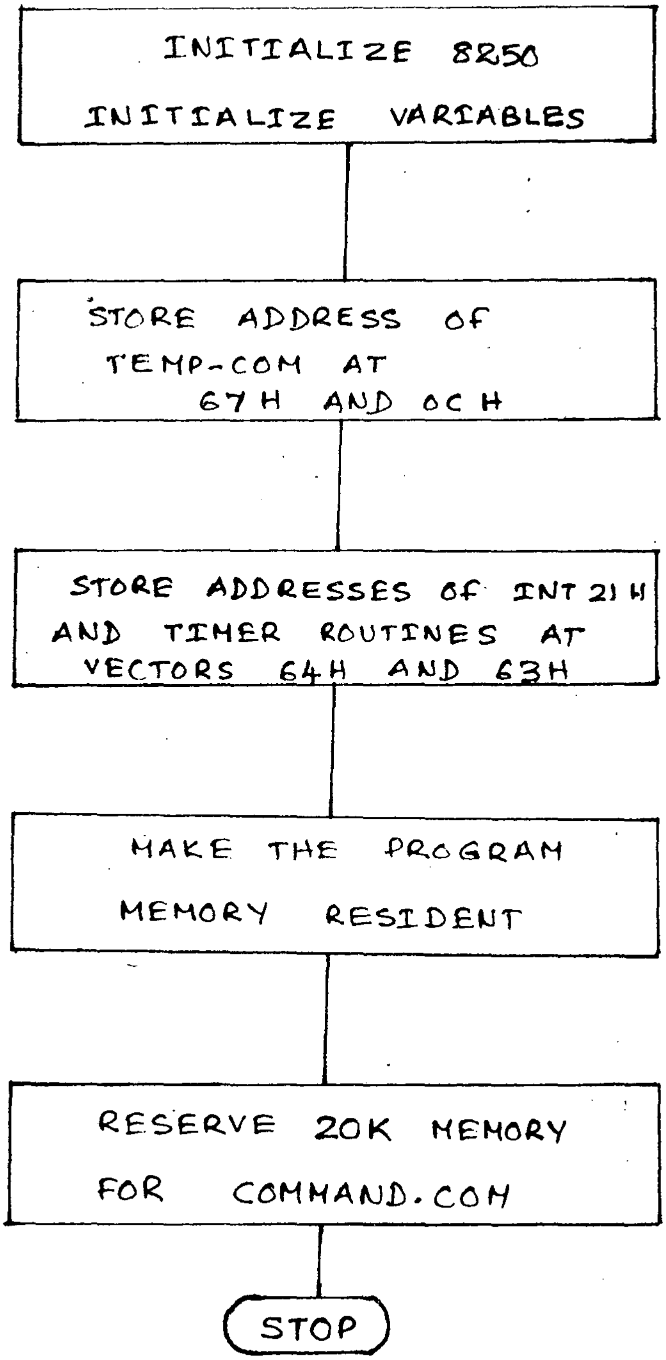

INITIALIZE 8250
INITIALIZE VARIABLES
STORE ADDRESS OF TEMP-COM AT 67H AND 0CH
STORE ADDRESSES OF INT 21H AND TIMER ROUTINES AT VECTORS 64H AND 63H
MAKE THE PROGRAM MEMORY RESIDENT
RESERVE 20K MEMORY FOR COMMAND.COM
STOP

FIG 3.5 (CONTD)

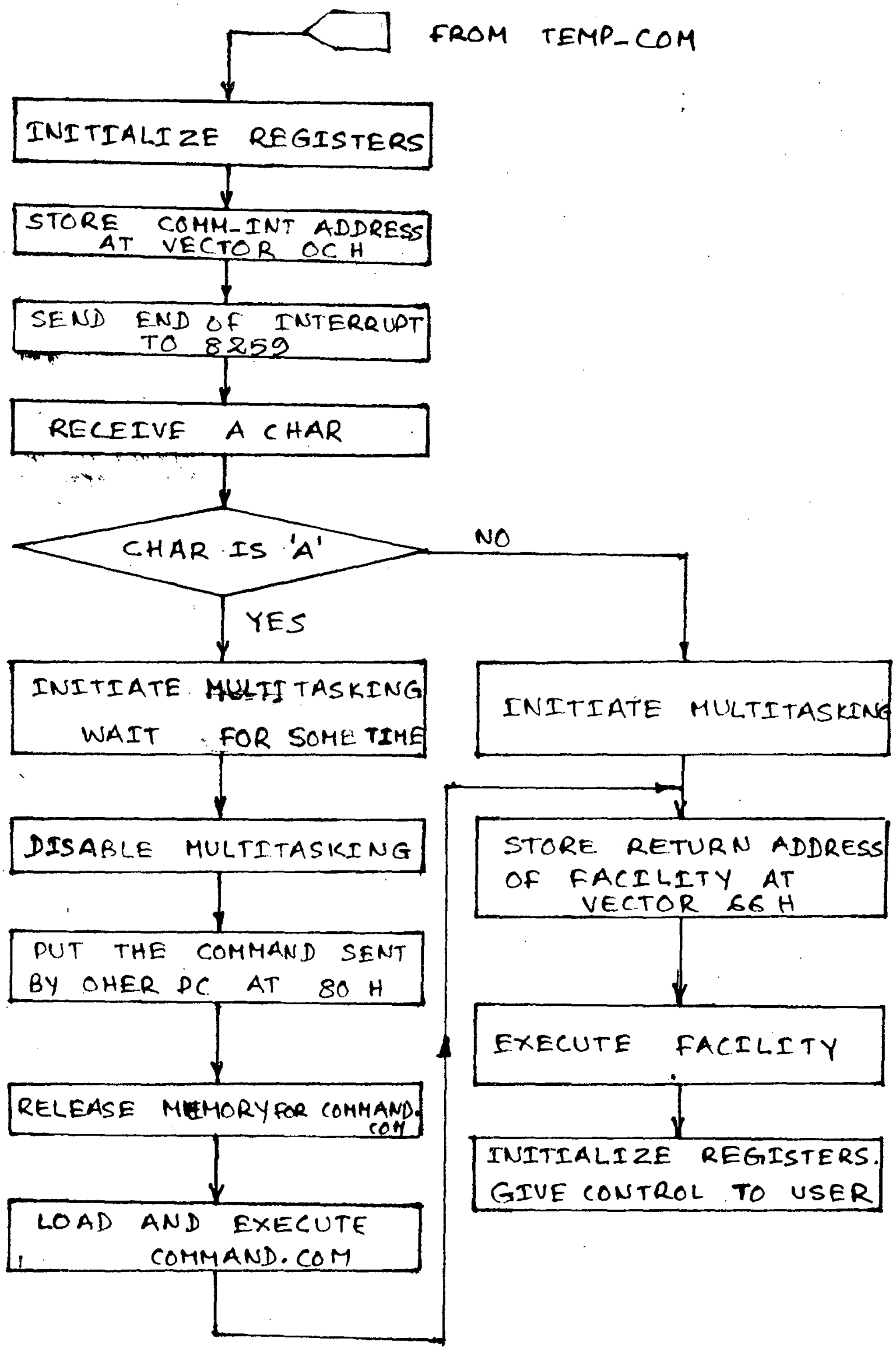

FROM TEMP_COM
INITIALIZE REGISTERS
STORE COMM_INT ADDRESS AT VECTOR OC H
SEND END OF INTERRUPT TO 8259
RECEIVE A CHAR
CHAR IS 'A'
NO
YES
INITIATE MULTITASKING WAIT FOR SOME TIME
INITIATE MULTITASKING
DISABLE MULTITASKING
STORE RETURN ADDRESS OF FACILITY AT VECTOR 66 H
PUT THE COMMAND SENT BY OHER PC AT 80 H
EXECUTE FACILITY
RELEASE MEMORY FOR COMMAND.COM
INITIALIZE REGISTERS. GIVE CONTROL TO USER
LOAD AND EXECUTE COMMAND.COM

FIG 3.5

PCNET initializes 8250 by calling INIT subroutine for communication parameters of 9600 baud rate,8 bits,no parity and one stop bit and to generate interrupt, whenever a character is recieved.Then it loads the starting address of TEMPCOM at 0Ch and 67H,stores the addresses of the original timer and interrupt 21H service routines in 63H and 64H respectively,then stores the SYSINT address at vector 21H and makes the whole program memory resident by reserving another 20k of memory space.

When TEMPCOM gives control to PCNET after receiving an escape character,it initializes all the registers to execute this process and then copies COMM_INT address from 60H to 0CH.It sends a character 'C' to the other computer so that it can goahead. If the received character is other than 'A',it stores the address of the TIME_INT routine at vector 08H and initializes multitasking. It then stores the return address of the pascal program at vector 66H and calls the pascal program FACILITY for further service.

If a user on one pc wants to run a command on the other pc,he will send an escape character ,followed by the character 'A'.To execute MSDOS commands,we should release around 17k of memory allocated to the current process to load a copy of COMMAND.COM into this memory.Then place the command string at offset 80H with the string length as the first byte and a carriage return as the string terminator .Then make the DS:DX to point to the string COMMAND.COM and make ES:BX point to the parameter block and load AL with zero,AH with 4BH,save SS and SP registers in an area other than stack and execute interrupt 21H,which loads and excutes COMMAND.COM. This COMMAND.COM picks up the command stored at offset 80H and executes it.

On return from the executed command,most of the registers have been changed,including SS and SP.These registers have to be restored.

If the received character is 'A', it stores the TIME_INT address at 08H,waits for some time and restores the original timer routine at 0CH. Then it reads the command sent by the other user and places it in 80H and executes the command as explained above.Then it calls the module FACILITY,by placing the character 'D' in variable TRAY.The flowchart is given in Fig 3.5.

3.3.5 FACILITY : This program,written in pascal consists of external and internal procedures and functions.All external procedures and functions are coded in assebmly.Let us see how assembly programs are called from Turbo pascal and how parameters are passed.

When an assembly routine is to be called from a pascal program as a procedure/function,it should be defined as external procedure/function in the pascal program.The assembly program has to be separately assembled,linked and converted to binary form by using EXE2BIN utility.

Let us consider a pascal program and an assembly program.

Pascal program

```
program pascal_assembly_interface;
function   decr(var   n   :   integer)   :   integer;
external 'decr.bin';
var i,j : integer;
begin
   i := 1;
   j := decr(i);
   write('i = ',i);
end.
```

Assembly program

```
; function decr(var n : integer);integer;
  decr  proc   near
        PUSH   BP
        MOV    BP,SP
        LES    DI,[BP+4]
        MOV    AX,ES:[DI]
        DEC    AX
        MOV    ES:[DI],AX
        POP    BP
        RET    6
        endp
```

where i is a variable in pascal initialized to 1.The assembly function DECR is called with the parameter i.The function takes the variable i,decriments it and returns the decrimented value.The pascal program then prints this returned value.

Let us see how the parameters are passed.Turbo Pascal passes parameters through stack.

At entry,the stack pointer points to the stacked return address of the caller to this routine.The higher address (sp+2) contains the address of the parameter passed by the caller.To access the parameter, we use the BP register.Since this BP register would have been used in the calling program,we must save BP as the first step in the assembly program.In principle,all the registers that are being used in the assembly routine have to be saved,and then restored when returning control to the caller.Then the current stack pointer is assigned to BP. Both SP and BP now address the value of the saved BP register.The return address and the BP register values are each of two bytes,hence the parameter is found on the stack at location [BP+4].The parameter is taken from this area,incremented and put back at the same location. BP register is restored and control is returned to the caller by executing RET. RET pops only the return address from the stack. Since we mustalso pop the paremeter,we shold use RET 6.

The following external procedures are used -

GETKEY : This function checks,if there is any input from the keyboard and returns the data if any,to the called program.The flow chart is given in Fig 3.6.

GETKEY

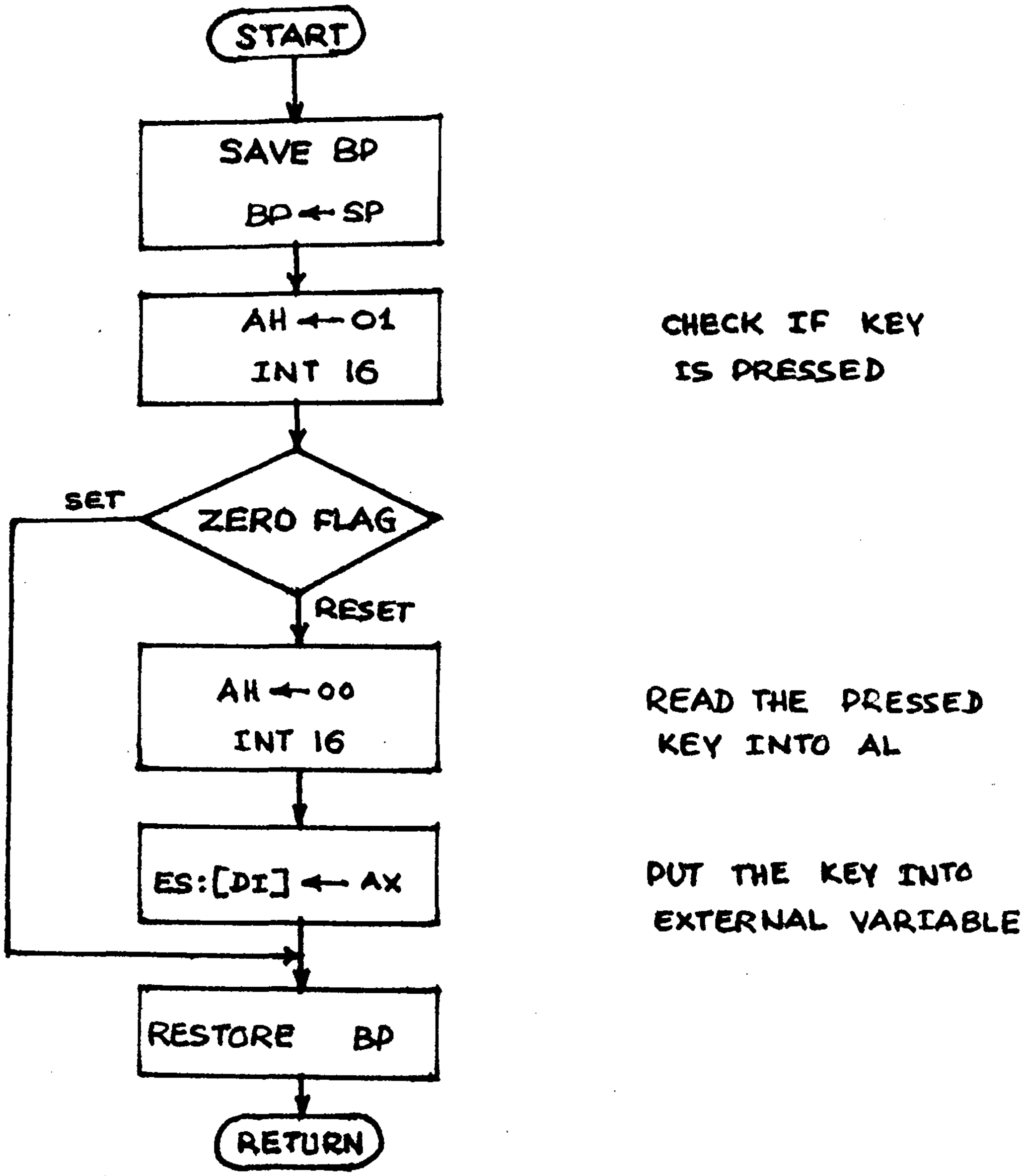


FIG 3.6

INT 16H BIOS routine provides different functions,depending on the value loaded in reg AH. AH=0 returns the code for a pressed key in AL. AH=1 returns the zero flag=0 if a key has been pressed. INT 16 is called with AH=1. If zflag is set,there is no input from the keyboard and execution returns to the caller.If the zflag is 0,the keyboard input is read into AL and the value returned.

GETBUFF : This function checks if there is date in the circular buffer and returns the data,if there is any.The flow chart is given in Fig 3.7

All the registers are saved.The contents of [0000:0184] are loaded into DS,so that the variables of RESPC are accessible here.Once DS points to the data segment,the variables within the date segment are accessible as off sets using the registers BX and DI.

By comparing the head and the tail pointers,a check is made to see if there are any characters in the buffer. If not,the execution is returned to the caller.If a character is available in the buffer,it is read and the head pointer updated to point to the next available character.If the pointer is at the top of the space allocated for the buffer,the pointer is wrapped around to the start of the buffer. The read character is then passed on to the external variable. As discussed earlier,this function also checks the xoff_sent flag and sends an XON if there is enough space in the buffer.

INTPAS : This routine stores the starting address of FACILITY at address 65H and makes the whole program memory resident and gives control to MSDOS.

RETURN : After the FACILITY program is called and executed from PCNET,this RETURN procedure takes the return address stored at vector 66H and gives control to PCNET.

NOSWAP : This procedure disables multitasking by placing the address of the MSDOS timer routine at vector 08H.

The additional procedures are -

FINDCUR : Finds the position of the cursor by loading 03 into AH, 00 into BX and executing interrupt 10H.The column number is contained in DL and the row number in DH. Row number varies from 0 to 23 and column number varies from 0 to 79 in PC,while they vary from 1 to 24 and 1 to 80 respectively in normal use so a 1 is added to the row and column numbers determined above.

POSCUR : Positions the cursor at the given row and column,by loading 02 in AH,00 into BX ,rownumber-1 in DH,column number-1 in DL and executing interrupt 10H.A one is subtracted because of the same arguement as above.

DISPLAY :This Is used in displaying a character with a given attribute.When characters are to be displayed in a mode other than normal,the attribute byte is set and this procedure is called to display the character in the required

FUNCTION GETBUFF

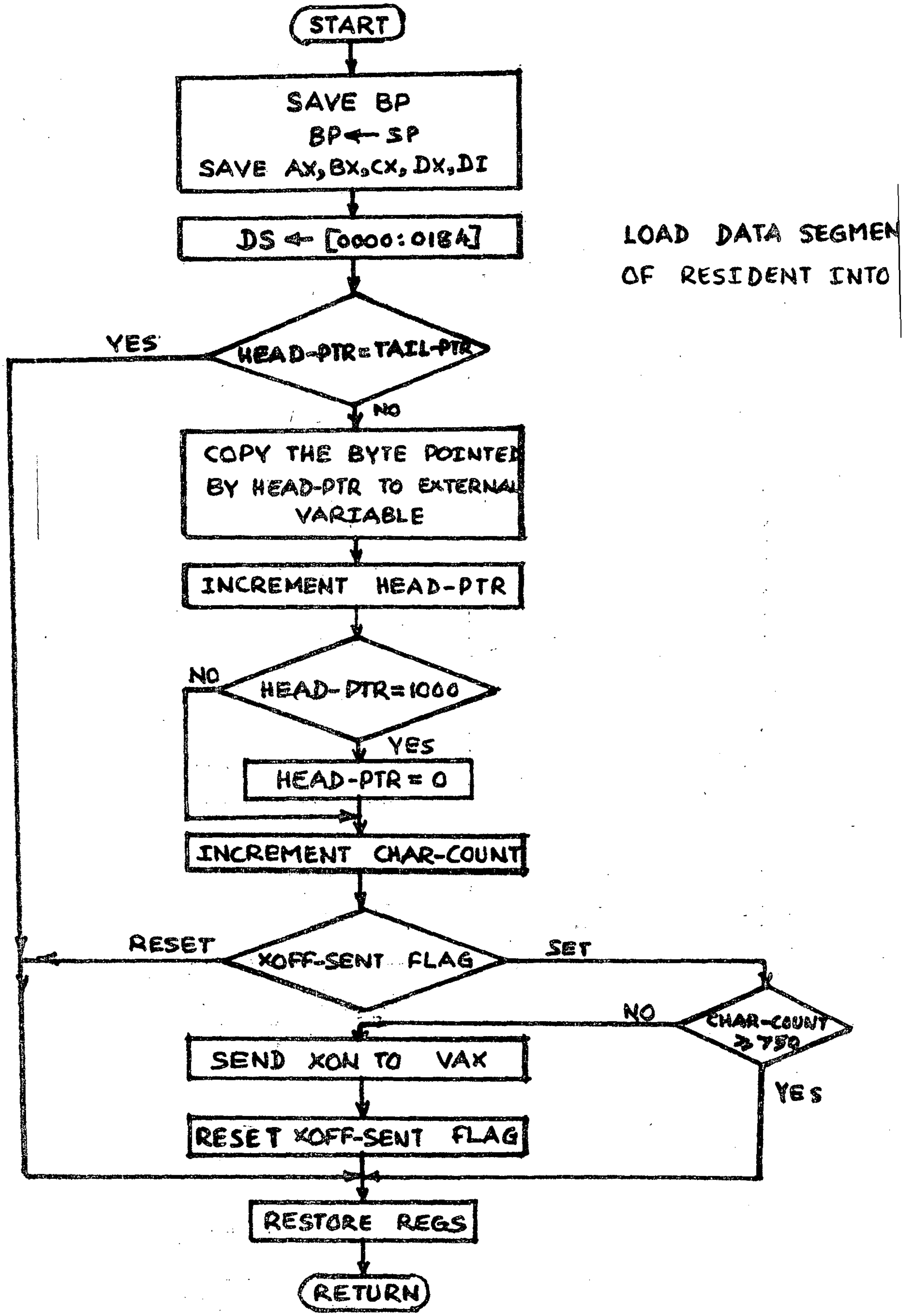


FIG 3.7

attribute.For carriage return,line feed and tab,the characters are displayed as they are.For the rest,the character is loaded in AL,09 into AH,the attribute into BL,the number of characters into CL and interrupt 10H is executed.The cursor is moved to the next column.

SET_DISPLAY : Displays a given string with a given attribute.It repeatedly calls the above procedure for each character of the string.

GETCHAR : In some cases it is necessary to wait till a character is received.This procedure waits till a character is received by repeatedly calling GETBUFF.

SEND : Sends an integer to the host.It reads the line status register of COM1 and checks if bits 5 and 6 corresponding to transmitter holding register empty and transmitter shift register empty are set.If they are set,then the data is sent to the output port [03F8].

READFILENAME :Reads the filename sent by the other user into the string variable called filename.If it is unsuccessfull in reading the filename,it sets fflag.

SENDFILE: This is the actual procedure which runs in the multitasking mode.It reads data from the required file and sends it to the other pc.Data is read from the file in blocks due to the following reason.When SENDFILE runs in the background and the user runs the directory command in the foreground,MSDOS flushes all it's file discriptors in the

SENDFILE

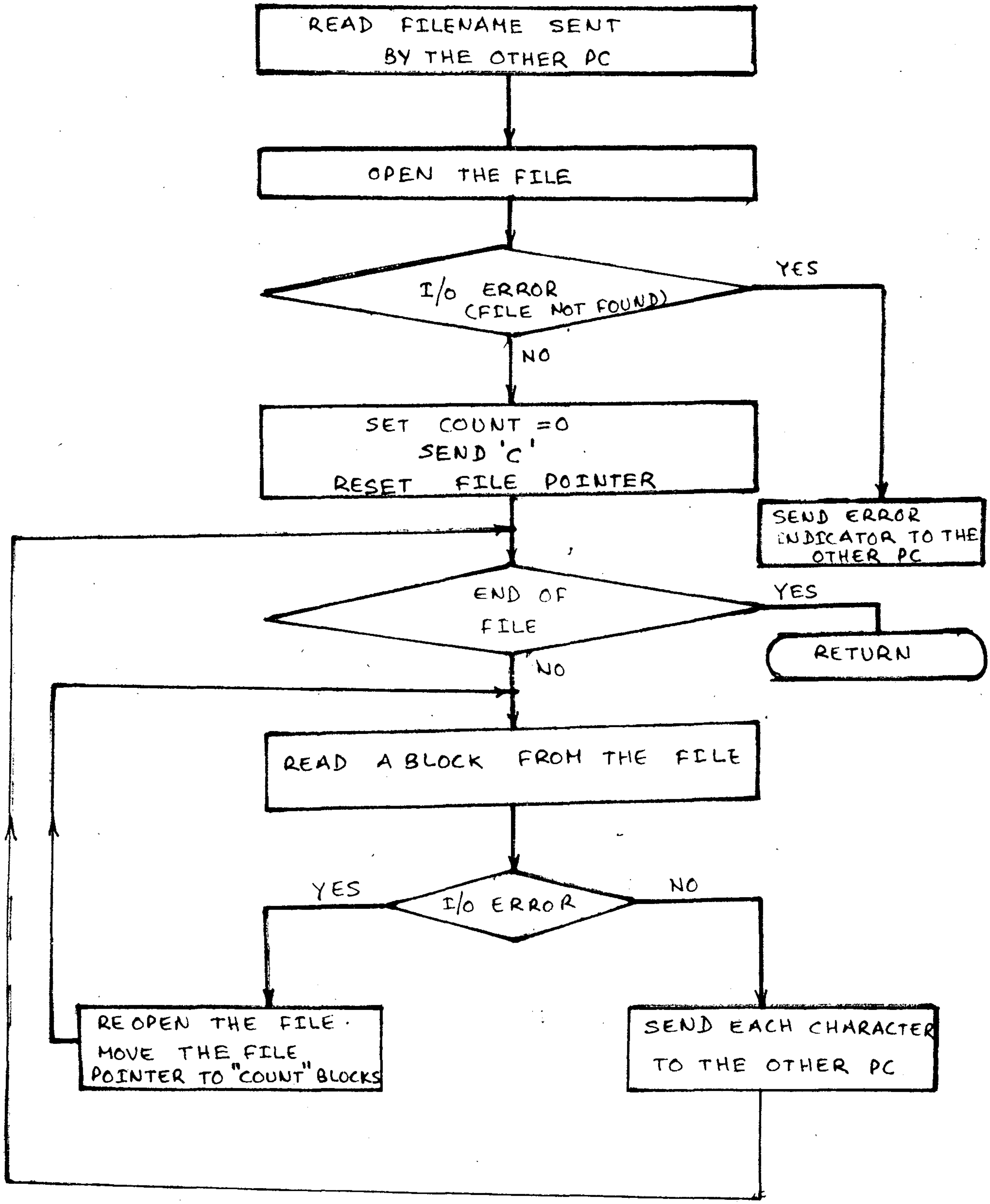


FIG 3.8

memory and hence the file handle of the current file used in the background job will be lost and there will be I/O error. To overcome this problem,this program reads the file in blocks and keeps a count of the number of blocks read.If there is any error in reading the file,it reopens the file and positions the file pointer at the next block to be read.

This SENDFILE procedure reads the filename and opens that file.If the file is not existing,it sends an error message to the other pc, then disables multitasking and returns to the main program. It reads the file block wise and sends the characters one after the other.Then it closes the file and disables multitasking.The flow chart is given in fig 3.8

GETFILE : This procedure is called when a user on pc2 mails a file.It disables the multitasking by calling NOSWAP,reads the filename sent by pc2,creats a file with that name and reads the contents of the file sent by pc2 and stores them in disk till the end of file is encountered.If it is unable to create a file,it sends an error message to the other pc.

SPEAK : This procedure is called,when a user on pc2 makes a phone call to pc1.It prompts the user for his permission.If the permission is granted,it creats two windows on the screen and maintains the message profiles in these two windows.It displays both the data sent and data received.This phone utility is terminated with a c,and the other user is also taken out of PHONE.

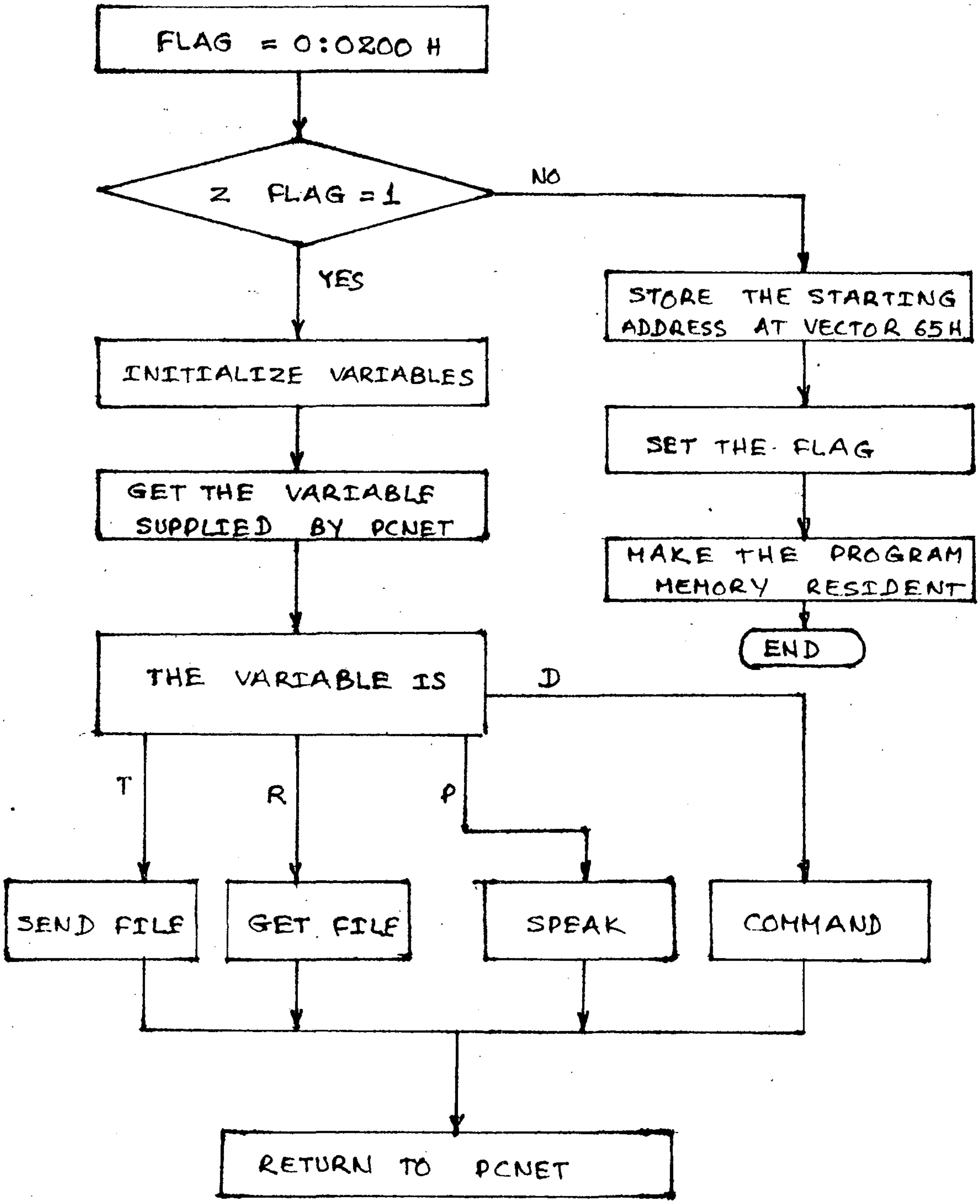

FLAG = 0:0200 H
Z FLAG = 1
NO
YES
STORE THE STARTING ADDRESS AT VECTOR 65H
INITIALIZE VARIABLES
SET THE FLAG
GET THE VARIABLE SUPPLIED BY PCNET
MAKE THE PROGRAM MEMORY RESIDENT
END
THE VARIABLE IS
D
T
R
P
SEND FILE
GET FILE
SPEAK
COMMAND
RETURN TO PCNET

FIG 3.9

COMMAND : This procedure is called after the execution of the DOS call, specified by the user on pc2.This will send the output of that command to the pc2,which is stored in the file C:REDIRE$CT.

The main program FACILITY is called from the assembly progarm PCNET by keeping the request in the common variable TRAY.It checks the value in TRAY.If it is -

'T' - calls SENDFILE procedure.

'R' - calls GETFILE procedure.

'P' - calls SPEAK procedure.

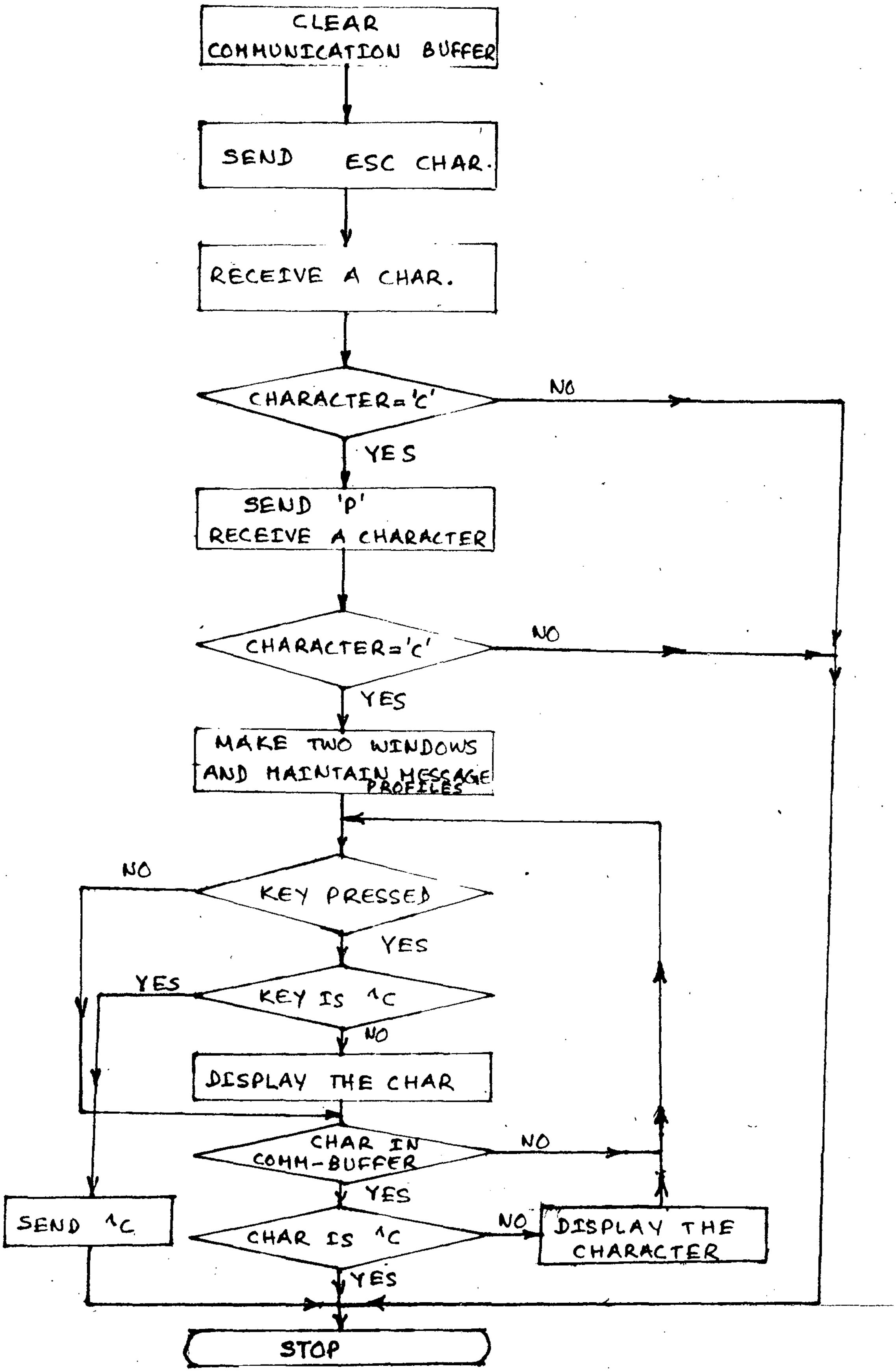
'D' - calls COMMAND procedure.

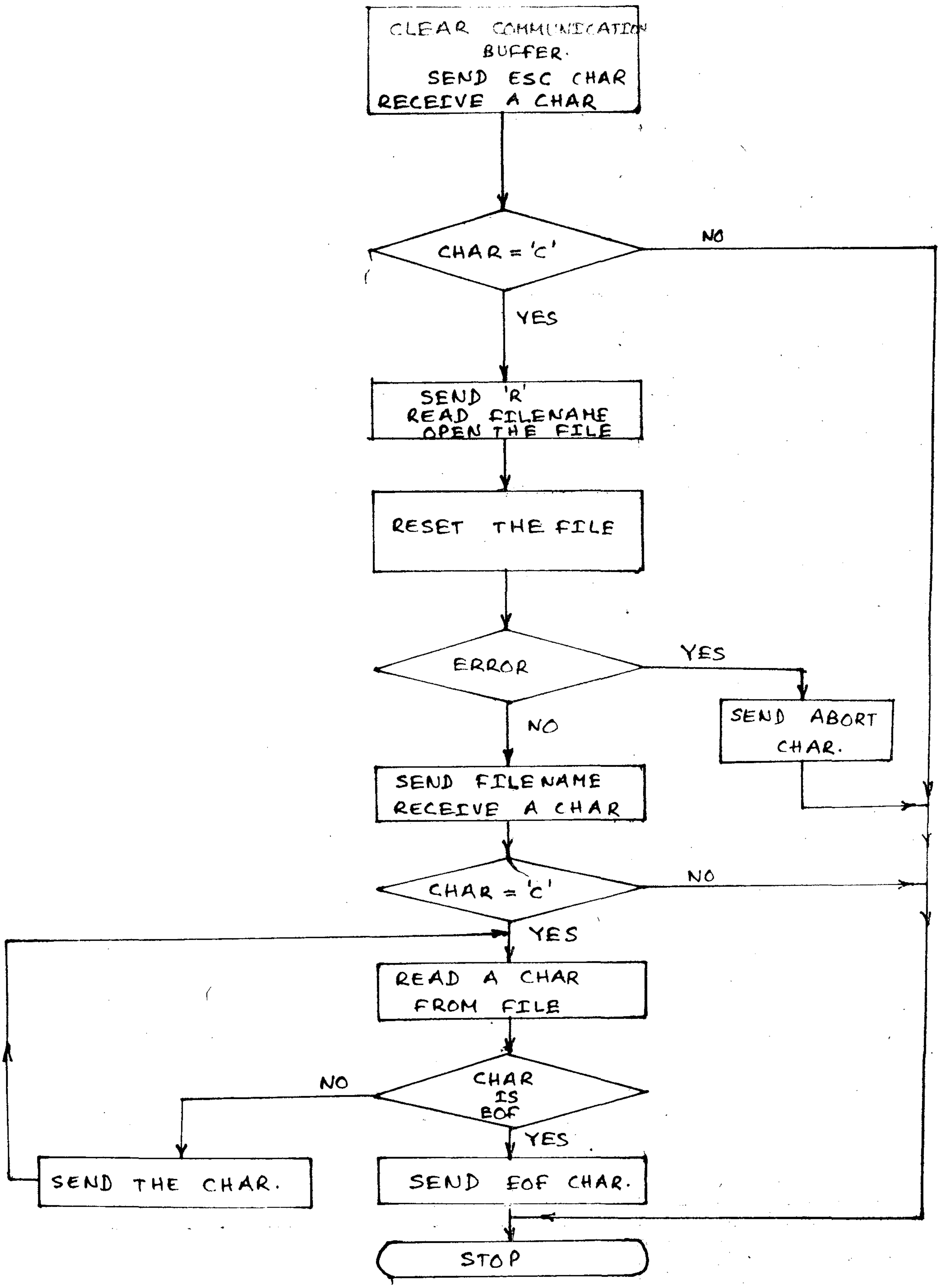
After the execution of the procedure,it returns control to the PCNET which returns control to the user.The flow chart is given in fig 3.9.

3.3.6 PHONE : This is an independent pascal program and has to be run separately to invoke PHONE facility.To use the phone facility,PHONE has be run on one of the pcs . When run, it checks to see if the other user is interested in PHONE and proceeds just SPEAK procedure described above. The flowchart is given in Fig 3.10.

3.3.7 MAIL: This is an independent program written in pascal.It is invoked when a user wants to mail a file to the other user.It sends the filename and the contents of the file to the other user.Before sending a character,a check is made to see if an XOFF character is sent by the other pc.If an XOFF is received,character transmission is suspended till an XON character is received.This makes sure that characters

PHONE


CLEAR COMMUNICATION BUFFER
SEND ESC CHAR.
RECEIVE A CHAR.
CHARACTER='C'
NO
YES
SEND 'P'
RECEIVE A CHARACTER
CHARACTER='C'
NO
YES
MAKE TWO WINDOWS AND MAINTAIN MESSAGE PROFILES
NO
KEY PRESSED
YES
YES
KEY IS ^C
NO
DISPLAY THE CHAR
CHAR IN COMM-BUFFER
NO
YES
SEND ^C
CHAR IS ^C
NO
DISPLAY THE CHARACTER
YES
STOP


CLEAR COMMUNICATION BUFFER.
SEND ESC CHAR
RECEIVE A CHAR
CHAR = 'C'
NO
YES
SEND 'R'
READ FILENAME
OPEN THE FILE
RESET THE FILE
ERROR
YES
SEND ABORT CHAR.
NO
SEND FILENAME
RECEIVE A CHAR
CHAR = 'C'
NO
YES
READ A CHAR FROM FILE
CHAR IS EOF
NO
YES
SEND THE CHAR.
SEND EOF CHAR.
STOP

are not lost due to buffer overflow.If there is any error in creating a file at the other node,it will receive an error indicator and aborts with an appropriate message. The flowchart is given in Fig 3.11.

3.3.8 ASKFILE : This an independent pascal program,which has to be run to request a file transfer from the other node.The other system sends the data in the file with the consent of the user.It sends the data in the file (if the file is existing).ASKFILE receives this data and stores it in a file.If the file is not existing at the other node,it receives an error message and aborts with an appropriate message. The flowchart is given in Fig 3.12.

3.3.9 DOS : When a user on pc1 wants to run a program on pc2,he has to invoke this program.Typical application of this facility is to use the printer connected to the other pc with a simple command from this pc.Similarly he can see the directory on the other pc from his pc.This program when run,prompts for the command to be run on the other pc.This command is sent to the other pc for execution.Any output of this command is redirected back to the host pc and displayedon screen.When the command is entered,this command is appended with C:REDIRE$CT string ,and a new string is created with the string length at the first position followed by C.This string is then sent to the other pc,where it is executed by loading the COMMAND.COM. Any output is buffered in to a file and is transmitted back to

ASKFILE

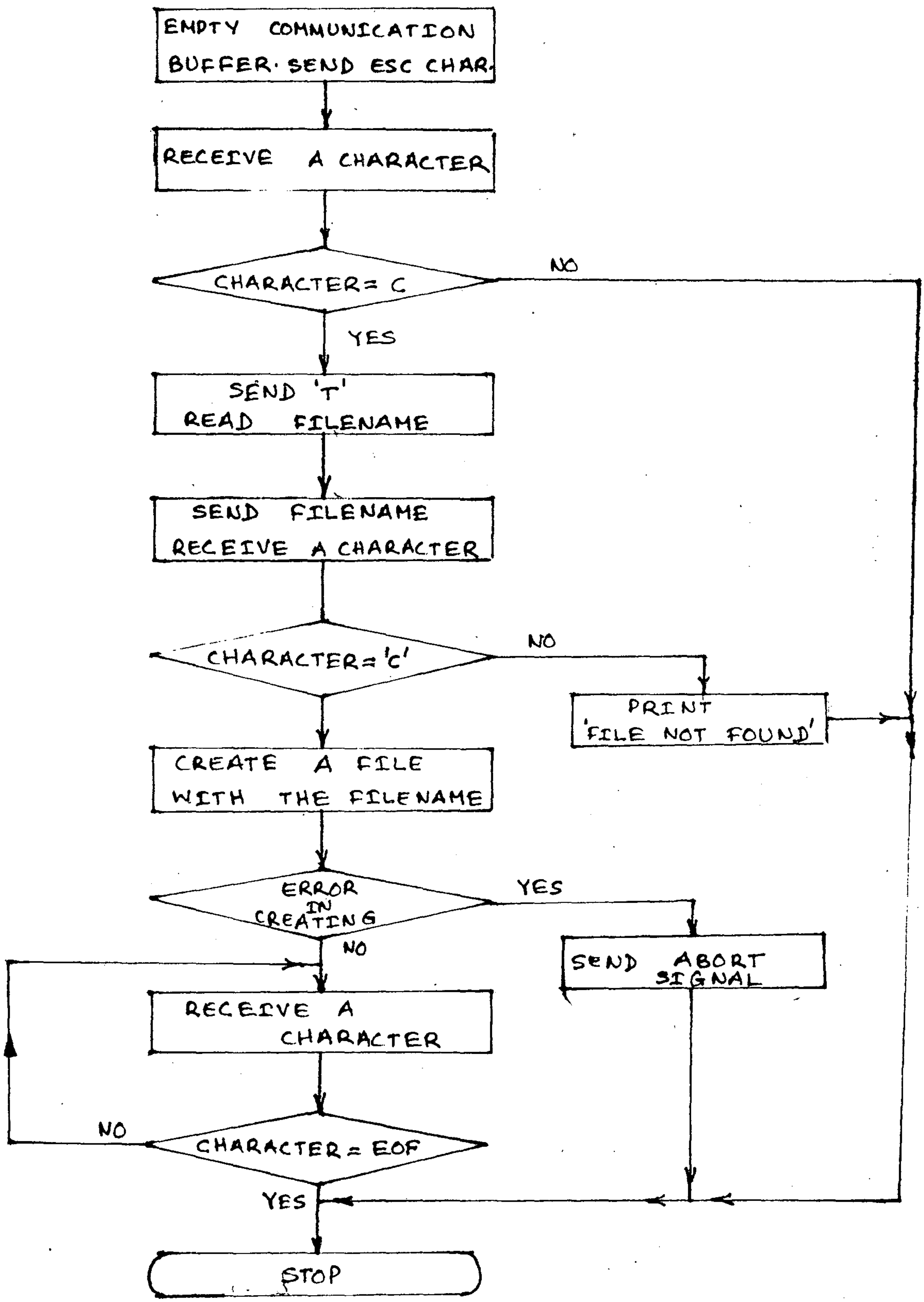


FIG 3.12

this PC and is dsplayed on the screen.

IV INSTRUCTIONS FOR USE

To use the PC to PC communication utilities, proceed as follows - step1 : install C drive and copy COMMAND.COM onto it.

step2 : run RESPC

step3 : run FACILITY

step4 : run PCNET

You need to follow the above steps on both the PCs only once,when you boot the system.

If you are currently using PC1 and want to get a file from PC2 , run ASKFILE .

When prompted,give the filename. With the approval of the other user,the file will be transferred to your disk.

To mail a file to the other user ,run MAIL. When prompted,give the filename. With the consent of the other user,the file will be mailed to him.

To make a phone with the other user, run PHONE . The other user will be informed and if interested,will go into PHONE and you can proceed. Exit the phone with ^C.

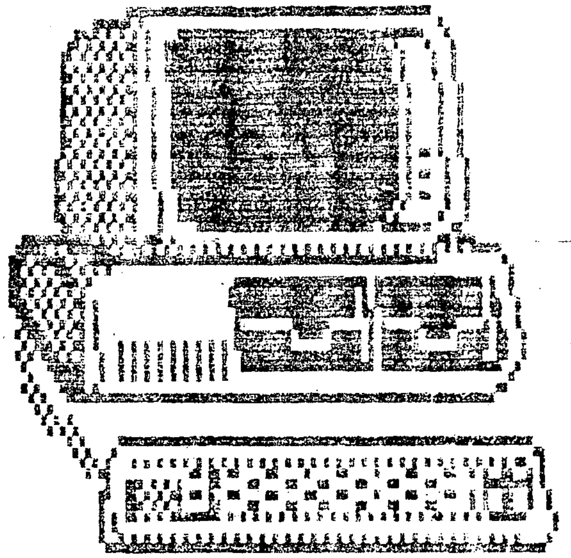
To run DOS commands on the other pc, run the program DOS . When prompted,give the command you want to execute.You can use this facility to print your file on the printer connected to the other pc.However,to print a file,first you have to mail the file to the other system.You can also see the directory of the other user.

V FUTURE EXTENSIONS AND MODIFICATIONS

This package can be extended and modified to give various other facilities to the user.

The file transfer utilities can be modified to include sub directory and/or wild card specifications.These file transfer utilities transfer only character files.Integer files like object files can not be transferred.The program can be modified to transfer integer files by using character stuffing.

This PC to PC communication facility can be extended to connect more than two pcs. Collision detection should be incorporated. If at least one of the system is a PC-XT,it can serve as a file server and a typical LAN sytem can be build up.The ultimate and most usefull architecture is the one,in which any PC can either communicate with the other PC or with VAX with simple software control.



APPENDIX A

8088

instruction

set

ADD

ADD destination,source
Addition

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	ADD CX, DX
register, memory	9 + EA	1	2-4	ADD DI, [BX] ALPHA
memory, register	16 + EA	2	2-4	ADD TEMP, CL
register, immediate	4	—	3-4	ADD CL, 2
memory, immediate	17 + EA	2	3-6	ADD ALPHA, 2
accumulator, immediate	4	—	2-3	ADD AX, 200

AND

AND destination,source
Logical and

Flags	O	D	I	T	S	Z	A	P	C
	0				X	X	U	X	0

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	AND AL,BL
register, memory	9 + EA	1	2-4	AND CX,FLAG_WORD
memory, register	16 + EA	2	2-4	AND ASCII [DI],AL
register, immediate	4	—	3-4	AND CX,0F0H
memory, immediate	17 + EA	2	3-6	AND BETA,01H
accumulator, immediate	4	—	2-3	AND AX, 01010000B

CALL

CALL target
Call a procedure

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Examples
near-proc	19	1	3	CALL NEAR_PROC
far-proc	28	2	5	CALL FAR_PROC
memptr 16	21 + EA	2	2-4	CALL PROC_TABLE [SI]
regptr 16	16	1	2	CALL AX
memptr 32	37 + EA	4	2-4	CALL [BX].TASK [SI]

CBW

CBW (no operands)
Convert byte to word

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CBW

CLC

CLC (no operands)
Clear carry flag

Flags	O	D	I	T	S	Z	A	P	C
									0

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLC

CLD

CLD (no operands)
Clear direction flag

Flags	O	D	I	T	S	Z	A	P	C
		0							

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLD

* For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer. Mnemonics © Intel, 1978.

CLI

CLI (no operands)
Clear interrupt flag

Flags: O D I T S Z A P C — I: 0

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLI

CMC

CMC (no operands)
Complement carry flag

Flags: O D I T S Z A P C — C: X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CMC

CMP

CMP destination, source
Compare destination to source

Flags: O D I T S Z A P C — O: X, S: X, Z: X, A: X, P: X, C: X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	CMP BX, CX
register, memory	9 + EA	1	2-4	CMP DH, ALPHA
memory, register	9 + EA	1	2-4	CMP [BP + 2], SI
register, immediate	4	—	3-4	CMP BL, 02H
memory, immediate	10 + EA	1	3-6	CMP [BX].RADAR[DI], 3420H
accumulator, immediate	4	—	2-3	CMP AL, 00010000B

CMPS

CMPS dest-string, source-string
Compare string

Flags: O D I T S Z A P C — O: X, S: X, Z: X, A: X, P: X, C: X

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string, source-string	22	2	1	CMPS BUFF1, BUFF2
(repeat) dest-string, source-string	9 + 22/rep	2/rep	1	REPE CMPS ID, KEY

CWD

CWD (no operands)
Convert word to doubleword

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	5	—	1	CWD

DAA

DAA (no operands)
Decimal adjust for addition

Flags: O D I T S Z A P C — O: X, S: X, Z: X, A: X, P: X, C: X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	DAA

DAS

DAS (no operands)
Decimal adjust for subtraction

Flags: O D I T S Z A P C — O: U, S: X, Z: X, A: X, P: X, C: X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	DAS

* For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

DEC	DEC destination Decrement by 1	Flags: O D I T S Z A P C X X X X X

Operands	Clocks	Transfers*	Bytes	Coding Example
reg16	2	—	1	DEC AX
reg8	3	—	2	DEC AL
memory	15 + EA	2	2-4	DEC ARRAY [SI]

DIV	DIV source Division, unsigned	Flags: O D I T S Z A P C U U U U U U

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	80-90	—	2	DIV CL
reg16	144-162	—	2	DIV BX
mem8	(86-96) + EA	1	2-4	DIV ALPHA
mem16	(150-168) + EA	1	2-4	DIV TABLE [SI]

ESC	ESC external-opcode,source Escape	Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
immediate, memory	8 + EA	1	2-4	ESC 6,ARRAY [SI]
immediate, register	2	—	2	ESC 20,AL

HLT	HLT (no operands) Halt	Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	HLT

IDIV	IDIV source Integer division	Flags: O D I T S Z A P C U U U U U U

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	101-112	—	2	IDIV BL
reg16	165-184	—	2	IDIV CX
mem8	(107-118) + EA	1	2-4	IDIV DIVISOR_BYTE [SI]
mem16	(171-190) + EA	1	2-4	IDIV [BX].DIVISOR_WORD

IMUL

IMUL source
Integer multiplication

Flags	O	D	I	T	S	Z	A	P	C
	X				U	U	U	U	X

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	80-98	—	2	IMUL CL
reg16	128-154	—	2	IMUL BX
mem8	(86-104) + EA	1	2-4	IMUL RATE BYTE
mem16	(134-160) + EA	1	2-4	IMUL RATE WORD'BP][DI]

IN

IN accumulator.port
Input byte or word

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
accumulator, immed8	10	1	2	IN AL, 0FFEAH
accumulator, DX	8	1	1	IN AX, DX

INC

INC destination
Increment by 1

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	

Operands	Clocks	Transfers*	Bytes	Coding Example
reg16	2	—	1	[illegible]X
reg8	3	—	2	INC BL
memory	15 + EA	2	2-4	INC ALPHA DI BX

INT

INT interrupt-type
Interrupt

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
immed8 (type = 3)	52	5	1	INT 3
immed8 (type ≠ 3)	51	5	2	INT 67

INTR†

INTR (external maskable interrupt)
Interrupt if INTR and IF=1

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	61	7	N/A	N/A

INTO

INTO (no operands)
Interrupt if overflow

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	53 or 4	5	1	INTO

IRET	IRET (no operands) Interrupt Return			Flags O D I T S Z A P C R R R R R R R R R
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	24	3	1	IRET

JA/JNBE	JA/JNBE short-label Jump if above/Jump if not below nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JA ABOVE

JAE/JNB	JAE/JNB short-label Jump if above or equal/Jump if not below			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JAE ABOVE EQUAL

JB/JNAE	JB/JNAE short-label Jump if below/Jump if not above nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JB BELOW

JBE/JNA	JBE/JNA short-label Jump if below or equal/Jump if not above			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNA NOT ABOVE

JC	JC short-label Jump if carry			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JC CARRY SET

JCXZ	JCXZ short-label Jump if CX is zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	18 or 6	—	2	JCXZ COUNT DONE

JE/JZ	JE/JZ short-label Jump if equal/Jump if zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JZ ZERO

JG/JNLE

JG/JNLE short-label
Jump if greater/Jump if not less nor equal

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JG GREATER

JGE/JNL

JGE/JNL short-label
Jump if greater or equal/Jump if not less

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JGE GREATER_EQUAL

JL/JNGE

JL/JNGE short-label
Jump if less/Jump if not greater nor equal

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JL LESS

JLE/JNG

JLE/JNG short-label
Jump if less or equal/Jump if not greater

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNG NOT_GREATER

JMP

JMP target
Jump

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	15	—	2	JMP SHORT
near-label	15	—	3	JMP WITHIN_SEGMENT
far-label	15	—	5	JMP FAR_LABEL
memptr16	18 + EA	1	2-4	JMP [BX] TARGET
regptr16	11	—	2	JMP CX
memptr32	24 + EA	2	2-4	JMP OTHER SEG [SI]

JNC

JNC short-label
Jump if not carry

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNC NOT_CARRY

JNE/JNZ

JNE/JNZ short-label
Jump if not equal/Jump if not zero

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNE NOT_EQUAL

JNO	JNO short-label Jump if not overflow			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNO NO_OVERFLOW

JNP/JPO	JNP/JPO short-label Jump if not parity/Jump if parity odd			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JPO ODD_PARITY

JNS	JNS short-label Jump if not sign			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNS POSITIVE

JO	JO short-label Jump if overflow			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JO SIGNED_OVRFLW

JP/JPE	JP/JPE short-label Jump if parity/Jump if parity even			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JPE EVEN_PARITY

JS	JS short-label Jump if sign			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JS NEGATIVE

LAHF	LAHF (no operands) Load AH from flags			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	LAHF

LDS	LDS destination,source Load pointer using DS			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers**	**Bytes**	**Coding Example**
reg16, mem32	16 + EA	2	2-4	LDS SI,DATA_SEG [DI]

LEA	LEA destination,source Load effective address		Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem16	2 + EA	—	2-4	LEA BX, [BP][DI]

LES	LES destination,source Load pointer using ES		Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem32	16 + EA	2	2-4	LES DI, [BX].TEXT BUFF

LOCK	LOCK (no operands) Lock bus		Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	LOCK XCHG FLAG,AL

LODS	LODS source-string Load string		Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-string	12	1	1	LODS CUSTOMER_NAME
(repeat) source-string	9 + 13/rep	1/rep	1	REP LODS NAME

LOOP	LOOP short-label Loop		Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	17/5	—	2	LOOP AGAIN

LOOPE/LOOPZ	LOOPE/LOOPZ short-label Loop if equal/Loop if zero		Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	18 or 6	—	2	LOOPE AGAIN

LOOPNE/LOOPNZ	LOOPNE/LOOPNZ short-label Loop if not equal/Loop if not zero		Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	19 or 5	—	2	LOOPNE AGAIN

NMI†	NMI (external nonmaskable interrupt) Interrupt if NMI = 1		Flags O S I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	50	5	N/A	N/A

MOV	MOV destination,source Move			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
memory, accumulator	10	1	3	MOV ARRAY [SI], AL
accumulator, memory	10	1	3	MOV AX, TEMP_RESULT
register, register	2	—	2	MOV AX,CX
register, memory	8 + EA	1	2-4	MOV BP, STACK_TOP
memory, register	9 + EA	1	2-4	MOV COUNT [DI], CX
register, immediate	4	—	2-3	MOV CL, 2
memory, immediate	10 + EA	1	3-6	MOV MASK [BX] [SI], 2CH
seg-reg, reg16	2	—	2	MOV ES, CX
seg-reg, mem16	8 + EA	1	2-4	MOV DS, SEGMENT_BASE
reg16, seg-reg	2	—	2	MOV BP, SS.
memory, seg-reg	9 + EA	1	2-4	MOV [BX].SEG_SAVE, CS

MOVS	MOVS dest-string,source-string Move string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, source-string	18	2	1	MOVS LINE EDIT_DATA
(repeat) dest-string, source-string	9 + 17/rep	2/rep	1	REP MOVS SCREEN, BUFFER

MOVSB/MOVSW	MOVSB/MOVSW (no operands) Move string (byte/word)			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	18	2	1	MOVSB
(repeat) (no operands)	9 + 17/rep	2/rep	1	REP MOVSW

MUL	MUL source Multiplication, unsigned			Flags O D I T S Z A P C X U U U U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	70-77	—	2	MUL BL
reg16	118-133	—	2	MUL CX
mem8	(76-83) + EA	1	2-4	MUL MONTH [SI]
mem16	(124-139) + EA	1	2-4	MUL BAUD_RATE

NEG	NEG destination Negate			Flags O D I T S Z A P C X X X X X 1*
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	3	—	2	NEG AL
memory	16 + EA	2	2-4	NEG MULTIPLIER

*0 if destination = 0

NOP

NOP (no operands)
No Operation

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	3	—	1	NOP

NOT

NOT destination
Logical not

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
register	3	—	2	NOT AX
memory	16+EA	2	2-4	NOT CHARACTER

OR

OR destination,source
Logical inclusive or

Flags	O	D	I	T	S	Z	A	P	C
	0				X	X	U	X	0

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	OR AL, BL
register, memory	9+EA	1	2-4	OR DX, PORT ID [DI]
memory, register	16+EA	2	2-4	OR FLAG BYTE, CL
accumulator, immediate	4	—	2-3	OR AL, 01101100B
register, immediate	4	—	3-4	OR CX,01H
memory, immediate	17+EA	2	3-6	OR [BX] CMD WORD,0CFH

OUT

OUT port,accumulator
Output byte or word

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
immed8, accumulator	10	1	2	OUT 44, AX
DX, accumulator	8	1	1	OUT DX, AL

POP

POP destination
Pop word off stack

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
register	8	1	1	POP DX
seg-reg (CS illegal)	8	1	1	POP DS
memory	17+EA	2	2-4	POP PARAMETER

POPF

POPF (no operands)
Pop flags off stack

Flags	O	D	I	T	S	Z	A	P	C
	R	R	R	R	R	R	R	R	R

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	8	1	1	POPF

PUSH	PUSH source Push word onto stack			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	11	1	1	PUSH SI
seg-reg (CS legal)	10	1	1	PUSH ES
memory	16 + EA	2	2-4	PUSH RETURN_CODE [SI]

PUSHF	PUSHF (no operands) Push flags onto stack			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	10	1	1	PUSHF

RCL	RCL destination,count Rotate left through carry			Flags O D I T S Z A P C X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, 1	2	—	2	RCL CX, 1
register, CL	8 + 4/bit	—	2	RCL AL, CL
memory, 1	15 + EA	2	2-4	RCL ALPHA, 1
memory, CL	20 + EA + 4/bit	2	2-4	RCL [BP].PARM, CL

RCR	RCR designation,count Rotate right through carry			Flags O D I T S Z A P C X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, 1	2	—	2	RCR BX, 1
register, CL	8 + 4/bit	—	2	RCR BL, CL
memory, 1	15 + EA	2	2-4	RCR [BX].STATUS, 1
memory, CL	20 + EA + 4/bit	2	2-4	RCR ARRAY [DI], CL

REP	REP (no operands) Repeat string operation			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	REP MOVS DEST, SRCE

REPE/REPZ	REPE/REPZ (no operands) Repeat string operation while equal/while zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	REPE CMPS DATA, KEY

REPNE/REPNZ

REPNE/REPNZ (no operands)
Repeat string operation while not equal/not zero

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REPNE SCAS INPUT LINE

RET

RET optional-pop-value
Return from procedure

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(intra-segment, no pop)	8	1	1	RET
(intra-segment, pop)	12	1	3	RET 4
(inter-segment, no pop)	18	2	1	RET
(inter-segment, pop)	17	2	3	RET 2

ROL

ROL destination,count
Rotate left

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers	Bytes	Coding Examples
register, 1	2	—	2	ROL BX, 1
register, CL	8 + 4/bit	—	2	ROL DI, CL
memory, 1	15 + EA	2	2-4	ROL FLAG BYTE [DI],1
memory, CL	20 + EA + 4/bit	2	2-4	ROL ALPHA, CL

ROR

ROR destination,count
Rotate right

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operand	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	ROR AL, 1
register, CL	8 + 4/bit	—	2	ROR BX, CL
memory, 1	15 + EA	2	2-4	ROR PORT STATUS, 1
memory, CL	20 + EA + 4/bit	2	2-4	ROR CMD WORD, CL

SAHF

SAHF (no operands)
Store AH into flags

Flags	O	D	I	T	S	Z	A	P	C
					R	R	R	R	R

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	SAHF

SAL/SHL

SAL/SHL destination,count
Shift arithmetic left/Shift logical left

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers*	Bytes	Coding Examples
register,1	2	—	2	SAL AL,1
register, CL	8 + 4/bit	—	2	SHL DI, CL
memory,1	15 + EA	2	2-4	SHL [BX] OVERDRAW, 1
memory, CL	20 + EA + 4/bit	2	2-4	SAL STORE COUNT, CL

SAR

SAR destination,source
Shift arithmetic right

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	U	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	SAR DX, 1
register, CL	8 + 4/bit	—	2	SAR DI, CL
memory, 1	15 + EA	2	2-4	SAR N BLOCKS, 1
memory, CL	20 + EA + 4/bit	2	2-4	SAR N BLOCKS, CL

SBB

SBB destination,source
Subtract with borrow

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	SBB BX, CX
register, memory	9 + EA	1	2-4	SBB DI, [BX] PAYMENT
memory, register	16 + EA	2	2-4	SBB BALANCE, AX
accumulator, immediate	4	—	2-3	SBB AX, 2
register, immediate	4	—	3-4	SBB CL, 1
memory, immediate	17 + EA	2	3-6	SBB COUNT [SI], 10

SCAS

SCAS dest-string
Scan string

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string	15	1	1	SCAS INPUT LINE
(repeat) dest-string	9 + 15/rep	1/rep	1	REPNE SCAS BUFFER

SEGMENT†

SEGMENT override prefix
Override to specified segment

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	MOV SS:PARAMETER, AX

SHR

SHR destination,count
Shift logical right

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	SHR SI, 1
register, CL	8 + 4/bit	—	2	SHR SI, CL
memory, 1	15 + EA	2	2-4	SHR ID BYTE [SI][BX], 1
memory, CL	20 + EA + 4/bit	2	2-4	SHR INPUT WORD, CL

SINGLE STEP†

SINGLE STEP (Trap flag interrupt)
Interrupt if TF = 1

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	50	5	N/A	N/A

* For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

STC

STC (no operands)
Set carry flag

Flags	O	D	I	T	S	Z	A	P	C
									1

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STC

STD

STD (no operands)
Set direction flag

Flags	O	D	I	T	S	Z	A	P	C
		1							

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STD

STI

STI (no operands)
Set interrupt enable flag

Flags	O	D	I	T	S	Z	A	P	C
			1						

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STI

STOS

STOS dest-string
Store byte or word string

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string	11	1	1	STOS PRINT_LINE
(repeat) dest-string	9 + 10/rep	1/rep	1	REP STOS DISPLAY

SUB

SUB destination,source
Subtraction

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	SUB CX, BX
register, memory	9 + EA	1	2-4	SUB DX, MATH_TOTAL [SI]
memory, register	16 + EA	2	2-4	SUB [BP + 2], CL
accumulator, immediate	4	—	2-3	SUB AL, 10
register, immediate	4	—	3-4	SUB SI, 5280
memory, immediate	17 + EA	2	3-6	SUB [BP].BALANCE, 1000

TEST

TEST destination,source
Test or non-destructive logical and

Flags	O	D	I	T	S	Z	A	P	C
	0				X	X	U	X	0

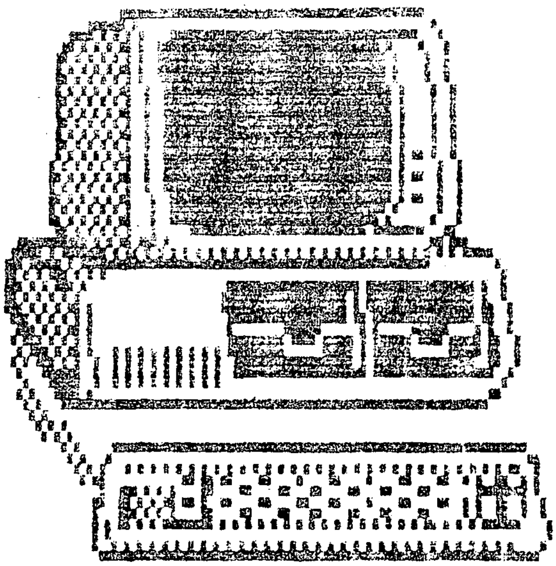
Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	TEST SI, DI
register, memory	9 + EA	1	2-4	TEST SI, END_COUNT
accumulator, immediate	4	—	2-3	TEST AL, 00100000B
register, immediate	5	—	3-4	TEST BX, 0CC4H
memory, immediate	11 + EA	—	3-6	TEST RETURN_CODE, 01H

WAIT	WAIT (no operands) Wait while $\overline{TEST}$ pin not asserted			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	3 + 5n	—	1	WAIT

XCHG	XCHG destination,source Exchange			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
accumulator, reg16	3	—	1	XCHG AX, BX
memory, register	17 + EA	2	2-4	XCHG SEMAPHORE, AX
register, register	4	—	2	XCHG AL, BL

XLAT	XLAT source-table Translate			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-table	11	1	1	XLAT ASCII_TAB

XOR	XOR destination,source Logical exclusive or			Flags O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	XOR CX, BX
register, memory	9 + EA	1	2-4	XOR CL, MASK_BYTE
memory, register	16 + EA	2	2-4	XOR ALPHA [SI], DX
accumulator, immediate	4	—	2-3	XOR AL, 01000010B
register, immediate	4	—	3-4	XOR SI, 00C2H
memory, immediate	17 + EA	2	3-6	XOR RETURN_CODE, 0D2H



APPENDIX B

programming
8250
UART

IBM Asynchronous Communications Adapter

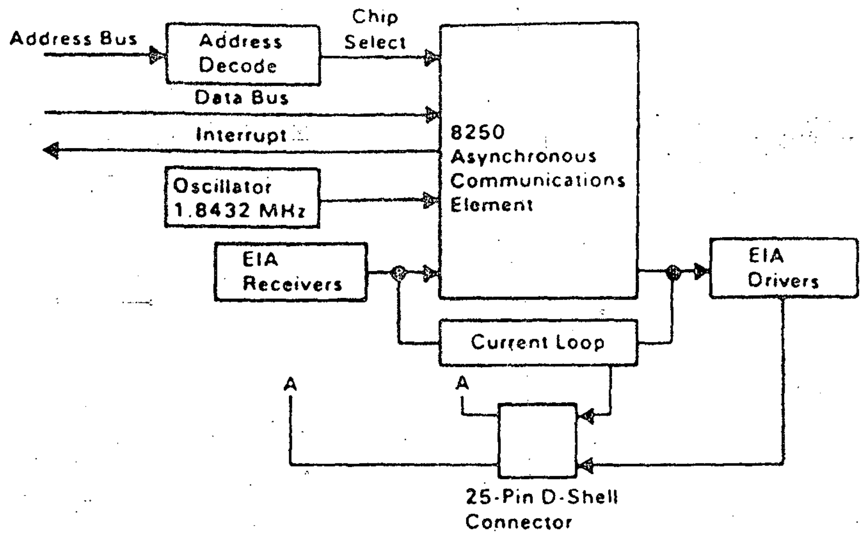
The asynchronous communications adapter system control signals and voltage requirements are provided through a 2 by 31 position card edge tab. Two jumper modules are provided on the adapter. One jumper module selects either RS-232C or current-loop operation. The other jumper module selects one of two addresses for the adapter, so two adapters may be used in one system.

The adapter is fully programmable and supports asynchronous communciations only. It will add and remove start bits, stop bits, and parity bits. A programmable baud rate generator allows operation from 50 baud to 9600 baud. Five, six, seven or eight bit characters with 1, 1-1/2, or 2 stop bits are supported. A fully prioritized interrupt system controls transmit, receive, error, line status, and data set interrupts. Diagnostic capabilities provide loopback functions of transmit/receive and input/output signals.

The heart of the adapter is a INS8250 LSI chip or functional equivalent. Features in addition to those listed above are:

- Full double buffering eliminates need for precise synchronization.
- Independent receiver clock input.
- Modem control functions: clear to send (CTS), request to send (RTS), data set ready (DSR), data terminal ready (DTR), ring indicator (RI), and carrier detect.
- False-start bit detection.
- Line-break generation and detection.

All communications protocol is a function of the system microcode and must be loaded before the adapter is operational. All pacing of the interface and control signal status must be handled by the system software. The figure below is a block diagram of the asynchronous communications adapter.



Asynchronous Communications Adapter Block Diagram

Modes of Operation

The different modes of operation are selected by programming the 8250 asynchronous communications element. This is done by selecting the I/O address (hex 3F8 to 3FF primary, and hex 2F8 to 2FF secondary) and writing data out to the card. Address bits A0, A1, and A2 select the different registers that define the modes of operation. Also, the divisor latch access bit (bit 7) of the line control register is used to select certain registers.

I/O Decode (in Hex)		Register Selected	DLAB State
Primary Adapter	Alternate Adapter		
3F8	2F8	TX Buffer	DLAB=0 (Write)
3F8	2F8	RX Buffer	DLAB=0 (Read)
3F8	2F8	Divisor Latch LSB	DLAB=1
3F9	2F9	Divisor Latch MSB	DLAB=1
3F9	2F9	Interrupt Enable Register	
3FA	3FA	Interrupt Identification Registers	
3FB	2FB	Line Control Register	
3FC	2FC	Modem Control Register	
3FD	2FD	Line Status Register	
3FE	2FE	Modem Status Register	

I/O Decodes

Hex Address 3F8 to 3FF and 2F8 to 2FF											
A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	DLAB	Register
1	1/0	1	1	1	1	1	x	x	x		
							0	0	0	0	Receive Buffer (read), Transmit Holding Reg. (write)
							0	0	1	0	Interrupt Enable
							0	1	0	x	Interrupt Identification
							0	1	1	x	Line Control
							1	0	0	x	Modem Control
							1	0	1	x	Line Status
							1	1	0	x	Modem Status
							1	1	1	x	None
							0	0	0	1	Divisor Latch (LSB)
							0	0	1	1	Divisor Latch (MSB)

Note: Bit 8 will be logical 1 for the adapter designated as primary or a logical 0 for the adapter designated as alternate (as defined by the address jumper module on the adapter).

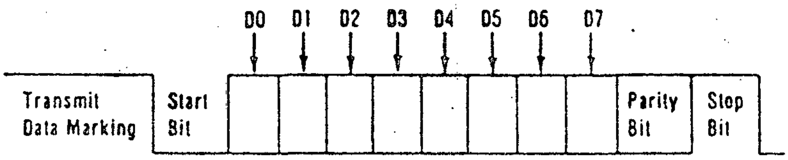
A2, A1 and A0 bits are "don't cares" and are used to select the different register of the communications chip.

Address Bits

Interrupts

One interrupt line is provided to the system. This interrupt is IRQ4 for a primary adapter or IRQ3 for an alternate adapter, and is positive active. To allow the communications card to send interrupts to the system, bit 3 of the modem control register must be set to 1 (high). At this point, any interrupts allowed by the interrupt enable register will cause an interrupt.

The data format will be as follows:



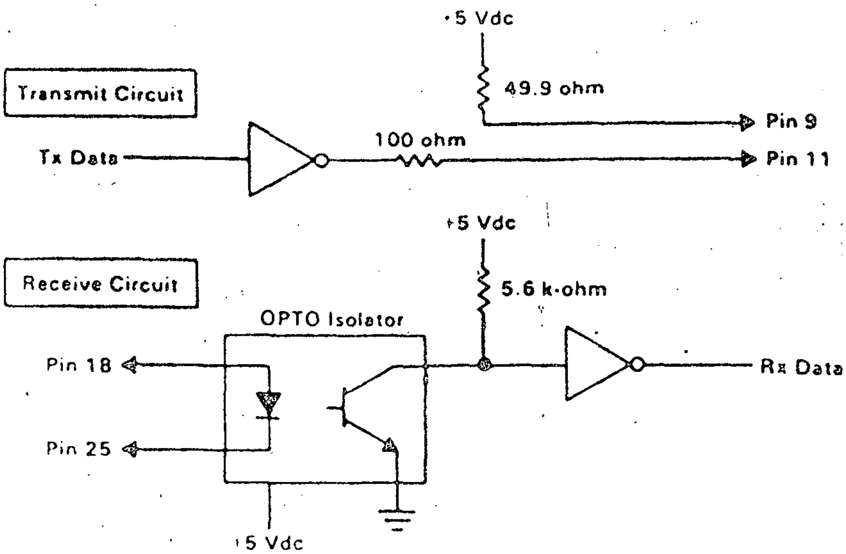
Data bit 0 is the first bit to be transmitted or received. The adapter automatically inserts the start bit, the correct parity bit if programmed to do so, and the stop bit (1, 1-1/2, or 2 depending on the command in the line-control register).

Interface Description

The communications adapter provides an EIA RS-232C-like interface. One 25-pin D-shell, male type connector is provided to attach various peripheral devices. In addition, a current loop interface is also located in this same connector. A jumper block is provided to manually select either the voltage interface, or the current loop interface.

The current loop interface is provided to attach certain printers provided by IBM that use this particular type of interface.

Pin 18 + receive current loop data
Pin 25 - receive current loop return
Pin 9 + transmit current loop return
Pin 11 - transmit current loop data



Current Loop Interface

The voltage interface is a serial interface. It supports certain data and control signals, as listed below.

Pin 2	Transmitted Data
Pin 3	Received Data
Pin 4	Request to Send
Pin 5	Clear to Send
Pin 6	Data Set Ready
Pin 7	Signal Ground
Pin 8	Carrier Detect
Pin 20	Data Terminal Ready
Pin 22	Ring Indicator

The adapter converts these signals to/from TTL levels to EIA voltage levels. These signals are sampled or generated by the communications control chip. These signals can then be sensed by the system software to determine the state of the interface or peripheral device.

Voltage Interchange Information

Interchange Voltage	Binary State	Signal Condition	Interface Control Function
Positive Voltage =	Binary (0)	= Spacing	= On
Negative Voltage =	Binary (1)	= Marking	= Off

Invalid Levels

+15 Vdc ------------------------------

On Function

+3 Vdc ------------------------------

0 Vdc — Invalid Levels

-3 Vdc ------------------------------

Off Function

-15 Vdc ------------------------------

Invalid Levels

The signal will be considered in the "marking" condition when the voltage on the interchange circuit, measured at the interface point, is more negative than -3 Vdc with respect to signal ground. The signal will be considered in the "spacing" condition when the voltage is more positive than +3 Vdc with respect to signal ground. The region between +3 Vdc and -3 Vdc is defined as the transition region, and considered an invalid level. The voltage that is more negative than -15 Vdc or more positive than +15 Vdc will also be considered an invalid level.

During the transmission of data, the "marking" condition will be used to denote the binary state "1" and "spacing" condition will be used to denote the binary state "0."

For interface control circuits, the function is "on" when the voltage is more positive than +3 Vdc with respect to signal ground and is "off" when the voltage is more negative than -3 Vdc with respect to signal ground.

INS8250 Functional Pin Description

The following describes the function of all INS8250 input/output pins. Some of these descriptions reference internal circuits.

Note: In the following descriptions, a low represents a logical 0 (0 Vdc nominal) and a high represents a logical 1 (+2.4 Vdc nominal).

Input Signals

Chip Select (CS0, CS1, $\overline{\text{CS2}}$), Pins 12-14: When CS0 and CS1 are high and $\overline{\text{CS2}}$ is low, the chip is selected. Chip selection is complete when the decoded chip select signal is latched with an active (low) address strobe ($\overline{\text{ADS}}$) input. This enables communications between the INS8250 and the processor.

Data Input Strobe (DISTR, $\overline{\text{DISTR}}$) Pins 22 and 21: When DISTR is high or $\overline{\text{DISTR}}$ is low while the chip is selected, allows the processor to read status information or data from a selected register of the INS8250.

Note: Only an active DISTR or $\overline{\text{DISTR}}$ input is required to transfer data from the INS8250 during a read operation. Therefore, tie either the DISTR input permanently low or the $\overline{\text{DISTR}}$ input permanently high, if not used.

Data Output Strobe (DOSTR, $\overline{\text{DOSTR}}$), Pins 19 and 18: When DOSTR is high or $\overline{\text{DOSTR}}$ is low while the chip is selected, allows the processor to write data or control words into a selected register of the INS8250.

Note: Only an active DOSTR or $\overline{\text{DOSTR}}$ input is required to transfer data to the INS8250 during a write operation. Therefore, tie either the DOSTR input permanently low or the $\overline{\text{DOSTR}}$ input permanently high, if not used.

Address Strobe ($\overline{\text{ADS}}$), Pin 25: When low, provides latching for the register select (A0, A1, A2) and chip select (CS0, CS1, $\overline{\text{CS2}}$) signals.

Note: An active $\overline{ADS}$ input is required when the register select (A0, A1, A2) signals are not stable for the duration of a read or write operation. If not required, tie the $\overline{ADS}$ input permanently low.

Register Select (A0, A1, A2), Pins 26-28: These three inputs are used during a read or write operation to select an INS8250 register to read or write to as indicated in the table below. Note that the state of the divisor latch access bit (DLAB), which is the most significant bit of the line control register, effects the selection of certain INS8250 registers. The DLAB must be set high by the system software to access the baud generator divisor latches.'

DLAB	A2	A1	A0	Register
0	0	0	0	Receiver Buffer (Read), Transmitter Holding Register (Write)
0	0	0	1	Interrupt Enable
X	0	1	0	Interrupt Identification (Read Only)
X	0	1	1	Line Control
X	1	0	0	Modem Control
X	1	0	1	Line Status
X	1	1	0	Modem Status
X	1	1	1	None
1	0	0	0	Divisor Latch (Least Significant Bit)
1	0	0	1	Divisor Latch (Most Significant Bit)

Master Reset (MR), Pin 35: When high, clears all the registers (except the receiver buffer, transmitter holding, and divisor latches), and the control logic of the INS8250. Also, the state of various output signals (SOUT, INTRPT, $\overline{OUT\ 1}$, $\overline{OUT\ 2}$, $\overline{RTS}$, $\overline{DTR}$) are affected by an active MR input. Refer to the "Asynchronous Communications Reset Functions" table.

Receiver Clock (RCLK), Pin 9: This input is the 16 x baud rate clock for the receiver section of the chip.

Serial Input (SIN), Pin 10: Serial data input from the communications link (peripheral device, modem, or data set).

Clear to Send ($\overline{\text{CTS}}$), Pin 36: The $\overline{\text{CTS}}$ signal is a modem control function input whose condition can be tested by the processor by reading bit 4 (CTS) of the modem status register. Bit 0 (DCTS) of the modem status register indicates whether the CTS input has changed state since the previous reading of the modem status register.

Note: Whenever the CTS bit of the modem status register changes state, an interrupt is generated if the modem status interrupt is enabled.

Data Set Ready ($\overline{\text{DSR}}$), Pin 37: When low, indicates that the modem or data set is ready to establish the communications link and transfer data with the INS8250. The $\overline{\text{DSR}}$ signal is a modem-control function input whose condition can be tested by the processor by reading bit 5 (DSR) of the modem status register. Bit 1 (DDSR) of the modem status register indicates whether the $\overline{\text{DSR}}$ input has changed since the previous reading of the modem status register.

Note: Whenever the DSR bit of the modem status register changes state, an interrupt is generated if the modem status interrupt is enabled.

Received Line Signal Detect ($\overline{\text{RLSD}}$), Pin 38: When low, indicates that the data carrier had been detected by the modem or data set. The $\overline{\text{RLSD}}$ signal is a modem-control function input whose condition can be tested by the processor by reading bit 7 (RLSD) of the modem status register. Bit 3 (DRLSD) of the modem status register indicates whether the $\overline{\text{RLSD}}$ input has changed state since the previous reading of the modem status register.

Note: Whenever the RLSD bit of the modem status register changes state, an interrupt is generated if the modem status interrupt is enabled.

Ring Indicator ($\overline{RI}$), Pin 39: When low, indicates that a telephone ringing signal has been received by the modem or data set. The $\overline{RI}$ signal is a modem-control function input whose condition can be tested by the processor by reading bit 6 (RI) of the modem status register. Bit 2 (TERI) of the modem status register indicates whether the $\overline{RI}$ input has changed from a low to high state since the previous reading of the modem status register.

Note: Whenever the RI bit of the modem status register changes from a high to a low state, an interrupt is generated if the modem status interrupt is enabled.

VCC, Pin 40: +5 Vdc supply.

VSS, Pin 20: Ground (0 Vdc) reference.

Output Signals

Data Terminal Ready ($\overline{DTR}$), Pin 33: When low, informs the modem or data set that the INS8250 is ready to communicate. The DTR output signal can be set to an active low by programming bit 0 (DTR) of the modem control register to a high level. The $\overline{DTR}$ signal is set high upon a master reset operation

Request to Send ($\overline{RTS}$), Pin 32: When low, informs the modem or data set that the INS8250 is ready to transmit data. The $\overline{RTS}$ output signal can be set to an active low by programming bit 1 (RTS) of the modem control register. The $\overline{RTS}$ signal is set high upon a master reset operation.

Output 1 ($\overline{OUT\ 1}$), Pin 34: User-designated output that can be set to an active low by programming bit 2 ($\overline{OUT\ 1}$) of the modem control register to a high level. The $\overline{OUT\ 1}$ signal is set high upon a master reset operation.

Output 2 ($\overline{OUT\ 2}$), Pin 31: User-designated output that can be set to an active low by programming bit 3 ($\overline{OUT\ 2}$) of the modem control register to a high level. The $\overline{OUT\ 2}$ signal is set high upon a master reset operation.

Chip Select Out (CSOUT), Pin 24: When high, indicates that the chip has been selected by active CS0, CS1, and $\overline{\text{CS2}}$ inputs. No data transfer can be initiated until the CSOUT signal is a logical 1.

Driver Disable (DDIS), Pin 23: Goes low whenever the processor is reading data from the INS8250. A high-level DDIS output can be used to disable an external transceiver (if used between the processor and INS8250 on the D7-D0 data bus) at all times, except when the processor is reading data.

Baud Out ($\overline{\text{BAUDOUT}}$), Pin 15: 16 x clock signal for the transmitter section of the INS8250. The clock rate is equal to the main reference oscillator frequency divided by the specified divisor in the baud generator divisor latches. The $\overline{\text{BAUDOUT}}$ may also be used for the receiver section by typing this output to the RCLK input of the chip.

Interrupt (INTRPT), Pin 30: Goes high whenever any one of the following interrupt types has an active high condition and is enabled through the IER: receiver error flag, received data available, transmitter holding register empty, or modem status. The INTRPT signal is reset low upon the appropriate interrupt service or a master reset operation.

Serial Output (SOUT), Pin 11: Composite serial data output to the communications link (peripheral, modem or data set). The SOUT signal is set to the marking (logical 1) state upon a master reset operation.

Input/Output Signals

Data (D7-D0) Bus, Pins 1-8: This bus comprises eight tri-state input/output lines. The bus provides bidirectional communications between the INS8250 and the processor. Data, control words, and status information are transferred through the D7-D0 Data bus.

External Clock Input/Output (XTAL1, XTAL2), Pins 16 and 17: These two pins connect the main timing reference (crystal or signal clock) to the INS8250.

Programming Considerations

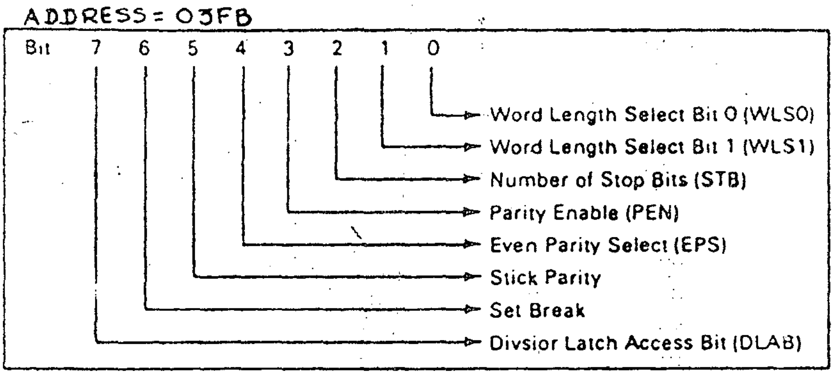
The INS8250 has a number of accessible registers. The system programmer may access or control any of the INS8250 registers through the processor. These registers are used to control INS8250 operations and to transmit and receive data. A table listing and description of the accessible registers follows.

Register/Signal	Reset Control	Reset State
Interrupt Enable Register	Master Reset	All Bits Low (0-3 Forced and 4-7 Permanent)
Interrupt Identification Register	Master Reset	Bit 0 is High, Bits 1 and 2 Low Bits 3-7 are Permanently Low
Line Control Register	Master Reset	All Bits Low
Modem Control Register	Master Reset	All Bits Low
Line Status Register	Master Reset	Except Bits 5 and 6 are High
Modem Status Register	Master Reset	Bits 0-3 Low Bits 4-7 - Input Signal
SOUT	Master Reset	High
INTRPT (RCVR Errors)	Read LSR/MR	Low
INTRPT (RCVR Data Ready)	Read RBR/MR	Low
INTRPT (RCVR Data Ready)	Read IIR/Write THR/MR	Low
INTRPT (Modem Status Changes)	Read MSR/MR	Low
OUT 2	Master Reset	High
RTS	Master Reset	High
DTR	Master Reset	High
OUT 1	Master Reset	High

Asychronous Communications Reset Functions

Line-Control Register

The system programmer specifies the format of the asynchronous data communications exchange through the line-control register. In addition to controlling the format, the programmer may retrieve the contents of the line-control register for inspection. This feature simplifies system programming and eliminates the need for separate storage in system memory of the line characteristics. The contents of the line-control register are indicated and described below.



Line-Control Register (LCR)

Bits 0 and 1: These two bits specify the number of bits in each transmitted or received serial character. The encoding of bits 0 and 1 is as follows:

Bit 1	Bit 0	Word Length
0	0	5 Bits
0	1	6 Bits
1	0	7 Bits
1	1	8 Bits

Bit 2: This bit specifies the number of stop bits in each transmitted or received serial character. If bit 2 is a logical 0, one stop bit is generated or checked in the transmit or receive data, respectively. If bit 2 is logical 1 when a 5-bit word length is selected through bits 0 and 1, 1-1/2 stop bits are generated or checked. If bit 2 is logical 1 when either a 6-, 7-, or 8-bit word length is selected, two stop bits are generated or checked.

Bit 3: This bit is the parity enable bit. When bit 3 is a logical 1, a parity bit is generated (transmit data) or checked (receive data) between the last data word bit and stop bit of the serial data. (The parity bit is used to produce an even or odd number of 1's when the data word bits and the parity bit are summed.)

Bit 4: This bit is the even parity select bit. When bit 3 is a logical 1 and bit 4 is a logical 0, an odd number of logical 1's is transmitted or checked in the data word bits and parity bit. When bit 3 is a logical 1 and bit 4 is a logical 1, an even number of bits is transmitted or checked.

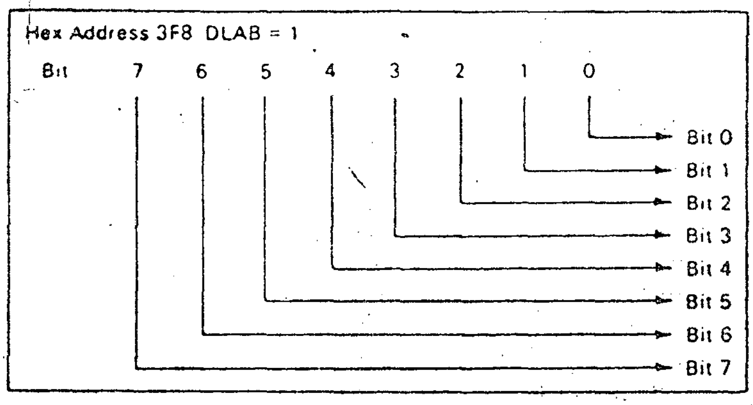
Bit 5: This bit is the stick parity bit. When bit 3 is a logical 1 and bit 5 is a logical 1, the parity bit is transmitted and then detected by the receiver as a logical 0 if bit 4 is a logical 1, or as a logical 1 if bit 4 is a logical 0.

Bit 6: This bit is the set break control bit. When bit 6 is a logical 1, the serial output (SOUT) is forced to the spacing (logical 0) state and remains there regardless of other transmitter activity. The set break is disabled by setting bit 6 to a logical 0. This feature enables the processor to alert a terminal in a computer communications system.

Bit 7: This bit is the divisor latch access bit (DLAB). It must be set high (logical 1) to access the divisor latches of the baud rate generator during a read or write operation. It must be set low (logical 0) to access the receiver buffer, the transmitter holding register, or the interrupt enable register.

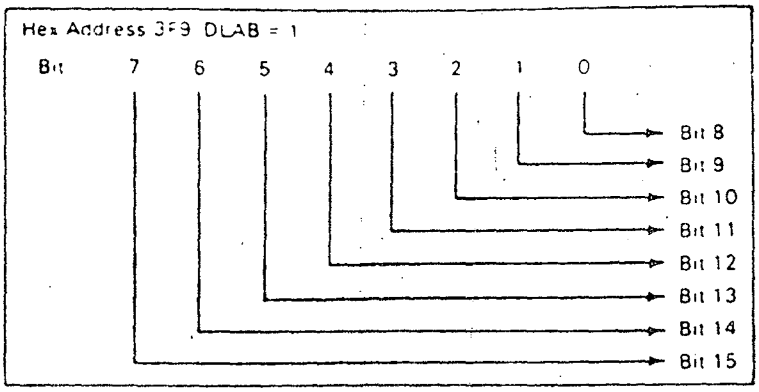
Programmable Baud Rate Generator

The INS8250 contains a programmable baud rate generator that is capable of taking the clock input (1.8432 MHz) and dividing it by any divisor from 1 to (2^{16}-1). The output frequency of the baud generator is 16 x the baud rate [divisor # = (frequency input)/(baud rate x 16)]. Two 8-bit latches store the divisor in a 16-bit binary format. These divisor latches must be loaded during initialization in order to ensure desired operation of the baud rate generator. Upon loading either of the divisor latches, a 16-bit baud counter is immediately loaded. This prevents long counts on initial load.



Divisor Latch Least Significant Bit (DLL)

Byte



Divisor Latch Most Significant Bit (DLM)

The following figure illustrates the use of the baud rate generator with a frequency of 1.8432 MHz. For baud rates of 9600 and below, the error obtained is minimal.

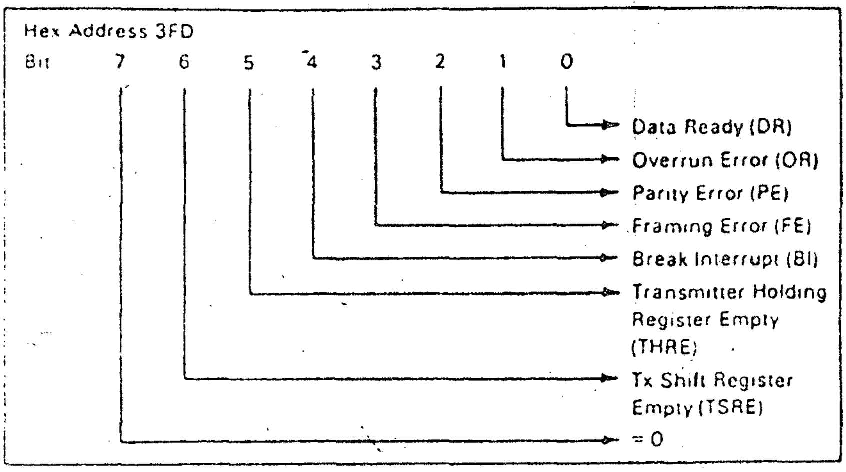
Note: The maximum operating frequency of the baud generator is 3.1 MHz. In no case should the data rate be greater than 9600 baud.

Desired Baud Rate	Divisor Used to Generate 16x Clock (Decimal)	(Hex)	Percent Error Difference Between Desired and Actual
50	2304	900	—
75	1536	600	—
110	1047	417	0.026
134.5	857	359	0.058
150	768	300	—
300	384	180	—
600	192	0C0	—
1200	96	060	—
1800	64	040	—
2000	58	03A	0.69
2400	48	030	—
3600	32	020	—
4800	24	018	—
7200	16	010	—
9600	12	00C	—

Baud Rate at 1.843 MHz

Line Status Register

This 8-bit register provides status information on the processor concerning the data transfer. The contents of the line status register are indicated and described below:



Line Status Register (LSR)

Bit 0: This bit is the receiver data ready (DR) indicator. Bit 0 is set to a logical 1 whenever a complete incoming character has been received and transferred into the receiver buffer register. Bit 0 may be reset to a logical 0 either by the processor reading the data in the receiver buffer register or by writing a logical 0 into it from the processor.

Bit 1: This bit is the overrun error (OE) indicator. Bit 1 indicates that data in the receiver buffer register was not read by the processor before the next character was transferred into the receiver buffer register, thereby destroying the previous character. The OE indicator is reset whenever the processor reads the contents of the line status register.

Bit 2: This bit is the parity error (PE) indicator. Bit 2 indicates that the received data character does not have the correct even or odd parity, as selected by the even parity-select bit. The PE bit is set to a logical 1 upon detection of a parity error and is reset to a logical 0 whenever the processor reads the contents of the line status register.

Bit 3: This bit is the framing error (FE) indicator. Bit 3 indicates that the received character did not have a valid stop bit. Bit 3 is set to a logical 1 whenever the stop bit following the last data bit or parity is detected as a zero bit (spacing level).

Bit 4: This bit is the break interrupt (BI) indicator. Bit 4 is set to a logical 1 whenever the received data input is held in the spacing (logical 0) state for longer than a full word transmission time (that is, the total time of start bit + data bits + parity + stop bits).

Note: Bits 1 through 4 are the error conditions that produce a receiver line status interrupt whenever any of the corresponding conditions are detected.

Bit 5: This bit is the transmitter holding register empty (THRE) indicator. Bit 5 indicates that the INS8250 is ready to accept a new character for transmission. In addition, this bit causes the INS8250 to issue an interrupt to the processor when the transmit holding register empty interrupt enable is set high. The THRE bit is set to a logical 1 when a character is transferred from the transmitter holding register into the transmitter shift register. The bit is reset to logical 0 concurrently with the loading of the transmitter holding register by the processor.

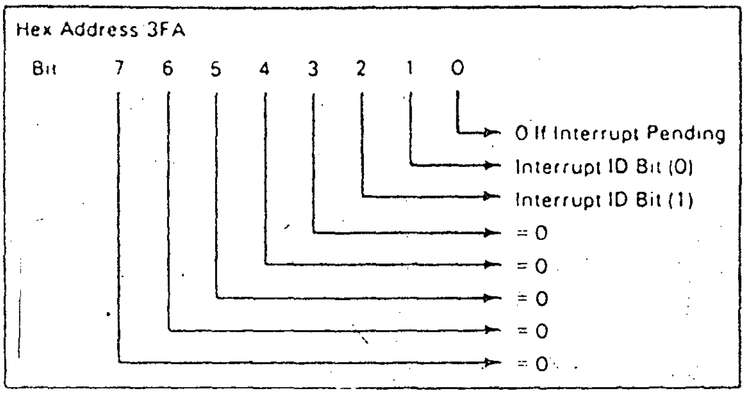
Bit 6: This bit is the transmitter shift register empty (TSRE) indicator. Bit 6 is set to a logical 1 whenever the transmitter shift register is idle. It is reset to logical 0 upon a data transfer from the transmitter holding register to the transmitter shift register. Bit 6 is a read-only bit.

Bit 7: This bit is permanently set to logical 0.

Interrupt Identification Register

The INS8250 has an on-chip interrupt capability that allows for complete flexibility in interfacing to all the popular microprocessors presently available. In order to provide minimum software overhead during data character transfers, the INS8250 prioritizes interrupts into four levels: receiver line status (priority 1), received data ready (priority 2), transmitter holding register empty (priority 3), and modem status (priority 4).

Information indicating that a prioritized interrupt is pending and the type of prioritized interrupt is stored in the interrupt identification register. Refer to the "Interrupt Control Functions" table. The interrupt identification register (IIR), when addressed during chip-select time, freezes the highest priority interrupt pending, and no other interrupts are acknowledged until that particular interrupt is serviced by the processor. The contents of the IIR are indicated and described below.



Interrupt Identification Register (IIR)

Bit 0: This bit can be used in either a hard-wired prioritized or polled environment to indicate whether an interrupt is pending and the IIR contents may be used as a pointer to the appropriate interrupt service routine. When bit 0 is a logical 1, no interrupt is pending and polling (if used) is continued.

Bits 1 and 2: These two bits of the IIR are used to identify the highest priority interrupt pending as indicated in the "Interrupt Control Functions" table.

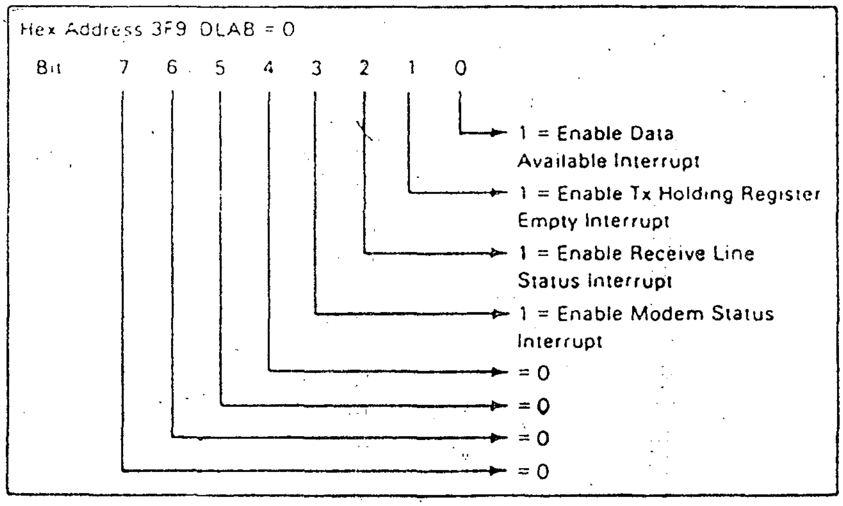
Bits 3 through 7: These five bits of the IIR are always logical 0.

Interrupt ID Register				Interrupt Set and Reset Functions		
Bit 2	Bit 1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	0	1		None	None	
1	1	0	Highest	Receiver Line Status	Overrun Error or Parity Error or Framing Error or Break Interrupt	Reading the Line Status Register
1	0	0	Second	Received Data Available	Receiver Data Available	Reading the Receiver Buffer Register
0	1	0	Third	Transmitter Holding Register Empty	Transmitter Holding Register Empty	Reading the IIR Register (if source of interrupt) or Writing [illegible] the Trans[illegible] Holding Register
0	0	0	Fourth	Modem Status	Clear to Send or Data Set Ready or Ring Indicator or Received Line Signal Direct	Reading the Modem Status Register

Interrupt Control Functions

Interrupt Enable Register

This eight-bit register enables the four types of interrupt of the INS8250 to separately activate the chip interrupt (INTRPT) output signal. It is possible to totally disable the interrupt system by resetting bits 0 through 3 of the interrupt enable register. Similarly, by setting the appropriate bits of this register to a logical 1, selected interrupts can be enabled. Disabling the interrupt system inhibits the interrupt identification register and the active (high) INTRPT output from the chip. All other system functions operate in their normal manner, including the setting of the line status and modem status registers. The contents of the interrupt enable register are indicated and described below:



Interrupt Enable Register (IER)

Bit 0: This bit enables the received data available interrupt when set to logical 1.

Bit 1: This bit enables the transmitter holding register empty interrupt when set to logical 1.

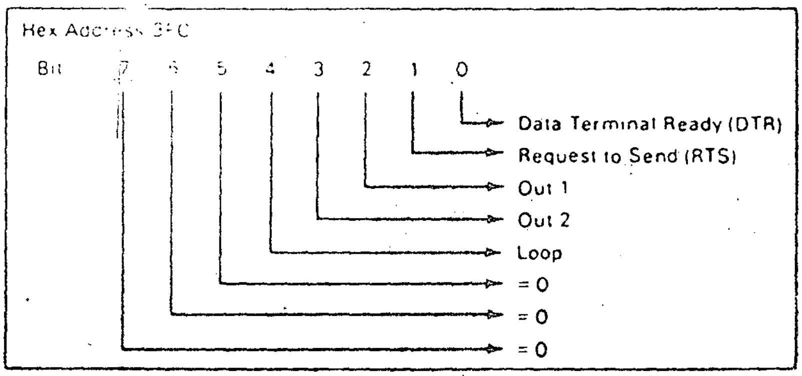
Bit 2: This bit enables the receiver line status interrupt when set to logical 1.

Bit 3: This bit enables the modem status interrupt when set to logical 1.

Bits 4 through 7: These four bits are always logical 0.

Modem Control Register

This eight-bit register controls the interface with the modem or data set (or a peripheral device emulating a modem). The contents of the modem control register are indicated and described below:



Modem Control Register (MCR)

Bit 0: This bit controls the data terminal ready ($\overline{DTR}$) output. When bit 0 is set to a logical 1, the $\overline{DTR}$ output is forced to a logical 0. When bit 0 is reset to a logical 0, the $\overline{DTR}$ output is forced to a logical 1.

Note: The $\overline{DTR}$ output of the INS8250 may be applied to an EIA inverting line driver (such as the DS1488) to obtain the proper polarity input at the succeeding modem or data set.

Bit 1: This bit controls the request to send ($\overline{RTS}$) output. Bit 1 affects the $\overline{RTS}$ output in a manner identical to that described above for bit 0.

Bit 2: This bit controls the output 1 ($\overline{\text{OUT 1}}$) signal, which is an auxiliary user-designated output. Bit 2 affects the $\overline{\text{OUT 1}}$ output in a manner identical to that described above for bit 0.

Bit 3: This bit controls the output 2 ($\overline{\text{OUT 2}}$) signal, which is an auxiliary user-designated output. Bit 3 affects the $\overline{\text{OUT 2}}$ output in a manner identical to that described above for bit 0.

Bit 4: This bit provides a loopback feature for diagnostic testing of the INS8250. When bit 4 is set to logical 1, the following occurs: the transmitter serial output (SOUT) is set to the marking (logical 1) state; the receiver serial input (SIN) is disconnected; the output of the transmitter shift register is "looped back" into the receiver shift register input; the four modem control inputs ($\overline{\text{CTS}}$, $\overline{\text{DSR}}$, $\overline{\text{RLSD}}$, AND $\overline{\text{RI}}$) are disconnected; and the four modem control outputs ($\overline{\text{DTR}}$, $\overline{\text{RTS}}$, $\overline{\text{OUT 1}}$, and $\overline{\text{OUT 2}}$) are internally connected to the four modem control inputs. In the diagnostic mode, data that is transmitted is immediately received. This feature allows the processor to verify the transmit- and receive-data paths of the INS8250.

In the diagnostic mode, the receiver and transmitter interrupts are fully operational. The modem control interrupts are also operational but the interrupts' sources are now the lower four bits of the modem control register instead of the four modem control inputs. The interrupts are still controlled by the interrupt enable register.

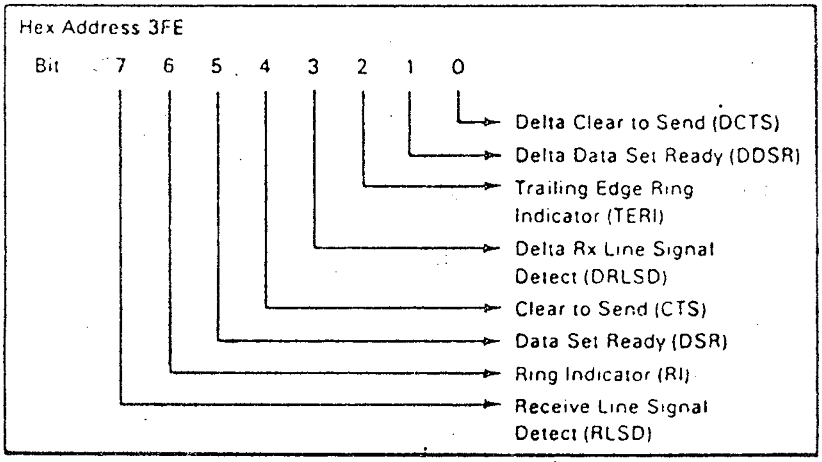
The INS8250 interrupt system can be tested by writing into the lower four bits of the modem status register. Setting any of these bits to a logical 1 generates the appropriate interrupt (if enabled). The resetting of these interrupts is the same as in normal INS8250 operation. To return to normal operation, the registers must be reprogrammed for normal operation and then bit 4 of the modem control register must be reset to logical 0.

Bits 5 through 7: These bits are permanently set to logical 0.

Modem Status Register

This eight-bit register provides the current state of the control lines from the modem (or peripheral device) to the processor. In addition to this current-state information, four bits of the modem status register provide change information. These bits are set to a logical 1 whenever a control input from the modem changes state. They are reset to logical 0 whenever the processor reads the modem status register.

The content of the modem status register are indicated and described below:



Modem Status Register (MSR)

Bit 0: This bit is the delta clear to send (DCTS) indicator. Bit 0 indicates that the $\overline{CTS}$ input to the chip has changed state since the last time it was read by the processor.

Bit 1: This bit is the delta data set ready (DDSR) indicator. Bit 1 indicates that the $\overline{DSR}$ input to the chip has changed state since the last time it was read by the processor.

Bit 2: This bit is the trailing edge of ring indicator (TERI) detector. Bit 2 indicates that the $\overline{RI}$ input to the chip has changed from an On (logical 1) to an Off (logical 0) condition.

Bit 3: This bit is the delta received line signal detector (DRLSD) indicator. Bit 3 indicates that the $\overline{RLSD}$ input to the chip has changed state.

Note: Whenever bit 0, 1, 2, or 3 is set to a logical 1, a modem status interrupt is generated.

Bit 4: This bit is the complement of the clear to send ($\overline{CTS}$) input. If bit 4 (loop) of the MCR is set to a logical 1, this is equivalent to RTS in the MCR.

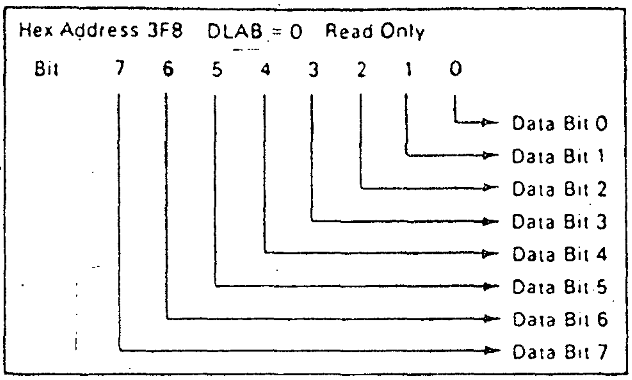
Bit 5: This bit is the complement of the data set ready ($\overline{DSR}$) input. If bit 4 of the MCR is set to a logical 1, this bit is equivalent to DTR in the MCR.

Bit 6: This bit is the complement of the ring indicator ($\overline{RI}$) input. If bit 4 of the MCR is set to a logical 1, this bit is equivalent to OUT 1 in the MCR.

Bit 7: This bit is the complement of the received line signal detect ($\overline{RLSD}$) input. If bit 4 of the MCR is set to a logical 1, this bit is equivalent to OUT 2 of the MCR.

Receiver Buffer Register

The receiver buffer register contains the received character as defined below:

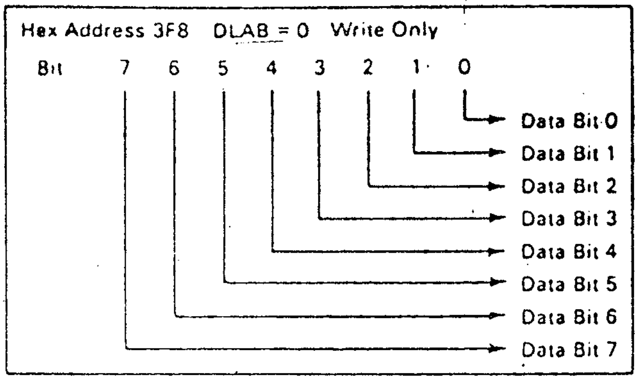


Receiver Buffer Register (RBR)

Bit 0 is the least significant bit and is the first bit serially received.

Transmitter Holding Register

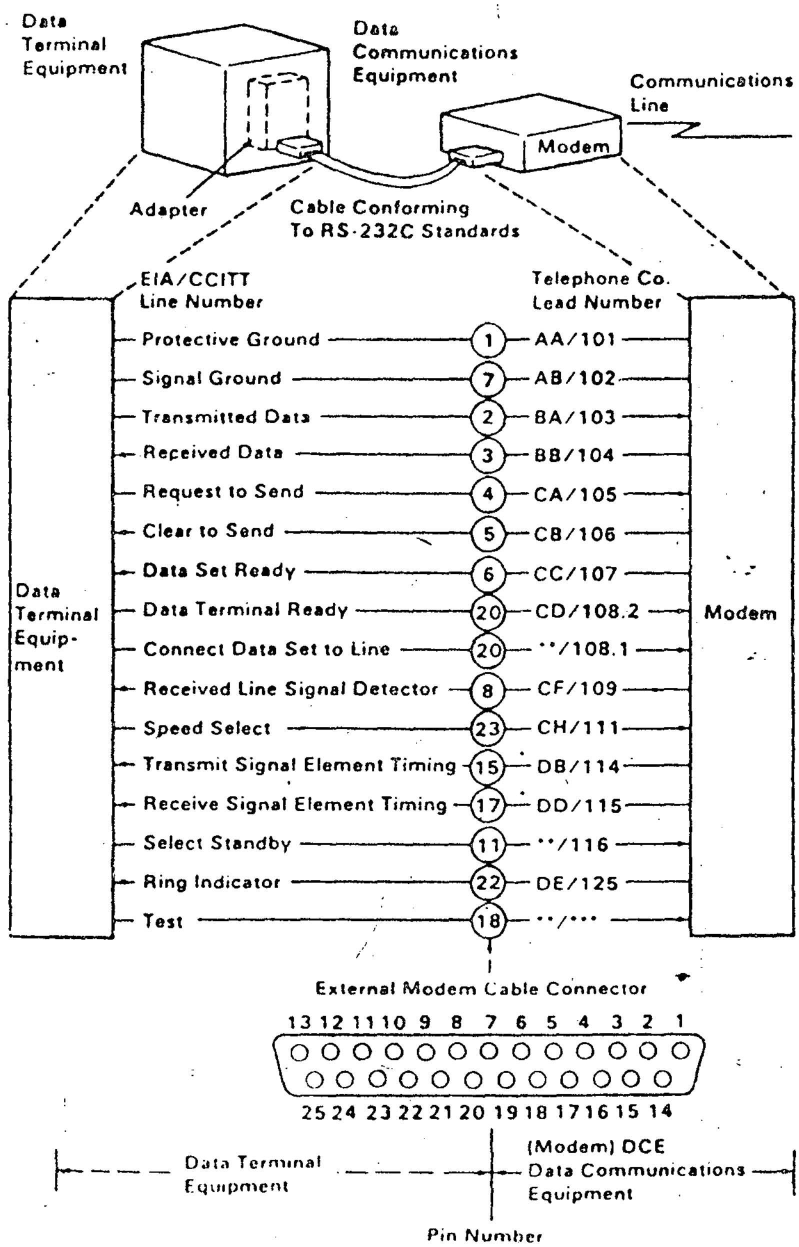
The transmitter holding register contains the character to be serially transmitted and is defined below:



Transmitter Holding Register (THR)

Bit 0 is the least significant bit and is the first bit serially transmitted.

The following is an illustration of data terminal equipment connected to an external modem using connections defined by the RS-232C interface standard:



*Not used when business machine clocking is used.
**Not standardized by EIA (Electronics Industry Association).
***Not standardized by CCITT

RESPC.ASM

```
DATA_HERE     SEGMENT
  HEAD_PTR       DW      0                  ; Head pointer to the queue
  TAIL_PTR       DW      0                  ; Tail pointer to the queue
  CHAR_COUNT     DW      0                  ; No.of chars in the buffer
  XOFF_SENT      DW      0                  ; Flag=1 if xoff is sent  .
  QUEUE          DB      1000  DUP(0)       ; Queue of 1000 chars
DATA_HERE     ENDS

      XON          EQU           17         ; DC1 character
      XOFF         EQU           19         ; DC3 character

CODE_HERE     SEGMENT
              ASSUME CS:CODE_HERE,DS:DATA_HERE

START:           MOV     AX,DATA_HERE       ; Load Data Segment Register
                 MOV     DS,AX              ;

                 MOV     CX,AX              ;
                 MOV     AX,0000            ; Set data segment=0000
                 MOV     DS,AX              ;
                 MOV     BX,0184H           ;
                 MOV     [BX],CX            ; Store ds in 0184H
                 MOV     DS,CX              ;
                 MOV     HEAD_PTR,00        ; Initialize head and tail
                 MOV     TAIL_PTR,00        ;   pointers
                 MOV     XOFF_SENT,00       ; Set xoff_sent flag to 00
                 MOV     CHAR_COUNT,00      ; Set char_count to 00

                 PUSH    DS                 ;
                 MOV     AL,60H             ;
                 MOV     BX,SEG COMM_INT;
                 MOV     DS,BX              ; Set interrupt vector 0CH
                 MOV     DX,OFFSET COMM_INT ; to our comm_int service
                 MOV     AH,25H             ;   routine
                 INT     21H                ;
                 POP     DS                 ;

                 JMP     OVER               ;
```

```
                 ;COMMUNICATION INTERRUPT SERVICE ROUTINE

COMM_INT:       STI                          ;
                PUSH    AX                   ;
                PUSH    BX                   ;
                PUSH    CX                   ;
                PUSH    DX                   ;
                PUSH    DI                   ;
                PUSH    DS                   ;

                MOV     DX,03F8H             ; Read char from 8250
                IN      AL,DX                ;
                PUSH    AX                   ;
                MOV     AX,DATA_HERE         ; Load Data Segment Register
                MOV     DS,AX                ;
                POP     AX                   ;
                MOV     DI,TAIL_PTR          ;
                INC     DI                   ; Increment tail pointer
                CMP     DI,1000              ; If end of queue make it
                JNE     FULL_CHECK           ;   circular
                MOV     DI,00                ;
FULL_CHECK:     CMP     DI,HEAD_PTR          ; Check if queue is full,
                JE      NO_MORE              ;  if full go to no more
                MOV     BX,TAIL_PTR          ;
                MOV     QUEUE[BX],AL         ; Store char in queue
                MOV     TAIL_PTR,DI          ; Restore tail_ptr
                INC     CHAR_COUNT           ; Increment char_count
                MOV     AX,CHAR_COUNT        ;
                CMP     AX,900               ; If char_count > 900,
                JL      GO                   ;   send XOFF to VAX
NO_MORE:        MOV     AL,19                ;   and set xoff_sent flag
                MOV     DX,03F8H             ;
                OUT     DX,AL                ;
                MOV     XOFF_SENT,01         ;
GO:             MOV     AL,20H               ; End of interrupt signal
                OUT     20H,AL               ;   to 8259

                POP     DS                   ;
                POP     DI                   ;
                POP     DX                   ;
                POP     CX                   ;
                POP     BX                   ;
                POP     AX                   ;

                IRET                         ;

OVER:           MOV     DX,OFFSET COMM_INT-OFFSET PG_LEN
                INT     27H                  ; Make the program
PG_LEN          LABEL   BYTE                 ;   memory  resident

CODE_HERE     ENDS
              END
```

PCNET.ASM

```
;; This  program installs interupt routines
;; and responds to the requests.

                INCLUDE MYLIB.EQU
                IF1
                INCLUDE MY_LIB.MAC
                ENDIF
BLK_SIZ         EQU     128
XOFF            EQU     19
XON             EQU     17
CODE            SEGMENT

ASSUME          CS:CODE,DS:CODE

                ORG     2CH             ;environment segment
ENV_SEG         LABEL   WORD            ;
                ORG     5CH             ;
PARAM1          LABEL   BYTE            ;  parameter1
                ORG     6CH             ;  parameter2
PARAM2          LABEL   BYTE            ;
                ORG     80H             ;  buffer length
BUF_CNT         LABEL   BYTE            ;
                ORG     82H             ;  command tail
BUFFER          LABEL   BYTE            ;
                ORG     100H            ;
 STRT:                                  ;
                JMP      OVER_DATA      ;
 TRAY           DB      ?               ;
 STKPTR         DW      ?               ; save stack
 STKSEG         DW      ?               ; save stack
 dummy          DB      'XRDQN'         ;
 PGM_NAME       DB      'C:COMMAND.COM',0   ;
                DB       20 DUP(0)      ;
 PRMBLK         DW      0               ;
 C0             DW      OFFSET BUFFER   ;
 C1             DW      ?               ; SEG cmd_line
                DW      OFFSET PARAM1   ;
 C2             DW      ?               ; SEG param1
                DW      OFFSET PARAM2   ;
```

```
 C3             DW      ?                 ; SEG param2
 BUFF           DB      20, "/C DIR>C:SDIR.C " , 13
FLAG            DW       0
FILE_NAME       DB       20 DUP(?)
HANDLE1         DW       0
HANDLE          DW       0
NO_OFCHAR       DW       0
INTFLAG         DW       0
KEYFLAG         DW       0
REMOVE          DW       0

P1IP            DW       0000             ;
P1CS            DW       0000             ;
P1PSW           DW       0000             ;
P1AX            DW       0000             ;
P1BX            DW       0000             ;
P1CX            DW       0000             ;
P1DX            DW       0000             ;
P1SS            DW       0000             ;
P1SP            DW       0000             ;
P1BP            DW       0000             ;
P1ES            DW       0000             ; This is process
P1DS            DW       0000             ; control block
P1DI            DW       0000             ;
P1SI            DW       0000             ;
TIP             DW       0000             ;
TCS             DW       0000             ;
TPSW            DW       0000             ;
TEMP            DW       0000             ;
TEMP2           DW       0000             ;
TEMP3           DW       0000             ;
                                          ;
      ;  This procedure unmaskes comm_int in 8259 ,
OVER_DATA:
                CALL     INIT             ; intialize 8250.
                MOV      DX,03F8H         ;
                IN       AL,DX            ;
                MOV      FLAG,00          ;
                                          ;
                MOV      INTFLAG,0        ;
                MOV      KEYFLAG,0        ;
                                          ;
                PUSH     DS               ;
                MOV      AL,0CH           ;
                MOV      DX,OFFSET TEMPCOM ; store the address of
                                 ; TEMPCOM
                MOV      AH,25H           ;  at 0CH.
                INT      21H              ;
                POP      DS               ;
```

```
                MOV     DS,AX           ;TIMER and store it
                MOV     AL,63H          ;in int 63H(unused in dos)
                MOV     AH,25H          ;
                INT     21H             ;
                POP     DS

                PUSH    DS              ;
                MOV     AL,21H          ;
                MOV     AH,35H          ;
                INT     21H             ;
                MOV     AX,ES           ;Get interrupt vector 21H
                MOV     DS,AX           ;and store it in int 64H.
                MOV     DX,BX           ;(unused in dos)
                MOV     AL,64H          ;
                MOV     AH,25H          ;
                INT     21H             ;
                POP     DS

                MOV     DX,OFFSET TEMPCOM ; store the adress of
                MOV     AX,2567H        ; TENPCOM at vector 67H.
                INT     21H             ;

               JMP     RESIDENT

SYSINT:
                PUSHF                   ;
                STI                     ;  push flags of the
                   ;  interrupted
                PUSH    AX              ;  process.
                PUSH    BX              ;
                PUSH    DS              ;
                                        ;
                PUSH    CS              ;
                POP     DS              ;
                                        ;
                CMP     INTFLAG,1       ;check if intflag is set
                JE      PROS            ;if set jump to pros
                CMP     AH,0AH          ;     else
                CMP     AH,01           ;    to read from keyboards
                JZ      TRY             ;    goto try
                CMP     AH,07           ;
                JE      TRY             ;
                CMP     AH,08           ;  jump to pros
                JNE     PROS            ;
TRY:                                    ;
                MOV     AH,01           ; check if keay is pressed
                INT     16H             ;if not loop until pressed
                JZ      TRY             ;
                MOV     KEYFLAG,01      ;  set the keyflag
                                        ;
PROS:                                   ;
                PUSH    CS              ;
                POP     DS              ;
```

```
                POP     DS              ;
                POP     BX              ; restore registes of
                POP     AX              ;  interrupted process
                POPF                    ;
                INT     64H             ; execute system routine
                                        ;
                PUSH    CX              ;
                PUSHF                   ;
                POP     CX              ;
                PUSH    BP              ;
                MOV     BP,SP           ;
                ADD     SP,0AH          ;
                PUSH    CX              ; give flags to the
                MOV     SP,BP           ; interrupted routine.
                POP     BP              ;
                POP     CX              ;
                PUSH    DS              ;
                PUSH    CS              ;
                POP     DS              ;
                DEC     INTFLAG         ; reset inflag
                                        ;
                CMP     KEYFLAG,01      ; chek if key flag is set
                JNE     NO_KEY          ;
                CMP     AL,00           ; chek if the pressed key
                JNE     NO_KEY          ; is extended key then
                MOV     INTFLAG,01      ;   set intflag
                MOV     KEYFLAG,00
NO_KEY:                                 ;
                POP     DS              ; return to the
                IRET                    ; interrupted process.
                                        ;



TIME_INT:                               ;
                                        ;
                INT     63H             ; execute actual timer

                           ; routine for system time
              ; keeping
                PUSH    DS              ;
                PUSH    CS              ; store DS in temp2 location
                POP     DS              ;  and make it to point to
                POP     TEMP2           ;  data of this routine.
                                        ;
                POP     TIP             ; pop the stack which containe
                POP     TCS             ; IP,CS and FLAGS of intrrupte
                POP     TPSW            ; program into temp locations.
                                        ;
                PUSH    AX
                MOV     AX,TCS          ;

                CMP     INTFLAG,01
```

```
                JE      SAME            ;    service routine.
                CMP     AX,0E9CH        ;
                JE      SAME            ;
                CMP     AX,0070H        ;
                JE      SAME            ;
                CMP     AX,0D91H        ;
                JE      SAME            ;
                POP     AX              ;

                XCHG    AX,P1AX         ;
                XCHG    BX,P1BX         ;
                XCHG    CX,P1CX         ;  exchange all registers of
                                        ;  interrupted process and
                XCHG    DX,P1DX         ;   next process to be execute
                XCHG    P1DI,DI         ;
                XCHG    P1SI,SI         ;
                XCHG    P1BP,BP         ;
                XCHG    P1SP,SP         ;
                                        ;
                MOV     TEMP,AX         ;
                MOV     AX,TEMP2        ;
                XCHG    P1DS,AX         ;
                MOV     TEMP2,AX        ;
                MOV     AX,SS           ;{ when we give controle to
                XCHG    P1SS,AX         ;other program we should
                MOV     SS,AX           ;restore all of it's registers
                MOV     AX,ES           ;and save the rigisters of
                XCHG    P1ES,AX         ;interrupted program in PCB}
                MOV     ES,AX           ;

                PUSH    P1PSW           ;  push IP,CS and FLAGS of
                PUSH    P1CS            ;   next process to be execute
                PUSH    P1IP            ;

                MOV     AX,TIP          ;
                MOV     P1IP,AX         ; store  IP,CS and FLAGS of
                MOV     AX,TCS          ;
                MOV     P1CS,AX         ; interrupted process in PCB
                MOV     AX,TPSW         ;
                MOV     P1PSW,AX        ;
                MOV     AX,TEMP         ;
                JMP     RETURN          ;

 SAME:
                POP     AX              ;
                PUSH    TPSW            ; give control to the next
                PUSH    TCS             ; process to be executed.
                PUSH    TIP             ;
                                        ;
 RETURN:        MOV     DS,TEMP2        ;
                IRET                    ;
```

```
TEMPCOM:
                STI
                PUSH    AX              ; disable 8259 IR4
                IN      AL,21H          ;  by masking bit 4 of
                OR      AL,10H          ;  mask register.
                OUT     21H,AL          ;
                POP     AX              ;
                PUSH    DS              ;
                PUSH    CS              ;
                POP     DS              ;
                POP     P1DS            ;
                MOV     P1AX,AX         ;
                MOV     P1BX,BX         ;
                MOV     P1CX,CX         ;
                MOV     P1DX,DX         ; store all registers of
                MOV     P1DI,DI         ;  interrupted process in
                MOV     P1BP,BP         ;  process control block.
                MOV     P1SI,SI         ;
                MOV     P1ES,ES         ;
                MOV     P1SS,SS         ;
                POP     P1IP            ;
                POP     P1CS            ;
                POP     P1PSW           ;
                MOV     P1SP,SP         ;
                                        ;

                MOV     DX,03F8H        ;read charector from UART
                IN      AL,DX           ;and chek if it is ESC.
                CMP     AL,1BH          ;
                JNE     GO_BACK         ;
                JMP     MULTI           ;
GO_BACK:                                ; enable ir4 by unmasking bit
                IN      AL,21H          ; in IMR of 8259 so new char
                AND     AL,0ECH         ;  in UART can interrupt
                OUT     21H,AL          ;
                                        ;
                MOV     AL,20H          ; end of interrupt signal
                OUT     20H,AL          ;   to 8259

                CALL    DISABLE         ; give control back to the
                                        ; interrupted process.
MULTI:
                MOV     AX,CS           ;
                MOV     DS,AX           ;
                MOV     ES,AX           ;
                MOV     SS,AX           ; intialize registers.
                MOV     SP,0FFFEH       ;
                MOV     STKPTR,SP       ;
                MOV     STKSEG,SS       ;
                                        ;
                MOV     AL,60H          ;
                MOV     AH,35H          ;
                INT     21H             ;
                MOV     AX,ES           ;Get interrupt vector for
                MOV     DS,AX           ;comm_int and store it
                MOV     DX,BX           ;at vector 0Ch(unused in dos)
```

```
                MOV     AH,25H          ;
                INT     21H             ;
                MOV     AL,20H          ; end of interrupt signal
                OUT     20H,AL          ;   to 8259
                                        ; enable ir4 by unmasking bit
                IN      AL,21H          ; in IMR of 8259 so new char
                AND     AL,0ECH         ;  in UART can interrupt
                OUT     21H,AL          ;

                MOV     AX,CS
                MOV     DS,AX
                MOV     ES,AX
BEGIN:
                MOV     INTFLAG,0       ; send char 'C' to inform
                MOV     AL,43H          ;
                CALL    SEND            ;  the other PC .
                CALL    GETCHAR
                CMP     AL,41H          ; if the recieved charector
                JE      SYSTEM          ;is other than 'A' then call
                MOV     TRAY,AL         ;  facility program else
PASCAL:                                 ;  jump to label system.
                MOV     AL,08H                ;
                MOV     DX,OFFSET TIME_INT    ;
                MOV     AH,25H                ;
                INT     21H                   ;

SCALL:
                MOV     DX,OFFSET COME_BACK; store  the   address
                MOV     AX,2566H        ;  come_back in vector 66H
                INT     21H             ;
                                        ;

                MOV     AX,3565H        ; get the address of the
                INT     21H             ;  facility program and
                MOV     TEMP3,ES        ;   give control to it.
                MOV     DS,TEMP3        ;
                PUSHF                   ;
                PUSH    ES              ;
                PUSH    BX              ;
                IRET                    ;
COME_BACK:                              ;
                MOV     AX,CS           ; after rhe execution of the
                MOV     ES,AX           ;   facility program control
                MOV     DS,AX           ;   comes ro this label.
                                        ;
                MOV     SP,STKPTR       ;
                MOV     SS,STKSEG       ;
                                        ;
                CALL   DISABLE          ;   give control back to the
                                        ;   user
                                        ;
                                        ;
SYSTEM:                                 ;
                CALL    GETCHAR         ;
                MOV     AL,08H          ;
```

```
                MOV     DX,OFFSET TIME_INT ;make vectopr 08H to tim

                MOV     AH,25H          ; routine
                INT     21H             ;
                                        ;
                MOV     CX,0EH          ;
NO:             PUSH    CX              ;
                MOV     CX,0EFFH        ;  wait for some time
NET:            LOOP    NET             ;
                POP     CX              ;
                LOOP    NO              ;
                                        ;
                PUSH    DS              ;
                MOV     AL,63H          ;
                MOV     AH,35H          ;
                INT     21H             ;
                MOV     AX,ES           ;Get interrupt vector for
                MOV     DS,AX           ;TIMER and store it
                MOV     DX,BX           ;in int 08H(unused in dos)
                MOV     AL,08H          ;
                MOV     AH,25H          ;
                INT     21H             ;
                POP     DS

                MOV     AX,CS           ; intialize registers
                MOV     ES,AX           ;
                MOV     DS,AX           ;
                                        ;
                MOV     DI,80H          ;
NEXT:                                   ;
                CALL    GETCHAR         ;
                MOV     BYTE PTR [DI],AL; read the command sent by
                CMP     AL,CR              ;    user on  other PC  and
                JE      FNAME_OVER;     ;    place it offset 80H.
                INC     DI              ;
                JMP     NEXT            ;
                                        ;
FNAME_OVER:                             ;
                MOV     [C1],DS         ; cmd line
                MOV     [C2],DS         ; FCB1
                MOV     [C3],DS         ; FCB2
                MOV     BX,1200         ; release memory to load
                MOV     AH,4AH          ;  command.com
                INT     21H             ;
                JC      NORED                 ;
                MOV     DX,OFFSET PGM_NAME ;
                MOV     Al,00           ; load and execute command.com
                MOV     BX,OFFSET PRMBLK      ;
                MOV     AH,4BH          ;
                INT     21H             ;
                MOV     SP,STKPTR       ; restore stack.
                MOV     SS,STKSEG       ;
                                        ;
                MOV     AX,CS           ; call the facility program to
                MOV     DS,AX           ;   send the output of  the
                MOV     ES,AX           ;   executed.
```

```
                MOV     TRAY,44H        ;
                JMP     SCALL           ;
                CALL    DISABLE         ;
                                        ;
NORED:          @WRITE  'MEMORY REDUCTION FAILED'
                CALL    DISABLE

                INCLUDE  SUB.LIB        ;
RESIDENT:                               ;
                MOV      AX,00          ;  store the address of SYSINT
                MOV      DS,AX          ;   at vector 21H.
                MOV      DI,0084H          ;
                MOV      CX,OFFSET SYSINT  ;
                MOV      [DI],CX           ;
                MOV      [DI+2],CS         ;
                                           ;
                MOV      AX,3103H       ; make the program memory
                MOV      DX,2000H       ;  resident and reserve
                INT      21H            ;  20K to load command.com.
                                        ;
PG_LEN          LABEL    BYTE           ;
CODE            ENDS                    ;
                END     STRT
```

```
;;This librery provides some procedures used in the PCNET program

GETBUFF   PROC    NEAR

                  PUSH    BX                        ;
                  PUSH    CX                        ;
                  PUSH    DX                        ;
                  PUSH    DI                        ;
                  MOV     CX,DS                     ;

                  MOV     AX,0000                   ;make DS register to
                  MOV     DS,AX                     ;point comm_int routine
                  MOV     BX,0184H                  ;
                  MOV     DS,[BX]                   ;
                  MOV     BX,0000                   ;
                  MOV     DI,[BX]                   ;move head_ptr to di
                  CMP     DI,[BX+2]                 ;compare with tail_ptr
                  JE      NO_CHAR                   ;if equal queue is full
                  MOV     AL,[BX+8][DI]             ;quit.load  char
                  INC     DI                        ;pointed by head_ptr,
                  CMP     DI,1000                   ;to al incr di make the
                  JNE     OK                        ;queue circular.
                  MOV     DI,00                     ;
OK:               MOV     [BX],DI                   ;  restore head_ptr
                  DEC     WORD PTR [BX+4]           ;  decrement  char_coun
                  CMP     WORD PTR [BX+6],1         ;  Check for xoff
                  JNE     NO_CHAR                   ;
                  CMP     WORD PTR [BX+4],750       ;  if char_count<=750
                  JGE     NO_CHAR                   ;
                  MOV     DX,03F8H                  ;     send xon
                  MOV     AL,17                     ;
                  OUT     DX,AL                     ;
                  MOV     WORD PTR [BX+6],0         ;

NO_CHAR:          MOV     DS,CX                     ;
                  POP     DX                        ;
                  POP     CX                        ;
                  POP     BX                        ;
                  POP     DI                        ;
```

```
                RET                                     ;
GETBUFF     ENDP                                        ;

                                                        ;
;;this procedure sends a charector to the comm1.

SEND            PROC        NEAR

                PUSH    DX
                MOV     DX,03FDH
                PUSH    AX
RETRY:
                IN      AL,DX                           ;
                AND     AL,20H                          ;chek weather THR reg
                JZ      RETRY                           ; is empty.
RETRY1:
                IN      AL,DX
                AND     AL,40H                          ;check weather TXSHR
                JZ      RETRY1                          ; is empty.
                POP     AX                              ;
                MOV     DX,03F8H                        ; send the charector
                OUT     DX,AL                           ;  in al to the comm1.
                POP     DX                              ;
                RET                                     ;
SEND            ENDP                                    ;

GETCHAR         PROC         NEAR

GETIT:
                PUSH    AX
                PUSH    BX

                MOV     AL,0
                MOV     AH,14
                MOV     BH,0
                INT     10H
                MOV     AL,8
                MOV     AH,14
                MOV     BH,0
                INT     10H

                POP     BX
                POP     AX

                MOV     AL,0                            ;This procedure returr

                CALL    GETBUFF                         ; a charector from
                CMP     AL,0                            ; the queue.
                JE      GETIT                           ;
                RET                                     ;
GETCHAR         ENDP                                    ;


DISABLE         PROC      NEAR
                POP     REMOVE
```

```
                MOV     DX,OFFSET TEMPCOM       ; store the offset of
                MOV     AX,250CH                ; TEMPCOM at vector 0CH
                INT     21H                     ;
                PUSH    CS
                POP     DS
                MOV     AX,P1AX                 ;
                MOV     BX,P1BX                 ; restore all user
                MOV     CX,P1CX                 ; registers and
                MOV     DX,P1DX                 ; give full control.
                MOV     DI,P1DI                 ;
                MOV     SI,P1SI                 ;
                MOV     BP,P1BP                 ;
                MOV     SP,P1SP                 ;

                MOV     SS,P1SS                 ;
                MOV     ES,P1ES                 ;
                PUSH    P1PSW                   ;
                PUSH    P1CS                    ;
                PUSH    P1IP                    ;
                MOV     DS,P1DS                 ;
                IRET                            ;
 DISABLE        ENDP


     ;  This procedure unmaskes comm_int in 8259 ,
     ;  initializes 8250  and transfers the comm_int
     ;  vector stored at 2Dh to 0Ch .
 INIT    PROC   NEAR
                PUSH    DS
                PUSH    DI
                PUSH    DX
                PUSH    CX
                PUSH    BX
                PUSH    AX

                IN      AL,21H                  ;
                AND     AL,0ACH                 ;unmask irq4(comm int)
                OUT     21H,AL                  ;

                MOV     DX,03FBH                ;
                MOV     AL,80H                  ;Initialize 8250
                OUT     DX,AL                   ;set DLAB t0 1
                DEC     DX                      ;
                DEC     DX                      ;
                MOV     AL,00                   ;
                OUT     DX,AL                   ;set baud rate low byt
                DEC     DX                      ;
                MOV     AL,0CH                  ;
                OUT     DX,AL                   ;set baud rate high by
                INC     DX                      ;       (9600)
                INC     DX                      ;
                INC     DX                      ;
                MOV     AL,03                   ;no.of bits 8,no parit
                OUT     DX,AL                   ;1 stop bit
```

```
                DEC     DX                      ;
                DEC     DX                      ;
                MOV     AL,01                   ;
                OUT     DX,AL                   ;enable 8250 COMM_INT

                POP     AX
                POP     BX
                POP     CX
                POP     DX
                POP     DI
                POP     DS
                RET
INIT        ENDP
```

FACILITY.PAS

```
{ This program serves the requests for phone,mail,transefer file
  and sends the output of dos commands }

program facility;
procedure  intpas                    ;external 'intpas.com';
procedure  return                    ;external 'return.com' ;
procedure  noswap                    ;external 'noswap.com' ;
function getkey (var i : integer):integer  ;external 'getkey.bin' ;
function getbuff (var i : integer):integer ;external 'getbuff.bin';

type
    registers = record
      ax,bx,cx,dx,bp,sp,di,si,cs,ds,es,ss,flags:integer;
    end;
    mes=string[30];

var
  filename : string[12];
  fp  : FILE;
  fp1 : text;
  c : char;
  reg:registers;
  ccr,clf,csp                               : char;
  st        : array[1..80] of char;
  bufsiz,wblkno,rblkno,flag,result1,result:integer;
  flag1,row,col,fflag,i,j,tray:integer;
  buff:array[1..128] of char;
  complete,cleartosend:boolean;

{ This procedure returns a character from comm-buffer
  It waits till a character is received.}
procedure getcharp(var i:integer);
var j : integer;
  begin
    i := 0;
    while i = 0 do
```

```
      begin
       j := getbuff(i);
       row:=wherex;
       col:=wherey;
       write(chr(0));
       gotoxy(row,col)
      end;
   end;

   procedure getchar(var i:integer);
     begin
       i:=0;
       while(i=0) do j:=getbuff(i);
     end;

{ This procedure sends a char  through 8250  it
  waits untill shift reg and transmitter buffer
  are empty  }
procedure send(w : integer);
var x,y,z,i : integer;
begin
     y := 0;z := 0;
     while ((y = 0) or (z = 0)) do
       begin
         x := port[$03FD];
         y := x and $0020;
         z := x and $0040;
       end;
     port[$03F8] := w;
end;


{  This procedure finds the cursor position on the screen
   using DOS interrupt 10h  }

procedure findcur(var i,j :integer);
begin
     reg.ax := $0300;
     reg.bx := 0;
     intr($10,reg);
     i := reg.dx div 256 + 1;
     j := reg.dx and $FF + 1;
end;

{ This procedure positions the cursor on the screen
  at given row and column   }

procedure poscur(row,col : integer);
begin
     reg.ax := $200;
     reg.bx := 0;
```

```
     reg.dx := ((row-1) * 256) or ((col-1));
     intr($10,reg);
end;


{ This procedure displays a character at the current cursor position
  in the given attribute and advances the cursor to next column.
  The first arguement is the character to be displayed and
  the second is the attribute  }

procedure display(i,j :integer);
var k : integer;
begin
     case i of
       10 : write(clf);          { print line feed as it is }
       13 : write(ccr);          { print carriage return as it is }
       09 : for k:=1 to 8 do write(csp);  { expand tab  }
       else begin
              reg.ax := i or $0900;
              reg.bx := j;
              reg.cx := 01;
              intr($10,reg);
              findcur(row,col);
              poscur(row,col+1);
            end;
     end;
end;

{  This procedure waits for a char from keyboard
   and returns the same    }

procedure readkbd(var i : integer);
var j :integer;
begin
  i := 255;
  while i=255 do j := getkey(i);
end;


{ This procedure displays the given string at the current
  cursor position in the given mode   }

procedure set_display(st:mes;mode:integer);
var i,j : integer;
begin
    for i:=1 to length(st) do display(ord(st[i]),mode);
end;

{ This procedure reads the file name sent by the other
   PC and sets fflag if error in reading.}
```

```
procedure readfilename;
 var
   i,j,k  :integer;
   c,s:char;
    begin
      filename := '
      i := 1;
      getcharp(j);
      if(j<>03) then
        begin
         while j<>13 do
           begin
             filename[i] := chr(j);
             i := i + 1;
             getcharp(j);
            end;
         end
        else
         fflag:=1;
    end;

{This procedure sends the file  }

procedure sendfile;
 var
 i,j,k:integer;
 c,s:char;

 begin
  readfilename;
  if (fflag<>1) then
   begin
    assign(fp,filename);
     {$i-}
    reset(fp);
      {$i+}
    if (ioresult <> 0) then send(03)
    else
      begin
        send(67);
        {$i-}
{ This procedure reads the file blockwise and sends it charector
  by  charector.if error in  reading  it  reopens  the file and
   moves  reading head to the sector to read. }
          while NOT EOF(fp) do
            begin
              blockread(fp,buff,1,result1);
              while(Ioresult<>0) do
                 begin
                   assign(fp,filename);
                   reset(fp);
```

```
                    seek(fp,rblkno);
                    blockread(fp,buff,1,result1);
                   end;
                 rblkno:=rblkno+result1;
                 i:=1;
                 while ( (i<=128) and NOT complete) do
                   begin
                     if(buff[i]<> chr(26)) then
                       begin
                        for j:=1 to 200 do;
                        send(ord(buff[i]));
                        i:=i+1;
                       end
                     else
                       begin
                        complete := TRUE;
                        send(26);
                       end;
                    end;
                { j := getbuff(i);
                 if i = 19 then while i <> 17 do j := getbuff(i);}
               end;
            close(fp);
        end;
     end;
{ This  procedure disables multitasking }
     noswap;
    end;
{ This procedure recieves  a file sent by the other PC }
procedure  getfile;
  begin
    noswap;
    readfilename;
    if ( fflag <> 1 ) then
      begin
       assign(fp1,filename);
       {$i-}
       rewrite(fp1);

       if (Ioresult=0 ) then
         begin
           send(67);  i:=0;
{ This recives a file and stores it in disk }
              while  (i<>26) do
               begin
                 getcharp(i);
                 write(fp1,chr(i));
               end;
             close(fp1);
         end
       else
```

```
        begin
          send(03);
          writeln( 'CAN NOT CREATE FILE');
        end;
       end;
 end;

{ This procedure responds to the phone call   }
procedure speak;
const
     LF    = 10;          { line feed                          }
     CR    = 13;          { carriage return                    }
     ESC   = 27;          { escape                             }
     SPACE = 32;          { space                              }

var
    i,j,k,n1,n2,code,index,scancode,appm : integer;
    row,col,saverow,savecol,attrib,ptr     : integer;
    wintop,winbottom,fgcolor,bgcolor       : integer;
    r1,c1,r2,c2 : integer;
    continue : boolean;

      begin
        noswap;
        writeln('      YOUR HAVE A PHONE CALL :   PRESS (Y/N)    ');
         read(kbd,c);
         if((c='y') or (c='Y')) then
          begin
{ This creates two windows on the screen  }
           send(67);
           window(1,1,80,25);
           clrscr;
           gotoxy(30,1);
           set_display(' PCNET PHONE FACILITY ',$70);
           gotoxy(1,2);
           for i:= 1 to 79 do write('-');
           gotoxy(1,13);
           for i:= 1 to 79 do write('-');
           gotoxy(1,14);
           for i:= 1 to 79 do write('-');
           gotoxy(1,25);
           for i:= 1 to 79 do write('-');
           gotoxy(1,1);
           r1 := 1;c1 := 1;
           r2 := 1;c2 := 1;
           j := 1;
          continue := TRUE;
{ this displayes and sends the key pressed and displayes
  charectors recieved from other PC }
          while continue do
            begin
```

```
              i  := 255;
              j  := getkey(i);
              i  := lo(i);
              if  i  = 03  then
                  begin
                  send(03);
                  continue := FALSE;
                  clrscr;
                  writeln('     YOU HAVE COME BACK TO YOUR PROCESS
                  end
              else
                if i  <> 255  then
                  begin
                   send(i);
                   window(1,3,80,12);
                   textbackground(0);
                   textcolor(5);
                   gotoxy(r1,c1);
                   write(chr(i));
                   if  i=13  then write(chr(10));
                   r1  := wherex;c1  := wherey;
                  end;
                i  := 0;
                j  := getbuff(i);
                i:=lo(i);
                if  i  = 03  then continue:=FALSE  else
                 if  i  <> 0  then
                   begin
                    window(1,15,80,24);
                    textbackground(0);
                    textcolor(5);
                    gotoxy(r2,c2);
                    write(chr(i));
                    if  i=13  then write(chr(10));
                      r2  := wherex;c2  := wherey;
                   end;
             end;
          end   else send(03);
       end;

PROCEDURE COMMAND;

   var
    i,k,j:integer;
     begin
     {$i-}
       assign(fp1,'C:dire$ct');
       reset(fp1);
       if(ioresult<>0) then write('  OPEN FAILED ')
       else
       while NOT EOF(fp1) do
```

```
      begin
        read(fp1,c);
        send(ord(c));
      end;
      send(26);
      rewrite(fp1);
      close(fp1);
    end;

{ MAIN PROGRAM STARTS HERE }
 begin
     flag:=0;
     flag:=mem[0:$200];
     if (flag<>0) then
      begin
        complete:=FALSE;
        bufsiz:=256;
        rblkno:=0;
        wblkno:=0;
        result1:=0;
        result:=0;
        fflag:=0;
{ This is to read the request from other PC.   }
        i:=memw[0000:$019A];
        tray:=mem[i:$0103];
        case tray of
          $54 : sendfile;
          $52 : getfile;
          $50 : speak;
          $44 : command;
        end;
{ This gives control back to PCNET.  }
     return;
    end
    else
{ This procedure makes the whole program memory resident }
        intpas;
   end.
```

GETKEY

```
CODE_HERE       SEGMENT
                ASSUME CS:CODE_HERE

         ;   PUBLIC   GETKEY                        ; This function returns
the
        GETKEY  PROC   NEAR                         ;   input from the keybo
ard
                   PUSH     BP                      ;   if any.

                   MOV      AH,01                   ; Check if keay is press
ed
                   INT      16H                     ;
                   JZ       QUIT                    ; If not goto quit,
                   MOV      AH,00                   ;
                   INT      16H                     ; else read the key,
        CONT:      MOV      BP,SP                   ;
                   LES      DI,[BP+4]               ; Transfer this key to
                   MOV      ES:[DI],AX              ;  the external variable

        QUIT:      POP      BP                      ;
                   RET      6                       ;
        GETKEY  ENDP
        CODE_HERE          ENDS
                   END
```

GETBUFF

```
         CODE_HERE     SEGMENT
                       ASSUME CS:CODE_HERE
                                                          ;
                 ;  PUBLIC  GETBUFF                       ; This function returns
a
         GETBUFF  PROC   NEAR                             ;   char,if any from buf
fer


                         PUSH     BP                      ;
                         MOV      BP,SP                   ;
                         PUSH     AX                      ;
                         PUSH     BX                      ;
                         PUSH     CX                      ;
                         PUSH     DX                      ;
                         PUSH     DI                      ;
                         MOV      CX,DS                   ;

                         MOV      AX,0000                 ;  Load DS reg with
                         MOV      DS,AX                   ;    Data segment of Res
ident
                         MOV      BX,0184H                ;
                         MOV      DS,[BX]                 ;
                         MOV      BX,0000                 ;
                         MOV      DI,[BX]                 ;  Move head_ptr to di
                         CMP      DI,[BX+2]               ;  Compare with tail_ptr
                         JE       NO_CHAR                 ;  If equal queue is ful
l,quit.
                         MOV      AL,[BX+8][DI]           ;  load  char pointed by
 head_ptr
                         PUSH     DI                      ;  into the external var
iable
                         LES      DI,[BP+4]               ;
                         MOV      ES:[DI],AX              ;
                         POP      DI                      ;
                         INC      DI                      ;  Increment head_ptr
                         CMP      DI,1000                 ;  If head_ptr=1000 then
                         JNE      OK                      ;    make it circular
                         MOV      DI,00                   ;
         OK:             MOV      [BX],DI                 ;  Restore head_ptr
                         DEC      WORD PTR [BX+4]         ;  Decrement  char_count
                         CMP      WORD PTR [BX+6],1       ;  If xoff_sent and
                         JNE      NO_CHAR                 ;
                         CMP      WORD PTR [BX+4],750     ;    char_count <= 750,
                         JGE      NO_CHAR                 ;
                         MOV      DX,03F8H                ;    send XON to VAX,
                         MOV      AL,17                   ;
                         OUT      DX,AL                   ;    and reset xoff_sent
 flag.
                         MOV      WORD PTR [BX+6],0       ;

         NO_CHAR:        MOV      DS,CX                   ;
                         POP      DI                      ;
                         POP      DX                      ;
                         POP      CX                      ;
                         POP      BX                      ;
                         POP      AX
                         POP      BP
```

NOSWAP

```
;;This procedure disables multitasking.
CODE            SEGMENT
ASSUME          CS:CODE,DS:CODE
NO_SWAP       PROC        NEAR

                PUSH    DS
                PUSH    ES
                PUSH    DX
                PUSH    BX
                PUSH    AX
                MOV     AL,63H          ;
                MOV     AH,35H          ;
                INT     21H             ;
                MOV     AX,ES           ;Get interrupt vector for
                MOV     DS,AX           ;TIMER and store it
                MOV     DX,BX           ;in int 08H(unused in dos)
                MOV     AL,08H          ;
                MOV     AH,25H          ;
                INT     21H             ;
                POP     AX
                POP     BX
                POP     DX
                POP     ES
                POP     DS
                RET
NO_SWAP       ENDP
CODE            ENDS
                END
```

RETURN.ASM

```
CODE_HERE       SEGMENT
                ASSUME CS:CODE_HERE
RETURN        PROC NEAR

                    POP       AX
                    MOV       AX,3563H
                    INT       21H
                    PUSHF
                    PUSH      ES
                    PUSH      BX
                    IRET

                    RET       0
RETURN              ENDP

CODE_HERE       ENDS
                END
```

INTPAS

```
;;This procedure makes the facilty program memory
;;resident,sets the flag and stroes the starting
address at vector 65H
CODE                  SEGMENT
           ASSUME   CS:CODE,DS:CODE
INTPAS       PROC        NEAR
              POP     AX
              MOV     AX,00
              MOV     DS,AX            ; set the flag.
              MOV     DI,200H          ;
              MOV     AX,01            ;
              MOV     DS:[DI],AX       ;
              MOV     AX,CS            ;
              MOV     ES,AX            ;
              MOV     DS,AX            ;
              MOV     DX,100H          ; store the starting address
              MOV     AX,2565H         ;  at vector 65H
              INT     21H
              MOV     AX,3103H         ;  make the program memory
              MOV     DX,1500H         ;  resident and return to dos.
              INT     21H
              RET
INTPAS    ENDP

CODE      ENDS
          END
```

ASKFILE.PAS

```
{ This is to request and get a file from the other pc.
  If the user on the other PC is willing to send file,
  it reads and sends the file name,if there is no error
  indicator  from the other side the file is recived and
  stored on your current directory   }


PROGRAM  askfile(input,output);
function getkey (var i : integer):integer  ;external 'getkey.bin' ;
function getbuff (var i : integer):integer ;external 'getbuff.bin';


type
    {This record contains the varioues registers in 8088
     Used in interrupt routines within TURBO  }
    registers = record
      ax,bx,cx,dx,bp,si,ds,es,flags : integer;
    end;
    mes = string[80];

var
    i,j,k,l :integer;
    c:char;
    filename : string[20];
    fp : text;
    reg : registers;
    cleartosend : boolean;

{ This procedure returns a character from   comm-buffer
  It waits till a character is received from another PC }

procedure getchar(var i:integer);
var j : integer;
  begin
    i := 0;
    while i = 0 do j := getbuff(i);
  end;
```

```
{ This procedure swaps interrupt vectors v1 and v2 }
procedure swap_vectors(v1,v2 : integer);
begin
   reg.ax := $3500;
   reg.ax := reg.ax or v1;
   intr($21,reg);
   reg.ax := reg.es;
   reg.ds := reg.ax;
   reg.dx := reg.bx;
   reg.ax := $2500;
   reg.ax := reg.ax or v2;
   intr($21,reg);
end;

{ This procedure sends a char to another PC through 8250
  It waits untill shift reg and transmitter buffer
  are empty }

procedure send(w : integer);
var x,y,z,i : integer;
begin
     y := 0;z := 0;
     while ((y = 0) or (z = 0)) do
       begin
         x := port[$03FD];
         y := x and $0020;
         z := x and $0040;
       end;
     port[$03F8] := w;
end;

{ MAIN PROGRAM STARTS HERE }

begin
{ This procedure stores the comm_int address at vector 0CH }
     swap_vectors($60,$0C);
{ read a charector from 8250 }
     i:=port[$03f8];
     i := 255;
{ clears comm_buffer }
     while i <> 0 do
       begin
         i := 0;
         j := getbuff(i);
       end;
     send(27);
     getchar(i);
      if (chr(i)= 'C') then
          begin
             send(84);
             writeln;
             write(' GIVE FILE NAME :');
             readln(filename);
{ This sends file name to the other pc }
             for i:= 1 to length(filename) do send(ord(filename[i
             send(13);
```

```
              getchar(i);
              if(lo(i)=67) then
                begin
                  assign(fp,filename);
                  rewrite(fp);
                  getchar(i);
 {This recives  file from  other PC }
                  while(i<>26) do
                    begin
                      write(fp,chr(i));
                      if(lo(i)=$03) then i:=26;
                      getchar(i);
                    end;
                  close(fp);
                end
              else writeln('file not found at the other node');
        end
        else  send($03);
 { This stores the address of tempcom at vector 0CH
   from vector 67h }
       swap_vectors($67,$0C);
     end.
```

MAIL.PAS

```
{ This  program is to mail a file to the other PC,it prompts for
    file name to be mailed and  reads the file name and sends the
    file name and file to the other PC. }

program  mail (input,output);
function getkey (var i : integer):integer  ;external 'getkey.bin'
function getbuff (var i : integer):integer ;external 'getbuff.bin'


type
     {This record contains the varioues registers in 8088
      Used in interrupt routines within TURBO  }
     registers = record
       ax,bx,cx,dx,bp,si,ds,es,flags : integer;
     end;
     mes = string[80];

var
     i,j,k,l:integer;
     c:char;
     filename : string[20];
     fp : text;
     reg : registers;
     quit,cleartosend : boolean;

{ This procedure returns a character from comm-buffer
  It waits till a character is received from other PC }

procedure getchar(var i:integer);
var j : integer;
  begin
    i := 0;
    while i = 0 do j := getbuff(i);
  end;

{  This procedure swaps vetors v1 and v2  }
```

```
procedure swap_vectors(v1,v2 : integer);
begin
   reg.ax := $3500;
   reg.ax := reg.ax or v1;
   intr($21,reg);
   reg.ax := reg.es;
   reg.ds := reg.ax;
   reg.dx := reg.bx;
   reg.ax := $2500;
   reg.ax := reg.ax or v2;
   intr($21,reg);
end;

{ This procedure sends a char to other PC through 8250
  It  waits  untill  shift reg and  transmitter buffer
  are empty  }

procedure send(w : integer);
var x,y,z,i : integer;
begin
     y := 0;z := 0;
     while ((y = 0) or (z = 0)) do
       begin
         x := port[$03FD];
         y := x and $0020;
         z := x and $0040;
       end;
     port[$03F8] := w;
end;

{ MAIN PROGRAMM }
begin
{ This procedure store the address of comm_int at vector
  0CH from vector 60H }
  swap_vectors($60,$0C);
  i:=port[$03f8];
  i := 255;
{ This  makes clears the comm_buffer }
  while i <> 0 do
    begin
      i := 0;
      j := getbuff(i);
    end;
  send(27);
  getchar(i);
  if (chr(i)= 'C') then
     begin
        send(82);
        write(' GIVE FILE NAME : ');
        readln(filename);
```

```
        {$i-}
        assign(fp,filename);
        reset(fp);
        {$i+}
        if (Ioresult<>0) then
           begin
           send(03);write('FILE NOT FOUND ');
           end
        else
           begin
{ This sends the file name to the other PC }
             for i:= 1 to length(filename) do
               send(ord(filename[i]));
             send(13);
             getchar(i);
{ This reads and sends the file to the other PC }

             if( i=67) then
              begin
               read(fp,c);
               while not EOF(fp)  do
                begin
                 send(ord(c));
                 read(fp,c);
                 j := getbuff(i);
                 if i = 19 then
                  while i <> 17 do
                   j := getbuff(i);
                end;
               send(26);
              end;
            end;
        end;
{this stores the address of tempcom at vector 0CH
  from vector 67H }
      swap_vectors($67,$0C);
    end.
```

PHONE.PAS

```
{This program is to make a phone call to the user on other pc}

program phone(input,output);
function getkey (var i : integer):integer  ;external 'getkey.bin'
function getbuff(var i : integer):integer  ;external 'getbuff.bin'

const
     LF    = 10;          { line feed                    }
     CR    = 13;          { carriage return              }
     ESC   = 27;          { escape                       }
     SPACE = 32;          { space                        }


type
    {This record contains the varioues registers in 8088.
     It is used in procedures within TURBO  }
    registers = record
      ax,bx,cx,dx,bp,si,ds,es,flags : integer;
    end;
    mes = string[30];

var
    i,j,k,n1,n2,code,index,scancode,appm : integer;
    row,col,saverow,savecol,attrib,ptr     : integer;
    wintop,winbottom,fgcolor,bgcolor       : integer;
    c,ccr,clf,csp,lastchar                 : char;
    reg  : registers;
    r1,c1,r2,c2 : integer;
    continue : boolean;

{  This procedure finds the cursor position on the screen
   using DOS interrupt 10h  }

procedure findcur(var i,j :integer);
begin
     reg.ax := $0300;
     reg.bx := 0;
     intr($10,reg);
     i := reg.dx div 256 + 1;
```

```
      j := reg.dx and $FF + 1;
end;

{ This procedure positions the cursor on the screen
  at given row and column   }

procedure poscur(row,col : integer);
begin
      reg.ax := $200;
      reg.bx := 0;
      reg.dx := ((row-1) * 256) or ((col-1));
      intr($10,reg);
end;

{ This procedure swaps interrupt vectors v1 and v2  }

procedure swap_vectors(v1,v2 : integer);
begin
    reg.ax := $3500;
    reg.ax := reg.ax or v1;
    intr($21,reg);
    reg.ax := reg.es;
    reg.ds := reg.ax;
    reg.dx := reg.bx;
    reg.ax := $2500;
    reg.ax := reg.ax or v2;
    intr($21,reg);
end;

{ This procedure sends a char to another PC through 8250
  It  waits  untill  shift  reg and  transmitter  buffer
  are empty  }

procedure send(w : integer);
var x,y,z,i : integer;
begin
      y := 0;z := 0;
      while ((y = 0) or (z = 0)) do
        begin
          x := port[$03FD];
          y := x and $0020;
          z := x and $0040;
        end;
      port[$03F8] := w;
end;


{ This procedure displays a character at the current cursor posi
  in the given attribute and advances the cursor to next column
  The first arguement is the character to be displayed and
  the second is the attribute  }

procedure display(i,j :integer);
```

```
begin
     case i of
       10 : write(clf);         { print line feed as it is }
       13 : write(ccr);         { print carriage return as it is
       09 : for k:=1 to 8 do write(csp);  { expand tab  }
       else begin
              reg.ax := i or $0900;
              reg.bx := j;
              reg.cx := 01;
              intr($10,reg);
              findcur(row,col);
              poscur(row,col+1);
            end;
     end;
end;

{ This procedure returns a character from comm-buffer
  It waits till a character is received from PC2    }

procedure getchar(var i:integer);
var j : integer;
  begin
    i := 0;
    while i = 0 do j := getbuff(i);
  end;

{  This procedure waits for a char from keyboard
   and returns the same    }

procedure readkbd(var i : integer);
var j :integer;
begin
  i := 255;
  while i=255 do j := getkey(i);
end;

{ This procedure displays the given string at the current
  cursor position in the given mode   }

procedure set_display(st:mes;mode:integer);
var i,j : integer;
begin
    for i:=1 to length(st) do display(ord(st[i]),mode);
end;

{ MAIN PROGRAM STARTS HERE }

 begin
  swap_vectors($60,$0C);
 { read a charector from 8250 }
  i:=port[$03f8];
  i := 255;
  { empty the comm_buffer }
  while i <> 0 do
```

```
      i := 0;
      j := getbuff(i);
    end;
   send(27);
   getchar(i);
   if (chr(i)= 'C') then
      begin
        send(80);
        getchar(i);
         if(lo(i)=67) then
          begin
{ This makes two windows on the screen }

           window(1,1,80,25);
           clrscr;
           gotoxy(30,1);
           set_display(' PCNET PHONE FACILITY ',$70);
           gotoxy(1,2);
           for i:= 1 to 79 do write('-');
           gotoxy(1,13);
           for i:= 1 to 79 do write('-');
           gotoxy(1,14);
           for i:= 1 to 79 do write('-');
           gotoxy(1,25);
           for i:= 1 to 79 do write('-');
           gotoxy(1,1);
           r1 := 1;c1 := 1;
           r2 := 1;c2 := 1;
           j := 1;
          continue := TRUE;

{ This displayes a key if pressed by the  user,sends  it
  to the other PC and reads a charector from comm_buffer
  if there is and displayes it }

          while continue do
            begin
             i := 255;
             j := getkey(i);
             i := lo(i);
             if i = 03 then
              begin
                continue := FALSE;
                send(03);
                clrscr;
              end
             else
               if i <> 255 then
                 begin
                  send(i);
                  window(1,3,80,12);
                  textbackground(0);
                  textcolor(5);
                  gotoxy(r1,c1);
                  write(chr(i));
```

```
                    if i=13 then write(chr(10));
                    r1 := wherex;c1 := wherey;
                  end;
                i := 0;
                j := getbuff(i);
                i:=lo(i);
                if i = 03 then continue:=FALSE  else
                 if i <> 0 then
                   begin
                    window(1,15,80,24);
                    textbackground(0);
                    textcolor(5);
                    gotoxy(r2,c2);
                    write(chr(i));
                    if i=13 then write(chr(10));
                      r2 := wherex;c2 := wherey;
                   end;
            end;
        end;
    end;
    swap_vectors($67,$0C);
 end.
```

DOS.PAS

```
{ This program reads  DOS commands from the key  board,
   and appends the  string >c:redire‡ct at the end  and
   resultent string length,' C / '   at the beginning.}

program  dos (input,output);
function getkey (var i : integer):integer  ;external 'getkey.bin' ;
function getbuff (var i : integer):integer ;external 'getbuff.bin';


type
    {This record contains the varioues registers in 8088
     Used in interrupt routines within TURBO  }
    registers = record
      ax,bx,cx,dx,bp,si,ds,es,flags : integer;
    end;

var
    i,j,k,l,flength :integer;
    c :char;
    filename,doscmd,redirect : string[20];
    command  : string[40];
    fp : text;
    reg : registers;
    quit,cleartosend : boolean;

{ This procedure returns a character from  comm_buffer
  It waits till a character is received from other PC }

procedure getchar(var i:integer);
var j : integer;
  begin
    i := 0;
    while i = 0 do j := getbuff(i);
  end;

{ This procedure swaps vectors v1 and v2 }

procedure swap_vectors(v1,v2 : integer);
begin
```

```
    reg.ax := $3500;
    reg.ax := reg.ax or v1;
    intr($21,reg);
    reg.ax := reg.es;
    reg.ds := reg.ax;
    reg.dx := reg.bx;
    reg.ax := $2500;
    reg.ax := reg.ax or v2;
    intr($21,reg);
 end;

 { This procedure sends a char to VAX through 8250
   It waits untill shift reg and transmitter buffer
   are empty  }
 procedure send(w : integer);
 var x,y,z,i : integer;
 begin
      y := 0;z := 0;
      while ((y = 0) or (z = 0)) do
        begin
          x := port[$03FD];
          y := x and $0020;
          z := x and $0040;
        end;
      port[$03F8] := w;
 end;

 begin
 { This procedure stores the address of comm_int at
 vector 0Ch from vector 60H }
      swap_vectors($60,$0C);
      i:=port[$03f8];
      i := 255;
 { this cleares the comm_buffer }
      while i <> 0 do
        begin
          i := 0;
          j := getbuff(i);
        end;
      send(27);
      getchar(i);
      if (chr(i)= 'C') then
       begin
          send($41);
          send($44);
          redirect:='>c:dire$ct';
          writeln;
          write('COMMAND>');

          readln(doscmd);
          flength:=length(doscmd)+length(redirect) + 5;
          command[1]:=chr(flength);
```

```
          for j := 1 to length(redirect) do
            begin
             command[i]:=redirect[j];
             i:=i+1;
            end;
          command[i]:=chr(13);
          i:=1;
{ This sends the command string }
          while(command[i]<>chr(13)) do
            begin
             send(ord(command[i]));
             i:=i+1;
            end;
          send(13);
          i:=0;
          getchar(i);
{this reads message sent by other PC and  displayes it }
          while( i <> 26 ) do
            begin
              write(chr(i));
              getchar(i);
            end;
      end;
{ This store the address of TMPCOM at vector 0CH from
  vector 67H }
      swap_vectors($67,$0C);
    end.
```

BIBLIOGRAPHY

1. Microprocessors and interfacing - Programming and hardware by Douglas V. Hall
2. Micro computer Systems : The 8086/8088 family by Yu-Cheng Liu and Glenn A. Gibson
3. Computer Networks by Andrew S. Tanenbaum
4. Turbo Pascal manual
5. The MSDOS handbook by Richard Allen King
6. Assembly Language Techniques by Alan R. Millar
7. IBM pc Technical reference manual
8. Programmer's guide to the IBM pc by Peter Norton
9. DOS reference manual
10. Designing and implementing Local Area Networks by Chorafas