Communication Between PC and VAX 11/780

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

D. JAGADESH

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

CERTIFICATE

This work,embodied in the dissertation titled,

COMMUNICATION BETWEEN PC and VAX 11/780

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

This work is original and has not been submitted or 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

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

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 extremly 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.

I express my heartfelt gratitude to Dr.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 Dr.Karmeshu,who has shown special interest in my work.

(D. JAGADESH)

SYNOPSIS

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

To establish communication between a TANDY1000 PC and VAX 11/780 system and then provide facilities to transfer files from PC to VAX and from VAX to PC.The PC acts as a termianal to VAX and runs most of the VAX software including the editor.The user can change the communication parameters of the terminal within the session.Files can be transferred from PC to VAX and from VAX to PC with simple commands.Error checking is incorporated and files are transferred using, one bit sliding window protocols.

This PC to VAX connection can be improved with more facilities.

CONTENTS

1. Introduction 1

2. PC and Communication

2.1 8088 Architecture 4

2.2 Interrupts and interrupt service routins 9

2.3 Serial asynchronous communication 15

2.4 RS-232C serial data transfers 19

3. The VAX 11/780 system

3.1 Hardware system 21

3.2 Software system 24

3.3 Peripheral controllers 26

3.5 The VT220 terminal 28

4. Terminal emulation of VT220 on PC

4.1 Languages chosen 32

4.2 Initializing the communication port 33

4.3 The program RESIDENT 36

4.4 The program CONNECT 41

5. File transfer utilities

5.1 Data link protocols 70

5.2 Calculating the checksum 71

5.3 Implementation details

5.3.1 File sender 75

5.3.2 File receiver 80

5.3.3 Tracing errors 84

~~6. Instructions for use~~

~~7. Future extensions and modifications~~

Appendix A 8088 instruction set 111

Appendix B Programming 8250 UART 127

Appendix C Escape and control sequences of VT220 156

Appendix D Codes and scancodes of TANDY1000 176

Appendix E ASCII character set 173

Appendix F Interrupt vectors 170

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.Our 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,we started with a subset of it.

We wanted to connect one of the pcs as a terminal to VAX 11/780,so that a user sitting in a distant room working on a pc can get a connection to VAX and proceed by running a simple software,provided the terminal is provided with an RS-232C interface.At any time he can disconnect and use the pc in isolation.As the next step,we want to provide facilities for transferring files from pc to the VAX anf from VAX into his diskette.This file transfer must be reliable and has to be carried out with error checking.

The next step is to connect all the pcs to the VAX through the above pc.This needs interconnecting all the pcs. As a first step towards this,we wanted to interconnect two pcs through RS-232C,so that thay can exchange information.This can be extended to interconnect all the pcs ,so that not only they can communaicate with each other,they can also commniacate with VAX through the master pc. 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. PC and COMMUNICATIONS

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,

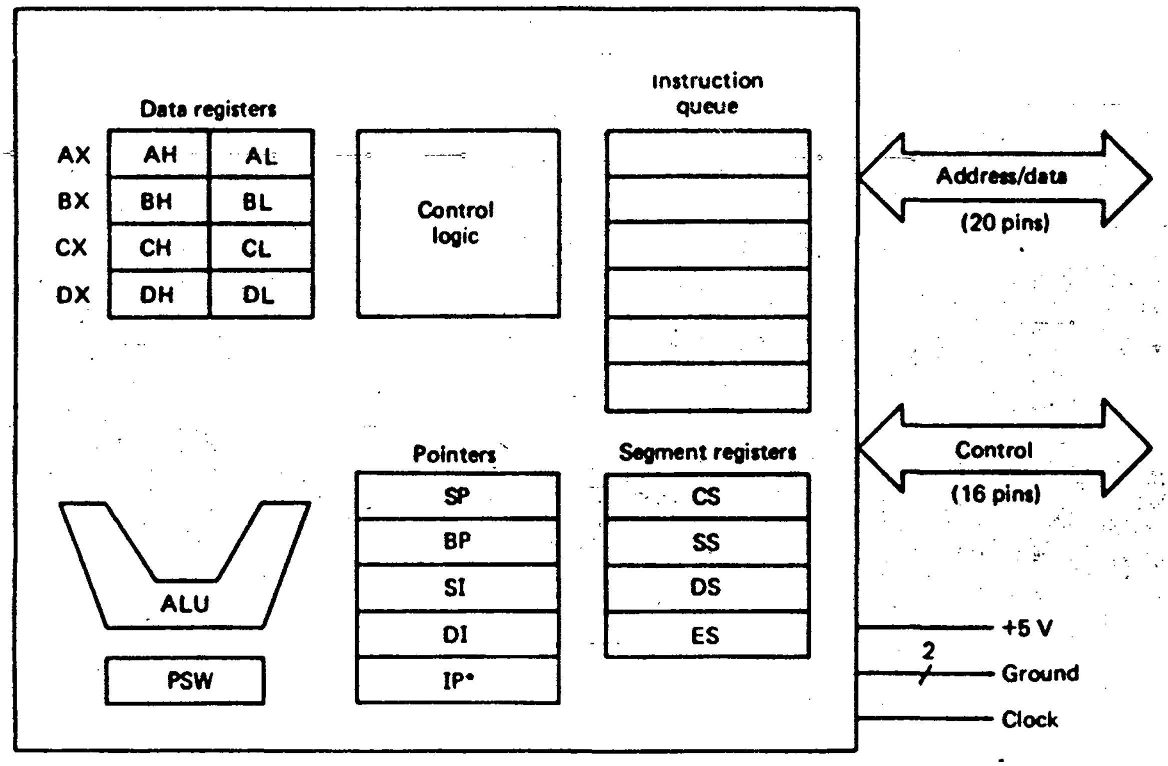
ii.The pointer group ,which includes base and index registers, but also contains the program counter and stack pointer,

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,



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

FIG 2.1 8088 BLOCK DIAGRAM

the BX,CX and DX registers play special addressing,counting and I/O roles.

BX may be used as a base register in address 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 occures.

					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.A complete list of interrupt vectors is given in appendix F.

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.

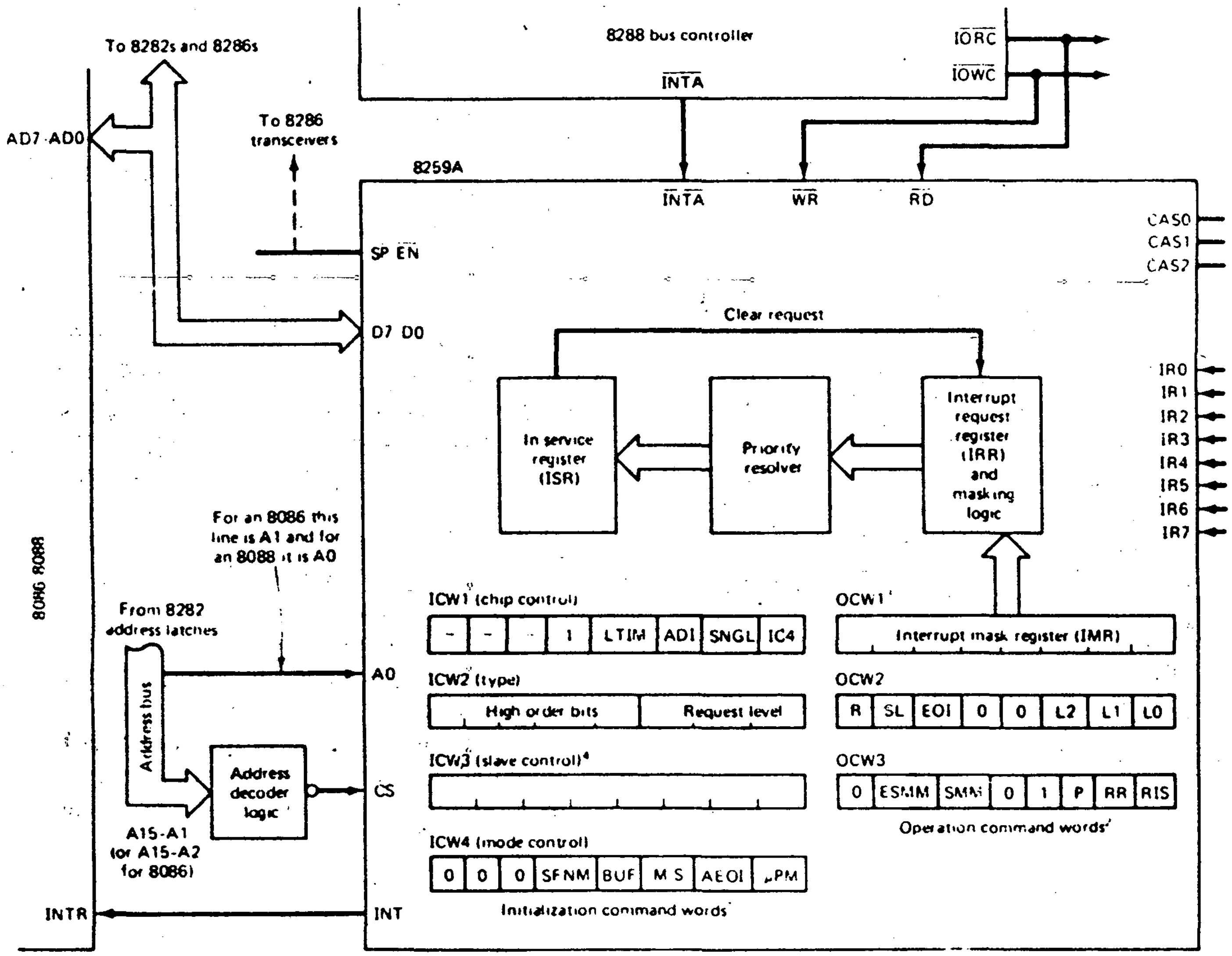
Regardless of the type of the interrupt,the action that results from an interrupt are the same and are known as the interrupt sequense. Some kind of interrupts are controlled by the IF and TF flags and in those cases,these flags must be properly set or else the interrupt action is blocked. If the conditions 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.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.


8288 bus controller
IORC
IOWC
INTA
To 8282s and 8286s
AD7-AD0
To 8286 transceivers
8259A
INTA
WR
RD
CAS0
CAS1
CAS2
SP EN
D7 D0
Clear request
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
In service register (ISR)
Priority resolver
Interrupt request register (IRR) and masking logic
8086 8088
For an 8086 this line is A1 and for an 8088 it is A0
From 8282 address latches
Address bus
Address decoder logic
A15-A1 (or A15-A2 for 8086)
A0
CS
INTR
INT
ICW1 (chip control)
- | - | - | 1 | LTIM | ADI | SNGL | IC4
ICW2 (type)
High order bits | Request level
ICW3 (slave control)⁴
ICW4 (mode control)
0 | 0 | 0 | SFNM | BUF | M S | AEOI | μPM
Initialization command words
OCW1
Interrupt mask register (IMR)
OCW2
R | SL | EOI | 0 | 0 | L2 | L1 | L0
OCW3
0 | ESMM | SMM | 0 | 1 | P | RR | RIS
Operation command words

[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

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.

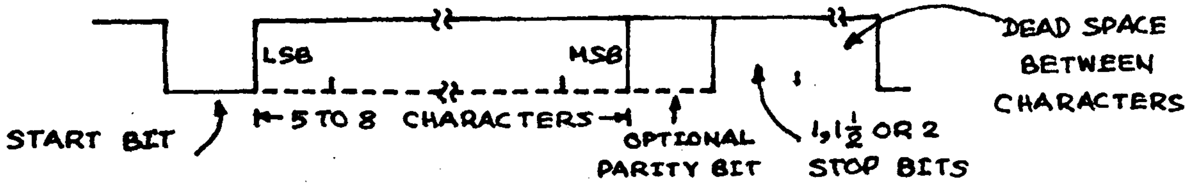


Fig 2.3 Asynchronous communication format

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 shacking 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

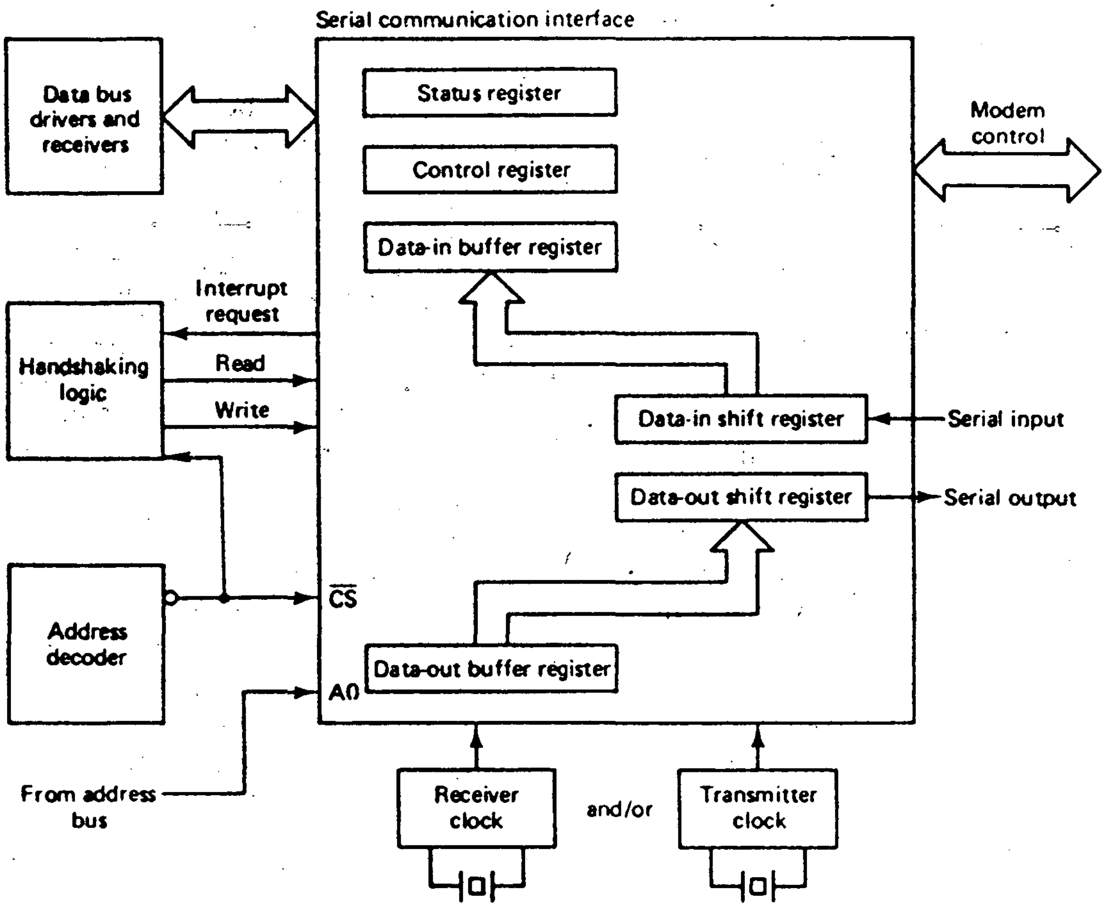


FIG 2.4

8250 UART BLOCK DIAGRAM

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 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 THE VAX 11/780 SYSTEM

This chapter gives a brief introduction to the hardware and software systems of **VAX 11/780** system. Of particular interest to us are the terminal controllers which control the behaviour of the terminals.

The VAX 11/780 is a multi user,multi language,multi programming and high performance system.It combines a 32 bit architecture with a virtual memory operating sysytem (VAX/VMS) and a memory management system.

The hardware (microcode) contains a native instruction set that includes integral floating point,packed decimal arithmetic,character and string instructions.The hardware also includes the compatibility mode instructon set that is a subset of the PDP 11/70. Programs can be executed concurrently in the native and compatibility modes.Some of the instructions in the native set are direct counterparts to high level language statements.The software system supports high level languages that use this instructions to produce compiled code.

3.1 HARDWARE SYSTEM

The VAX 11/780 system contains a CPU,a main memory system,an I/O sub system,a console sub system and peripheral equipment.The overall block diagram is given in

Fig 3.1

3.1.1 CPU

The VAX 11/780 CPU is high speed microprogrammed 32 bit computer that provides a full 32 bit operational capability (32 bit data and 32 bit address).

The CPU operates on data as defined by user program.The CPU receives the program data and instructions via the main system interconnect and manipulated data is sent back to the storage.

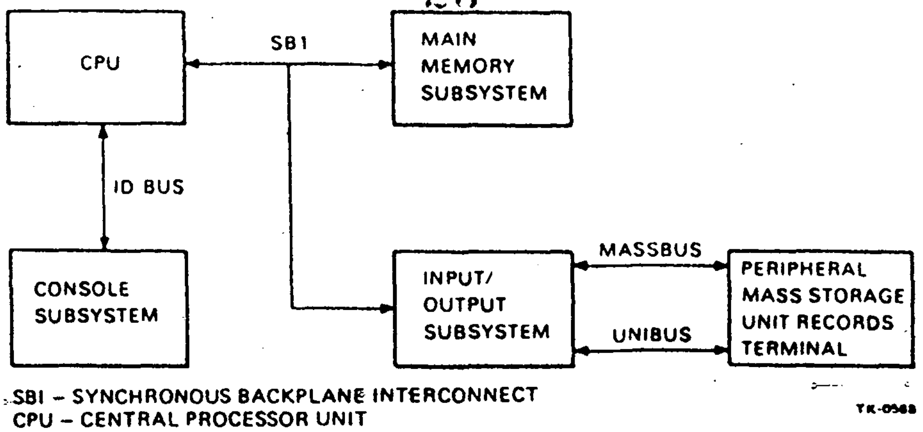
The two microprogrammed instructions (VAX 11/780 native mode and PDP 11 compatibility mode) are contained in the writable diagnostic control store(WDCS).The basic instruction sets are contained in ROM,but can be modified and added to in RAM.

3.1.2 I/O SUB SYSTEM

The I/O sub system contains the synchronous backplane interconnect(SBI),the mass bus and the uni bus.Fig 3.2 is a functional diagram of the I/O sub system and the peripheral devices.

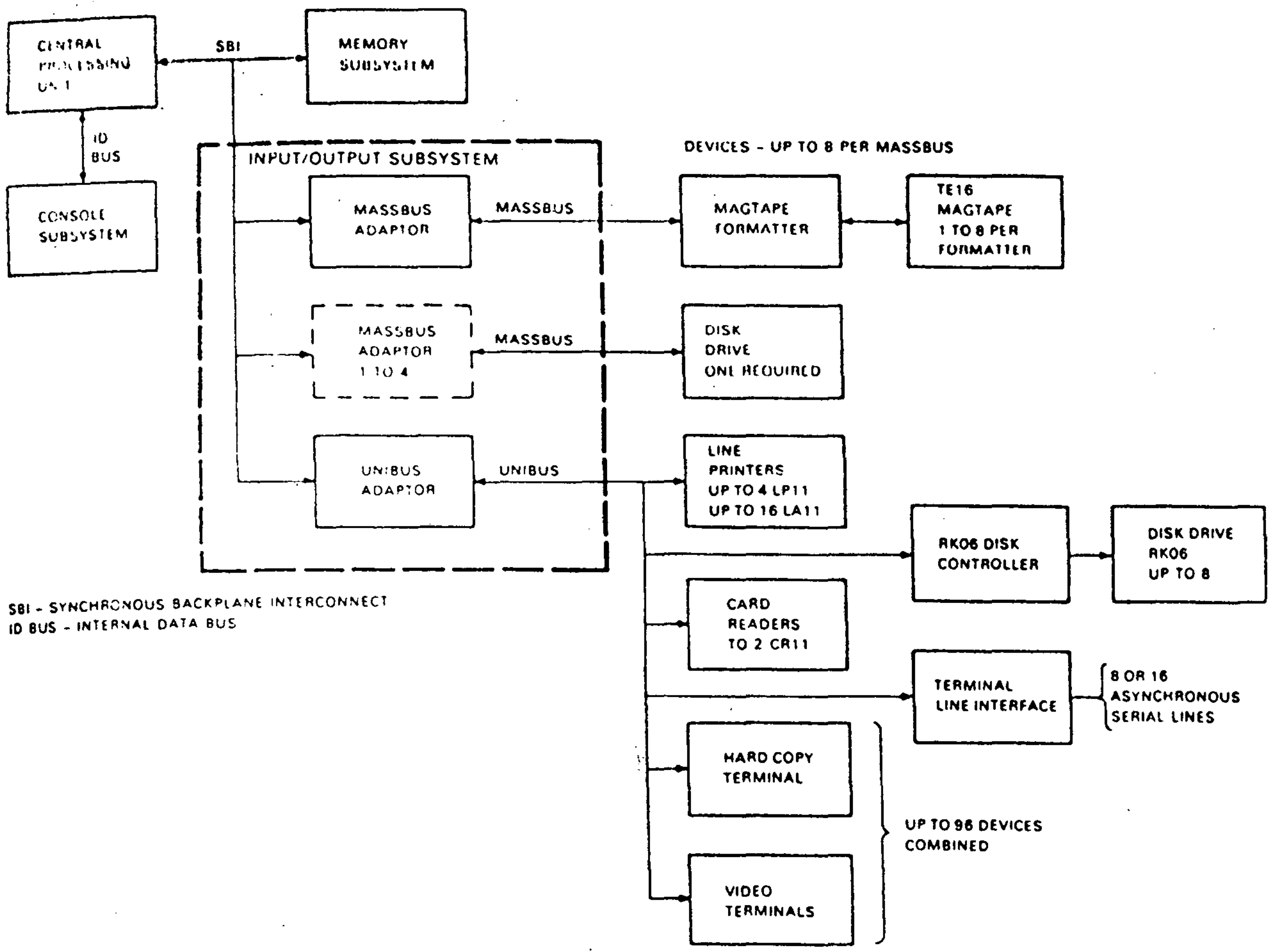
3.1.2.1 SBI

The SBI is the system's internal backplane and bus that conveys address,data and control information betweem the CPU,main memory and peripheral devices. The SBI includes several error checking and diagnostic mechanism such as -



SBI – SYNCHRONOUS BACKPLANE INTERCONNECT
CPU – CENTRAL PROCESSOR UNIT
ID BUS – INTERNAL DATA BUS

FIG 3.1
VAX 11/780 BLOCK DIAGRAM



SBI - SYNCHRONOUS BACKPLANE INTERCONNECT
ID BUS - INTERNAL DATA BUS

FIG 3.2
VAX 11/780 I/O

parity checking on data,address and commands,
protocol checks in each interface,
a history silo of the last 16 SBI cycles.

3.1.2.2 MASSBUS

Mass storage devices like disk drives and magnetic tapes are connected to massbus.The processor interface for a massbus peripheral is the massbus adapter.The massbus adapter performs control arbitration,and buffering functions. Upto four massbus adapters can be placed on the SBI.

3.1.2.3. UNIBUS

This is an asynchronous bidirectional bus.All devices other than disk drives and magnetic tapes are connected to the unibus.The unibus is connected to the SBI through the unibus adapter.The unibus adapter does priority arbitration among the devices on the unibus.

3.1.2.4 MAIN MEMORY SUBSYSTEM

The main memory sub system contains one or two memory controllers.Each controller can handle from one to 16 MOS RAM arrays.Two memory controllers can be controlled to the SBI yielding a maximum of 8 MB of physical memory that can be available on the system.

3.2 VAX SOFTWARE SYSTEM

The VAX 11/780 software system includes the **VAX**

virtual memory operating system(VAX/VMS),file and record management facilities,I/O drivers,the application migration executive batch capabilities,On-line diagnostics and error log utilities in the support languages and utility programs.

3.3 PERIPHERAL DEVICES

The VAX 11/780 supports four types of peripheral devices.

1. Mass storage peripherals such as disks and tapes
2. Unit record peripherals like line printers and card readers
3. Interprocessor communication links
4. Terminals and terminal line interface

The mass storage peripherals include RPO5,RPO6,RMO3,RKO6,RA60 and RA81 disk drives and TE16 magnetic tape transport.The unit record peipherals include LP11 and LA11 line printers and CR11 card reader.

The interprocess communication link (DMC11) is designed for high performance point to point interprocessor connection based on the DIGITAL data communication protocol.The DMC provides local or remote connection of two computers over a serial synchronous link.Both computers can include the DMC and the DECNET software,or both computers can include the DMC11 and implement their own communication software. For remote operations,a DMC11 can also communicate with a different type of synchronous interface,provided that the remote sytem has implemented the DDCMP protocal.

3.4 TERMINAL CONTROLLERS

The VAX has two types of termianl controllers, the DMF32 and DMZ32. They can support different types of terminals like VT52, VT100 and VT220.

3.4.1 DMF32 The DMF32 is an intelligent VAX family unibus controller that supports a combination of I/O devices includung

a. eight asynchronous lines

b. one synchronous line

c. one DMA line printer interface or one enhanced DR11-C functional parallel I/O port

The asynchronous multiplexer supports eight transmit and eight receive lines. Each pair of lines can be programmed to operate at one of 16 baudrates, ranging from 50bps to 19.2 kbps. Line 0 and line 1 have split-speed capability and full modem control. The asynchronous multiplexer also supports the auto echo function.

Transmission can be selected for DMA or SILO operation. In SILO mode, each line transmits characters from its own 32 bit character buffer. These buffers are loaded under host software control. In DMA mode, a transmit line transmits characters from the main memory location specified by the buffer address and the character count. All eight characters share a 48 character receive SILO. There is a programmable SILO timeout period for the receive SILO.An interrupt can be generated when

a. sixteen characters have entered the SILO

b. The SILO has been nonempty for a time greater than a timeout period.The time out period can be set to zero.

The asynchronous lines are connected either to a data terminal equipment(DTE) or data communication equipment(DCE) via standard EIA RS232-C 25 pin connector.The signal levels of the synchronous multiplexer are RS423 compatible.

The synchronous interface is a single line DMA communication interface with full modem control.The interface supports various protocols like SDLC,HDLC and DDCMP.This line can transfer the messages,generate and check CRC and DMA these messages to and from host memory.The host level software performs all message acknowledgements and higher level network functions.

3.4.2 DMZ32 The DMZ32 asynchronous multiplexer contains three octets of eight transmit and eight receive lines,each making a total of 24 lines available for data.These 24 lines can be programmed to operate at one of 14 baudrates from 50bps to 19.2kbps.All 24 lines have the capability of operating with different receive and transmit baudrates.All lines have modem control and each receive and transmit line can be independently enabled or disabled.There is a separate receive and transmit interrupt vector for each of the 3 octets.These vectors may be enabled or disabled independently.Separate TXReady and Rxdata available bits exist for each octet to allow for non interrupt driven device operation.These octets can be operated independently.For example,each octet can be reset without effecting any of the other octet.The DMZ32 may be programmed to echo all the received characters.

3.5 THE VT220 TERMIANL

The VT220 is a general purpose video display terminal that lets you interact with a computer.The user sends characters to the computer by typing on the keypad.Characters sent by the application program appear as text on the terminal screen.The terminal operates by executing standard ANSI functions.

Character set modes : The VT220 has two basic character modes,multinational and national.Multinational mode supports the DEC multinational character set(DECMCS).The DECMCS is an eight bit character set that contains most characters used in the major European languages.Thr ASCII character set is included in the DECMCS.National mode supports the natioanl replacement character sets(NRC sets).The NRC sets are a group of eleven 7 bit caharacter sets.the national character set available is determined by the keyboard selected in setup.Only one national character set is available for use at any time.National mode restricts compatibility to a 7 bit environment in which the use of the DECMCS is disabled.

Keyboard : See Fig4.10(page 56) for layout of the keyboard.The keyboard consists of -

A main keypad - This keypad operates like a standard typewriter keyboard.The ⌫ (delete) key sends a DEL character.Normally erases one character to the left of the cursor.

Editing keypad - This keypad is used to control the cursor and edit data that is already entered.

Auxiliary keypad - This keypad allows us to enter numeric data.This set of keys have special purpose in application programs such as editor.The terminal sends different codes for normal mode and application mode.

Function keys - The top row function keys have functions assigned by the application software.

COMMUNICATION :

The terminal operates on full-duplex asynchronous lines only.It operates with the following national and international communication standards.

EIA standard RS232C/RS423

CCITT V.24

CCITT V.26 (V1.0)

CCITT X.20 (V.21)

The terminal can be either connected directly to a local host computer or to remot host through modem.

The terminal sends and receives characters serially formatted.The character format is shown below -

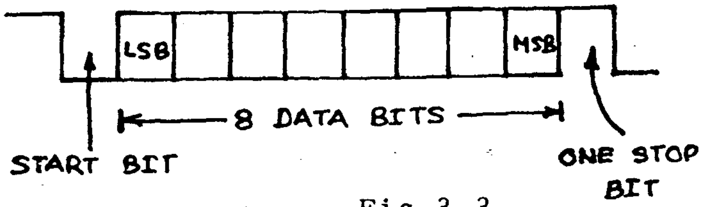


Fig 3.3

The terminal stores incoming characters in a character input buffer and processes the characters on a first-in/first-out basis.The size of the input buffer is 254 characters.When the input buffer fills to 64 or 128 characters (selectable in SET-UP),the terminal sends an XOFF character (if enabled) to stop the host computer from sending more characters.If the computer fails to respond to the XOFF character,the terminal sends a second XOFF character when the input buffer fills to 220 characters.The terminal sends a third XOFF character when the buffer is full.

When the input buffer contents falls below 32 characters,the terminal sends an XOFF character to tell the host computer to start sending characters again.If XON/XOFF is enabled,the terminal recognizes XON and XOFF characters.When it receives XOFF,it stops sending data.

ESCAPE and CONTROL sequences :

VAX sends certain escape and control sequences for cursor positioning,window definition,inserting text,deleting text and so on.An escape sequence is an escape character(27) followed by an ASCII character.A control sequence is an escape character followed by an opening square bracket ([) followed by a sequence of ASCII caharactters. The complete list of escape and control sequences is given appendix C.

For example,ESCk stands for "clear the display from the cursor position to the end of line".Similarly the control sequence ESC[7m instructs that the following characters should be displayed in reverse video.The sequence ESC[n;mh instructs to position the cursor at row n and column m.

Since we are writing our own routines for escape and control sequences,we can introduce our own sequences.When the application program on VAX sends that sequence,our module traps that sequence and executes the code which is of interest to us.This idea is used in file transfer from PC to VAX and from VAX to PC and for getting back to MSDOS.

When a file is to be transferred from PC to VAX,a program **PCTOVAX** has to be run on VAX.This program sends the sequence ESCT ,which is trapped by our **CONNECT** program and the file transfer program is initiated on PC.

IV TERMINAL EMULATION

In chapter 2,we discussed how asynchronous serial data can be sent or received with an 8250 on a polled or an interrupt basis.This chapter gives the implementation details of how the PC sends and receives characters from the host (VAX) and how it emulates a VT220.

4.1 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 **RESIDENT** 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,READMEM** 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.

The language C would also have been a good choice,since it provides pointer arithmetic,which would have been very useful in file transfer.But at this time,we don't have a fast C compiler on PC with all or most of the above features.

Since VAX supports C language,the programs developed on VAX are coded in C language.Due to the various features of C language like pointer arithmetic and post increment operation,the code turned out to be very compact.

4.2 Initializing the communication port

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 VAX 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.

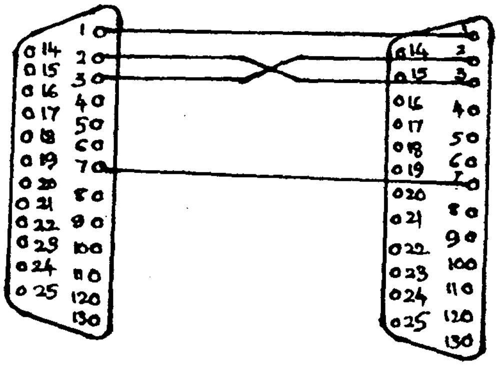


Fig 4.1

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.

Why interrupt driven

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 cahracters are received by polling the 8250 and displayed,and input from the keyboard is sent to VAX.

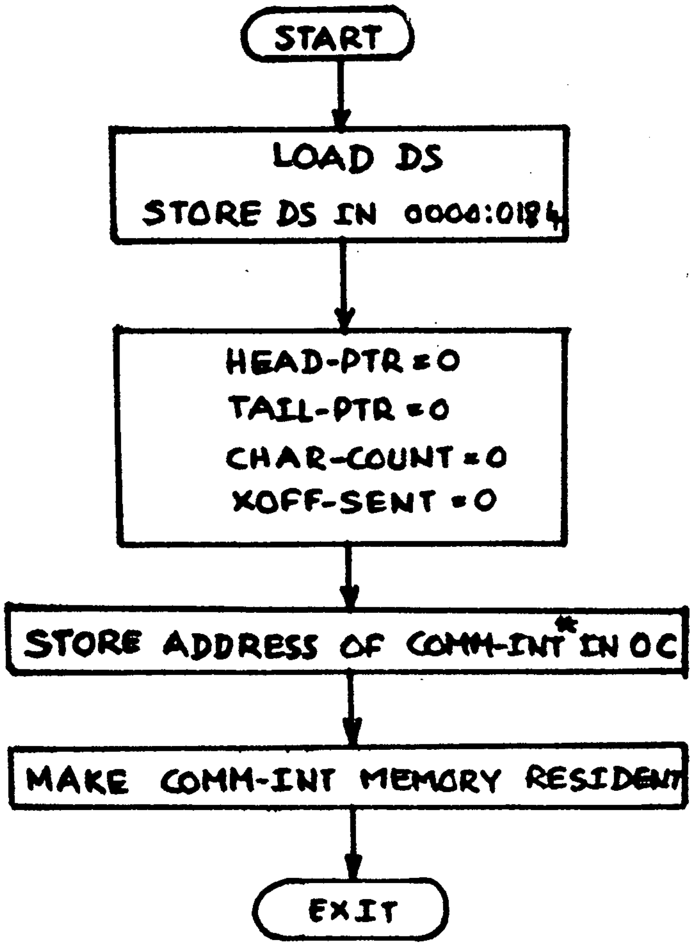
```
Initialize 8250
repeat
    if keypressed,then read key and send it
    if UART has a character,then read the
        character and display it
forever.
```

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.

4.3 The program RESIDENT

See Fig4.2 for a flowchart of **RESIDENT**.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

RESIDENT



* SEE NEXT PAGE

FIG 4.2

parameters. In this implementation,the Data Segment of **RESIDENT** 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 0C,using DOS function call 25H.

The communication interrupt service routine,which is resident all the time in memory, receives characters from VAX and stores them in circular buffer.The flow chart is given in fig 4.3.

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 is a circular buffer.The following figure attempts to show how this works.

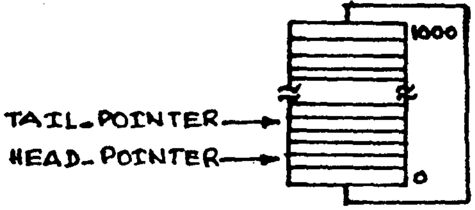


Fig 4.4 (circular buffer)

COMM-INT

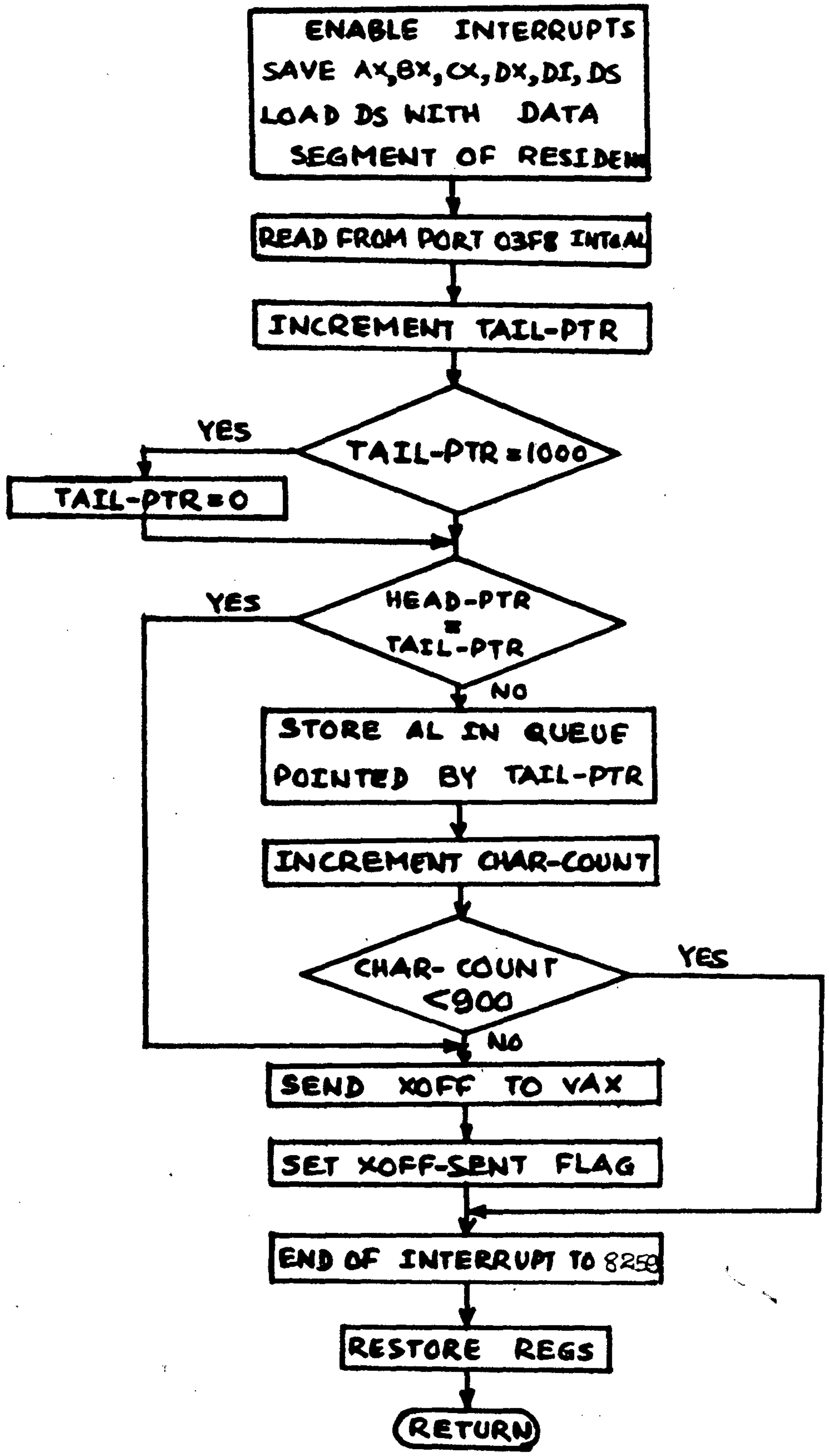


FIG 4.3

One 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 numbar 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 VAX to stop it from sending more characters and the xoff-sent flag is set.But some characters may be sent by VAX 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 RESIDENT 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.

4.4 The main program CONNECT

The main program is written in pascal.It basically repeats two steps.First it checks if there is any input from the keyboard.It reads the input if there is any,processes it and sends the appropriate code.It then checks if there is any data in the buffer,reads and processes the data and displays it.The flow chart is given in Fig 4.5. The function **GETKEY** checks if there is any input from the keyboard.The function **GETBUFF** checks,if there is any data received from host.These are functions written in assembly language and are called from the pascal program **CONNECT**.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 incr(var n : integer) : integer;
                      external 'incr.bin';
    var i,j : integer;
    begin
       i := 1;
       j := incr(i);
       write('i = ',i);
    end.
```

BLOCK DIAGRAM OF CONNECT

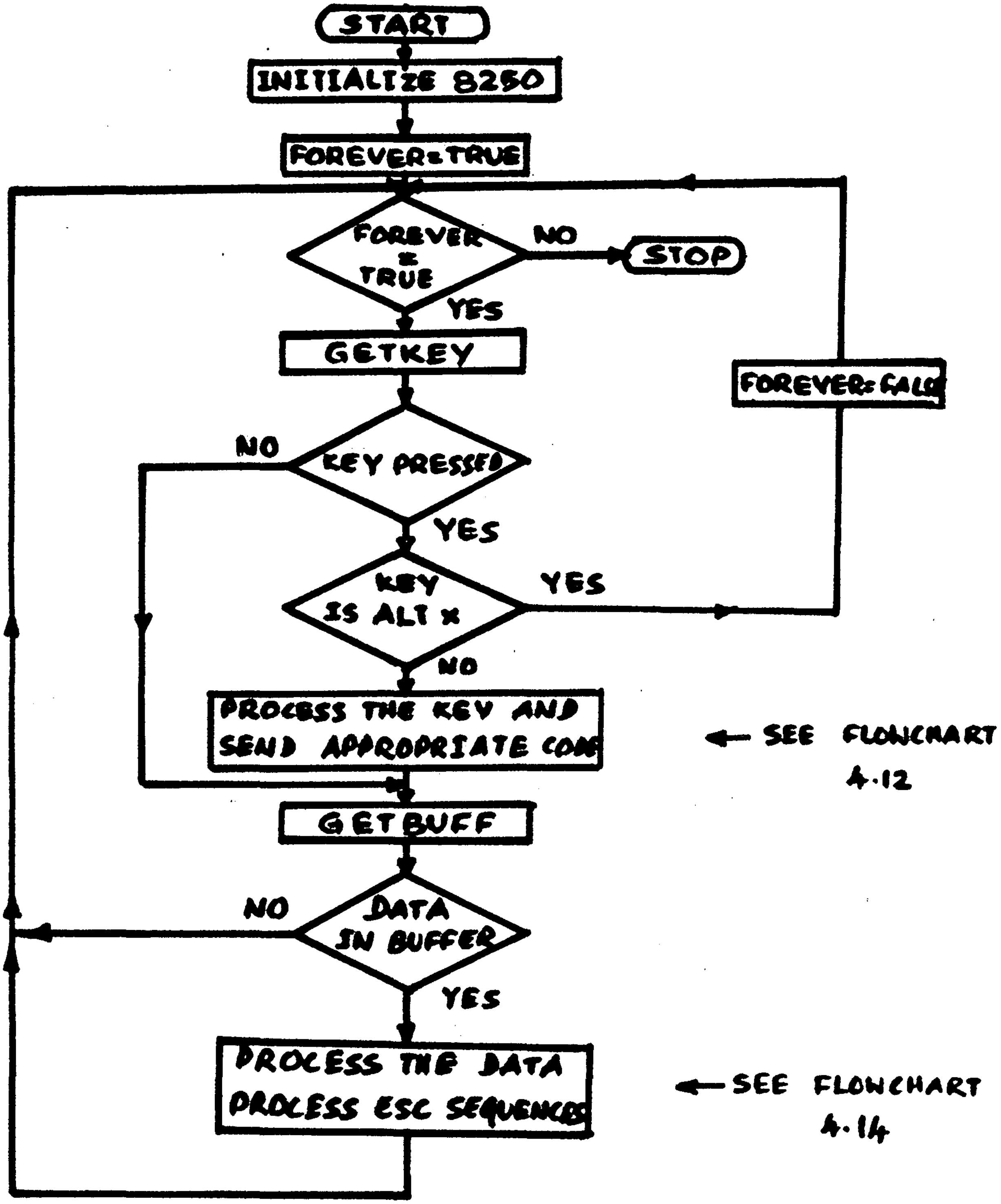


FIG 4.5

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

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

Let us see how the parameters are passed.Turbo Pascal passes parameters through stack.Fig 4.6 shows the state of stack when a function with a single parameter,say incr(i) is called.

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

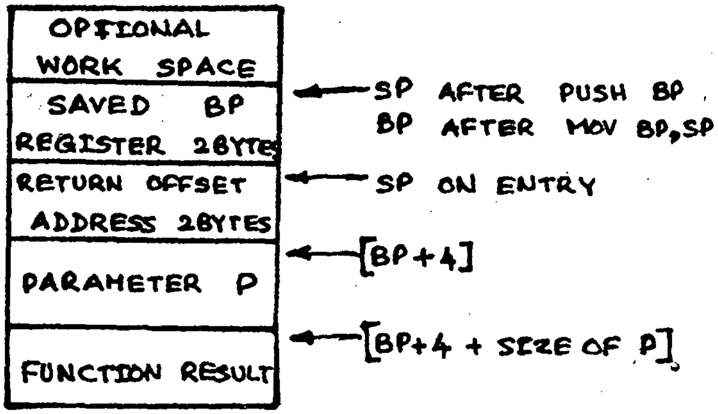


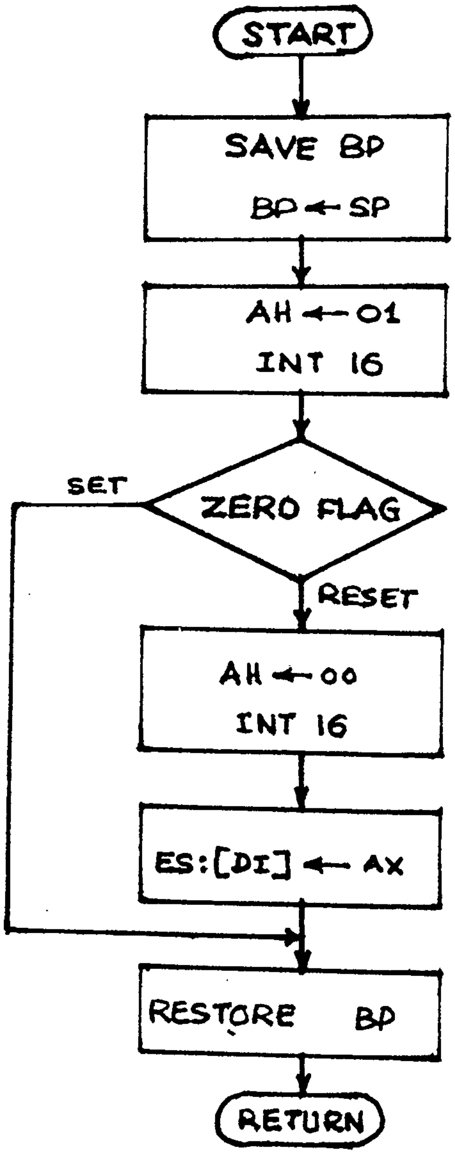
Fig 4.6

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 must also pop the paremeter,we shold give RET 6.

The main program contains the external functions **GETKEY** and **GETBUFF** ,and various other procedures.

Function 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 4.7.

FUNCTION GETKEY



CHECK IF KEY IS PRESSED

READ THE PRESSED KEY INTO AL

PUT THE KEY INTO EXTERNAL VARIABLE

FIG 4.7

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.

Function 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 4.8.

All the registers are saved.The contents of [0000:0184] are loaded into DS,so that the variables of RESIDENT 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

FUNCTION GETBUFF

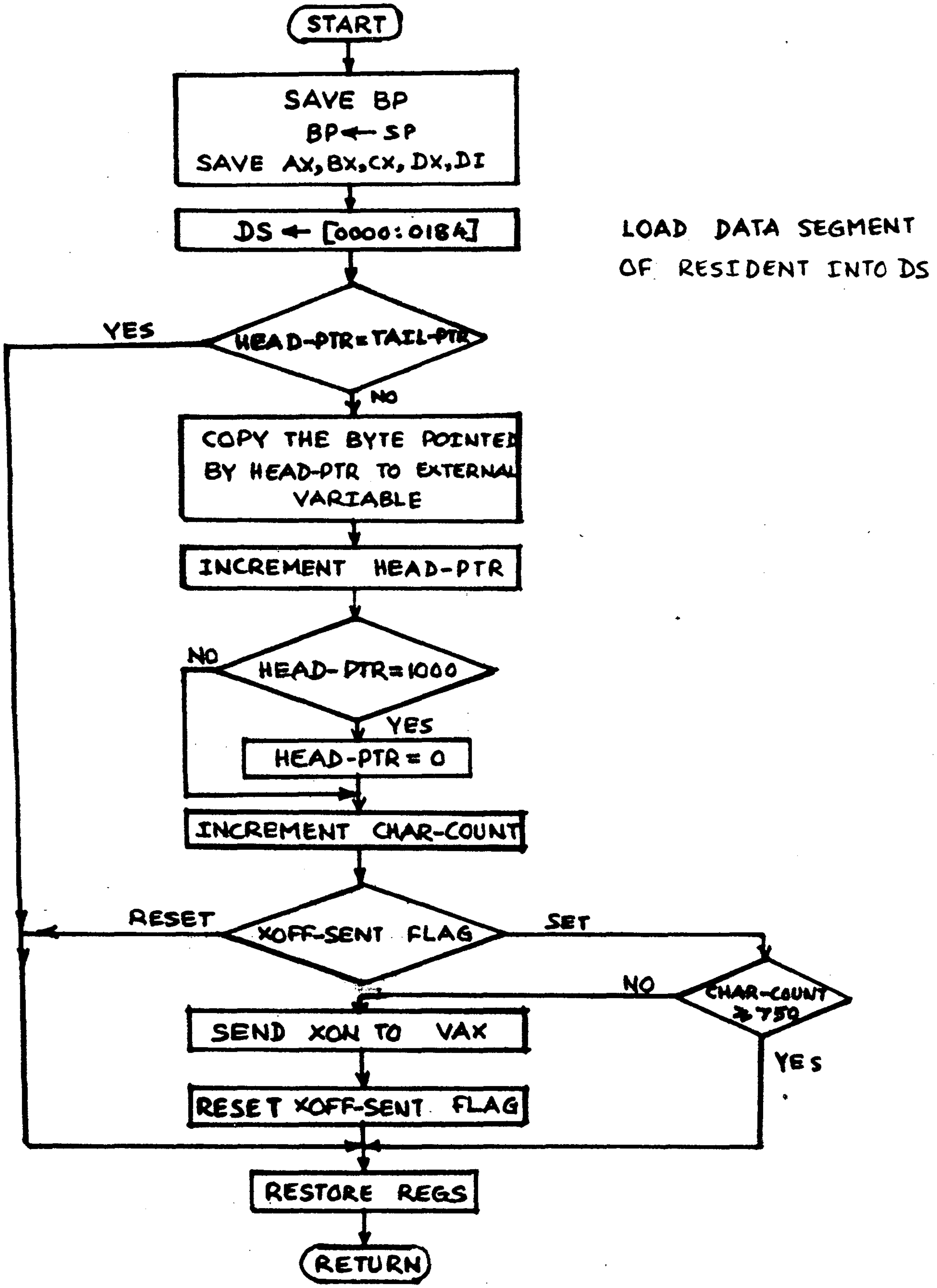


FIG 4.8

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.

Other procedures used in **CONNECT** :

.The program **CONNECT** contains many procedures.

Cursor addressing in TURBO pascal is relative with respect to a predefined window.But the cursor addressing in VT220 is absolute.In order to come over this problem,two procedures are defined,one to position the cursor and the other to find the cursor position.

FINDCUR : Finds the position of the cursor by loading 03 into AH, 00 into BX and executing interrup 10H.The column number is contained in DL and the row number in DH. Row number varies from 0 to 24 and column number varies from 0 to 79 in PC,while they vary from 1 to 23 and 1 to 80 respectively in VT220,hence 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 : 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 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.This is mainly used in set_up.

SET_BAUDRATE : Calls the above procedure to display the baud rate in reverse video in set_up.

SET_UP : Enables the user to examine and alter the communication parameters.It also facilitates an online help for information on how to transfer files ,keyboard mapping etc.This menu is displayed when CONNECT is run or F3 is pressed.

It displays a title "TANVAX VER 1.0" (a name we gave to our package),folloed by SETUP, followed by a menu to choose from.The first option for examining and altering the communicaton parameters.The second option is default parameters for communication,the third is help and the fourth is to exit from SETUP.The cursor can be moved with the up and down arrows.It is mooved to the required option and ENTER pressed to exercise that option.Only options 2 and 4 will take the user out of SETUP.The first option displays the current communication parameters.It displays the current baudrate,the number of bits,parity and the number of stop bits and an EXIT to quit this menu.The cursor is taken to the appropriate option and ENTER is repeatedly pressed to choose the next available option.When we quit SETUP,the 8250 is programmed to correspond to this settings.The cursor is also programmed to a blinking block cursor.

The second option programs the 8250 to the default settings and quits.The third option gives the help.In order to get help,the user should have NETHELP file on his floppy.The help text is read from this file and displayed.The fourth option is to quit from SETUP.

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

READKBD : Waits till there is input from the keyboard,by repeatedly calling **GETKEY**.This procedure is called in SETUP and is necessary because we need both the code and the scancode of the pressed key.

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].

SCROLL_UP : Scrolls the screen up,by one column.Register AH is loaded with 6 and register AL with 1 (the no.of lines to scroll),register CX is loaded with the top of scroll region,register DX is loades with the bottom of the scroll region and interrupt 10H is executed.The top and bottom margins are encountered when he termianl receives a control sequence to set the window limits.

SCROLL_DOWN : Same as above except it scrolls down.Register AH is loaded with 07.

The above two procedures are used in scrolling a window,which is mostly used in editor.

READMEM : Reads a character from the display memory at the given cursor position.Reg.AH is loaded with 08,reg.BX with 0 and interrupt 10H is executed.Reg AL contains the character.This is used in inserting and deleting text.

GETINT : To facilitate easy recognition of control sequences,this procedure is designed.For example,to position the cursor,the different combinations VAX can send are -

ESC [h --> position the cursor at 1,1

ESC [;10h --> position the cursor at row 1 and col 10

ESC [9;h --> position the cursor at row 9 and col 1

ESC [5;13h --> position the cursor at row 5 and col 13

In general,the sequence is ESC [n1;n2h where n1 or n2 or both can be missing,in which case they shold be taken as 1. Also note that n1 and/or n2 can be either a single digit or a two digit number.This procedure walks over all these problms.See flowchart 4.9. After encountering ESC and [,this procedure is called.It checks the next character in the buffer.If it is not an integer,it returns -1 and that character. If the first character is an integer(n1),it reads the next character.If this character is not an integer,it returns n1 folowed by this character.If the second charcter is also an integer,it computes the integer value out of these characters i.e. n1*10+n2, then reads the next character and both the computed value and the last character read are returned.When the prtogram calls **GETINT**,it checks the returned integer and character.Basing on these two,it takes the next step.See the flow chart in Fig 4.9.

REPORT : If the received control sequence does not match

PROCEDURE GETINT

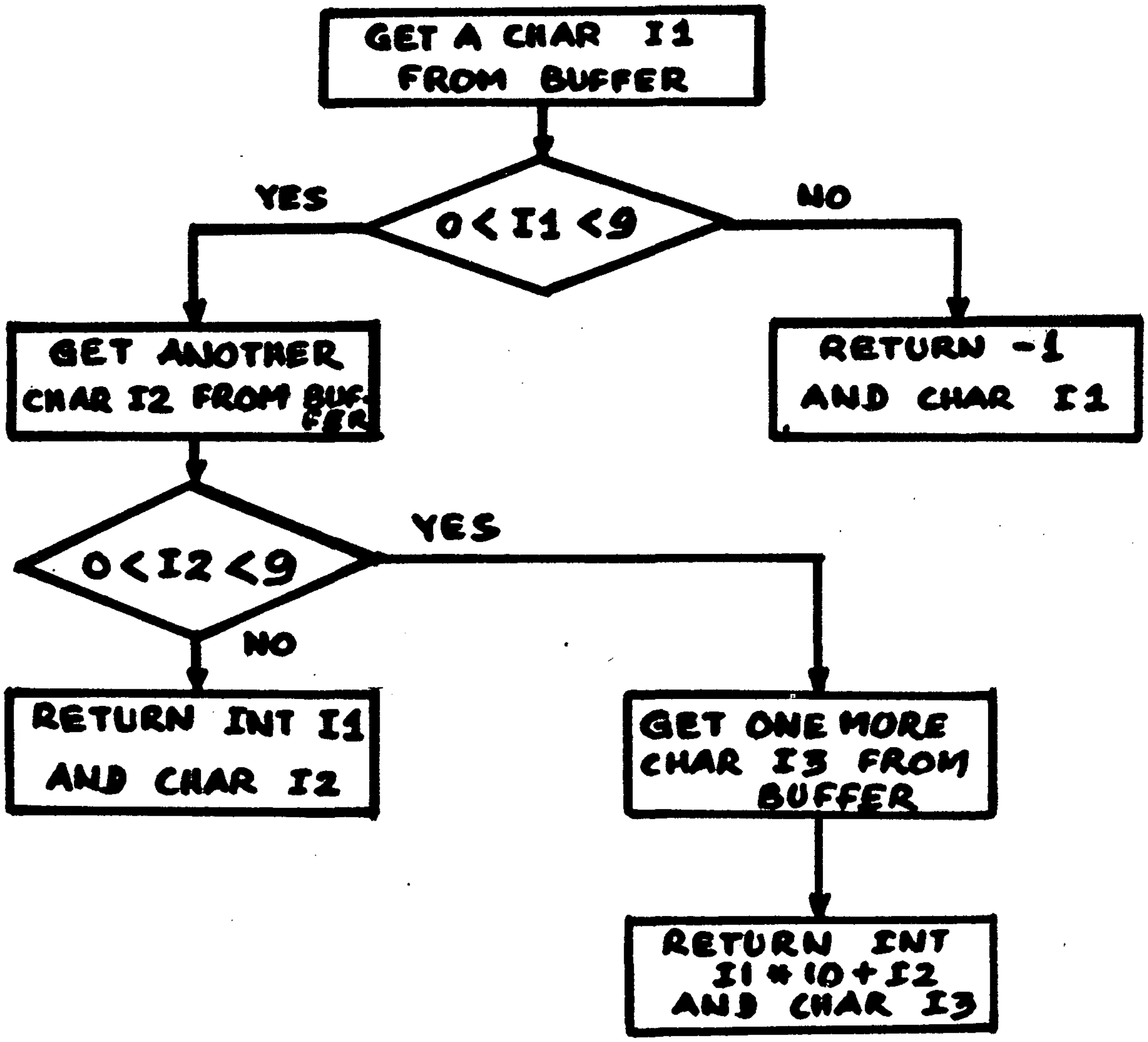

GET A CHAR I1 FROM BUFFER
0<I1<9
YES
NO
GET ANOTHER CHAR I2 FROM BUFFER
RETURN -1 AND CHAR I1
0<I2<9
YES
NO
RETURN INT I1 AND CHAR I2
GET ONE MORE CHAR I3 FROM BUFFER
RETURN INT I1*10+I2 AND CHAR I3

FIG 4.9

any of the routines provided,this procedure is called.It prints a message 'Unrecognized control sequence' followed by the sequence.This is usefull in future modifications and correcting errors where certain control sequences were not taken care of.

The main program CONNECT

The flow chart of the basic sequence in the main program was give in Fig 4.5.

Initialization : As described earlier,IR4 of 8259 is reset to allow interrupts.The communication port 8250 is programmed by calling set_up with the default parameters of 9600 baud rate,8 data bits,1 stop bit and no parity.Since a colour monitor provides changing the colors of the foreground and background,we gave an option to change these colors.**ALTf** changes the foreground color and **ALT b** changes the background color. The default colors are blue for background and megenta for foreground.

Key board mapping : Since there are many differences between the VT220 keypad and the TANDY1000 keypad,these keypads have to be mapped.Since the number of keys on the PC is less than that of VT220,some control and alter characters have to be used for certain key strokes.The basic layout of the VT220 keyboard is shown in Fig 4. 10 and that of TANDY1000 in Fig 4.11.

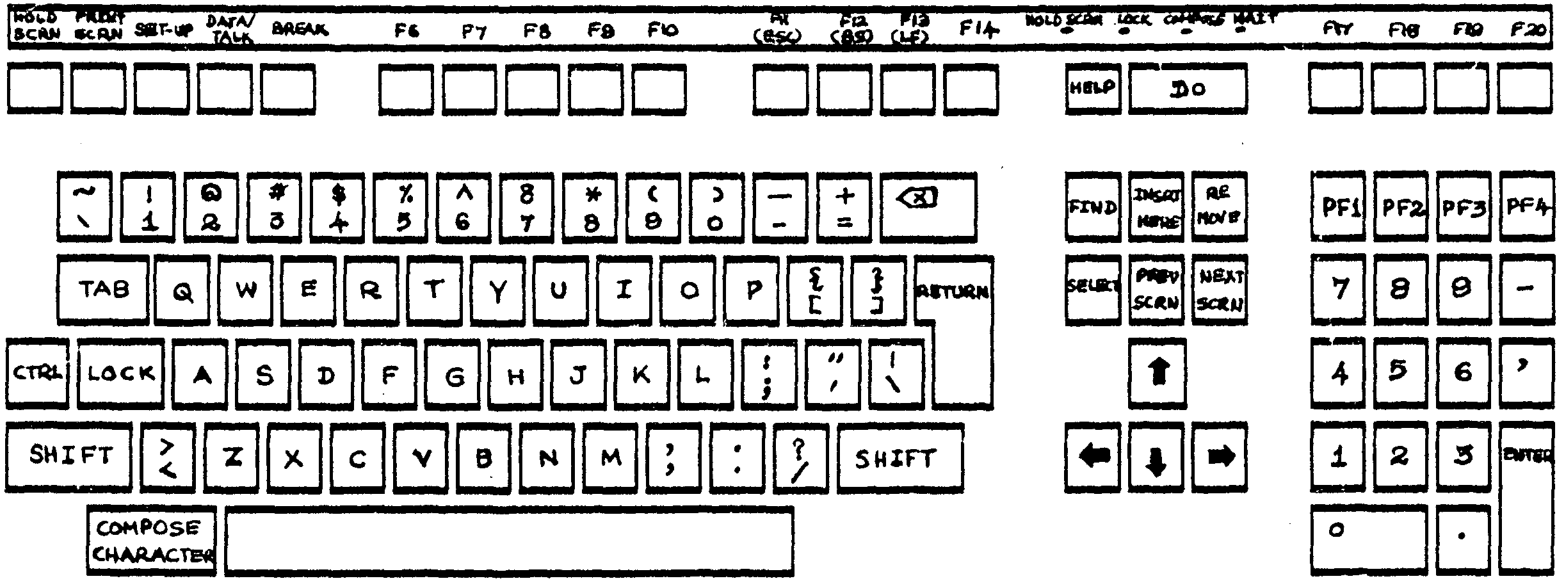


FIG 4.10 VT220 KEYBOARD

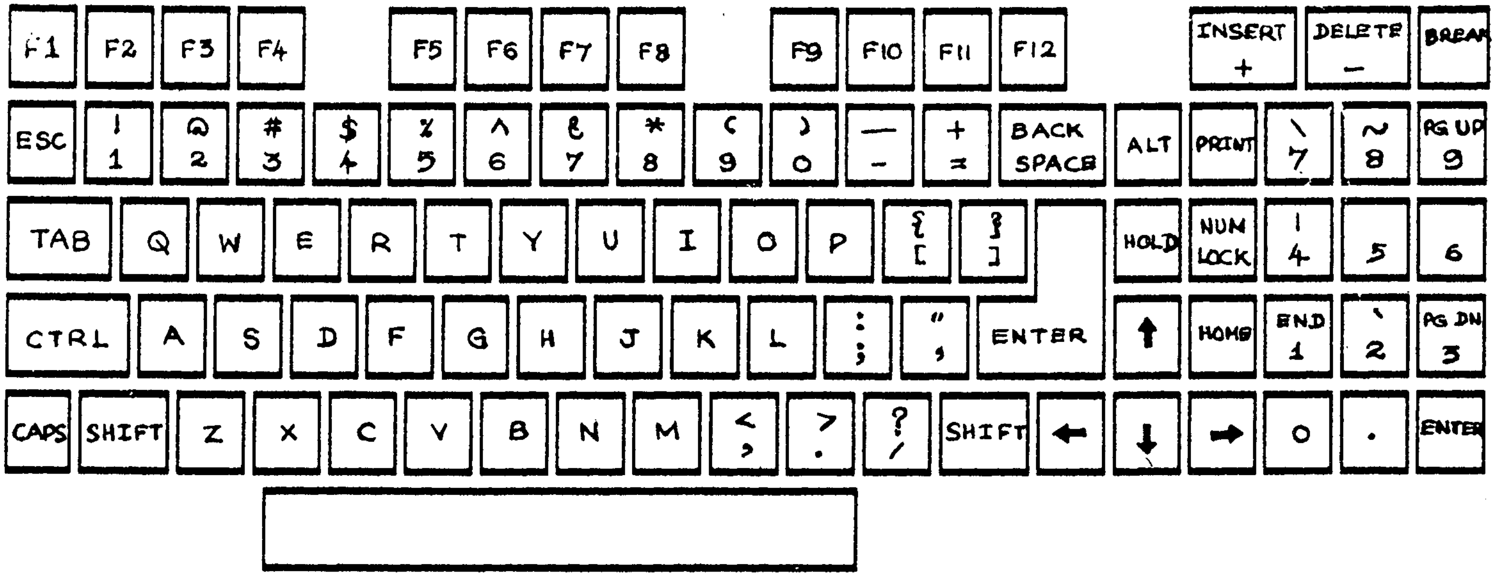

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 INSERT + DELETE − BREAK
ESC ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — - + = BACK SPACE ALT PRINT \ 7 ~ 8 PG UP 9
TAB Q W E R T Y U I O P { [}] HOLD NUM LOCK | 4 5 6
CTRL A S D F G H J K L : ; " , ENTER ↑ HOME END 1 ` 2 PG DN 3
CAPS SHIFT Z X C V B N M < , > . ? / SHIFT ← ↓ → 0 . ENTER

FIG 4.11 TANDY1000 KEYBOARD

For the numeric keypad, the VT220 sends one code when in normal mode and another code when in application mode. This mode is set by an escape sequence. See Appendix C for the codes sent by the VT220 for each key.

The mapping of the two keypads is given below -

MAIN KEYPAD : The keyboard of TANDY1000 is almost equivalent to the main key board (North american) on VT220, exept that

1. There is no compose key on TANDY.

2. The characters \, `. | and ~ are mixed with the numeric keypad on TANDY.

3. ⌫ key on VT220 is mapped with the back space on TANDY1000. Back space produces the code 08, but 7FH (erase character) is sent as is done by VT220.

Rest of the keys produce the same codes and are sent as it is both in normal mode and application mode.

EDITING KEYPAD : These keys are used in editing. Since TANDY has lesser number of keys, the mapping is

VT220	TANDY1000
↑	↑
↓	↓
-->	-->

<--	<--
prev screen	^pg up
next screen	^pg dn
insert here	^insert
remoove	^delete
find	^home
select	^end

TOP ROW FUNCTION KEYS : Since TANDY has only 12 function keys,it can recognize only uptp F12 compared to upto F20 of VT220.

The mapping is-

VT220	TANDY1000
hold screen	F1
print screen	none
set up	F3
data/talk	none
break	none
F6 to F12	F6 to F12

AUXILIARY KEYPAD : This keypad produces the usual ASCII codes of the keys in normal mode. In application mode,they produce different codes,which are mainly used in editing.However TANDY produces the same code for the

ENTER key on the main keypad,and ENTER key1 on the auxiliary keypad.To distinguish between the two,the scan code is also considered.Each key has different scan code.When GETKEY is called,it returns a value,the lower byte of which contains the ASCII code and the higher byte contains the scan code.The complete list of codes and scan codes for the PC keyboard is given in Appendix D.

The mapping for the keypad is -

VT220	TANDY1000
PF4	BREAK
PF3	DELETE
PF2	INSERT
PF1	ALT INSERT
ENTER	ENTER
.	.
0	0
1	1
2	2
3	3
'	6
6	5
5	4
4	^4

KEYBOARD MAPPING

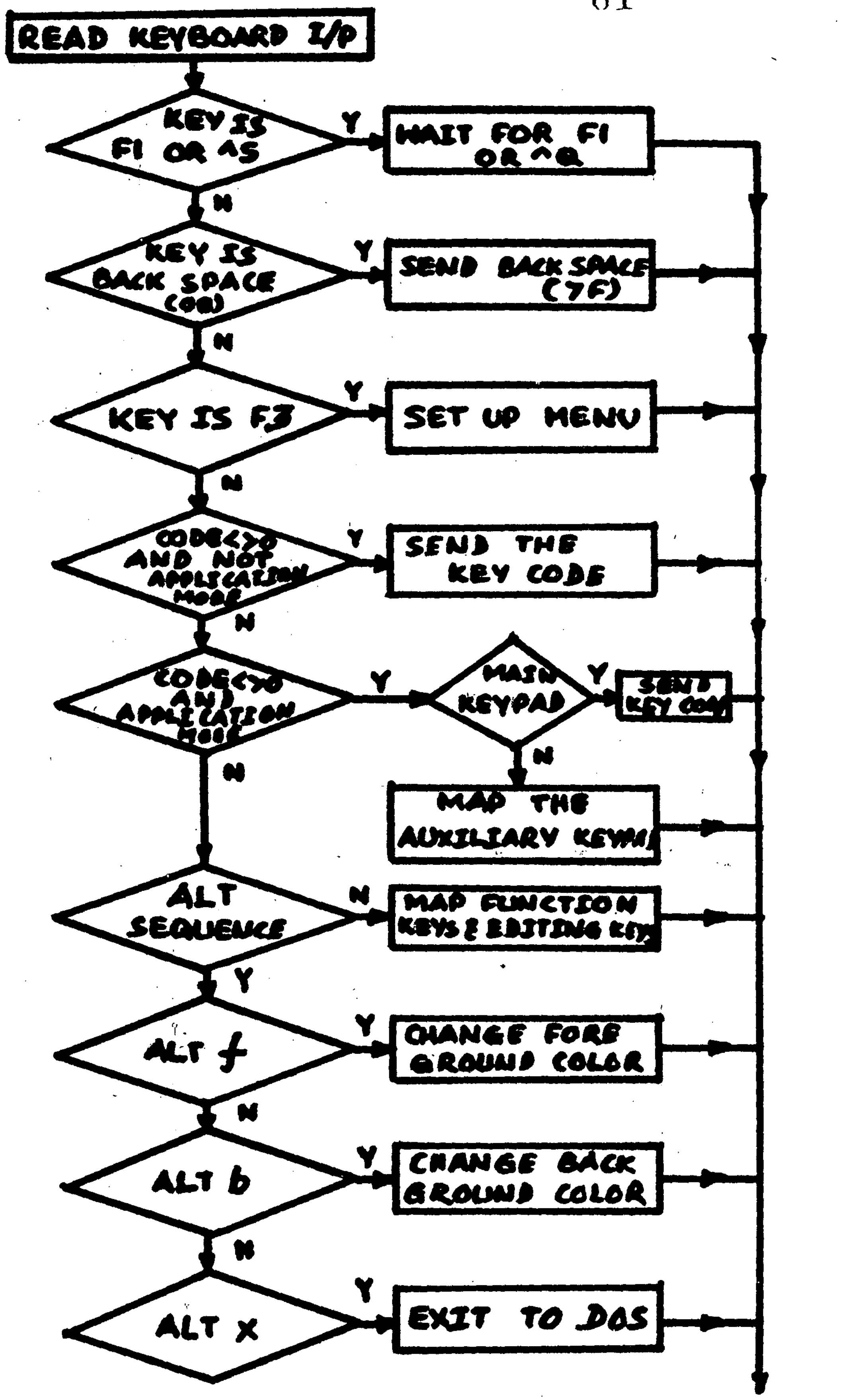


FIG 4.12

-	9
9	8
8	7
7	^7

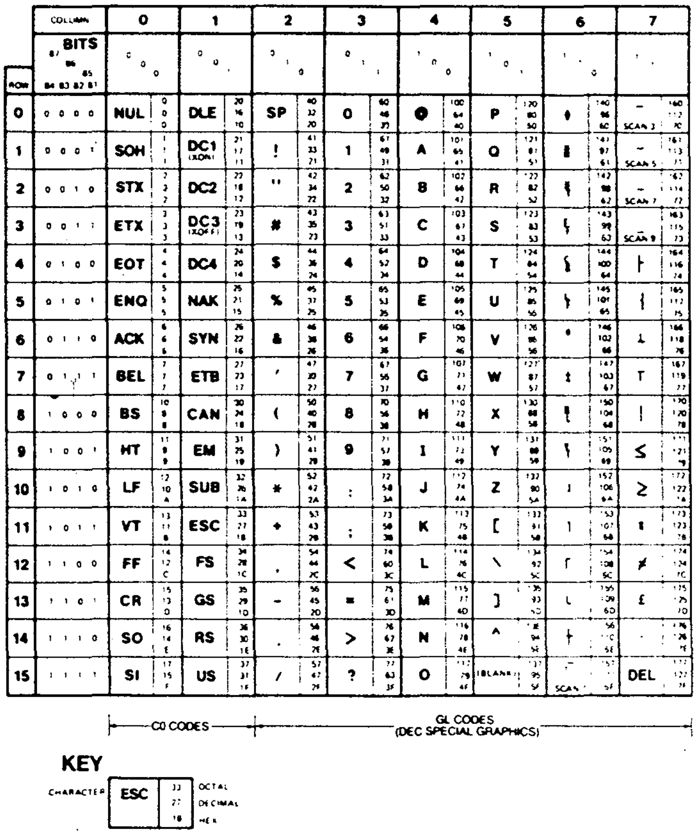
In mapping this keypad,the physical position of the key is given importance compared to the key number. For example,the key below PF4 erases one word from the display,similarly the key below BREAK does the same thing in TANDY.

NOTE : Since this keypad produces ASCII codes for digits in VT220,the NUM LOCH key should be set on TANDY,so that this auxiliary pad produces codes for numbers.However,while entering \,|,` and ~, the NUM LOCK should be reset,the required data entered and the NUM LOCK set again.The flowchart for the keyboard mapping is given in Fig 4.12.

PROCESSING DATA IN BUFFER : The next part is to read a character from the circular buffer and process it for displaying ordinary characters as well as graphic characters.VAX sends code 14 or the escape sequence ESC(D to instruct the VT220 to enter graphics mode.Similarly the code 15 or the sequence ESC(B resets the graphics mode.The DEC special graphics character set

6

DEC Special Graphics



	COLUMN	0		1		2		3		4		5		6		7	
ROW	BITS b7 b6 b5 b4 b3 b2 b1	0 0 0		0 0 1		0 1 0		0 1 1		1 0 0		1 0 1		1 1 0		1 1 1	
0	0 0 0 0	NUL	0 0 0	DLE	20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	◆	140 96 60	⎺ SCAN 3	160 112 70
1	0 0 0 1	SOH	1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	▒	141 97 61	─ SCAN 5	161 113 71
2	0 0 1 0	STX	2 2 2	DC2	22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	␉	142 98 62	⎼ SCAN 7	162 114 72
3	0 0 1 1	ETX	3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	␌	143 99 63	_ SCAN 9	163 115 73
4	0 1 0 0	EOT	4 4 4	DC4	24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	␍	144 100 64	├	164 116 74
5	0 1 0 1	ENQ	5 5 5	NAK	25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	␊	145 101 65	┤	165 117 75
6	0 1 1 0	ACK	6 6 6	SYN	26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	°	146 102 66	┴	166 118 76
7	0 1 1 1	BEL	7 7 7	ETB	27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	±	147 103 67	┬	167 119 77
8	1 0 0 0	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	␤	150 104 68	│	170 120 78
9	1 0 0 1	HT	11 9 9	EM	31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	␋	151 105 69	≤	171 121 79
10	1 0 1 0	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	┘	152 106 6A	≥	172 122 7A
11	1 0 1 1	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	┐	153 107 6B	π	173 123 7B
12	1 1 0 0	FF	14 12 C	FS	34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	┌	154 108 6C	≠	174 124 7C
13	1 1 0 1	CR	15 13 D	GS	35 29 1D	-	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	└	155 109 6D	£	175 125 7D
14	1 1 1 0	SO	16 14 E	RS	36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	┼	156 110 6E	·	176 126 7E
15	1 1 1 1	SI	17 15 F	US	37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	(BLANK)	137 95 5F	⎺ SCAN 1	157 111 6F	DEL	177 127 7F

C0 CODES

GL CODES
(DEC SPECIAL GRAPHICS)

KEY

CHARACTER	ESC	33 27 1B	OCTAL DECIMAL HEX

FIG 4.13

is given in Fig 4.13. Since the ASCII characters between 80H and FFH are used for displaying graphic characters in a PC, the DEC special graphic characters are mapped with these characters.See appendix E for a complete table of ASCII codes.The flow chart for escape amd control sequence identification and that of graphic charcater recognition is given in Fig 4.14. The implementation of escape and control sequences is mostly self explanatory from the program.

Among the important ones are -

Displaying characters with an attribute :

When VAX sends the control sequence ESC[7m , the characters have to be displayed in reverse video.Similarly it sends ESC[5;7m to display the following cahracters in blinking reverse video mode.

MSDOS interrupt 10H provides a facility to display a character in given mode.The procedure diaplay uses this facility.The table below lists the options available on PC for the varios attribute values.

attribute	effect
00	No display

PROCESSING DATA IN BUFFER

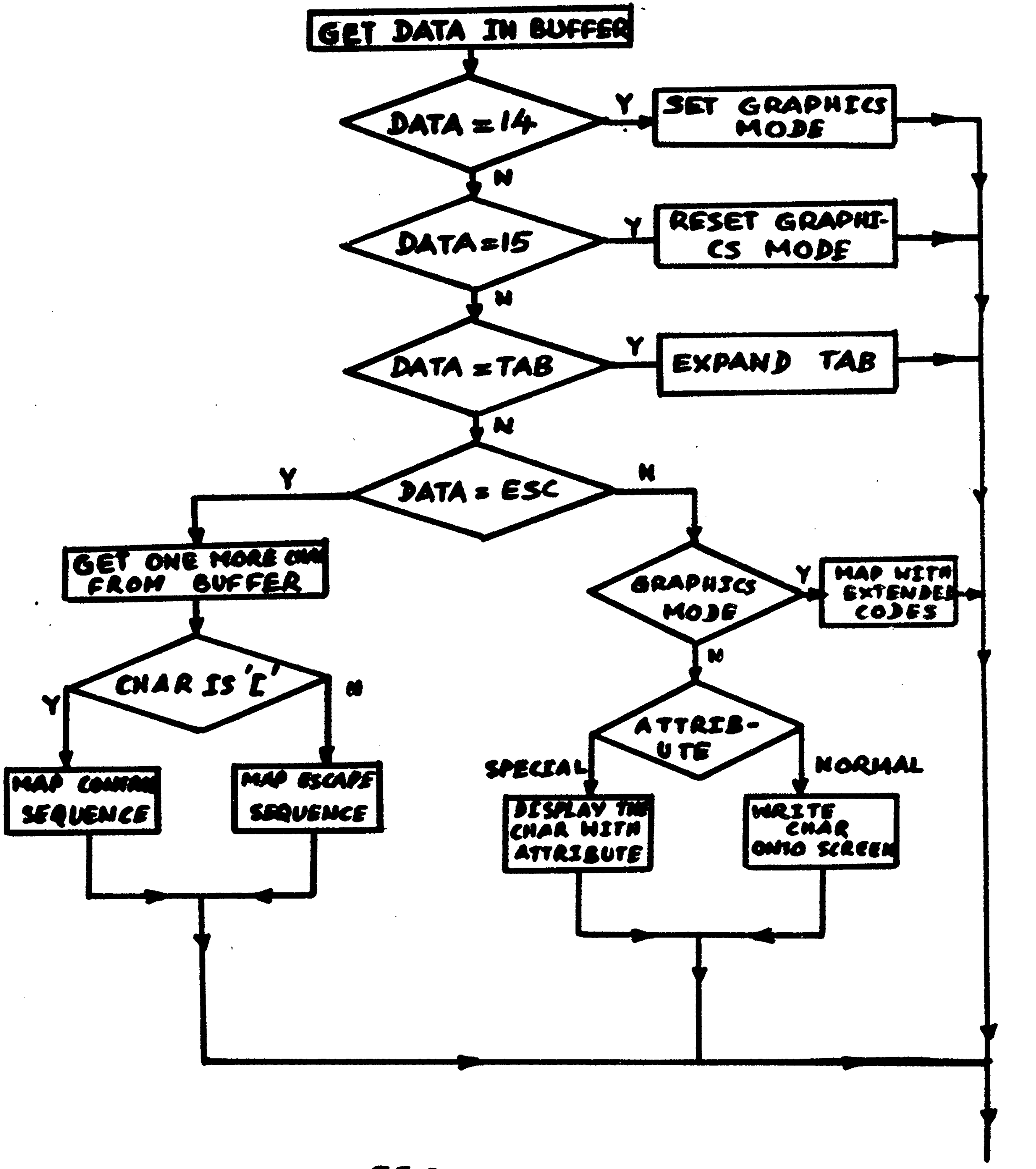


FIG 4.14

01	underlined characters
07	normal characters,white on black
08	no diaplay
09	underlined high intensity char.
70	reverse video
71	underline only,not reverse video
78	reverse video,high intensity
79	high intensity underlined
81	blinking underlined
87	blinking noraml
89	blinking underlined highintensity
F0	blinking reverse video
F8	blinking high intensity reverse video

When the attribute is other than zero,it has to dispalyed in a mode other than normal.If the rerceived attribute is 7,then display is called with mode=70.If the received attributes are 5;7, then the mode is set to F7H and display is called.This attribute byte is reset to zero when ESC[m or ESC[0m is received.

Defining windows :

When the control sequence ESC[3;22r is received,the window has to be set from row 3 to row

22.This is accomplished with window command in TURBO pascal.The variables wintop and winbottom are assigned to the top row and bottom row respectively.These variables are usefull when deciding whether to scroll the display or not.Every time a window is defined,the cursor is positioned at home.

Inserting and deleting characters :

If a character has to be inserted in a row, at a column,then the present cursor position is saved,the string of characters from the current cursor position to the 80th column are read into a buffer by repeatedly calling **READMEM**.This string is reexamined and blank characters on the right side are discarded.The cursor is positioned at the same place,the new characters is inserted and the string is written back from the next cursor position.The cursor is restored.

Similarly for deleting a character,characters from the next cursor position are read into the string and put back from the current cursor position.The last character is erased and the string is flushed.

Coming back to DOS :

Since we chose to insert our own escape sequences,the escape sequence **ESCq** is introduced to abort **CONNECT** and return back to DOS.A program TODOS on VAX,when run sends ESCq. **CONNECT** traps this and gives the user an option either to logout or still remain in session.Then **CONNECT** is aborted.

The input **ALTx** also aborts **CONNECT**,but disconnecting by running **TODOS** is more advisable,in order to avoid disconnecting while in editor or some application program.

V . FILE TRANSFER UTILITIES

Since we could establish a connection between PC and VAX,the next step naturally is to exchange information from one system to the other.This chapter describes the protocols and utilities for transferring files from PC to VAX and fron VAX to PC.Error free file transfer is the backbone for other utilities like mail.The mail and phone facilities are provided in PC to PC communication.

This error checking for communication between two computers is the job of the data link layer in a computer network,which is normally provided in hardware.Since this hardware is not available in PC,it is implemented by software.

The basic idea in file transfer is as follows-

A program on the PC reads characters from a file and transmits them over the communication line.Another program on VAX receives the characters and stores them in a file.If the above scheme works well,without loosing any information,it would have been very simple.However real world communication circutes have the nasty property of making errors now and then,which causes transmitted data to get mangled or even get lost altogether.Furthermore,when data is transmitted continuosly at a baud rate of 9600 it is very likely that the charecters will be lost,because the reciever has only a limited buffer space.To avoid this,we have to devide the data into frames,send them and make sure that they are recived properly at the other node .

5.1 DATA LINK PROTOCOLSP

Let us consider that a file is to be transefered from PC to VAX. A program in PC reads data from the file.This data is devided into frames.The receiving program receives the frame,checks if the check sums and the sequence numbers match.If they match,an acknowledgement followed by the sequence number is sent,otherwise a negative acknowledgement followed by the sequence number is sent.A frame consists of severel fields.

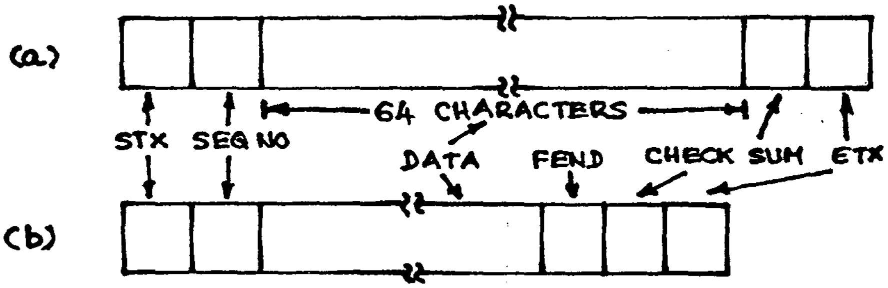


Fig 5.1

The first field is the **STX** (Start of text) which is a control information to the receiver that a new frame is following.The second field is the **SEQNO** which tells the number of frame currently being transferred.This is necessary to detect missing frames or duplicate frames.Then follows the data.Our frame size is set to 64 bytes.If there are as many characters in the file as the framesize,then all the 64 characters are put in the frame(Fig5.1a).However if an end of file is encounterd before 64 characters,an **FEND** (End of file) character is put after the data(Fig5.2).Then a **check** sum is put in the frame followed by an **ETX**(End of text) character.The file transfer is complete,if zero or more frames of type a folled by one frame of type b are received at the other node.

5.2 Calculating the checksum : In practice,cyclic redundancy code(CRC) method is used for error checking.However constructing and checking the CRC checksum is either provided in the hardware or carried out with a single machine code instruction(like the CRC in VAX).However the pc has neither the hardware to support generation and checking of CRC check sums,nor a single instruction to do the same.If this were implemented in software,it would be inefficient,since the PC is much slower than VAX.

Since in our case,the actual problem is that characters are lost sometimes,but never mangled.Hence a simpler method is followed for computing the checksum.Before proceeding with the construction of a frame,a variable **CHECK** is initialized to zero.Eveytime we read a character,we exclusive or this character with check.Then this character is put in the buffer.After we read 64 characters or encounter an end of file,the contents of check becomes the checksum.Similarly on the receiving end,chech sum is calculated after receiving each character.If these two checksums match,data is not lost.However,this checksum can be any number between 00 to 127.ASCII values 00 to 31 are special characters.These cahracters,if sent as they are,will be interpreted by the operating system.Hence when the checksum is less than 31,the value 31 is added to checksum,so that the checksum is always greater than 31.Similarly the character 121 is the BACKSPACE character.When sent,the previous character in the input stream is erased.To avoid this,the checksum is changed from 127 to 126.A similar transformation is carried out at the other end.This protocal will fail if -

1. An even number of characters change in the same bit position.

2. The check sum on the sender side turned out to be X ($X < 31$) and the checksum on the receiver turned out to be X + 31.Hence 31 will be added to X on the sender side,and X + 31 will be sent.The checksum at the receiver is already X + 31 and they do match and it will be presumed that the

frame is correctly received,though there is an error.

But the above conditions rarely occour.

SPECIAL CHARACTERS : The **STX,ETX** and **FEND** characters can not be ordinary text characters,otherwise they get mixed up with the data.One way to avoid the problem is to use character stuffing.We chose to do in another way.These **STX,ETX** and **FEND** characters are predefined as equivalent to certain control characters.Since control characters can not be part of a text file,we can safely send them.However we should see that the operating system should not trap them.

One more problem remains to be solved.The terminal controller of VAX is normally programmed to echo back whatever is sent from the terminal.Hence characters sent from the PC will be echoed back.However we want files to be transferred from PC to VAX,they need not be echoed back.In fact,they should not be echoed back,lest they should interfere with acknowledgements and frame numbers,that we will be receiving from the other side.To avoid the above two problems,we execute the VMS command

```
$ set terminal/noecho
```

When this command is executed,not only that the characters are not echoed back,certain control cahracters are also handed over to the user program,without the operating system traping them.

INITIALIZING THE FILE TRANSFER SESSION

One way to transfer a file from PC to VAX is to run a program(FTPC) on PC to send frames and process acknowledgements, and another program(FTVAX) on VAX to receive these frames and process them.To proceed this way,one has to get a connection to VAX on the PC,run the program PCTOVAX,then disconnect to come back to MSDOS and run the other progarm FTPC.Both programs will be running and the file will be transferred.However we chose to do it in another simpler way.Since we developed our own code to process escape sequences,we introduced our own escape sequences.The user gets a connection to VAX on the PC by running **CONNECT**.If a file is to be transferred from PC to VAX,the user runs a program **PCTOVAX** on VAX.This program will send the escape sequence **ESCT** and then will wait to receive frames.The program **CONNECT** traps this sequence,and then runs it's own program for file transfer,which constructs frames,sends them and procersses the acknowledgements.This program is a part of **CONNECT** and in fact is a subroutine.This is possible since **ESCT** is not one of the escape sequences sent by VMS.This procedure runs on PC and

PCTOVAX runs on VAX.Once the file transfer is complete,we are back to the VMS prompt.Similarly the file transfer program from VAX to PC is initiated by a progarm **VAXTOPC** on VAX.It sends the escape sequence **ESCF**.

5.3 IMPLEMENTATION DETAILS

Since file transfer from PC to VAX and VAX to PC are almost similar,we describe here two algorithms,one the file sender and the other file receiver.For file transfer from PC to VAX,the sender program runs on PC and the receiver program runs on VAX.For file transfe from VAX to PC,the progarms will be interchanged.The flowchart for file transfer sender is given in Fig5.2 and the flowchart for file receiver is given in Fig5.3.

5.3.1 FILESENDER :

This program first reads the name of the file to be transferred.If the file is availabe,it will send character C, instructing the receiver to continue,otherwise it will send the character Q,instructing the receiver to abort and gives a message that the file is not found.When a file is sent from PC to VAX,a file with the same file name will be created on VAX.However,the drive specification that

FILE TRANSFER - SENDER

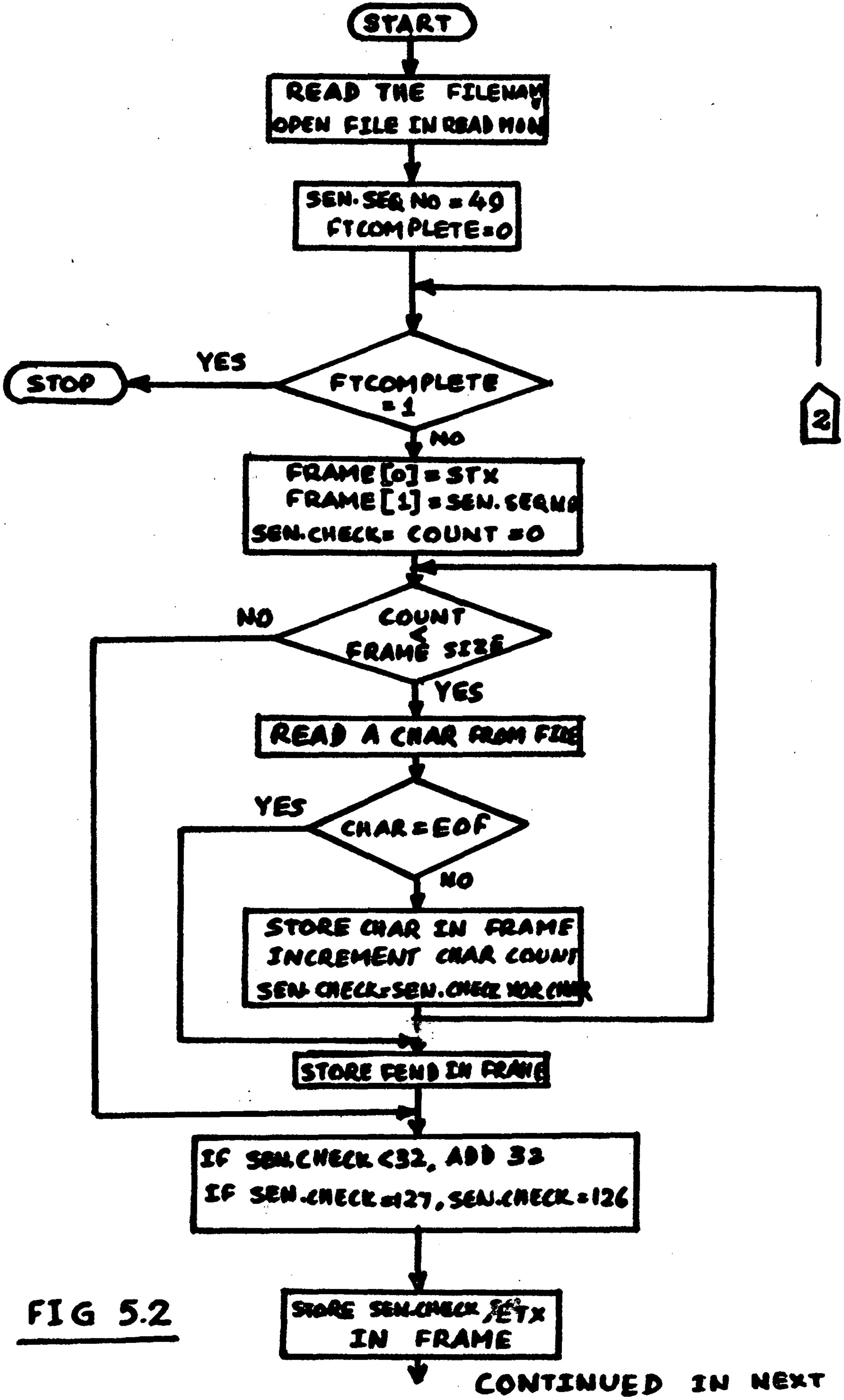


FIG 5.2

CONTINUED IN NEXT PAGE

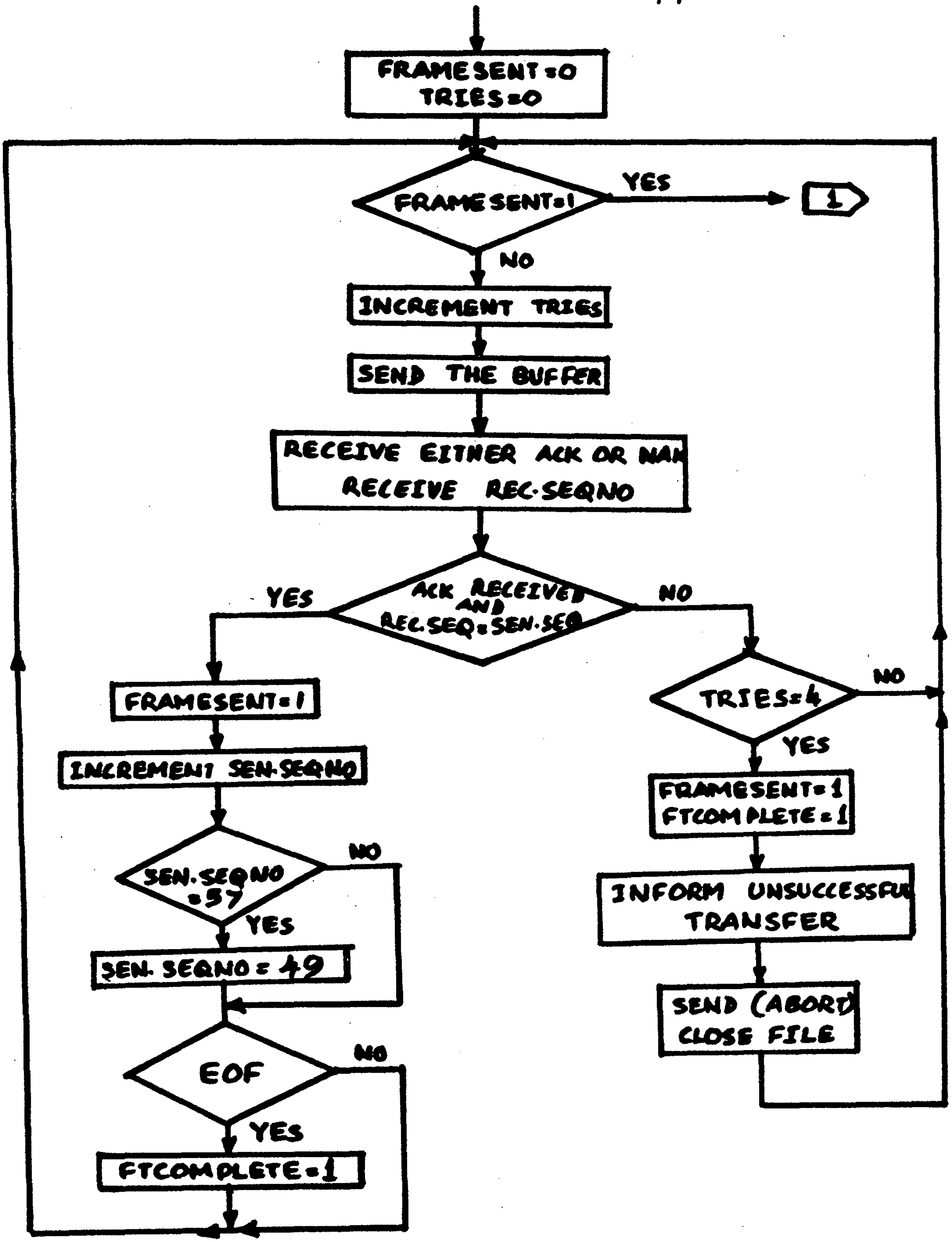

FRAMESENT=0
TRIES=0
FRAMESENT=1
YES
1
NO
INCREMENT TRIES
SEND THE BUFFER
RECEIVE EITHER ACK OR NAK
RECEIVE REC.SEQNO
ACK RECEIVED AND REC.SEQ=SEN.SEQ
YES
NO
FRAMESENT=1
INCREMENT SEN.SEQNO
SEN.SEQNO =57
NO
YES
SEN. SEQNO = 49
EOF
NO
YES
FTCOMPLETE=1
TRIES=4
NO
YES
FRAMESENT=1
FTCOMPLETE=1
INFORM UNSUCCESSFUL TRANSFER
SEND (ABORT)
CLOSE FILE

FIG 5.2 (CONTD)

might have been given for the filename on the PC will be discarded.However when a file is transferred from VAX to PC,both the filename on VAX and the filename on the PC are read.This is necessary because -

1. The user may wish to give the drive specification,when giving the filename on PC.

2. The user may give a different filename on PC so that,if a file with the same name as that of VAX may not get erased during file transfer.This is however not a problem when file is transferred from PC to VAX,since VMS creates a file with a new version number.

Check is initialized to zero and the sequence number of the farme is initialized to 49 (character '1'). The sequence number is incremented after a frame is successfully received at the other end.However this increment will be modulo 9,so that after sequence number 8,the next sequence number will be 1.

The first character in the frame is STX,the second character is the sequence number.Characters are read in one by one,till either end of file is encountered or 64 characters are read.Before the character is put in the ferame,it is exclusuve ored with the check.Since VMS stores both a carriage return and a line feed when it receives a carriage return,line feeds with in the file on PC are discarded in file transfer fron PC to VAX.If an end of ile is encounterd,an FEND is put in the frame.Then the computed checksum is put in the frame,followed by the ETX character,signalling the end of the frame.This frame is then sent to the receiver,which processes it and sends acknowledgement followed by the sequence number.If an acknowledgement followed by the sequence number is received with in certain time limit,and the sequence numbar received matches with the sender's sequence number,then the frame is successfully transferred.It will increment sequence number.If an end of file was encountered earlier,file transfer is complete.Otherwise it will construct the next frame and repeat the process again.If it did not receive the acknowledgement and sequence number within certain time limit,or it received a negative acknowledgement or the sequence numbers are matching,then the frame is not properly received at the other node.The frame will be transmitted again.It will try a maximum of four times,and if it could not succeed,signals an unsuccessfull transfer and aborts.

5.3.2 FILE RECEIVER :

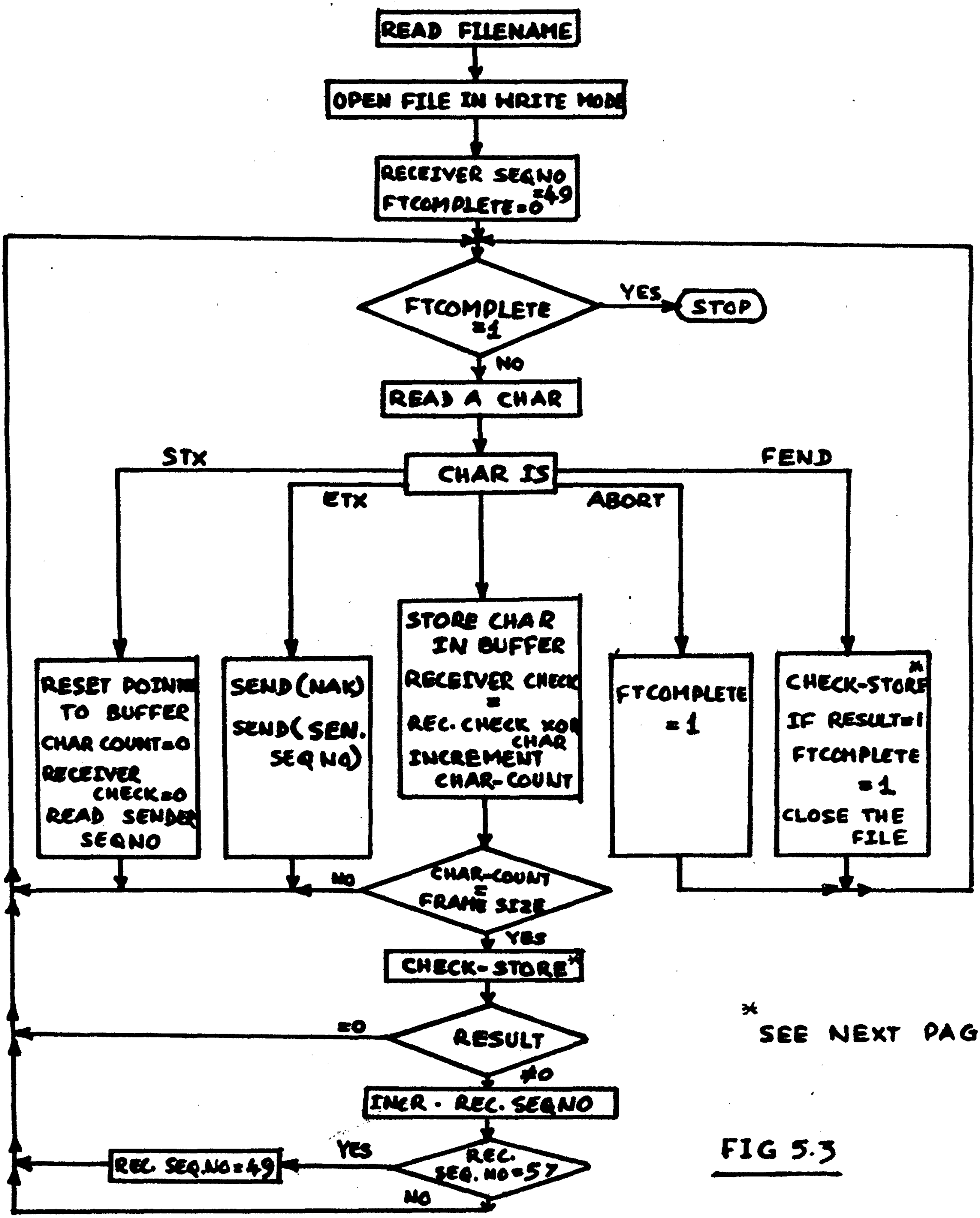
This program reads the filename and creates the file with that name for writing.If it could not create a file,it will send a message to the sender to abort the file transfer.Then it will initilalize it's sequence number to '1'.When it receives characters,it takes the appropriate action depending on what is received.

If an **STX** is received,it means a new frame is following.So it will flush it's internal buffer,resets it's checksum and character_count and reads the next character which is the sequence number of the sender.

If an **ETX** is received, it is an error.Since **ETX** can not be encountered before 64 characters,a negative acknowledgement followed by the sender's sequence number is sent.

If an **FEND** is received,it means that an end of file is received at the sender side and the current frame is the last frame.The next character will be checksum and the next will be **ETX**.It will compare the checksums and the sequence numbers.If they match,an acknowledgement followed by the sequence number is sent.The file trsnsfer is complete,hence the file is closed.If the checksums do not

FILE TRANSFER -RECEIVER



* SEE NEXT PAGE

FIG 5.3

FUNCTION CHECK-STORE

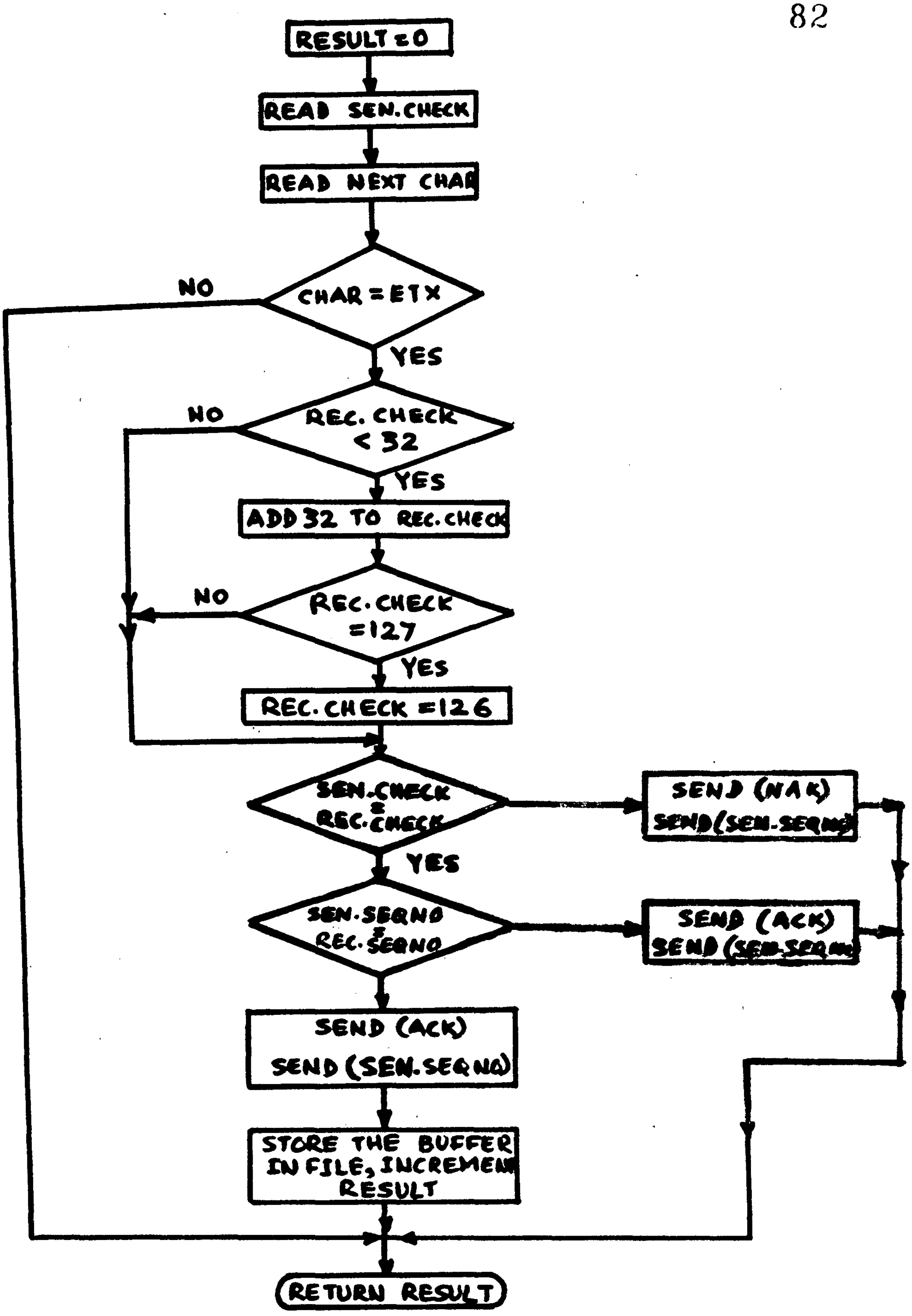


FIG 5.3 (CONTD)

match,a negative acknowledgement followed by the sender's sequence number is sent.If the sequence numbers do not match,this is a duplicate frame,hence an acknowledgement followed by the sender's sequence number is sent

If an **ABORT** is received,it means that the sender sent the same frame four times unsuccessfully and the file transfer is to be aborted.The file is closed and the program is aborted.

If the character is none of the above,it should be data.It is stored in it's own frame,calculates it's own checksum,and the charactercount is incremented.If 64 characters are received,the next character is checksum of the sender and the next is **ETX**.If the checksums and the sequence numbers match,the frame is correctly received and is stored in the file.The sequence number is incremented and an acknowledgement followed by the sequence number is sent.If the checksums do not match, a negative acknowledgement followed by the sender's sequence number is sent.If the sequence numbers do not match,this is a duplicate frame,hence an acknowledgement followed by the sender's sequence number is sent.

5.3.3 TRACING ERRORS :

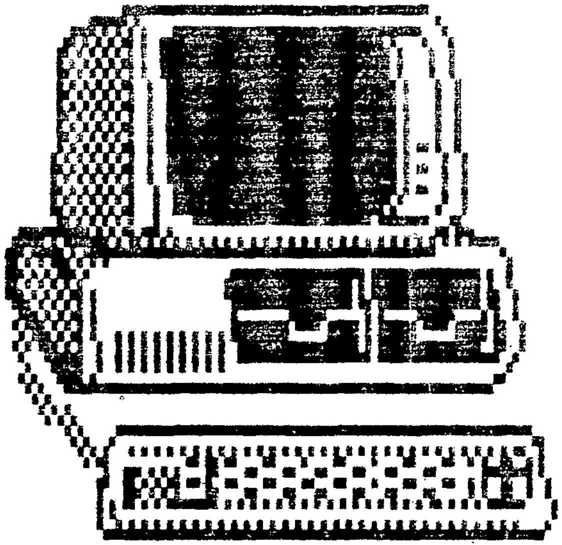
Let us consider two examples as to how errors are detected.

a. Suppose that one character gets lost in the transfer -

The checksum of the sender will be received as the 64'th character and **ETX** will be received as the chechsum.This is an error and will be trapped.A negative acknowledgement will be sent.Also note that the checksums do not match.

b. The acknowledgement sent by the receiver gets lost -

The receiver has correctly received the frame and sent the acknowledgement followed by the sequence number.It increments it's sequence number.However,the acknowledgement got destroyed in between.The sender will wait for the acknowledgement for a finite time,then time out and sends the frame again.Since it could not receive the acknoeledgement,it will not increment it's sequence number.Hence a duplicate frame will be received at the receiver.This will be traced,since the sequence numbers do not match.The receiver then sends an acknowledgement followed by the sender's sequence number.



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 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 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 X X X X X 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 X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, source-string (repeat) dest-string, source-string	22 9 + 22/rep	2 2/rep	1 1	CMPS BUFF1, BUFF2 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 X X X X X 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 U X X X X 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. Mnemonics © Intel, 1978.

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

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

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

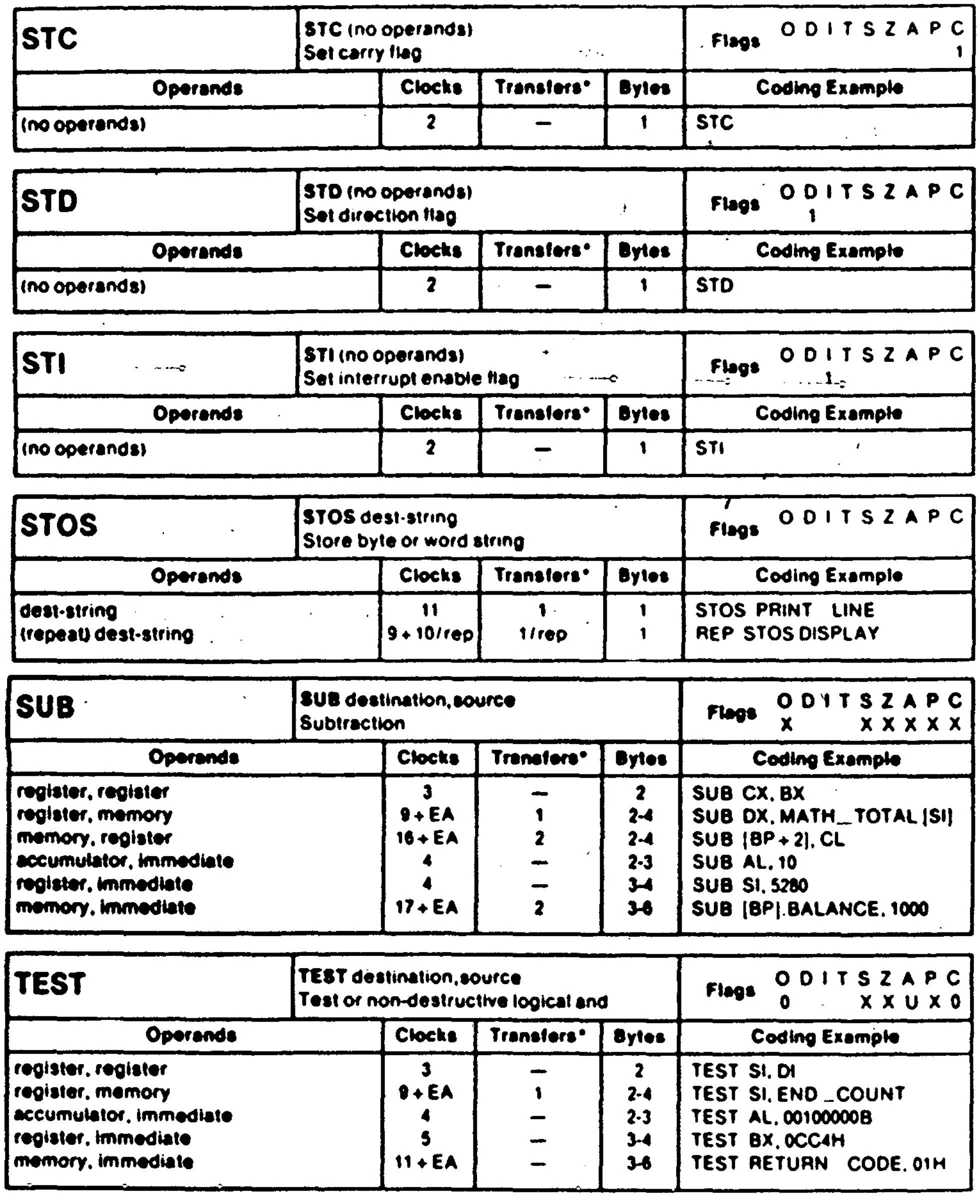
Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	SHR SI, 1
register, CL	8 + 4/bit	—	2	SHR SI, CL
memory, 1	15 + EA	2	2-4	SHR ID BYTE |SI||BX| 1
memory, CL	20 + EA + 4/bit	2	2-4	SHR INPUT WORD, CL

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

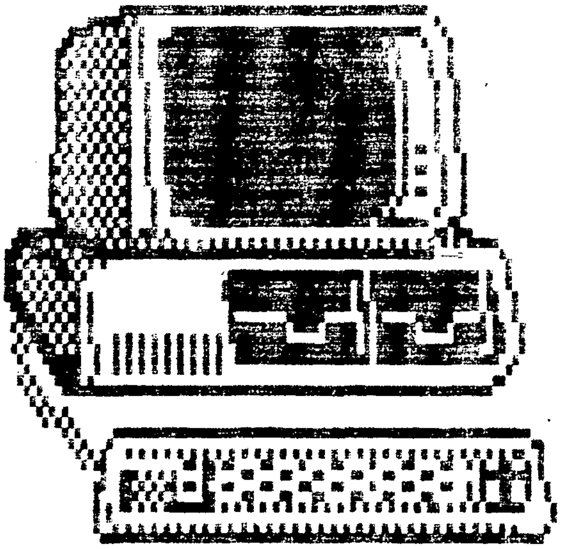
Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	TEST SI, DI
register, memory	9+EA	1	2-4	TEST SI, END_COUNT
accumulator, immediate	4	—	2-3	TEST AL, 00100000B
register, immediate	5	—	3-4	TEST BX, 0CC4H
memory, immediate	11+EA	—	3-6	TEST RETURN CODE, 01H

WAIT	WAIT (no operands) Wait while $\overline{\text{TEST}}$ 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

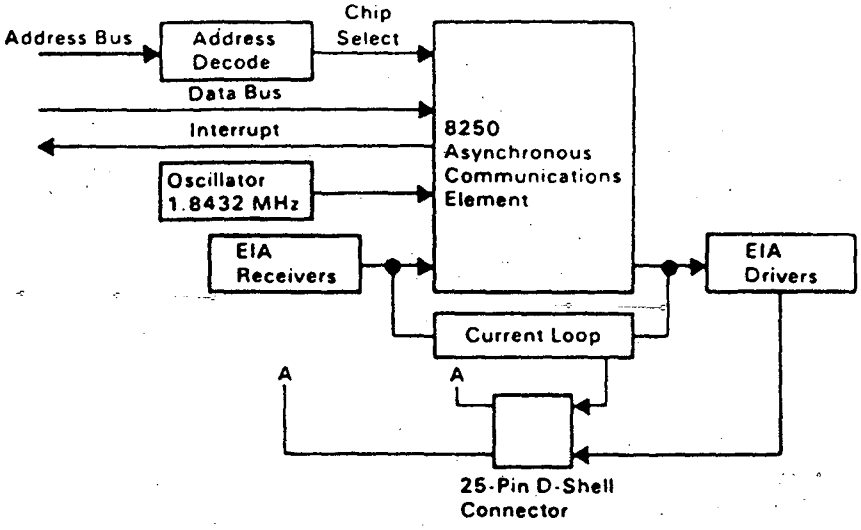
The asynchronous communications adapter system control signals and voltage requirements are provided through a 2 by 31 position card edge tab. Two jumper modules are provided on the adapter. One jumper module selects either RS-232C or current-loop operation. The other jumper module selects one of two 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)			
Primary Adapter	Alternate Adapter	Register Selected	DLAB State
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).

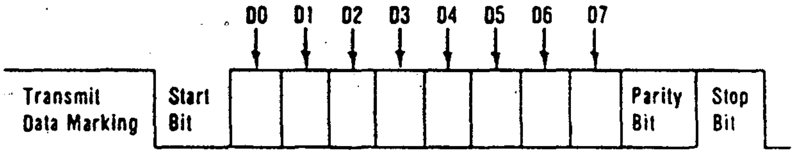
A2, A1 and A0 bits are "don't cares" and are used to select the different register of the communications chip.

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:



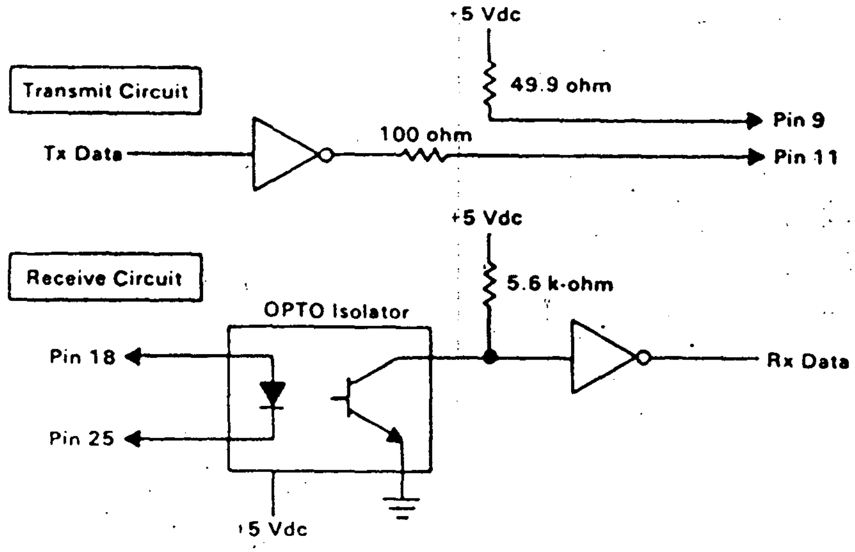
Data bit 0 is the first bit to be transmitted or received. The adapter automatically inserts the start bit, the correct parity bit if programmed to do so, and the stop bit (1, 1-1/2, or 2 depending on the command in the line-control register).

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

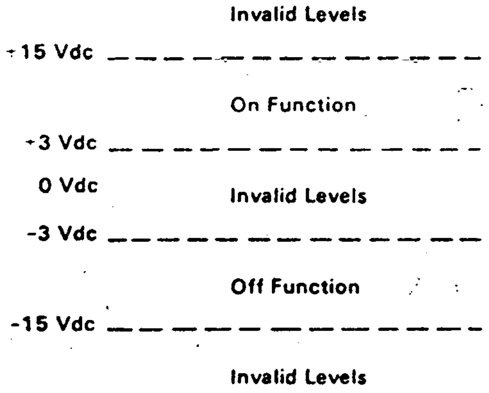
The voltage interface is a serial interface. It supports certain data and control signals, as listed below:

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

'he adapter converts these signals to/from TTL levels to EIA oltage levels. These signals are sampled or generated by the ommunications control chip. These signals can then be sensed y the system software to determine the state of the interface or eripheral 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



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.

DOS

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

ADS : 0

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 ($\overline{\text{RI}}$), Pin 39: When low, indicates that a telephone ringing signal has been received by the modem or data set. The $\overline{\text{RI}}$ 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 $\overline{\text{RI}}$ 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 ($\overline{\text{DTR}}$), 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 $\overline{\text{DTR}}$ signal is set high upon a master reset operation

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

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

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

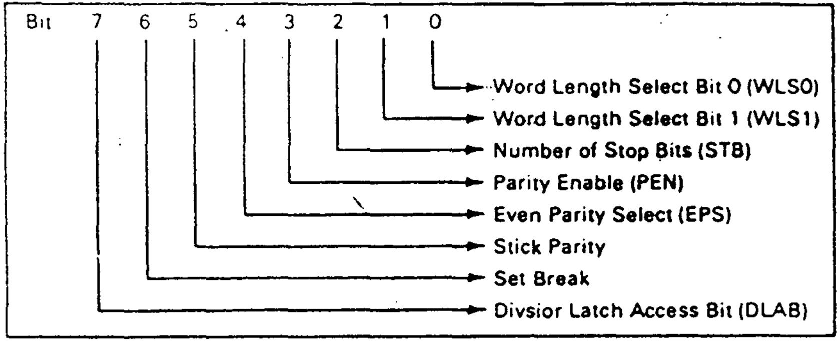
Register/Signal	Reset Control	Reset State
Interrupt Enable Register	Master Reset	All Bits Low (0-3 Forced and 4-7 Permanent)
Interrupt Identification Register	Master Reset	Bit 0 is High. Bits 1 and 2 Low Bits 3-7 are Permanently Low
Line Control Register	Master Reset	All Bits Low
Modem Control Register	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.

ADDRESS = 03FB


Bit 7 6 5 4 3 2 1 0
Word Length Select Bit 0 (WLS0)
Word Length Select Bit 1 (WLS1)
Number of Stop Bits (STB)
Parity Enable (PEN)
Even Parity Select (EPS)
Stick Parity
Set Break
Divsior Latch Access Bit (DLAB)

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.

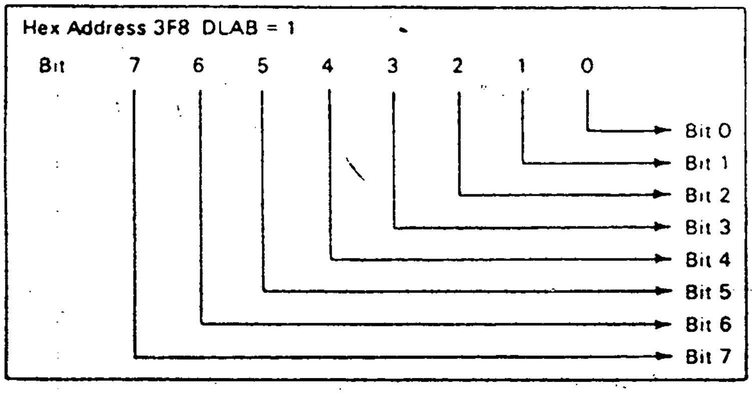
Bit 5: This bit is the stick parity bit. When bit 3 is a logical 1 and bit 5 is a logical 1, the parity bit is transmitted and then detected by the receiver as a logical 0 if bit 4 is a logical 1, or as a logical 1 if bit 4 is a logical 0.

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.

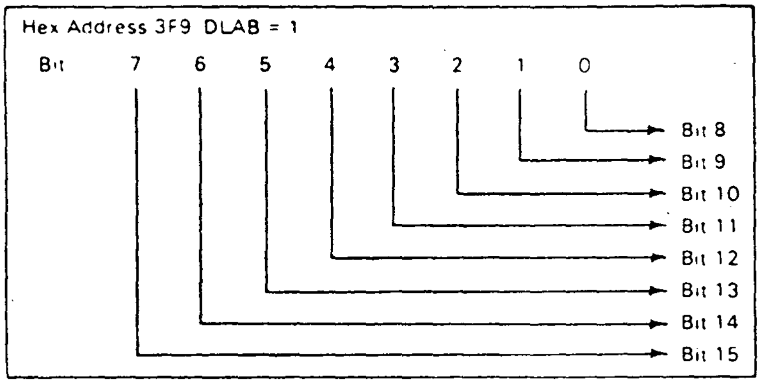
Bit 7: This bit is the divisor latch access bit (DLAB). It must be set high (logical 1) to access the divisor latches of the baud rate generator during a read or write operation. It must be set low (logical 0) to access the receiver buffer, the transmitter holding register, or the interrupt enable register.

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)



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.

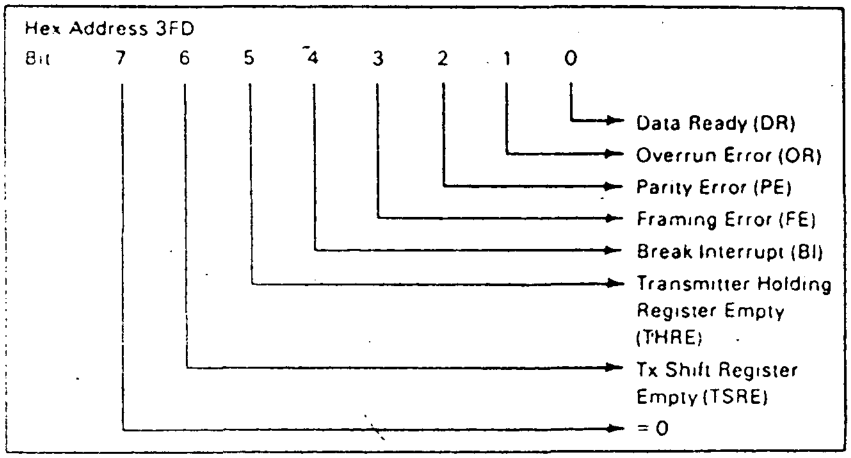
Note: The maximum operating frequency of the baud generator is 3.1 MHz. In no case should the data rate be greater than 9600 baud.

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.

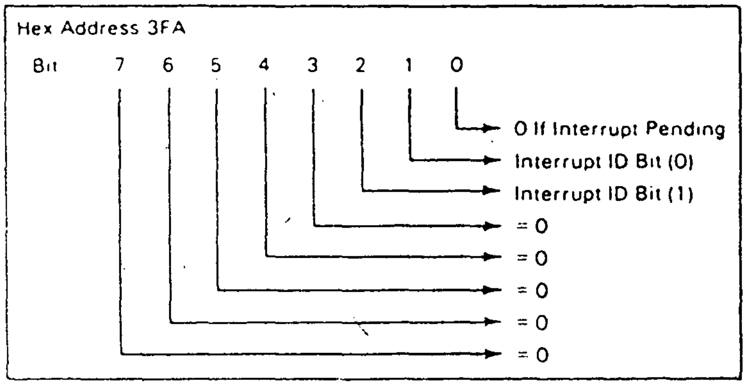
Bit 6: This bit is the transmitter shift register empty (TSRE) indicator. Bit 6 is set to a logical 1 whenever the transmitter shift register is idle. It is reset to logical 0 upon a data transfer from the transmitter holding register to the transmitter shift register. Bit 6 is a read-only bit.

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.

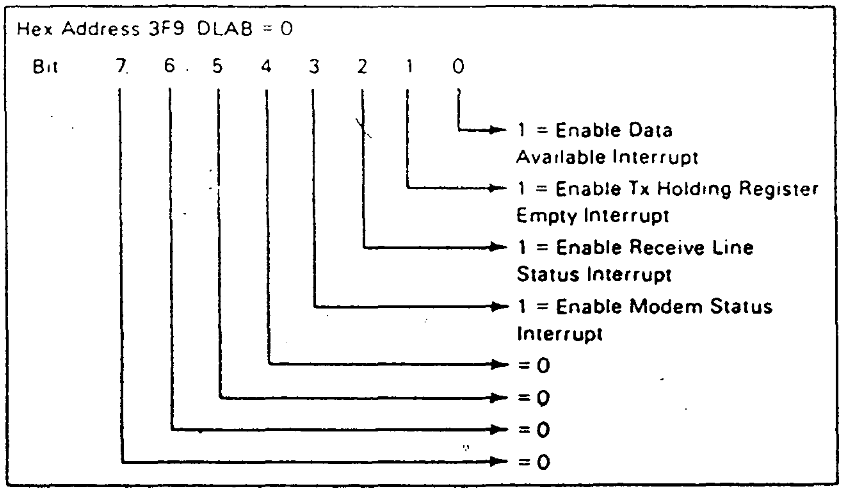
Bits 3 through 7: These five bits of the IIR are always logical 0.

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 into the Transmitter 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:


Hex Address 3F9 DLAB = 0
Bit 7 6 5 4 3 2 1 0
1 = Enable Data Available Interrupt
1 = Enable Tx Holding Register Empty Interrupt
1 = Enable Receive Line Status Interrupt
1 = Enable Modem Status Interrupt
= 0
= 0
= 0
= 0

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.

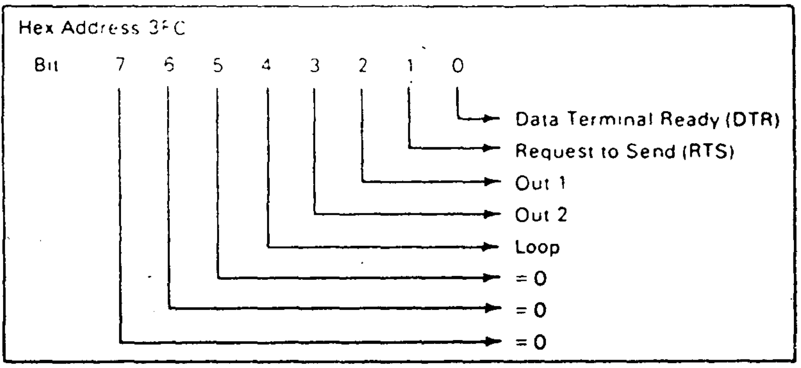
Bit 2: This bit enables the receiver line status interrupt when set to logical 1.

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 ($\overline{\text{DTR}}$) output. When bit 0 is set to a logical 1, the $\overline{\text{DTR}}$ output is forced to a logical 0. When bit 0 is reset to a logical 0, the $\overline{\text{DTR}}$ output is forced to a logical 1.

Note: The $\overline{\text{DIR}}$ 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 ($\overline{\text{RTS}}$) output. Bit 1 affects the $\overline{\text{RTS}}$ 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.

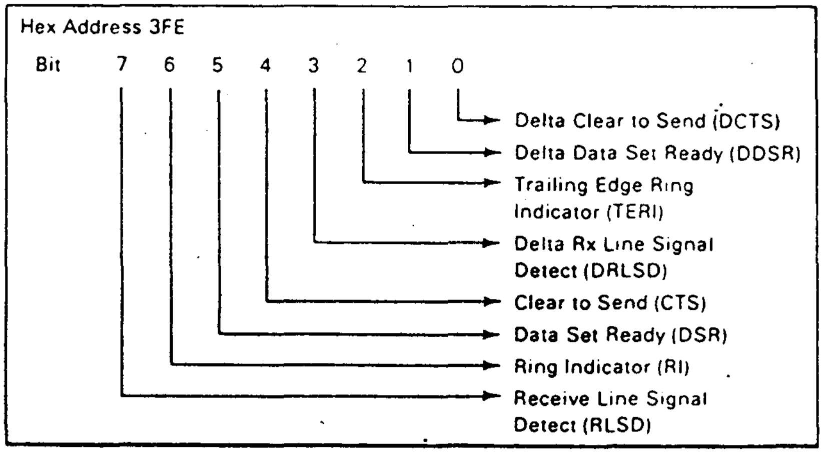
The INS8250 interrupt system can be tested by writing into the lower four bits of the modem status register. Setting any of these bits to a logical 1 generates the appropriate interrupt (if enabled). The resetting of these interrupts is the same as in normal INS8250 operation. To return to normal operation, 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:


Hex Address 3FE
Bit 7 6 5 4 3 2 1 0
Delta Clear to Send (DCTS)
Delta Data Set Ready (DDSR)
Trailing Edge Ring Indicator (TERI)
Delta Rx Line Signal Detect (DRLSD)
Clear to Send (CTS)
Data Set Ready (DSR)
Ring Indicator (RI)
Receive Line Signal Detect (RLSD)

Modem Status Register (MSR)

Bit 0: This bit is the delta clear to send (DCTS) indicator. Bit 0 indicates that the $\overline{\text{CTS}}$ 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 $\overline{\text{DSR}}$ 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 $\overline{\text{RI}}$ 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 $\overline{\text{RLSD}}$ 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 ($\overline{\text{CTS}}$) input. If bit 4 (loop) of the MCR is set to a logical 1, this is equivalent to RTS in the MCR.

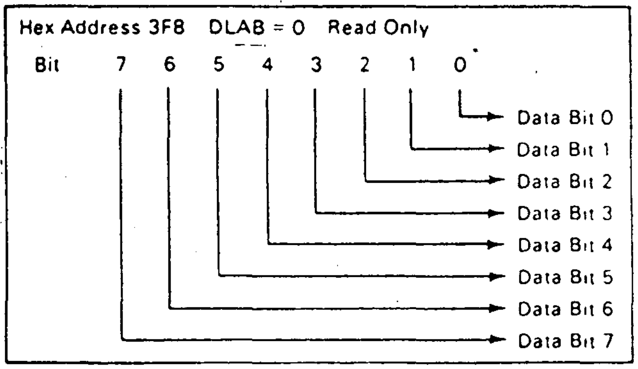
Bit 5: This bit is the complement of the data set ready ($\overline{\text{DSR}}$) input. If bit 4 of the MCR is set to a logical 1, this bit is equivalent to DTR in the MCR.

Bit 6: This bit is the complement of the ring indicator ($\overline{\text{RI}}$) 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 ($\overline{\text{RLSD}}$) 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:

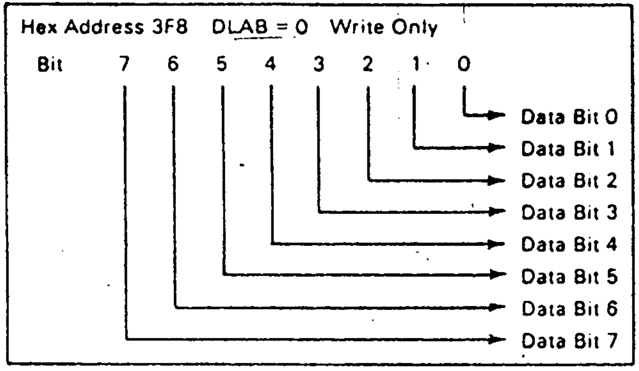


Receiver Buffer Register (RBR)

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

Transmitter Holding Register

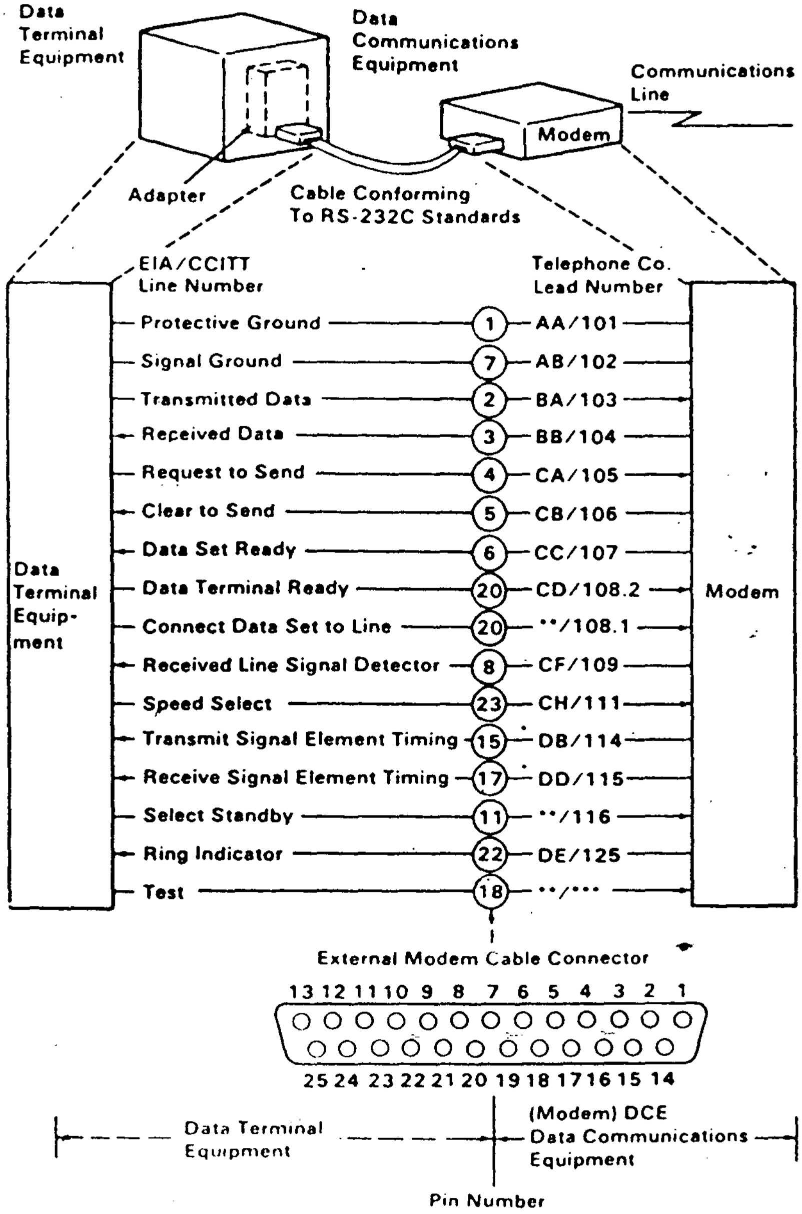
The transmitter holding register contains the character to be serially transmitted and is defined below:



Transmitter Holding Register (THR)

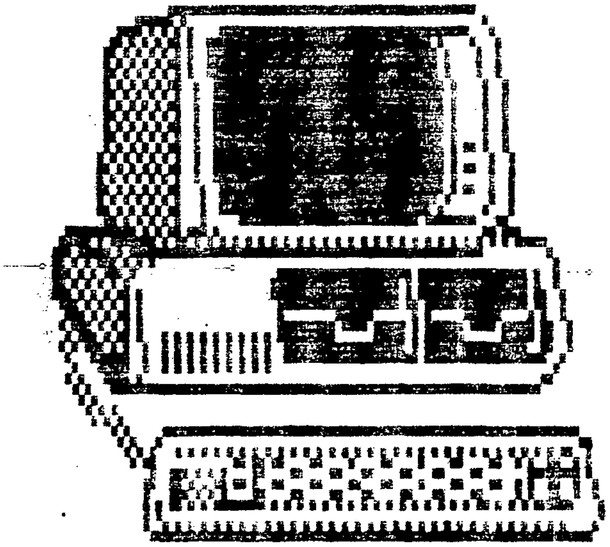
Bit 0 is the least significant bit and is the first bit serially transmitted.

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



EIA/CCITT Line Number		Telephone Co. Lead Number
Protective Ground	1	AA/101
Signal Ground	7	AB/102
Transmitted Data	2	BA/103
Received Data	3	BB/104
Request to Send	4	CA/105
Clear to Send	5	CB/106
Data Set Ready	6	CC/107
Data Terminal Ready	20	CD/108.2
Connect Data Set to Line	20	**/108.1
Received Line Signal Detector	8	CF/109
Speed Select	23	CH/111
Transmit Signal Element Timing	15	*DB/114
Receive Signal Element Timing	17	*DD/115
Select Standby	11	**/116
Ring Indicator	22	DE/125
Test	18	**/***

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



APPENDIX C

VT220 terminal

Escape Sequences

An escape sequence begins with the C0 character ESC, followed by one or more ASCII graphic characters. For example,

```
ESC  #  6
```

is an escape sequence that changes the current line of text to double-width characters. Escape sequences use only 7-bit characters, and can be used in 7-bit or 8-bit environments.

Control Sequences

A control sequence begins with CSI (9/11), followed by one or more ASCII graphic characters. CSI can also be expressed as the 7-bit code extension ESC [. So you can express all control sequences as escape sequences whose second character code is [. For example, the following two sequences are equivalent sequences that perform the same function (they cause the display to use 132 columns per line rather than 80).

```
CSI  ?  3  h

ESC  [  ?  3  h
```

Whenever possible, use CSI instead of ESC [to introduce a control sequence. CSI can only be used in an 8-bit environment.

Device Control Strings

A device control string is a delimited string of characters used in a data st.eam as a logical entity for control purposes. It consists of an opening delimiter DCS, a command string (data), and a closing delimiter ST.

DCS is an 8-bit control character that can also be expressed as ESC P when coding for a 7-bit environment.

ST is an 8-bit control character that can also be expressed as ESC / when coding for a 7-bit environment.

TRANSMITTED CODES

Main Keypad Function Keys

Key	Code Transmitted
⌫	DEL character
Tab	HT character
Return	CR character only or a CR character and an LF character, depending on the set/reset state of line feed/new line mode (LNM).
Ctrl	Does not send a code.
Lock	Does not send a code.
Shift (2 keys)	Does not send a code.
Space bar	SP character
Compose Character	Does not send a code.

Editing Keys

Key	Code Generated VT200 Mode	VT100, VT52 Modes
Find	CSI 1 ~	None
Insert Here	CSI 2 ~	None
Remove	CSI 3 ~	None
Select	CSI 4 ~	None
Prev Screen	CSI 5 ~	None
Next Screen	CSI 6 ~	None

Cursor Control Keys

	ANSI Mode*		VT52 Mode*	
	Cursor Key Mode			
	Reset	Set		
Key	Normal	Application	Normal	Application
↑	CSI A	SS3 A	ESC A	ESC A
↓	CSI B	SS3 B	ESC B	ESC B
→	CSI C	SS3 C	ESC C	ESC C
←	CSI D	SS3 D	ESC D	ESC D

* ANSI mode applies to VT200 and VT100 modes. VT52 mode is an ANSI-incompatible mode.

Auxiliary Keypad Keys

	VT100/VT200 ANSI Mode*		VT52 Mode*	
Key	Keypad Numeric Mode	Keypad Application Mode	Keypad Numeric Mode	Keypad Application Mode
0	0	SS3 p	0	ESC ? p
1	1	SS3 q	1	ESC ? q
2	2	SS3 r	2	ESC ? r
3	3	SS3 s	3	ESC ? s
4	4	SS3 t	4	ESC ? t
5	5	SS3 u	5	ESC ? u
6	6	SS3 v	6	ESC ? v
7	7	SS3 w	7	ESC ? w
8	8	SS3 x	8	ESC ? x
9	9	SS3 y	9	ESC ? y
–	–(minus)	SS3 m	–	ESC ? m
,	,(comma)	SS3 l	,	ESC ? l†
.	.(period)	SS3 n	.	ESC ? n
Enter	CR or CR LF	SS3 M	CR or CR LF	ESC ? M‡
PF1	SS3 P	SS3 P	ESC P	ESC P
PF2	SS3 Q	SS3 Q	ESC Q	ESC Q
PF3	SS3 R	SS3 R	ESC R	ESC R
PF4	SS3 S	SS3 S	ESC S	ESC S†

* ANSI mode applies to VT200 and VT100 modes. VT52 mode is an ANSI-incompatible mode.

† You cannot generate these sequences on a VT52 terminal.

‡ Keypad numeric mode. **Enter** generates the same codes as **Return**. You can change the code generated by **Return** with the line feed/new line mode. When reset, line feed/new line mode causes **Return** to generate a single control character (CR). When set, the mode causes **Return** to generate two control characters (CR, LF).

Top Row Function Keys

		Code Generated	
Name on Legend Strip	Generic Name	VT200 Mode	VT100, VT52 Modes
Hold Screen	(F1)*	–	–
Print Screen	(F2)*	–	–
Set-Up	(F3)*	–	–
Data/Talk	(F4)*	–	–
Break	(F5)*	–	–
F6	F6	CSI 1 7 ~	–
F7	F7	CSI 1 8 ~	–
F8	F8	CSI 1 9 ~	–
F9	F9	CSI 2 0 ~	–
F10	F10	CSI 2 1 ~	–
F11 (ESC)	F11	CSI 2 3 ~	ESC
F12 (BS)	F12	CSI 2 4 ~	BS
F13 (LF)	F13	CSI 2 5 ~	LF
F14	F14	CSI 2 6 ~	–
Help	(F15)	CSI 2 8 ~	–
Do	(F16)	CSI 2 9 ~	–
F17	F17	CSI 3 1 ~	–
F18	F18	CSI 3 2 ~	–
F19	F19	CSI 3 3 ~	–
F20	F20	CSI 3 4 ~	–

* F1 through F5 are local function keys and do not generate codes.

Keys Used to Generate 7-Bit Control Characters

Control Character Mnemonic	Key Pressed With Ctrl (All Modes)	Dedicated Function Key
NUL	2, space	
SOH	A	
STX	B	
ETX	C	
EOT	D	
ENQ	E	
ACK	F	
BEL	G	
BS	H	F12 (BS)*
HT	I	Tab
LF	J	F13 (LF)*
VT	K	
FF	L	
CR	M	Return
SO	N	
SI	O	
DLE	P	
DC1	Q†	
DC2	R	
DC3	S†	
DC4	T	
NAK	U	
SYN	V	
ETB	W	
CAN	X	
EM	Y	
SUB	Z	
ESC	3, [	F11 (ESC)*
FS	4, /	
GS	5,]	
RS	6, ~	
US	7, ?	
DEL	8	Delete

* Keys F11, F12, and F13 generate these 7-bit control characters only when the terminal is in VT100 mode or VT52 mode.

† These keystrokes are enabled only if XOFF support is disabled. If XOFF support is enabled, then **CTRL-S** is a "hold screen" local function and **CTRL-Q** is a "release screen" local function.

RECEIVED CODES

Compatibility Level (DECSCL)

Sequence	Action
CSI 6 1 " p	Set terminal for level 1 (VT100 mode).
CSI 6 2 " p	Set terminal for level 2 (VT200 mode, 8-bit controls).
CSI 6 2 ; 0 " p	Set terminal for level 2 (VT200 mode, 8-bit controls).
CSI 6 2 ; 1 " p	Set terminal for level 2 (VT200 mode, 7-bit controls).
CSI 6 2 ; 2 " p	Set terminal for level 2 (VT200 mode, 8-bit controls).

C0 (ASCII) Control Characters Recognized

Mnemonic	Name	Action
NUL	Null	Ignored when received.
ENQ	Enquiry	Generates answerback message.
BEL	Bell	Generates bell tone if bell is enabled.
BS	Backspace	Moves cursor to the left one character position; if cursor is at left margin, no action occurs.
HT	Horizontal tabulation	Moves cursor to next tab stop, or to right margin if there are no more tab stops. Does not cause auto wrap.
LF	Linefeed	Causes a line feed or a new line operation, depending on the setting of new line mode.
VT	Vertical tabulation	Processed as LF.
FF	Form feed	Processed as LF.
CR	Carriage return	Moves cursor to left margin on current line.
SO (LS1)	Shift out (lock shift G1)	Invokes G1 character set into GL. G1 is designated by a select-character-set (SCS) sequence.

C0 (ASCII) Control Characters Recognized (Cont)

Mnemonic	Name	Action
SI (LSO)	Shift in (lock shift G0)	Invokes G0 character set into GL. G0 is designated by a select-character-set (SCS) sequence.
DC1	Device control 1	Also referred to as XON. If XOFF support is enabled, DC1 clears DC3 (XOFF), causing the terminal to continue sending characters (keyboard unlocks) unless KAM mode is currently set.
DC3	Device control 3	Also referred to as XOFF. If XOFF support is enabled, DC3 causes the terminal to stop sending characters until a DC1 control character is received.
CAN	Cancel	If received during an escape or control sequence, terminates and cancels the sequence. No error character is displayed. If received during a device control string, the DCS is terminated and no error character is displayed.
SUB	Substitute	If received during escape or control sequence, terminates and cancels the sequence. Causes a reverse question mark to be displayed. If received during a device control sequence, the DSC is terminated and reverse question mark is displayed.
ESC	Escape	Processed as escape sequence introducer. Terminates any escape, control or device control sequence which is in progress.
DEL	Delete	Ignored when received. Note: May not be used as a time fill character.

C1 Control Characters Recognized

Mnemonic	Equivalent 7-Bit Code Extension	Name	Action
IND	ESC D	Index	Moves cursor down one line in same column. If cursor is at bottom margin, screen performs a scroll up.
NEL	ESC E	Next line	Moves cursor to first position on next line. If cursor is at bottom margin, screen performs a scroll up.
HTS	ESC H	Horizontal tab set	Sets one horizontal tab stop at the column where the cursor is.
RI	ESC M	Reverse index	Moves cursor up one line in same column. If cursor is at top margin, screen performs a scroll down.
SS2	ESC N	Single shift G2	Temporarily invokes G2 character set into GL for the next graphic character. G2 is designated by a select-character-set (SCS) sequence.
SS3	ESC O	Single shift G3	Temporarily invokes G3 character set into GL for the next graphic character. G3 is designated by a select-character-set (SCS) sequence.
DCS	ESC P	Device control string	Processed as opening delimiter of a device control string for device control use.

C1 Control Characters Recognized (Cont)

Mnemonic	Equivalent 7-Bit Code Extension	Name	Action
CSI	ESC [	Control sequence introducer	Processed as control sequence introducer.
ST	ESC /	String terminator	Processed as closing delimiter of a string opened by DCS.

CHARACTER SET SELECTION (SCS)

Designating Hard Character Sets

Use the following list of escape sequence formats to designate hard character sets as G0 through G3.

Escape Sequence	Designate As:
ESC ({final}	G0
ESC) {final}	G1
ESC * {final}	G2
ESC + {final}	G3

The following is a list of available character sets and their associated final character.

Character Sets	Final Character
ASCII	B
DEC supplemental (VT200 mode only)	<
DEC special graphics	0
National replacement character sets	**NOTE** **Only one national character set is available for use at any one time (national mode).**
British	A
Dutch	4
Finnish	C or 5
French	R
French Canadian	Q
German	K
Italian	Y

Character Sets	Final Character
Norwegian/Danish	E or 6
Spanish	Z
Swedish	H or 7
Swiss	=

Examples

ASCII as G0	ESC (B
British as G3	ESC * A

Designating Soft (Down-Line-Loadable) Character Sets

Escape Sequence	Designate As:
ESC (Dscs	G0
ESC) Dscs	G1
ESC * Dscs	G2
ESC + Dscs	G3

Dscs can consist of zero, one, or two intermediate characters and a final character.

Intermediate characters are in the range of 2/0 to 2/15; final characters are in the range of 3/0 to 7/14. (See ASCII Code Table for column/row notation.)

Invoking Character Sets Using Lock Shifts

Control Name	Coding	Function
LS0 – lock shift G0	SI	Invoke G0 into GL (default).
LS1 – lock shift G1	SO	Invoke G1 into GL.
LS1R – lock shift G1, right	ESC ~	Invoke G1 into GR (VT200 mode only).
LS2 – lock shift G2	ESC n	Invoke G2 into GL (VT200 mode only).
LS2R – lock shift G2, right	ESC }	Invoke G2 into GR (default,VT200 mode only).
LS3 – lock shift G3	ESC o	Invoke G3 into GL (VT200 mode only).
LS3R – lock shift G3, right	ESC |	Invoke G3 into GR (VT200 mode only).

Invoking Character Sets Using Single Shifts

Control Name	Coding	Function
SS2 - single shift G2	SS2 ESC N	Invokes G2 into GL for the next graphic character.
SS3 - single shift G3	SS3 ESC O	Invokes G3 into GL for the next graphic character.

Select C1 Control Transmission

Control Name	Sequence*	Action
7-bit C1 control transmission (S7C1T)	ESC sp F	Converts all C1 codes returned to the application to their equivalent 7-bit code extensions.

NOTE
The S7C1T sequence is ignored when the terminal is in VT100 or VT52 mode.

Control Name	Sequence*	Action
8-bit C1 control transmission (S8C1T)	ESC sp G	Returns C1 codes to the application without converting them to their equivalent 7-bit code extensions.

* sp is a space character

Terminal Modes

Name	Mnemonic	Set Mode	Reset Mode*
Keyboard action†	KAM	Locked CSI 2 h	Unlocked CSI 2 l
Insertion-replacement	IRM	Insert CSI 4 h	Replace CSI 4 l
Send-receive	SRM	Off CSI 12 h	On CSI 12 l
Line feed-new line	LNM	New line CSI 20 h	Line feed CSI 20 l
Cursor key	DECCKM	Application CSI ? 1 h	Cursor CSI ? 1 l
ANSI/VT52	DECANM	N/A CSI ? 2 l	VT52
Column	DECCOLM	132 column CSI ? 3 h	80 column CSI ? 3 l
Scrolling†	DECSCLM	Smooth CSI ? 4 h	Jump CSI ? 4 l
Screen†	DECSCNM	Reverse CSI ? 5 h	Normal CSI ? 5 l
Auto wrap	DECAWM	On CSI ? 7 h	Off CSI ? 7 l
Auto repeat†	DECARM	On CSI ? 8 h	Off CSI ? 8 l
Print form feed	DECPFF	On CSI ? 18 h	Off CSI ? 18 l
Print extent	DECPEX	Full screen CSI ? 19 h	Scrolling region CSI ? 19 l
Text cursor enable	DECTCEM	On CSI ? 25 h	Off CSI ? 25 l
Keypad	DECKPAM DECKPNM	Application ESC =	Numeric ESC >
Character set	DECNRCM	National CSI ? 42 h	Multinational CSI ? 42 l

* The last character of each sequence is lowercase L (6/12).

† User preference feature

Cursor Positioning

Name	Control Character	Sequence	Action
Cursor up (CUU)	–	CSI Pn A	Moves cursor up Pn lines in the same column.
Cursor down (CUD)	–	CSI Pn B	Moves cursor down Pn lines in the same column.
Cursor forward (CUF)	–	CSI Pn C	Moves cursor right Pn columns.
Cursor backward (CUB)	–	CSI Pn D	Moves cursor left Pn columns.
Cursor position (CUP)	–	CSI Pl ; Pc H	Moves cursor to line Pl, column Pc. The numbering of the lines and columns depends on the state (set/reset) of origin mode (DECOM).
Horizontal and vertical position (HVP)	–	CSI Pl ; Pc f	Moves cursor to line Pl, column Pc. The numbering of the lines and columns depends on the state (set/reset) of origin mode (DECOM). Digital recommends using CUP instead of HVP.
Index (IND)	IND	ESC D	Moves cursor down one line in the same column. If the cursor is at the bottom margin the screen performs a scroll-up.
Reverse index (RI)	RI	ESC M	Moves cursor up one line in the same column. If the cursor is at the top margin the screen performs a scroll-down.

Cursor Positioning (Cont)

Name	Control Character	Sequence	Action
Next line (NEL)	NEL	ESC E	Moves the cursor to the first position on the next line. If the cursor is at the bottom margin the screen performs a scroll-up.
Save cursor (DECSC)	–	ESC 7	Saves the following in terminal memory. • cursor position • graphic rendition • character set shift state • state of wrap flag • state of origin mode • state of selective erase
Restore cursor (DECRC)	–	ESC 8	Restores the states described for (DECSC) above. If none of these characteristics were saved; the cursor moves to home position, origin mode is reset, no character attributes are assigned, and the default character set mapping is established.

Tab Stops

> **NOTE**
> **These sequences are affected by the user preference lock in set-up.**

Name	Control Character	Sequence	Action
Horizontal tab set (HTS)	HTS	ESC H	Sets a tab stop at the current column.
Tabulation clear (TBC)	–	CSI g	Clears a horizontal tab stop at cursor position.
		CSI 0 g	Clears a horizontal tab stop at cursor position.
		CSI 3 g	Clears all horizontal tab stops.

Select Graphic Rendition (SGR)

You can select one or more character renditions at a time using the following format.

CSI Ps ; ... Ps m

When you use multiple parameters, they are executed in sequence. The effects are cumulative. For example, to change from increased intensity to blinking-underlined, you can use:

CSI 0 ; 4 ; 5 m

When you select a single parameter, no delimiter (3/11) is used.

Ps	Action
0	All attributes off.
1	Display at increased intensity.
4	Display underscored.
5	Display blinking.
7	Display negative (reverse) image.
2 2	Display normal intensity.
2 4	Display not underlined.
2 5	Display not blinking.
2 7	Display positive image.

Select Character Attributes (DECSCA)

You can select all subsequent characters to be erasable or not erasable using the following format. (See "Erasing" section.)

> **NOTE**
> **This sequence is supported only in VT200 mode.**

CSI Ps " q

where:

Ps	Action
0	All attributes off. (Does not apply to SGR.)
1	Designate character as "not erasable" by DECSEL/DECSED (attribute on).
2	Designate character as "erasable" by DECSEL/DECSED (attribute off).

Line Attributes

Name	Sequence	
	Top Half	Bottom Half
Double-height line (DECDHL)	ESC # 3	ESC # 4
	The same character must be used on both lines to form full character. If the line was previously single-width, single-height, all characters to the right of center are lost.	
Single-width line (DECSWL)	ESC # 5	
Double-width line (DECDWL)	ESC # 6	

Editing

Name	Sequence	Action
Insert line (IL)	CSI Pn L	Inserts Pn lines at the cursor.
Delete line (DL)	CSI Pn M	Deletes Pn lines starting at the line with the cursor.
Insert characters (ICH) (VT200 mode only)	CSI Pn @	Insert Pn blank characters at the cursor position, with the character attributes set to normal.
Delete character (DCH)	CSI Pn P	Deletes Pn characters starting with the character at the cursor position.

Erasing

Name	Sequence	Action
Erase character (ECH) (VT200 mode only)	CSI Pn X	Erases characters at the cursor position and the next n-1 character.
Erase in line (EL)	CSI K	Erases from the cursor to the end of the line, including the cursor position.
	CSI 0 K	Same as above.
	CSI 1 K	Erases from the beginning of the line to the cursor, including the cursor position.
	CSI 2 K	Erases the complete line.
Erase in display (ED)	CSI J	Erases from the cursor to the end of the screen, including the cursor position.
	CSI 0 J	Same as above.
	CSI 1 J	Erases from the beginning of the screen to the cursor, including the cursor position.
	CSI 2 J	Erases the complete display.

Erasing (Cont)

Name	Sequence	Action
Selective erase in line (DECSEL) (VT200 mode only)	CSI ? K	Erases all "erasable" characters (DECSCA) from the cursor to the end of the line.
	CSI ? 0 K	Same as above.
	CSI ? 1 K	Erases all "erasable" characters (DECSCA) from the beginning of the line to and including the cursor position.
	CSI ? 2 K	Erases all "erasable" characters (DECSCA) on the line.
Selective erase in display (DECSED) (VT200 mode only)	CSI ? J	Erases all "erasable" characters (DECSCA) from and including the cursor to the end of the screen.
	CSI ? 0 J	Same as above.
	CSI ? 1 J	Erases all "erasable" characters (DECSCA) from the beginning of the screen to and including the cursor.
	CSI ? 2 J	Erases all "erasable" characters (DECSCA) in the entire display.

Set Top and Bottom Margins (DECSTBM)

CSI Pt ; Pb r

Selects top and bottom margins defining the scrolling region. Pt is the line number of the first line in the scrolling region. Pb is the line number of the bottom line. If you do not select either Pt or Pb, they default to top and bottom respectively. Lines are counted from 1.

Printing

Before you select a print operation, check printer status using the print status report (DSR). (See "Reports" section.)

Name	Sequence	Action
Auto print mode	CSI ? 5 i	Turns on auto print mode. All following display lines print when you move the cursor off the line using a line feed, form feed, vertical tab, or auto wrap. The printed line ends with a carriage return and the character (LF, FF, or VT) which moved the cursor off the previous line. Auto wrap lines end with a line feed.
	CSI ? 4 i	Turns off auto print mode.
Printer controller	CSI 5 i	Turns on printer controller mode. The terminal sends received characters to the printer without displaying them on the screen. All characters and character sequences, except NUL, XON, XOFF, CSI 5 i, and CSI 4 i are sent to the printer. The terminal does not insert or delete spaces, or provide line delimiters, or select the correct printer character set. Printer controller mode has a higher priority than auto print mode. It can be selected during auto print mode. When in printer controller mode, keyboard activity continues to be directed to the host.
	CSI 4 i	Turns off printer controller mode.
Print cursor line	CSI ? 1 i	Prints the display line containing the cursor. The cursor position does not change. The print-cursor-line sequence is complete when the line prints.

Printing (Cont)

Name	Sequence	Action
Print screen	CSI i	Prints the screen display (full screen or scrolling region depending on the print extent DECEXT selection). Printer form feed mode (DECPFF) selects either a form feed (FF) or nothing as the print terminator. The print screen sequence is complete when the screen prints.
	CSI 0 i	Same as above.

User Defined Keys (DECUDK)

The device control string format for down-line-loading UDK functions is:

DCS Pc;Pl | Ky1/st1;ky2/st2;...kyn/stn ST

where:

Pc	Meaning
None	Clear all keys before loading new values.
0	Clear all keys before loading new values.
1	Load new key values, clear old only where defined.

Pl	Meaning
None	Lock the keys against future redefinition.
0	Lock the keys against future redefinition.
1	Do not lock the keys against future redefinition.

Key	Value (kyn)	Key	Value (kyn)
F6	17	F14	26
F7	18	HELP	28
F8	19	DO	29
F9	20		
F10	21	F17	31
F11	23	F18	32
F12	24	F19	33
F13	25	F20	34

Stn is a string of hex pairs of ASCII characters that define the specified key.

NOTE
To access the programmed values of the keys, you type Shift – (function key).

Down-Line-Loading Characters (DRCS)

You can down-line-load your DRCS character set using the following DECDLD device control string format.

DCS Pfn;Pcn;Pe;Pcms;Pw;Pt { Dscs Sxbp1;Sxbp2;...;Sxbpn ST

Parameter descriptions are as follows.

DECDLD Parameter Characters

Parameter	Name	Description
Pfn	Font number	0 and 1.
Pcn	Starting character number	Selects starting character in DRCS font buffer to be loaded.
Pe	Erase control	0 = erase all characters in this DRCS set 1 = erase only the characters that are being reloaded 2 = erase all characters in all DRCS sets (this font buffer number and other font buffer numbers)
	Character Matrix size	0 = device default (7 × 10) 1 = (not used) 2 = 5 × 10 3 = 6 × 10 4 = 7 × 10
Pw	Width attribute	0 = device default (80 columns) 1 = 80 column 2 = 132 column
Pt	Text/ full-cell	0 = device default (text) 1 = text 2 = full-cell (not used)

Dscs defines the character set name for the soft font, and is used in the SCS (select character set) escape sequence.

Sxbp1;Sxbp2;...;Sxbpn are sixel bit patterns (1 to 94 patterns) for characters separated by semicolons. Each sixel bit pattern has the form:

S...S/...S

where the first S....S represents the upper columns (sixels) of the DRCS character, the slash advances the sixel pattern to the lower columns of the DRCS character, and the second S....S represents the lower columns (sixels) of the DRCS.

Clearing a Down-Line-Loaded Character Set

You can clear a character set that you have down-line-loaded using the following DECDLD control sequence.

DCS 1;1;2 { sp @ ST

Down-line-loaded character sets are also cleared by the following actions.

- Performing the power-up self-test.
- Using the set-up "Recall" or "Default" features.
- Using RIS or ESC c sequences.

Reports

Device Attributes (DA)

Communication	Sequence	Meaning
Host to VT220 (primary DA request)	CSI c or CSI 0 c	"What is your service class code and what are your attributes?"
VT220 to host (primary DA response)	CSI ? 62; 1; 2; 6; 7; 8; 9 c	"I am a service class 2 (VT200 family) terminal (62) with 132 columns (1), printer port (2), selective erase (6), DRCS (7), UDK (8). I support 7-bit national replacement character sets (9)."
Host to VT220 (secondary DA request)	CSI > c or CSI > 0 c	"What type of terminal are you, what is your firmware version, and what hardware options do you have installed?"
VT220 to host (secondary DA response)	CSI > 1; Pv; Po c	"I am a VT220 (identification code of 1, my firmware version is _____ (Pv), and I have P0 options installed.

EXAMPLE: CSI>1;10;0c = VT220 version 1.0, no options

> **NOTE**
> **If the terminal is in VT100 mode and an ID other than VT220 ID is selected, then the following primary exchanges apply.**

Communication	Sequence	Meaning
VT220 to host (VT100 ID selected in set-up)	ESC [? 1; 2 c	"I am a VT100 terminal with AVO."
VT220 to host (VT101 ID selected in set-up)	ESC [? 1; 0 c	"I am a VT101 terminal."
VT220 to host (VT102 ID selected in set-up)	ESC [? 6 c	"I am a VT102 terminal."

Device Status Report (DSR)

Communication	Sequence	Meaning
Host to VT220 (request for terminal status)	CSI 5 n	"Please report your operating status using a DSR control sequence. Are you in good operating condition or do you have a malfunction?"
VT220 to host (DA response)	CSI 0 n	"I have no malfunction."
	CSI 3 n	"I have a malfunction."
Host to VT220 (request for cursor position)	CSI 6 n	"Please report your cursor position using a CPR (not DSR) control sequence."

VT220 to host (CPR response)	CSI Pv; Ph R	"My cursor is positioned at ______ (Pv); ______ (Ph)."
		Where:
		Pv = vertical position (row)
		Ph = horizontal position (column)

DSR – Printer Port

Communication	Sequence	Meaning
Host to VT220 (request for printer status)	CSI ? 15 n	"What is the printer status?"
VT220 to host	CSI ? 13 n	"DTR has not been asserted on the printer port since power up or reset–in essence, I have no printer."
	CSI ? 10 n	"DTR is asserted on the printer port. The printer is ready."
	CSI ? 11 n	"DTR is not currently asserted on the printer port. The printer is not ready."

DSR – User Defined Keys (VT200 mode only)

Communication	Sequence	Meaning
Host to VT220 (request for UDK status)	CSI ? 25 n	"Are User Defined Keys locked or unlocked?"
VT220 to host	CSI ? 20 n	"User Defined Keys are unlocked."
	CSI ? 21 n	"User Defined Keys are locked."

DSR – Keyboard Language

Communication	Sequence	Meaning
Host to VT220 (request for keyboard language)	CSI ? 26 n	"What is the keyboard language?"
VT220 to host	CSI ? 27; Pn n	"My keyboard language is _____ (Pn)."

where:

Pn	Language
0	Unknown*
1	North American
2	British
3	Flemish
4	French Canadian
5	Danish
6	Finnish
7	German
8	Dutch
9	Italian
10	Swiss (French)
11	Swiss (German)
12	Swedish
13	Norwegian
14	French/Belgian
15	Spanish

* Sent by a terminal that for some reason cannot determine its keyboard language. The VT220 will never send this response.

Identification (DECID)

ESC Z

Causes the terminal to send a primary DA response sequence. DECID, however, is not recommended. You should use the primary DA request for this purpose.

Terminal Reset

Name	Sequence	Action
Soft terminal reset (DECSTR)	CSI ! p	Sets terminal to power-up default states
Hard terminal reset (RIS)	ESC c	Replaces all set-up parameters with NVR values or power-up default values if NVR values do not exist.

Tests (DECTST)

The sequence format for invoking terminal tests is as follows.

CSI 4 ; ..... ; Ps y

where:

Ps	Test
0	Test 1, 2, 3, and 6
1	Power-up self-test
2	EIA port data loopback test
3	Printer port loopback test
4	(not used)
5	(not used)
6	EIA port modem control line loopback test
7	20 mA port loopback test
8	(not used)
9	Repeat other test in parameter string
10 and up	(not used)

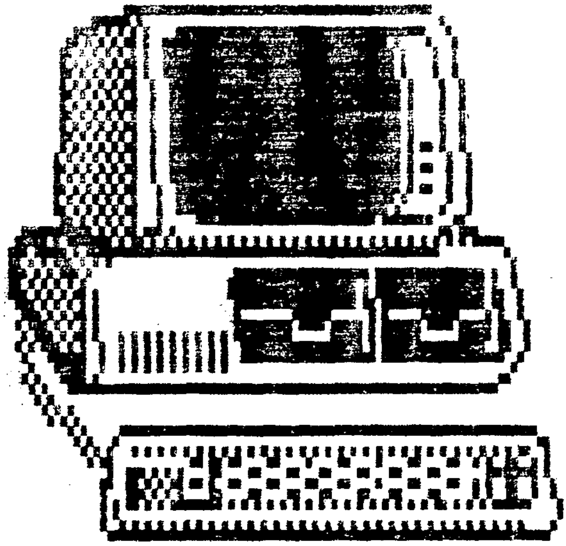
NOTE
DECTST causes a communications line disconnect.

Adjustments (DECALN)

ESC # 8 Displays screen alignment pattern (full screen of E's).

VT52 Escape Sequences

Escape Sequence	Function
ESC A	Cursor up
ESC B	Cursor down
ESC C	Cursor right
ESC D	Cursor left
ESC F	Enter graphics mode
ESC G	Exit graphics mode
ESC H	Cursor to home
ESC I	Reverse line feed
ESC J	Erase to end of screen
ESC K	Erase to end of line
ESC Y Line Column	Direct cursor address
ESC Z	Identify
ESC =	Enter alternate keypad mode
ESC >	Exit alternate keypad mode
ESC <	Enter ANSI mode
ESC	Enter auto print mode
ESC _	Exit auto print mode
ESC W	Enter printer controller mode
ESC X	Exit printer controller mode
ESC]	Print screen
ESC V	Print cursor line



APPENDIX F

interrupt vectors

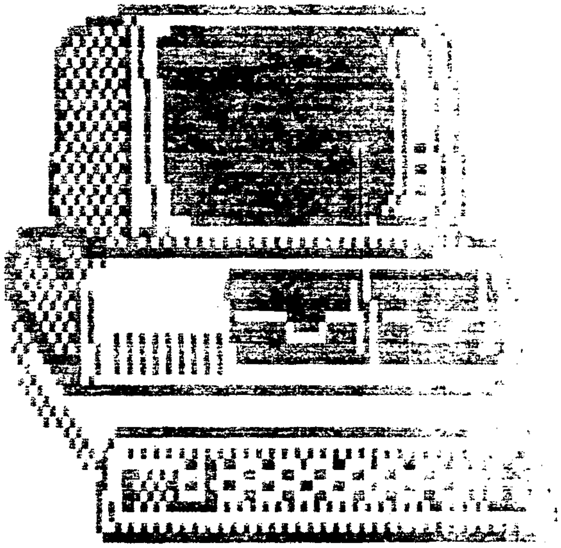
Interrupt Vectors 0H through 1FH

Interrupt Vector	Address (Hex)	Purpose	Notes
0	00–03	Divide by zero trap	
1	04–07	Single step	
2	08–0B	Nonmaskable interrupt	ROM BIOS
3	0C–0F	Breakpoint trap	
4	10–13	Overflow detection	
5	14–17	Print screen	ROM BIOS
6	18–1B		
7	1C–1F		
8	20–23	Timer	
9	24–27	Keyboard	
A	28–2B		8259
B	2C–2F		interrupt
C	30–33	(reserved for communications)	vectors
D	34–37		
E	38–3B		
F	3C–3F	(reserved for printer)	
10	40–43	Video I/O	
11	44–47	Equipment check	
12	48–4B	Memory size	
13	4C–4F	Diskette access	
14	50–53	Communications	
15	54–57	Cassette I/O	ROM BIOS
16	58–5B	Keyboard I/O	vectors
17	5C–5F	Printer I/O	
18	60–63	Cassette BASIC	
19	64–67	Bootstrap	
1A	68–6B	Time of day	
1B	6C–6F	Ctrl-Break handling	
1C	70–73	Timer tick	
1D	74–77	Video initialization	

1E	78–7B	Diskette parameters	
1F	7C–7F	Video graphics characters, ASCII	128–255

MS-DOS Interrupt Vectors 20H through 27H

Interrupt Vector	Address (Hex)	Purpose
20	80–83	General program termination
21	84–87	DOS function request
22	88–8B	Called program termination address
23	8C–8F	Ctrl-Break termination address
24	9C-93	Critical error handler
25	94–97	Absolute disk read
26	98–9B	Absolute disk write
27	9C–9F	Terminate but stay resident



APPENDIX E

ASCII character set

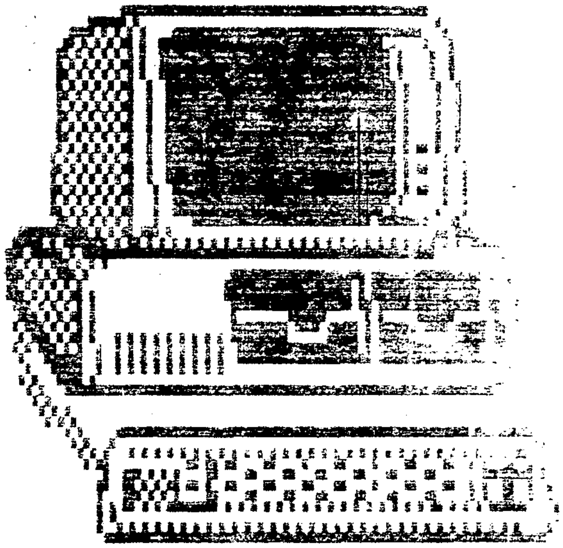
Character Set (00-7F) Quick Reference

DECIMAL VALUE	➡	0	16	32	48	64	80	96	112
⬇	HEXA DECIMAL VALUE	0	1	2	3	4	5	6	7
0	0	BLANK (NULL)	►	BLANK (SPACE)	0	@	P	‘	p
1	1	☺	◄	!	1	A	Q	a	q
2	2	☻	↕	"	2	B	R	b	r
3	3	♥	‼	#	3	C	S	c	s
4	4	♦	¶	$	4	D	T	d	t
5	5	♣	§	%	5	E	U	e	u
6	6	♠	▬	&	6	F	V	f	v
7	7	•	↨	'	7	G	W	g	w
8	8	◘	↑	(	8	H	X	h	x
9	9	○	↓	)	9	I	Y	i	y
10	A	◙	→	*	:	J	Z	j	z
11	B	♂	←	+	;	K	[	k	{
12	C	♀	∟	,	<	L	\	l	¦
13	D	♪	↔	—	=	M	]	m	}
14	E	♫	▲	.	>	N	^	n	~
15	F	☼	▼	/	?	O	_	o	△

Character Set (88-FF) Quick Reference

DECIMAL VALUE	➡	128	144	160	176	192	208	224	240
⬇	HEXA DECIMAL VALUE	8	9	A	B	C	D	E	F
0	0	Ç	É	á	░	└	╨	∝	≡
1	1	ü	æ	í	▒	┴	╤	β	±
2	2	é	Æ	ó	▓	┬	╥	Γ	≥
3	3	â	ô	ú	│	├	╙	π	≤
4	4	ä	ö	ñ	┤	─	╘	Σ	⌠
5	5	à	ò	Ñ	╡	┼	╒	σ	⌡
6	6	å	û	ª	╢	╞	╓	μ	÷
7	7	ç	ù	º	╖	╟	╫	τ	≈
8	8	ê	ÿ	¿	╕	╚	╪	Φ	°
9	9	ë	Ö	⌐	╣	╔	┘	θ	●
10	A	è	Ü	¬	║	╩	┌	Ω	•
11	B	ï	¢	½	╗	╦	█	δ	√
12	C	î	£	¼	╝	╠	▄	∞	n
13	D	ì	¥	¡	╜	═	▌	ϕ	2
14	E	Ä	₧	«	╛	╬	▐	∈	▮
15	F	Å	ƒ	»	┐	╧	▀	∩	BLANK FF

Technical Reference Manual, pp.7-12–7-13, © 1984 International Business Machines Corporation



APPENDIX D

scancodes of TANDY1000

Appendix B

ASCII AND SCAN CODES

The following table lists the keys, in scan code order, and the ASCII codes generated by each (which depends on the shift status). The entries in the table are:

- SCAN CODE — a value in the range Ø1H-5AH (hexadecimal) that uniquely describes which key is pressed.
- KEYBOARD LEGEND — the physical marking(s) on the key. If multiple markings exist, they are listed from top to bottom.
- NORMAL — the normal (unshifted) ASCII value (returned when only the indicated key is pressed).
- SHIFT — the shifted ASCII value (returned when SHIFT is also pressed).
- CTRL — the control ASCII value (returned when CTRL is also pressed).
- ALT — the alternate ASCII value (returned when ALT is also pressed).
- REMARK — any remarks or special functions.

All numeric values in the table are expressed in hexadecimal. Those values preceded by an *x* are extended ASCII codes (they are preceded by an ASCII NUL [=ØØ]).

A marking of — indicates that no ASCII code is generated. A marking of ** indicates that no ASCII code is generated and that, instead, the special function described in the REMARK column is performed.

SCAN	KEYBOARD	ASCII CODES				
CODE	LEGEND	NORMAL	SHIFT	CTRL	ALT	REMARK
01	ESC	01B	01B	01B	—	
02	! 1	031	021	—	X078	
03	@ 2	032	040	X000	X079	
04	# 3	033	023	—	X07A	
05	$ 4	034	024	—	X07B	
06	% 5	035	025	—	X07C	
07	` 6	036	05E	01E	X07D	
08	& 7	037	026	—	X07E	
09	* 8	038	02A	—	X07F	
0A	(9	039	028	—	X080	
0B	) 0	030	029	—	X081	
0C	___	02D	05F	01F	X082	
0D	+ =	03D	02B	—	X083	
0E	BACK SPACE	008	008	07F	X08C	
0F	TAB	009	X00F	X08D	X08E	
10	Q	071	051	011	X010	
11	W	077	057	017	X011	
12	E	065	045	005	X012	
13	R	072	052	012	X013	
14	T	074	054	014	X014	
15	Y	079	059	019	X015	
16	U	075	055	015	X016	
17	I	069	049	009	X017	
18	O	06F	04F	00F	X018	
19	P	070	050	010	X019	
1A	[{	05B	07B	01B	—	
1B	] }	a05D	07D	01D	—	
1C	ENTER	00D	00D	00A	X08F	(mail keyboard)
1D	CTRL	*	—	—	—	control mode
1E	A	061	041	001	X01E	
1F	S	073	053	013	X01F	
20	D	064	044	004	X020	
21	F	066	046	006	X021	
22	G	067	047	007	X022	
23	H	068	048	008	X023	
24	J	06A	04A	00A	X024	
25	K	06B	04B	00B	X025	
26	L	06C	04C	00C	X026	
27	; :	03B	03A	—	—	
28	' "	027	022	—	—	
29	↑	X048	X085	X09	X091	
2A	LEFT SHIFT	*	—	—	—	SHIFT
2B	←	X04B	X087	X073	X092	
2C	Z	07A	05A	01A	X02C	
2D	X	078	058	018	X02D	
2E	C	063	043	003	X02E	
2F	V	076	056	016	X02F	
30	B	0a62	042	002	X030	
31	N	06E	04E	00E	X031	

SCAN	KEYBOARD	ASCII CODES				
CODE	LEGEND	NORMAL	SHIFT	CTRL	ALT	REMARK
32	M	06D	04D	00D	X032	
33	, <	02C	03C	—	—	
34	. >	02E	03E	—	—	
35	/ ?	02F	03F	—	—	
36	RIGHT SHIFT	*	—	—	—	SHIFT
37	PRINT SCREEN	015	*	X072	X046	print screen
38	ALT	*	—	—	—	alternate mode
39	SPACE BAR	020	020	020	x020	
3A	CAPS LOCK	*		—	—	caps lock
3B	F1	X03B	X054	X05E	X068	
3C	F2	X03C	X055	X05F	X069	
3D	F3	X03D	X056	X060	X06A	
3E	F4	X03E	X057	X061	X06B	
3F	F5	X03F	X058	X062	X06C	
40	F6	X040	X059	X063	X06D	
41	F7	X041	X05A	X064	X06E	
42	F8	X042	X05B	X065	X06F	
43	F9	X043	X05C	X066	X070	
44	F10	X044	X05D	X067	X071	
45	NUM LOCK	*	—	—	—	number lock
46	HOLD	*	*	*	*	freeze display
47	7 \	037	05C	X093	÷	
48	8 ~	038	07E	X094	÷	
49	9 pG UP	039	X049	X084	÷	
4A	↓	X050	X086	X096	X097	
4B	4 |	034	07C	X095		
4C	5	035	—	—	÷	
4D	6	036	—	—	÷	
4E	→	X04D	X088	074	—	
4F	END 1	031	X04F	X075	—	
50	2 `	032	060	X09A	÷	
51	3 PG DN	033	X051	X076	÷	
52	0	030	X09B	X09C	÷	
53	- DELTE	02D	X053	X09D	X09E	
54	BREAK	000	000	*	X400	control break routine (INT 1BH)
55	+ INSERT	02B	X052	X09F	X0A0	
56	.	02E	X0A1	X0A4	X0A5	(numeric keypad)
57	ENTER	00D	00D	00A	X08F	(numeric keypad)
58	HOME	X047	X04A	X077	X0A6	
59	F11	X098	X0A2	X0AC	X0B6	
5A	F12	X099	X0A3	X0AD	X0B7	

* Indicates special functions performed

— means this key combination is suppressed in the keyboard driver

X values preceded by X are extended ASCII codes (codes preceded by an ASCII NUL)

† The ALT key provides a way to generate the ASCII codes of decimal numbers between 1 and 255. Hold down the ALT key while you type *on the numeric keypad* any decimal number between 1 and 255. When you release ALT, the ASCII code of the number typed is generated and displayed.

Note: When the NUM LOCK light is off, the Normal and SHIFT columns for these keys should be reversed.

ASCII Character Codes

The ASCII and Scan Codes table listed the ASCII codes (in hexadecimal) generated by each key. This table lists the characters generated by those ASCII codes.

> **Note:** All ASCII codes in this table are expressed in decimal form.

You can display the characters listed by doing either of the following:

- Using the BASIC statement PRINT CHR$(*code*), where *code* is the ASCII code.
- Pressing ALT and, without releasing it, typing the ASCII code on the numeric keypad.

  For Codes 0-31, the table also lists the standard interpretations. The interpretations are usually used for control functions or communications.

> **Note:** The BASIC program editor has its own special interpretation of some codes and may not display the character listed.

ASCII CHARACTER CODES

Chr	Dec	Hex	Chr	Dec	Hex
NUL	000	00H	SPACE	032	20H
OH	001	01H	!	033	21H
STX	002	02H	”	034	22H
ETX	003	03H	#	035	23H
EOT	004	04H	$	036	24H
ENQ	005	05H	%	037	25H
ACK	006	06H	&	038	26H
BEL	007	07H	’	039	27H
BS	008	08H	(	040	28H
HT	009	09H	)	041	29H
LF	010	0AH	*	042	2AH
VT	011	0BH	+	043	2BH
FF	012	0CH	,	044	2CH
CR	013	0DH	-	045	2DH
SO	014	0EH	.	046	2EH
SI	015	0FH	/	047	2FH
DLE	016	10H	0	048	30H
DC1	017	11H	1	049	31H
DC2	018	12H	2	050	32H
DC3	019	13H	3	051	33H
DC4	020	14H	4	052	34H
NAK	021	15H	5	053	35H
SYN	022	16H	6	054	36H
ETB	023	17H	7	055	37H
CAN	024	18H	8	056	38H
EM	025	19H	~~8~~9	057	39H
SUB	026	1AH	:	058	3AH
ESCAPE	027	1BH	;	059	3BH
FS	028	1CH	<	060	3CH
GS	029	1DH	=	061	3DH
RS	030	1EH	>	062	3EH
US	031	1FH	?	063	3FH

Chr	Dec	Hex	Chr	Dec	Hex
@	064	40H	’	096	60H
A	065	41H	a	097	61H
B	066	42H	b	098	62H
C	067	43H	c	099	63H
D	068	44H	d	100	64H
E	069	45H	e	101	65H
F	070	46H	f	102	66H
G	071	47H	g	103	67H
H	072	48H	h	104	68H
I	073	49H	i	105	69H
J	074	4AH	j	106	6AH
K	075	4BH	k	107	6BH
L	076	4CH	l	108	6CH
M	077	4DH	m	109	6DH
N	078	4EH	n	110	6EH
O	079	4FH	o	111	6FH
P	080	50H	p	112	70H
Q	081	51H	q	113	71H
R	082	52H	r	114	72H
S	083	53H	s	115	73H
T	084	54H	t	116	74H
U	085	55H	u	117	75H
V	086	56H	v	118	76H
W	087	57H	w	119	77H
X	088	58H	x	120	78H
Y	089	59H	y	121	79H
Z	090	5AH	z	122	7AH
[	091	5BH	{	123	7BH
\	092	5CH	|	124	7CH
]	093	5DH	}	125	7DH
^	094	5EH	~	126	7EH
_	095	5FH	DEL	127	7FH

Dec = decimal, Hexadecimal(H), CHR = character,
LF = Line Feed, FF = Form Feed, CR = Carriage Return,
DEL = Rub out

ASCII CHARACTER CODES

ASCII Code	Character	Control Character
000	(null)	NUL
001	☺	SOH
002	☻	STX
003	♥	ETX
004	♦	EOT
005	♣	ENQ
006	♠	ACK
007	(beep)	BEL
008	◘	BS
009	(tab)	HT
010	(line feed)	LF
011	(home)	VT
012	(form feed)	FF
013	(carriage return)	CR
014	♫	SO
015	☼	SI
016	►	DLE
017	◄	DC1
018	↕	DC2
019	‼	DC3
020	¶	DC4
021	§	NAK
022	▬	SYN
023	↨	ETB
024	↑	CAN
025	↓	EM
026	→	SUB
027	←	ESC
028	(cursor right)	FS
029	(cursor left)	GS
030	(cursor up)	RS
031	(cursor down)	US

ASCII CHARACTER CODES

ASCII Code	Character	ASCII Code	Character
032	(space)	067	C
033	!	068	D
034	"	069	E
035	#	070	F
036	$	071	G
037	%	072	H
038	&	073	I
039	,	074	J
040	(	075	K
041	)	076	L
042	*	077	M
043	+	078	N
044	'	079	O
045	-	080	P
046	.	081	Q
047	/	082	R
048	0	083	S
049	1	084	T
050	2	085	U
051	3	086	V
052	4	087	W
053	5	088	X
054	6	089	Y
055	7	090	Z
056	8	091	[
057	9	092	\
058	:	093	]
059	;	094	^
060	<	095	—
061	=	096	'
062	>	097	a
063	?	098	b
064	@	099	c
065	A	100	d
066	B	101	e

ASCII CHARACTER CODES

ASCII Code	Character	ASCII Code	Character
102	f	137	ë
103	g	138	è
104	h	139	ï
105	i	140	î
106	j	141	ì
107	k	142	Ä
108	l	143	Å
109	m	144	É
110	n	145	æ
111	o	146	Æ
112	p	147	ô
113	q	148	ö
114	r	149	ò
115	s	150	û
116	t	151	ù
117	u	152	ÿ
118	v	153	Ö
119	w	154	Ü
120	x	155	¢
121	y	156	£
122	z	157	¥
123	{	158	Pt
124	¦	159	ƒ
125	}	160	á
126	~	161	í
127	⌂	162	ó
128	Ç	163	ú
129	ü	164	ñ
130	é	165	Ñ
131	â	166	ª
132	ä	167	º
133	à	168	¿
134	å	169	⌐
135	c	170	¬
136	ê	171	½

ASCII CHARACTER CODES

ASCII Code	Character	ASCII Code	Character
172	¼	207	╧
173	¡	208	╨
174	«	209	╤
175	»	210	╥
176	░	211	╙
177	▒	212	╘
178	▓	213	╒
179	│	214	╓
180	┤	215	╫
181	╡	216	╪
182	╢	217	┘
183	╖	218	┌
184	╕	219	█
185	╣	220	▄
186	║	221	▌
187	╗	222	▐
188	╝	223	▀
189	╜	224	α
190	╛	225	β
191	┐	226	Γ
192	└	227	π
193	┴	228	Σ
194	┬	229	σ
195	├	230	μ
196	─	231	τ
197	┼	232	Φ
198	╞	233	Θ
199	╟	234	Ω
200	╚	235	δ
201	╔	236	∞
202	╩	237	Ø
203	╦	238	ε
204	╠	239	∩
205	═	240	≡
206	╬	241	±

ASCII CHARACTER CODES

ASCII Code	Character	ASCII Code	Character
242	≥	249	●
243	≤	250	•
244	⌠	251	√
245	⌡	252	η
246	÷	253	^{2}
247	≈	254	■
248	°	255	(blank 'FF')

BIBLIOGRAPHY

1. Microprocessors and interfacing - Prgramming 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. VAX 11/780 hardware reference manual

9. Programmer's guide to the IBM pc by Peter Norton

10. The VT220 user manual

11. DOS reference manual

12. Designing and implementing Local Area Networks by Chorafas