NCR

NCR DECISION MATE V

User Information

FEDERAL COMMUNICATIONS COMMISSION RADIO FREQUENCY INTERFERENCE STATEMENT

WARNING: This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

CP/M-80, CP/M-86 and DR-GRAPH are registered trademarks of Digital Research Inc.
GW-BASIC, MS-BASIC-80, MS-BASIC-86 and MS-DOS are trademarks of Microsoft Inc.

Second Edition, June 1983
This document contains the latest information available at the time of publication. However, NCR reserves the right to modify the contents of this material at any time. Also, all features, functions, and operations described herein may not be marketed by NCR in all parts of the world. Therefore, before using this document, consult your nearest dealer of NCR office for the information that is applicable and current.

FOREWORD

Congratulations on your selection of NCR DECISION MATE V as your new professional business partner. Using state-of-the-art technology and a modular design philosophy, NCR DECISION MATE V offers features for both the experienced and first-time user, providing flexibility in choosing processing capabilities and options. Best of all, your NCR DECISION MATE V is very friendly, with ease of operation perhaps its most attractive feature.

Regardless of your computer background, take time to read this manual, it contains important information about your computer.

The manual, arranged in five sections, is the primary introduction to your NCR DECISION MATE V. The first two sections describe installation procedures, and introduce the main elements of a computer, the hardware and software. The next section concentrates on operations — how to use the hardware and how to start the system. And, finally, the last sections describe what to do should you have problems and other helpful hints for the care of your system.

After you have read this manual, you will be thoroughly comfortable with your NCR DECISION MATE V and able to perform many functions. You will then want to study the other manuals that fully describe the capabilities of your computer including those manuals referenced in *System Introduction.*

One final point should be made about your computer. NCR DECISION MATE V is offered from a company committed to computer technology. Ongoing research and development will produce new features and options, allowing you to easily upgrade your computer to match your processing needs. Contact your NCR representative or dealer who will always be up-to-date on current offerings.

NCR DECISION MATE V
USER INFORMATION

CONTENTS

INSTALLATION

SYSTEM INTRODUCTION

HARDWARE OPERATION

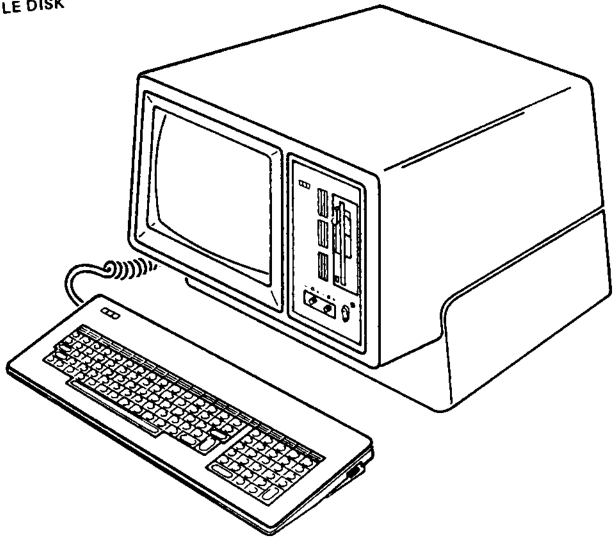
WHAT IF?

HELPFUL HINTS

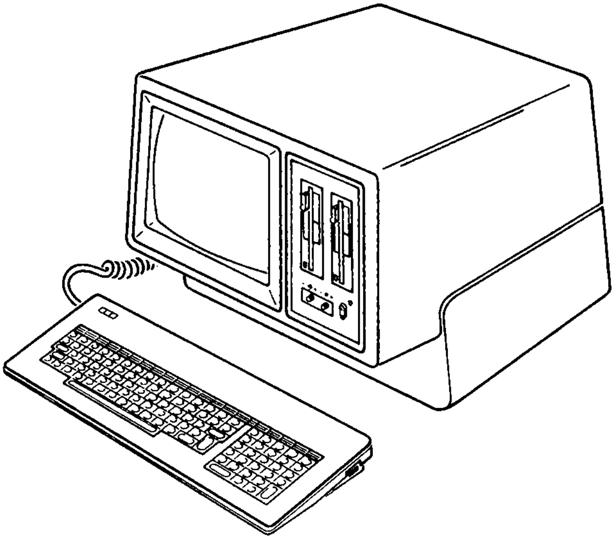
Printed in the Federal Republic of Germany

NCR DECISION MATE V INSTALLATION

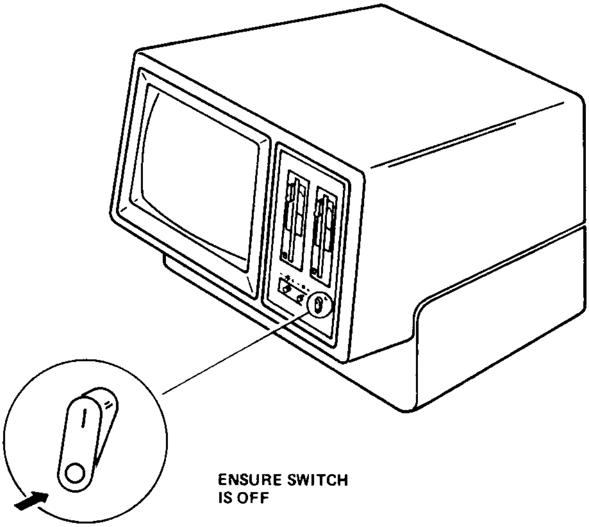
WITH FIXED WINCHESTER AND FLEXIBLE DISK



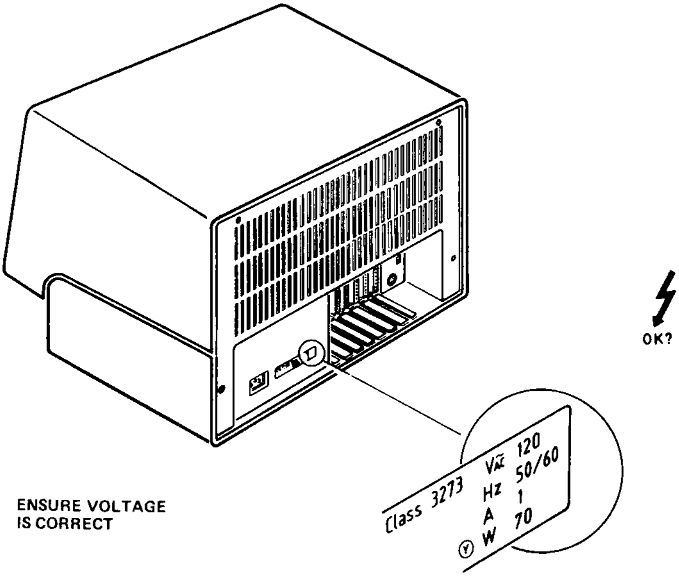
WITH 2 FLEXIBLE DISKS



CHECK VOLTAGE



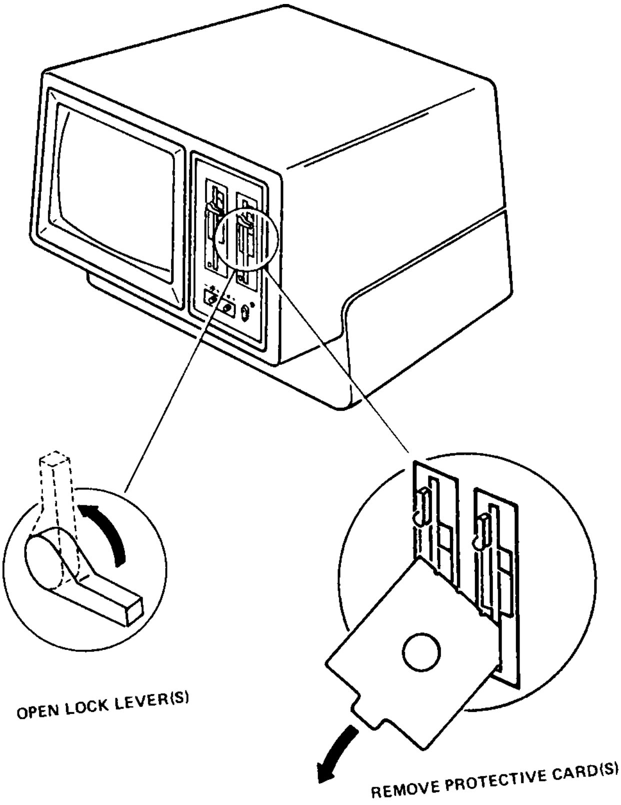
ENSURE SWITCH IS OFF



OK?

ENSURE VOLTAGE IS CORRECT

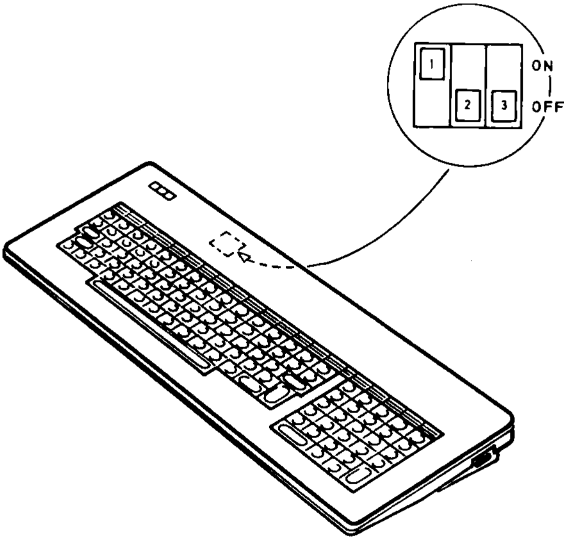
PREPARE DISK DRIVES



OPEN LOCK LEVER(S)

REMOVE PROTECTIVE CARD(S)

SET LANGUAGE CODE SWITCHES



VERSION 1

Language Code

	1	2	3
US English	○	○	○
UK/Int. English	●	○	○
Danish	○	●	○
German	●	●	○
Swedish/Finnish	○	○	●
Norwegian	●	○	●
Spanish	○	●	●
Italian	●	●	●
	1	2	3

○ = off ● = on

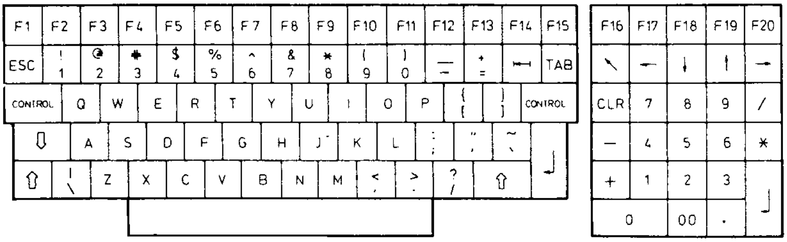
VERSION 2

Language Code

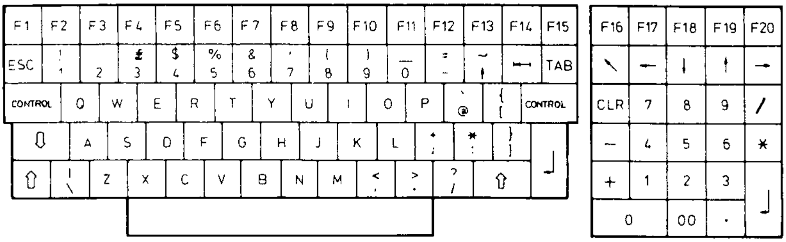
	1	2	3
Swiss-German	○	○	○
Swiss-French	●	○	○
French	○	●	○
Canadian/Australian	●	●	○
Canadian(Bilingual)	○	○	●
South African	●	○	●
Portuguese	○	●	●
Yugoslavian	●	●	●
	1	2	3

○ = off ● = on

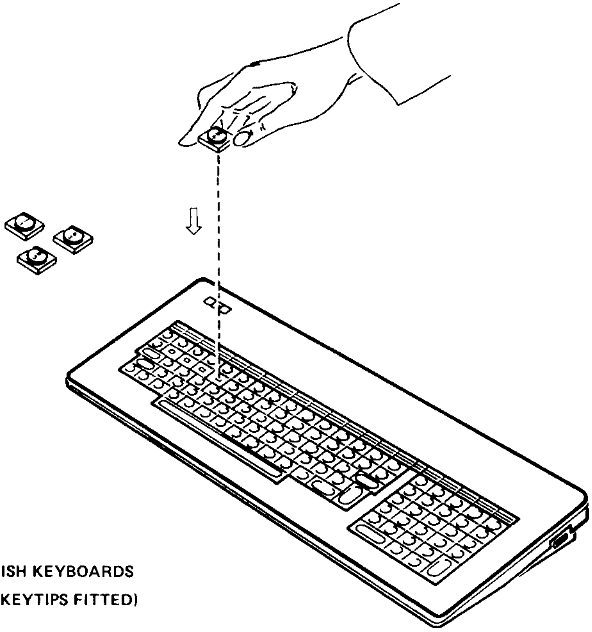
FIT KEYTIPS


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = ⟼ TAB
↖ ← ↓ ↑ →
CONTROL Q W E R T Y U I O P { [}] CONTROL
CLR 7 8 9 /
⇩ A S D F G H J K L : ; " ' ~ `
– 4 5 6 *
⇧ | \ Z X C V B N M < , > . ? / ⇧
+ 1 2 3
0 00 .

US ENGLISH KEYBOARD LAYOUT

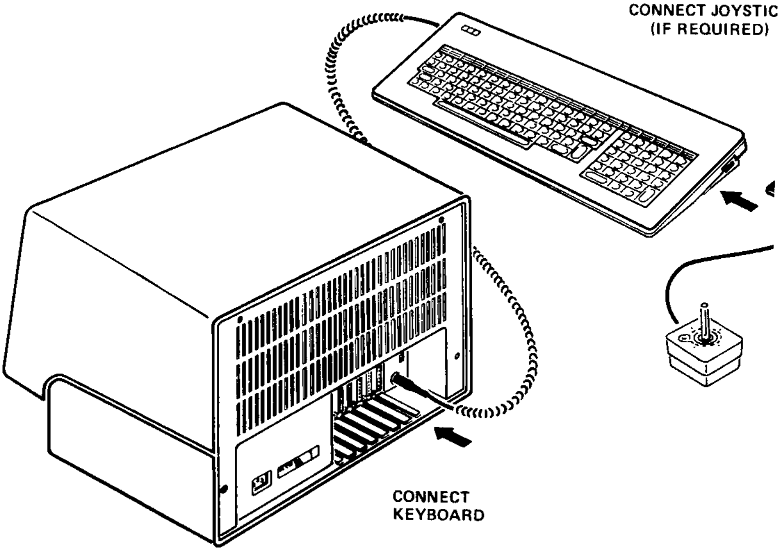

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 2 £ 3 $ 4 % 5 & 6 ' 7 (8) 9 _ 0 = - ~ ↑ ⟼ TAB
↖ ← ↓ ↑ →
CONTROL Q W E R T Y U I O P ` @ { [CONTROL
CLR 7 8 9 /
⇩ A S D F G H J K L + ; * : }]
– 4 5 6 *
⇧ | \ Z X C V B N M < , > . ? / ⇧
+ 1 2 3
0 00 .

UK/INTERNATIONAL KEYBOARD LAYOUT

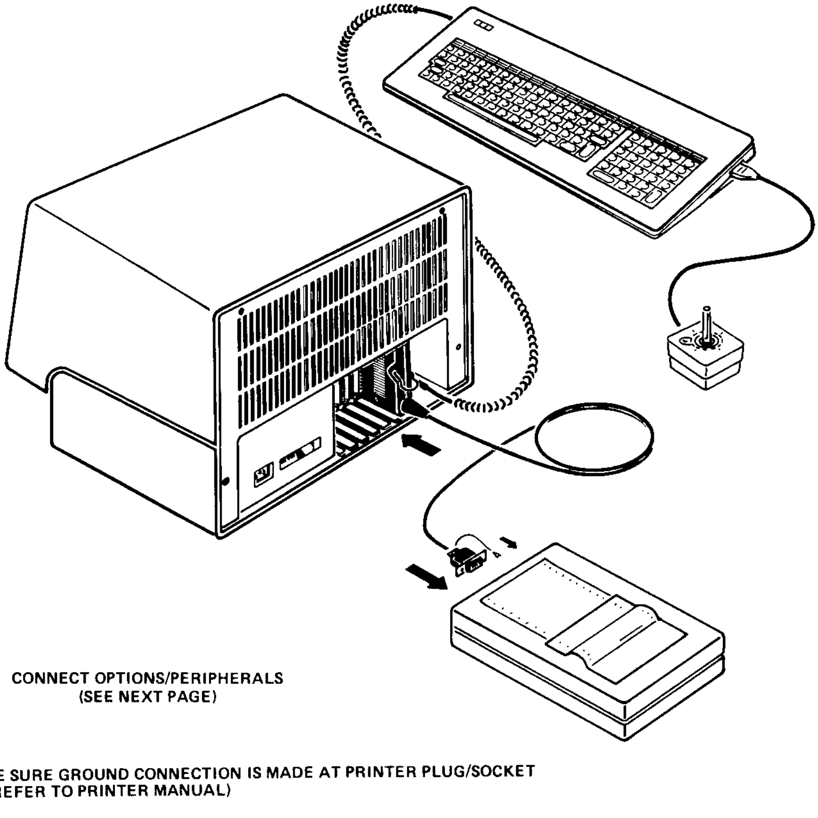


(US - ENGLISH KEYBOARDS
HAVE ALL KEYTIPS FITTED)

CONNECT KEYBOARD



CONNECT OPTIONS



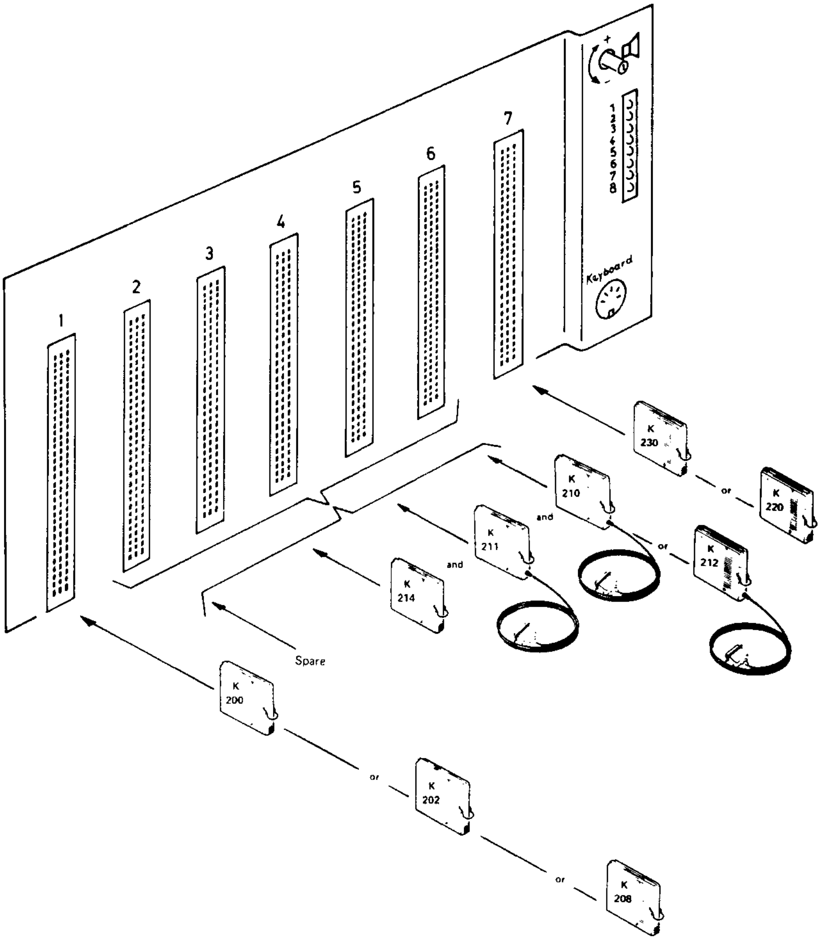
CONNECT OPTIONS/PERIPHERALS
(SEE NEXT PAGE)

BE SURE GROUND CONNECTION IS MADE AT PRINTER PLUG/SOCKET
(REFER TO PRINTER MANUAL)

OPTION LOCATIONS

- Fit any peripheral adapter that has a cable to the rightmost slot (slot 6)
- If two adapters have cables always fit printer adapter (K210 or K212) to slot 6 and communication adapter (K211) to slot 5
- Do not leave any slots between adapters open

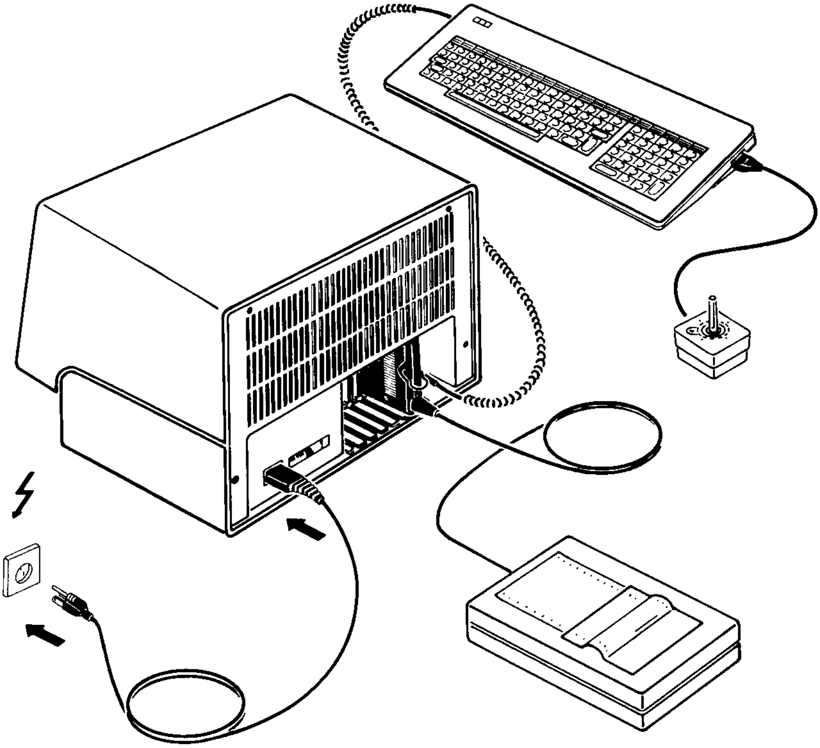
Note slot 7 is reserved for Diagnostic module or 16-bit processor, slot 1 is reserved for Memory expansion module.



K200	64 KB MEMORY EXPANSION
K202	192 KB MEMORY EXPANSION
K208	448 KB MEMORY EXPANSION
K210	CENTRONICS PARALLEL PRINTER INTERFACE
K211	RS-232C COMMUNICATION INTERFACE
K212	RS-232C SERIAL PRINTER INTERFACE
K214	BLANK ADAPTER AND BUS CONNECTOR
K220	DIAGNOSTIC MODULE
K230	8/16-BIT UPGRADE KIT

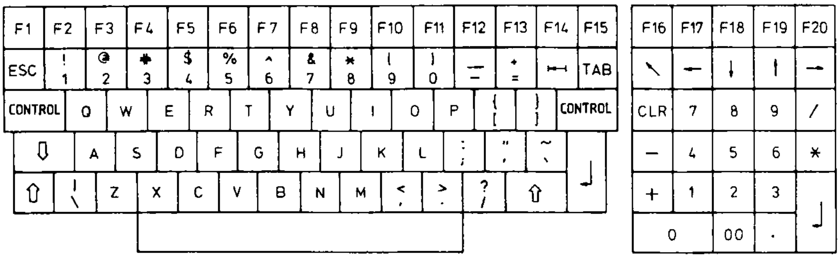
OPTION/PERIPHERAL LOCATIONS

CONNECT POWER

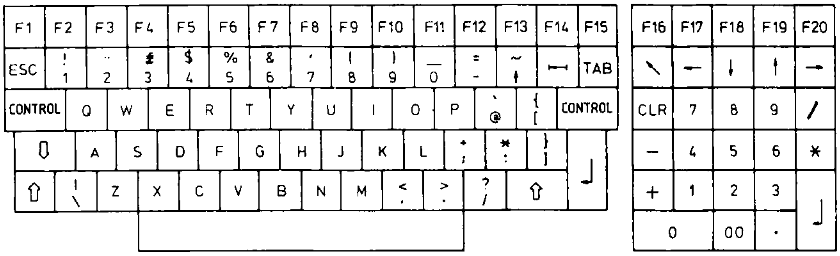


CONNECT POWER CABLE

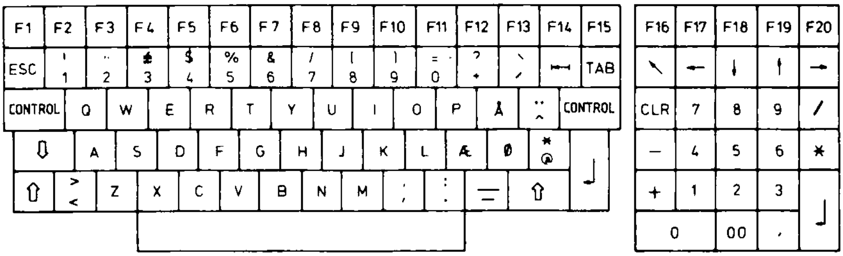
KEYBOARD LAYOUTS


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = TAB
CONTROL Q W E R T Y U I O P { [}] CONTROL
A S D F G H J K L : ; " ' ~ `
| \ Z X C V B N M < , > . ? /
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 .

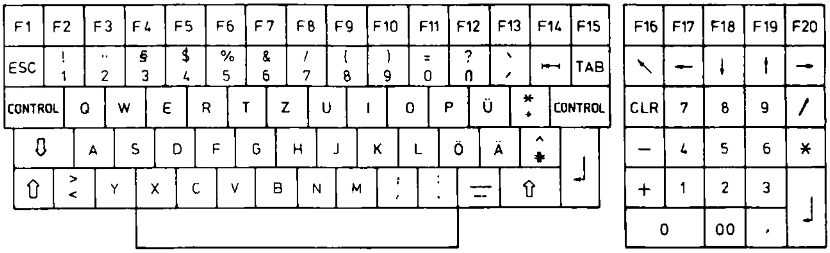
US ENGLISH


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 £ 3 $ 4 % 5 & 6 ' 7 (8) 9 _ 0 = - ~ ↑ TAB
CONTROL Q W E R T Y U I O P ` @ { [CONTROL
A S D F G H J K L + ; * : }]
| \ Z X C V B N M < , > . ? /
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 .

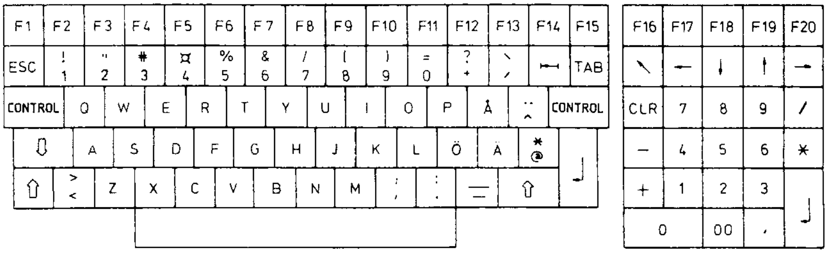
UK/INTERNATIONAL ENGLISH


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 £ 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? + ` ´ TAB
CONTROL Q W E R T Y U I O P Å ¨ ^ CONTROL
A S D F G H J K L Æ Ø * @
> < Z X C V B N M ; , : . _ -
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 ,

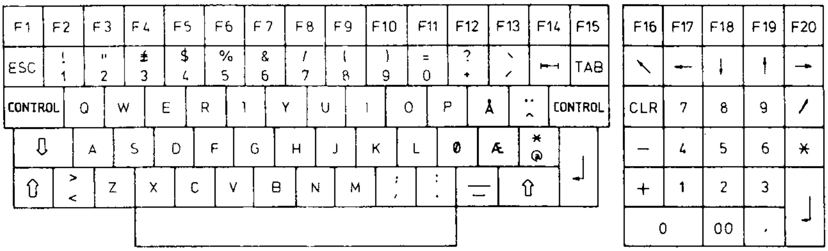
DANISH


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 § 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? ß ` ´ TAB
CONTROL Q W E R T Z U I O P Ü * + CONTROL
A S D F G H J K L Ö Ä ^ #
> < Y X C V B N M ; , : . _ -
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 ,

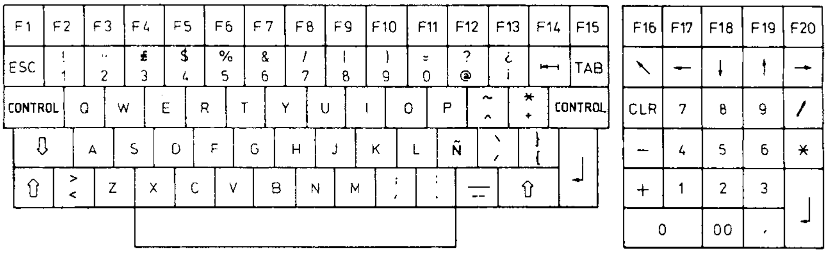
GERMAN


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 # 3 ¤ 4 % 5 & 6 / 7 (8) 9 = 0 ? + ` ´ TAB
CONTROL Q W E R T Y U I O P Å ¨ ^ CONTROL
CLR 7 8 9 /
A S D F G H J K L Ö Ä * @
− 4 5 6 *
> < Z X C V B N M ; , : .
+ 1 2 3
0 00 .

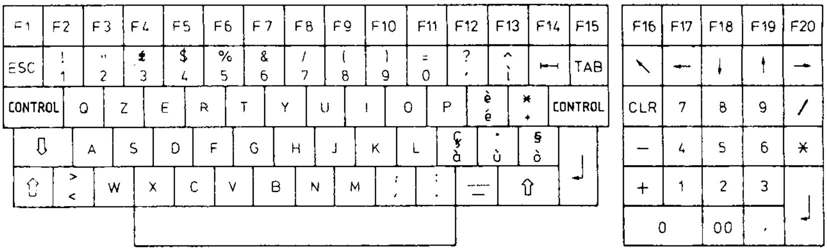
SWEDISH,FINNISH


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 £ 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? + ` ´ TAB
CONTROL Q W E R T Y U I O P Å ¨ ^ CONTROL
CLR 7 8 9 /
A S D F G H J K L Ø Æ * @
− 4 5 6 *
> < Z X C V B N M ; , : .
+ 1 2 3
0 00 .

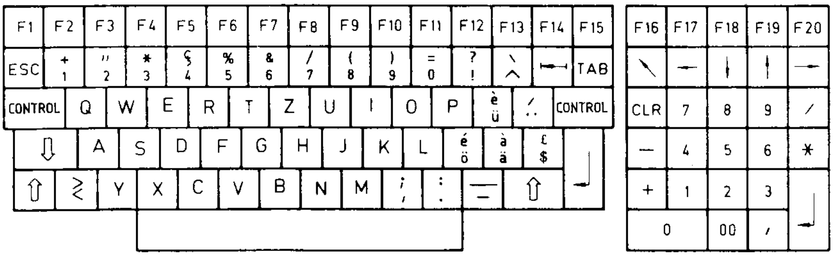
NORWEGIAN


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 ¨ 2 £ 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? @ ¿ ¡ TAB
CONTROL Q W E R T Y U I O P ~ ^ * + CONTROL
CLR 7 8 9 /
A S D F G H J K L Ñ ` ´ } {
− 4 5 6 *
> < Z X C V B N M ; , : .
+ 1 2 3
0 00 .

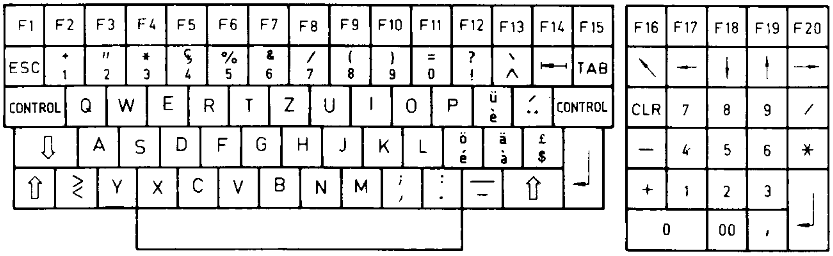
SPANISH


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 £ 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? ' ^ ì TAB
CONTROL Q Z E R T Y U I O P è é * + CONTROL
CLR 7 8 9 /
A S D F G H J K L ç à · ù § ò
− 4 5 6 *
> < W X C V B N M ; , : .
+ 1 2 3
0 00 .

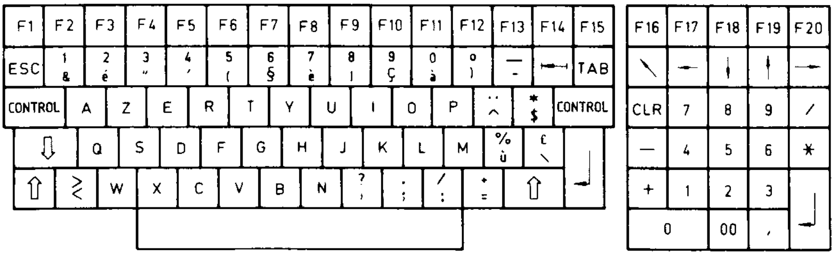
ITALIAN


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC + 1 " 2 * 3 ç 4 % 5 & 6 / 7 (8) 9 = 0 ? ! \ ^ TAB
CONTROL Q W E R T Z U I O P è ü CONTROL
A S D F G H J K L é ö à ä £ $
Y X C V B N M ; , : . — _
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 ,

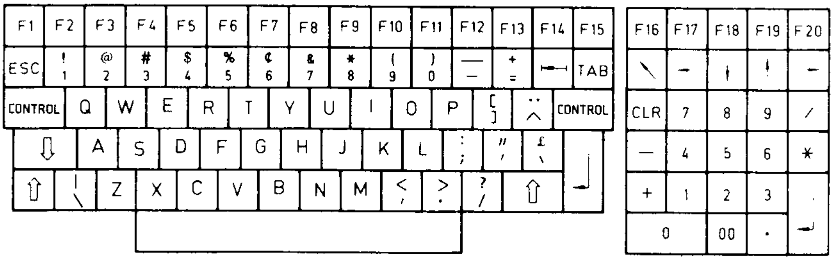
SWISS-GERMAN


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC + 1 " 2 * 3 ç 4 % 5 & 6 / 7 (8) 9 = 0 ? ! \ ^ TAB
CONTROL Q W E R T Z U I O P ü è CONTROL
A S D F G H J K L ö é ä à £ $
Y X C V B N M ; , : . — _
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 ,

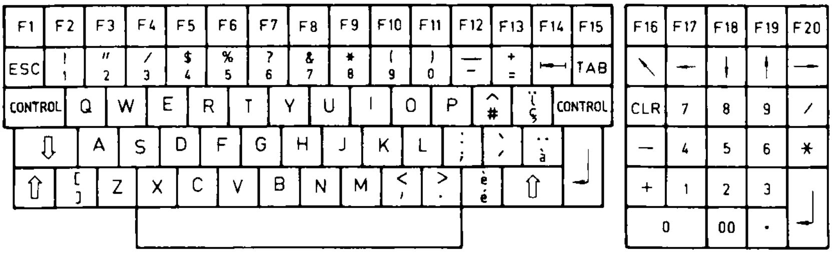
SWISS-FRENCH


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC 1 & 2 é 3 " 4 ' 5 (6 § 7 è 8 ! 9 ç 0 à °) — - TAB
CONTROL A Z E R T Y U I O P ¨ ^ * $ CONTROL
Q S D F G H J K L M % ù £ \
W X C V B N ? , . ; / : + =
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 ,

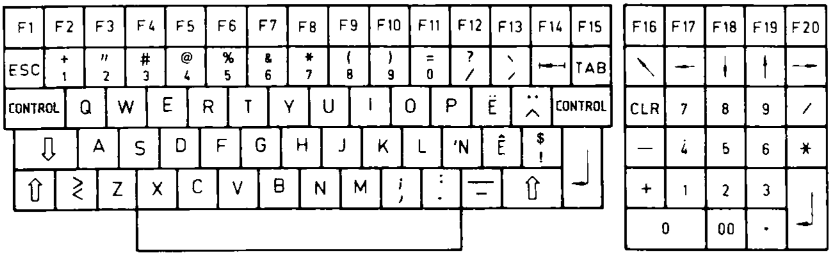
FRENCH


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 @ 2 # 3 $ 4 % 5 ¢ 6 & 7 * 8 (9) 0 — — + = TAB
CONTROL Q W E R T Y U I O P [ ] ¨ ^ CONTROL
A S D F G H J K L : ; " ' £ \
| \ Z X C V B N M < , > . ? /
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 .

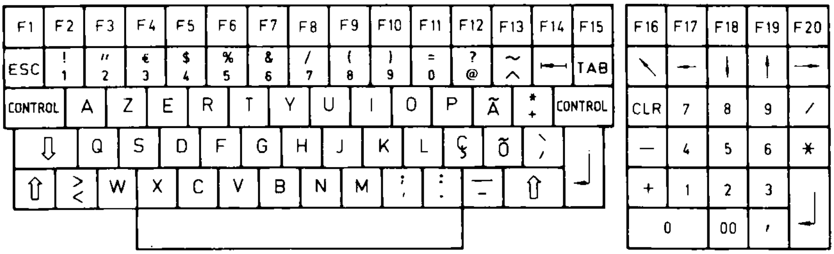
AUSTRALIAN (PRELIMINARY)


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 / 3 $ 4 % 5 ? 6 & 7 * 8 (9) 0 — - + = TAB
CONTROL Q W E R T Y U I O P ^ # ¨ ç CONTROL
A S D F G H J K L : ; ` ´ ¨ à
[ ] Z X C V B N M < , > . è é
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 .

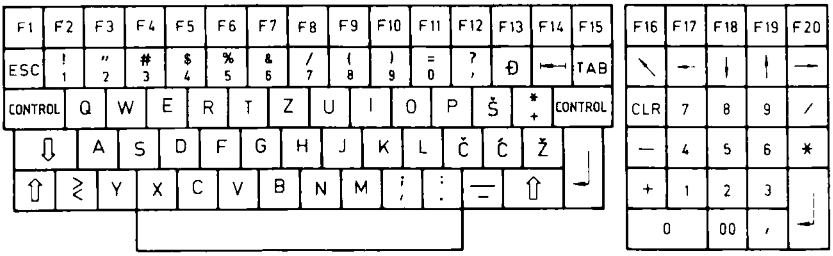
CANADIAN (BILINGUAL — PRELIMINARY)


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC + 1 " 2 # 3 @ 4 % 5 & 6 * 7 (8) 9 = 0 ? / ` ´ TAB
CONTROL Q W E R T Y U I O P Ë ¨ ^ CONTROL
A S D F G H J K L 'N Ê $!
≥ ≤ Z X C V B N M ; , : . — —
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 .

SOUTH AFRICAN


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 £ 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? @ ~ ^ TAB
CONTROL A Z E R T Y U I O P Ã * + CONTROL
Q S D F G H J K L Ç Õ ` ´
≥ ≤ W X C V B N M ; , : . — —
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 ,

PORTUGUESE


F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20
ESC ! 1 " 2 # 3 $ 4 % 5 & 6 / 7 (8) 9 = 0 ? ' Đ TAB
CONTROL Q W E R T Z U I O P Š * + CONTROL
A S D F G H J K L Č Ć Ž
≥ ≤ Y X C V B N M ; , : . — —
CLR 7 8 9 /
— 4 5 6 *
+ 1 2 3
0 00 ,

YUGOSLAVIAN

PRINTERS

RECOMMENDED PRINTERS FOR NCR DECISION MATE V.

- NCR 6411 (Serial or Parallel)
- NCR 6442 (Parallel only)
- NCR 6455 (Serial only)

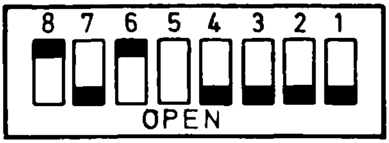
Strappings shown are for use with NCR DECISION MATE V. For further information refer to relevant printer documentation.

- NCR 6411 Printer Owner's Manual
- NCR 6442 Matrix Printer Operator Information
- NCR 6442 Matrix Printer Hardware Installation
- NCR 6455 Printer Owner's Manual

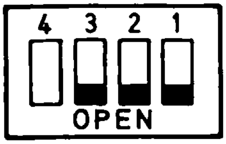
Only one printer may be connected to the system.

NCR 6411
Serial mode with RS-232C peripheral adapter (K212)

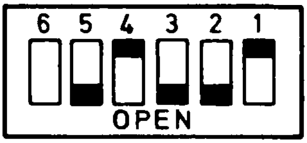
SWITCH SW21			
SWITCH NUMBER	POSITION	FUNCTION	
1	OPEN	NUMBER OF STOP BIT, 1	
2	OPEN	SD	
3	OPEN	PARITY CHECK	EVEN
4	OPEN	PARITY CHECK	
5		NO FUNCTION	
6	CLOSED	7-BIT DATA	
7	OPEN	DATA PROTOCOL	X-ON/X-OFF
8	CLOSED	DATA PROTOCOL	



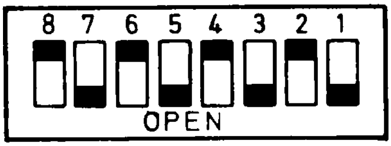
SWITCH SW22			
SWITCH NUMBER	POSITION	FUNCTION	
1	OPEN	DATA TRANSMISSION SPEED	9600 BPS
2	OPEN	DATA TRANSMISSION SPEED	
3	OPEN	DATA TRANSMISSION SPEED	
4		NO FUNCTION	



SWITCH SW23		
SWITCH NUMBER	POSITION	FUNCTION
1	CLOSED	RS-232C
2	OPEN	RS-232C
3	OPEN	DSR, RS-232C
4	CLOSED	DSR, RS-232C
5	OPEN	DSR, RS-232C
6		NO FUNCTION



SWITCH SW24		
SWITCH NUMBER	POSITION	FUNCTION
1	OPEN	DTR, X-ON/X-OFF
2	CLOSED	DTR, X-ON/X-OFF
3	OPEN	RTS
4	CLOSED	RTS
5	OPEN	USE OF CTS
6	CLOSED	USE OF CTS
7	OPEN	CD INVALID
8	CLOSED	CD INVALID



NCR 6411

Parallel mode with Centronics peripheral adapter (K210)

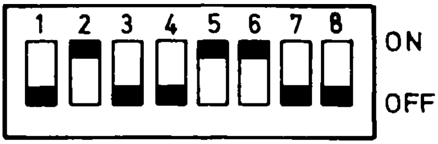
NO SPECIAL STRAPPING REQUIRED

NCR 6442
Parallel mode with Centronics peripheral adapter (K210)

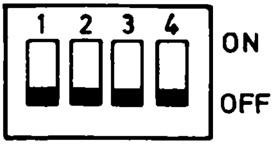
NO SPECIAL STRAPPING REQUIRED

NCR 6455
Serial mode with RS-232C peripheral adapter (K212)

SWITCH SW1		
SWITCH NUMBER	POSITION	FUNCTION
1	OFF	REMOTE MODE AT POWER ON
2	ON	LF SWITCH ACTS AS FORMS FEED
3	OFF	INDIVIDUAL HORIZONTAL TAB CLEARED
4	OFF	AUTO RETURN DISABLED
5	ON	X-ON/X-OFF PROTOCOL
6	ON	INTERRUPT SIGNAL SENT AT ALARM
7	OFF	FORMS LENGTH SWITCH DISABLE
8	OFF	NORMAL OPERATION (ERROR MONITOR)

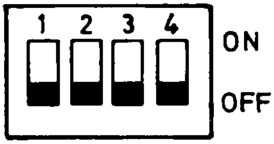


		SWITCH SW2
SWITCH NUMBER	POSITION	FUNCTION
1	OFF	CONSTANT PITCH
2	OFF	CONSTANT PITCH
3	OFF	NORMAL OPERATION (PAPER OUT)
4	OFF	NORMAL OPERATION (TEST)



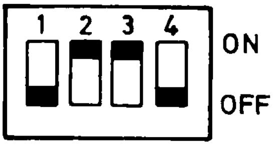
SWITCH SW3

SWITCH NUMBER	POSITION	FUNCTION
1	OFF	6 LINES PER INCH
2	OFF	10 CHARACTERS PER INCH
3	OFF	10 CHARACTERS PER INCH
4	OFF	10 CHARACTERS PER INCH

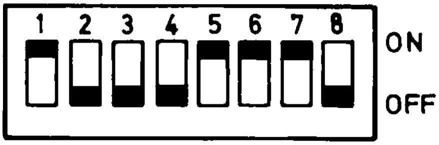


SWITCH SW4

SWITCH NUMBER	POSITION	FUNCTION
1	OFF	LOCAL LINE FEED SWITCH DISABLED
2	ON	RECEIVE/TRANSMIT PARITY CHECK – EVEN
3	ON	RECEIVE/TRANSMIT PARITY CHECK – EVEN
4	OFF	HALF DUPLEX



INTERFACE BOARD SWITCH SW1		
SWITCH NUMBER	POSITION	FUNCTION
1	ON	DATA SET READY ON
2	OFF	CLEAR TO SEND OFF
3	OFF	CARRIER DETECT CONTROL NORMAL (MODEM)
4	OFF	REVERSE CHANNEL ACTIVE HIGH
5	ON	TEST
6	ON	2K BUFFER
7	ON	NOT USED
8	OFF	HAMMER ENABLE



SPEED 8 = 9600 BAUD
LINES/FORM = 72
SELECT RS-232C X-ON/X-OFF BY CONTROL PANEL

TECHNICAL SPECIFICATIONS

Size Processor	Height 378 mm (14.9 in.) Width 461 mm (18.1 in.) Depth 370 mm (14.6 in.)
Keyboard	Height 37 mm (1.5 in.) Width 430 mm (16.9 in.) Depth 216 mm (8.5 in.)
Weight Processor Keyboard	22 kg (48.5 lb) 2 kg (4.4 lb)
Voltage	Nominal 100 volts ac Range 90 to 107 volts ac 120 104 to 127 220 198 to 235 230 207 to 246 240 216 to 257
Frequency	50/60 Hertz (49 to 61 Hertz)
Power (average)	70 Watts (basic computer without peripherals)
Temperature °C °F	Operating 10 to 35 Storage -10 to 50 Transit -40 to 60 (50 to 95) (14 to 120) (-40 to 140)
Temperature Change	10°C per hour (18°F per hour)
Relative Humidity	Range: 20% to 80%
Cable Lengths	Power 2.5 to 3.5 metres (8.2 to 11.5 feet) Keyboard 0.5 metres (1.6 feet) extendable Centronics Peripheral Adapter 2.0 metres (6.6 feet) RS-232 Peripheral Adapter 2.0 metres (6.6 feet)
Noise Rating Level	Idle mode 33 dB (A) Flex. Disk 41dB(A)
Product Safety	USA UL 478 UL listing mark used Canada CSA 22.2 – 154, CSA monogram used Europe IEC 380, Inhouse verification Germany VDE 0806, GS label granted by trade association
Radio Protection	USA FCC Docket No. 20780, Class B Germany VDE 0871, Class A certified by German Federal Post (FTZ)
Radiation Emission	USA Public Law 90 – 602 DHEW Publication No. (FDA) 75 – 8003 Germany X-Ray Emission Regulations

SYSTEM INTRODUCTION

Your NCR DECISON MATE V comprises two separate, though interrelated, elements: hardware and software. Hardware encompasses any of the physical components of the computer, while software defines the programs that "drive" it. In this section, you learn about the standard hardware of the NCR DECISION MATE V, plus software and options that you can order separately.

HARDWARE

The standard version of NCR DECISION MATE V consists of a processor unit and keyboard built and tested to meet the highest engineering and safety standards.

STANDARD CONFIGURATION

The processor unit consists of a compact cabinet, ideally suited for desk-top use, that is made up of the following major modules:

- 12-inch cathode ray tube monitor (monochrome or color)
- Two 5 1/4-inch flexible disk drive units, or
  One flexible disk and one fixed Winchester disk drive
- Accessible connectors for peripherals and features
- Processor (circuit) board with a 64 KB memory capacity
- Power supply

NOTE: A kilobyte (KB) = 1024 bytes (characters). A 64 KB memory capacity holds over 65,000 characters.

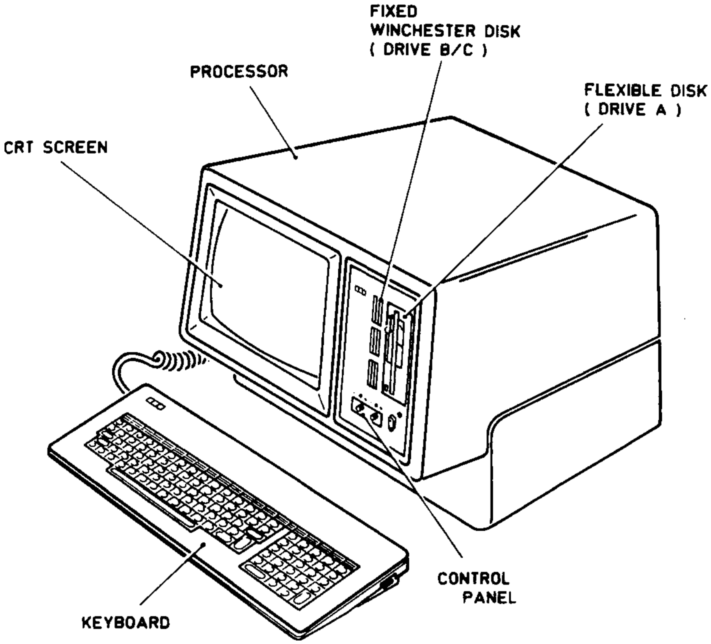
The processor is available in two types, 8-bit for use with 8-bit software, or dual 8/16-bit for use with either 8-or 16-bit software.

The keyboard is a freestanding unit that is connected to the processor by a coiled cable. The keyboard has an alphanumeric section with control and function keys, and, for rapid entry of numeric-only data, a 10-key numeric keypad.

NCR DECISION MATE V uses industry standard, 5 1/4-inch flexible disks with double-sided, double density format, providing a storage capacity of 360 KB per disk. While any disk that meets these specifications may be used, be sure to purchase quality disks either from NCR or another reputable manufacturer.

The integrated fixed Winchester Disk offers a far greater storage capacity. This disk can store ten megabytes (ten million characters).

The major hardware elements of a typical NCR DECISION MATE V are shown below. More detailed information on the function of these elements and how to operate each one are provided in — *Hardware Operation.*


FIXED WINCHESTER DISK (DRIVE B/C)
PROCESSOR
FLEXIBLE DISK (DRIVE A)
CRT SCREEN
CONTROL PANEL
KEYBOARD

OPTIONS AND ENHANCEMENTS

NCR DECISION MATE V is designed for worldwide use and, with the various features and kits that are or will be available, can easily be changed to match your requirements as they expand.

The features available include the following options:

- Dual 8/16-bit processor upgrade
- Memory expansion up to a maximum of 512KB
- Peripheral adapters (Centronics or RS-232C compatible) for connection of printers, terminals or modems
- Keyboard and language sets for most languages

- Blank interface adapter for designers who wish to connect some other equipment to the computer
- Diagnostic module for speedy problem identification
- Provision for connecting a joystick

Dual 8/16-Bit Processor Upgrade

You can plug this module into the rightmost slot at the rear of the processor and convert your 8-bit processor into a dual 8/16-bit processor, allowing the use of both 8-bit and 16-bit operating systems, and any of the compatible application software packages.

Memory Expansion

Simply determine how much memory you require, select the appropriate kit, and plug it into the leftmost slot at the rear of the processor. Three memory expansion kits are available, these are:

- K200 — increases memory capacity from 64KB to 128KB
- K202 — increases memory capacity from 64KB to 256KB
- K208 — increases memory capacity from 64KB to 512KB

Peripheral Adapters

Most RS-232C and Centronics compatible devices can be connected to the NCR DECISION MATE V. You simply select the correct adapter, plug it into one of the slots (2 to 6) at the rear of the processor, and connect the cable to the peripheral you wish to use. All adapters are complete with a 2-metre cable and a suitable connecting plug. Currently the following adapters are available:

- K210 — for the connection of a Centronics compatible printer (parallel input)
- K211 — for the connection of an RS-232C compatible modem
- K212 — for the connection of an RS-232C compatible printer (serial input)

PRINTERS

In theory, any RS-232C (Serial) or Centronics (parallel) compatible printer may be used with the DECISION MATE V: however, NCR recommends the following printers:

- NCR 6411 — with either parallel (Centronics) or serial (RS-232C) peripheral adapters
- NCR 6442 — with parallel (Centronics) peripheral adapter
- NCR 6455 — with serial (RS-232C) peripheral adapter

SOFTWARE

Generally, a computer uses three types of software: operating system software, application software, and programming language software. The following definitions may clarify these three categories.

- Operating system software controls the execution of computer programs and includes such functions as managing memory and disk space, and handling data and files.
- Application software, a collection of programs written for or by the user, performs a specific processing function.
- Programming language software, consisting of commands and the software to interpret them, provides the elements for a computer program.

Better than any definition, however, the importance of software can best be understood in terms of what it can do for you.

OPERATING SYSTEM SOFTWARE

Of all software, the operating system is the most important because, without it, you cannot use your computer or any other software. Besides containing control programs, the operating system includes all information needed to interface the hardware with the software. An operating system is usually on disk and is automatically read into the computer once the disk is inserted in the disk drive. Today, many standard operating systems are available. These "generic" packages are written for specific microprocessors and are, therefore, sometimes described as 8- or 16-bit software. Each of these operating systems, however, usually require some alteration to define the characteristics of the hardware.

Three highly popular operating systems are implemented for NCR DECISION MATE V: CP/M®-80, CP/M®-86 and MS™-DOS. Depending on the model of your computer, you can use one or all of these.

CP/M is an abbreviation for Control Program for Microprocessors: CP/M-80 is an operating system developed by Digital Research, Inc. for 8-bit processors.

CP/M-86, also developed by Digital Research, is an operating system for 16-bit processors. This software is fully compatible with CP/M-80 allowing all files to be used without conversion, it also allows you access to all CP/M-compatible, 16-bit application software.

MS-DOS is a Disk Operating System developed by Microsoft Corporation for 16-bit processors. This software uses a different file structure from CP/M-86, but is fully compatible with IBM file format and, like CP/M-86, allows access to a number of 16-bit application packages.

These operating systems were selected for NCR DECISION MATE V not only because they provide optimum system performance, but also because they provide the largest possible selection of both 8- and 16-bit application software.

APPLICATION/LANGUAGE SOFTWARE

Walk into any computer store today and look at the number of application packages. These packages, offered from many different companies, perform specific processing functions to save you the time and effort of doing your own programming. In theory, you simply insert the application disk into your computer, specify a few parameters describing your hardware, and begin processing. In fact, not all application software is so easy to use or offers good system performance. Because an application runs on your computer, doesn't necessarily mean it is the best one.

To help guide you in selecting applications, NCR has tested numerous packages on NCR DECISION MATE V and those applications that have proved so successful are recommended and distributed through NCR.

Besides the application packages, language packes are available should you want to write your own programs.

SOFTWARE PUBLICATIONS

The following publications provide detailed information on the software used with the NCR DECISION MATE V:

- Operating systems — CP/M-80, CP/M-86, MS-DOS
- Languages — MS-BASIC-80, MS-BASIC-86, GW-BASIC
- Applications — DR-GRAPH-80, DR-GRAPH-86

HARDWARE OPERATION

INTRODUCTION

Having installed NCR DECISION MATE V, you are now ready to learn how to operate the computer. This section describes the following components:

- Flexible disks and flexible disk drives
- Fixed Winchester disk drive
- Control panel
- CRT (cathode ray tube) screen
- Keyboard.

Read this entire section, which gives an overview of operating the computer. Then, follow the procedure for startup, and practice using the computer as described at the end of this section.

FLEXIBLE DISKS

NCR DECISION MATE V uses 5 1/4-inch double-sided flexible disks for the storage of software and data files. Examine the disk supplied with your computer. Carefully remove the disk from its protective envelope and, referring to Figure 3.1, identify the following items:

- Disk label
- Recording slot in disk jacket
- Index hole
- Write protect cutout
- Jacket
- Envelope

FLEXIBLE DISK HANDLING

At all times, except when the flexible disk is installed in the computer, it must be in its envelope. Keep these envelopes in the container box.

Disks containing important data such as, operating system and application software should be protected by fitting a 'write protect tab'. This ensures that the important data is not overwritten and destroyed. Operating system software supplied by NCR is permanently protected.

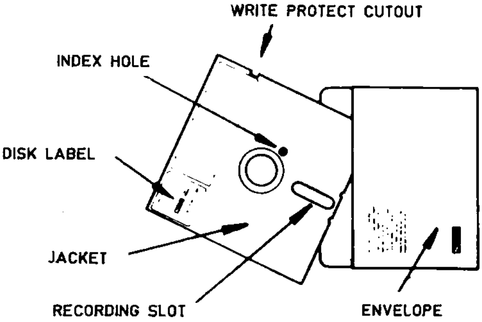


Figure 3.1 Flexible disk

Flexible disks must be permitted to reach the same temperature and humidity as that of the computer before they are used. If flexible disks are not kept in the same area as the computer, move them near the computer at least one hour before they are used.

FLEXIBLE DISK DRIVES

The NCR DECISION MATE V contains two similar 5 1/4-inch flexible disk drives. The leftmost drive is known as drive "A", the rightmost drive is known as drive "B". Each drive has a slot where the flexible disk is inserted, a locking device, and a red LED indicator, see Figure 3.2.

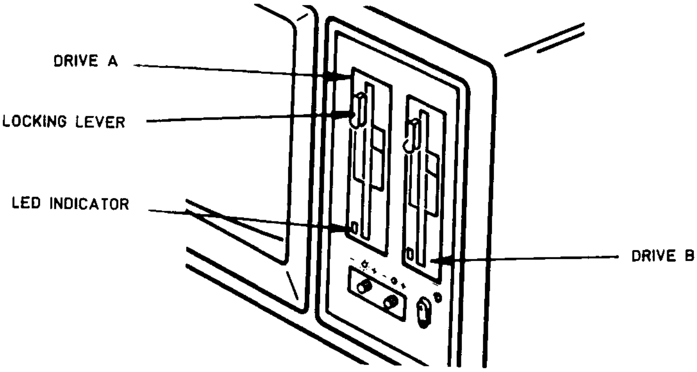


Figure 3.2 Flexible disk drives

The locking lever is turned counterclockwise to allow a flexible disk to be loaded into the drive, and clockwise to lock the flexible disk in the drive.

The red LED indicator will turn on whenever a drive is busy (data is either being read from or written to the flexible disk). Only one disk drive is busy at a time.

LOADING A DISK

To load a flexible disk into a drive unit proceed as follows:

- Open the disk drive by turning the locking lever counterclockwise to the vertical position as in Figure 3.2
- Select the correct flexible disk and remove it from the envelope
- Insert the flexible disk into the disk drive as shown in Figure 3.3. The recording slot must enter the disk drive first with the label side of the jacket facing the CRT screen.
- Turn the locking lever clockwise

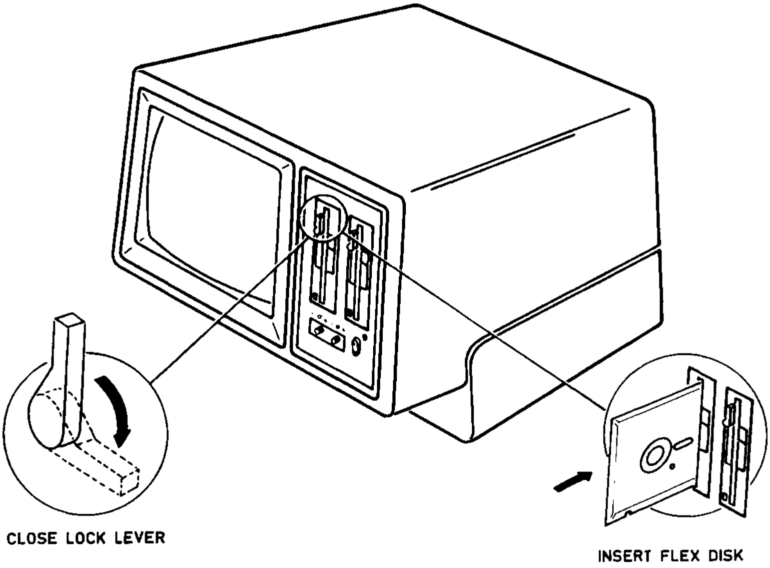


Figure 3.3 Loading a flexible disk

The flexible disk is now loaded and ready for use. At this time the red LED indicator is off: only when the drive mechanism is reading data from or writing data to the disk, is the LED turned on.

To remove a disk, use the following procedure:

- Wait until all processing has finished
- Turn the lock lever counterclockwise
- Remove the disk from the drive and return it to the envelope

CAUTION

Never attempt to remove a flexible disk until all processing is completed and a prompt message such as A > is displayed on the CRT screen.

FIXED WINCHESTER DISK

Generally, systems with a fixed Winchester disk, Figure 3.4, function similarly to systems with two flexible disks. The Winchester disk is a mass storage device with a capacity of ten megabytes. It has no operator controls and requires no hardware considerations by the operator. A small red light emitting diode illuminates when the drive is selected at switch on and remains on until the drive is deselected at switch off. The one flexible disk drive is located in the rightmost position of the cabinet and is designated drive "A." The fixed Winchester drive contains two disks which are designated "B" and "C."

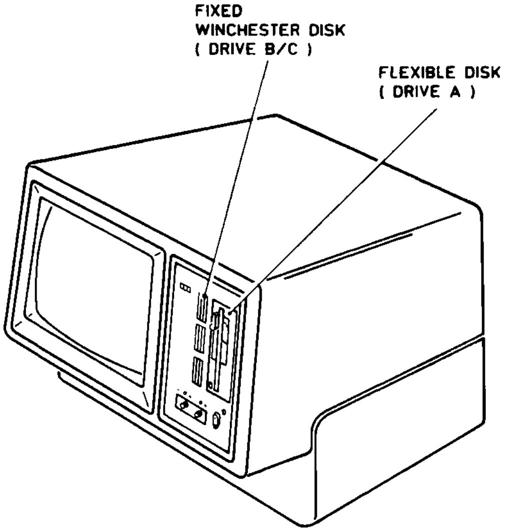


Figure 3.4 Winchester disk drive

CONTROL PANEL

The control panel, shown in Figure 3.5, is below the disk drives and contains a power LED indicator and switches for the power and the CRT screen adjustment.

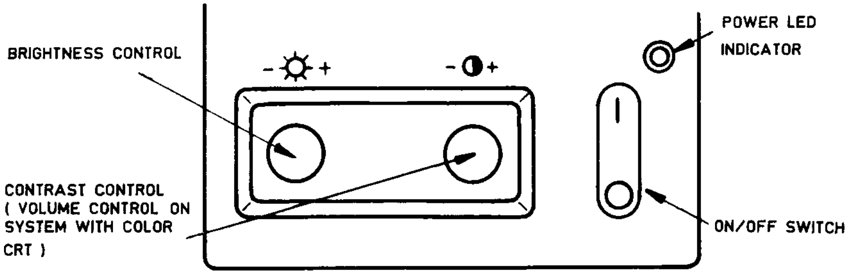

BRIGHTNESS CONTROL
CONTRAST CONTROL (VOLUME CONTROL ON SYSTEM WITH COLOR CRT)
POWER LED INDICATOR
ON/OFF SWITCH
- +
- +

Figure 3.5 Control panel

POWER SWITCH

To start using the computer, set the power switch to 1 (on), this provides power to the computer and turns the green power LED on. Leave the switch in this position until all processing is complete; then set the switch to the off position (0).

BRIGHTNESS AND CONTRAST CONTROLS

These two control knobs allow you to adjust the brigthness and contrast of the CRT screen to give the most comfortable display.

VOLUME CONTROL

The colume of the speaker can be adjusted by turning the control that is located at the rear of the processor, as shown in Figure 3.6, for monochrome systems. For color systems the volume is adjusted by a control on the front panel, see Figure 3.5.

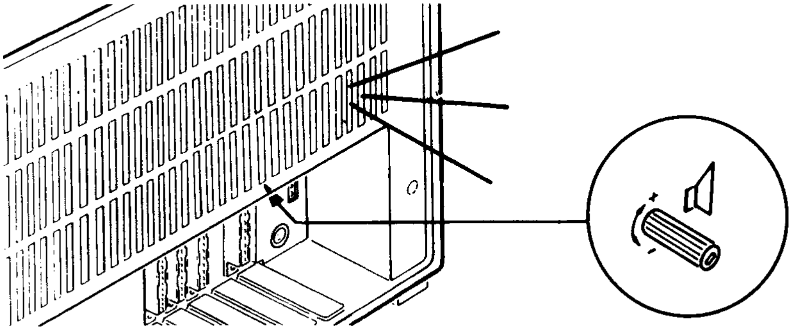


Figure 3.6 Volume control

CRT SCREEN

The CRT (cathode ray tube) screen displays keyboard input, program instructions, system messages, and other information to the operator. It can display a total of 25 lines with up to 80 characters per line. In the graphics mode, the screen has 640 horizontal and 400 vertical points, allowing for the display of both complex figures and a mixture of figures and characters.

The combination of green characters on a dark background and a nonreflective screen provide a clear and restful display. The brigthness and contrast controls on the control panel allow you to adjust the display for individual viewing preference and ambient light.

Systems with a color CRT can display the following colors: black, white, red, green, blue, yellow, magenta, and cyan. The color selection is controlled by the software. Systems with color CRT do not have a contrast control.

KEYBOARD

The keyboard is used to enter instructions and data into the NCR DECISION MATE V. A key roll-over feature enables fast operation: one key can be pressed while another is being released. Also, an 8-character buffer stores entries, allowing them to be made and held until the program is ready for the information. A repeat feature allows for the continuous entry of any character for as long as the key is held down.

If your entry is accepted you hear a "short" tone; if not, you hear a longer one. Simply re-enter the data if not accepted the first time.

The keyboard is divided into two areas: an alphanumeric section with a layout similar to that of a typewriter, and a numeric section. Each section contains a number of special keys. Those keys that are used in a standard manner with all applications are described in the following paragraphs. The other special keys that are program/application dependent are described in detail in the specific software.

ALPHANUMERIC KEYBOARD

The alphanumeric section of the keyboard has a standard typewriter keyboard arrangement and a number of special keys to control the computer. The figures used in this description are for the US English keyboard arrangement. The position and function of the special keys remains the same regardless for which language the keyboard is prepared. Minor changes are made to the alphanumeric keys to ensure the layout is similar to the standard layout for the country in which the computer is being used.

Alphabetic Keys

Whenever the alpha keys shown in Figure 3.7 are pressed, the characters shown on the key tips are entered into the computer and displayed on the CRT screen as small (lowercase) letters. If alpha keys are pressed with the Capital Mode key in the down position, the entered and displayed characters are capital (uppercase) letters.

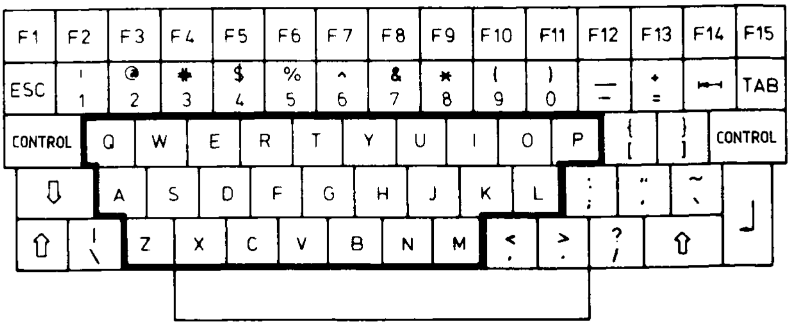

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
ESC ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — - + = ↦ TAB
CONTROL Q W E R T Y U I O P { [}] CONTROL
⇩ A S D F G H J K L : ; " ' ~ ` ↵
⇧ | \ Z X C V B N M < , > . ? / ⇧

Figure 3.7 Alpha keys

Capital Mode Key

When pressed, this key latches down and changes the alpha characters (Figure 3.7) from lowercase to uppercase. To release this key, press the key a second time, see Figure 3.8.

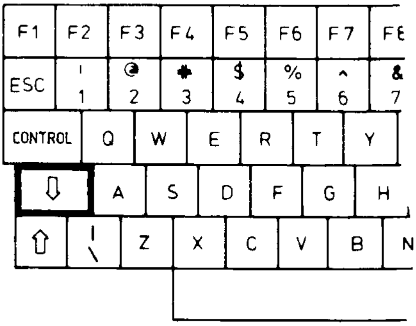


Figure 3.8 Capital mode key

Numeric Keys

Whenever the numeric keys or any other keys shown in Figure 3.9 are pressed, then the characters shown on the lower part of the key tips are entered and displayed. If these keys are pressed together with a shift key, then the characters shown on the upper part of the key tips are entered and displayed.

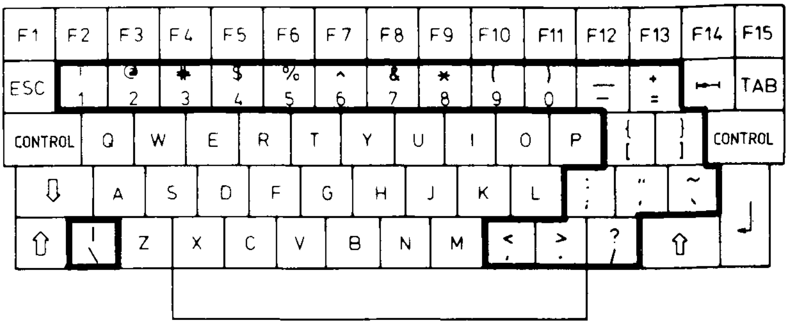

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
ESC ' 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — – + = TAB
CONTROL Q W E R T Y U I O P { [}] CONTROL
A S D F G H J K L : ; " ' ~ `
| \ Z X C V B N M < , > . ? /

Figure 3.9 Numeric and symbol keys

Shift Keys

For convenience two identical Shift keys (Figure 3.10) are on the keyboard. When either key is pressed with a numeric key, the character shown on the upper part of the key tip (see Figure 3.9) is entered and displayed. Similarly, the form of the alpha characters (Figure 3.7) is changed, from lowercase to uppercase (or from uppercase to lowercase if the Capital Mode key is down at the same time). These two keys do not latch down; when you take your finger off the key, they will restore.

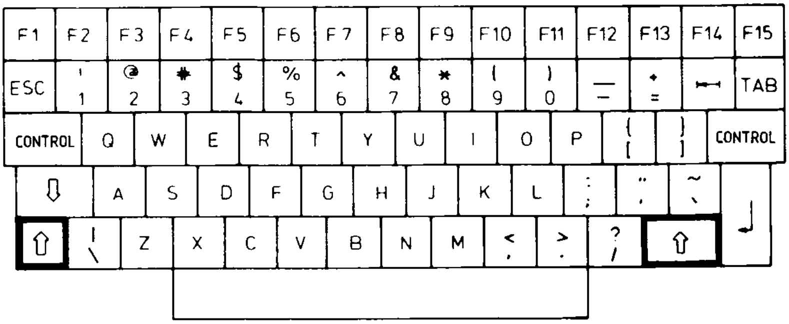

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
ESC ' 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — – + = TAB
CONTROL Q W E R T Y U I O P { [}] CONTROL
A S D F G H J K L : ; " ' ~ `
| \ Z X C V B N M < , > . ? /

Figure 3.10 Shift keys

Backspace Key

Pressing the Backspace key (Figure 3.11) moves the cursor on the CRT screen one position to the left, and normally clears the last character entered.

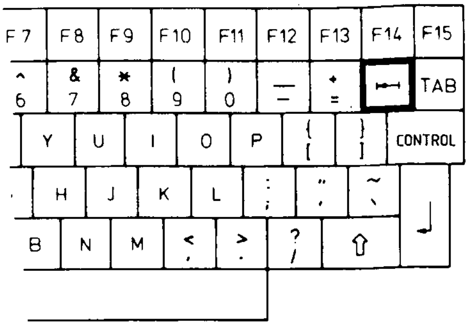

F7 F8 F9 F10 F11 F12 F13 F14 F15
^ 6 & 7 * 8 (9) 0 — – + = ⟼ TAB
Y U I O P { [}] CONTROL
H J K L : ; " ' ~ `
B N M < , > . ? / ⇧

Figure 3.11 Backspace key

Control Keys (CONTROL)

These two keys (Figure 3.12) change the function of some of the alpha keys. This is done to control the operation of the computer. These special functions are software dependent and are described in the software documentation. To use a special function, press a Control key down and hold it down while pressing the desired alpha key. When the control key is released, it restores automatically.

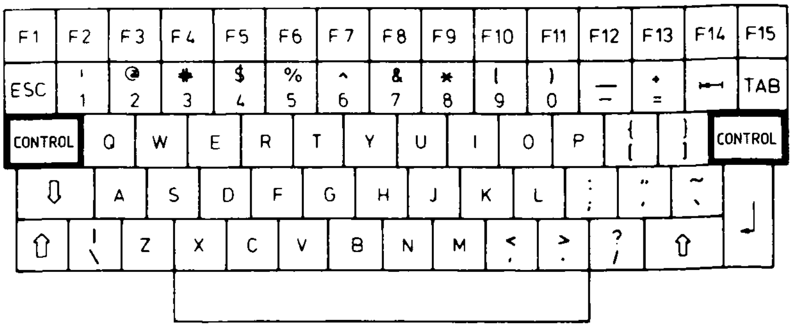

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
ESC ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — – + = ⟼ TAB
CONTROL Q W E R T Y U I O P { [}] CONTROL
⇩ A S D F G H J K L : ; " ' ~ `
⇧ | \ Z X C V B N M < , > . ? / ⇧

Figure 3.12 Control keys

Return Key ↵

The most used of the special keys, the Return key (Figure 3.13) appears on both the alphanumeric and the numeric sections of the keyboard. This key indicates to the computer that the entry is complete.

NOTE: In other publications you may see this key called "Carriage Return", "Field Terminate" or "Enter" key.

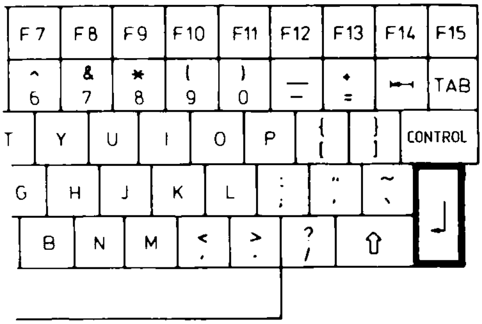

F7 F8 F9 F10 F11 F12 F13 F14 F15
^ 6 & 7 * 8 (9) 0 — – + = TAB
T Y U I O P { [}] CONTROL
G H J K L : ; " ' ~ `
B N M < , > . ? / ⇧

Figure 3.13 Return key

Programmable Keys

The remaining keys (Figure 3.14) in the alphanumeric section of the keyboard, ESC, TAB, and F1 through F15 are dependent on the program. The design of the keyboard allows for a descriptive mask above the function keys F1 through F15. This mask can contain short a description of the function of each key. The use of these keys is application dependent and is described in the application documentation.

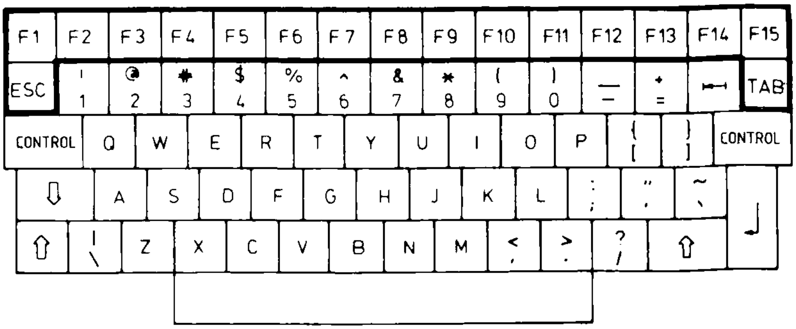

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
ESC ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — – + = TAB
CONTROL Q W E R T Y U I O P { [}] CONTROL
⇩ A S D F G H J K L : ; " ' ~ `
⇧ | \ Z X C V B N M < , > . ? / ⇧

Figure 3.14 Programmable keys

NUMERIC KEYBOARD

The numeric keyboard has a 10-key numeric keypad allowing the high-speed entry of numeric data. In addition to the 10 numeric keys, this keyboard section includes the following keys:

- Five programmable keys (F16 through F20)
- Five cursor positioning keys
- Arithmetic keys for add, subtract, multiply, divide, and clear
- A double zero key 00
- A decimal point key.
- A return key

Programmable Function Keys

These five (F16-F20) keys (Figure 3.15) operate exactly the same as the keys F1-F15 on the alphanumeric section of the keyboard. Similarly, provision is made for a descriptive mask to be placed above these keys.

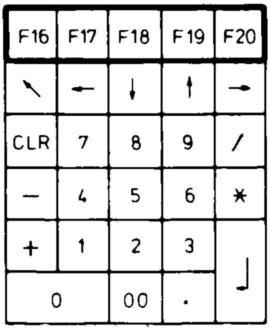


Figure 3.15 Programmable keys

Cursor Positioning Keys

Pressing any of these keys (Figure 3.16) moves the cursor on the CRT screen in the direction of the arrow on the respective key tips. A more detailed description is found in the application documentation.

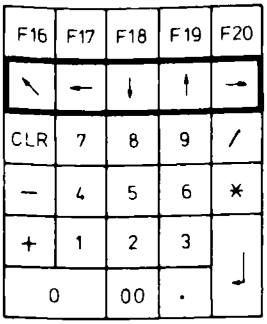


Figure 3.16 Cursor positioning keys

Arithmetic Keys

These keys (Figure 3.17) are used to perform the basic arithmetic functions of add, subtract, multiply, divide and clear, similar to those on your pocket calculator

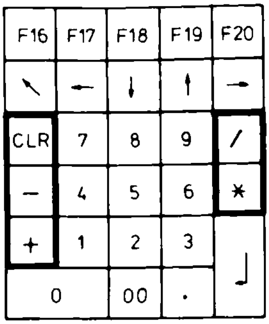


Figure 3.17 Arithmetic keys

Double Zero Key

Pressing this key (Figure 3.18) inputs two zeros to the computer, allowing for the quick entry of multiple zeros.

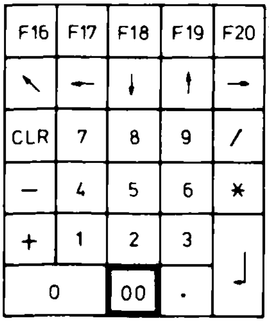


Figure 3.18 Double zero key

NOTE: Pressing the Zero and the Double Zero keys at the same time inputs three zeros.

Decimal Point Key

Use this key (Figure 3.19) to enter the position of the decimal point in a string of numbers. Should you require the computer to display (and print) a comma (,) instead of a decimal point (.) simply press a CONTROL key together with a decimal point key. The computer now displays (and prints) a comma whenever the decimal point key is used. Press the CONTROL key together with the decimal point key again and the display (and print) reverts to a decimal point. After a power down the display (and print) reverts to a decimal point.

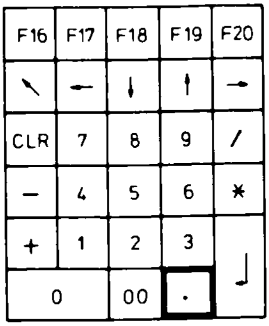


Figure 3.19 Decimal point key

Return Key ↵

Like the Return key on the alphanumeric section, this key (Figure 3.20) is pressed to indicate to the computer that the current entry is complete.

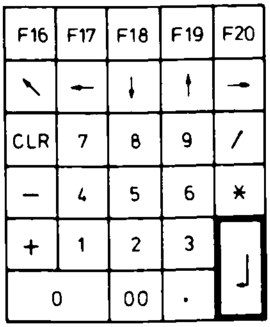


Figure 3.20 Return key

STARTUP PROCEDURE

You have now learned enough to switch on the system, insert a flexible disk, and respond through the keyboard to messages displayed on the CRT screen. To summarize ...

Push the on/off switch to the on (1) position and wait a few moments, the following message appears on the CRT:

```
DISK A: NOT READY <CR>
```

Select the correct operating system flexible disk
Insert the disk into drive "A" with the recording slot end first and the label side facing the CRT screen; then close the disk lock lever
Press the Return key (↵)
When the software heading appears in the top left hand area of the CRT screen, the computer is ready for use

NOTE: On systems with two flexible disk drives, the leftmost drive is designated drive "A." On systems with a fixed Winchester disk, the flexible disk drive which is located in the rightmost position is designated drive "A."

It is recommended to remove the flexible disks only when a system prompt message is displayed and before switching the computer off.

SOME PRACTICE EXERCISES

To become familiar with the keyboard, type any string of characters — your name, address, or whatever information you want.

```
A>enter any information and as much as you want
```

When finished, complete the entry by pressing the Return (↵) key. During this practice, you can't harm the hardware or the software — the computer simply won't know "what you're talking about" and will tell you so by displaying your first word and a question mark.

```
ENTER?
A>
```

Now, enter more data, practicing using the keys described in this section. Shift to uppercase (capital) letters:

```
A> ENTER DATA IN CAPITAL LETTERS
ENTER?
A> and then shift back to small ones
AND?
A>
```

Practice the effect of using the Capital Mode key, the Shift key, and the Capital Mode and Shift keys simultaneously, together with all other keys on the keyboard.

If you make mistakes, correct them with the Backspace key, which erases a single character or all the characters on the line. Soon you will be comfortable with the keyboard and ready for actual ("live") work. However, before proceeding take time to study the manual that comes with your operating system software. Learn how to format disks, how to make back-up copies of important disks, and how to prepare the system for your applications. Learn how to make the computer work for you.

WHAT IF?

PROBLEM ISOLATION

In this section guidelines are given on how to recover when problems arise. Never assume that anything has been done, or that anything is correct. A few minutes spent checking and rechecking often saves time that would be wasted while you wait for assistance.

If you have to contact your service engineer, ensure that you can accurately describe the problem. This assists the engineer in deciding what spare parts and service equipment he may require. Keep a written record of any problem or error messages, ensure that the record includes the following:

- Date and time of problem.
- The application/software disk being used.
- Exact message that was displayed on the CRT screen.
- Which level zero diagnostic LED indicator was on.
- If the diagnostic module was used, a record of the displays from this module.

SERVICING ARRANGEMENTS

Should you need the help of a skilled engineer, NCR has a large and highly trained team of field engineers operating from approximately 1200 offices throughout the world. You can select one of several servicing arrangements with NCR: your nearest NCR office is able to provide more detailed information.

- Full NCR service contract — For an annual fee, NCR trained engineers are available, whenever required, during normal business hours. The fee usually covers travelling time, repair time, and the cost of replacement parts. This plan offers the advantage of total servicing with a convenient, once-yearly fee.
- Time-and-material service — With this arrangement, you pay for the NCR field engineer's time and the cost of any re-

placement parts, each time he comes to your site.

- NCR depot service — Under this plan you bring your computer to your local NCR service office. When repairs are complete, NCR either returns the unit to you or advises you that it is ready for collection. You are charged on a time-and-material basis.

Other choices of servicing arrangements may be available:

- If you purchased your NCR DECISION MATE V from a supplier other than NCR, the supplier may have established his own servicing arrangements.
- Suitable staff within your own organization may, with training and the assistance of the service manual, be able to repair your NCR DECISION MATE V.

CAUTION

It is recommended that only trained persons with experience on servicing electronic equipment and handling printed circuit boards and integrated circuits should attempt to service the computer. Should you plan to train an engineer from your organization, contact the local NCR office for information on training courses.

RADIO FREQUENCY INTERFERENCE

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna
- Relocate the computer with respect to the receiver
- Move the computer away from the receiver
- Plug the computer into a different outlet so that computer and receiver are on different branch circuits

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:

"How to Identify and Resolve Radio-TV Interference Problems."

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402. Stock No. 004-000-00345-4.

U.S. CUSTOMER SUPPORT

For users in the U.S., support for both hardware and software problems is available on the following toll-free telephone numbers:

- 1 800 543 9935 (outside Ohio)
- 1 800 762 9275 (inside Ohio)

Problem	Reason	Remedy
Power LED not on	On/Off switch in off position	Press switch to on position
	Power cable not connected to processor	Connect cable to processor
	Power cable not connected to wall outlet	Connect cable to wall outlet
	No power at wall outlet	Inform site electrician
	Faulty power cable	Replace cable
	Internal processor problem	Inform service engineer (1)
Software heading message does not appear on startup	Contrast/Brightness controls not correctly adjusted	Adjust Contrast/Brightness controls
	Operating system disk not loaded the correct way	Reload disk into drive A correctly (refer Hardware Operation)
	Wrong type of disk	Select correct disk and load into drive A
	Disk loaded into drive (dual flox disk system)	Remove disk and load into drive A
	Operating System disk damage	Use backup copy of Operating System disk
	Internal problem in processor	Inform service engineer (1)
Unable to enter data through the keyboard correctly	Capital Mode Key not positioned correctly	Set Capital Mode key correctly
	Keyboard buffer full	Wait until current processing operation is completed
	Keyboard cable disconnected from the processor	Reconnect cable
	Internal keyboard or processor problem	Inform service engineer (1)

Problem	Reason	Remedy	
Unable to write to flexible disk	Disk not in the specified disk drive	Install correct disk in specified drive	
	Flexible disk incorrectly loaded	Reload disk correctly (refer to Hardware Operation)	
	Write protect tab fitted to flexible disk	Check why write protect tab is fitted. Only remove when you are certain it was wrongly fitted, or select correct flexible disk	
	Damaged flexible disk	Use backup copy of disk	
	Disk not formatted	Carry out format routine	
	Internal processor or flexible disk drive problem	Inform service engineer	(1)
Unable to read from flexible disk	Disk not installed in specified drive	Install disk into correct drive	
	Disk not installed correctly	Install disk correctly (refer to Hardware Operation)	
	Damaged flexible disk	Use backup copy of disk	
	Internal processor or flexible disk drive problem	Inform service engineer	(1)
Undefined problem – unable to continue any processing	Problem with operating system, and/or application program	Reload operating system and application program	
		Switch off computer, use backup copies of operating system and program and reload	
	Temporary problem with electrical supply	Inform site electrician	
	Internal processor problem	Inform service engineer	(1)

(1) Before contacting your service engineer, check that you can give him full details of the problem, including any messages that are displayed and any diagnostic LEDs that are on. If you have a diagnostic module, refer to the Service Manual and perform the appropriate diagnostic routines. Again provide the engineer with the information from the diagnostics.

HELPFUL HINTS

This section suggests some of the things that may help in the day-to-day operation of your computer. Hints are given on the working environment, on operator discipline when using the computer, and on what to do when finishing work. Problems are often caused by mishandling flexible disks, therefore, a separate section is devoted to the care of flexible disks. Should you decide to move your computer, hints on moving NCR DECISION MATE V are provided at the end of this section.

POSITIONING THE COMPUTER

When choosing a position for your NCR DECISION MATE V, consider the following points:

- Choose a position away from heavy office traffic and dust.
- Site the computer where there are no extremes in temperature and humidity: try not to position it in direct sunlight or too close to a heating system.
- Discourage smoking or drinking beverages near the computer.
- Be sure there are sufficient electrical outlets for the computer and any peripheral units.
- Be sure there is sufficient storage space for flexible disks, working media, and finished work.
- Arrange power cables and any interconnecting cables so that people are not likely to trip over them.
- Operating the computer near devices that produce strong magnetic fields may cause some instability of the CRT screen display: should this be a problem then it is suggested that you reposition your computer away from the source of the magnetic fields.

WORKING WITH NCR DECISION MATE V

The following hints may help in the day-to-day operation of your computer:

- Work in a logical and orderly manner. Be sure that you have plenty of space for the working media, and that work done is kept separate from work to be done.
- Never switch off the computer before a system prompt message is displayed on the CRT screen, otherwise data may be lost.
- Do not leave flexible disks lying about, as soon as you have finished with a disk return it to its storage place.
- Make backup copies of your disks at regular intervals. The backup copies should be stored in a separate location, to reduce the chances of losing your media.
- Before leaving the office, remove the power cable from the wall outlet, and cover the computer with a dust cover.

CARE OF FLEXIBLE DISKS

If the flexible disk is handled correctly, it can be used for a long period of time. In particular, pay attention to the following points:

- Use a felt tip pen to write on the flexible disk label. Deposits from lead pencils, erasers, grease pencils, or ball point pens can damage the recording surface. Write on the label before putting it on the flexible disk, or, if the label is already on the disk, write on the label only when the flexible disk is in the envelope.
- Do not put the flexible disk in direct sunlight.
- Do not put a label on the seamed side of the jacket.
- Do not put paper clips or rubber bands on the jacket.
- Do not touch the recording surface.
- Do not, by any method, clean the flexible disk. The inner surface of the jacket cleans the flexible disk during processing.
- To prevent the possibility of touching the recording surface hold the disk only at the corner of the jacket.
- Return the disk to its envelope as soon as you remove it from the disk drive.
- Do not place flexible disks on or near any magnetic object.

MOVING THE COMPUTER

PREPARATION

To move NCR DECISION MATE V either a short or a longer distance, first prepare the unit as follows:

- Set the on/off switch to the off position.
- Disconnect all cables and protect the cable connectors with suitable wrapping material.
- Coil the cables and secure with masking tape.
- Insert a protective card into each of the flexible disk drives.
- Carefully pack each piece of the system.

If the original packing material is no longer available, use strong boxes that are large enough to allow for plenty of cushioning material such as foam pads.

RE-INSTALLATION

To re-install your NCR DECISION MATE V, follow the setup procedure in *Installation.*

CLEANING PROCEDURE

No special cleaning procedure is necessary for the NCR DECISION MATE V however, regular dusting with a soft lint-free cloth will prevent the build up of dirt. Pay particular attention to the CRT screen. Any persistent marks may be removed with a soft cloth dampened with a mild soap solution.

Aerosol sprays and cleaning solvents should not be used. Be sure to switch the power off before starting to clean the computer.

Dear NCR MS-DOS User:

This binder contains the MS-DOS master diskette and documentation on how to use the software: the User's Guide provides simple descriptions of the features and functions of MS-DOS, while the Programmer's Manual provides technical information.

For your processing enjoyment, we have also included another diskette in the binder. This diskette contains application and demonstration software that is both informative and entertaining. You can, for example, create line, bar, and pie charts; you can play music or a game of chase.

Brief explanations for this easy-to-use software are in a special supplement following the User's Guide.

We hope you enjoy this processing "extra" and especially wish you success in using MS-DOS on your NCR DECISION MATE V.

Best regards,
NCR Corporation

NCR

CUSTOMER PROGRAM LICENSE AGREEMENT

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE OPENING THIS DISKETTE(S) PACKAGE. OPENING THIS DISKETTE(S) PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED; AND YOUR MONEY WILL BE REFUNDED.

NCR provides this Program(s) and licenses its use under these terms and conditions and under Copyright Law: You assume responsibility for the selection of the Program(s) to achieve your intended results, and for the installation, use and results obtained from the Program(s). This program is confidential, proprietary to and a trade secret of the owner, and should be safeguarded by you as such.

LICENSE

You may:

a. use the Program(s) only on a single machine at a single location;

b. copy the program into any machine readable or printed form for backup or modification purposes only, to support your use of the Program(s) on the single machine (Certain programs, however, may include mechanisms to limit or inhibit copying. They are marked "copy protected.");

c. modify the Program(s) and/or merge it into another program for your use on the single machine (Any portion of this Program(s) merged into another program will continue to be subject to the terms and conditions of this Agreement.); and

d. transfer the Program(s) and license to another party only if the other party agrees to accept the terms and conditions of this Agreement. You must advise NCR of the name and address of the other party and the other party must sign a copy of the NCR Customer Program License Agreement and have the same received by NCR. If you transfer the Program(s), you must at the same time either transfer all copies whether in printed or machine readable form to the same party or destroy any copies not transferred; this includes all modifications and portions of the Program(s) contained or merged into other programs.

You must reproduce and include any copyright notice and serial number on any copy, modification or portion merged into another program.

TERM

The license is effective until terminated. You may terminate it at any time by destroying the program together with all copies, modifications and merged portions in any form. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the Program(s) together with all copies, modifications and merged portions in any form.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE PROGRAM(S), OR ANY COPY, MODIFICATION OR MERGED PORTION, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE.

IF YOU TRANSFER POSSESSION OF ANY COPY, MODIFICATION OR MERGED PORTION OF THE PROGRAM TO ANOTHER PARTY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

EXCLUSION OF WARRANTY

THE PROGRAM(S) IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM(S) PROVE DEFECTIVE, YOU (AND NOT NCR OR ITS DEALER OR DISTRIBUTOR) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. NCR does not warrant that the functions contained in the Program(s) will meet your requirements or that the operation of the program will be uninterrupted or error free.

LIMITED WARRANTY

NCR warrants the diskette(s) on which the program is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

NCR's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette(s) not meeting NCR's "Limited Warranty" and which is returned to NCR or an authorized NCR dealer or distributor, with a copy of your receipt, or
2. if NCR or its authorized dealer or distributor is unable to deliver a replacement diskette(s) and repair is not practicable or cannot be timely made, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT WILL NCR BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DISKETTE(S) EVEN IF NCR OR AN AUTHORIZED NCR DEALER OR DISTRIBUTOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

Some states do not allow limitations on how long an implied warranty lasts, so the above exclusion may not apply to you.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

NCR

NCR DECISION MATE V

MS™- DOS User's Guide

MS and MULTIPLAN are trademarks of Microsoft Corporation. CP/M is a registered trademark of Digital Research.

Second Edition, October 1983

It is the policy of NCR Corporation to improve products as new technology, components, software, and firmware become available. NCR Corporation, therefore, reserves the right to change specifications without prior notice.

All features, functions, and operations described herein may not be marketed by NCR in all parts of the world. In some instances, photographs are of equipment prototypes. Therefore, before using this document, consult your nearest dealer or NCR office for information that is applicable and current.

MS-DOS USER'S GUIDE

CONTENTS

1. INTRODUCTION

2. GETTING STARTED

3. MORE ABOUT FILES

4. LEARNING ABOUT COMMANDS

5. MS-DOS COMMANDS

6. MS-DOS EDITING AND FUNCTION KEYS

7. LINE EDITOR (EDLIN)

8. FILE COMPARE (FC) UTILITY

9. LINKER PROGRAM (MS-LINK)

C. HOW TO OBTAIN AND INSTALL SOFTWARE

D. ADVANCED CONFIGURATION FEATURE

INTRODUCTION

MS^{TM}-DOS is a disk operating system for the 16-bit processor of NCR DECISION MATE V. Through MS-DOS, you communicate with the computer, disk drives, and printer (if available), managing these resources to your advantage.

WHAT IS AN OPERATING SYSTEM?

An operating system is your "silent partner" when you are using the computer. It provides the interface between the hardware and both you and the other software (application packages and your own programs). An operating system can be compared to the electricity in a house: You need it for the toaster and the blender to work, but you are not always aware that it's there.

Operating systems provide varying capabilities. With MS-DOS, you can create and keep track of files, run and link programs, and access peripheral devices (for example, printers and disk drives) that are attached to your computer.

HOW TO USE THIS MANUAL

This manual describes MS-DOS and how to use it. This chapter introduces some basic MS-DOS concepts; Chapter 2 discusses how to start using MS-DOS and how to format and back up your disks.

Chapter 3 tells you about files - - what they are and how to use them. Chapters 4 through 6 introduce MS-DOS commands and Chapter 7 describes the line editor, EDLIN. Read these chapters carefully - - they contain information on protecting your data, system commands, and the MS-DOS editing commands.

Chapter 8 explains how to use the MS-DOS File Comparison utility, FC. This utility is helpful when you need to compare the contents of two source or binary files.

If you are writing programs and want to link separately-produced object modules and create relocatable modules, Chapter 9 describes a useful MS-DOS utility, MS-LINK.

Appendices to this manual include instructions if you are using MS-DOS and other operating systems on NCR DECISION MATE V, disk error messages, and special guidelines on how to run MS-DOS-compatible applications on your computer.

If you want to know more, a companion manual, the *PROGRAMMER'S MANUAL*, contains information on the technical aspects of MS-DOS. It also describes MS-DOS system architecture, additional utilities, and system calls and interrupts.

SYNTAX NOTATION

The following syntax notation is used throughout this manual in descriptions of command and statement syntax. Don't be overwhelmed by this list; after you use the commands a few times, the notation becomes quickly familiar.

[ ] square brackets
Indicate that the enclosed entry is optional.

< > angle brackets
Indicate that you supply the text for this entry. When the angle brackets enclose lowercase text, type in an entry defined by the text; for example, <filename>.

braces
Indicate that you have a choice between two or more entries. At least one of the entries enclosed in braces must be chosen unless the entries are also enclosed in square brackets.

. . . ellipses
Indicate that an entry may be repeated as many times as needed or desired.

| a bar
When used with an MS-DOS filter, the bar indicates a pipe. (This feature is fully explained in Chapter 4, Learning About Commands.)

CAPS capital letters
Indicate portions of statements or commands that must be entered exactly as shown. Capital letters also indicate specific keys, such as <CR>.

All other punctuation, such as commas, colons, slash marks, and equal signs must be entered exactly as shown.

FLEXIBLE/FIXED DISK SYSTEMS CONSIDERATIONS

To simplify explanations in this manual, examples are shown based on a multi-drive, flexible disk system. However, if you have a flexible/fixed disk system, you will use only the flexible disk drive to format and make copies of flexible disks. In these situations, MS-DOS always "prompts" you to change disks and waits for you to insert the new disk. You then continue processing by pressing any key.

Turn to the next chapter and learn how to start your MS-DOS system and how to format and back up your disks.

GETTING STARTED

SYSTEM SETUP

The MS-DOS *master disk* (or diskette), the one you received with this book, contains all the operating system software files and all commands. In this chapter, you learn how to install the software, switch processing from one disk to another, protect your master disk, and format other new disks. Finally, you'll read about files.

Before actually loading your software, you may need to know a little more about NCR DECISION MATE V and those all-important disks. Depending on your computer model, you have either a flexible disk system or a flexible/fixed disk system. The types of disks are not important to MS-DOS; the software only wants to know where to get and put information.

Regardless of which disk system you have, you always start processing from flexible disk drive A. Let's do that now by loading MS-DOS into memory.

LOADING MS-DOS

Turn on your computer, insert the MS-DOS disk in drive A, and press the ↵ key. (This return key is also referred to as the <CR> key.) This is always the standard startup procedure. Depending on the size of memory, loading MS-DOS can take up to 25 seconds.

Once MS-DOS is loaded, the system searches the MS-DOS disk for the COMMAND.COM file and loads it into memory. The COMMAND.COM file is a program that processes the commands you enter and then runs the appropriate programs. It is also called the *command processor.*

When the command processor is loaded, you see a copyright and software identification message on your screen. Have you read the "sign on" message, too? If so, you're right where you should be in this guide to begin using MS-DOS.

COPYING YOUR MASTER SOFTWARE DISK

You start using MS-DOS by making a backup copy of your master software disk. Actually, the software on your MS-DOS master disk begins the process automatically. You have only to help by answering simple questions or following simple directions that MS-DOS displays.

While the displays are self-explanatory, you may prefer following along with printed text. A summary of the copy procedure is shown in table form with your actions marked with a ✓ .

The first question asks how many flexible disk drives you have. Answer the question and then . . .

- Turn to page 2-3 if you have 2 flexible disk drives
- Turn to page 2-5 if you have 1 flexible disk drive

Copying the Master Diskette — Sequence Summary

2 FLEXIBLE DISK DRIVES

Formatting begins . . .

```
A> FORMAT B:

Insert new diskette for drive B:
and strike any key when ready
```

✓ **Press any key!**

```
xxxxxx bytes total disk space
xxxxxx bytes available on disk

Format another (Y/N)?  N
```

✓ **Don't format any other disks now.**
Press N (for no).

Formatting complete

We're now ready to copy the master diskette in drive A to the diskette in drive B. Press any key when instructed to do so.

```
A> DISKCOPY A: B:/V

Insert source diskette into drive A:
Insert formatted target diskette into drive B:
Press any key when ready
```

✓ **Your disks are already in place.**
Press any key!

```
Copying . . .

Copying . . . Copying complete
```

cont.

```
Copy another (Y/N)? N
```

✓ **Don't copy any other disks now. Press N (for no).**

Remove the master diskette from drive A and save for system protection. Take the diskette from drive B and place it in drive A.

Copying the Master Diskette — Sequence Summary

1 FLEXIBLE DISK DRIVE

Formatting begins . . .

```
A> FORMAT A:
```

```
Insert new diskette for drive A:
and strike any key when ready
```

✓ **Remove the master diskette and insert a new diskette; press any key when ready.**

```
xxxxxx bytes total disk space
xxxxxx bytes available on disk
```

```
Format another (Y/N)? N
```

✓ **Don't format any other disks now. Press N (for no).**

```
Formatting complete
```

✓ Remove the formatted diskette from drive A and again insert the master diskette into drive A.

During the following copy procedure, change diskettes as instructed until the 'copy complete' message appears.

```
A> DISKCOPY/V
```

```
Insert formatted target diskette into drive A:
Press any key when ready
```

✓ **Change diskettes; press any key when ready**

cont.

```
Copy complete

Copy another (Y/N)?  N
```

✓ **Don't copy any other disks now. Press N (for no).**

LEARNING ABOUT YOUR DRIVE DESIGNATIONS

When you copied your master disk, you were directed to insert or change a disk. You were instructed to do this by *drive designation.* For example, "Insert a disk into drive A."

The drive designation, which is always an alphabetic character, tells MS-DOS where to get and put information. No matter what types of disk units you have, each drive always has its own designation. Consider the following disk configuration examples:

- You have two flexible disk drives. One drive is designated (and labelled) A; the other drive is designated (and labelled) B.
- You have one flexible disk drive and one fixed disk unit. The flexible disk drive is designated (and labelled) A, but what about the fixed disk? *A fixed disk unit contains two logical disk drives.* (You can't see them, and they aren't labelled, but they're there). In this example configuration, the logical disk drives of the fixed disk unit are B and C.
- Now, assume you have two flexible disk drives and three fixed disk units. What are the drive designations? The flexible disk drive designations are A and B; the fixed disk unit drive designations are C, D, E, F, G, and H.

Drive designations are assigned by MS-DOS, which assumes you have 2 flexible disk drives. If you don't have this configuration you must describe your configuration to MS-DOS.

DEFINING YOUR FIXED DISKS

If you have fixed disks (sometimes called hard disks), you must continue setup procedures by defining your disk configuration. (If you have only a flexible disk system, you can skip this section.)

You define your disk configuration with the CONFIG routine. With self-explanatory screens, this routine is easy to use: You simply enter the name of the routine and follow the displays, carrying on a conversation with MS-DOS.

To make the procedure even simpler, however, table 1 summarizes the entire configure procedure. The left column contains the complete conversation. **What you say (enter)** is in bold type, what MS-DOS responds is in normal type, and *actions you must perform* are in italics. If you need some guidance when performing the sequence, look at the right-hand column for help.

Display/**Enter**/*Action*	Comments
A>	You're using a flexible/fixed disk system, but MS-DOS still "thinks" you have only flexible disks. Let's tell the software about your fixed disk with the CONFIG utility. Enter CONFIG and press ↵ .
A> **CONFIG** ↵	
- - - - - - - - - - -	
CONFIG UTILITY	Study this main menu screen for a minute. These functions can all be performed with CONFIG. (You'll learn about them in the "MS-DOS Commands" chapter.)
1) Modify Function Keys 2) Select Printer (Serial/Parallel) 3) Modify Retry/Restore Counter 4) Modify Serial Printer Interface	
5) Modify Disk Configuration	This function is the one you want.
6) Exit Program	
* Enter Function **5**	Enter 5.
- - - - - - - - - - -	
CONFIG UTILITY	Another screen! You'll be using this one several times. Let's call it the *disk configuration* screen. First, modify the number of flexible disks.
Modify Disk Configuration	
1) Modify number of flexible disks	This function is the one you want.
2) Modify number of hard disks 3) Display configuration 4) Return to main menu	
* Enter Function **1**	Enter 1.

Table 1 Defining Your Disk Configuration

cont.

Display/Enter/*Action*	Comments
CONFIG UTILITY	
Modify Disk Configuration	
Modify Number of Flexible Disks	Specify the number of flexible disks you have.
1) One Flex Disk	This function is the one you want.
2) Two Flex Disks	
3) Return to Main Program	
* Enter Function 1	Enter 1.
— — — — — — — — — — —	
CONFIG UTILITY	Now, modify the number of fixed disks.
Modify Disk Configuration	
1) Modify number of flexible disks	
2) Modify number of hard disks	This function is the one you want.
3) Display configuration	
4) Return to main menu	
* Enter Function 2	Enter 2.

cont.

Display/Enter/*Action*	Comments
CONFIG UTILITY Modify Disk Configuration Modify Number of Hard Disks	Here you tell the software about your fixed disk.
1) No hard disk 2) One hard disk 3) Two hard disks 4) Three hard disks 5) Return to Main Program	This function is the one you want, assuming you have one fixed disk unit.
* Enter Function 2	Enter 2.
– – – – – – – – – – –	
CONFIG UTILITY Modify Disk Configuration	The disk configuration screen now appears. Select function 4. (If you want to see the the drive assignments, select function 3. Press ↵ to return to this screen.)
1) Modify number of flexible disks 2) Modify number of hard disks 3) Display configuration 4) Return to main menu	
* Enter Function 4	Enter 4.

cont.

Display/Enter/*Action*	Comments
CONFIG UTILITY	You've finished configuring.
1) Modify Function Keys 2) Select Printer (Serial/Parallel) 3) Modify Retry/Restore Counter 4) Modify Serial Printer Interface 5) Modify Disk Configuration	
6) Exit Program	This function is the one you want.
* Enter Function **6**	Enter 6.
- - - - - - - - - - -	
1 Update O.S. disk in drive A 2 Return to main program 3 Exit CONFIG	The exit program function is important. Here, you make the changes permanent by having them written to disk.
ATTENTION: Changes to the disk configuration must be written to disk (permanent) and must be followed by a restart. Update the disk, exit CONFIG, and then turn off and on the computer when the system prompt appears.	Read the "attention" and then perform the following sequence:
* Enter Function **1**	Enter 1 (You'll "hear" the changes being written to disk.)
* Enter Function **3**	Enter 3 (You're leaving the CONFIG utility.)
A>	
Switch your computer off and on. Complete setup procedures as described in the following text.	Be sure to do this!

NOTE: The sequence described assumes one flexible disk drive and one fixed disk unit. Adjust your entries for your specific configuration.

FORMATTING DISKS

You've already done a lot with your DECISION MATE V and MS-DOS. You've protected your software by making a copy of the master disk and, if you have a fixed disk, defined the configuration. You also saw how to format a disk. Remember, though that *each new disk must be formatted.*

Your fixed disks, for example, are not yet formatted for MS-DOS. Because the FORMAT routine is described in detail in chapter 5, its description is not repeated here; however, a couple of comments about formatting fixed disks must be noted.

- To format one logical drive of a fixed disk unit takes approximately 20 minutes. This time is calculated based on the "standard" number of 5 certifications (read and write checks). The formula is 4 mins + (3 min. x # of certifications). You have an option of increasing or decreasing the number of certifications before formatting begins.
- Before you format a fixed disk, always check that the logical disk drive hasn't already been formatted by MS-DOS or some other operating system. Use a command like MS-DOS CHKDSK to first determine the contents of the disk.

DEFINING A SERIAL PRINTER

Are you using a printer? MS-DOS assumes it is a parallel printer. If using a serial printer, you must define it to MS-DOS with the CONFIG utility. Because printer requirements vary, refer again to chapter 5 for a full description of CONFIG.

You've now completed all setup procedures. In the next sections of this chapter, you learn some more operating procedures and about files.

FREQUENTLY PERFORMED OPERATIONS

ENTERING THE DATE AND TIME

When you load MS-DOS into memory or restart your computer, you see the date and time prompts. You should enter this information which is extremely helpful in keeping track of when you created or updated data on your disk.

When you see:

```
Enter new date: _
```

Type today's date in an mm-dd-yy format, where:

- mm is a 1- or 2-digit number from 1-12 (representing month)
- dd is a 1- or 2-digit number from 1-31 (representing the day of the month)
- yy is a 2-digit number from 80-99 (the 19 is assumed), or a 4-digit number from 1980-2099 (representing year)

Any date is acceptable in answer to the new date prompt as long as it follows the above format. Separators between the numbers can be hyphens (-) or slashes (/). For example:

5-1-83 or 05/01/83

are both acceptable answers to the "Enter new date:" prompt.

If you enter an invalid date or form of date, the system prompts you again with "Enter new date:".

After you respond to the new date prompt and enter your answer by pressing the <CR> key, you see a prompt similar to this:

```
Current time is 0.00:00.00
Enter new time: _
```

Enter the current time in the hh:mm format, where:

- hh is a 1- or 2-digit number from 0-23 (representing hours)
- mm is a 1- or 2-digit number from 0-59 (representing minutes)

MS-DOS uses this time value to keep track of when you last updated and/or created files on the system. Notice that MS-DOS uses military time; for instance, 1:30 p.m is written 13:30.

Example:

```
Current time is 0:00:00.00
Enter new time: 9:05
```

Only use the colon (:) to separate hours and minutes. If you enter an invalid number separator, MS-DOS repeats the prompt.

NOTE: If you make a mistake while typing, press the CONTROL key on your keyboard, hold it down, and then press the C key. The <CONTROL-C> function aborts your current entry. You can then re-answer the prompt or type another command. To correct a line before you press <CR>, use the <BACKSPACE> key to erase one letter at a time.

CHANGING THE DEFAULT DRIVE

The A> is the MS-DOS prompt from the command processor. It tells you that MS-DOS is ready to accept commands.

The A in the previous prompt is the *default* disk drive. This means that MS-DOS searches only the disk in drive A for any filenames you enter and writes files to that disk unless you specify a different drive. You can ask MS-DOS to search a disk in another drive by changing the drive designation or by specifying it in a command. To change the disk drive designation, enter the new drive letter followed by a colon. For example:

```
A>
A>B: <CR>   (you have typed B: in response to the prompt)
B>
```

The system prompt B> appears and drive B is now the default drive. MS-DOS searches only the disk in drive B until you specify a different default drive. To move back to drive A, simply specify A:. (Don't forget the colon.)

```
A>B:
B>A: <CR>
A>
```

BACKING-UP YOUR DISKS

You've made a backup copy of your master software disk; you should make backup copies of all your disks. If a disk becomes damaged or if files are accidentally erased, you will still have all of the information on your backup disk.

You make backup copies of flexible disks with the DISKCOPY command; you make backup copies of fixed disks with the

BACKUP command. (Both of these commands are discussed in detail in Chapter 5, MS-DOS Commands.)

USING THE PROGRAMMABLE FUNCTION KEYS

Your NCR DECISION MATE V has a row of special keys. These keys are labelled F1 through F20 and are located on the top row of the keyboard. They are special because you can define (program) them to do any function you want.

Like the automatic-program-execution feature (see next section), the programmable function keys are convenient, especially for performing an often-used or difficult function. For example, you may always want to check the contents of a disk before you access it. You could assign the directory display (DIR) command to a function key. Then, to use the command, you could simply press the key instead of typing the command through the keyboard.

Function keys are defined with the MS-DOS CONFIG utility. (See Chapter 5.)

RUNNING PROGRAMS AUTOMATICALLY

If you want to run a specific program automatically each time you start MS-DOS, you can do so with Automatic Program Execution. For example, you may want to have MS-DOS display the names of your files each time you load MS-DOS.

When you start MS-DOS, the command processor searches for a file named AUTOEXEC.BAT on the MS-DOS disk. This file is a program that MS-DOS will run each time MS-DOS is started. Chapter 4, Learning About Commands, tells you how to create an AUTOEXEC.BAT file.

TURNING THE SYSTEM OFF

There is no "logoff" command in MS-DOS. To end your terminal session, open the disk drive doors and remove the disks. Then, simply turn your terminal off in response to a default drive prompt.

FILES

A file is a collection of related information. A file on your disk can be compared to a file folder in a desk drawer. For example, one file folder might contain the names and addresses of the employees who work in the office. You might name this file the Employee Master File. A file on your disk could also contain the names and addresses of employees in the office and could also be named Employee Master File.

All programs, text, and data on your disk reside in files and each file has a unique name. You refer to files by their names. Chapter 3, More About Files, tells you how to name your files.

You create a file each time you enter and save data or text at your terminal. Files are also created when you write and name programs and save them on your disks.

HOW MS-DOS KEEPS TRACK OF YOUR FILES

The names of files are kept in directories on a disk. These directories also contain information on the size of the files, their location on the disk, and the dates that they were created and updated. The directory you are working in is called your current or *working* directory.

An additional system area is called the File Allocation Table. It keeps track of the location of your files on the disk. It also allocates the free space on your disks so that you can create new files.

These two system areas, the directories and the File Allocation Table, enable MS-DOS to recognize and organize the files on your disks. The File Allocation Table is copied to a new disk when you format it with the MS-DOS FORMAT command; also, one empty directory is created, called the *root* directory.

THE DIR (SHOW DIRECTORY) COMMAND

If you want to know what files are on your disk, you can use the DIR command. This command tells MS-DOS to display all the files in the current directory on the disk that is named. For example, if your MS-DOS disk is in drive A and you want to see the listing for the current directory on that disk, type:

```
    DIR A:<CR>
```

MS-DOS responds with a directory listing of all the files in the current directory on your MS-DOS disk. To stop the screen to study the files, press the CONTROL key, hold it down, and then press the S key. To continue the display, press any key.

NOTE: Two MS-DOS system files, IO.SYS and MSDOS.SYS, are "hidden" files and do not appear when you issue the DIR command.

You can also get information about any file on your disk by typing DIR and a filename. For example, if you have created a file named MYFILE.TXT, the command

```
DIR MYFILE.TXT <CR>
```

gives you a display of all the directory information (name of file, size of file, date last edited) for the file MYFILE.TXT.

For more information on the DIR command, refer to Chapter 5, MS-DOS Commands.

CHECKING YOUR DISKS

The MS-DOS command CHKDSK is used to check your disks for consistency and errors, much like a secretary proofreading a letter. CHKDSK analyzes the directories and the File Allocation Table on the disk that you specify. It then produces a status report of any inconsistencies, such as files which have a non-zero size in their directory but really have no data in them.

To check the disk in drive A, type:

```
CHKDSK A: <CR>
```

MS-DOS displays a status report and any errors that it has found. An example of this display and more information on CHKDSK can be found in the description of the CHKDSK command in Chapter 5. You should run CHKDSK occasionally for each disk to ensure the integrity of your files.

CHAPTER REVIEW

- Always begin processing from disk drive A. Turn on your computer, insert a master disk in drive A, and press <CR>.
- The date and time messages are displayed whenever MS-DOS is read into memory. Although these messages can be bypassed, they provide important information about when data was created or updated on your disk.
- The MS-DOS master disk that came with this manual is *write protected.* You made a backup copy of the diskette and should only use the new copy for processing. (Put the original master diskette in a safe place for system protection.)
- On your MS-DOS master diskette, the software "thinks" you have two flexible disk drives and a parallel printer. To "tell" MS-DOS differently, you must use the CONFIG utility to define and write the configuration description on the new master diskette.
- Each disk drive has a unique name (drive designation), which is an alphabetic character. A fixed disk unit has two logical disk drives and, therefore, two drive designations.
- The A> is the system default prompt. It tells you which disk drive MS-DOS is using and that MS-DOS is waiting for your direction. You can change the default drive designation by entering the drive designation followed by a colon (:).
- Most entries you make on the keyboard must end by pressing the <CR> key. This function key tells MS-DOS you have completed an entry.
- Any new disk must be formatted with the FORMAT command before it can be used by MS-DOS. Because of the high storage capacity of a fixed disk, formatting it can take several minutes.
- All data on your disk is stored in files and each file name is listed in a directory. The DIR command displays the directory.
- Always make a copy of important data on your disk using the appropriate MS-DOS copy commands: DISKCOPY, COPY, or BACKUP.
- Keys F1-F20 on the top row of your computer are available for your own use. You can "program" them (with the CONFIG utility) to do anything you want.
- MS-DOS has an automatic program execution feature. Whenever you load MS-DOS, the defined program is automatically executed.
- When you have finished processing, remove the flexible disk and then turn off your computer.

MORE ABOUT FILES

In Chapter 2, you learned that directories contain the names of your files. In this chapter, you learn how to name and copy your files. You also learn more about the MS-DOS hierarchical directory structure that makes it easy for you to organize and locate your files.

NAMING YOUR FILES

The name of a typical MS-DOS file looks like this:

```
NEWFILE.EXE
```

The name of a file consists of two parts. The filename is NEWFILE and the filename extension is .EXE.

A filename can be from 1 to 8 characters long. The filename extension can be three or fewer characters. You can type any filename in small or capital letters and MS-DOS will translate these letters into uppercase characters.

In addition to the filename and the filename extension, the name of your file may include a drive designation. A drive designation tells MS-DOS to look on the disk in the designated drive to find the filename typed. For example, to find directory information about the file NEWFILE.EXE which is located on the disk in drive A (and drive A is NOT the default drive), type the following command:

```
DIR A:NEWFILE.EXE
```

Directory information about the file NEWFILE.EXE is now displayed on your screen.

If drive A is the default drive, MS-DOS will search only the disk in drive A for the filename NEWFILE and so the drive designation is not necessary. A drive designation is needed if you want to tell MS-DOS to look on the other drive to find a file.

Your filenames will probably be made up of letters and numbers, but other characters are allowed, too. Legal characters for filename extensions are the same as those for filenames. Here is a complete list of the characters you can use in filenames and extensions:

A - Z   0 - 9 $ &   #

%   '   (   )   -   @

\   ^   [   ]   ~   `   !

All of the parts of a filename comprise a file specification. The term file specification (or filespec) is used in this manual to indicate the following filename format:

[<drive designation:>] <filename> [<.filename extension>]

Remember that brackets indicate optional items. Angle brackets (< >) mean that you supply the text for the item. Note that the drive designation is not required unless you need to indicate to MS-DOS on which disk to search for a specific file. You do not have to give your filename a filename extension.

Examples of file specifications are:

```
B:MYPROG.COB
A:YOURPROG.EXT
A:NEWFILE.
TEXT
```

WILD CARDS

Two special characters (called wild cards) can be used in filenames and extensions: the asterisk (*) and the question mark (?). These special characters give you greater flexibility when using filenames in MS-DOS commands.

The ? Wild Card

A question mark (?) in a filename or filename extension indicates that any character can occupy that position. For example, the MS-DOS command

```
DIR TEST?RUN.EXE
```

lists all directory entries on the default drive that have 8 characters, begin with TEST, have any next character, end with the letters RUN, and have a filename extension of .EXE. Here are some examples of files that might be listed by the previous DIR command:

```
TEST1RUN.EXE
TEST2RUN.EXE
TEST6RUN.EXE
```

The * Wild Card

An asterisk (*) in a filename or filename extension indicates that any character can occupy that position or any of the remaining positions in the filename or extension. For example:

```
DIR TEST*.EXE
```

lists all directory entries on the default drive with filenames that begin with the characters TEST and have an extension of .EXE. Here are some examples of files that might be listed by this DIR command:

```
TEST1RUN.EXE
TEST2RUN.EXE
TEST6RUN.EXE
TESTALL.EXE
```

The wild card designation *.* refers to all files on the disk. Note that this designation can be very powerful and destructive when used in MS-DOS commands. For example, the command DEL *.* deletes all files on the default drive, regardless of filename or extension.

Examples

To list the directory entries for all files named NEWFILE on drive A (regardless of filename extensions), simply type:

```
DIR A:NEWFILE.*
```

To list the directory entries for all files with filename extensions of .TXT (regardless of filenames) on the disk in drive B, type:

```
DIR B:????????.TXT
```

This command is useful if, for example, you have given all your text programs a filename extension of .TXT. By using the DIR commands with the wild card characters, you can obtain a listing of all your text files even if you do not remember all of their filenames.

ILLEGAL FILENAMES

MS-DOS treats some device names specially, and certain 3-letter names are reserved for the names of these devices. The following 3-letter names cannot be used as filenames or extensions.

AUX
: Used when referring to input from or output to an auxiliary device (such as a printer or disk drive).

CON
: Used when referring to keyboard input or to output to the terminal console (screen).

LST or PRN
: Used when referring to the printer device.

NUL
: Used when you do not want to create a particular file, but the command requires an input or output filename.

Even if you add device designations or filename extensions to these filenames, they remain associated with the devices listed above. For example, A:CON.XXX always refers to the console and is not the name of a disk file.

COPYING YOUR FILES

Just as with paper files, you often need more than one copy of a disk file. The COPY command allows you to copy one or more files to another disk. You can also give the copy a different name if you specify the new name in the COPY command.

The COPY command can also make copies of files on the same disk. In this case, you must assign a different filename or you will overwrite the file. You cannot make a copy of a file on the same disk unless you specify a different filename for the new copy.

The format of the COPY command is:

```
COPY filespec [filespec]
```

For example,

```
COPY A:MYFILE.TXT  B:MYFILE.TXT
```

copies the file MYFILE.TXT on disk A to a file named MYFILE. TXT on disk B. A duplicate copy of MYFILE.TXT now exists.

Figure 3.1 illustrates how to copy files to another disk:

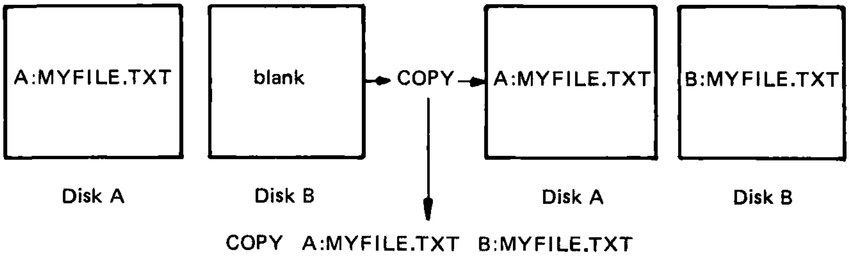


Figure 3.1 Copying files to another disk

If you want to duplicate the file named MYFILE. TXT on the same disk, type:

```
COPY A:MYFILE.TXT  A:NEWNAME.TXT
```

You now have two copies of your file on disk A, one named MYFILE.TXT and the other named NEWNAME.TXT. The following figure illustrates this example.

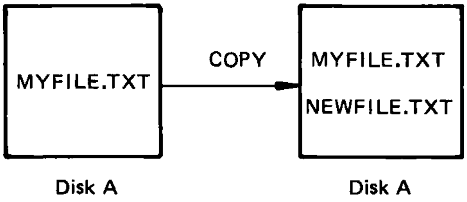


Figure 3.2 Copying files on the same disk

You can also copy all files on a disk to another disk (that is, make a backup copy) with the COPY command. Refer to Chapter 5, MS-DOS Commands, for more information on this process.

PROTECTING YOUR FILES

MS-DOS is a powerful and useful tool in processing your personal and business information. As with any information system, inadvertent errors may occur and information may be misused. If you are processing information that cannot be replaced or that requires a high level of security, you should take steps to ensure that your data and programs are protected from accidental or unauthorized use, modification, or destruction. Simple measures you can take - such as removing your disks when they are not in use, keeping backup copies of valuable information, and installing your equipment in a secure facility - can help you maintain the integrity of the information in your files. An MS-DOS command, CIPHER, can also be used to encrypt your files for total privacy. For more information on CIPHER, refer to Chapter 5, MS-DOS Commands.

DIRECTORIES

As you learned in Chapter 2, the names of your files are kept in a directory on each disk. The directory also contains information on the size of the files, their locations on the disk, and the dates that they were created and updated.

When there are multiple users on your computer, or when you are working on several different projects, the number of files in the directory can become large and unwieldy. You may want your own files kept separate from a co-worker's, or you may want to organize your programs into categories that are convenient for you.

In an office, you can separate files by putting them in different filing cabinets; in effect, creating different directories of information. MS-DOS allows you to organize the files on your disks into directories. Directories are a way of dividing your files into convenient groups of files. For example, you may want all of your accounting programs in one directory and text files in another. Any one directory can contain any reasonable number of files, and it may also contain other directories (referred to as subdirectories). This method of organizing your files is called a hierarchical directory structure.

A hierarchical directory structure can be thought of as a "tree" structure: directories are branches of the tree and files are the leaves, except that the "tree" grows downward; that is, the "root" is at the top. The root is the first level in the directory structure. It is the directory that is automatically created when you format a disk and start putting files in it. You can create additional directories and subdirectories by following the instructions in Chapter 4, Learning About Commands.

The tree or file structure grows as you create new directories for groups of files or for other people on the system. Within each new directory, files can be added, or new subdirectories can be created.

It is possible for you to "travel" around this tree; for instance, it is possible to find any file in the system by starting at the root and traveling down any of the branches to the desired file. Conversely, you can start where you are within the file system and travel towards the root.

The filenames discussed earlier in this chapter are relative to your current directory and do not apply system-wide. Thus, when you turn on your computer, you are "in" your directory. Unless you take special action when you create a file, the new file is created in the directory in which you are now working. Users can have files of the same name that are unrelated because each is in a different directory.

Figure 3.3 illustrates a typical hierarchical directory structure.

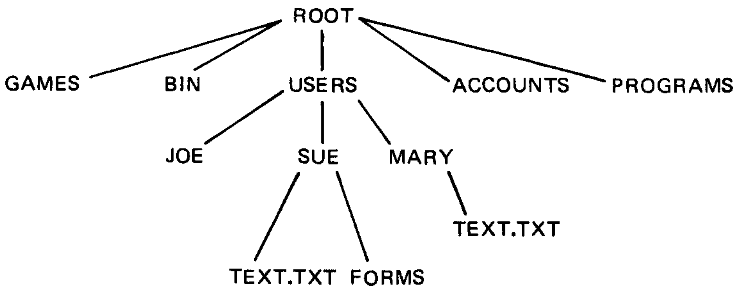


Figure 3.3 A sample hierarchical directory structure

The ROOT directory is the first level in the directory structure. You can create subdirectories from the ROOT by using the MKDIR command (refer to Chapter 5, MS-DOS Commands, for information on MKDIR). In this example, five subdirectories of ROOT have been created. These include:

- A directory of games, named GAMES
- A directory of all external commands, named BIN (refer to Chapter 4, Learning About Commands, for more information on the BIN directory)
- A USER directory containing separate subdirectories for all users of the system
- A directory containing accounting information, named ACCOUNTS
- A directory of programs, named PROGRAMS

Joe, Sue, and Mary each have their own directories which are subdirectories of the USER directory. Sue has a subdirectory under the \USER\SUE directory named FORMS. Sue and Mary have files in their directories, each named TEXT.TXT. Notice that Mary's text file is unrelated to Sue's.

This organization of files and directories is not important if you only work with files in your own directory, but, if you work with someone else or on several projects at one time, the hierarchical directory structure becomes extremely useful. For example, you could get a list of the files in Sue's FORMS directory by typing:

```
DIR \USER\SUE\FORMS
```

Note that the back slash mark (\) is used to separate directories from other directories and files.

To find out what files Mary has in her directory, you could type:

```
DIR \USER\MARY
```

FILENAMES AND PATHS

When you use hierarchical directories, you must tell MS-DOS where the files are located in the directory structure. Both Mary and Sue, for example, have files named TEXT.TXT. Each will have to tell MS-DOS in which directory her file resides if she wants to access it. This is done by giving MS-DOS a pathname to the file.

Pathnames

A simple filename is a sequence of characters that can optionally be preceded by a drive designation and followed by an extension. A pathname is a sequence of directory names followed by a simple filename, each separated from the previous one by a slash (\ ).

The syntax of pathnames is:

```
[<d>:] [<directory>] \ [<directory. . . >] \ [<filename>]
```

If a pathname begins with a slash, MS-DOS searches for the file beginning at the root (or top) of the tree; otherwise, MS-DOS begins at the user's current directory, known as the working directory, and searches downward from there. The pathname of Sue's TEXT.TXT file is \USER\SUE\TEXT. TXT.

When you are in your working directory, a filename and its corresponding pathname may be used interchangeably. The following list shows some sample names:

\
  Indicates the root directory.

\PROGRAMS
  Sample directory under the root directory containing program files.

\USER\MARY\FORMS\1A
  A typical full pathname. This one happens to be a file named 1A in the directory named FORMS belonging to the USER named MARY.

USER\SUE
  A relative pathname; it names the file or directory SUE in subdirectory USER of the working directory. If the working directory is the root (\ ), it names \BIN\SUE.

TEXT.TXT
  Name of a file or directory in the working directory.

MS-DOS provides special shorthand notations for the working directory and the parent directory (one level up) of the working directory. For example, in the sample of a hierarchical directory structure in this chapter, the parent of the directory JOE is USER.

. (single period)
  This shorthand notation indicates the name of the working directory in all hierarchical directory listings and is auto-

matically created by MS-DOS. For example, if your working directory is JOE and you issue the DIR command, MS-DOS displays a single period to represent your working directory instead of the filename.

. . (double period)

This is the shorthand notation for the working directory's parent directory. In the above example, MS-DOS displays a double period to represent the parent directory USER. You may use the double period when specifying a path to MS-DOS as a shorthand way of telling MS-DOS to go back one directory level. For example, if your working directory is JOE and you wish to find the file FORMS in USER SUE's directory, you can specify this in either of two ways:

```
\USER\SUE\FORMS
```

or

```
..\SUE\FORMS
```

The double period causes MS-DOS to go back one level and to continue the path from there.

Pathing and External Commands - External commands reside on disks as program files. They must be read from the disk before they execute. (For more information on external commands, refer to Chapter 4, Learning About Commands.)

When you are working with more than one directory, it is convenient to put all MS-DOS external commands into a separate directory so they do not clutter your other directories. When you issue an external command to MS-DOS, MS-DOS immediately checks your working directory to find that command. You must tell MS-DOS in which directory these external commands reside. This is done with the PATH command.

For example, if you are in a working directory named \BIN\PROG, and all MS-DOS external commands are in \BIN, you must tell MS-DOS to choose the \BIN path to find the FORMAT command. The command

```
PATH \BIN
```

tells MS-DOS to search in your working directory and the \BIN directory for all commands. You only have to specify this path once to MS-DOS during your terminal session. MS-DOS will now search in \BIN for the external commands. If you want to know what the current path is, type the word PATH and the current value of PATH will be printed.

For more information on the MS-DOS command PATH, refer to Chapter 5, MS-DOS Commands.

THIS PAGE INTENTIONALLY LEFT BLANK

Pathing and Internal Commands - Internal commands are the simplest, most commonly used commands. They execute immediately because they are incorporated into the command processor. (For more information on internal commands, refer to Chapter 4, Learning About Commands.)

Some internal commands can use paths. The four commands, COPY, DIR, DEL, and TYPE, have greater flexibility when you specify a pathname after the command.

COPY <pathname pathname>
: If the second pathname to COPY is a directory, all files are copied into that directory. The first pathname may only specify files in the working directory.

DEL <pathname>
: If the pathname is a directory, all the files in that directory are deleted. Note: The prompt "Are you sure (Y/N)?" is displayed if you try to delete a path. Type Y to complete the command, or type N for the command to abort.

DIR <pathname>
: Displays the directory for a specific path.

TYPE <pathname>
: You must specify a file in a path for this command. MS-DOS will display the file on your screen in response to the TYPE pathname command.

DISPLAYING YOUR WORKING DIRECTORY

All commands are executed while you are in your working directory. You can find out the name of the directory you are in by issuing the MS-DOS command CHDIR (Change Directory) with no options. For example, if your current directory is \USER\JOE, when you type:

```
CHDIR<RETURN>
```

you will see:

```
A: \USER\JOE
```

This is your current drive designation plus the working directory (\USER\JOE).

If you now want to see what is in the \USER\JOE directory, you can issue the MS-DOS command DIR. The following is an example of the display you might receive from the DIR command for a subdirectory:

```
Volume in drive A has no ID
Directory of A: \USER\JOE

.               <DIR>         5-09-83   10:09a
..              <DIR>         5-09-83   10:09a
TEXT            <DIR>         5-09-83   10:09a
FILE1    COM             5243 5-04-83    9:30a
         4 File(s)     250518 bytes free
```

A volume ID for this disk was not assigned when the disk was formatted. Note that MS-DOS lists both files and directories in this output. As you can see, Joe has another directory in this tree structure named TEXT. The '.' indicates the working directory \USER\JOE, and '. .' is the shorthand notation for the parent directory \USER. FILE1. COM is a file in the \USER\JOE directory. All of these directories and files reside on the disk in drive A.

Because files and directories are listed together (see previous display), MS-DOS does not allow you to give a subdirectory the same name as a file in that directory. For example, if you have a path \BIN\ USER\JOE where JOE is a subdirectory, you cannot create a file in the USER directory named JOE.

CREATING A DIRECTORY

To create a subdirectory in your working directory, use the MKDIR (Make Directory) command. For example, to create a new directory named NEWDIR under your working directory, simply type:

```
MKDIR NEWDIR
```

After this command is executed by MS-DOS, a new directory exists in your tree structure under your working directory. You can also make directories anywhere in the tree structure by specifying MKDIR and then a pathname. MS-DOS automatically creates the . and . . entries in the new directory.

To put files in the new directory, use the MS-DOS Line Editor, EDLIN. Chapter 7, Line Editor (EDLIN), describes how to use EDLIN to create and save files.

CHANGING YOUR WORKING DIRECTORY

Changing from your working directory to another directory is very easy in MS-DOS. Simply issue the CHDIR (Change Directory) command and supply a pathname. For example:

```
A:CHDIR \USER
```

changes the working directory from \USER\JOE to \USER. You can specify any pathname after the command to "travel" to different branches and leaves of the directory tree. The command "CHDIR . ." will always put you in the parent directory of your working directory.

REMOVING A DIRECTORY

To delete a directory in the tree structure, use the MS-DOS RMDIR (Remove Directory) command. For example, to remove the directory NEWDIR from the working directory, type:

```
RMDIR NEWDIR
```

Note that the directory NEWDIR must be empty except for the . and . . entries before it can be removed; this will prevent you from accidentally deleting files and directories. You can remove any directory by specifying its pathname. To remove the \BIN\USER\JOE directory, make sure that it has only the . and . . entries, then type:

```
RMDIR \BIN\USER\JOE
```

To remove all the files in a directory (except for the . and . . entries), type DEL and then the pathname of the directory. For example, to delete all files in the \BIN\USER\SUE directory, type:

```
DEL \BIN\USER\SUE
```

You cannot delete the . and . . entries. They are created by MS-DOS as part of the hierarchical directory structure.

In the next chapter, you will learn about MS-DOS commands.

LEARNING ABOUT COMMANDS

GENERAL INFORMATION

Commands are a way of communicating with the computer. By entering MS-DOS commands at your terminal, you can ask the system to perform useful tasks:

- Compare, copy, display, delete, and rename files
- Copy and format disks
- Execute system programs such as EDLIN, as well as your own programs
- Analyze and list directories
- Enter date, time, and remarks
- Set various printer and screen options
- Copy MS-DOS system files to another disk
- Request MS-DOS to wait for a specific period of time

TYPES OF MS-DOS COMMANDS

There are two types of MS-DOS commands: internal commands and external commands.

Internal Commands

Internal commands are the simplest, most commonly used commands. You cannot see these commands when you do a directory listing on your MS-DOS disk; they are part of the command processor. When you type these commands, they execute immediately. The following internal commands are described in Chapter 5:

BREAK
CHDIR (CD)
CLS
COPY
CTTY
DATE
DEL (ERASE)
DIR
ECHO
EXIT
FOR
GOTO
IF
MKDIR (MD)
PATH
PAUSE
PROMPT
REM
REN (RENAME)
RMDIR (RD)
SET
SHIFT
TIME
TYPE
VER
VERIFY
VOL

External Commands

External commands reside on disk as program files. They must be read from disk before they can execute. If the disk containing the command is not in the drive, MS-DOS will not be able to find and execute the command.

Any filename with a filename extension of .COM, .EXE or .BAT is considered an external command. For example, the program FORMAT.COM is an external command. Because all external commands reside on disk, you can create commands and add them to the system. Programs that you create with most languages (including assembly language) will be .EXE (executable files).

When you enter an external command, do not include its filename extension. The following external commands are described in Chapter 5:

BACKUP
CHKDSK
CIPHER
CONFIG
DISKCOPY
FIND
FORMAT
LOCATE
MORE
PRINT
RDCPM
RECOVER
SORT
SYS

COMMAND OPTIONS

Options can be included in your MS-DOS commands to specify additional information to the system. If you do not include some options, MS-DOS provides a default value. Refer to individual command descriptions in Chapter 5 for the default values.

The following is the format of all MS-DOS commands:

```
Command [options. . .]
```

where:

d:
Refers to the disk drive designation.

filename
Refers to any valid name for a disk file, including an optional filename extension. The filename option does not refer to a device or to a disk drive designation.

.ext
Refers to an optional filename extension consisting of a period and 1-3 characters. When used, filename extensions immediately follow filenames.

filespec

Refers to an optional drive designation, a filename, and an optional three letter filename extension in the following format:

```
[<d:>] <filename> [<.ext>]
```

pathname

Refers to a pathname or filename in the following format:

```
[<directory>] \ [<directory. . .>] \ [<filename>]
```

switches

Switches are options that control MS-DOS commands. They are preceded by a forward slash (for example, /P).

arguments

Provide more information to MS-DOS commands. You usually choose between arguments; for example, ON or OFF.

COMMON ENTRY CONVENTIONS

The following information applies to all MS-DOS commands:

1. Commands are usually followed by one or more options.
2. Commands and options may be entered in uppercase of lowercase, or a combination of keys.
3. Commands and options must be separated by delimiters. Because they are easiest, you will usually use the space and comma as delimiters. For example:

   ```
   DEL MYFILE.OLD NEWFILE.TXT
   RENAME,THISFILE THATFILE
   ```

   You can also use the semicolon (;), the equal sign (=), or the tab key as delimiters in MS-DOS commands. (In this manual, we use a space as the delimiter in commands.)
4. Do not separate a file specification with delimiters, since the colon and the period already serve as delimiters.
5. When instructions say "Press any key," you can press any alpha (A-Z) or numeric (0-9) key.
6. You must include the filename extension when referring to a file that already has a filename extension.
7. You can abort commands when they are running by pressing <CONTROL-C>.

8. Commands take effect only after you have pressed the<CR> key.
9. Wild cards (global filename characters) and device names (for example, PRN or CON) are not allowed in the names of any commands.
10. When commands produce a large amount of output on the screen, the display automatically scrolls to the next screen. You can press <CONTROL-S> to suspend the display. Press any key to resume the display on the screen.
11. MS-DOS editing and function keys can be used when entering commands. Refer to Chapter 6, MS-DOS Editing and Function Keys, for a complete description of these keys.
12. The prompt from the command processor is the default drive designation plus a greater-than (>) sign; for example, A>.
13. Disk drives will be referred to as source drives and destination drives. A source drive is the drive you transfer information from; a destination drive is the drive you transfer information to.

BATCH PROCESSING

Often you may find yourself typing the same sequence of commands over and over to perform some commonly used task. With MS-DOS, you can put the command sequence into a special file, called a batch file, and execute the entire sequence simply by typing the name of the batch file. "Batches" of your commands in such files are processed as if they were typed at a terminal. Each batch file must be named with the .BAT extension, and is executed by typing the filename without its extension.

You can create a batch file by using the Line Editor (EDLIN) or by typing the COPY command. Refer to the Creating an AUTOEXEC.BAT File section later in this chapter for more information on using the COPY command to create a batch file.

Two MS-DOS commands are available for use expressly in batch files: REM and PAUSE. REM permits you to include remarks and comments in your batch files without these remarks being executed as commands. PAUSE prompts you with an optional message and permits you to either continue or abort the batch process at a given point. REM and PAUSE are described in detail in Chapter 5.

Batch processing is useful if you want to execute several MS-DOS commands with one batch command, such as when you format and check a new disk. For example, a batch file for this purpose might look like this:

```
1: REM  This is a file to check new disks
2: REM  It is named NEWDISK.BAT
3: PAUSE  Insert new disk in drive B:
4: FORMAT B:
5: DIR B:
6: CHKDSK B:
```

To execute this .BAT file, simply type the filename without the .BAT extension:

```
NEWDISK
```

The result is the same as if each of the lines in the .BAT file was entered at the terminal as individual commands.

Figure 4.1 illustrates the 3 steps used to write, save, and execute an MS-DOS batch file.

The following list contains information that you should read before you execute a batch process with MS-DOS.

1. Do not enter the filename BATCH (unless the name of the file you want to execute is BATCH.BAT).
2. Enter only the filename to execute the batch file; do not enter the filename extension.
3. The commands in the file named <filename>.BAT are executed.
4. If you press <CONTROL-C> while in batch mode, this prompt appears:

   ```
   Terminate batch job (Y/N) ?
   ```

   If you press Y, the remainder of the commands in the batch file are ignored and the system prompt appears.

   If you press N, only the current command ends and batch processing continues with the next command in the file.

5. If you remove the disk containing a batch file being executed, MS-DOS prompts you to insert it again before the next command can be read.
6. The last command in a batch file may be the name of another batch file. This allows you to call one batch file from another when the first is finished.

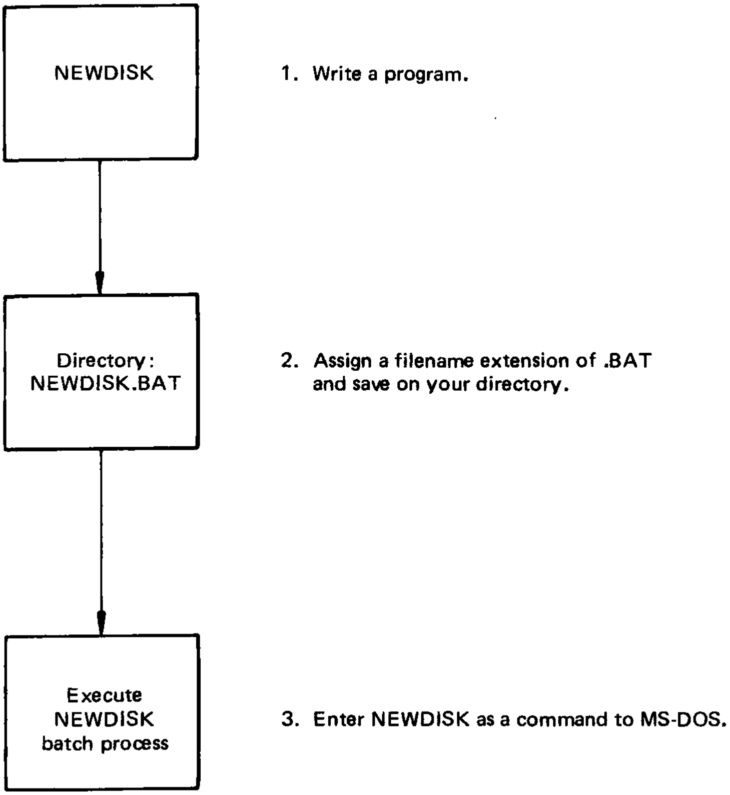

NEWDISK
1. Write a program.
Directory: NEWDISK.BAT
2. Assign a filename extension of .BAT and save on your directory.
Execute NEWDISK batch process
3. Enter NEWDISK as a command to MS-DOS.

Figure 4.1 MS-DOS batch file steps

THE AUTOEXEC.BAT FILE

As discussed in Chapter 2, an AUTOEXEC.BAT file allows you to automatically execute programs when you start MS-DOS. Automatic Program Execution is useful when you want to run a specific application package under MS-DOS, or when you want MS-DOS to execute a batch program automatically each time you start the system. You can avoid loading two separate disks to perform either of these tasks by using an AUTOEXEC.BAT file.

When you start MS-DOS, the command processor searches the MS-DOS disk for a file named AUTOEXEC.BAT. The AUTOEXEC.BAT file is a batch file that is automatically executed each time you start the system.

If MS-DOS finds the AUTOEXEC.BAT file, the file is immediately executed by the command processor and the date and time prompts are bypassed.

If MS-DOS does not find an AUTOEXEC.BAT file when you first load the MS-DOS disk, then the date and time prompts are issued. Figure 4.2 illustrates how MS-DOS uses the AUTOEXEC.BAT file.

CREATING AN AUTOEXEC.BAT FILE

To see how to create an AUTOEXEC.BAT file, assume that each time you start MS-DOS, you want to automatically load BASIC and run a program called MENU. You could create an AUTOEXEC.BAT file as follows:

1. Type:

   ```
   COPY CON: AUTOEXEC.BAT
   ```

   This statement tells MS-DOS to copy the information from the console (keyboard) into the AUTOEXEC.BAT file. Note that the AUTOEXEC.BAT file must be created in the root directory of your MS-DOS disk.
2. Now type:

   ```
   BASIC MENU
   ```

   This statement goes into the AUTOEXEC.BAT file. It tells MS-DOS to load BASIC and run the MENU program whenever MS-DOS is started.
3. Press the <CONTROL-Z> key; then press the <CR> key to put the command BASIC MENU in the AUTOEXEC.BAT file.
4. The MENU program will now run automatically whenever you start MS-DOS.

To run your own BASIC program, enter the name of your program in place of MENU in the second line of the example. You can enter any MS-DOS command or series of commands in the AUTOEXEC.BAT file.

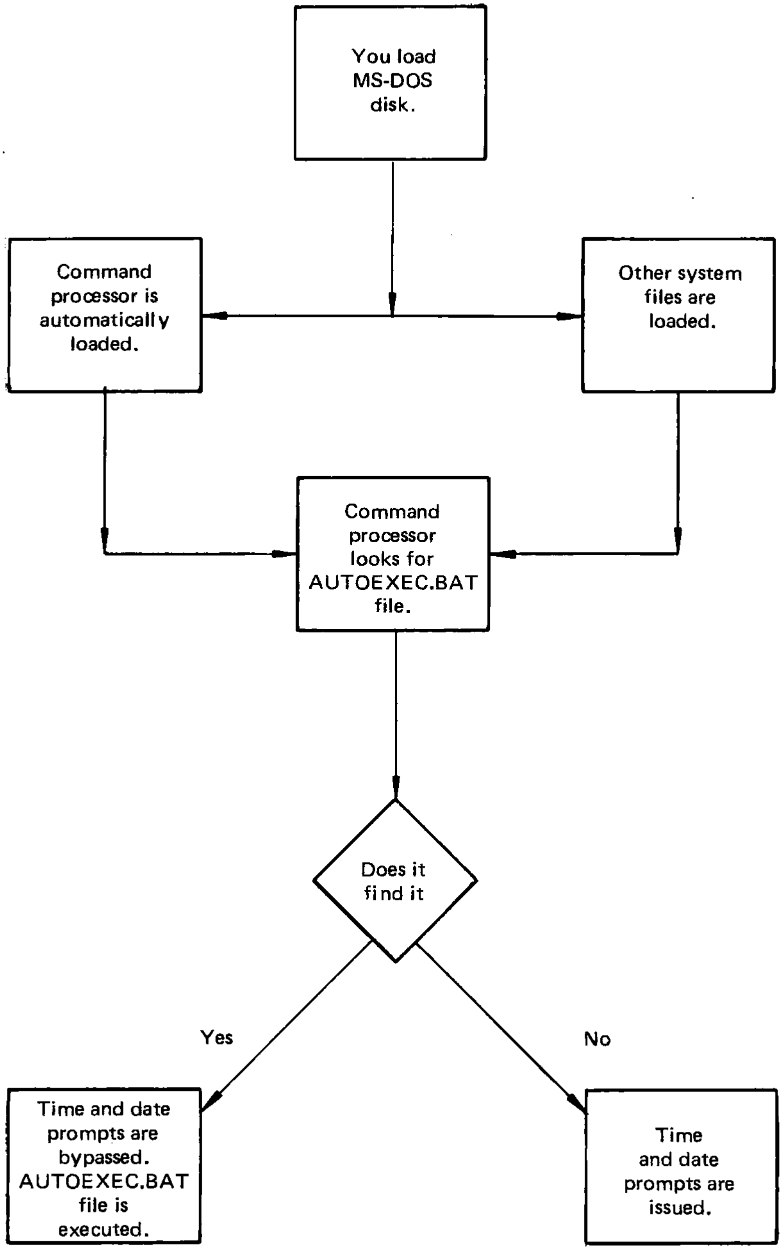

You load MS-DOS disk.
Command processor is automaticall y loaded.
Other system files are loaded.
Command processor looks for AUTOEXEC.BAT file.
Does it find it
Yes
No
Time and date prompts are bypassed. AUTOEXEC.BAT file is executed.
Time and date prompts are issued.

Figure 4.2 How MS-DOS uses the AUTOEXEC.BAT file

NOTE: Remember that if you use an AUTOEXEC.BAT file, MS-DOS does not prompt you for a current date and time unless you include the DATE and TIME commands in the AUTOEXEC.BAT file. You should include these two commands in your AUTOEXEC.BAT file, since MS-DOS uses this information to keep your directory current.

CREATING A .BAT FILE WITH REPLACEABLE PARAMETERS

There may be times when you want to create an application program and run it with different sets of data. This data may be stored in various MS-DOS files.

When used in MS-DOS commands, a parameter is an option that you define. With MS-DOS, you can create a batch (.BAT) file with dummy (replaceable) parameters. These parameters, named %0-%9, can be replaced by values supplied when the batch file executes.

For example, when you type the command line COPY CON MYFILE.BAT, the next lines you type are copied from the console to a file named MYFILE.BAT on the default drive:

```
A>COPY CON MYFILE.BAT
COPY %1.MAC %2.MAC
TYPE %2.PRN
TYPE %0.BAT
```

Now, press <CONTROL-Z> and then press <CR>. MS-DOS responds with this message:

```
1 File(s) copied
A>
```

The file MYFILE.BAT, which consists of three commands, now resides on the disk in the default drive.

The dummy parameters %1 and %2 are replaced sequentially by the parameters you supply when you execute the file. The dummy parameter %0 is always replaced by the drive designator, if specified, and the filename of the batch file (for example, MYFILE).

NOTE: Up to 10 dummy parameters (%0-%9) can be specified. Refer to the MS-DOS command SHIFT in Chapter 5 if you wish to specify more than 10 parameters. Also, if you use the percent sign as part of a filename within a batch file, you must type it twice. For example, to specify the file ABC%.EXE, you must type it as ABC%%.EXE in the batch file.

EXECUTING A .BAT FILE

To execute the batch file MYFILE.BAT and to specify the parameters that will replace the dummy parameters, you must enter the batch filename (without its extension) followed by the parameters you want MS-DOS to substitute for %1, %2, etc.

Remember that the file MYFILE.BAT consists of 3 lines:

```
COPY %1.MAC %2.MAC
TYPE %2.PRN
TYPE %0.BAT
```

To execute the MYFILE batch process, type:

```
MYFILE A:PROG1 B:PROG2
```

MYFILE is substituted for %0, A:PROG1 for %1, and B:PROG2 for %2.

The result is the same as if you had typed each of the commands in MYFILE with their parameters, as follows:

```
COPY A:PROG1.MAC B:PROG2.MAC
TYPE B:PROG2.PRN
TYPE MYFILE.BAT
```

The following table illustrates how MS-DOS replaces each of the above parameters:

BATCH FILENAME	PARAMETER1 (%0) (MYFILE)	PARAMETER2 (%1) (PROG1)	PARAMETER3 (%2) (PROG2)
MYFILE	MYFILE.BAT	PROG1.MAC	PROG2.MAC PROG2.PRN

Remember that the dummy parameter %0 is always replaced by the drive designator (if specified) and the filename of the batch file.

INPUT AND OUTPUT

MS-DOS always assumes that input comes from the keyboard and output goes to the terminal screen. However, the flow of command input and output can be redirected. Input can come from a file rather than a terminal keyboard, and output can go to a file or to a line printer instead of to the terminal. In addition, "pipes" can be created that allow output from one command to become the input to another. Redirection and pipes are discussed in the next sections.

REDIRECTING YOUR OUTPUT

Most commands produce output that is sent to your terminal. You can send this information to a file by using a greater-than sign (>) in your command. For example, the command

```
DIR
```

displays a directory listing of the disk in the default drive on the terminal screen. The same command can send this output to a file named MYFILES by designating the output file on the command line:

```
DIR >MYFILES
```

If the file MYFILES does not already exist, MS-DOS creates it and stores your directory listing in it. If MYFILES already exists, MS-DOS overwrites what is in the file with the new data.

If you want to append your directory or a file to another file (instead of replacing the entire file), two greater-than signs (>>) can be used to tell MS-DOS to append the output of the command (such as directory listing) to the end of a specified file. The command

```
DIR >>MYFILES
```

appends your directory listing to a currently existing file named MYFILES. If MYFILES does not exist, it is created.

It is often useful to have input for a command come from a file rather than from a terminal. This is possible in MS-DOS by using a less-than sign (<) in your command. For example, the command

```
    SORT <NAMES> LIST1
```

sorts the file NAMES and sends the sorted output to a file named LIST1.

FILTERS

A filter is a command that reads your input, transforms it in some way, and then outputs it, usually to your terminal or to a file. In this way, the data is said to have been "filtered" by the program. Since filters can be put together in many different ways, a few filters can take the place of a large number of specific commands.

MS-DOS filters include CIPHER, FIND, MORE, and SORT, and perform the following functions:

CIPHER
  Encrypts/decrypts a file.
FIND
  Searches for a constant string of text in a file.
MORE
  Takes standard terminal output and displays it, one screen at a time.
SORT
  Sorts text.

You can see how these filters are used in the next section.

COMMAND PIPING

If you want to give more than one command to the system at a time, you can "pipe" commands to MS-DOS. For example, you may occasionally need to have the output of one program sent as the input to another program. A typical case would be a program that produces output in columns. It could be desirable to have this columnar output sorted.

Piping is done by separating commands with the pipe separator, which is the vertical bar symbol (|). For example, the command

```
    DIR | SORT
```

gives you an alphabetically sorted listing of your directory. The vertical bar causes all output generated by the left side of the bar to be sent to the right side of the bar for processing.

Piping can also be used when you want to output to a file. If you want your directory sorted and sent to a new file (for example, DIREC.FIL), you could type:

```
DIR | SORT >DIREC.FIL
```

MS-DOS creates a file named DIREC.FIL on your default drive. DIREC.FIL contains a sorted listing of the directory on the default drive, since no other drive was specified in the command. To specify a drive other than the default drive, type:

```
DIR | SORT >B:DIREC.FIL
```

This sends the sorted data to a file named DIREC.FIL on drive B.

A pipeline may consist of more than two commands. For example,

```
DIR | SORT | MORE
```

sorts your directory, shows it to you one screen at a time, and puts "--MORE--" at the bottom of your screen when there is more output to be seen.

You will find many uses for piping commands and filters. You will also find more information on using filters in the next chapter, MS-DOS Commands.

MS-DOS COMMANDS

This section describes each of the MS-DOS commands, arranged in alphabetical order for quick reference. Certain commands are used only if you are writing batch programs. These commands, ECHO, FOR, GOTO, IF, and SHIFT, are noted as batch processing commands in the description. The individual command descriptions are preceded by a table summarizing the complete set.

Before studying or using any of the commands, be sure to become familiar with the notations that indicate how to format a command. (The notations were explained in an earlier chapter, but are important enough to repeat.)

- Words shown in capital letters are required entries. These words are called keywords and must be entered exactly as shown. You can enter these keywords in any combination of upper/lowercase; MS-DOS converts all keywords to uppercase.
- You supply the text for any items enclosed in angle brackets (< >). For example, you should enter the name of *your* file when <filename> is shown in the format.
- Items in square brackets ([ ]) are optional. If you include optional information,.do not include the square brackets, only the information within the brackets.
- An ellipsis (. . .) indicates that you may repeat an item as many times as you want.
- You must include all punctuation where shown (with the exception of square brackets), such as commas, equal signs, question marks, colons, or slashes.

MS-DOS COMMAND SUMMARY

Name (Synonym)	Purpose	Syntax
BACKUP	Copies fixed disk to flexible disks	BACKUP
BREAK	Sets CONTROL-C check	BREAK ON BREAK OFF
CHDIR (CD)	Changes directories; prints working directory	CHDIR [pathname]
CHKDSK	Scans the directory of the default or designated drive and checks for consistency	CHKDSK [d:] <filespec> [/F] [/V]
CIPHER	Encrypts/decrypts a file	CIPHER <keyword> [<filename>]
CLS	Clears screen	CLS
CONFIG	Defines configuration information	CONFIG
COPY	Copies file(s) specified	COPY <filespec> [filespec] [pathname] [pathname] [/V]
CTTY	Changes console TTY	CTTY \DEV\DEV
DATE	Displays and sets date	DATE [<mm>-<dd>-<yy>]
DEL (ERASE)	Deletes file(s) specified	DEL [filespec] [pathname]
DIR	Lists requested directory entries	DIR [filespec] [pathname] [/P] [/W]
DISKCOPY	Copies disks	DISKCOPY [d:] [d:]
ECHO	Turns batch file echo feature on/off	ECHO [ON message] ECHO [OFF message]
EXIT	Exits command and returns to lower level	EXIT
FIND	Searches for a constant string of text	FIND [/V /C /N] <string> [<filename . . .>]
FOR	Batch command extension	For batch processing: FOR %%<c> IN <set> DO <command> For interactive processing: FOR %<c> IN <set> DO <command>
FORMAT	Formats a disk to receive MS-DOS file • fixed disk • flexible disk	 FORMAT [d] : [/V] FORMAT [d:] [/V /J /D /I /O /S]
GOTO	Batch command extension	GOTO <label>
IF	Batch command extension	IF <condition> <command>

MS-DOS COMMAND SUMMARY (Cont.)

Name (Synonym)	Purpose	Syntax
LOCATE	Converts executable files to binary format	LOCATE <filespec> [d:] [<filename> [<.ext>]]
MKDIR (MD)	Makes a directory	MKDIR <pathname>
MORE	Displays output one screen at a time	MORE
PATH	Sets a command search path	PATH [<pathname>[;<pathname>] . . .]
PAUSE	Pauses for input in a batch file	PAUSE [comment]
PRINT	Background print feature	PRINT [[filespec] [/T] [/C] [/P]] . . .
PROMPT	Designates command prompt	PROMPT [<prompt-text>]
RECOVER	Recovers a bad disk	RECOVER <filename> RECOVER <d:>
REM	Displays a comment in a batch file	REM [comment]
REN (RENAME)	Renames first file as second file	REN <filespec> <filename>
RDCPM	Transfers CP/M files to an MS-DOS formatted disk	RDCPM DIR d: RDCPM d: filename [d:]
RMDIR (RD)	Removes a directory	RMDIR [d:] <pathname>
SET	Sets one string value to another	SET [<string = string>]
SHIFT	Increases number of replaceable parameters in batch process	SHIFT
SORT	Sorts data alphabetically, forward or backward	SORT [/R] [/+n]
SYS	Transfers MS-DOS system files from drive A: to the drive specified	SYS <d>:
TIME	Displays and sets time	TIME [<hh> [:<mm>]]
TYPE	Displays the contents of file specified	TYPE <filespec>
VER	Prints MS-DOS version number	VER
VERIFY	Verifies writes to disk	VERIFY [ON] VERIFY [OFF]
VOL	Prints volume identification number	VOL [d:]

BACKUP

NAME

BACKUP

TYPE

External

PURPOSE

Copies the contents of one of the logical fixed disks in the source drive to flexible disks; also restores the fixed disk.

SYNTAX

BACKUP

COMMENTS

Before copying begins, BACKUP asks for the source and destination drive designations, the 6-character volume ID (label) to be placed on each flexible disk, and if write with verify is to be performed.

After answering the questions, insert a formatted flexible disk in the destination disk drive and press <CR> to start the copy. (The flexible disks must be formatted using no switches.) As soon as one flexible disk is filled, BACKUP prompts you to insert the next disk.

NOTE: BACKUP copies the entire contents of one logical fixed disk. If you only want to copy selected files, use the COPY command.

To restore the fixed disk, simply reverse the copy: specify the flexible disk drive as the source and the fixed disk drive as the destination. Messages are displayed if the ID you enter does not match the ID on the flexible disk or if you insert a flexible disk out of sequence.

BREAK

NAME

BREAK

TYPE

Internal

PURPOSE

Sets CONTROL-C check.

SYNTAX

```
BREAK ON
BREAK OFF
```

COMMENTS

If you are running an application program that uses CONTROL-C function keys, you will want to turn off the MS-DOS CONTROL-C function so that when you press <CONTROL-C> you affect your program and not the operating system. Specify BREAK OFF to turn off CONTROL-C and BREAK ON when you have finished running your application program and are using MS-DOS.

CHDIR

NAME

CHDIR (CHANGE DIRECTORY)

TYPE

Internal

SYNONYM

CD

PURPOSE

Changes directory to a different path; displays current (working) directory.

SYNTAX

CHDIR [pathname]

COMMENTS

If your working directory is \BIN\USER\JOE and you want to change your path to another directory (such as \BIN\USER\JOE\FORMS), type:

```
    CHDIR \BIN\USER\JOE\FORMS
```

and MS-DOS puts you in the new directory. A shorthand notation is also available with this command:

```
    CHDIR ..
```

This command always puts you in the parent directory of your working directory.

CHDIR used without a pathname displays your working directory. If your working directory is \BIN\USER\JOE on drive B, and you type CHDIR <CR>, MS-DOS displays:

```
    B: \BIN\USER\JOE
```

This command is useful if you forget the name of your working directory.

CHKDSK

NAME

CHKDSK (CHECK DISK)

TYPE

External

PURPOSE

Scans the directory of the specified disk drive and checks it for consistency.

SYNTAX

```
CHKDSK [d:] <filespec> [/F] [/V]
```

COMMENTS

CHKDSK should be run occasionally on each disk to check for errors in the directory. If any errors are found, CHKDSK displays error messages, if any, and then a status report similar to the one below.

```
    160256   bytes total disk space
      8192   bytes in 2 hidden files
       512   bytes in 2 directories
     30720   bytes in 8 user files
    121344   bytes available on disk

     65536   bytes total memory
     53152   bytes free
```

CHKDSK does not correct the errors found in your directory unless you specify the /F (fix) switch. Typing /V causes CHKDSK to display messages while it is running.

You can redirect the output from CHKDSK to a file. Simply type:

```
    CHKDSK A:>filename
```

The errors are sent to the filename specified. Do not use the /F switch if you redirect CHKDSK output.

The following errors are corrected automatically if you specify the /F switch:

Invalid drive specification

Invalid parameter

Invalid sub-directory entry

Cannot CHDIR to <filename>
Tree past this point not processed

First cluster number is invalid
entry truncated

Allocation error, size adjusted

Has invalid cluster, file truncated

Disk error reading FAT

Disk error writing FAT

<filename> contains
non-contiguous blocks

All specified file(s) are contiguous

You must correct the following errors returned by CHKDSK, even if you specified the /F switch:

Incorrect DOS version

> You cannot run CHKDSK on versions of MS-DOS that are not 2.0 or higher.

Insufficient memory
Processing cannot continue

> There is not enough memory in your machine to process CHKDSK for this disk. You must obtain more memory to run CHKDSK.

Errors found, F parameter not specified
Corrections will not be written to disk

> You must specify the /F switch if you want the errors corrected by CHKDSK.

Invalid current directory
Processing cannot continue

Restart the system and rerun CHKDSK.

Cannot CHDIR to root
Processing cannot continue

The disk you are checking is bad. Try restarting MS-DOS and RECOVER the disk.

<filename> is cross linked on cluster

Make a copy of the file you want to keep, and then delete both files that are cross linked.

X lost clusters found in y chains
Convert lost chains to file (Y/N)?

If you respond Y to this prompt, CHKDSK creates a directory entry and a file for you to resolve this problem (files created by CHKDSK are named FILEnnnnnnnn).

CHKDSK then displays:

X bytes disk space freed

If you respond N to this prompt and have not specified the /F switch, CHKDSK frees the clusters and displays:

X bytes disk space would be freed

Probable non-DOS disk
Continue (Y/N)?

The disk you are using is a non-DOS disk. You must indicate whether or not you want CHKDSK to continue processing.

Insufficient room in root directory
Erase files in root and repeat CHKDSK

CHKDSK cannot process until you delete files in the root directory.

Unrecoverable error in directory
Convert directory to file (Y/N) ?

If you respond Y to this prompt, CHKDSK converts the bad directory into a file. You can then fix the directory yourself or delete it.

CIPHER

NAME

CIPHER

TYPE

External

PURPOSE

Encrypts and decrypts files based on a specified keyword.

SYNTAX

CIPHER <keyword> [<filename>]

COMMENTS

Use this command when you want to encrypt a file for security purposes. The CIPHER command uses a keyword that must be provided when encrypting the file. To encrypt the file NSA.CIA using the keyword "SECRET," enter:

```
CIPHER SECRET < NSA.CIA
```

This displays the encrypted file (NSA.CIA) on your screen. If you want the encrypted file sent to another file, enter:

```
CIPHER  SECRET <NSA.CIA >MYSTERY.NEW
```

where MYSTERY.NEW is the name of the file where you are storing the encrypted file. You may delete the original file NSA. CIA.

To decrypt the encrypted file MYSTERY.NEW, simply reverse the process:

```
CIPHER SECRET  <MYSTERY.NEW
```

This command decrypts the file MYSTERY.NEW and displays it on your screen. If you want the decrypted file sent to another file, called NOSECRET.XXX, type:

```
CIPHER SECRET  <MYSTERY.NEW >NOSECRET.XXX
```

NOTE: You must supply the same keyword that you encrypted the file with when you decrypt the file, or the CIPHER command will not work.

If you omit the less-than sign (<), CIPHER takes input from the keyboard and outputs to the screen. The file name is ignored. If you omit the greater-than sign (>), the encrypted file is not sent to another file but to the screeen. You can terminate either of these actions by pressing CONTROL-Z or CONTROL-C.

CLS

NAME

CLS

TYPE

Internal

PURPOSE

Clears the terminal screen.

SYNTAX

CLS

COMMENTS

The CLS command causes MS-DOS to send the ANSI escape sequence ESC[2J (that clears the screen) to the console.

CONFIG

NAME

CONFIG

TYPE

External

PURPOSE

Defines and modifies (temporarily or permanently) configuration information to MS-DOS.

SYNTAX

CONFIG

COMMENTS

Use this command to define your processing environment to MS-DOS: the type of printer and disks, any programmable function keys, and the number of retries to be performed on disk read and writes.

MS-DOS is initially set up with specific parameters. The following table shows these parameters and the changes that you can make with CONFIG.

	Initial Definition	With CONFIG
Programmable Function Keys	none	up to 20
Printer	parallel	serial
Serial Printer Interface:		
– Stop Bits	1	1 1/2 or 2
– Parity	even	disabled or odd
– Character length	7 bits	5, 6, or 8
– Baud rate	9600	50-19200
Disk	2 flexible	1 flexible, 1-3 fixed
Retry/Restore Counters:		
– Flexible	5, 5	1-9, 1-9
– Fixed	5, 5	1-9, 1-9

CONFIG is made up of a series of lead-through screens. To begin, type CONFIG and you see the main function screen.

```
CONFIG
      . . .
                CONFIG UTILITY

1)  Modify Function Keys
2)  Select Printer (Serial/Parallel)
3)  Modify Retry/Restore Counter
4)  Modify Serial Printer Interface
5)  Modify Disk Configuration
6)  Exit Program

*   Enter function
```

After you select the function, further screens guide you in defining your configuration. Although the screens are self-explanatory, some usage conventions should be noted.

None of the 20 programmable function keys are predefined. A single definition can be approximately 255 characters long (the exact length depends on the characters used). The definition may specify any function, but cannot include another function key (no characters in the range of 80-FF are accepted).

Function key definitions are placed in a table that can hold up to 492 characters. If more are entered a message is displayed. When you continue processing, the input definition of the key that caused the overflow is deleted, but any original contents is not.

The function keys are a convenience feature. An often-used function, for example, can be assigned to a function key and then initiated simply by pressing the key.

Assume you usually begin processing by displaying the directory on your system disk. You could assign the command DIR A: (plus the <CR> return function) to function key 1 with CONFIG. Request function 1 from the main screen, function 2 from the function key screen, and then press F1 in response to the Enter Function Key message. You will see:

```
    Function 01:
```

Now, enter the command, including the <CR> function.

```
    Function 01: DIR A:<CR>
```

NOTE: The control character keys with hexadecimal values from 00 through 1F are displayed between the symbols < >.

Check your definition and press the function key. (The definition always begins and ends by pressing the function key.) The key is now programmed to do a directory display. If you want to check the assignment, request the 'display definition' function.

As already discussed in Chapter 2, CONFIG must be used to define the disk system unless two flexible disks are being used. Modifications to the disk configuration parameters must be specified as "permanent," written to the operating system disk, and must be followed by a system restart. (A restart simply means turning off and on the computer.) The restart initializes the disk drives.

The Exit Program function may be used after each configuration function is performed or after all functions are completed. When requested, Exit Program displays three options.

```
    1) Update O.S. disk in drive A
    2) Return to main program
    3) Exit CONFIG
```

ATTENTION: Changes to the disk configuration must be written to disk (permanent) and must be followed by a restart. Update the disk, exit CONFIG, and then turn off and on the computer when the system prompt appears.

```
    *  Enter function
```

Function 1 is used to have the new configuration parameters written to disk. If the modifications are only temporary (for a specific run, for example), use function 3; the changes are only made in memory.

If you are making permanent changes and want to also update all other copies of your operating system disk, just insert another disk in drive A and request function 1. You can repeat this procedure and update all MS-DOS operating system disks.

You seldom have errors when using CONFIG, but you may – if your operating system disk is nearing its capacity. During the last phase of a "modify disk configuration" function, CONFIG writes the changes to disk in a file called CONFIG.SYS. If the directory or the file area itself is full, you will see either of two messages:

```
DIRECTORY FULL
DELETE A FILE FROM YOUR O.S. DISK; THEN REPEAT
THIS FUNCTION.
```

or

```
DISK FULL
DELETE OR SHORTEN A FILE FROM YOUR O.S. DISK;
THEN REPEAT THIS FUNCTION.
```

To correct the problem, simply delete or shorten a non-essential file (a scratch or backup file, perhaps); then, request CONFIG again, go directly to the Exit Program function, and specify function 1. The configuration changes are written to disk.

COPY

NAME

COPY

TYPE

Internal

PURPOSE

Copies one or more files to another disk. If you prefer, you can give the copies different names. This command can also copy files on the same disk.

SYNTAX

```
COPY <filespec> [filespec] [pathname] [pathname] [/V]
```

COMMENTS

Before using this command, be sure the destination disk contains sufficient space for the copy.

If the second filespec option is not given, the copy is to the default drive and has the same name as the original file (first filespec option). If the first filespec is on the default drive and the second filespec is not specified, the COPY is aborted (copying files to themselves is not allowed) and MS-DOS returns the error message:

```
File cannot be copied onto itself
0 File(s) copied
```

NOTE: You cannot copy a file on flexible disk to another flexible disk using a single flexible disk drive. To copy selected files, you must first copy the files to the fixed disk and then to another flexible disk.

The second option may take three forms:

1. If the second option is a drive designation (d:) only, the original file is copied with the original filename, to the designated drive.

2. If the second option is a filename only, the original file is copied to a file on the default drive with the filename specified.
3. If the second option is a full filespec, the original file is copied to a file on the default drive with the filename specified.

The /V switch causes MS-DOS to verify that the sectors written on the destination disk are recorded properly. Although there are rarely recording errors when you run COPY, you can verify that critical data has been correctly recorded. This option causes the COPY command to run more slowly because MS-DOS must check each entry recorded on the disk.

The COPY command also allows file concatenation (joining) while copying. Concatenation is accomplished by simply listing any number of files as options to COPY, separated by "+."

For example,

```
COPY A.XYZ + B.COM + B:C.TXT BIGFILE.CRP
```

This command concatenates files named A.XYZ, B.COM, and B:C.TXT and places them in the file on the default drive called BIGFILE.CRP.

To combine several files using wild cards into one file, you could type:

```
COPY *.LST COMBIN.PRN
```

This command would take all files with a filename extension of .LST and combine them into a file named COMBIN.PRN.

In the following example, for each file found matching *.LST, that file is combined with the corresponding .REF file. The result is a file with the same filename but with the extension .PRN. Thus, FILE1.LST will be combined with FILE1.REF to form FILE1.PRN; then XYZ.LST with XYZ.REF to form XYZ.PRN; and so on.

```
COPY *.LST + *.REF *.PRN
```

The following COPY command combines all files matching *.LST, then all files matching *.REF, into one file named COMBIN.PRN:

```
COPY *.LST + *.REF COMBIN.PRN
```

Do not enter a concatenation COPY command where one of the source filenames has the same extension as the destination. For example, the following command is an error if ALL.LST already exists:

```
COPY *.LST ALL.LST
```

The error would not be detected, however, until ALL.LST is appended. At this point it could have already been destroyed.

COPY compares the filenames of the input file with the filename of the destination. If they are the same, that one input file is skipped, and the error message "Content of destination lost before copy" is printed. Further concatenation proceeds normally. This allows "summing" files, as in this example:

```
COPY ALL.LST + *.LST
```

This command appends all *.LST files, except ALL.LST itself, to ALL.LST. This command does not produce an error message and is the correct way to append files using the COPY command.

Combining and copying files is normally performed in ASCII mode. This means that the system interprets the first CONTROL-Z character in the file as an end-of-file mark. You can, however, combine binary files or binary and ASCII files. Consider the following command line where:

/A means ASCII (default)

/B means binary

```
[/A] [/B] <filespec>[filespec] [pathname] [/A] [/B] [/V]
```

If you want to combine or copy binary files, use the /B argument at the beginning of the command line. In the following example, /B tells COPY that files MASM.ABC and MASM.DEF are both binary files. The /A or /B at the beginning applies to all subsequent files in the command line until another /A or /B is found.

THIS PAGE INTENTIONALLY LEFT BLANK

```
    COPY/B MASM.ABC+MASM.DEF MASM.EXE
```

If you want to combine ASCII and binary files, use the /A or /B argument after the filename in the command line. In the following example, the ASCII file A.XYZ, the binary file B.BIN, and the ASCII file C.TXT are combined into BIGFILE.CRP. Again, the /A or /B applies to all subsequent files in the command line until another /A or /B is found.

```
    COPY A.XYZ+B.BIN/B+B:C.TXT/A BIGFILE.CRP
```

A /A on the resulting file causes a CONTROL-Z to be added as the last character in that file. A /B on the resulting file means no CONTROL-Z character is added.

NOTE: Binary files output by development tools, e.g., macro-assembler locate, are recognized as such by the system provided you define the extension without wild card characters. If you combine or copy these files, you do not need to include the /B argument.

CTTY

NAME

CTTY

TYPE

Internal

PURPOSE

Allows you to change the device from which you issue commands (TTY represents the console).

SYNTAX

```
CTTY \DEV\DEV
```

COMMENTS

DEV stands for "device", which is the device from which you are giving commands to MS-DOS. This command is useful if you want to change the device on which you are working. The command

```
    CTTY \DEV\AUX
```

moves all command I/O (input/output) from the current device (the console) to the AUX port, such as a printer. The command

```
    CTTY \DEV\CON
```

moves I/O back to the original device (here, the console). Refer to "Illegal Filenames" in Chapter 3 for a list of valid device names to use with the CTTY command.

DATE

NAME

DATE

TYPE

Internal

PURPOSE

Enter or change the date known to the system. This date is recorded in the directory for any files you create or alter.

You can change the date from your terminal or from a batch file. (MS-DOS does not display a prompt for the date if you use an AUTOEXEC.BAT file, so you may want to include a DATE command in that file.)

SYNTAX

```
DATE [<mm>-<dd>-<yy>]
```

COMMENTS

If you type DATE, DATE responds with the message:

```
    Current date is<mm>-<dd>-<yy>
    Enter new date:_
```

Press<CR> if you do not want to change the date shown.

You can also type a particular date after the DATE command, as in:

```
    DATE 5-9-83
```

In this case, you do not have to answer the "Enter new date:" prompt.

The new date must be entered using numerals only; letters are not permitted. The allowed options are:

<mm> = 1-12
<dd> = 1-31
<yy> = 80-99 or 1980-2099

The date, month, and year entries may be separated by hyphens (-) or slashes (/).

If the options or separators are not valid, DATE displays the message:

```
Invalid date
Enter new date:_
```

DATE then waits for you to enter a valid date.

DEL

NAME

DEL (DELETE)

TYPE

Internal

SYNONYM

ERASE

PURPOSE

Deletes all files with the designated filespec.

SYNTAX

DEL [filespec] [pathname]

COMMENTS

If the filespec is *.*, the prompt "Are you sure?" appears. If a Y or y is typed as a response, then all files are deleted as requested. You can also type ERASE for the DELETE command.

DIR

NAME

DIR (DIRECTORY)

TYPE

Internal

SYNTAX

DIR [filespec] [pathname] [/P] [/W]

PURPOSE

Lists the files in a directory.

COMMENTS

If you just type DIR, all directory entries on the default drive are listed. If only the drive specification is given (DIR d:), all entries on the disk in the specified drive are listed. If only a filename is entered with no extension (DIR filename), then all files with the designated filename on the disk in the default drive are listed. If you designate a file specification (for example, DIR d:filename.ext), all files with the filename specified on the disk in the drive specified are listed. In all cases, files are listed with their size in bytes and with the time and date of their last modification.

The wild card characters ? and * (question mark and asterisk) may be used in the filename option. As examples, the following table shows equivalent command designations.

COMMAND	EQUIVALENT
DIR	DIR *.*
DIR FILENAME	DIR FILENAME.*
DIR .EXT	DIR *.EXT
DIR .	DIR *.

Two switches may be specified with DIR. The /P switch selects Page Mode. With /P, display of the directory pauses after the screen is filled. To resume display of output, press any key.

The /W switch selects Wide Display. With /W, only filenames are displayed, without other file information. Files are displayed five per line.

DISKCOPY

NAME

DISKCOPY

TYPE

External

PURPOSE

Copies the contents of the disk in the source drive to the disk in the destination drive.

SYNTAX

```
DISKCOPY [d:] [d:]
```

COMMENTS

The first option you specify is the source drive; the second option is the destination drive.

The disk in the destination drive must be formatted (by the same operating system and in the same format as the source disk) before using DISKCOPY.

You can specify the same drives or you may specify different drives. If the drives designated are the same, a single-drive copy operation is performed. You are prompted to insert the disks at the appropriate times. DISKCOPY waits for you to press any key before continuing.

After copying, DISKCOPY prompts:

```
    Copy another (Y/N)?_
```

If you press Y, the next copy is performed on the same drives that you originally specified, after you have been prompted to insert the proper disks.

To end the COPY, press N.

Before using DISKCOPY, also consider the following command characteristics:

- If you omit both options, a single-drive copy operation is performed on the default drive.
- If you omit the second option, the default drive is used as the destination drive.
- Both disks must have the same number of physical sectors and those sectors must be the same size.
- Disks that have had a lot of file creation and deletion activity become fragmented, because disk space is not allocated sequentially. The first free sector found is the next sector allocated, regardless of its location on the disk.

  A fragmented disk can cause poor performance due to delays involved in finding, reading, or writing a file. If this is the case, you must use the COPY command, instead of DISKCOPY, to copy your disk and eliminate the fragmentation.

  For example:

  ```
  COPY A:*.* B:
  ```

  copies all files from the disk in drive A to the disk in drive B.
- DISKCOPY automatically determines the number of sides to copy, based on the source drive and disk.
- If disk errors are encountered during a DISKCOPY, MS-DOS displays:

  ```
  DISK error while reading drive A
  Abort, Ignore, Retry?
  ```

Refer to Appendix B, Disk Errors, for information on this error message.

ECHO

NAME

ECHO

TYPE

Internal; Batch processing

PURPOSE

Turns batch echo feature on and off.

SYNTAX

```
ECHO [ON message]
ECHO [OFF message]
```

COMMENTS

Normally, commands in a batch file are displayed ("echoed") on the console when they are seen by the command processor. ECHO OFF turns off this feature. ECHO ON turns the echo back on.

If ON or OFF are not specified, the current setting is displayed.

EXIT

NAME

EXIT

TYPE

Internal

PURPOSE

Exits the program COMMAND.COM (the command processor) and returns to a previous level, if one exists.

SYNTAX

```
EXIT
```

COMMENTS

This command can be used when you are running an application program and want to start the MS-DOS command processor, then return to your program. For example, to look at a directory on drive B while running an application program, you must start the command processor by typing COMMAND in response to the default drive prompt:

```
    A>COMMAND
```

You can now type the DIR command and MS-DOS displays the directory for the default disk. When you type EXIT, you return to the previous level (your application program).

FIND

NAME

FIND

TYPE

External

PURPOSE

Searches for a specific string of text in a file or files.

SYNTAX

```
FIND [/V /C /N] <string> [<filename. . .>]
```

COMMENTS

FIND is a filter that takes as options a string and a series of filenames. It displays all lines that contain a specified string from the files specified in the command line.

If no files are specified, FIND takes the input on the screen and displays all lines that contain the specified string.

These switches can be used with FIND:

/V

This switch causes FIND to display all lines not containing the specified string.

/C

This switch causes FIND to display only the count of lines that contained a match in each of the files.

/N

This switch causes each line to be preceded by its relative line number in the file.

The string should be enclosed in quotes. For example,

```
FIND “Fool’s Paradise” BOOK1.TXT BOOK2.TXT
```

displays all lines from BOOK1.TXT and BOOK2.TXT (in that order) that contain the string “Fool’s Paradise.” The command

```
DIR B: | FIND /V "DAT"
```

causes MS-DOS to display all names of the files on the disk in drive B that do not contain the string DAT. Type double quotes around a string that already has quotes in it.

When an error is detected, FIND responds with one of the following error messages:

Incorrect DOS version
FIND only runs on versions of MS-DOS that are 2.0 or higher.

FIND: Invalid number of parameters
You did not specify a string when issuing the FIND command.

FIND: Syntax error
You typed an illegal string when issuing the FIND command.

FIND: File not found<filename>
The filename you have specified does not exist or FIND cannot find it.

FIND: Read error in<filename>
An error occurred when FIND tried to read the file specified in the command.

FIND: Invalid parameter<option-name>
You specified an option that does not exist.

FOR

NAME

FOR

TYPE

Internal; Batch processing

PURPOSE

Command extension used in batch and interactive file processing.

SYNTAX

- For batch processing:

  ```
  FOR %%<c> IN <set> DO <command>
  ```

- For interactive processing:

  ```
  FOR %<c> IN <set> DO <command>
  ```

COMMENTS

<c> can be any character except 0, 1, 2, 3, . . , 9 to avoid confusion with the %0-%9 batch parameters.

<set> is (<item>. . .)

The %%<c> variable is set sequentially to each member of <set>, and then <command> is evaluated. If a member of <set> is an expression involving * and/or ?, then the variable is set to each matching pattern from disk. In this case, only one such <item> may be in the set, and any <item> besides the first is ignored.

NOTE: The words IN, FOR, and DO must be in uppercase.

Consider these examples:

```
FOR %%f IN ( *.ASM ) DO MASM %%f;
FOR %%f IN (FOO BAR BLECH) DO REM %%f
```

The ‘%%’ is needed so that after batch parameter (%0-%9) processing is done, there is one ‘%’ left. If only ‘%f’ were there, the batch parameter processor would see the ‘%’, look at ‘f’, decide that ‘%f’ was an error (bad parameter reference) and throw out the ‘%f’, so that the command FOR would never see it. If the FOR is not in a batch file, then only one ‘%’ should be used.

FORMAT

NAME

FORMAT

TYPE

External

PURPOSE

Formats the disk in the specified drive to accept MS-DOS files.

SYNTAX

```
FORMAT [d:] [/V /J /D /I /O /S]   (flexible disks)
FORMAT [d:] [/V]                  (fixed disks)
```

COMMENTS

This command formats the disk and initializes the directory and file allocation tables. If no drive is specified, the disk in the default drive is formatted. All disks are formatted at double density, double sided and either at 9 sectors per track for a flexible disk or 17 sectors per track for a fixed disk.

When formatting a fixed disk, FORMAT displays a message asking for the number of certifications (read-after-write checks on each track). The default value is 5; increasing the number will significantly increase the time for formatting.

Six switches control options that can be requested when formatting a flexible disk; only the /V switch is valid when formatting a fixed disk. Of the switches, two are used more frequently than the others.

/V

Causes FORMAT to pause in the formatting process and display a message asking for a volume label (useful in disk identification).

/S

Causes FORMAT to copy the operating system files from the disk in the default drive to the newly formatted disk. The files are copied in the following order: IO.SYS, MSDOS.SYS, COMMAND.COM. (Other files can then be selectively copied with the COPY command.) When used with other switches, /S must be entered last.

NOTE: To copy an entire system disk, format with *no switches* and then use DISKCOPY. DISKCOPY produces a "mirror image" and writes over any label.

The next group of switches allows some other format to be created on a flexible disk. (These formats may be needed if you want to copy and use information from a non-NCR format disk.) If no switch is specified, the default format is assumed: 9 sectors per track; double sided, double density (360 KB disk capacity).

/J

Formats at 9 sectors per track; single sided, double density (180KB disk capacity).

/D

Formats at 8 sectors per track; double sided, double density (320KB disk capacity).

/I

Formats at 8 sectors per track; single sided, double density (160KB disk capacity).

The /O switch generates an E5 character in the first position of an empty (available) directory entry. Use this switch only if you need to maintain compatibility with older versions of MS-DOS; the current standard entry to indicate empty directory entries is 00.

GOTO

NAME

GOTO

TYPE

Internal; Batch processing

PURPOSE

Command extension used in batch file processing.

SYNTAX

GOTO <label>

COMMENTS

GOTO causes commands to be taken from the batch file beginning with the line after the <label> definition. If no label has been defined, the current batch file terminates.

For example:

```
    :foo
    REM looping . . .
    GOTO foo
```

produces an infinite sequence of messages: REM looping . . . .

Starting a line in a batch file with ':' causes the line to be ignored by batch processing. The characters following GOTO define a label, but this procedure may also be used to put in comment lines.

IF

NAME

IF

TYPE

Internal; Batch processing

PURPOSE

Command extension used in batch file processing.

SYNTAX

IF <condition> <command>

COMMENTS

The parameter <condition> is one of the following:

ERRORLEVEL <number>
True if and only if the previous program executed by COMMAND had an exit code of <number> or higher.

<string1> == <string2>
True if and only if <string1> and <string2> are identical after parameter substitution. Strings may not have embedded separators.

EXIST <filename>
True if and only if <filename> exists.

NOT <condition>
True if and only if <condition> is false.

The IF statement allows conditional execution of commands. When the <condition> is true, then the <command> is executed. Otherwise, the <command> is ignored.

NOTE: The words ERRORLEVEL, EXIST, and NOT must be uppercase.

Consider the following examples:

```
IF NOT EXIST \TMP\FOO ECHO Can't find file

IF NOT ERRORLEVEL 3 LINK $1, , ;
```

LOCATE

NAME

LOCATE

TYPE

External

PURPOSE

Converts .EXE (executable) files to binary format. This results in a saving of disk space and faster program loading.

SYNTAX

```
LOCATE <filespec> [d:] [<filename>[<.ext>] ]
```

COMMENTS

This command is useful only if you want to convert .EXE files to binary format. The file named by filespec is the input file. If no extension is specified, it defaults to .EXE. The input file is converted to .COM file format (memory image of the program) and placed in the output file. If you do not specify a drive, the drive of the input file is used. If you do not specify an output filename, the input filename is used. If you do not specify a filename extension in the output filename, the new file is given an extension of .BIN.

The input file must be in valid .EXE format produced by the linker. The resident, or actual code and data part of the file must be less than 64K. There must be no STACK segment.

Two kinds of conversions are possible, depending on whether the initial CS:IP (Code Segment:Instruction Pointer) is specified in the .EXE file.

1. If CS:IP is not specified in the .EXE file, a pure binary conversion is assumed. If segment fixups are necessary (that is, the program contains instructions requiring segment relocation), you are prompted for the fixup value. This value is the absolute segment at which the program is to be loaded. The resulting program is usable only when loaded at the absolute

memory address specified by a user application. The command processor will not be capable of properly loading the program.

2. If CS:IP is specified as 0000:100H, it is assumed that the file is to be run as a .COM file with the location pointer set at 100H by the assembler statement ORG; the first 100H bytes of the file are deleted. No segment fixups are allowed, as .COM files must be segment relocatable; that is, they must assume the entry conditions explained in the *PROGRAMMER'S MANUAL.* Once the conversion is complete, you may rename the resulting file with a .COM extension. Then the command processor is able to load and execute the program in the same way as the .COM programs supplied on your MS-DOS disk.

If CS:IP does not meet either of these criteria, or if it meets the .COM file criterion but has segment fixups, the following message is displayed:

```
File cannot be converted
```

This message is also displayed if the file is not a valid executable file.

If LOCATE finds an error, one or more of the following error messages is displayed:

File not found
The file is not on the disk specified.

Insufficient memory
There is not enough memory to run LOCATE.

File creation error
LOCATE cannot create the output file. Run CHKDSK to determine if the directory is full, or if some other condition caused the error.

Insufficient disk space
There is not enough disk space to create a new file.

Fixups needed - base segment (hex):
The source (.EXE) file contained information indicating that a load segment is required for the file. Specify the absolute segment address at which the finished module is to be located.

File cannot be converted
    The input file is not in the correct format.

WARNING - Read error on .EXE file.
Amount read less than size in header
    This is a warning message only.

MKDIR

NAME

MKDIR

TYPE

Internal

SYNONYM

MD

PURPOSE

Makes a new directory.

SYNTAX

MKDIR < pathname>

COMMENTS

This command is used to create a hierarchical directory structure. When you are in your root directory, you can create subdirectories by using the MKDIR command. The command

```
    MKDIR \USER
```

creates a subdirectory \USER in your root directory. To create a directory named JOE under \USER, type:

```
    MKDIR \USER\JOE
```

MORE

NAME

MORE

TYPE

External

PURPOSE

Sends output to console one screen at a time.

SYNTAX

MORE

COMMENTS

MORE is a filter that reads from standard input (such as a command from your terminal) and displays one screen of information at a time. The MORE command then pauses and displays the - -MORE- - message at the bottom of your screen.

Pressing the <CR> key displays another screen of information. This process continues until all the input data has been read.

The MORE command is useful for viewing a long file one screen at a time. If you type

```
    TYPE MYFILES.COM  |  MORE
```

MS-DOS displays the file MYFILES.COM (on the default drive) one screen at a time.

PATH

NAME

PATH

TYPE

Internal

PURPOSE

Sets a command path.

SYNTAX

```
PATH [<pathname>[;<pathname>] . . . ]
```

COMMENTS

This command allows you to tell MS-DOS which directories to search for external commands after MS-DOS searches your working directory. The default value is \BIN, where \BIN is the name of the directory in which all MS-DOS external commands reside.

To tell MS-DOS to search your \BIN\USER\JOE directory for external commands (in addition to a search of the \BIN directory), type:

```
PATH \BIN\USER\JOE
```

MS-DOS now also searches the \BIN\USER\JOE directory for external commands until you set another path or shut down MS-DOS.

You can tell MS-DOS to search more than one path by specifying several pathnames separated by semicolons. For example,

```
PATH \BIN\USER\JOE;\BIN\USER\SUE;\BIN\DEV
```

tells MS-DOS to search the directories specified by the above pathnames to find external commands. MS-DOS searches the pathnames in the order specified in the PATH command.

The command PATH with no options prints the current path. If you specify PATH;, MS-DOS sets the NUL path, meaning that only the working directory is searched for external commands.

PAUSE

NAME

PAUSE

TYPE

Internal

PURPOSE

Suspends execution of the batch file.

SYNTAX

PAUSE [comment]

COMMENTS

During the execution of a batch file, you may need to change disks or perform some other action. PAUSE suspends execution until you press any key, except <CONTROL-C>.

When the command processor encounters PAUSE, it prints:

```
Strike a key when ready . . .
```

If you press <CONTROL-C>, another prompt will be displayed:

```
Abort batch job (Y/N)?
```

If you type Y in response to this prompt, execution of the remainder of the batch command file is aborted and control is returned to the operating system command level. Therefore, PAUSE is used to break a batch file into pieces, allowing you to end the batch command file at an intermediate point.

The comment is optional and is entered on the same line as PAUSE. You may want to prompt the user of the batch file with some meaningful message when the batch file pauses. For example, you may want to change disks in one of the drives. An optional prompt message may be given in such cases. The comment prompt is displayed before the "Strike a key" message.

PRINT

NAME

PRINT

TYPE

External

PURPOSE

Prints a text file on a line printer while you are processing other MS-DOS commands (usually called "background printing").

SYNTAX

```
PRINT [ [filespec] [/T] [/C] [/P] ] . . .
```

COMMENTS

You use the PRINT command only if you have a line printer attached to your computer. The following switches are provided with this command:

/T - TERMINATE

This switch deletes all files in the print queue (those waiting to be printed). A message to this effect is printed.

/C - CANCEL

This switch turns on cancel mode. The preceding filespec and all following filespecs are suspended in the print queue until you type a /P switch.

/P - PRINT

This switch turns on print mode. The preceding filespec and all following filespecs are added to the print queue until you issue a /C switch.

PRINT with no options displays the contents of the print queue on your screen without affecting the queue.

Consider the following examples:

```
PRINT /T
```

empties the print queue.

```
PRINT /T *.ASM
```

empties the print queue and queues all .ASM files on the default drive.

```
PRINT A:TEMP1.TST/C A:TEMP2.TST A:TEMP3.TST
```

removes the three files indicated from the print queue.

```
PRINT TEMP1.TST /C TEMP2.TST /P TEMP3.TST
```

removes TEMP1.TST from the queue, and adds TEMP2.TST and TEMP3.TST to the queue.

If an error is detected, PRINT displays one of the following error messages:

Name of list device [PRN:]
: This prompt appears when PRINT is run the first time. Any current device may be specified and that device then becomes the PRINT output device. As indicated in the [ ], simply pressing<CR> results in the device PRN being used.

List output is not assigned to a device
: This message is displayed if the "Name of list device" specified to the preceding prompt is invalid. Subsequent attempts return the same message until a valid device is specified.

PRINT queue is full
: There is room for 10 files in the queue. If you attempt to put more than 10 files in the queue, this message appears on the console.

PRINT queue is empty
: There are no files in the print queue.

No files match d:XXXXXXXX.XXX
: A filespec was given for files to add to the queue, but no files match a specification. (If there are no files in the queue to match a cancelled filespec, no error message appears.)

Drive not ready

If this message occurs when PRINT attempts a disk access, PRINT keeps trying until the drive is ready. Any other error causes the current file to be cancelled. In such a case, an error message is output to your printer.

All files cancelled

If the /T (TERMINATE) switch is issued, the message "All files cancelled by operator" is output on your printer. If the current file being printed is cancelled by a /C, the message "File cancelled by operator" is printed.

PROMPT

NAME

PROMPT

TYPE

Internal

PURPOSE

Changes the MS-DOS command prompt.

SYNTAX

PROMPT [<prompt-text>]

COMMENTS

This command allows you to change the MS-DOS system prompt. If no text is typed, the prompt is set to the default prompt, which is the default drive designation. You can set the prompt to a special prompt by using the characters indicated below.

The following characters can be used in the prompt command to specify special prompts. They must all be preceded by a dollar sign ($) in the prompt command:

<table>
<tr><th>Specify This Character</th><th>To Get This Prompt:</th></tr>
<tr><td>$</td><td>— The '$' character</td></tr>
<tr><td>t</td><td>— The current time</td></tr>
<tr><td>d</td><td>— The current date</td></tr>
<tr><td>p</td><td>— The current directory of the default drive</td></tr>
<tr><td>v</td><td>— The version number</td></tr>
<tr><td>n</td><td>— The default drive</td></tr>
<tr><td>g</td><td>— The '>' character</td></tr>
<tr><td>l</td><td>— The '<' character</td></tr>
<tr><td>b</td><td>— The '|' character</td></tr>
<tr><td>_</td><td>— A CR LF sequence</td></tr>
<tr><td>s</td><td>— A space</td></tr>
<tr><td>h</td><td>— A backspace</td></tr>
<tr><td>e</td><td>— ASCII code X'1B' (escape)</td></tr>
</table>

Consider the following example:

```
PROMPT $n:
```

Sets the prompt to the default drive followed by a colon.

You can also use escape sequences in your prompts. For example:

```
PROMPT $e[7m$n$g$e[m
```

Sets the prompts in inverse video mode and returns to video mode for other text.

RDCPM

NAME

RDCPM

TYPE

External

PURPOSE

Transfers NCR CP/M® files to an MS-DOS formatted disk.

SYNTAX

```
RDCPM DIR d:                (displays directory on CP/M disk)
RDCPM d: filename [d:]      (transfers CP/M file to MS-DOS disk)
```

COMMENTS

RDCPM reads the file from an NCR CP/M formatted disk and transfers it to an MS-DOS formatted disk. Once transferred, the file is an MS-DOS file.

The DIR variation of RDCPM displays the directory of the CP/M disk, so you can see the names of the files. The drive designation of the CP/M disk must be specified.

To transfer the file, you must specify the drive designation of the CP/M disk and the filename. The wild-card characters (* and ?) may be used to transfer several files. (See Chapter 3, More about Files, for a description of naming files with wild cards.) The destination drive designation is optional; and, if not specified, the disk in the default drive is assumed to be the destination disk.

Consider the following examples:

```
A>RDCPM C: MYFILE.TXT B:
A>RDCPM C: MYFILE.TXT
```

The first command transfers MYFILE.TXT from the disk in drive C to the disk in drive B; the second command transfers the same file to the disk in drive A, the default drive.

NOTE: The source and destination drive designations must be different. Therefore, if you have a single flexible disk drive and want to transfer a CP/M file from a flexible disk, you must first copy the file (with the CP/M operating system) to another disk drive. Then, use the RDCPM commana.

The following messages may be displayed; most are self-explanatory.

Hard disk error on CP/M drive
The disk specified by the source drive designation may not be a CP/M disk.

Source and destination drives must not be the same
The CP/M and MS-DOS disks must be on different drives.

Drive not available for CP/M reading
MS-DOS cannot access the specified drive. This error occurs if your MS-DOS disk configuration is not correct. For example, you specified

```
RDCPM C: YOURFILE B:
```

but the system is configured for a 2-flexible disk system with drives A and B. (No fixed disk was defined.) If the configuration definition is the cause of the error, use the CONFIG utility to modify the definition and then run RDCPM again.

Insufficient disk space
The MS-DOS destination disk does not have enough space for the CP/M file(s).

No room in directory to create file
The directory on the MS-DOS disk has no space to create an entry for the CP/M file(s).

Source file name missing
The specified file is not on the NCR CP/M disk. Check that the filename was entered correctly.

Source file not found
The specified file is not on the NCR CP/M disk. Check that the filename was entered correctly.

File transfer complete
The specified file(s) was successfully transferred.

RECOVER

NAME

RECOVER

TYPE

External

PURPOSE

Recovers a file or an entire disk containing bad sectors.

SYNTAX

```
RECOVER <filename>
RECOVER <d:>
```

COMMENTS

If a sector on a disk is bad, you can recover either the file containing that sector (without the bad sector) or the entire disk (if the bad sector was in the directory).

To recover a particular file, type:

```
    RECOVER <filename>
```

This causes MS-DOS to read the file sector by sector and to skip the bad sector(s). When MS-DOS finds the bad sector(s), the sector(s) are marked and MS-DOS no longer allocates your data to that sector.

To recover a disk, type:

```
    RECOVER <d:>
```

where d: is the letter of the drive containing the disk to be recovered.

If there is not enough room in the root directory, RECOVER prints a message and stores information about the extra files in the File Allocation Table. You can run RECOVER again to regain these files when there is more room in the root directory.

REM

NAME

REM (REMARK)

TYPE

Internal

PURPOSE

Displays remarks that are on the same line as the REM command in a batch file during execution of that batch file.

SYNTAX

REM [comment]

COMMENTS

The only separators allowed in the comment are the space, tab, and comma. Consider the following example:

```
1: REM  This file checks new disks
2: REM  It is named NEWDISK.BAT
3: PAUSE  Insert new disk in drive B:
4: FORMAT B:/S
5: DIR B:
6: CHKDSK B:
```

REN

NAME

REN (RENAME)

TYPE

Internal

SYNONYM

RENAME

PURPOSE

Changes the name of the first option (filespec) to the second option (filename).

SYNTAX

REN <filespec> <filename>

COMMENTS

The first option (filespec) must be given a drive designation if the disk resides in a drive other than the default drive. Any drive designation for the second option (filename) is ignored. The file remains on the disk where it currently resides.

The wild card characters may be used in either option. All files matching the first filespec are renamed. If wild card characters appear in the second filename, corresponding character positions are not changed.

For example, the following command changes the names of all files with the .LST extension to similar names with the .PRN extension:

```
REN *.LST *.PRN
```

In the next example, REN renames the file ABODE on drive B to ADOBE:

```
REN B:ABODE ?D?B?
```

The file remains on drive B.

An attempt to rename a filespec to a name already present in the directory results in the error message "File not found."

RMDIR

NAME

RMDIR (REMOVE DIRECTORY)

TYPE

Internal

SYNONYM

RD

PURPOSE

Removes a directory from a hierarchical directory structure.

SYNTAX

RMDIR [d:] <pathname>

COMMENTS

This command removes a directory *that is empty* except for the . and . . shorthand symbols.

To remove the \BIN\USER\JOE directory, first issue a DIR command for that path to ensure that the directory does not contain any important files that you do not want deleted. Then type:

```
    RMDIR \BIN\USER\JOE
```

The directory is deleted from the directory structure.

SET

NAME

SET

TYPE

Internal

PURPOSE

Sets one string value equivalent to another string for use in later programs.

SYNTAX

SET [<string = string>]

COMMENTS

This command is meaningful only if you want to set values that will be used by programs you have written. An application program can check all values that have been set with the SET command by issuing SET with no options. For example, SET TTY = VT52 sets your TTY value to VT52 until you change it with another SET command.

The SET command can also be used in batch processing. In this way, you can define your replaceable parameters with names instead of numbers. If your batch file contains the statement "LINK %FILE%", you can set the name that MS-DOS will use for that variable with the SET command. The command SET FILE = DOMORE replaces the %FILE% parameter with the filename DOMORE. Therefore, you do not need to edit each batch file to change the replaceable parameter names. Note that when you use text (instead of numbers) as replaceable parameters, the name must be ended by a percent sign.

SHIFT

NAME

SHIFT

TYPE

Internal; Batch processing

PURPOSE

Allows access to more than 10 replaceable parameters in batch file processing.

SYNTAX

SHIFT

COMMENTS

Usually, command files are limited to handling 10 parameters, %0 through %9. To allow access to more than ten parameters, use SHIFT to change the command line parameters. For example, if

```
%0 = "foo"
%1 = "bar"
%2 = "name"
%3 . . .%9 are empty
```

then a SHIFT results in the following:

```
%0 = "bar"
%1 = "name"
%2 . . .%9 are empty
```

If there are more than 10 parameters given on a command line, those that appear after the 10th (%9) are shifted one at a time into %9 by successive shifts.

SORT

NAME

SORT

TYPE

External

PURPOSE

SORT reads input from your terminal, sorts the data, then writes it to your terminal screen or files.

SYNTAX

```
SORT [/R]  [/+n]
```

COMMENTS

SORT can be used, for example, to alphabetize a file by a certain column. There are two switches that allow you to select options:

/R
: Reverses the sort; that is, sorts from Z to A.

/+n
: Sorts starting with column n where n is some number. If you do not specify this switch, SORT begins sorting from column 1.

Consider the following examples. In the first one, the command reads the file UNSORT.TXT, reverses the sort, and then writes the output to a file named SORT.TXT:

```
SORT /R <UNSORT.TXT >SORT.TXT
```

The next command pipes the output of the directory command to the SORT filter. The SORT filter sorts the directory listing starting with column 14 (this is the column in the directory listing that contains the file size), then sends the output to the console. Thus, the result of this command is a directory sorted by file size:

```
DIR | SORT /+14
```

The command

```
DIR | SORT /+14 | MORE
```

does the same thing as the command in the previous example, except that the MORE filter gives you a chance to read the sorted directory one screen at a time.

NOTE: A>SORT leads to a keyboard entry. This entry can be terminated only by CONTROL-Z.

SYS

NAME

SYS (SYSTEM)

TYPE

External

PURPOSE

Transfers the MS-DOS system files from the disk in the default drive to the disk in the drive specified by d:.

SYNTAX

```
SYS <d>:
```

COMMENTS

SYS is normally used to update the system or to place the system on a formatted disk that contains no files. An entry for d: is required.

If IO.SYS and MSDOS.SYS are on the destination disk, they must take up the same amount of space on the disk as the new system will need. This means that you cannot transfer system files from an MS-DOS 2.0 disk to an MS-DOS 1.1 disk. You must reformat the MS-DOS 1.1 disk with the MS-DOS FORMAT command before the SYS command will work.

The destination disk must be completely blank or already have the system files IO.SYS and MSDOS.SYS.

The transferred files are copied in the following order:

```
IO.SYS
MSDOS.SYS
```

IO.SYS and MSDOS.SYS are both hidden files that do not appear when the DIR command is executed. COMMAND.COM (the command processor) is not transferred. You must use the COPY command to transfer COMMAND.COM.

If SYS detects an error, one of the following messages will be displayed:

No room for system on destination disk
: There is not enough room on the destination disk for the IO.SYS and MSDOS.SYS files.

Incompatible system size
: The system files IO.SYS and MSDOS.SYS do not take up the same amount of space on the destination disk as the new system will need.

TIME

NAME

TIME

TYPE

Internal

PURPOSE

Displays and sets the time.

SYNTAX

```
TIME [<hh>[:<mm>] ]
```

COMMENTS

If the TIME command is entered without any arguments, the following message is displayed:

```
Current time is <hh>:<mm>:<ss>.<cc>
Enter new time: _
```

Press the <CR> key if you do not want to change the time shown. A new time may be given as an option to the TIME command as in:

```
TIME 8:20
```

The new time must be entered using numerals only; letters are not allowed. The allowed options are:

```
<hh>  = 00-24
<mm> = 00-59
```

The hour and minute entries must be separated by colons. You do not have to type the <ss> (seconds) or <cc> (hundredths of seconds) options.

MS-DOS uses the time entered as the new time if the options and separators are valid. If the options or separators are not valid, MS-DOS displays the message:

```
Invalid time
Enter new time: _
```

MS-DOS then waits for you to type a valid time.

TYPE

NAME

TYPE

TYPE

Internal

PURPOSE

Displays the contents of the file on the console screen.

SYNTAX

```
TYPE <filespec>
```

COMMENTS

Use this command to examine a file without modifying it. (Use DIR to find the name of a file and EDLIN to alter the contents of a file.) The only formatting performed by TYPE is that tabs are expanded to spaces consistent with tab stops every eighth column. Note that a display of binary files causes control characters (such as CONTROL-Z) to be sent to your computer, including bells, form feeds, and escape sequences.

VER

NAME

VER

TYPE

Internal

PURPOSE

Prints MS-DOS version number.

SYNTAX

VER

COMMENTS

If you want to know what version of MS-DOS you are using, type VER. The version number is displayed on your screen.

VERIFY

NAME

VERIFY

TYPE

Internal

PURPOSE

Turns the verify switch on or off when writing to disk.

SYNTAX

```
VERIFY [ON]
VERIFY [OFF]
```

COMMENTS

This command has the same purpose as the /V switch in the COPY command. If you want to verify that all files are written correctly to disk, you can use the VERIFY command to tell MS-DOS to verify that your files are intact (no bad sectors, for example). MS-DOS performs a VERIFY each time you write data to a disk. You receive an error message only if MS-DOS was unable to successfully write your data to disk.

VERIFY ON remains in effect until you change it in a program (by a SET VERIFY system call), or until you issue a VERIFY OFF command to MS-DOS.

If you want to know what the current setting of VERIFY is, type VERIFY with no options.

VOL

NAME

VOL (VOLUME)

TYPE

Internal

PURPOSE

Displays disk volume number, if it exists.

SYNTAX

VOL [d:]

COMMENTS

This command prints the volume ID of the disk in drive d:. If no drive is specified, MS-DOS prints the volume ID of the disk in the default drive.

MS-DOS EDITING AND FUNCTION KEYS

SPECIAL EDITING KEYS

The special editing keys deserve particular emphasis because they depart from the way in which most operating systems handle command input. You do not have to type the same sequences of keys repeatedly, because the last command line is automatically placed in a special storage area called a template.

By using the template and the special editing keys, you can take advantage of the following MS-DOS features:

- A command line can be instantly repeated by pressing two keys.
- If you make a mistake in the command line, you can edit it and retry without having to retype the entire command line.
- A command line that is similar to a preceding command line can be edited and executed with a minimum of typing by pressing a special editing key.

The relationship between the command line and the template is shown in Figure 6.1.

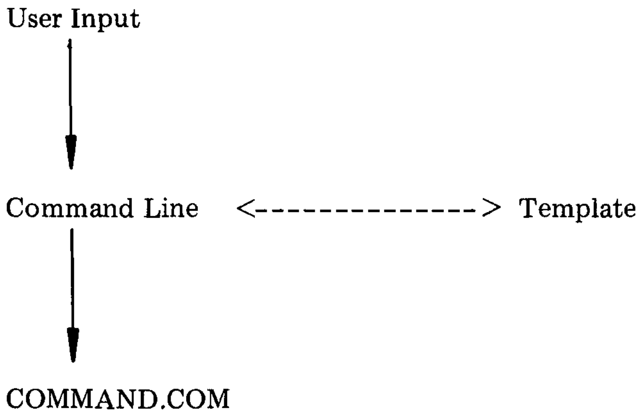


Figure 6.1 Command line and template

You type a command to MS-DOS on the command line. When you press the <CR> key, the command is automatically sent to the command processor (COMMAND.COM) for execution. At the same time, a copy of this command is sent to the template. You can now recall the command or modify it with MS-DOS special editing keys.

Table 6.2 contains a complete list of the special editing keys. Each of these keys is more fully described in Chapter 7, Line Editor (EDLIN), where they can be used to edit your text files.

Function	Key(s) *	Description
Copy one character	<COPY1> ESC S	Copies one character from the template to the command line.
Copy up to character	<COPYUP> ESC T	Copies characters up to the character specified in the template and puts these characters on the command line.
Copy template	<COPYALL> ESC U	Copies all remaining characters in the template to the command line.
Skip one character	<SKIP1> ESC V	Skips over (does not copy) a character in the template.
Skip up to character	<SKIPUP> ESC W	Skips over (does not copy) the characters in the template up to the character specified.
Quit input	<VOID> ESC E	Voids the current input; leaves the template unchanged.
Kill line	<KILL> ESC J	Voids line on template; current input sent to template.
Insert mode	<INSERT> ESC P	Enters insert mode.
Replace mode	<EXIT> ESC Q	Turns insert mode off; this is the default mode.
New template	<NEWLINE> ↵	Makes the new line the new template.

* Most functions require a 2-key entry. Do not press the keys simultaneously.

Table 6.2 Special editing functions

Notice in the table that an editing function (except for <NEWLINE>) is initiated with two keys. Press the ESC key first and then the editing key. Do not press the keys simultaneously.

Consider the following examples. In the examples, the name of the function key is used, not the actual key.

If you type the following command

```
DIR PROG.COM
```

MS-DOS displays information about the file PROG.COM on your screen. The command line is also saved in the template. To repeat the command, just use the editing keys: <COPYALL> and <CR>.

The repeated command is displayed on the screen as you type:

```
<COPYALL>DIR PROG.COM<CR>
```

Notice that pressing the <COPYALL> key causes the contents of the template to be copied to the command line; pressing <CR> causes the command line to be sent to the command processor for execution.

If you want to display information about a file named PROG.ASM, you can use the contents of the template and type:

```
<COPYALL>C
```

Typing <COPYALL>C copies all characters from the template to the command line, up to but not including "C". MS-DOS displays:

```
DIR PROG._
```

Note that the underline is your cursor. Now type:

```
.ASM
```

The result is:

```
DIR PROG.ASM_
```

The command line "DIR PROG.ASM" is now in the template and ready to be sent to the command processor for execution. To do this, press <CR>.

Now assume that you want to execute the following command:

```
TYPE PROG.ASM
```

To do this, type:

```
TYPE<INSERT> <COPYALL><RETURN>
```

Notice that when you are typing, the characters are entered directly into the command line and overwrite corresponding characters in the template. This automatic replacement feature is turned off when you press the insert key. Thus, the characters "TYPE" replace the characters "DIR" in the template. To insert a space between "TYPE" and "PROG.ASM", you pressed <INSERT> and then the space bar. Finally, to copy the rest of the template to the command line, you pressed <COPYALL> and then <CR>. The command TYPE PROG.ASM has been processed by MS-DOS, and the template becomes "TYPE PROG.ASM".

If you had misspelled "TYPE" as "BYTE", a command error would have occurred. Still, instead of throwing away the whole command, you could save the misspelled line before you press <CR> by creating a new template with the <NEWLINE> key:

```
BYTE PROG.ASM<NEWLINE>
```

You could then edit this erroneous command by typing:

```
T<COPY1>P<COPYALL>
```

The <COPY1> key copies a single character from the template to the command line. The resulting command line is then the command that you want:

```
TYPE PROG.ASM
```

As an alternative, you can use the same template containing BYTE PROG.ASM and then use the <SKIP1> and <INSERT> keys to achieve the same result:

```
<SKIP1><SKIP1><COPY1><INSERT>YP<COPYALL>
```

To illustrate how the command line is affected as you type, examine the keys typed on the left; their effect on the command line is shown on the right:

<SKIP1>	–	Skips over 1st template character
<SKIP1>	–	Skips over 2nd template character
<COPY1>	T	Copies 3rd template character
<INSERT>YP	TYP	Inserts two characters
<COPYALL>	TYPE PROG.ASM	Copies rest of template

Notice that <SKIP1> does not affect the command line. It affects the template by deleting the first character. Similarly, <SKIPUP> deletes characters in the template, up to but not including a given character.

These special editing keys can add to your effectiveness at the keyboard. The next section describes control character functions that can also help when you are typing commands.

CONTROL CHARACTER FUNCTIONS

A control character function is a function that affects the command line. You have already learned about <CONTROL-C> and <CONTROL-S>. Other control character functions are summarized in the following table.

Remember that when you type a control character, such as <CONTROL-C>, you must hold down the control key and then press the "C" key.

Control Character	Function
<CONTROL-C>	Aborts current command.
<CONTROL-H>	Removes last character from command line, and erases character from terminal screen (same as Backspace key).
<CONTROL-J>	Inserts physical end-of-line, but does not empty command line. Use the <LINE FEED> key to extend the current logical line beyond the physical limits of one terminal screen.
<CONTROL-P> or <CONTROL-N>	Echoes terminal output to the line printer. Press this key again to cancel echoing.
<CONTROL-S>	Suspends display of output to terminal screen. Press any key to resume.
<CONTROL-X>	Cancels the current line; empties the command line; and then outputs a back slash (\), carriage return, and line feed. The template used by the special editing commands is not affected.

NOTE: If you press CONTROL-P during a printout and continue keyboard entry, the printout will be a mixture of the keyboard entries and information already on the print spool. Be sure your printout is completed before using the CONTROL-P character.

Table 6.3 Control character functions

LINE EDITOR (EDLIN)

GENERAL INFORMATION

In this chapter, you learn how to use the Line Editor (EDLIN). You can use EDLIN to create, change, and display files, whether they are source program or text files. Specifically, you can use EDLIN to perform the following functions:

- Create new source files and save them.
- Update existing files and save both the updated and original files.
- Delete, edit, insert, and display lines.
- Search for, delete, or replace text within one or more lines.

The text in files created or edited by EDLIN is divided into lines, each up to 253 characters long. Line numbers are generated and displayed by EDLIN during the editing process, but are not actually present in the saved file.

When you insert lines, all line numbers following the inserted text advance automatically by the number of lines being inserted. When you delete lines in a file, all line numbers following the deleted text decrease automatically by the number of lines deleted. As a result, lines are always numbered consecutively in your file.

HOW TO START EDLIN

To start EDLIN, type:

```
EDLIN <filespec>
```

If you are creating a new file, the <filespec> should be the name of the file you wish to create. If EDLIN does not find this file on a drive, EDLIN creates a new file with the name you specify. The following message and prompt are displayed:

```
New file
*
 _
```

Notice that the prompt for EDLIN is an asterisk (*).

You can now type lines of text into your new file. To begin entering text, you must enter an I (Insert) command to insert lines. The I command is discussed later in this chapter.

If you want to edit an existing file, <filespec> should be the name of the file you want to edit. When EDLIN finds the file you specify on the designated or default drive, the file is loaded into memory. If the entire file can be loaded, EDLIN displays the following message on your screen:

```
End of input file
*
```

You can then edit the file using EDLIN editing commands.

If the file is too large to be loaded into memory, EDLIN loads lines until memory is 3/4 full, and then displays the * prompt. You can then edit the portion of the file that is in memory.

To edit the remainder of the file, you must save some of the edited lines on disk to free memory; then EDLIN can load the unedited lines from disk into memory. Refer to the Write and Append commands in this chapter for the procedure.

When you complete the editing session, you can save the original and the updated (new) files by using the End command. The End command is discussed in this chapter in the section EDLIN Commands. The original file is renamed with an extension of .BAK, and the new file has the filename and extension you specify in the EDLIN command. The original .BAK file is not erased until the end of the editing session, or until disk space is needed by the editor (EDLIN).

Do not try to edit a file with a filename extension of .BAK because EDLIN assumes that any .BAK file is a backup file. If you find it necessary to edit such a file, rename the file with another extension (using the MS-DOS RENAME command discussed in Chapter 5); then start EDLIN and specify the new <filespec>.

SPECIAL EDITING KEYS

The special editing keys and template discussed in Chapter 6 can be used to edit your text files. These keys are discussed in detail in this section.

Table 7.1 summarizes the commands, codes, and functions. Descriptions of the special editing keys follow the table.

Function	Key(s) *	Description
Copy one character	<COPY1> ESC S	Copies one character from the template to the new line.
Copy up to character	<COPYUP> ESC T	Copies all characters from the template to the new line, up to the character specified.
Copy template	<COPYALL> ESC U	Copies all remaining characters in the template to the screen.
Skip one character	<SKIP1> ESC V	Does not copy (skips over) a character.
Skip up to character	<SKIPUP> ESC W	Does not copy (skips over) the characters in the template, up to the character specified.
Quit input	<VOID> ESC E	Voids the current input; leaves the template unchanged.
Kill line	<KILL> ESC J	Voids line on template; current input sent to template.
Insert mode	<INSERT> ESC P	Enters insert mode.
Replace mode	<EXIT> ESC Q	Turns insert mode off; this is the default.
New template	<NEWLINE> ↵	Makes the new line the new template.

* Most functions require a 2-key entry. Do not press the keys simultaneously.

Table 7.1 Special editing keys

<COPY1>

KEY

ESC S

PURPOSE

Copies one character from the template to the command line.

COMMENTS

Pressing the <COPY1> key copies one character from the template to the command line. When the <COPY1> key is pressed, one character is inserted in the command line and insert mode is automatically turned off.

EXAMPLE

Assume that the screen shows:

```
1:*This is a sample file.
1:* _
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <COPY1> key copies the first character (T) to the second of the two lines displayed:

```
        1:*This is a sample file
<COPY1> 1:*T_
```

Each time the <COPY1> key is pressed, one more character appears:

```
<COPY1> 1:*Th_
<COPY1> 1:*Thi_
<COPY1> 1:*This_
```

<COPYUP>

KEY

ESC T

PURPOSE

Copies multiple characters up to a given character.

COMMENTS

Pressing the <COPYUP> key copies all characters up to a given character from the template to the command line. The given character is the next character typed after <COPYUP>; it is not copied or displayed on the screen. Pressing the <COPYUP> key causes the cursor to move to the single character that is specified in the command. If the template does not contain the specified character, nothing is copied. Pressing <COPYUP> also automatically turns off insert mode.

EXAMPLE

Assume that the screen shows:

```
    1:*This is a sample file.
    1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <COPYUP> key copies all characters up to the character specified immediately after the <COPYUP> key.

```
             1:*This is a sample file
   <COPYUP>p 1:*This is a sam_
```

<COPYALL>

KEY

ESC U

PURPOSE

Copies template to command line.

COMMENTS

Pressing the <COPYALL> key copies all remaining characters from the template to the command line. Regardless of the cursor position at the time the <COPYALL> key is pressed, the rest of the line appears, and the cursor is positioned after the last character on the line.

EXAMPLE

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <COPYALL> key copies all characters from the template (shown in the upper line displayed) to the line with the cursor (the lower line displayed):

```
          1:*This is a sample file    (template)
<COPYALL> 1:*This is a sample file._  (command line)
```

Also, insert mode is automatically turned off.

<SKIP1>

KEY

ESC V

PURPOSE

Skips over one character in the template.

COMMENTS

Pressing the <SKIP1> key skips over one character in the template. Each time you press the <SKIP1> key, one character is not copied from the template. The action of the <SKIP1> key is similar to the <COPY1> key, except that <SKIP1> skips a character in the template rather than copying it to the command line.

EXAMPLE

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <SKIP1> key skips over the first character ("T").

```
        1:*This is a sample file
<SKIP1> 1:*_
```

The cursor position does not change and only the template is affected. To see how much of the line has been skipped over, press the <COPYALL> key, which moves the cursor beyond the last character of the line.

```
            1:*This is a sample file.
    <SKIP1> 1:*_
<COPYALL> 1:*his is a sample file._
```

<SKIPUP>

KEY

ESC W

PURPOSE

Skips multiple characters in the template up to the specified character.

COMMENTS

Pressing the <SKIPUP> key skips over all characters up to a given character in the template. This character is not copied and is not shown on the screen. If the template does not contain the specified character, nothing is skipped over. The action of the <SKIPUP> key is similar to the <COPYUP> key, except that <SKIPUP> skips over characters in the template rather than copying them to the command line.

EXAMPLE

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <SKIPUP> key skips over all the characters in the template up to the character pressed after the <SKIPUP> key:

```
            1:*This is a sample file
<SKIPUP>p 1:*_
```

The cursor position does not change. To see how much of the line has been skipped over, press the <COPYALL> key to copy the template. This moves the cursor beyond the last character of the line:

```
             1:*This is a sample file:
 <SKIPUP>p  1:*_
<COPYALL>  1:*ple file._
```

<VOID>

KEY

ESC E

PURPOSE

Quits input and empties the command line.

COMMENTS

Pressing the <VOID> key empties the command line, but it leaves the template unchanged. <VOID> also prints a back slash (\), carriage return, and line feed, and turns insert mode off. The cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <COPYALL> key copies the template to the command line and the command line appears as it was before <VOID> was pressed.

EXAMPLE

Assume that the screen shows:

```
    1:*This is a sample file.
    1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Assume that you want to replace the line with "Sample File":

```
    1:*This is a sample file.
    1:*Sample File_
```

To cancel the line you just entered (Sample File), and to keep "This is a sample file.", press <VOID>. Notice that a backslash appears on the Sample File line to tell you it is cancelled.

```
            1:*This is a sample file.
    <VOID>  1:*Sample File\
            1:_
```

Press <CR> to keep the original line, or to perform any other editing functions. If <COPYALL> is pressed, the original template is copied to the command line:

```
    <COPYALL> 1: This is a sample file._
```

<INSERT>

KEY

ESC P

PURPOSE

Enters insert mode.

COMMENTS

Pressing the <INSERT> key causes EDLIN to enter insert mode. The current cursor position in the template is not changed. The cursor does move as each character is inserted. However, when you are finished inserting characters, the cursor is positioned at the same character as it was before the insertion began. Thus, characters are inserted in front of the character to which the cursor points.

EXAMPLE

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Assume that you press the <COPYUP> and "f" keys:

```
             1:*This is a sample file
<COPYUP>f 1:*This is a sample _
```

Now press the <INSERT> key and insert the characters "edit" and a space:

```
                1:*This is a sample file.
<COPYUP>f       1:*This is a sample _
<COPYUP>edit 1:*This is a sample edit _
```

If you now press the <COPYALL> key, the rest of the template is copied to the line:

```
                1:*This is a sample edit
<COPYALL>  1:*This is a sample edit file._
```

If you press the <CR> key, the remainder of the template is truncated, and the command line ends at the end of the insert:

```
<INSERT>edit <CR> 1:*This is a sample edit _
```

<EXIT>

KEY

ESC Q

PURPOSE

Enters replace mode.

COMMENTS

Pressing the <EXIT> key causes EDLIN to exit insert mode and to enter replace mode. All the characters you type overstrike and replace characters in the template. When you start to edit a line, replace mode is in effect. If the <CR> key is pressed, the remainder of the template is deleted.

EXAMPLE

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Assume that you then press <COPYUP>m, <INSERT>lary, <EXIT> tax, and then <COPYALL>:

```
                  1:*This is a sample file.
<COPYUP>m         1:*This is a sa_
<INSERT>lary      1:*This is a salary_
<EXIT> tax        1:*This is a salary tax_
<COPYALL>         1:*This is a salary tax file._
```

Notice that you inserted "lary" and replaced "mple" with "tax." If you type characters that extend beyond the length of the template, the remaining characters in the template are automatically appended when you press <COPYALL>.

<NEWLINE>

KEY

↵

PURPOSE

Creates a new template.

COMMENTS

Pressing the <NEWLINE> key copies the current command line to the template. The contents of the old template are deleted. Pressing <NEWLINE> outputs an @ ("at sign" character), a carriage return, and a line feed. The command line is also emptied and insert mode is turned off.

NOTE: <NEWLINE> performs the same function as the <VOID> key, except that the template is changed and an @ ("at sign" character) is printed instead of a \ (backslash).

EXAMPLE

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Assume that you enter <COPYUP>m, <INSERT>lary, <EXIT> tax, and then <COPYALL>:

```
                  1:*This is a sample file.
<COPYUP>m         1:*This is a sa_
<INSERT>lary      1:*This is a salary_
<EXIT> tax        1:*This is a salary tax_
<COPYALL>         1:*This is a salary tax file._
```

At this point, assume that you want this line to be the new template; press the <NEWLINE> key:

```
<NEWLINE>1:*This is a salary tax file. @
```

The @ indicates that this new line is now the new template. Additional editing can be done using the new template.

EDLIN COMMANDS

This section describes the individual EDLIN commands that perform editing functions on lines of text. Before using an EDLIN command, read the conventions and options that apply to all commands.

FORMAT CONVENTIONS

1. Pathnames are acceptable as options to commands. For example, typing EDLIN\BIN\USER\JOE\TEXT.TXT allows you to edit the TEXT.TXT file in the subdirectory JOE.
2. You can reference line numbers relative to the current line (the line with the asterisk). Use a minus sign with a number to indicate lines before the current line. Use a plus sign with a number to indicate lines after the current line.

   Example:

   ```
   -10, +10L
   ```

   This command lists 10 lines before the current line, the current line, and 10 lines after the current line.
3. Multiple commands may be issued on one command line. When you issue a command to edit a single line using a line number (<line>), a semicolon must separate commands on the line. Otherwise, one command may follow another without any special separators. In the case of a Search or Replace command, the <string> may be ended by a <CONTROL-Z> instead of a <CR>.

   Examples:

   ```
   15;-5,+5L
   ```

   The command line in the next example searches for "This string" and then displays 5 lines before and 5 lines after the line containing the matched string. If the search fails, then the displayed lines are those line numbers relative to the current line.

   ```
   SThis string<CONTROL-Z>-5,+L
   ```

4. You can type EDLIN commands with or without a space between the line number and command. For example, to delete line 6, the command 6D is the same as 6 D.
5. It is possible to insert a control character (such as CONTROL-C) into text by using the quote character CONTROL-V before it while in insert mode. CONTROL-V tells MS-DOS to recognize the next capital letter typed as a control character. It is also possible to use a control character in any of the string arguments of Search or Replace by using the special quote character. For example:

   S<CONTROL-V>Z
   finds the first occurrence
   of CONTROL-Z in a file

   R<CONTROL-V>Z<CONTROL-Z>foo
   replaces all occurrences
   of CONTROL-Z in a file with foo

   S<CONTROL-V>C<CONTROL-Z>bar
   replaces all occurrences
   of CONTROL-C with bar

   It is possible to insert CONTROL-V into the text by typing CONTROL-V-V.
6. The CONTROL-Z character ordinarily tells EDLIN, "This is the end of the file." If you have CONTROL-Z characters elsewhere in your file, you must tell EDLIN that these other control characters do not mean "End of File." Use the /B switch to tell EDLIN to ignore any CONTROL-Z characters in the file and to show you the entire file.

The EDLIN commands are summarized in the following table. They are also described in further detail following the description of command options.

Command	Purpose
<line>	Edits line no.
A	Appends lines
C	Copies lines
D	Deletes lines
E	Ends editing
I	Inserts lines
L	Lists text
M	Moves lines
P	Pages text
Q	Quits editing
R	Replaces lines
S	Searches text
T	Transfers text
W	Writes lines

Table 7.2 EDLIN commands

COMMAND OPTIONS

Several EDLIN commands accept one or more options. The effect of a command option varies, depending on with which command it is used. The following list describes each option.

<line>

<line> indicates a line number that you type. Line numbers must be separated by a comma or a space from other line numbers, other options, and from the command.

<line> may be specified in one of three ways:

- Number (n). Any number less than 65534 - - If a number larger than the largest existing line number is specified, then <line> means the line after the last line number.
- Period (.) - - If a period is specified for <line>, then <line> means the current line number. The current line is the last line edited, and is not necessarily the last line displayed. The current line is marked on your screen by an asterisk (*) between the line number and the first character.
- Pound (#) - - The pound sign indicates the line after the last line number. If you specify # for <line>, this has the same effect as specifying a number larger than the last line number.

<CR>

A carriage return entered without any of the <line> specifiers directs EDLIN to use a default value appropriate to the command.

?

The question mark option directs EDLIN to ask you if the correct string has been found. The question mark is used only with the Replace and Search commands. Before continuing, EDLIN waits for either a Y or <CR> for a yes response, or for any other key for a no response.

<string>

<string> represents text to be found, to be replaced, or to replace other text. The <string> option is used only with the Search and Replace commands. Each <string> must be ended by a <CONTROL-Z> or a <CR> (see the Replace command for details). Do not leave spaces between strings or between a string and its command letter, unless you want those spaces to be part of the string.

(A)PPEND

NAME

Append

PURPOSE

Adds the specified number of lines from disk to the file being edited in memory. The lines are added at the end of lines that are currently in memory.

SYNTAX

```
[<n>] A
```

COMMENTS

This command is meaningful only if the file being edited is too large to fit into memory. As many lines as possible are read into memory for editing when you start EDLIN.

To edit the remainder of the file that will not fit into memory, lines that have already been edited must be written to disk. Then you can load unedited lines from disk into memory with the Append command. (Refer to the Write command in this chapter for information on how to write edited lines to disk.)

If you do not specify the number of lines to append, lines are appended to memory until available memory is 3/4 full. No action is taken if available memory is already 3/4 full.

The message "End of input file" is displayed when the Append command has read the last line of the file into memory.

(C)OPY

NAME

Copy

PURPOSE

Copies a range of lines to a specified line number. The lines can be copied as many times as you want by using the =countÖ option.

SYNTAX

```
[<line>] , [<line>] ,<line> , [<count>]C
```

COMMENTS

If you do not specify a number in <count>, EDLIN copies the lines one time. If the first or the second <line> is omitted, the default is the current line. The file is renumbered automatically after the copy.

The line numbers must not overlap or you will get an "Entry error" message. For example, 3,20,15C would result in an error message.

EXAMPLES

Assume that the following file exists and is ready to edit:

```
    1:  This is a sample file
    2:  used to show copying lines.
    3:  See what happens when you use
    4:  the Copy command
    5:  (the C command)
    6:  to copy text in your file.
```

You can copy this entire block of text by issuing the following command:

```
    1,6,7C
```

The result is:

```
    1:  This is a sample file
    2:  used to show copying lines.
```

```
 3: See what happens when you use
 4: the Copy command
 5: (the C command)
 6: to copy text in your file.
 7: This is a sample file
 8: used to show copying lines.
 9: See what happens when you use
10: the Copy command
11: (the C command)
12: to copy text in your file.
```

If you want to place the text within other text, the third <line> option should specify the line before which you want the copied text to appear. For example, assume that you want to copy lines and insert them within the following file:

```
 1: This is a sample file
 2: used to show copying lines.
 3: See what happens when you use
 4: the Copy command
 5: (the C command)
 6: to copy text in your file.
 7: You can also use COPY
 8: to copy lines of text
 9: to the middle of your file.
10: End of sample file.
```

The command 3,6,9C results in the following file:

```
 1: This is a sample file
 2: used to show copying lines.
 3: See what happens when you use
 4: the Copy command
 5: (the C command)
 6: to copy text in your file.
 7: You can also use COPY
 8: to copy lines of text
 9: to the middle of your file.
10: See what happens when you use
11: the Copy command
12: (the C command)
13: to copy text in your file.
14: End of sample file.
```

(D)ELETE

NAME

Delete

PURPOSE

Deletes a specified range of lines in a file.

SYNTAX

```
[<line>] [,<line>] D
```

COMMENTS

If the first <line> is omitted, that option will default to the current line (the line with the asterisk next to the line number). If the second <line> is omitted, then just the first <line> will be deleted. When lines have been deleted, the line immediately after the deleted section becomes the current line and has the same line number as the first deleted <line> had before the deletion occurred.

EXAMPLES

Assume that the following file exists and is ready to edit:

```
  1:  This is a sample file
  2:  used to show dynamic line numbers.
  3:  See what happens when you use
  4:  Delete and Insert
   .
   .
   .
 25:  (the D and I commands)
 26:  to edit the text
 27:* in your file.
```

To delete multiple lines, type <line>,<line>D:

```
5,24D
```

The result is:

```
  1:  This is a sample file
  2:  used to show dynamic line numbers.
```

```
3:  See what happens when you use
4:  Delete and Insert
5:  (the D and I commands)
6:  to edit text
7:* in your file.
```

To delete a single line, type:

```
6D
```

The result is:

```
1:  This is a sample file
2:  used to show dynamic line numbers.
3:  See what happens when you use
4:  Delete and Insert
5:  (the D and I commands)
6:* in your file.
```

Next, delete a range of lines from the following file:

```
1:  This is a sample file
2:  used to show dynamic line numbers.
3:* See what happens when you use
4:  Delete and Insert
5:  (the D and I commands)
6:  to edit text
7:  in your file.
```

To delete a range of lines beginning with the current line, type:

```
,6D
```

The result is:

```
1:  This is a sample file
2:  used to show dynamic line numbers.
3:* in your file.
```

Notice that the lines are automatically renumbered.

<line> EDIT

NAME

Edit

PURPOSE

Edits line of text.

SYNTAX

```
[<line>]
```

COMMENTS

When a line number is typed, EDLIN displays the line number and text; then, on the line below, EDLIN reprints the line number. The line is now ready for editing. You may use any of the EDLIN editing commands to edit the line. The existing text of the line serves as the template until the <CR> key is pressed.

If no line number is typed (that is, if only the <CR> key is pressed), the line after the current line (marked with an asterisk) is edited. If no changes to the current line are needed and the cursor is at the beginning or end of the line, press the <CR> key to accept the line as is.

CAUTION

> If the <CR> key is pressed while the cursor is in the middle of the line, the remainder of the line is deleted.

EXAMPLE

Assume that the following file exists and is ready to edit:

```
1:  This is a sample file.
2:  used to show
3:  the editing of line
4:* four.
```

To edit line 4, type:

```
4
```

The contents of the line are displayed with a cursor below the line:

```
4:* four.
4:*_
```

Now, using the <COPYALL> special editing key, type:

```
<INSERT>number        4: number_
<COPYALL><CR>         4: number four.
                      5:*_
```

(E)ND

NAME

End

PURPOSE

Ends the editing session.

SYNTAX

```
E
```

COMMENTS

This command saves the edited file on disk, renames the original input file <filename>.BAK, and then exits EDLIN. If the file is created during the editing session, no .BAK file is created.

The E command takes no options. Therefore, you cannot tell EDLIN on which drive to save the file. The drive you want to save the file on must be selected when the editing session is started. If the drive is not selected when EDLIN is started, the file is saved on the disk in the default drive. It is still possible to COPY the file to a different drive using the MS-DOS COPY command.

You must be sure that the disk contains enough free space for the entire file. If the disk does not contain enough free space, the write is aborted and the edited file is lost, although part of the file might be written out to the disk.

EXAMPLE

```
    E<CR>
```

After execution of the E command, the MS-DOS default drive prompt (for example, A>) is displayed.

(I)NSERT

NAME

Insert

PURPOSE

Inserts text immediately before the specified <line>.

SYNTAX

[<line>] I

COMMENTS

If you are creating a new file, the I command must be given before text can be typed (inserted). Text begins with line number 1. Successive line numbers appear automatically each time <CR> is pressed.

EDLIN remains in insert mode until <CONTROL-C> is typed. When the insert is completed and insert mode has been exited, the line immediately following the inserted lines becomes the current line. All line numbers following the inserted section are incremented by the number of lines inserted.

If <line> is not specified, the default is the current line number and the lines are inserted immediately before the current line. If <line> is any number larger than the last line number, or if a pound sign (#) is specified as <line>, the inserted lines are appended to the end of the file. In this case, the last line inserted becomes the current line.

EXAMPLES

Assume that the following file exists and is ready to edit:

```
    1:  This is a sample file
    2:  used to show dynamic line numbers.
    3:  See what happens when you use
    4:  Delete and Insert
    5:  (the D and I commands)
    6:  to edit text
    7:* in your file.
```

To insert text before a specific line that is not the current line, type <line>I:

```
7I
```

The result is:

```
7:_
```

Now, type the new text for line 7:

```
7: and renumber lines
```

Then to end the insertion, press <CONTROL-Z> on the next line:

```
8: <CONTROL-Z>
```

Now type L to list the file. The result is:

```
1: This is a sample file
2: used to show dynamic line numbers.
3: See what happens when you use
4: Delete and Insert
5: (the D and I commands)
6: to edit text
7: and renumber lines
8:* in your file.
```

To insert lines immediately before the current line, type:

```
I
```

The result is:

```
8: _
```

Now, insert the following text and terminate with a <CONTROL-Z> on the next line:

```
8: so they are consecutive
9: <CONTROL-Z>
```

Now to list the file and see the result, type L:

The result is:

```
     1:  This is a sample file
     2:  used to show dynamic line numbers.
     3:  See what happens when you use
     4:  Delete and Insert
     5:  (the D and I commands)
     6:  to edit text
     7:  and renumber lines
     8:  so they are consecutive
     9:* in your file.
```

To append new lines to the end of the file, type:

```
     10I
```

This produces the following:

```
     10:_
```

Now, type the following new lines:

```
    10:  The insert command can place new lines
    11:  in the file; there's no problem
    12:  because the line numbers are dynamic;
    13:  they'll go all the way to 65533.
```

End the insertion by pressing <CONTROL-Z> on line 14. The new lines appear at the end of all previous lines in the file. Now type the list command, L:

The result is:

```
     1:  This is a sample file
     2:  used to show dynamic line numbers.
     3:  See what happens when you use
     4:  Delete and Insert
     5:  (the D and I commands)
     6:  to edit text
     7:  and renumber lines
     8:  so they are consecutive
     9:  in your file.
    10:  The insert command can place new lines
    11:  in the file; there's no problem
    12:  because the line numbers are dynamic;
    13:  they'll go all the way to 65533.
```

(L)IST

NAME

List

PURPOSE

Lists a range of lines, including the two lines specified.

SYNTAX

[<line>] [,<line>] L

COMMENTS

Default values are provided if either one or both of the options are omitted. If you omit the first option, as in:

```
,<line>L
```

the display starts 11 lines before the current line and ends with the specified <line>. The beginning comma is required to indicate the omitted first option.

NOTE: If the specified <line> is more than 11 lines before the current line, the display is the same as if you omitted both options.

If you omit the second option, as in

```
<line>L
```

23 lines are displayed, starting with the specified <line>.

If you omit both parameters, as in

```
L
```

23 lines are displayed: the 11 lines before the current line, the current line, and the 11 lines after the current line. If there are less than 11 lines before the current line, more than 11 lines after the current line are displayed to make a total of 23 lines.

EXAMPLES

Assume that the following file exists and is ready to edit:

```
  1:  This is a sample file
  2:  used to show dynamic line numbers
  3:  See what happens when you use
  4:  Delete and Insert
  5:  (the D and I commands)
   .
   .
   .
 15:* The current line contains an asterisk.
   .
   .
   .
 26:  to edit text
 27:  in your file.
```

To list a range of lines without reference to the current line, type<line> , <line>L:

```
  2,5L
```

The result is:

```
  2:  used to show dynamic line numbers.
  3:  See what happens when you use
  4:  Delete and Insert
  5:  (the D and I commands)
```

To list a range of lines beginning with the current line, type, <line> L:

```
  ,26L
```

The result is:

```
 15:* The current line contains an asterisk.
    .
    .
    .
 26:  to edit text
```

To list a range of 23 lines centered around the current line, type only L:

```
    L
```

The result is:

```
     4:  Delete and Insert
     5:  (the D and I commands)
      .
      .
      .
    13:  The current line is listed in the middle of the range.
    14:  The current line remains unchanged by the L command.
    15:* The current line contains an asterisk.
      .
      .
      .
    26:  to edit text.
```

(M)OVE

NAME

Move

PURPOSE

Moves a range of text to the line specified.

SYNTAX

```
[<line>],[<line>],<line>M
```

COMMENTS

Use the Move command to move a block of text (from the first <line> to the second <line> to another location in the file. The lines are renumbered according to the direction of the move. For example,

```
,+25,100M
```

moves the text from the current line plus 25 lines to line 100. If the line numbers overlap, EDLIN displays an "Entry error" message.

To move lines 20-30 to line 100, type:

```
20,30,100M
```

(P)AGE

NAME

Page

PURPOSE

Pages through a file 23 lines at a time.

SYNTAX

```
[<line>] [,<line>]P
```

COMMENTS

If the first <line> is omitted, that number defaults to the current line plus one. If the second <line> is omitted, 23 lines are listed. The new current line becomes the last line displayed and is marked with an asterisk.

(Q)UIT

NAME

Quit

PURPOSE

Quits the editing session, does not save any editing changes, and exits to the MS-DOS operating system.

SYNTAX

```
Q
```

COMMENTS

EDLIN prompts you to make sure you don't want to save the changes.

Type Y if you want to quit the editing session. No editing changes are saved and no .BAK file is created. Refer to the End command in this chapter for information about the .BAK file.

Type N or any other character if you want to continue the editing session.

NOTE: When started, EDLIN erases any previous copy of the file with an extension of .BAK to make room to save the new copy. If you reply Y to the "Abort edit (Y/N)?" message, your previous backup copy no longer exists.

EXAMPLE

```
    Q
    Abort edit (Y/N)?Y<CR>
A>_
```

(R)EPLACE

NAME

Replace

PURPOSE

Replaces all occurrences of a string of text in the specified range with a different string of text or blanks.

SYNTAX

```
[<line>] [,<line>] [?] R<string1><CONTROL-Z><string2>
```

COMMENTS

As each occurrence of <string1> is found, it is replaced by <string2>. Each line in which a replacement occurs is displayed. If a line contains two or more replacements of <string1> with <string2>, then the line is displayed once for each occurrence. When all occurrences of <string1> in the specified range are replaced by <string2>, the R command terminates and the asterisk prompt reappears.

If a second string is to be given as a replacement, then <string1> must be separated from <string2> with a <CONTROL-Z>. <String2> must also be ended with a <CONTROL-Z> <CR> combination or with a simple <CR>.

If <string1> is omitted, then Replace takes the old <string1> as its value. If there is no old <string1> (that is, this is the first replace done), then the replacement process is terminated immediately. If <string2> is omitted, then <string1> may be ended with a <CR>. If the first <line> is omitted in the range argument (as in, <line>) then the first <line> defaults to the line after the current line. If the second <line> is omitted (as in <line> or <line>,), the second <line> defaults to # . Remember that # indicates the line after the last line of the file.

If <string1> is ended with a <CONTROL-Z> and there is no <string2>, <string2> is taken as an empty string and becomes the new replace string. For example,

```
R<string2><CONTROL-Z><CR>
```

deletes occurrences of <string1>, but

```
R<string1><CR>      and
R<CR>
```

replaces <string1> by the old <string2> and the old <string1> with the old <string2>, respectively. Note that "old" here refers to a previous string specified either in a Search or a Replace command.

If the question mark (?) option is given, the Replace command stops at each line with a string that matches <string1>, displays the line with <string2> in place, and then displays the prompt "O.K.?." If you press Y or the <CR> key, then <string2> replaces <string1>, and the next occurrence of <string1> is found. Again, the "O.K.?" prompt is displayed. This process continues until the end of the range or until the end of the file. After the last occur-rence of <string1> is found, EDLIN displays the asterisk prompt.

If you press any key besides Y or <CR> after the "O.K.?" prompt, the <string1> is left as it was in the line, and Replace goes to the next occurrence of <string1>. If <string1> occurs more than once in a line, each occurrence of <string1> is replaced individually, and the "O.K.?" prompt is displayed after each replacement. In this way, only the desired <string1> is replaced, and you can prevent unwanted substitutions.

EXAMPLES

Assume that the following file exists and is ready for editing:

```
 1: This is a sample file
 2: used to show dynamic line numbers.
 3: See what happens when you use
 4: Delete and Insert
 5: (the D and I commands)
 6: to edit text
 7: in your file.
 8: The insert command can place new lines
 9: in the file; there's no problem
10: because the line numbers are dynamic;
11: they'll go all the way to 65533.
```

To replace all occurrences of <string1> with <string2> in a specified range, type:

```
2,12 Rand<CONTROL-Z> or <CR>
```

The result is:

```
4: Delete or Insert
5: (the D or I commors)
8: The insert commor can place new lines
```

Note that in the replacements, some unwanted substitutions have occurred. To avoid these and to confirm each replacement, the same original file can be used with a slightly different command.

In the next example, to replace only certain occurrences of the first <string> with the second <string>, type:

```
2? Rand <CONTROL-Z> or <CR>
```

The result is:

```
4: Delete or Insert
O.K.? Y
5: (the D or I commands)
O.K.? Y
5: (the D or I commors)
O.K.? N
8: The insert commor can place new lines
O.K.? N
*_
```

Now, type the List command (L) to see the result of all these changes:

```
    .
    .
4: Delete or Insert
5: (The D or I commands)
    .
8: The insert command can place new lines
    .
    .
```

(S)EARCH

NAME

Search

PURPOSE

Searches the specified range of lines for a specified string of text.

SYNTAX

```
[<line>] [,<line>] [?]S<string><CR>
```

COMMENTS

The <string> must be ended with a <CR>. The first line that matches <string> is displayed and becomes the current line. If the question mark option is not specified, the Search command terminates when a match is found. If no line contains a match for <string>, the message "Not found" is displayed.

If the question mark option (?) is included in the command, EDLIN displays the first line with a matching string; it then prompts you with the message "O.K.?". If you press either the Y or <CR> key, the line becomes the current line and the search terminates. If you press any other key, the search continues until another match is found, or until all lines are searched (and the "Not found" message is displayed).

If the first <line> is omitted (as in ,<line>S <string>), the first <line> defaults to the line after the current line. If the second <line> is omitted (as in <line> S <string> or <line>, S <string>), the second <line> defaults to # (line after last line of file), which is the same as <line>, # S<string>. If <string> is omitted, Search takes the old string if there is one. (Note that "old" here refers to a string specified in a previous Search or Replace command.) If there is not an old string (that is, no previous search or replace has been done), the command terminates immediately.

EXAMPLES

Assume that the following file exists and is ready for editing:

```
    1: This is a sample file
    2: used to show dynamic line numbers.
    3: See what happens when you use
```

```
  4: Delete and Insert
  5: (the D and I commands)
  6: to edit text
  7: in your file.
  8: The insert command can place new lines
  9: in the file; there's no problem
 10: because the line numbers are dynamic;
 11:* they'll go all the way to 65533.
```

To search for the first occurrence of the string "and," type

```
2,12 Sand<CR>
```

The following line is displayed:

```
4: Delete and Insert
```

to get the "and" in line 5, modify the search command by typing:

```
<SKIP1><COPYALL>,12 Sand<CR>
```

The search then continues from the line after the current line (line 4), since no first line was given. The result is:

```
5: (the D and I commands)
```

To search through several occurrences of a string until the correct string is found, type:

```
1, ? Sand
```

The result is:

```
4: Delete and Insert
O.K.?_
```

If you press any key (except Y or <CR>), the search continues, so type N here:

```
O.K.? N
```

Continue:

```
5: (the D and I commands)
O.K.?_
```

Now press Y to terminate the search:

```
O.K.? Y
*_
```

To search for string XYZ without the verification (O.K.?), type:

```
SXYZ
```

EDLIN reports a match and continues to search for the same string when you issue the S command:

```
S
```

EDLIN reports another match.

```
S
```

EDLIN reports the string is not found.

Note that <string> defaults to any string specified by a previous Replace or Search command.

(T)RANSFER

NAME

Transfer

PURPOSE

Inserts (merges) the contents of <filename> into the file currently being edited at <line>. If <line> is omitted, then the current line is used.

SYNTAX

[<line>] T <filename>

COMMENTS

This command is useful if you want to put the contents of a file into another file or into the text you are typing. The transferred text is inserted at the line number specified by <line> and the lines are renumbered.

(W)RITE

NAME

Write

PURPOSE

Writes a specified number of lines to disk from the lines that are being edited in memory. Lines are written to disk beginning with line number 1.

SYNTAX

[<n>] W

COMMENTS

This command is meaningful only if the file you are editing is too large to fit into memory. When you start EDLIN, EDLIN reads lines into memory until memory is 3/4 full.

To edit the remainder of your file, you must write edited lines in memory to disk. Then you can load additional lines from disk into memory by using the Append command.

NOTE: If you do not specify the number of lines, lines are written until memory is 3/4 full. No action is taken if available memory is already more than 3/4 full. All lines are renumbered, so that the first remaining line becomes line number 1.

ERROR MESSAGES

When EDLIN finds an error, one of the following error messages is displayed:

Cannot edit .BAK file- -rename file

Explanation

You attempted to edit a file with a filename extension of .BAK. .BAK files cannot be edited because this extension is reserved for backup copies.

Action

If you need the .BAK file for editing purposes, you must either RENAME the file with a different extension, or COPY the .BAK file and give it a different filename extension.

No room in directory for file

Explanation

When you attempted to create a new file, either the file directory was full or you specified an illegal disk drive or an illegal filename.

Action

Check the command line that started EDLIN for illegal filename and illegal disk drive entries. If the command is no longer on the screen and if you have not yet typed a new command, the EDLIN start command can be recovered by pressing the <COPYALL> key.

If this command line contains no illegal entries, run the CHKDSK program for the specified disk drive. If the status report shows that the disk directory is full, remove the disk. Insert and format a new disk.

Entry Error

Explanation

The last command typed contained a syntax error.

Action

Retype the command with the correct syntax and press <CR>.

Line too long

Explanation

During a Replace command, the string given as the replacement caused the line to expand beyond the limit of 253 characters. EDLIN aborted the Replace command.

Action

Divide the long line into two lines; then try the Replace command twice.

Disk Full- -file write not completed

Explanation

You gave the End command, but the disk did not contain enough free space for the whole file. EDLIN aborted the E command and returned you to the operating system. Some of the file may have been written to the disk.

Action

Only a portion (if any) of the file has been saved. You should probably delete that portion of the file and restart the editing session. The file is not available after this error. Always be sure that the disk has sufficient free space for the file to be written to disk before you begin your editing session.

Incorrect DOS version

Explanation

You attempted to run EDLIN under a version of MS-DOS that was not 2.0 or higher.

Action

You must make sure that the version of MS-DOS that you are using is 2.0 or higher.

Invalid drive name or file

Explanation

You have not specified a valid drive or filename when starting EDLIN.

Action

Specify the correct drive or filename.

Filename must be specified

Explanation

You did not specify a filename when you started EDLIN.

Action

Specify a filename.

Invalid parameter

Explanation

You specified a switch other than /B when starting EDLIN.

Action

Specify the /B switch when you start EDLIN.

Insufficient memory

Explanation

There is not enough memory to run EDLIN.

Action

You must free some memory by writing files to disk or by deleting files before restarting EDLIN.

File not found

Explanation

The filename specified during a Transfer command was not found.

Action

Specify a valid filename when issuing a Transfer command.

Must specify destination number

Explanation

A destination line number was not specified for a Copy or Move command.

Action

Reissue the command with a destination line number.

Not enough room to merge the entire file

Explanation

There was not enough room in memory to hold the file during a Transfer command.

Action

You must free some memory by writing some files to disk or by deleting some files before you can transfer this file.

FILE COMPARE (FC) UTILITY

GENERAL INFORMATION

It is sometimes useful to compare files on your disk. If you have copied a file and later want to compare copies to see which one is current, you can use the MS-DOS File Compare (FC) Utility.

The File Compare Utility compares the contents of two files. The difference between the two files can be output to the console or to a third file. The files being compared may be either source files (files containing source statements of a programming language), or binary files (files output by the assembler, the MS-LINK Linker utility, or by a high-level language compiler).

The comparisons are made in one of two ways: on a line-by-line or a byte-by-byte basis. The line-by-line comparison isolates blocks of lines that are different between the two files and prints those blocks of lines. The byte-by-byte comparison displays the bytes that are different between the two files.

LIMITATIONS ON SOURCE COMPARISONS

FC uses a large amount of memory as buffer (storage) space to hold the source files. If the source files are larger than available memory, FC compares what can be loaded into the buffer space. If no lines match in the portions of the files in the buffer space, FC displays only the message:

```
FILES ARE DIFFERENT
```

For binary files larger than available memory, FC compares both files completely, overlaying the portion in memory with the next portion from disk. All differences are output in the same manner as those files that fit completely in memory.

FILE SPECIFICATIONS

All file specifications use the following syntax:

```
[d:] <filename> [<.ext>]
```

d: is the letter designating a disk drive. If the drive designation is omitted, FC defaults to the operating system's (current) default drive.

filename is a 1- to 8-character name of the file.

.ext is a 1- to 3-character extension to the filename.

HOW TO USE FILE COMPARE

The syntax of FC is as follows:

```
FC [/# /B /W /C] <filename1> <filename2>
```

FC matches the first file (filename1) against the second (filename2) and reports any differences between them. Both filenames can be pathnames. For example,

```
FC B:\FOO\BAR\FILE1.TXT  \BAR\FILE2.TXT
```

FC takes FILE1.TXT in the \FOO\BAR directory of disk B and compares it with FILE2.TXT in the \BAR directory. Since no drive is specified for filename2, FC assumes that the \BAR directory is on the disk in the default drive.

FC SWITCHES

There are four switches that you can use with the File Compare Utility:

/B

Forces a binary comparison of both files. The two files are compared byte-to-byte, with no attempt to re-synchronize after a mismatch. The mismatches are printed as follows:

```
--ADDRS----F1----F2-
xxxxxxxx    yy    zz
```

(where xxxxxxxx is the relative address of the pair of bytes from the beginning of the file). Addresses start at 00000000; yy and zz are the mismatched bytes from file1 and file2, respectively. If one of the files contains less data than the other, then a message is printed out. For example, if file1 ends before file2, then FC displays:

Data left in F2

/#,

stands for a number from 1 to 9. This switch specifies the number of lines required to match for the files to be considered as matching again after a difference is found. If this switch is not specified, it defaults to 3. This switch is used only in source comparisons.

/W

Causes FC to compress "whites" (tabs and spaces) during the comparison. Thus, multiple contiguous whites in any line are considered as a single white space. Note that although FC compresses whites, it does not ignore them. The two exceptions are beginning and ending whites in a line, which are ignored. For example (note that an underscore represents a white)

__More_data_to_be_found__

matches with

More_data_to_be_found

and with

____More___data_to_be___found____

but does not match with

___Moredata_to_be_found

This switch is used only in source comparisons.

/C

Causes the matching process to ignore the case of letters. All letters in the files are considered uppercase letters. For example,

Much_MORE_data_IS_NOT_FOUND

matches

much_more_data_is_not_found

If both the /W and /C options are specified, then FC compresses whites and ignores case. For example,

__DATA_was_found__

will match:

data_was_found

This switch is used only in source comparisons.

DIFFERENCE REPORTING

The File Compare Utility reports the differences between the two files you specify by displaying the first filename, followed by the lines that differ between the files, followed by the first line to match in both files. FC then displays the name of the second file followed by the lines that are different, followed by the first line that matches. The default for the number of lines to match between the files is 3. (If you want to change this default, specify the number of lines with the /# switch.) For example,

```
              . . .
              . . .

----------<filename1>
<difference>
<1st line to match file 2 in file1>

----------<filename2>
<difference>
<1st line to match file1 in file2>

-------------------------------------


              . . .
              . . .
```

FC continues to list each difference.

If there are too many differences (involving too many lines), the program simply reports that the files are different and stops.

If no matches are found after the first difference is found, FC displays:

```
*** Files are different ***
```

and returns to the MS-DOS default drive prompt (for example, A>).

REDIRECTING FC OUTPUT TO A FILE

The differences and matches between the two files you specify are displayed on your screen unless you redirect the output to a file. This is accomplished in the same way as MS-DOS command redirection (refer to Chapter 4, Learning About Commands).

To compare File1 and File2 and then send the FC output to DIFFER.TXT, type:

```
FC File1 File2 > DIFFER.TXT
```

The differences and matches between File1 and File2 are put into DIFFER.TXT on the default drive.

EXAMPLES

Example 1 - - Compare (No Switches)

Assume these two ASCII files are on disk:

```
ALPHA.ASM        BETA.ASM

FILE A           FILE B
------------------------------
A                A
B                B
C                C
D                G
E                H
F                I
G                J
H                1
I                2
M                P
N                Q
O                R
P                S
Q                T
R                U
S                V
```

```
    T                    4
    U                    5
    V                    W
    W                    X
    X                    Y
    Y                    Z
    Z
```

To compare the two files and display the differences on the terminal screen, type:

```
    FC ALPHA.ASM BETA.ASM
```

FC compares ALPHA.ASM with BETA.ASM and displays the differences on the terminal screen. All other defaults remain intact. (The defaults are: do not use tabs, spaces, or comments for matches, and do a source comparison on the two files.)

The output appears as follows on the terminal screen (the Notes do not appear):

```
-----------ALPHA.ASM
D                                   NOTE: ALPHA file
E                                   contains defg,
F                                   BETA contains g.
G

-----------BETA.ASM
G

------------------------------

-----------ALPHA.ASM
M                                   NOTE: ALPHA file
N                                   contains mno where
O                                   BETA contains j12.
P
```

```
-----------BETA.ASM
J
1
2
P

-----------------------------

-----------ALPHA.ASM
W                                  NOTE: ALPHA file
                                   contains w where
-----------BETA.ASM                BETA contains 45w.
4
5
W
```

Example 2 - - Compare with Number Switch

You can print the differences on the line printer using the same two source files. In this example, four successive lines must be the same to constitute a match.

Type:

```
FC /4 ALPHA.ASM BETA.ASM > PRN
```

The following output appears on the line printer:

```
----------ALPHA.ASM
D
E
F
G
H
I
M
N                                  NOTE: p is the 1st of
O                                  a string of 4 matches.
P
```

```
----------BETA.ASM
G
H
I
J
1
2
P

-----------------------------

----------ALPHA.ASM
W
                                     NOTE: w is the 1st of a
----------BETA.ASM                   string of 4 matches.
4
5
W
```

Example 3 - - Compare with Binary Switch

This example forces a binary comparison and then displays the differences on the terminal screen using the same two source files as were used in the previous examples.

Type:

```
   FC /B ALPHA.ASM BETA.ASM
```

The /B switch in this example forces binary comparison. This switch and any others must be typed before the filenames in the FC command line. The following display appears:

```
--ADDRS----F1---F2--
 00000009    44   47
 0000000C    45   48
 0000000F    46   49
 00000012    47   4A
 00000015    48   31
 00000018    49   32
 0000001B    4D   50
 0000001E    4E   51
 00000021    4F   52
 00000024    50   53
 00000027    51   54
```

```
0000002A     52   55
0000002D     53   56
00000030     54   34
00000033     55   35
00000036     56   57
00000039     57   58
0000003C     58   59
0000003F     59   5A
00000042     5A   1A
```

ERROR MESSAGES

When the File Compare Utility detects an error, one or more of the following error messages are displayed:

Incorrect DOS version
    You are running FC under a version of MS-DOS that is not 2.0 or higher.

Invalid parameter:<option>
    One of the switches that you have specified is invalid.

File not found:<filename>
    FC could not find the filename you specified.

Read error in:<filename>
    FC could not read the entire file.

Invalid number of parameters
    You have specified the wrong number of options on the FC command line.

LINKER PROGRAM (MS-LINK)

GENERAL INFORMATION

In this chapter you learn about the Linker program, called MS-LINK. Read the entire chapter before you use MS-LINK.

NOTE: If you are not going to compile and link programs, you do not need to read this chapter.

MS-LINK is a program that performs the following functions:

- Combines separately produced object modules into one relocatable load module - - a program you can run.
- Searches library files for definitions of unresolved external references.
- Resolves external cross-references.
- Produces a listing that shows both the resolution of external references and error messages.

PROGRAM OVERVIEW

When you write a program, you write it in source code. This source code is passed through a compiler which produces object modules. The object modules must be passed through the link process to produce machine language that the computer can understand directly. This machine language is in the form required for running programs.

You may wish to link (combine) several programs and run them together. Each of your programs may refer to a symbol that is defined in another object module. This reference is called an external reference.

MS-LINK combines several object modules into one relocatable load module, or Run file (called an .EXE or Executable file). As it combines modules, MS-LINK makes sure that all external references between object modules are defined. LINK can search

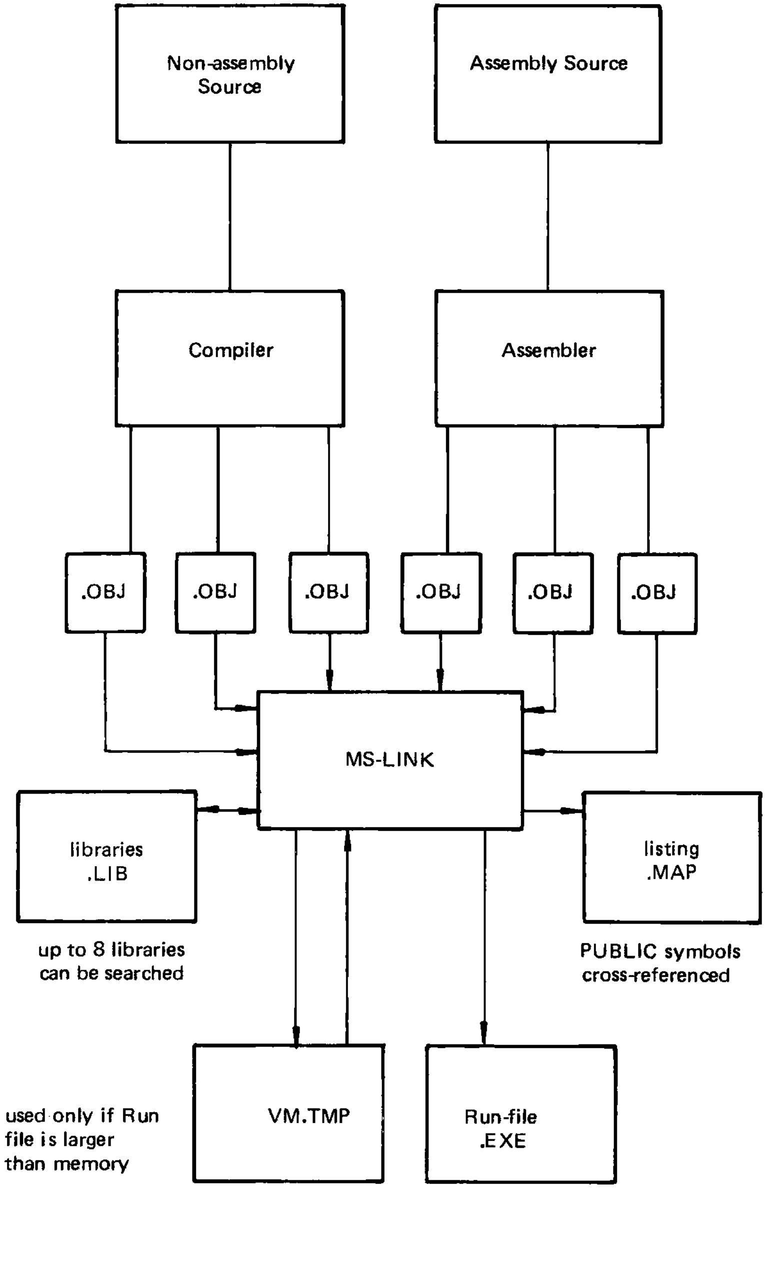


Figure 9.1 The MS-LINK Operation

several library files for definitions of any external references that are not defined in the object modules.

MS-LINK also produces a List file that shows external references resolved, and it also displays any error messages.

MS-LINK uses available memory as much as possible. When available memory is exhausted, MS-LINK creates a temporary disk file named VM.TMP.

Figure 9.1 illustrates the various parts of the MS-LINK operation.

DEFINITIONS YOU'LL NEED TO KNOW

Some of the terms used in this chapter are explained below to help you understand how MS-LINK works. Generally, if you are linking object modules compiled from BASIC, Pascal, or a high-level language, you do not need to know these terms. If you are writing and compiling programs in assembly language, however, you need to understand MS-LINK and the definitions described in this section.

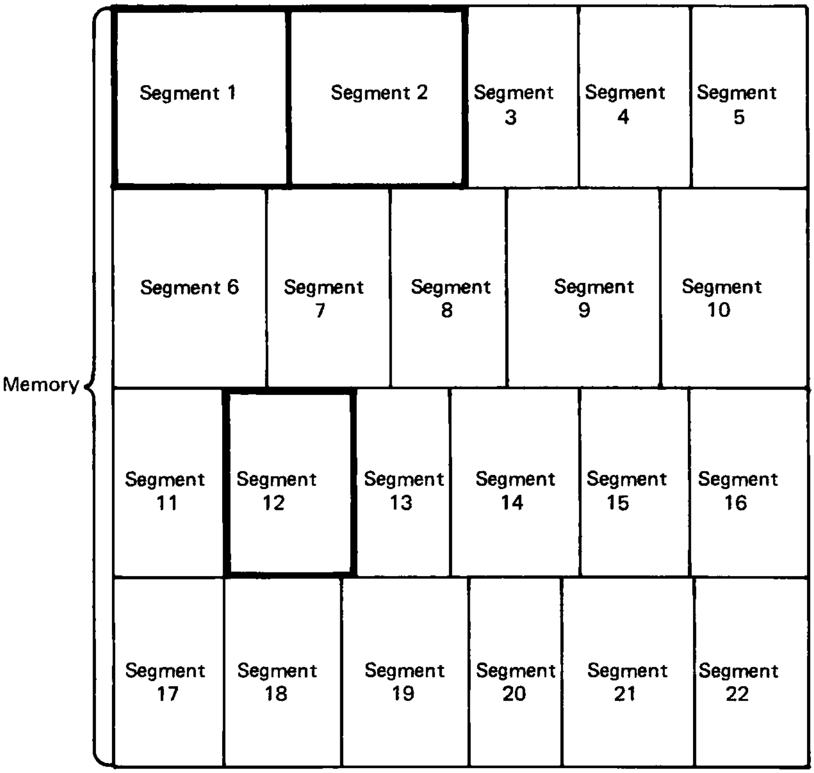
In MS-DOS, memory can be divided into segments, classes, and groups. Figure 9.2 illustrates these concepts.

Assume that the three segments have the following names.

	Segment Name	Segment Class Name
Segment 1	PROG.1	CODE
Segment 2	PROG.2	CODE
Segment 3	PROG.3	DATA

Note that segments 1, 2, and 12 have different segment names but may or may not have the same segment class name. Segments 1, 2, and 12 form a group with a group address of the lowest address of segment 1 (that is, the lowest address in memory).

Each segment has a segment name and a class name. MS-LINK loads all segments into memory by class name from the first segment encountered to the last. All segments assigned to the same class are loaded into memory contiguously.



Highlighted area = a group (64 K bytes addressable)

Figure 9.2 How memory is divided

During processing, MS-LINK references segments by their addresses in memory (where they are located). MS-LINK does this by finding groups of segments.

A group is a collection of segments that fit within a 64K byte area of memory. The segments do not need to be contiguous to form a group (see illustration). The address of any group is the lowest address of the segments in that group. At link time, MS-LINK analyzes the groups, then references the segments by the address in memory of that group. A program may consist of one or more groups.

If you are writing in assembly language, you may assign the group and class names in your program. In high-level languages (BASIC, COBOL, FORTRAN, Pascal), the naming is done automatically by the compiler.

FILES THAT MS-LINK USES

MS-LINK works with one or more input files, produces two output files, may create a temporary disk file, and may be directed to search up to eight library files.

For each type of file, you may give a 3-part file specification. The format for MS-LINK file specifications is the same as that of a disk file:

```
[d:]<filename>[<.ext>]
```

- d: is the drive designation. Permissible drive designations for MS-LINK are A: through O:. The colon is always required as part of the drive designation.
- filename is any legal filename of one to eight characters.
- .ext is a 1- to 3-character extension to the filename. The period is always required as part of the extension.

Input File Extensions

If no filename extensions are given in the input (object) file specifications, MS-LINK recognizes the following extensions by default:

.OBJ	Object
.LIB	Library

Output File Extensions

MS-LINK appends the following default extensions to the output (Run and List) files:

.EXE	Run (may not be overridden)
.MAP	List (may be overridden)

VM.TMP (Temporary) File

MS-LINK uses available memory for the link session. If the files to be linked create an output file that exceeds available memory, MS-LINK creates a temporary file, names it VM.TMP, and puts it on the disk in the default drive. If MS-LINK creates VM.TMP, it displays the message:

```
VM.TMP has been created.
Do not change disk in drive,<d:>
```

Once this message has been displayed, you must not remove the disk from the default drive until the link session ends. If the disk is removed, the operation of MS-LINK will be unpredictable, and MS-LINK might display the error message:

```
Unexpected end of file on VM.TMP
```

The contents of VM.TMP are written to the file named following the "Run File:" prompt. VM.TMP is a working file only and is deleted at the end of the linking session.

CAUTION

Do not use VM.TMP as a filename for any file. If you have a file named VM.TMP on the default drive and MS-LINK requires the VM.TMP file, MS-LINK deletes the VM.TMP already on disk and creates a new VM.TMP. Thus, the contents of the previous VM.TMP file will be lost.

USING MS-LINK

STARTING MS-LINK

MS-LINK requires two types of input: a command to start MS-LINK and responses to command prompts. In addition, six switches control MS-LINK features. Usually, you type all the commands to MS-LINK on the terminal keyboard. As an option, answers to the command prompts and any switches may be contained in a response file. Command characters can be used to assist you while giving commands to MS-LINK.

You may start MS-LINK in any of three ways. The first method is to type the commands in response to individual prompts. In the second method, you type all commands on the line used to start MS-LINK. To start MS-LINK by the third method, you must create a response file that contains all the necessary commands and tell MS-LINK where that file is when you start MS-LINK.

Method 1	LINK
Method 2	LINK <filenames> [/switches]
Method 3	LINK @<filespec>

Summary of methods to start MS-LINK

Method 1: Prompts

To start MS-LINK with method 1, type:

```
LINK
```

MS-LINK is loaded into memory, and then MS-LINK displays four text prompts that appear one at a time. You answer the prompts to tell MS-LINK to perform specific tasks.

At the end of each line, you may type one or more switches, preceded by the switch character (in this case, a forward slash).

The command prompts are summarized in the following table and are described in more detail in "Command Prompts."

PROMPT	RESPONSES
Object Modules [.OBJ]:	List .OBJ files to be linked. They must be separated by spaces or plus signs (+). If a plus sign is the last character typed, the prompt reappears. There is no default; a response is required.
Run File [Object-file.EXE]:	Give filename for executable object code. The default is first-object-filename.EXE. (You cannot change the output extension.)

List File [Run-file.MAP]:	Give filename for listing. The default is RUN filename.
Libraries [ ]:	List filenames to be searched, separated by spaces or plus signs (+). If a plus sign is the last character typed, the prompt reappears. The default is no search. (Extensions will be changed to .LIB.)

Method 2: Command Line

To start MS-LINK using method 2, type all commands on one line. The entries following LINK are responses to the command prompts. The entry fields for the different prompts must be separated by commas. Use the following syntax:

```
LINK <object-list>,<runfile>,<listfile>,<lib-list>[/switch... ]
```

- object-list is a list of object modules, separated by plus signs.
- runfile is the name of the file to receive the executable output.
- listfile is the name of the file to receive the listing.
- lib-list is a list of library modules to be searched.
- /switch refers to optional switches, which may be placed following any of the response entries (just before any of the commas or after the <lib-list>, as shown).

To select the default for a field, simply type a second comma with no spaces between the two commas.

```
LINK
FUN+TEXT+TABLE+CARE/P/M, ,FUNLIST,COBLIB.LIB
```

This command causes MS-LINK to be loaded, followed by the object modules FUN.OBJ, TEXT.OBJ, TABLE.OBJ, and CARE.OBJ. MS-LINK then pauses (as a result of using the /P switch). MS-LINK links the object modules when you press any key, and produces a global symbol map (the /M switch); defaults to FUN.EXE Run file; creates a List file named FUNLIST.MAP; and searches the Library file COBLIB.LIB.

Method 3: Response File

To start MS-LINK with method 3, type:

```
LINK @<filespec>
```

filespec is the name of a response file. A response file contains answers to the MS-LINK prompts (shown in method 1) and may also contain any of the switches. When naming a response file, the use of filename extensions is optional. Method 3 permits the command that starts MS-LINK to be entered from the keyboard or within a batch file without requiring you to take any further action.

To use this option, you must create a response file containing several lines of text, each of which is the response to an MS-LINK prompt. The responses must be in the same order as the MS-LINK prompts discussed in method 1. If desired, a long response to the "Object Modules:" or "Libraries:" prompt may be typed on several lines by using a plus sign (+) to continue the same response onto the next line.

Use switches and command characters in the response file the same way as they are used for responses typed on the terminal keyboard.

When the MS-LINK session begins, each prompt is displayed in order with the responses from the response file. If the response file does not contain answers for all the prompts (in the form of filenames, the semicolon command character, or carriage returns), MS-LINK displays the prompt that does not have a response and then waits for you to type a legal response. When a legal response is typed, MS-LINK continues the link session.

Consider the following example:

```
FUN TEXT TABLE CARE
/PAUSE/MAP
FUNLIST
COBLIB.LIB
```

This response file tells MS-LINK to load the four object modules named FUN, TEXT, TABLE, and CARE. MS-LINK pauses before producing a public symbol map to permit you to swap disks (see discussion under /PAUSE in the Switches section before using this feature). When you press any key, the output files are named FUN.EXE and FUNLIST.MAP. MS-LINK searches the library file COBLIB.LIB and uses the default setting for the switches.

COMMAND CHARACTERS

MS-LINK provides three command characters.

Plus sign

Use the plus sign (+) to separate entries and to extend the current line in response to the "Object Modules:" and "Libraries:" prompts. (A space may be used to separate object modules.) To type a large number of responses (each may be very long), type a plus sign and a <CR> at the end of the line to extend it. If the plus sign and <CR> is the last entry following these two prompts, MS-LINK prompts you for more module names. When the "Object Modules:" or "Libraries:" prompt appears again, continue to type responses. When all the modules to be linked and libraries to be searched are listed be sure the response line ends with a module name and a <CR> and not a plus sign and <CR>.

Example:

```
Object Modules [.OBJ] : FUN TEXT
TABLE CARE+<CR>
Object Modules [.OBJ] :
FOO+FLIPFLOP+JUNQUE+<CR>
Object Modules [.OBJ] :
CORSAIR<CR>
```

Semicolon

To select default responses to the remaining prompts, use a single semicolon (;) followed immediately by a carriage return at any time after the first prompt (Run File:). This feature saves time and overrides the need to press a series of <CR> keys.

NOTE: Once the semicolon has been typed and entered (by pressing the <CR> key), you can no longer respond to any of the prompts for that link session. Therefore, do not use the semicolon to skip some prompts. To skip prompts, use the <CR> key.

Example:

```
Object Modules
[.OBJ] : FUN TEXT TABLE CARE<CR>
Run Module [FUN.EXT] : ;<CR>
```

No other prompts appear, and MS-LINK uses the default values (including FUN.MAP for the List file).

<CONTROL-C>

Use the <CONTROL-C> key to abort the link session at any time. If you type an erroneous response, such as the wrong filename or an incorrectly spelled filename, you must press <CONTROL-C> to exit MS-LINK then restart MS-LINK. If you typed the error but did not press the <CR> key, you may delete the erroneous characters with the backspace key, but for that line only.

COMMAND PROMPTS

MS-LINK asks you for responses to four text prompts. When you type a response to a prompt and press <CR>, the next prompt appears. When the last prompt is answered, MS-LINK begins linking automatically without further command. When the link session is finished, MS-LINK exits to the operating system. When the operating system prompt appears, MS-LINK has finished successfully. If the link session is unsuccessful, MS-LINK displays the appropriate error message.

MS-LINK prompts the user for the names of Object, Run, and List files, and for Libraries. The prompts are listed in order of appearance. The default response is shown in square brackets ([ ]) following the prompt, for prompts which can default to preset responses. The "Object Modules:" prompt, however, has no preset filename response and requires you to type a filename.

Object Modules [.OBJ] :

Type a list of the object modules to be linked. MS-LINK assumes by default that the filename extension is .OBJ. If an object module has any other filename extension, the extension must be given. Otherwise, the extension may be omitted.

Modules must be separated by plus signs (+).

Remember that MS-LINK loads segments into classes in the order encountered. You can use this information to set the order in which the object modules are read by MS-LINK.

Run File [First-Object-filename.EXE] :

Typing a filename creates a file for storing the Run (executable) file that results from the link session. All Run files receive the filename extension .EXE, even if you specify an extension other than .EXE.

If no response is typed to the "Run File:" prompt, MS-LINK uses the first filename typed in response to the "Object Modules:" prompt as the RUN filename.

Example:

Run File [FUN.EXE] : B:PAYROLL/P

This response directs MS-LINK to create the Run file PAYROLL.EXE on drive B:. Also, MS-LINK pauses, which allows you to insert a new disk to receive the Run file.

List File [Run-Filename.MAP] :

The List file contains an entry for each segment in the input (object) modules. Each entry shows the addressing in the Run file.

The default response is the Run filename with the default filename extension .MAP.

Libraries [ ] :

The valid responses are up to eight library filenames or simply a carriage return. (A carriage return means no library search.) Library files must have been created by a library utility. MS-LINK assumes by default that the filename extension is .LIB for library files.

Library filenames must be separated by spaces or plus signs (+).

MS-LINK searches library files in the order listed to resolve external references. When it finds the module that defines the external symbol, MS-LINK processes that module as another object module.

If MS-LINK cannot find a library file on the disks in the disk drives, it displays the message:

```
Cannot find library <library-name>
Type new drive letter:
```

Press the letter for the drive designation (for example, B).

MS-LINK SWITCHES

The six MS-LINK switches control various MS-LINK functions. Switches must be typed at the end of a prompt response, regardless of the method used to start MS-LINK. Switches may be grouped at the end of any response, or may be scattered at the end of several. If more than one switch is typed at the end of one response, each switch must be preceded by a forward slash (/).

All switches may be abbreviated. The only restriction is that an abbreviation must be sequential from the first letter through the last typed; no gaps or transpositions are allowed. For example, examine the following lists of valid and invalid abbreviations.

Legal	Illegal
/D	/DSL
/DS	/DAL
/DSA	/DLC
/DSALLOCA	/DSALLOCT

/DSALLOCATE

Using the /DSALLOCATE switch tells MS-LINK to load all data at the high end of the Data Segment. Otherwise, MS-LINK loads all data at the low end of the Data Segment. At runtime, the DS pointer is set to the lowest possible address to allow the entire DS segment to be used. Use of the /DSALLOCATE switch in combination with the default load low (that is, the /HIGH switch is not used) permits the user application to dynamically allocate any available memory below the area specifically allocated within DGroup, yet to remain addressable by the same DS pointer. This dynamic allocation is needed for Pascal and FORTRAN programs.

NOTE: Your application program may dynamically allocate up to 64K bytes (or the actual amount of memory available) less the amount allocated within DGroup.

/HIGH

Use of the /HIGH switch causes MS-LINK to place the Run file as high as possible in memory. Otherwise, MS-LINK places the Run file as low as possible.

CAUTION

Do not use the /HIGH switch with Pascal or FORTRAN programs.

/LINENUMBERS

The /LINENUMBERS switch tells MS-LINK to include in the List file the line numbers and addresses of the source statements in the input modules. Otherwise, line numbers are not included in the List file.

NOTE: Not all compilers produce object modules that contain line number information. In these cases, of course, MS-LINK cannot include line numbers.

/MAP

/MAP directs MS-LINK to list all public (global) symbols defined in the input modules. If /MAP is not given, MS-LINK lists only errors (including undefined globals).

The symbols are listed alphabetically. For each symbol, MS-LINK lists its value and its segment:offset location in the Run file. The symbols are listed at the end of the List file.

/PAUSE

The /PAUSE switch causes MS-LINK to pause in the link session when the switch is encountered. Normally, MS-LINK performs the linking session from beginning to end without stopping. This switch allows the user to swap the disks before MS-LINK outputs the Run (.EXE) file.

When MS-LINK encounters the /PAUSE switch, it displays the message:

```
About to generate .EXE file
Change disks<hit any key>
```

MS-LINK resumes processing when the user presses any key.

CAUTION

Do not remove the disk that receives the List file, or the disk used for the VM.TMP file, if one has been created.

/STACK:<number>

The number entry represents any positive numeric value (in hexadecimal radix) up to 65536 bytes. If a value from 1 to 511 is typed, MS-LINK uses 512. If the /STACK switch is not used for a link session, MS-LINK calculates the necessary stack size automatically.

All compilers and assemblers should provide information in the object modules that allow the linker to compute the required stack size.

At least one object (input) module must contain a stack allocation statement. If not, MS-LINK displays the following error message:

```
WARNING: NO STACK STATEMENT
```

SAMPLE MS-LINK SESSION

This sample shows you the type of information that is displayed during an MS-LINK session.

In response to the MS-DOS prompt, type:

```
LINK
```

The system displays the following messages and prompts (your answers are underlined):

```
Microsoft Object Linker V.2.00
(C) Copyright 1982 by Microsoft Inc.

Object Modules [.OBJ] : NCRIO SYSINIT
Run File [NCRIO.EXE] :
List File [NUL.MAP] : NCRIO /MAP
Libraries [.LIB] : ;
```

Consider how your answers direct MS-LINK and how others affect the output:

- By specifying /MAP, you get both an alphabetic listing and a chronological listing of public symbols.

- By responding PRN to the "List File:" prompt, you can redirect your output to the printer.
- By specifying the /LINE switch, MS-LINK gives you a listing of all line numbers for all modules. (Note that the /LINE switch can generate a large volume of output.)
- By pressing <CR> in response to the "Libraries:" prompt, an automatic library search is performed.

Once MS-LINK locates all libraries, the linker map displays a list of segments in the order of their appearance within the load module. The list might look like this:

```
Start      Stop       Length     Name
00000H     009ECH     09EDH      CODE
009F0H     01166H     0777H      SYSINITSEG
```

The information in the Start and Stop columns shows the 20-bit hex address of each segment relative to location zero. Location zero is the beginning of the load module.

The addresses displayed are not the absolute addresses where these segments are loaded. Consult the *PROGRAMMER'S MANUAL* for information on how to determine where relative zero is actually located, and also on how to determine the absolute address of a segment.

Because the /MAP switch was used, MS-LINK displays the public symbols by name and value. For example:

```
ADDRESS            PUBLICS_BY_NAME
009F:0012          BUFFERS
009F:0005          CURRENT_DOS_LOCATION
009F:0011          DEFAULT_DRIVE
009F:000B          DEVICE_LIST
009F:0013          FILES
009F:0009          FINAL_DOS_LOCATION
009F:000F          MEMORY_SIZE
009F:0000          SYSINIT

ADDRESS            PUBLICS BY VALUE
009F:0000          SYSINIT
009F:0005          CURRENT_DOS_LOCATION
009F:0009          FINAL_DOS_LOCATION
```

```
009F:000B      DEVICE_LIST
009F:000F      MEMORY_SIZE
009F:0011      DEFAULT_DRIVE
009F:0012      BUFFERS
009F:0013      FILES
```

ERROR MESSAGES

All errors cause the link session to abort. After you find the cause of the error and correct it, rerun MS-LINK. The following error messages are displayed by MS-LINK; they are mostly self-explanatory.

ATTEMPT TO ACCESS DATA OUTSIDE OF SEGMENT BOUNDS, POSSIBLY BAD OBJECT MODULE
There is probably a bad object file.

BAD NUMERIC PARAMETER
Numeric value is not in digits.

CANNOT OPEN TEMPORARY FILE
MS-LINK is unable to create the file VM.TMP because the disk directory is full. Insert a new disk. Do not remove the disk that will receive the List.MAP file.

ERROR: DUP RECORD TOO COMPLEX
DUP record in assembly language module is too complex. Simplify DUP record in assembly language program.

ERROR: FIXUP OFFSET EXCEEDS FIELD WIDTH
An assembly language instruction refers to an address with a short instruction instead of a long instruction. Edit assembly language source and reassemble.

INPUT FILE READ ERROR
There is probably a bad object file.

INVALID OBJECT MODULE
An object module(s) is incorrectly formed or incomplete (as when assembly is stopped in the middle).

SYMBOL DEFINED MORE THAN ONCE

MS-LINK found two or more modules that define a single symbol name.

PROGRAM SIZE OR NUMBER OF SEGMENTS EXCEEDS CAPACITY OF LINKER

The total size may not exceed 384K bytes and the number of segments may not exceed 255.

REQUESTED STACK SIZE EXCEEDS 64K

Specify a size greater than or equal to 64K bytes with the /STACK switch.

SEGMENT SIZE EXCEEDS 64K

64K bytes is the addressing system limit.

SYMBOL TABLE CAPACITY EXCEEDED

Very many and/or very long names were typed, exceeding the limit of approximately 25K bytes.

TOO MANY EXTERNAL SYMBOLS IN ONE MODULE

The limit is 256 external symbols per module.

TOO MANY GROUPS

The limit is 10 groups.

TOO MANY LIBRARIES SPECIFIED

The limit is 8 libraries.

TOO MANY PUBLIC SYMBOLS

The limit is 1024 public symbols.

TOO MANY SEGMENTS OR CLASSES

The limit is 256 (segments and classes taken together).

UNRESOLVED EXTERNALS: <list>

The external symbols listed have no defining module among the modules or library files specified.

VM READ ERROR

This is a disk error; it is not caused by MS-LINK.

WARNING: NO STACK SEGMENT

None of the object modules specified contains a statement allocating stack space, but the user typed the /STACK switch.

WARNING: SEGMENT OF ABSOLUTE OR UNKNOWN TYPE

There is a bad object module or an attempt has been made to link modules that MS-LINK cannot handle (e.g., an absolute module).

WRITE ERROR IN TMP FILE

No more disk space remains to expand VM.TMP file.

WRITE ERROR ON RUN FILE

Usually, there is not enough disk space for the Run file.

DUAL-OPERATING SYSTEMS CONSIDERATIONS

You may use your NCR DECISION MATE V with more than one operating system. The dual-processor model, for example, provides processing capabilities of both 8- and 16-bit based applications. If you are planning to use MS-DOS and another operating system, you are responsible for protecting your data files and other disk software. Generally, data protection is simply a matter of keeping the disks properly labelled so that you always process using compatible ones. You must also be sure that only compatible software and data files are on the same disk.

This procedure of 'separation' is also valid for a fixed disk, where one of the logical units may be formatted for use by MS-DOS while the other is reserved for another operating system. When the fixed disk is shared, you should put a label on the unit clearly showing which logical disk is for which operating system.

Properly used, dual-operating sytems significantly increase your processing capabilities. Just remember the guidelines:

- Keep compatible operating system software, application software, and data on the same disks.
- Clearly label all disks, especially the fixed disk where the logical units are not visible.
- Never use a command that could destroy the contents of a disk without first finding out what's on the disk. If you are not sure about the contents, use a command that identifies the contents (MS-DOS CHKDSK for example).
- Finally, be sure you always have backup copies of all important software and data.

DISK ERRORS

If a disk error occurs at any time during a command or program, MS-DOS retries the operation three times. If the operation cannot be completed successfully, MS-DOS returns an error message in the following format:

```
<yyy> ERROR WHILE <I/O action> ON DRIVE x
Abort, Ignore, Retry:_
```

In this message, <yyy> may be one of the following:

- WRITE PROTECT
- NOT READY
- SEEK
- DATA
- SECTOR NOT FOUND
- WRITE FAULT
- DISK

The <I/O-action> may be either of the following:

- READING
- WRITING

The drive <x> indicates the drive in which the error occurred.

MS-DOS waits for you to enter one of the following responses:

A (Abort)
Terminate the program requesting the disk read or write.

I (Ignore)
Ignore the bad sector and pretend the error did not occur.

R (Retry)
Repeat the operation. Use this response when the error is corrected (such as with NOT READY or WRITE PROTECT errors).

Usually, you will want to attempt recovery by entering responses in this order:

R (to try again)
A (to terminate program and try a new disk)

One other error message might be related to faulty disk read or write:

FILE ALLOCATION TABLE BAD FOR DRIVE x

This message means that the copy in memory of one of the allocation tables has pointers to nonexistent blocks. Possibly the disk was incorrectly formatted or not formatted before use. If this error persists, the disk is currently unusable and must be formatted prior to use.

HOW TO OBTAIN AND INSTALL SOFTWARE

MAKING THE RIGHT PURCHASE

Your NCR DECISION MATE V with MS-DOS is highly flexible, accommodating nearly any application and language software as long as it is MS-DOS compatible. Think of the software you can use on your computer in three categories:

- Software that is distributed by NCR.
- Software that has been run by NCR on the DECISION MATE V and is available to any MS-DOS user.
- Software that has *not* been tested by NCR, but is available to any MS-DOS user.

Those packages in the first group can be obtained directly from NCR by contacting your NCR representative or licensed dealer. The software in the other categories can be purchased from an NCR licensed dealer or any reputable software house. You will want to be sure, however, that the software will run on your computer. Knowing what questions to ask and working with a knowledgeable dealer will help you to make the right decision.

1. Ask if the software is on an NCR MS-DOS format disk.
2. If the software is not available on an NCR MS-DOS format disk, ask if it is available on an IBM-PC format disk. (A 5 1/4-inch disk with either 160/180/320/360KB capacity.) Any of these disks can be used on your computer.
3. Most application packages and some language software require an installation program to interface with the specific hardware. Ask if the software is already installed to be used on an NCR DECISION MATE V, or if the package includes an installation (or install) program so that you can tailor the software yourself. If the software is installed for an MS-DOS ANSI device driver or a Lear Siegler terminal, it can be installed simply on your computer (see next section).

INSTALLING THE SOFTWARE

To fully use every feature on NCR DECISION MATE V, each application must be tailored to, or installed on, your computer. The installation procedure is different for each application, so every "off the shelf" package, such as MULTIPLAN™, contains its own program for accomplishing this task. The documentation that accompanies the application disk you purchase describes the way this program is run. We recommend that before you customize your application, you make a copy of the purchased disk and use the copy as your work disk. (See the FORMAT and DISKCOPY commands in chapter 5.) Save the original disk in a safe place and use it only to make copies.

To begin, most installation programs display a list of computer terminals. If this list includes NCR DECISION MATE V, select it; otherwise, choose an MS-DOS ANSI device driver. the Lear Siegler ADM-31, or the Lear Siegler ADM-3A terminal. Refer to table 1 for the functions that are active if you install your application as for an MS-DOS ANSI device driver, or table 2 if you install your application as on a Lear Siegler terminal.

If none of the above terminals are listed, you must describe the terminal characteristics of NCR DECISION MATE V to the application's installation program. Use the tables as a guide if you must enter these codes to install an application package. (All of these control characters/sequences are for use by applications, not for entry from the keyboard.)

Other tables include information you may need, depending on the particular application. Table 3 contains miscellaneous information and table 4 gives notes, frequencies, and cycles for programming music applications. For further reference, conversion and translation tables are included at the end of this section.

FUNCTIONAL CHARACTERISTICS INFORMATION

The following functions must be preceded by an ESCAPE <ESC> character (hex value 1B) and a left bracket <[> character (hex value 5B).

MS-DOS ANSI DRIVER TABLE (TABLE 1)

Function	ASCII string	hex string
Cursor Position	(row#);(col#)H or (row#);(col#)f	(row#)3B(col#)48 (row#)3B(col#)66
Cursor Left	(# of cols)D	(# of cols)44
Cursor Right	(# of cols)C	(# of cols)43
Cursor Down	(# of rows)B	(# of rows)42
Cursor Up	(# of rows)A	(# of rows)41
Device Status Report	6n	366E
Cursor Position Report (returned after 6n function)	(row#);(col#)R	(row#)3B(col#)52
Save Cursor Position	s	73
Restore Cursor Position	u	75
Erase Screen	2J	324A
Erase to End of Line	K	4B
Define Function Key	0;(fk#);"string"p	303B(fk#)3B22string2270
Disable Function Key Ext.	0;0p	303B3070
Enable Function Key Ext.	0;99p	303B393970

The information in parentheses represents data you supply. You do not enter the parentheses.

For example, to assign CHKDSK <CR> to function key 18, enter

<ESC>[0;18;"CHKDSK";13p

where 13 stands for <CR>, hex value 0D.

TERMINAL FUNCTION CODES (TABLE 2)

Function	ASCII string	hex string	Terminal*
Position Cursor	<ESC> =	1B3D	A,B
Row + Offset	(row#+32)	(row#+20hex)	A,B
Column + Offset	(col#+32)	(col#+20hex)	A,B
Cursor Left	^H	08	A,B
Cursor Right	^L	0C	A,B
Cursor Down	^J	0A	A,B
Cursor Up	^K	0B	A,B
Clear Screen & Cursor Home	^Z	1A	A,B
Clear to End of Line	^W	17	
Carriage Return	^M	0D	A,B
Escape	<ESC>	1B	A,B
Bell	^G	07	A,B
Home Cursor	^~	1E	
The following functions must be preceded by an ESCAPE <ESC> character, hex value 1B.			
Clear to End of Line	T or t	54 or 74	A
Clear to End of Screen	Y or y	59 or 79	A
Clear Screen and Cusor Home	: or *	3A or 2A	
Half Intensity On	)	29	A
Half Intensity Off	(	28	A
Reverse Video On	G4	4734	A
Blinking On	G2	4732	A

The information in parentheses represents data you supply. You do not enter the parentheses.

* A = Lear Siegler ADM-31; B = Lear Siegler ADM-3A

cont.

TERMINAL FUNCTION CODES (TABLE 2) . . . cont.

Function	ASCII string	hex string	Terminal*
Rev. Video & Blinking Off	G0	4730	A
Insert Line	E	45	A
Insert Character	Q	51	A
Delete Line	R	52	A
Delete Character	W	57	A
Play Music **	M	4D	

The information in parentheses represents data you supply. You do not enter the parentheses.

* A = Lear Siegler ADM-31; B = Lear Siegler ADM-3A

** Music can be programmed on the NCR DECISION MATE V. On receiving the string <ESC> M, the CRT driver accepts the next two numbers as frequency and tone length, respectively. Refer to Table 4 at the end of this section for the corresponding note, frequency, and number of cycles.

MISCELLANEOUS INFORMATION – Table 3

	ANSI	Lear-Siegler ADM 31/A
Number of rows *	24 (1-24)	24 (0-23)
Number of columns	80 (1-80)	80 (0-79)
Cursor origin	1/1	0/0
Input/Output technique	MS-DOS calls and commands (e.g. TYPE)	MS-DOS calls and commands (e.g. TYPE)
Cursor on/off	not active	not active
Keyboard click on/off	not active	not active

* Row 25 does not scroll.

MUSIC CODES — Table 4

Note	Frequency			Cycles
	dec.	hex	key	
Pause	32	20	Space	—
A	33	21	!	110
A#	34	22	"	116.5
B	35	23	#	123.5
C	36	24	$	131
C#	37	25	%	138.6
D	38	26	&	146.8
D#	39	27	'	155.8
E	40	28	(	164.8
F	41	29	)	174.6
F#	42	2A	*	185
G	43	2B	+	196
G#	44	2C	,	208
A	45	2D	—	220
A#	46	2E	.	233
B	47	2F	/	246.9
C (Middle C)	48	30	0	261.6
C#	49	31	1	277.4
D	50	32	2	293.7
D#	51	33	3	311
E	52	34	4	329.6
F	53	35	5	349.2
F#	54	36	6	370
G	55	37	7	392
G#	56	38	8	415
A	57	39	9	440
A#	58	3A	:	465
B	59	3B	;	493.9

cont.

MUSIC CODES — Table 4 (cont.)

Note	Frequency			Cycles
	dec.	hex	key	
C	60	3C	<	523.2
C#	61	3D	=	553
D	62	3E	>	587.3
D#	63	3F	?	622
E	64	40	@	659.3
F	65	41	A	698.5
F#	66	42	B	740
G	67	43	C	784
G#	68	44	D	830
A	69	45	E	880
A#	70	46	F	932
B	71	47	G	987.8
C	72	48	H	1046.5
C#	73	49	1	1108.7
D	74	4A	J	1174.7

GRAPHIC ATTRIBUTES (TABLE 5)		
Function	ASCII Code, preceded by ESC[	Hexadecimal Code, preceded by 1B 5B
GRAPHIC ATTRIBUTES OFF	0m	30 6D
HALF INTENSITY OFF	1m	31 6D
BLINKING ON	5m	35 6D
REVERSE VIDEO ON	7m	37 6D
HALF INTENSITY ON	8m	38 6D
BLACK FOREGROUND	30m	33 30 6D
RED FOREGROUND	31m	33 31 6D
GREEN FOREGROUND	32m	33 32 6D
YELLOW FOREGROUND	33m	33 33 6D
BLUE FOREGROUND	34m	33 34 6D
MAGENTA FOREGROUND	35m	33 35 6D
CYAN FOREGROUND	36m	33 36 6D
WHITE FOREGROUND	37m	33 37 6D
BLACK FOREGROUND	40m	34 30 6D
RED BACKGROUND	41m	34 31 6D
GREEN BACKGROUND	42m	34 32 6D
YELLOW BACKGROUND	43m	34 33 6D
BLUE BACKGROUND	44m	34 34 6D
MAGENTA BACKGROUND	45m	34 35 6D
CYAN BACKGROUND	46m	34 36 6D
WHITE BACKGROUND	47m	34 37 6D

TRANSLATION/CONVERSION INFORMATION

Because of the special language requirements of different countries, examples in this manual of characters, either input or displayed, may not match the characters on the keyboard. The following table shows substitute characters which produce the same hexadecimal code and satisfy the requirements of the operating system. Also, for users who need to refer to the complete hexadecimal chart, the ASCII code chart with the USASI code set is given after the keyboard table.

Country	Hex Codes and Corresponding Characters											
	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
US-English	#	$	@	[	\	]	^	`	{	|	}	~
UK-English	£	$	@	[	\	]	↑	`	{	|	}	~
French	£	$	à	°	ç	§	^	`	é	ù	è	¨
German	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
Swedish/Finnish	#	¤	@	Ä	Ö	Å	^	`	ä	ö	å	¨
Danish/Norwegian	£	$	@	Æ	Ø	Å	^	`	æ	ø	å	¨
Spanish	£	$	@	¡	Ñ	¿	^	`	{	ñ	}	~
Italian	£	$	§	°	ç	é	^	ù	à	ò	è	ì
Swiss	£	$	ç	à	é	è	^	`	ä	ö	ü	¨
Canadian	#	$	@	[	\	]	^	`	£	|	¢	¨
Canadian (bilingual)	#	$	@	[	ç	]	^	`	é	è	¢	¨
South African	#	$	@	Ê	'N	Ë	^	`	ê	'n	ë	¨
Portuguese	£	$	@	Ã	Õ	Ç	`	`	ã	õ	ç	~
Yugoslavian	#	$	Đ	Ć	Č	Š	Ž	đ	ć	č	š	ž

Special country keyboard definitions

ASCII CODE CHART

Binary b_4 - b_1 ►		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
b_8 - b_5 ▼	HEX	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0000	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010	2	b̸	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	∧	—
0110	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	DEL

LEGEND: (For Control Characters in USASI Code Set)

NUL	Null
SOH	Start of Heading
STX	Start of Text
ETX	End of Text
EOT	End of Transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell (Audible or Attention Signal)
BS	Backspace
HT	Horizontal Tabulation (Punched Card Strip)
LF	Line Feed
VT	Vertical Tabulation
FF	Form Feed
CR	Carriage Return
SO	Shift Out
SI	Shift In
DLE	Data Link Escape
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
NAK	Negative Acknowledge
SYN	Synchronous DLE (Sync Code)
ETB	End of Transmission Block
CAN	Cancel (Void Data)
EM	End of Media
SUB	Substitute
ESC	Escape
FS	File Separation (End of File)
GS	Group Separate
RS	Record Separator (End of Record)
US	Unit Separator (End of Field)
DEL	Delete

ADVANCED CONFIGURATION FEATURE

In many cases, there are installation-specific settings for MS-DOS that need to be configured at system startup. An example of this is a standard device driver, such as an online printer.

The MS-DOS configuration file (CONFIG.SYS) allows you to configure your system with a minimum of effort. With this file, you can add device drivers to your system at startup. The configuration file is simply an ASCII file that has certain commands for MS-DOS startup (boot). The boot process is as follows:

1. The disk boot sector is read. This contains enough code to read MS-DOS code and the installation's BIOS (machine-dependent code).
2. The MS-DOS code and BIOS are read.
3. A variety of BIOS initializations are done.
4. A system initialization routine reads the configuration file (CONFIG.SYS), if it exists, to perform device installation and other user options. Its final task is to execute the command interpreter, which finishes the MS-DOS boot process.

CHANGING THE CONFIG.SYS FILE

If there is not a CONFIG.SYS file on the MS-DOS disk, you can use the MS-DOS editor, EDLIN, to create a file; then save it on the MS-DOS disk in your root directory.

The following is a list of commands for the configuration file CONFIG.SYS:

BUFFERS = <number>
  Where the number is between 1 and 99. This is the number of sector buffers that MS-DOS should allocate in memory when it starts up. It is installation-dependent. If not set, 10 is a reasonable number. The default value is 2.

FILES = <number>
  Where the number is between 1 and 99. This is the number of open files that the XENIX system calls can access. It is in-

stallation-dependent. If not set, 10 is a reasonable number. The default value is 8.

DEVICE = <filename>

This installs the device driver in <filename> into the system list. (See below.)

BREAK = <ON or OFF>

If ON is specified (the default is OFF), a check for CONTROL-C as input will be made every time the system is called. ON improves the ability to abort programs over previous versions of the MS-DOS. BREAK ON/OFF may be changed by issuing a BREAK command (see Chapter 5).

SHELL = <filename>

This begins execution of the shell (top-level command processor) from <filename>.

A typical configuration file might look like this:

```
    Buffers = 10
    Files   = 10
    Device  = \BIN\NETWORK.SYS
    Break   = ON
    Shell   = A:\BIN\COMMAND.COM A:\BIN /P
```

Note here that the Buffers and Files parameters are set to 10x. The system initialization routine will search for the filename \BIN\NETWORK.SYS to find the device that is being added to the system. This file is usually supplied on disk with your device. Make sure that you save the device file in the pathname that you specify in the Device parameter.

This configuration file also sets the MS-DOS command EXEC to the COMMAND.COM file located on disk A: in the \BIN directory. The A:\BIN tells COMMAND.COM where to look for itself when it needs to be re-read from disk. The /P tells COMMAND.COM that it is the first program running on the system so that it can process the MS-DOS EXIT command.

INDEX

Supplement

Ladder and CatChum are trademarks of Yahoo Software. MS-BASIC is a registered trademark of Microsoft Corporation.

INTRODUCTION

This supplement contains descriptions of some application and demonstration software featured with your NCR DECISION MATE V.

If you have a single flexible disk drive, follow the steps below to start using any of the software. If you have two flexible disk drives, note the instructions in parentheses.

1. Insert MS-DOS. (For two flexible disk drives, insert MS-DOS in A and the supplement disk in B.)
2. The system displays A>. Insert the supplement disk. Enter the name of the application and press RETURN. (For two flexible disk drives, enter B: and press RETURN. When the system prompt B> is displayed, enter the name of the application and press RETURN.) The game instructions appear on the screen.

LADDER

LADDER is a game for use on the NCR DECISION MATE V. To start the program, enter LADDER ↵ . The main screen appears.

Enter "I" (Instructions) to obtain the description of the game. Read the explanations and then press ↵ to return to the main screen. There you will find the directions on how to play the game.

Let's now start the game by entering P. You can leave the program at any time by pressing ↑ C (hold down the CONTROL key and press C).

CATCHUM

CATCHUM is a game for one or two players. To start the program, enter CATCHUM ↵ . The main screen appears.

Enter "I" (Instructions) to obtain the description of the game. Read the explanations and then press ↵ to return to the main screen. There you will find the directions on how to play the game. Enter 1 if you are the first player, or 2 if you are the second player.

You can leave the program at any time by pressing ↑ C.

DEMO5

DEMO5 is a continuous running graphics demonstration showing the excellent resolution and speed of the NCR DECISION MATE V. To run the program, enter DEMO5 ↵ . The program can be aborted at any time with ↑ C (hold down the CONTROL key and press C).

CLOK

CLOK displays a running clock on the CRT screen. To run the program, enter CLOK ↵ . The current date is then entered in the following format:

MM,DD,YY <CR> (e.g. 4,26,83)

Next, enter the current time using the following format:

HH,MM <CR> (e.g. 12,31)

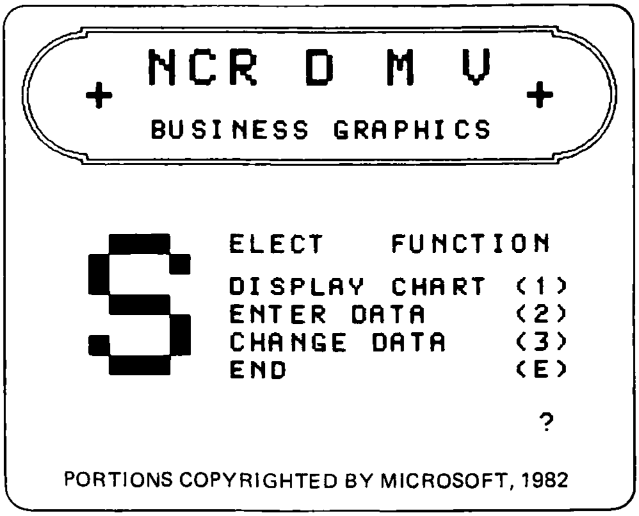
The program can be aborted at any time with ↑ C (hold down the CONTROL key and press C).

MUSIC

MUSIC plays 11 different tunes on the NCR DECISION MATE V. To run the program, enter MUSIC ↵ . The music menu is displayed and you can choose song number 1-9, A, or B. Just type the number or letter of the selection you want to hear (do not press <CR>), and the song is played. When the tune is finished, the menu is redisplayed. Choose another song or press E to end. (This program cannot be aborted with ↑ C.)

VEGAS

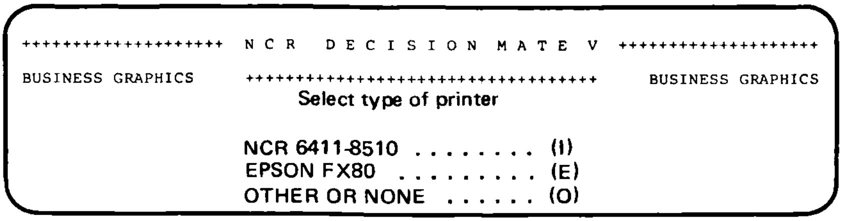
VEGAS (Very Easy Graphic Application System) is a very useful program for displaying business graphics. This program allows you to create line, bar, and pie charts and then output them to a printer. NCR Business Graphics is written in MS-BASIC® with the NCR graphics extension. To run the program, enter VEGAS ↵ .
The main menu now appears and processing may begin.



```
+ NCR D M V +
BUSINESS GRAPHICS

S ELECT    FUNCTION
  DISPLAY CHART  (1)
  ENTER DATA     (2)
  CHANGE DATA    (3)
  END            (E)

                  ?

PORTIONS COPYRIGHTED BY MICROSOFT, 1982
```

Figure 1 Main Menu

Each time VEGAS is called from the disk, the following screen is displayed after a selection (1-3) is made on the main menu. (This is the only time this screen appears during the VEGAS session.)



```
+++++++++++++++++++ N C R  D E C I S I O N  M A T E  V +++++++++++++++++++
BUSINESS GRAPHICS   ++++++++++++++++++++++++++++++++++   BUSINESS GRAPHICS
                        Select type of printer

                     NCR 6411-8510 ........ (I)
                     EPSON FX80 ......... (E)
                     OTHER OR NONE ...... (O)
```

Figure 1a Printer Definition

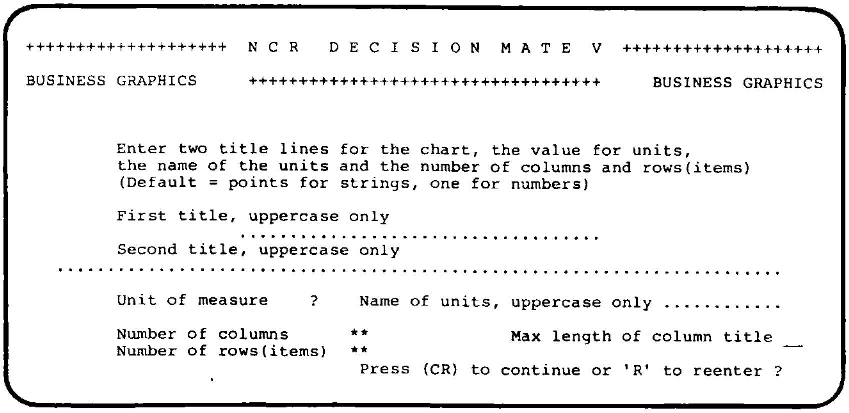
Enter I if you are using an NCR 6411-8510 (ITOH M8510A) printer; E if you have an EPSON MX82; and O if you have another type of printer or no printer at all.

Before continuing with the next section, the following general information should be noted:

- (CR) is the Carriage Return key (↵)
- For questions that require only a 1-character response, (CR) should not be pressed after the character is entered.
- After all data has been entered on most screens, the message PRESS (CR) TO CONTINUE OR "R" TO REENTER? is displayed. R allows you to return to the top of the screen and reenter all information.
- Rows (Items) refer to the segments or parts in each column.
- Current Chart refers to the chart last used.

ENTERING DATA

The second choice on the main menu (Figure 1) is Enter Data. This selection allows you to enter information to create a new graph. Enter 2 to choose this function.



```
+++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


          Enter two title lines for the chart, the value for units,
          the name of the units and the number of columns and rows(items)
          (Default = points for strings, one for numbers)

          First title, uppercase only
                           ....................................
          Second title, uppercase only
    .........................................................................

          Unit of measure     ?    Name of units, uppercase only ...........

          Number of columns      **               Max length of column title __
          Number of rows(items)  **
                                   Press (CR) to continue or 'R' to reenter ?
```

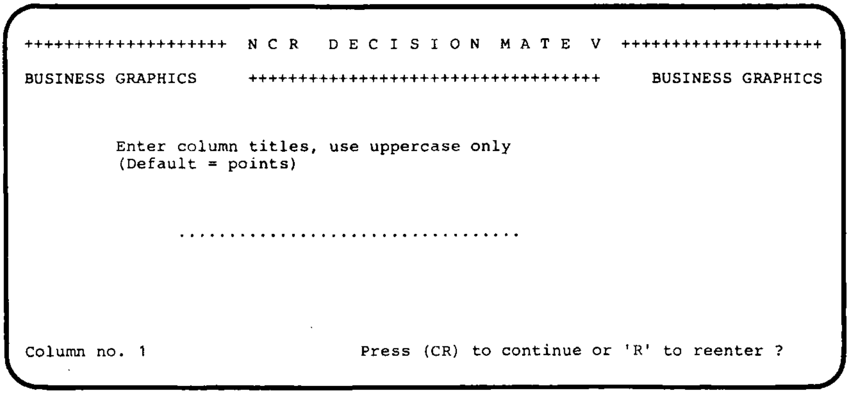
Figure 2

To begin, each question on the screen is displayed one at a time and the following information must be entered.

First Title:	uppercase only, 36 chars. max, no comma
Second Title:	uppercase only, 72 chars. max, no comma
Unit of Measure:	0-9, default = 0 (0 = units, 1 = tens, 2 = hundreds, 3 = thousands, etc.)
Name of Units:	uppercase only, 12 chars. max (e.g. DOLLARS, MACHINES, etc.)
Number of Columns:	48 maximum, default = 1
Number of Rows (Items):	12 maximum, default = 1

A response of (CR) only to the above questions enters the default value. For questions that require a string response the default is points (. . . . .). When all information on the screen is completed, press (CR) to continue.

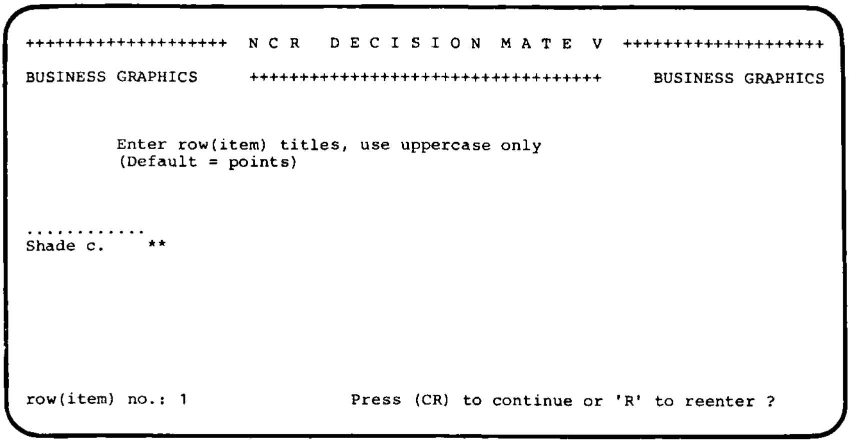
The next type of data that must be entered is column titles: uppercase only, 59 characters maximum, default = points, see Figure 3. (The maximum number of characters for the titles may decrease, depending on the number of columns in your graph.) After each title is entered, press (CR).



```
++++++++++++++++++++  N C R   D E C I S I O N   M A T E   V  ++++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


         Enter column titles, use uppercase only
         (Default = points)


               ...................................




Column no. 1                        Press (CR) to continue or 'R' to reenter ?
```

Figure 3

Next, information about the rows is entered (see Figure 4).



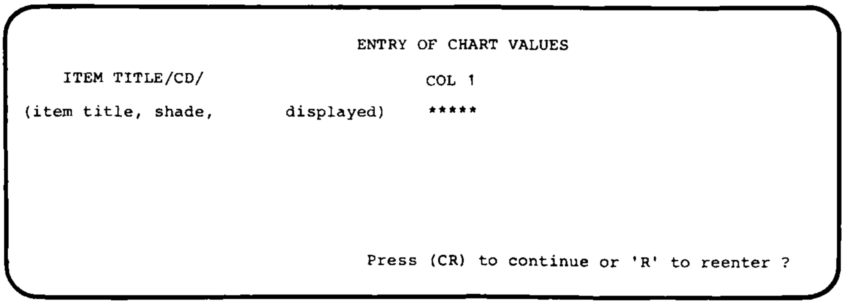
```
++++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  ++++++++++++++++++++
BUSINESS GRAPHICS     ++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


         Enter row(item) titles, use uppercase only
         (Default = points)


............
Shade c.     **






row(item) no.: 1                Press (CR) to continue or 'R' to reenter ?
```

Figure 4

Row Titles: uppercase only, 12 chars. maximum The maximum number may decrease, depending on the number of rows.

Shade: 0-15, default = 0 (Refer to the end of this section for the shading patterns.)

The values for every item in each column are entered next. The item title and shade are displayed, and asterisks appear where you enter the value. See Figure 5.

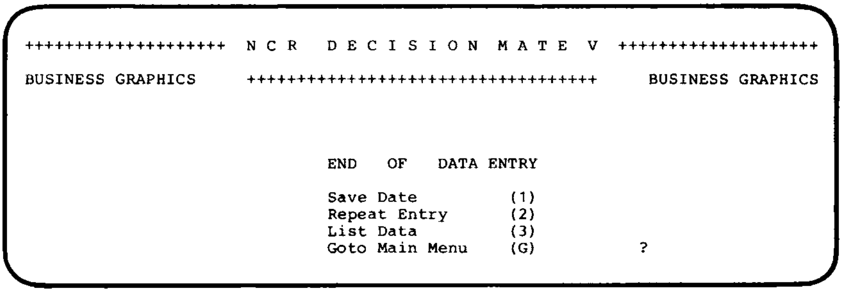


```
                                ENTRY OF CHART VALUES

    ITEM TITLE/CD/                     COL 1

(item title, shade,        displayed)   *****




                                 Press (CR) to continue or 'R' to reenter ?
```

Figure 5

Values:	The value for each row is entered, integers only. No integer larger than 32760 may be entered.

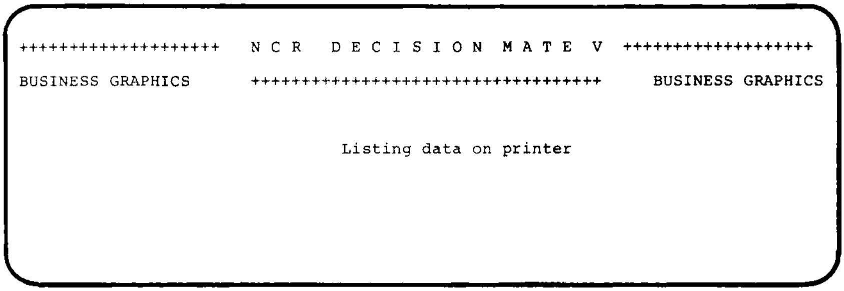
When all of the above information is entered the end of data entry menu is displayed. At this point you have the chance to save the data, reenter all of the information once again, print a listing of the data or return to the main menu. See Figure 6.



```
++++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  ++++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS



                               END   OF   DATA ENTRY

                               Save Date        (1)
                               Repeat Entry     (2)
                               List Data        (3)
                               Goto Main Menu   (G)          ?
```

Figure 6

It is a good idea to list the data first so that it can be checked. If you choose to do this, the following screen is displayed.

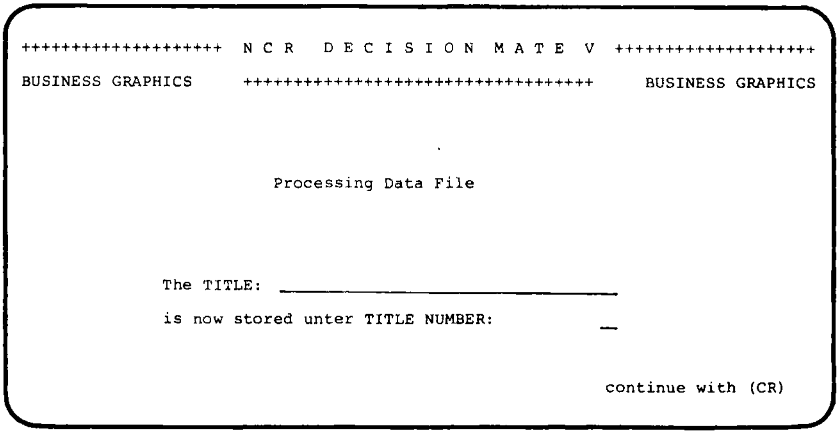


```
++++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                                Listing data on printer
```

Figure 7

When the listing is finished, the end of data entry menu is redisplayed. Now, save the data or, if mistakes were made, reenter the information. (If you choose to reenter the data, all of the information must be repeated. If only 1 or 2 fields must be changed, it is better to save the data and then select Function 3 on the main menu, Change Data. Refer to the section on changing data for the fields that can be changed with that function.)

As the data is saved on the disk, the following question is displayed: REPLACE(R) LAST RECORD IN FILE OR WRITE NEW(N) RECORDS? To delete the last record in the file and replace it with the current information, enter R. To add a new record to the file, enter N. The message: PRESS (CR) TO CONTINUE OR "R" TO REENTER? is displayed next. If you made a mistake, press R, otherwise, press (CR) to continue. The program now assigns a number to this record and displays it on the screen. See Figure 8.



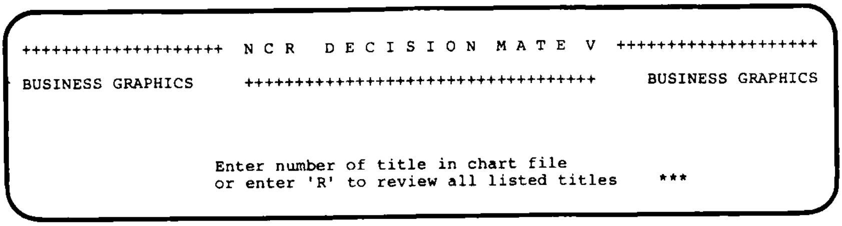
```
++++++++++++++++++++  N C R   D E C I S I O N   M A T E   V  ++++++++++++++++++++
BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                          Processing Data File


                 The TITLE:  _________________________
                 is now stored unter TITLE NUMBER:   _

                                                        continue with (CR)
```

Figure 8

Note this number and press (CR) to continue. The end of data entry menu is redisplayed. You are now ready to select G and return to the main menu to choose another function.

DISPLAYING CHARTS

The first choice on the main menu is Display Chart (refer to Figure 1). This selection allows you to display and print line, bar, and pie charts. Enter 1 to choose this function. Either the question USE CURRENT CHART ? (Y/N) or the following screen is displayed.

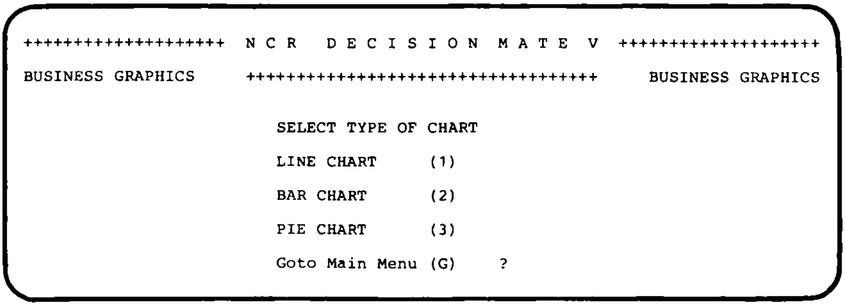


```
++++++++++++++++++++ N C R   D E C I S I O N   M A T E   V ++++++++++++++++++++

BUSINESS GRAPHICS    +++++++++++++++++++++++++++++++++++    BUSINESS GRAPHICS



                     Enter number of title in chart file
                     or enter 'R' to review all listed titles    ***
```

Figure 9

If you are not going to use the current chart, you must enter the number assigned to the one you want to use. To obtain a listing of all titles and numbers, enter R and press (CR). (For this version of VEGAS, a demonstration chart, number 2, is saved on the disk.) When you have found the number of your chart press (CR) to continue and the main menu is redisplayed. Then select Function 1 once again and enter the title number.

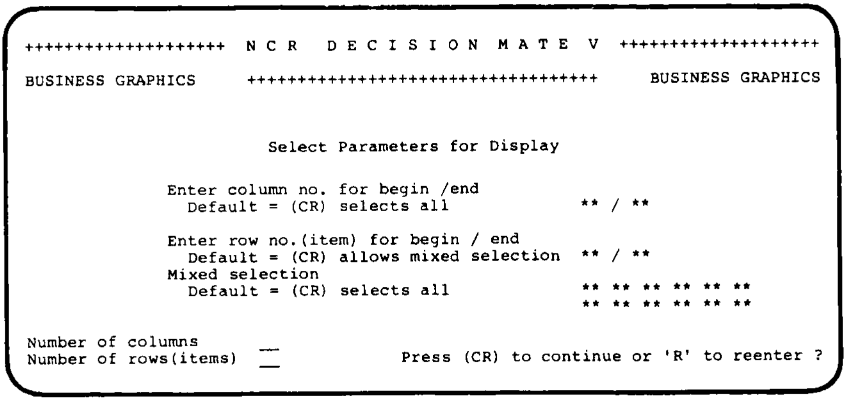
Once the title number is entered, a short menu appears and you choose the type of graph you want to see. Refer to Figure 10.



```
++++++++++++++++++++ N C R   D E C I S I O N   M A T E   V ++++++++++++++++++++

BUSINESS GRAPHICS    +++++++++++++++++++++++++++++++++++    BUSINESS GRAPHICS

                          SELECT TYPE OF CHART

                          LINE CHART      (1)

                          BAR CHART       (2)

                          PIE CHART       (3)

                          Goto Main Menu  (G)     ?
```

Figure 10

Before a line graph is displayed, certain parameters must be entered. See Figure 11.



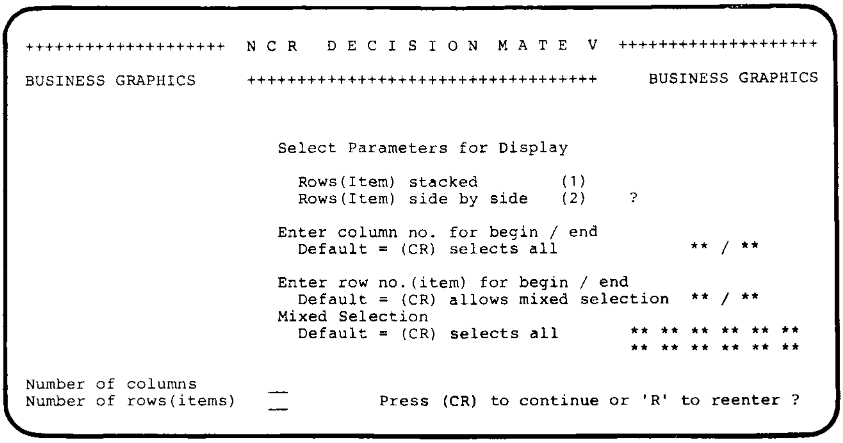
```
++++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  ++++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                        Select Parameters for Display

              Enter column no. for begin /end
                Default = (CR) selects all              ** / **

              Enter row no.(item) for begin / end
                Default = (CR) allows mixed selection   ** / **
              Mixed selection
                Default = (CR) selects all              ** ** ** ** ** **
                                                        ** ** ** ** ** **

Number of columns       __
Number of rows(items)   __            Press (CR) to continue or 'R' to reenter ?
```

Figure 11

First, enter the number of the columns you want to display. For example, if your chart has 4 columns, you may want to see only columns 3 and 4 on the graph. Pressing (CR) only in response to this question selects all columns.

Next, enter the row numbers to appear on the chart. For example, if each column has 3 different parts to it, you can choose to display 1, 2, or all of them. A response of (CR) selects all rows and also allows you to change the order in which the rows are displayed (mixed selection). For example, you can display row 3 first, row 2, and then row 1. The default response of (CR) displays the rows in the order in which they were originally entered. When all information has been entered, press (CR) to continue and the chart is displayed.

Before displaying a bar chart, select the type of chart you want – bars stacked, or side by side. Then enter the columns and rows to be displayed. See Figure 12.

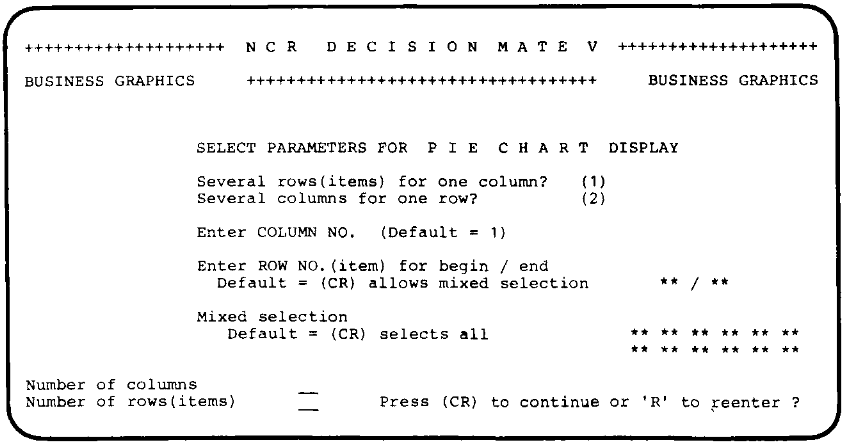


```
+++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS    ++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                         Select Parameters for Display

                           Rows(Item) stacked        (1)
                           Rows(Item) side by side   (2)     ?

                         Enter column no. for begin / end
                           Default = (CR) selects all                ** / **

                         Enter row no.(item) for begin / end
                           Default = (CR) allows mixed selection   ** / **
                         Mixed Selection
                           Default = (CR) selects all      ** ** ** ** ** **
                                                           ** ** ** ** ** **

Number of columns      __
Number of rows(items)  __        Press (CR) to continue or 'R' to reenter ?
```

Figure 12

When all information has been entered, press (CR) to continue and the chart is displayed.

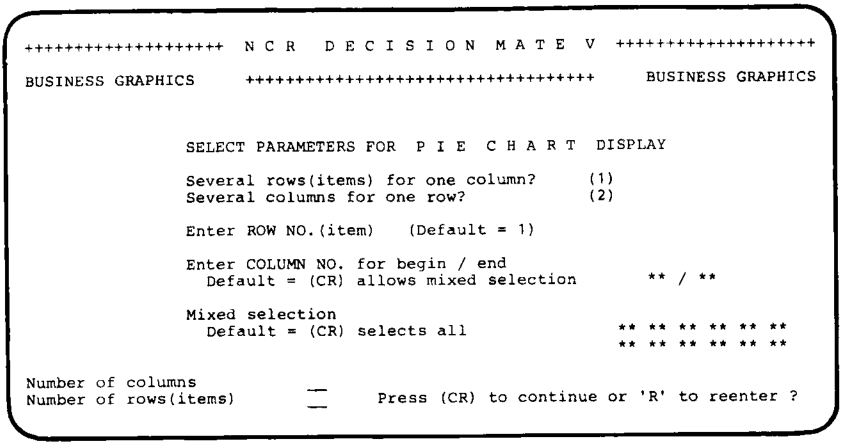
A pie chart can be divided into a maximum of 12 pieces. To begin, choose what to display – one column or one row covering several columns. If you select one column, you must enter the column number and how many rows within the column should be displayed. See Figure 13.



```
+++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS    ++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                 SELECT PARAMETERS FOR  P I E  C H A R T  DISPLAY

                 Several rows(items) for one column?   (1)
                 Several columns for one row?          (2)

                 Enter COLUMN NO.  (Default = 1)

                 Enter ROW NO.(item) for begin / end
                   Default = (CR) allows mixed selection       ** / **

                 Mixed selection
                    Default = (CR) selects all         ** ** ** ** ** **
                                                       ** ** ** ** ** **

Number of columns        __
Number of rows(items)    __     Press (CR) to continue or 'R' to reenter ?
```

Figure 13

If you choose to display one row covering several columns, enter the row number and column numbers. As with the line and bar charts, you may display the column or the rows in any order (mixed selection). See Figure 14.



```
+++++++++++++++++++  NCR  DECISION  MATE  V  +++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                SELECT PARAMETERS FOR  PIE  CHART  DISPLAY

                Several rows(items) for one column?      (1)
                Several columns for one row?             (2)

                Enter ROW NO.(item)    (Default = 1)

                Enter COLUMN NO. for begin / end
                  Default = (CR) allows mixed selection         ** / **

                Mixed selection
                  Default = (CR) selects all                 ** ** ** ** ** **
                                                             ** ** ** ** ** **

Number of columns            _
Number of rows(items)        _       Press (CR) to continue or 'R' to reenter ?
```

Figure 14

The pie chart displays a legend to the right of the pie. This legend tells you the title of the row or column, the value, and the percent of the pie for each value. When pieces of the pie are less than 1.5%, they are collected and displayed as one piece labeled MISC. However, the legend lists each individual value separately.

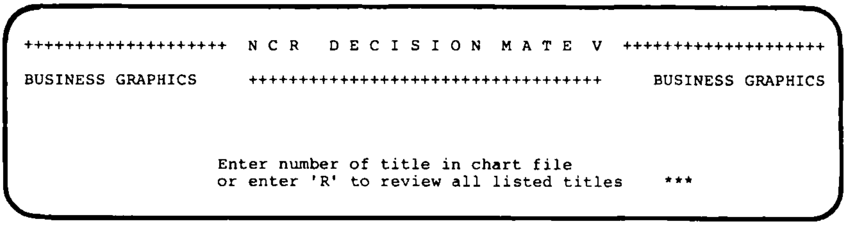
In the lower left corner of the CRT a few additional features are listed when the chart is displayed. E ends the display and returns to the chart menu. C switches the screen between inverse video (dark on green background) and the normal mode (green on dark background). P or D prints the chart on your printer. (D prints in double density mode; the characters are printed twice so that it comes out darker.)

To print charts with black printing on a white background, change to the normal mode and then plot it.

In addition, with pie charts you can choose to "slice" the pie. This is done by making sure the chart is in the inverse video mode, then select the number of the slice and press that key. All slices can be separated, but cannot be brought back together again while the chart is displayed on the screen.

CHANGING DATA

The third choice on the main menu is Change Data (refer to Figure 1). This selection allows you to update the information for a chart that is already saved on the disk. Enter 3 to choose this function. Either the question USE CURRENT CHART ? (Y/N) or the following screen is displayed.

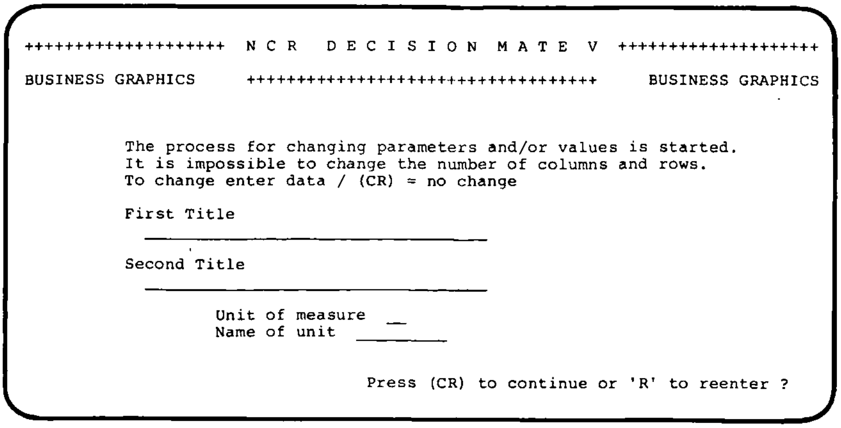


```
++++++++++++++++++++  N C R   D E C I S I O N  M A T E  V  ++++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS



                   Enter number of title in chart file
                   or enter 'R' to review all listed titles    ***
```

Figure 15

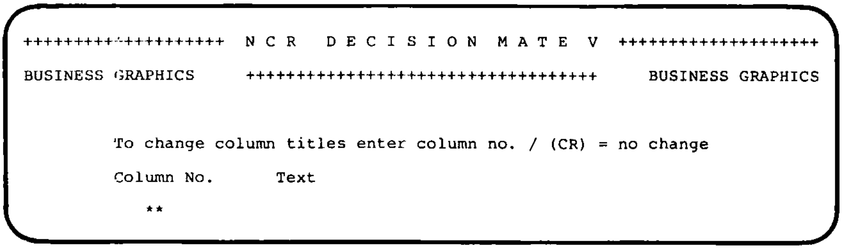
If you are not going to use the current chart, you must enter the number assigned to the one you want to use. To obtain a listing of all titles and numbers, enter R and press (CR). When you have found the number of your chart, press (CR) to continue and the main menu is redisplayed. Then select function 3 once again and enter the title number.

The data for your chart is displayed one field at a time and you now have the option of changing it. If no change is necessary, press (CR) and the cursor moves to the next field. The only information that cannot be changed is the number of columns and rows and, if the unit of measure is changed, all values must be altered to match it. Refer to the following screens for the order in which the information is displayed.



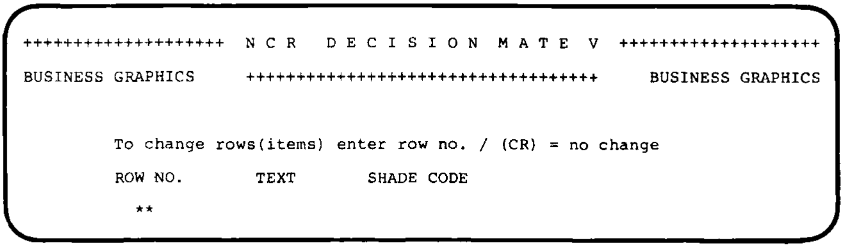
```
+++++++++++++++++++  N C R   D E C I S I O N   M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


          The process for changing parameters and/or values is started.
          It is impossible to change the number of columns and rows.
          To change enter data / (CR) = no change

          First Title
            __________________________________

          Second Title
            __________________________________

                   Unit of measure  __
                   Name of unit   _________

                                  Press (CR) to continue or 'R' to reenter ?
```

Figure 16



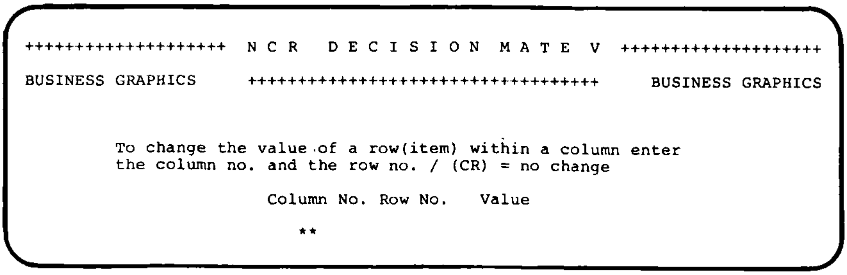
```
+++++++++++++++++++  N C R   D E C I S I O N   M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


         To change column titles enter column no. / (CR) = no change

         Column No.      Text

            **
```

Figure 17



```
+++++++++++++++++++  N C R   D E C I S I O N   M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


         To change rows(items) enter row no. / (CR) = no change

         ROW NO.       TEXT       SHADE CODE

           **
```

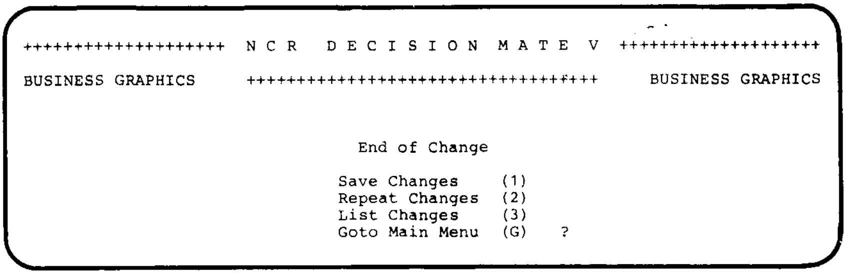
Figure 18



```
+++++++++++++++++++  N C R   D E C I S I O N   M A T E  V  +++++++++++++++++++

BUSINESS GRAPHICS     +++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


         To change the value of a row(item) within a column enter
         the column no. and the row no. / (CR) = no change

                        Column No. Row No.   Value

                          **
```

Figure 19

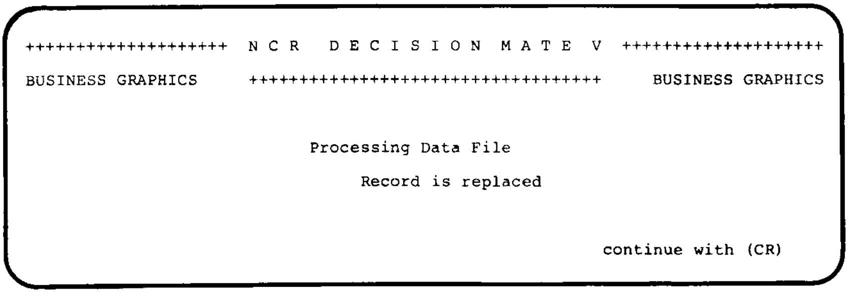
When you have finished with all data, the end of change menu (Figure 20) is displayed. List the changes to make sure that everything was input correctly, and if errors are detected, repeat the process.



```
++++++++++++++++++++ N C R  D E C I S I O N  M A T E  V ++++++++++++++++++++

BUSINESS GRAPHICS    +++++++++++++++++++++++++++++++++++    BUSINESS GRAPHICS


                              End of Change

                            Save Changes    (1)
                            Repeat Changes  (2)
                            List Changes    (3)
                            Goto Main Menu  (G)   ?
```

Figure 20

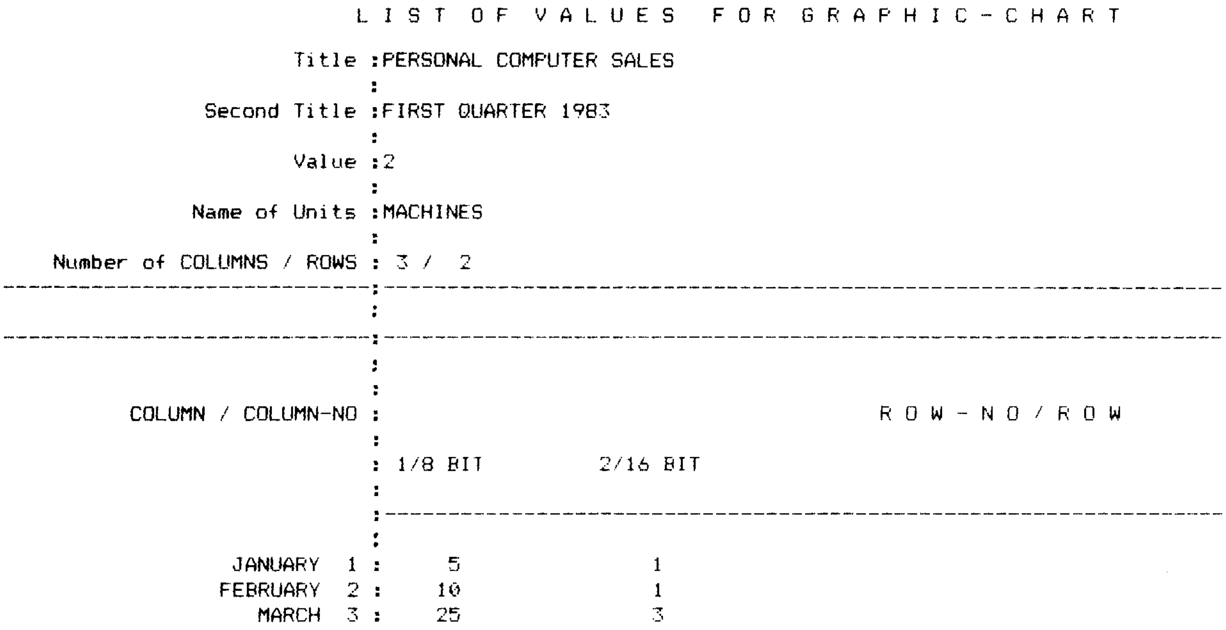
When you are satisfied that everything is correct, save the changes by selecting function 1. The following screen is displayed when the data is saved (Figure 21).



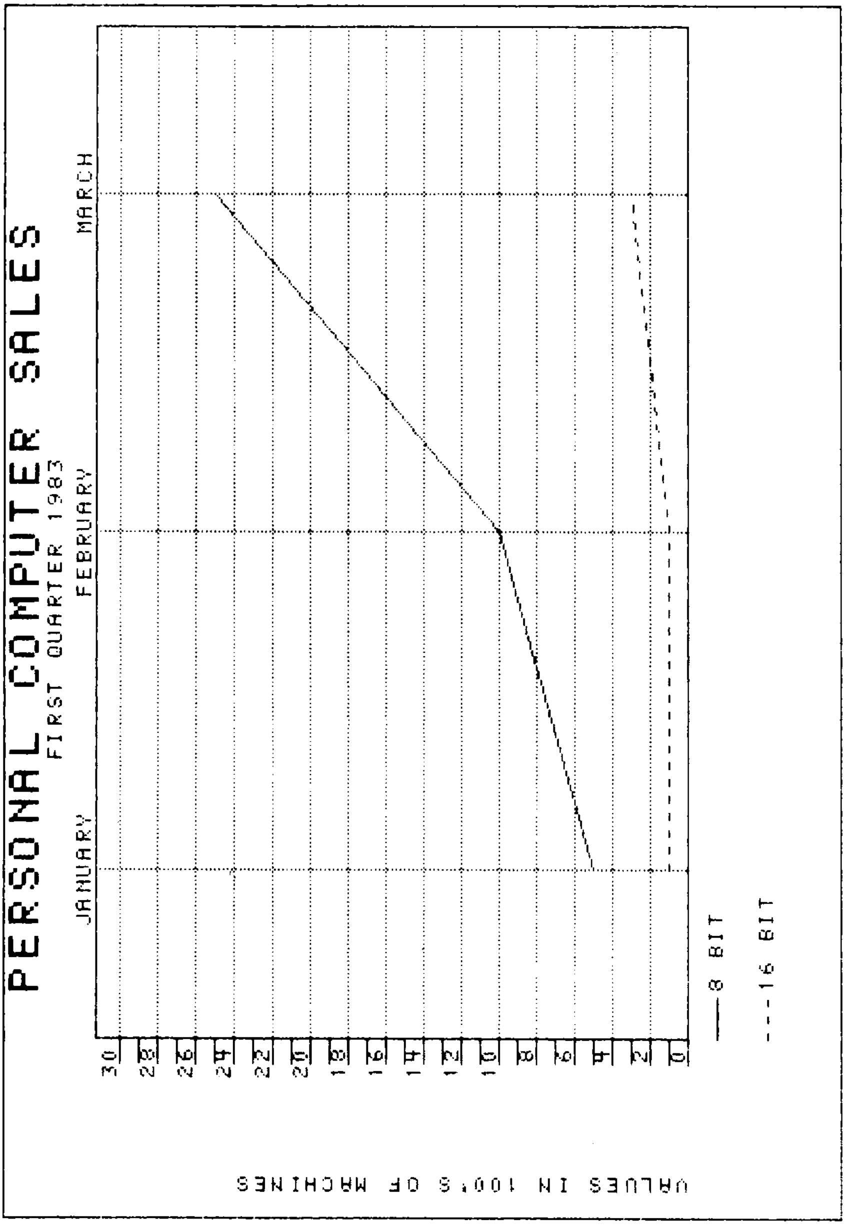
```
++++++++++++++++++++ N C R  D E C I S I O N  M A T E  V ++++++++++++++++++++

BUSINESS GRAPHICS    +++++++++++++++++++++++++++++++++++    BUSINESS GRAPHICS


                         Processing Data File

                              Record is replaced



                                                    continue with (CR)
```

Figure 21

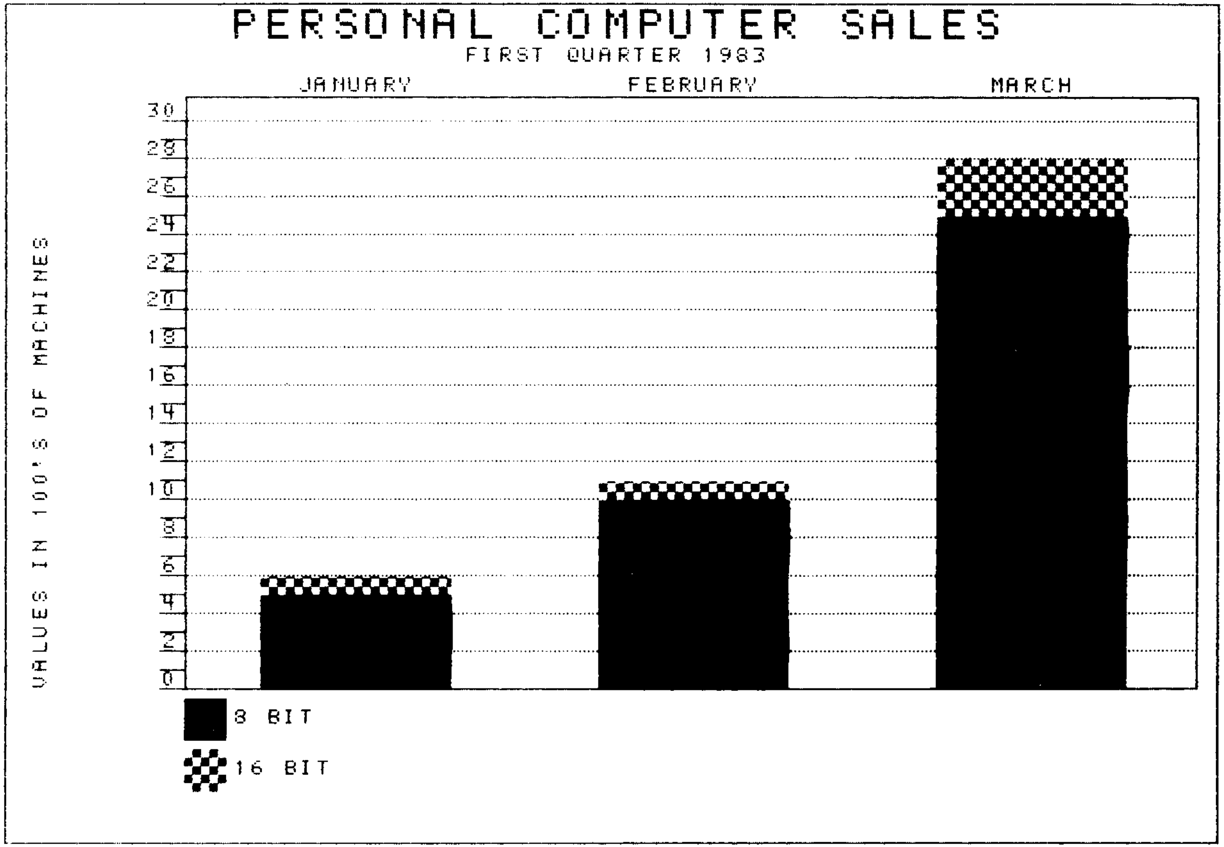
Press (CR) to continue; the end of change menu is redisplayed (see Figure 20). You are now ready to return to the main menu and select another function.



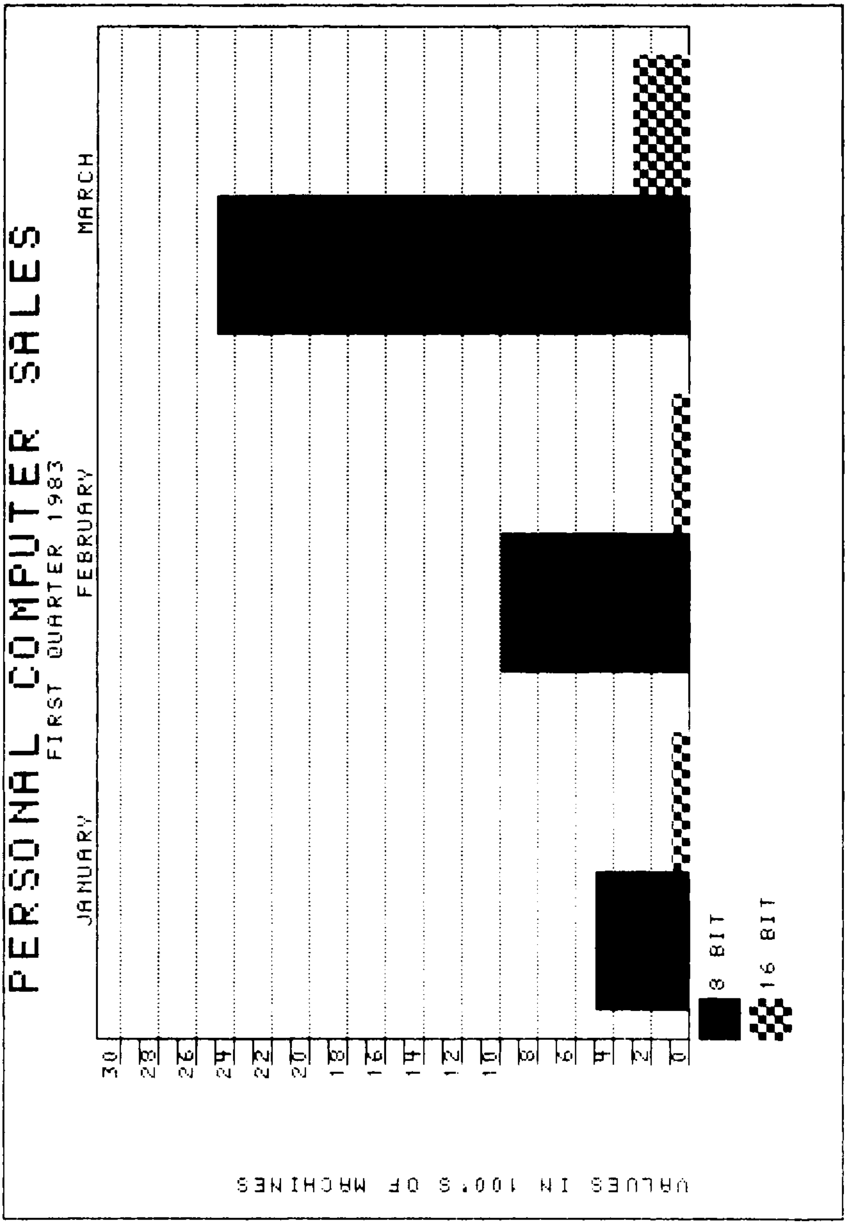
```
                    L I S T  O F  V A L U E S   F O R  G R A P H I C - C H A R T

                  Title :PERSONAL COMPUTER SALES
                        :
           Second Title :FIRST QUARTER 1983
                        :
                  Value :2
                        :
          Name of Units :MACHINES
                        :
 Number of COLUMNS / ROWS : 3 /  2
------------------------:-----------------------------------------------------------
                        :
------------------------:-----------------------------------------------------------
                        :
                        :
    COLUMN / COLUMN-NO :                                    R O W - N O / R O W
                        :
                        : 1/8 BIT          2/16 BIT
                        :
                        :-----------------------------------------------------------
                        :
          JANUARY   1 :     5               1
         FEBRUARY   2 :    10               1
            MARCH   3 :    25               3
```


PERSONAL COMPUTER SALES
FIRST QUARTER 1983
JANUARY
FEBRUARY
MARCH
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
8 BIT
16 BIT
VALUES IN 100'S OF MACHINES

Line Chart

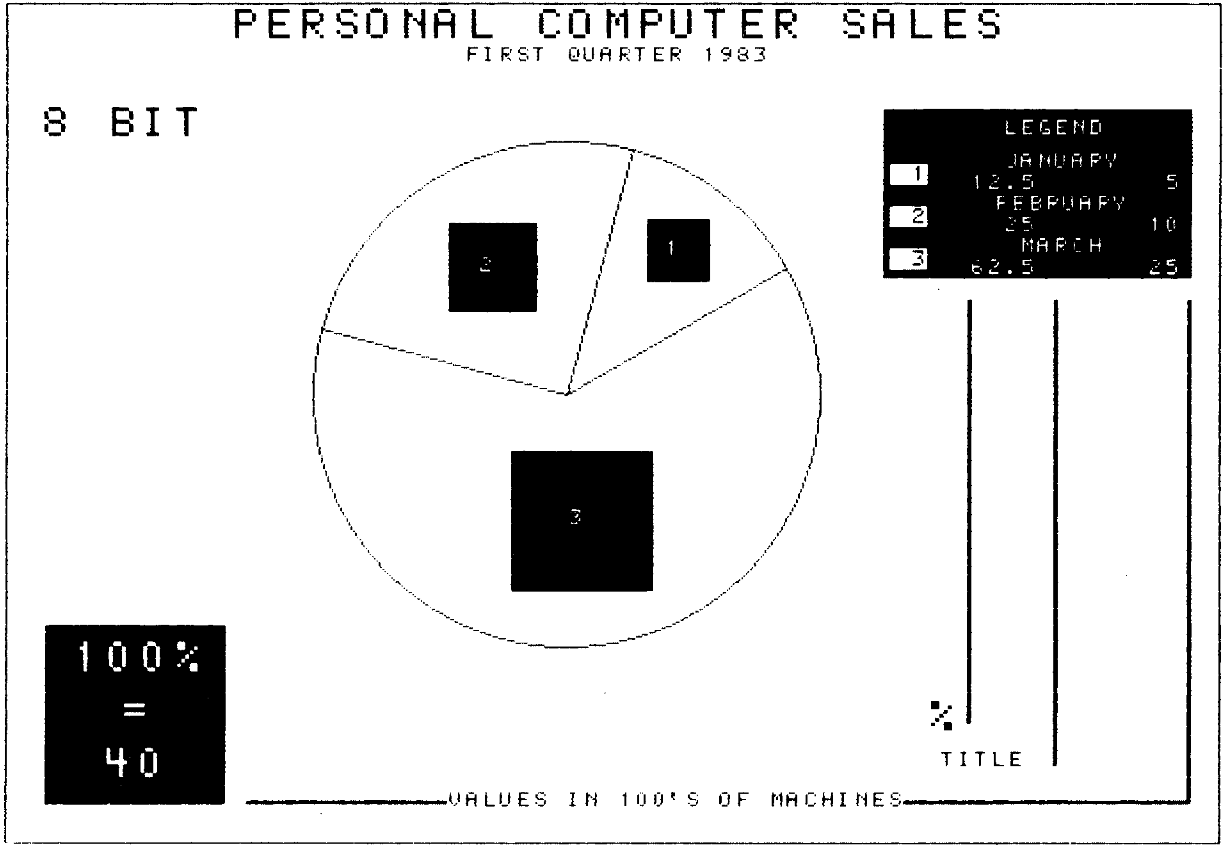


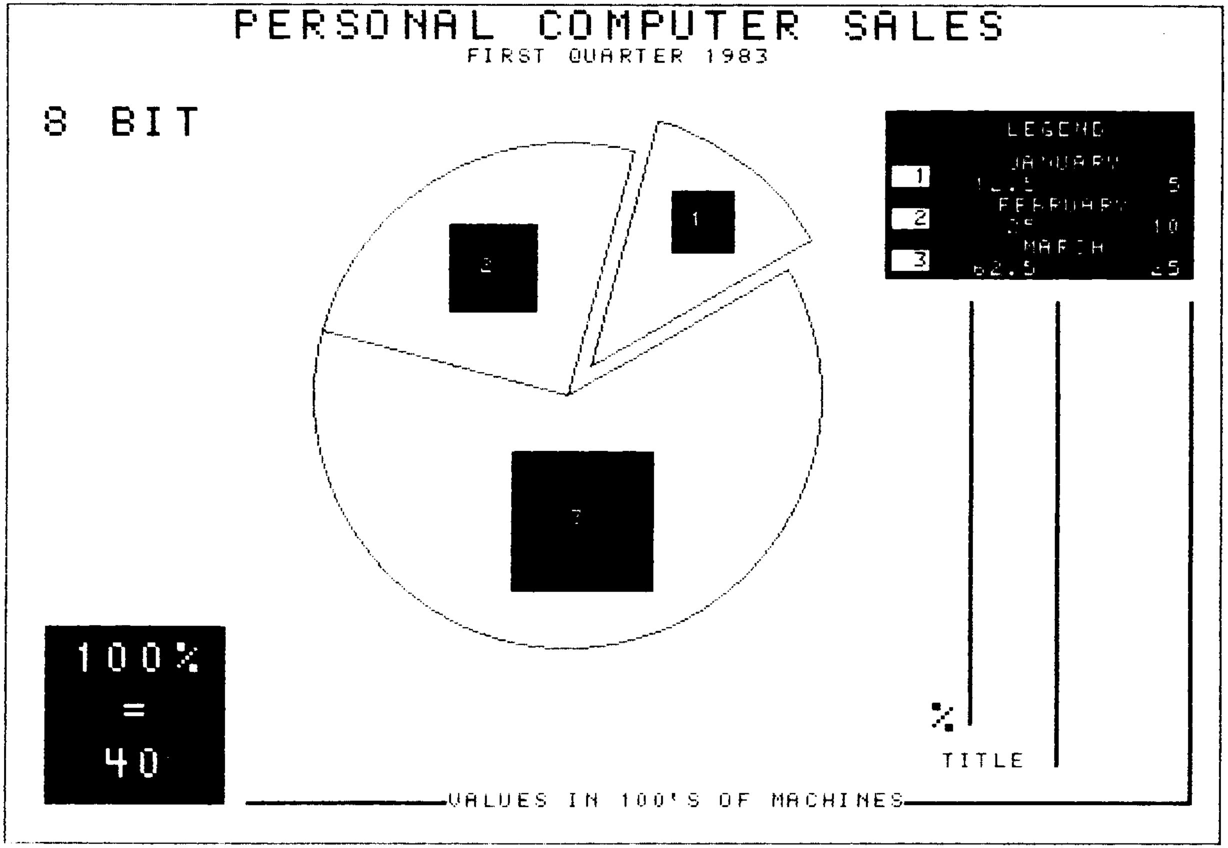
Bar Chart (stacked)



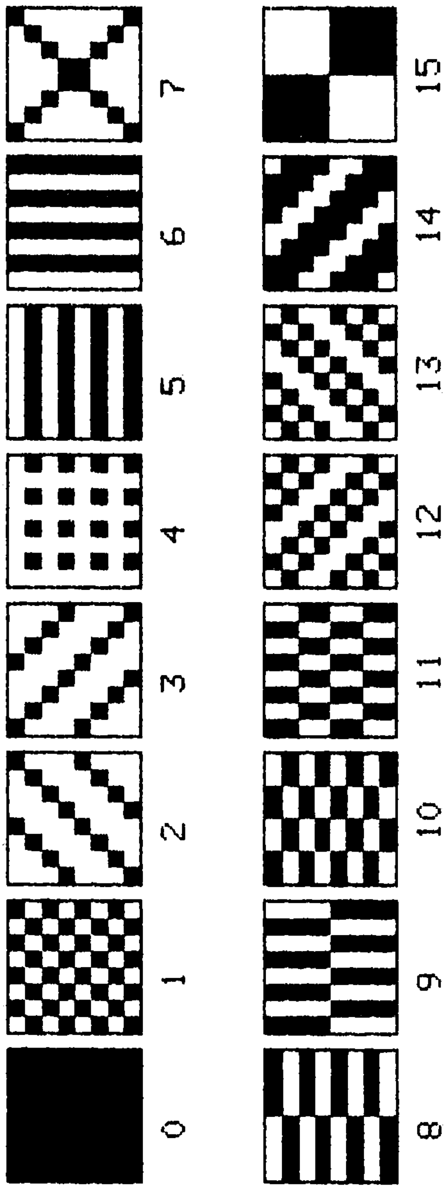
Bar Chart (side by side)

Pie Chart


PERSONAL COMPUTER SALES
FIRST QUARTER 1983
8 BIT
LEGEND
1 JANUARY 12.5 5
2 FEBRUARY 25 10
3 MARCH 62.5 25
1
2
3
100% = 40
%
TITLE
VALUES IN 100'S OF MACHINES



Pie Chart (sliced)



Shade Codes

ERROR CONDITIONS

When invalid information (for example, alphabetic characters in a numeric field) is entered into a field, VEGAS erases the information typed in and prompts for a new input. No error message is displayed.

A system error message can occur when you are saving information for a new chart. VEGAS has 2 files in which it stores all values for your charts: TITLE-F and VALUE-F. When there is no more room on the disk to expand these files with new information, the following message is displayed.

```
Disk Full at address xxxx
```

The NCR DECISION MATE V returns to the system level prompt and you choose one of 3 actions.

1. Delete the files TITLE-F and VALUE-F from your disk. (All previous charts are erased.)
2. Move the files TITLE-F and VALUE-F to another disk. (The previous information is not lost.)
3. Move all VEGAS files to another disk and use this new disk for future processing. (The previous information is not lost.)

The VEGAS files are all files on your disk that begin with the letters VE and end with the extension COM. The file BRUN.COM must also be moved. So, for example, if you are moving the VEGAS files from drive A to drive B, the PIP commands are as follows:

```
PIP B: = A:VE*.COM
PIP B: = A:BRUN.COM
```

NOTE: In this case, do not move TITLE-F and VALUE-F to the new disk. These files are automatically created by VEGAS.

NCR

NCR DECISION MATE V

MS™- DOS Programmer's Manual

First Edition, June 1983

It is the policy of NCR Corporation to improve products as new technology, components, software, and firmware become available. NCR Corporation, therefore, reserves the right to change specifications without prior notice.

All features, functions, and operations described herein may not be marketed by NCR in all parts of the world. In some instances, photographs are of equipment prototypes. Therefore, before using this document, consult your nearest dealer or NCR office for information that is applicable and current.

General Introduction

Chapter 1 System Calls

Function Requests

General Introduction

The **Microsoft (R) MS(tm)-DOS Programmer's Reference Manual** is a technical reference manual for system programmers. This manual contains a description and examples of all MS-DOS 2.0 system calls and interrupts (Chapter 1). Chapter 2, "MS-DOS 2.0 Device Drivers" contains information on how to install your own device drivers on MS-DOS. Two examples of device driver programs (one serial and one block) are included in Chapter 2. Chapter 3 through 5 contain technical information about MS-DOS, including MS-DOS disk allocation (Chapter 3), MS-DOS control blocks and work areas (Chapter 4), and EXE file structure and loading (Chapter 5).

Chapter 1
System Calls

1.1 INTRODUCTION

MS-DOS provides two types of system calls: interrupts and function requests. This chapter describes the environments from which these routines can be called, how to call them, and the processing performed by each.

1.2 PROGRAMMING CONSIDERATIONS

The system calls mean you don't have to invent your own ways to perform these primitive functions, and make it easier to write machine-independent programs.

1.2.1 Calling From Macro Assembler

The system calls can be invoked from Macro Assembler simply by moving any required data into registers and issuing an interrupt. Some of the calls destroy registers, so you may have to save registers before using a system call. The system calls can be used in macros and procedures to make your programs more readable; this technique is used to show examples of the calls.

1.2.2 Calling From A High-Level Language

The system calls can be invoked from any high-level language whose modules can be linked with assembly-language modules.

Calling from Microsoft Basic: Different techniques are used to invoke system calls from the compiler and interpreter. Compiled modules can be linked with assembly-language modules; from the interpreter, the CALL statement or USER function can be used to execute the appropriate 8086 object code.

Calling from Microsoft Pascal: In addition to linking with an assembly-language module, Microsoft Pascal includes a function (DOSXQQ) that can be used directly from a Pascal program to call a function request.

Calling from Microsoft FORTRAN: Modules compiled with Microsoft FORTRAN can be linked with assembly-language modules.

1.2.3 Returning Control To MS-DOS

Control can be returned to MS-DOS in any of four ways:

1. Call Function Request 4CH

   ```
   MOV  AH,4CH
   INT  21H
   ```

   This is the preferred method.

2. Call Interrupt 20H:

   ```
   INT  20H
   ```

3. Jump to location 0 (the beginning of the Program Segment Prefix):

   ```
   JMP  0
   ```

   Location 0 of the Program Segment Prefix contains an INT 20 H instruction, so this technique is simply one step removed from the first.

4. Call Function Request 00H:

   ```
   MOV  AH,00H
   INT  21H
   ```

   This causes a jump to location 0, so it is simply one step removed from technique 2, or two steps removed from technique 1.

1.2.4 Console And Printer Input/Output Calls

The console and printer system calls let you read from and write to the console device and print on the printer without using any machine-specific codes. You can still take advantage of specific capabilities (display attributes such as positioning the cursor or erasing the screen, printer attributes such as double-strike or underline, etc.) by using constants for these codes and reassembling once with the correct constant values for the attributes.

1.2.5 Disk I/O System Calls

Many of the system calls that perform disk input and output require placing values into or reading values from two system control blocks: the File Control Block (FCB) and directory entry.

1.3 FILE CONTROL BLOCK (FCB)

The Program Segment Prefix includes room for two FCBs at offsets 5CH and 6CH. The system call descriptions refer to unopened and opened FCBs. An **unopened** FCB is one that contains only a drive specifier and filename, which can contain wild card characters (* and ?). An **opened** FCB contains all fields filled by the Open File system call (Function 0FH). Table 1.1 describes the fields of the FCB.

Table 1.1 Fields of File Control Block (FCB)

	Size	Offset	
Name	(bytes)	Hex	Decimal
Drive number	1	00H	0
Filename	8	01-08H	1-8
Extension	3	09-0BH	9-11
Current block	2	0CH,0DH	12,13
Record size	2	0EH,0FH	14,15
File size	4	10-13H	16-19
Date of last write	2	14H,15H	20,21
Time of last write	2	16H,17H	22,23
Reserved	8	18-1FH	24-31
Current record	1	20H	32
Relative record	4	21-24H	33-36

1.3.1 Fields Of The FCB

Drive Number (offset 00H): Specifies the disk drive; 1 means drive A: and 2 means drive B:. If the FCB is to be used to create or open a file, this field can be set to 0 to specify the default drive; the Open File system call Function (0FH) sets the field to the number of the default drive.

Filename (offset 01H): Eight characters, left-aligned and padded (if necessary) with blanks. If you specify a reserved device name (such as LPT1), do not put a colon at the end.

Extension (offset 09H): Three characters, left-aligned and padded (if necessary) with blanks. This field can be all blanks (no extension).

Current Block (offset 0CH): Points to the block (group of 128 records) that contains the current record. This field and the Current Record field (offset 20H) make up the record pointer. This field is set to 0 by the Open File system call.

Record Size (offset 0EH): The size of a logical record, in bytes. Set to 128 by the Open File system call. If the record size is not 128 bytes, you must set this field after opening the file.

File Size (offset 10H): The size of the file, in bytes. The first word of this 4-byte field is the low-order part of the size.

Date of Last Write (offset 14H): The date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

```
Offset 15H
| Y | Y | Y | Y | Y | Y | Y | M |
 15                         9  8

Offset 14H
| M | M | M | D | D | D | D | D |
           5  4                0
```

Time of Last Write (offset 16H): The time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

```
Offset 17H
| H | H | H | H | H | M | M | M |
 15              11  10

Offset 16H
| M | M | M | S | S | S | S | S |
           5  4                0
```

Reserved (offset 18H): These fields are reserved for use by MS-DOS.

Current Record (offset 20H): Points to one of the 128 records in the current block. This field and the Current Block field (offset 0CH) make up the record pointer. This field is **not** initialized by the Open File system call. You must set it before doing a sequential read or write to the file.

Relative Record (offset 21H): Points to the currently selected record, counting from the beginning of the file (starting with 0). This field is **not** initialized by the Open File system call. You must set it before doing a random read or write to the file. If the record size is less than 64 bytes, both words of this field are used; if the record size ist 64 bytes or more, only the first three bytes are used.

NOTE

If you use the FCB at offset 5CH of the Program Segment Prefix, the last byte of the Relative Record field is the first byte of the unformatted parameter area that starts at offset 80H. This is the default Disk Transfer Address.

1.3.2 Extended FCB

The Extended File Control Block is used to create or search for directory entries of files with special attributes. It adds the following 7-byte prefix to the FCB:

Name	Size (bytes)	Offset (Decimal)
Flag byte (255, or FFH)	1	-7
Reserved	5	-6
Attribute byte: 02H = Hidden file 04H = System file	1	-1

1.3.3 Directory Entry

A directory contains one entry for each file on the disk. Each entry is 32 bytes; Table 1.2 describes the fields of an entry.

Table 1.2 Fields of Directory Entry

	Size	Offset	
Name	(bytes)	Hex	Decimal
Filename	8	00-07H	0-7
Extension	3	08-0AH	8-10
Attributes	1	0BH	11
Reserved	10	0C-15H	12-21
Time of last write	2	16H,17H	22,23
Date of last read	2	18H,19H	24,25
Reserved	2	1AH,1BH	26,27
File size	4	1C-1FH	28-31

1.3.4 Fields Of The FCB

Filename (offset 00H): Eight characters, left-aligned and padded (if necessary) with blanks. MS-DOS uses the first byte of this field for two special codes:

00H	(0)	End of allocated directory
E5H	(229)	Free directory entry

Extension (offset 08H): Three characters, left-aligned and padded (if necessary) with blanks. This field can be all blanks (no extension).

Attributes (offset 0BH): Attributes of the file:

Value			
Hex	Binary	Dec	Meaning
01H	0000 0001	1	Read-only
02H	0000 0010	2	Hidden
04H	0000 0100	4	System
07H	0000 0111	7	Changeable with CHGMOD
08H	0000 1000	8	Volume-ID
0AH	0001 0000	10	Directory
16H	0001 0110	22	Hard attributes for FINDENTRY
20H	0010 0000	32	Archive

Reserved (offset 0CH): Reserved for MS-DOS.

Time of Last Write (offset 16H): The time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

```
Offset 17H
| H | H | H | H | H | M | M | M |
 15              11 10

Offset 16H
| M | M | M | S | S | S | S | S |
           5 4                 0
```

Date of Last Write (offset 18H): The date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

Offset 19H

```
| Y | Y | Y | Y | Y | Y | Y | M |
 15                        9 8
```

Offset 18H

```
| M | M | M | D | D | D | D | D |
           5 4                 0
```

File Size (offset 1CH): The size of the file, in bytes. The first word of this 4-byte field is the low-order part of the size.

1.4 SYSTEM CALL DESCRIPTIONS

Many system calls require that parameters be loaded into one or more registers before the call is issued; most calls return information in the registers (usually a code that describes the success or failure of the operation). The description of system calls 00H-2EH includes the following:

A drawing of the 8088 registers that shows their contents before and after the system call.

A more complete description of the register contents required before the system call.

A description of the processing performed.

A more complete description of the register contents after the system call.

An example of its use.

The description of system calls 2FH-57H includes the following:

A drawing of the 8088 registers that shows their contents before and after the system call.

A more complete description of the register contents repuired before the system call.

A description of the processing performed.

Error returns from the system call.

An example of its use.

Figure 1 is an example of how each system call is described. Function 27H, Random Block Read, is shown.

Call
AH = 27H
DS:DX
  Opened FCB
CX
  Number of blocks to read

Return
AL
  0 = Read completed successfully
  1 = EOF
  2 = End of segment
  3 = EOF, partial record
CX
  Number of blocks read

Figure 1. Example of System Call Description

1.4.1 Programming Examples

A macro is defined for each system call, then used in some examples. In addition, a few other macros are defined for use in the examples. The use of macros allows the examples to be more complete programs, rather than isolated uses of the system calls. All macro definitions are listed at the end of the chapter.

The examples are not intended to represent good programming practice. In particular, error checking and good human interface design have been sacrificed to conserve space. You may, however, find the macros a convenient way to include system calls in your assembly language programs.

A detailed description of each system call follows. They are listed in numeric order; the interrupts are described first, then the function requests.

NOTE

Unless otherwise stated, all numbers in the system call descriptions - both text and code - are in hex.

1.5 XENIX COMPATIBLE CALLS

MS-DOS 2.0 supports hierarchical (i.e., tree-structured) directories, similar to those found in the Xenix operating system. (For information on tree-structured directories, refer to the **MS-DOS User's Guide.**)

The following system calls are compatible with the Xenix system:

Function 39H	Create Sub-Directory
Function 3AH	Remove a Directory Entry
Function 3BH	Change the Current Directory
Function 3CH	Create a File
Function 3DH	Open a File
Function 3FH	Read From File/Device
Function 40H	Write to a File or Device
Function 41H	Delete a Directory Entry
Function 42H	Move a File Pointer
Function 43H	Change Attributes
Function 44H	I/O Control for Devices
Function 45H	Duplicate a File Handle
Function 46H	Force a Duplicate of a Handle
Function 4BH	Load and Execute a Program
Function 4CH	Terminate a Process
Function 4DH	Retrieve Return Code of a Child

There is no restriction in MS-DOS 2.0 on the depth of a tree (the length of the longest path from root to leaf) except in the number of allocation units available. The root directory will have a fixed number of entries (64 for the single sided disk). For non-root directories, the number of files per directory is only limited by the number of allocation units available.

Pre-2.0 disks will appear to MS-DOS 2.0 as having only a root directory with files in it and no subdirectories.

Implementation of the tree structure is simple. The root directory is the pre-2.0 directory. Subdirectories of the root have a special attribute set indicating that they are directories. The subdirectories themselves are files, linked through the FAT as usual. Their contents are identical in character to the contents of the root directory.

Pre-2.0 programs that use system calls not described in this chapter will be unable to make use of files in other directories. Those files not necessary for the current task will be placed in other directories.

Attributes apply to the tree-structured directories in the following manner:

Attribute	Meaning/Function for files	Meaning/Function for directories
volume-id	Present at the root. Only one file may have this set.	Meaningless.
directory	Meaningless.	Indicates that the directory entry is a directory. Cannot be changed with 43H.
read-only	Old fcb-create, new Create, new open (for write or read/write) will fail.	Meaningless.
archive	Set when file is written. Set/reset via Function 43H.	Meaningless.
hidden/ system	Prevents file from being found in search first/search next. Old open will fail.	Prevents directory entry from being found. Function 3BH will still work.

1.6 INTERRUPTS

MS-DOS reserves interrupts 20H through 3FH for its own use. The table of interrupt routine addresses (vectors) is maintained in locations 80H-FCH. Table 1.3 lists the interrupts in numeric order; Table 1.4 lists the interrupts in alphabetic order (of the description). User programs should only issue Interrupts 20H, 21H, 25H, 26H, and 27H. (Function Requests 4CH and 31H are the preferred method for Interrupts 20H and 27H for versions of MS-DOS that are 2.0 and higher.)

NOTE

> Interrupts 22H, 23H, and 24H are not interrupts that can be issued by user programs; they are simply locations where a segment and offset address are stored.

Table 1.3 MS-DOS Interrupts, Numeric Order

Interrupt		
Hex	Dec	Description
20H	32	Program Terminate
21H	33	Function Request
22H	34	Terminate Address
23H	35	<CTRL-C> Exit Address
24H	36	Fatal Error Abort Address
25H	37	Absolute Disk Read
26H	38	Absolute Disk Write
27H	39	Terminate But Stay Resident
28-40H	40-64	RESERVED - DO NOT USE

Table 1.4 MS-DOS Interrupts, Alphabetic Order

	Interrupt	
Description	Hex	Dec
Absolute Disk Read	25H	37
Absolute Disk Write	26H	38
<CTRL-C>Exit Address	23H	35
Fatal Error Abort Address	24H	36
Function Request	21H	33
Program Terminate	20H	32
RESERVED - DO NOT USE	28-40H	40-64
Terminate Address	22H	34
Terminate But Stay Resident	27H	39

Program Terminate (Interrupt 20H)

Call
CS
Segment address of Program Segment Prefix

Return
None

Interrupt 20H causes the current process to terminate and returns control to its parent process. All open file handles are closed and the disk cache is cleaned. This interrupt is almost always used in old .COM files for termination.

The CS register must contain the segment address of the Program Segment Prefix before you call this interrupt.

The following exit addresses are restored from the Program Segment Prefix:

Exit Address	Offset
Program Terminate	0AH
CONTROL-C	0EH
Critical Error	12H

All file buffers are flushed to disk.

NOTE

> Close all files that have changed in length before issuing this interrupt. If a changed file is not closed, its length is not recorded correctly in the directory. See Functions 10H and 3EH for a description of the Close File system calls.

Interrupt 20H is provided for compatibility with versions of MS-DOS prior to 2.0. New programs should use Function Request 4CH, Terminate a Process.

```
Macro Definition: terminate macro
                            int 20H
                            endm
```

Example

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
        INT 20H
;There is no return from this interrupt
```

Function Request (Interrupt 21H)

Call
AH
  Function number
Other registers as specified in individual function

Return
As specified in individual function

The AH register must contain the number of the system function. See Section 1.7. ”Function Requests”, for a description of the MS-DOS system functions.

NOTE

> No macro is defined for this interrupt, because all function descriptions in this chapter that define a macro include Interrupt 21H.

Example
To call the Get Time function:

```
mov  ah,2CH     ;Get Time is Function 2CH
int  21H        ;THIS INTERRUPT
```

Terminate Address (Interrupt 22H)
CONTROL-C Exit Address (Interrupt 23H)
Fatal Error Abort Address (Interrupt 24H)

These are not true interrupts, but rather storage locations for a segment and offset address. The interrupts are issued by MS-DOS under the specified circumstance. You can change any of these addresses with Function Request 25H (Set Vector) if you prefer to write your own interrupt handlers.

Interrupt 22H -- Terminate Address
When a program terminates, control transfers to the address at offset 0AH of the Program Segment Prefix. This address is copied into the Program Segment Prefix, from the Interrupt 22H vector, when the segment is created.

Interrupt 23H - CONTROL-C Exit Address
If the user types CONTROL-C during keyboard input or display output, control transfers to the INT 23H vector in the interrupt table. This address is copied into the Program Segment Prefix, from the Interrupt 23H vector, when the segment is created.
If the CONTROL-C routine preserves all registers, it can end with an IRET instruction (return from interrupt) to continue program execution. When the interrupt occurs, all registers are set to the value they had when the original call to MS-DOS was made. There are no restrictions on what a CONTROL-C handler can do - including MS-DOS function calls - so long as the registers are unchanged if IRET is used.
If Function 09H or 0AH (Display String of Buffered Keyboard Input) is interrupted by CONTROL-C, the three-byte sequence 03H-0DH-0AH (ETX-CR-LF) is sent to the display and the function resumes at the beginning of the next line.
If the program creates a new segment and loads a second program that changes the CONTROL-C address, termination of the second program restores the CONTROL-C address to its value before execution of the second program.

Interrupt 24H - Fatal Error Abort Address

If a fatal disk error occurs during execution of one of the disk I/O function calls, control transfers to the INT 24H vector in the vector table. This address is copied into the Program Segment Prefix, from the Interrupt 24H vector, when the segment is created.
BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved.

NOTE

> Interrupt 24H is not issued if the failure occurs during execution of Interrupt 25H (Absolute Disk Read) or Interrupt 26H (Absolute Disk Write). These errors are usually handled by the MS-DOS error routine in COMMAND.COM that retries the disk operation, then gives the user the choice of aborting, retrying the operation, or ignoring the error. The following topics give you the information you need about interpreting the error codes, managing the registers and stack, and controlling the system's response to the error in order to write your own error-handling routines.

Error Codes

When an error-handling program gains control from Interrupt 24H, the AX and DI registers can contain codes that describe the error. If Bit 7 of AH is 1, the error is either a bad image of the File Allocation Table or an error occurred on a character device. The device header passed in BP:SI can be examined to determine which case exists. If the attribute byte high order bit indicates a block device, then the error was a bad FAT. Otherwise, the error is on a character device.

The following are error codes for Interrupt 24H:

Error Code	Description
0	Attempt to write on write-protected disk
1	Unknown unit
2	Drive not ready
3	Unknown command
4	Data error
5	Bad request structure length
6	Seek error
7	Unknown media type
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure

The user stack will be in effect (the first item described below is at the top of the stack), and will contain the following from top to bottom:

IP	MS-DOS registers from
CS	issuing INT 24H
FLAGS	
AX	User registers at time of original
BX	INT 21H request
CX	
DX	
SI	
DI	
BP	
DS	
ES	
IP	From the original INT 21H
CS	from the user to MS-DOS
FLAGS	

The registers are set such that if an IRET is executed, MS-DOS will respond according to (AL) as follows:

(AL) = 0	ignore the error
= 1	retry the operation
= 2	terminate the program via INT 23H

Notes:

1. Before giving this routine control for disk errors, MS-DOS performs five retries.
2. For disk errors, this exit is taken only for errors occurring during an Interrupt 21H. It is not used for errors during Interrupts 25H or 26H.
3. This routine is entered in a disabled state.
4. The SS, SP, DS, ES, BX, CX, and DX registers must be preserved.
5. This interrupt handler should refrain from using MS-DOS function calls. If necessary, it may use calls 01H through 0CH. Use of any other call will destroy the MS-DOS stack and will leave MS-DOS in an unpredictable state.
6. The interrupt handler must not change the contents of the device header.
7. If the interrupt handler will handle errors rather than returning to MS-DOS, it should restore the application program's registers from the stack, remove all but the last three words on the stack, then issue an IRET. This will return to the program immediately after the INT 21H that experienced the error. Note that if this is done, MS-DOS will be in an unstable state until a function call higher than 0CH is issued.

Absolute Disk Read (Interrupt 25H)

```
Call
AL
  Drive number
DS:BX
  Disk Transfer Address
CX
  Number of sectors
DX
  Beginning relative sector

Return
AL
  Error code if CF = 1
FlagsL
  CF = 0 if successful
     = 1 if not successful
```

The registers must contain the following:

AL	Drive number (0 = A, 1 = B, etc.).
BX	Offset of Disk Transfer Address (from segment address in DS).
CX	Number of sectors to read.
DX	Beginning relative sector.

This interrupt transfers control to the MS-DOS BIOS. The number of sectors specified in CX is read from the disk to the Disk Transfer Address. Its requirements and processing are identical to Interrupt 26H, except data is read rather than written.

NOTE

> All registers except the segment registers are destroyed by this call. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still there upon return. (This is necessary because data is passed back in the flags.) Be sure to pop the stack upon return to prevent uncontrolled growth.

If the disk operation was successful, the Carry Flag (CF) is 0. If the disk operation was not successful, CF is 1 and AL contains the MS-DOS error code (see Interrupt 24H earlier in this section for the codes and their meaning).

Macro Definition:

```
abs-disk-read  macro   disk,buffer,num-sectors,start
               mov     al, disk
               mov     bx,offset buffer
               mov     cx,num-sectors
               mov     dh,start
               int     25H
               endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:. It uses a buffer of 32K bytes:

```
prompt         db      "Source in A, target in B",13,10
               db      "Any Key to start. $"
start          dw      0
buffer         db      64 dup (512 dup (?))   ;64 sectors
               .
               .
int-25H:       display prompt ;see Function 09H
               read-kbd       ;see Function 08H
               mov   cx,5     ;copy 5 groups of
                                ;64 sectors
copy:          push   cx      ;save the loop counter
               abs-disk-read 0,buffer,64,start   ;THIS INTERRUPT
               abs-disk-write 1,buffer,64,start   ;see INT 26H
               add start,64   ;do the next 64 sectors
               pop cx         ;restore the loop counter
               loop copy
```

Absolute Disk Write (Interrupt 26H)

Call
AL
  Drive number
DS:BX
  Disk Transfer Address
CX
  Number of sectors
DX
  Beginning relative sector

Return
AL
  Error code if CF = 1
FLAGSL
  CF = 0 if successful
    = 1 if not successful

The registers must contain the following:

AL	Drive number (0 = A, 1 = B, etc.).
BX	Offset of Disk Transfer Address (from segment address in DS).
CX	Number of sectors to write.
DX	Beginning relative sector.

This interrupt transfers control to the MS-DOS BIOS. The number of sectors specified in CX is written from the Disk Transfer Address to the disk. Its requirements and processing are identical to Interrupt 25H, except data is written to the disk rather than read from it.

NOTE

All registers except the segment registers are destroyed by this call. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still there upon return. (This is necessary because data is passed back in the flags.) Be sure to pop the stack upon return to prevent uncontrolled growth.

If the disk operation was successful, the Carry Flag (CF) is 0. If the disk operation was not successful, CF is 1 and AL contains the MS-DOS error code (see Interrupt 24H for the codes and their meaning).

Macro Definition:

```
abs-disk-write  macro   disk,buffer,num-sectors,start
                mov     al,disk
                mov     bx,offset buffer
                mov     cx,num-sectors
                mov     dh,start
                int     26H
                endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:, verifying each write. It uses a buffer of 32K bytes:

```
off             equu    0
on              equ     1
                .
                .
prompt          db      "Source in A, target in B",13,10
                db      "Any key to start. $"
start           dw      0
buffer          db      64 dup (512 dup (?))   ;64 sectors
                .
                .
int-26H:        display prompt ;see Function 09H
                read-kbd       ;see Function 08H
                verify on      ;see Function 2EH
                mov   cx,5     ;copy 5 groups of 64 sectors
copy:           push   cx      ;save the loop counter
                abs-disk-read  0,buffer,64,start   ;see INT 25H
                abs-disk-write 1,buffer,64,start   ;THIS INTERRUPT
                add start,64   ;do the next 64 sectors
                pop cx         ;restore the loop counter
                loop copy
                verify off     ;see Function 2EH
```

Terminate But Stay Resident (Interrupt 27H)

Call
CS:DX
First byte following
last byte of code

Return
None

The Terminate But Stay Resident call is used to make a piece of code remain resident in the system after its termination. Typically, this call is used in .COM files to allow some device-specific interrupt handler to remain resident to process asynchronous interrupts.

DX must contain the offset (from the segment address in CS) of the first byte following the last byte of code in the program. When Interrupt 27H is executed, the program terminates but is treated as an extension of MS-DOS; it remains resident and is not overlaid by other programs when it terminates.

This interrupt is provided for compatibility with versions of MS-DOS prior to 2.0. New programs should use Function 31H, Keep Process.

Macro Definition:

```
stay-resident  macro   last-instruc
               mov     dx,offset last-instruc
               inc     dx
               int     27H
               endm
```

Example

```
;CS must be equal to PSP values given at program start
; (ES and DS values)
  mov     DX,LastAddress
  int     27H
;There is no return from this interrupt
```

1.7 FUNCTION REQUESTS

Most of the MS-DOS function calls require input to be passed to them in registers. After setting the proper register values, the function may be invoked in one of the following ways:

1. Place the function number in AH and execute a long call to offset 50H in your Program Segment Prefix. Note that programs using this method will not operate correctly on versions of MS-DOS that are lower than 2.0.
2. Place the function number in AH and issue Interrupt 21H. All of the examples in this chapter use this method.
3. An additional method exists for programs that were written with different calling conventions. This method should be avoided for all new programs. The function number is placed in the CL register and other registers are set according to the function specification. Then, an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the MS-DOS function dispatcher. Register AX is always destroyed if this method is used; otherwise, it is the same as normal function calls. Note that this method is valid only for Function Requests 00H through 024H.

1.7.1 CP/M(R)-Compatible Calling Sequence

A different sequence can be used for programs that must conform to CP/M calling conventions:

1. Move any required data into the appropriate registers (just as in the standard sequence).
2. Move the function number into the CL register.
3. Execute an intrasegment call to location 5 in the current code segment.

This method can only be used with functions 00H through 24H that do not pass a parameter in AL. Register AX is always destroyed when a function is called in this manner.

1.7.2 Treatment Of Registers

When MS-DOS takes control after a function call, it switches to an internal stack. Registers not used to return information (except AX) are preserved. The calling program's stack must be large enough to accommodate the interrupt system - at least 128 bytes in addition to other needs.

IMPORTANT NOTE

> The macro definitions and extended example for MS-DOS system calls 00H through 2EH can be found at the end of this chapter.

Table 1.5 lists the function requests in numeric order; Table 1.6 lists the function requests in alphabetic order (of the description).

Table 1.5 MS-DOS Function Requests, Numeric Order

Function Number	Function Name
00H	Terminate Program
01H	Read Keyboard and Echo
02H	Display Character
03H	Auxiliary Input
04H	Auxiliary Output
05H	Print Character
06H	Direct Console I/0
07H	Direct Console Input
08H	Read Keyboard
09H	Display String
0AH	Buffered Keyboard Input
0BH	Check Keyboard Status
0CH	Flush Buffer, Read Keyboard
0DH	Disk Reset
0EH	Select Disk
0FH	Open File
10H	Close File
11H	Search for First Entry
12H	Search for Next Entry
13H	Delete File
14H	Sequential Read
15H	Sequential Write

16H	Create File
17H	Rename File
19H	Current Disk
1AH	Set Disk Transfer Address
21H	Random Read
22H	Random Write
23H	File Size
24H	Set Relative Record
25H	Set Vector
27H	Random Block Read
28H	Random Block Write
29H	Parse File Name
2AH	Get Date
2BH	Set Date
2CH	Get Time
2DH	Set Time
2EH	Set/Reset Verify Flag
2FH	Get Disk Transfer Address
30H	Get DOS Version Number
31H	Keep Process
33H	CONTROL-C Check
35H	Get Interrupt Vector
36H	Get Disk Free Space
38H	Return Country-Dependent Info.
39H	Create Sub-Directory
3AH	Remove a Directory Entry
3BH	Change the Current Directory
3CH	Create a File
3DH	Open a File
3EH	Close a File Handle
3FH	Read From File/Device
40H	Write to a File/Device
41H	Delete a Directory Entry
42H	Move a File Pointer
43H	Change Attributes
44H	I/O Control for Devices
45H	Duplicate a File Handle
46H	Force a Duplicate of a Handle
47H	Return Text of Current Directory
48H	Allocate Memory
49H	Free Allocated Memory
4AH	Modify Allocated Memory Blocks
4BH	Load and Execute a Program
4CH	Terminate a Process

4DH	Retrieve the Return Code of a Child
4EH	Find Match File
4FH	Step Through a Directory Matching Files
54H	Return Current Setting of Verify
56H	Move a Directory Entry
57H	Get/Set Date/Time of File

Table 1.6 MS-DOS Function Requests, Alphabetic Order

Function Name	Number
Allocate Memory	48H
Auxiliary Input	03H
Auxiliary Output	04H
Buffered Keyboard Input	0AH
Change Attributes	43H
Change the Current Directory	3BH
Check Keyboard Status	0BH
Close a File Handle	3EH
Close File	10H
CONTROL-C Check	33H
Create a File	3CH
Create File	16H
Create Sub-Directory	39H
Current Disk	19H
Delete a Directory Entry	41H
Delete File	13H
Direct Console Input	07H
Direct Console I/O	06H
Disk Reset	0DH
Display Character	02H
Display String	09H
Duplicate a File Handle	45H
File Size	23H
Find Match File	4EH
Flush Buffer, Read Keyboard	0CH
Force a Duplicate of a Handle	46H
Free Allocated Memory	49H
Get Date	2AH
Get Disk Free Space	36H
Get Disk Transfer Address	2FH
Get DOS Version Number	30H
Get Interrupt Vector	35H

Get Time	2CH
Get/Set Date/Time of File	57H
I/D Control for Devices	44H
Keep Process	31H
Load and Execute a Program	4BH
Modify Allocated Memory Blocks	4AH
Move a Directory Entry	56H
Move a File Pointer	42H
Open a File	3DH
Open File	0FH
Parse File Name	29H
Print Character	05H
Random Block Read	27H
Random Block Write	28H
Random Read	21H
Random Write	22H
Read From File/Device	3FH
Read Keyboard	08H
Read Keyboard and Echo	01H
Remove a Directory Entry	3AH
Rename File	17H
Retrieve the Return Code of a Child	4DH
Return Current Setting of Verify	54H
Return Country-Dependent Info.	38H
Return Text of Current Directory	47H
Search for First Entry	11H
Search for Next Entry	12H
Select Disk	0EH
Sequential Read	14H
Sequential Write	15H
Set Date	2BH
Set Disk Transfer Address	1AH
Set Relative Record	24H
Set Time	2DH
Set Vector	25H
Set/Reset Verify Flag	2EH
Step Through a Directory Matching	4FH
Terminate a Process	4CH
Terminate Program	00H
Write to a File/Device	40H

Terminate Program (Function 00H)

Call
AH = 00H
CS
  Segment address of
  Program Segment Prefix

Return
None

Function 00H is called by Interrupt 20H; it performs the same processing.

The CS register must contain the segment address of the Program Segment Prefix before you call this interrupt.

The following exit addresses are restored from the specified offsets in the Program Segment Prefix:

Program terminate	0AH
CONTROL-C	0EH
Critical error	12H

All file buffers are flushed to disk.

Warning: Close all files that have changed in length before calling this function. If a changed file is not closed, its length is not recorded correctly in the directory. See Function 10H for a description of the Close File system call.

```
Macro Definition:   terminate-program   macro
                                        xor     ah,ah
                                        int     21H
                                        endm
```

Example

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
        mov     ah,0
        int     21H
;There are no returns from this interrupt
```

Read Keyboard and Echo (Function 01H)

Call
AH = 01H

Return
AL
Character typed

Function 01H waits for a character to be typed at the keyboard, then echoes the character to the display and returns it in AL. If the character is CONTROL-C, Interrupt 23H is executed.

```
Macro Definition:   read-kbd-and-echo   macro
                                        mov     ah, 01H
                                        int     21H
                                        endm
```

Example

The following program both displays and prints characters as they are typed. If RETURN is pressed, the program sends Line Feed-Carriage Return to both the display and the printer:

```
func-01H: read-kbd-and-echo            ;THIS FUNCTION
          print-char    al             ;see Function 05H
          cmp           al,0DH         ;is it a CR?
          jne           func-01H       ;no, print it
          print-char    10             ;see Function 05H
          display-char  10             ;see Function 02H
          jmp           func-01H       ;get another character
```

Display Character (Function 02H)

Call
AH = 02H
DL
  Character to be displayed

Return
None

Function 02H displays the character in DL. If CONTROL-C is typed, Interrupt 23H is issued.

```
Macro Definition:  display-char   macro   character
                                  mov     dl,character
                                  mov     ah, 02H
                                  int     21H
                                  endm
```

Example

The following program converts lowercase characters to uppercase before displaying them:

```
func-02H:   read-kbd                    ;see Function 08H
            cmp      al,“a”
            jl       uppercase          ;don’t convert
            cmp      al,“z”
            jg       uppercase          ;don’t convert
            sub      al,20H             ;convert to ASCII code
                                        ;for uppercase
uppercase:  display-char al             ;THIS FUNCTION
            jmp      func-02H:          ;get another character
```

Auxiliary Input (Function 03H)

Call
AH = 03H

Return
AL
  Character from auxiliary device

Function 03H waits for a character from the auxiliary input device, then returns the character in AL. This system call does not return a status or error code.

If a CONTROL-C has been typed at console input, Interrupt 23H is issued.

```
Macro Definition:   aux-input    macro
                                 mov     ah,03H
                                 int     21H
                                 endm
```

Example

The following program prints characters as they are received from the auxiliary device. It stops printing when an end-of-file character (ASCII 1AH, or CONTROL-Z) is received:

```
func-03H:  aux-input               ;THIS FUNCTION
           cmp       al,1AH        ;end of file?
           je        continue      ;yes, all done
           print-char  al          ;see Function 05H
           jmp       func-03H      ;get another character
continue:  .
```

Auxiliary Output (Function 04H)

Call
AH = 04H
DL
  Character for auxiliary device

Return
None

Function 04H sends the character in DL to the auxiliary output device. This system call does not return a status or error code.
If a CONTROL-C has been typed at console input, Interrupt 23H is issued.

```
Macro Definition:  aux-output   macro   character
                                mov     dl,character
                                mov     ah,04H
                                int     21H
                                endm
```

Example

The following program gets a series of strings of up to 80 bytes from the keyboard, sending each to the auxiliary device. It stops when a null string (CR only) is typed:

```
string      db     81 dup(?) ;see Function 0AH
            .
func-04H:   get-string  80,string          ;see Function 0AH
            cmp    string[1],0             ;null string?
            je     continue                ;yes, all done
            mov    cx, word ptr string[1]  ;get string length
            mov    bx,0                    ;set index to 0
send-it:    aux-output string[bx+2]        ;THIS FUNCTION
            inc    bx                      ;bump index
            loop   send-it                 ;send another character
            jmp    func-04H                ;get another string
continue:   .
            .
```

Print Character (Function 05H)

Call
AH = 05H
DL
  Character for printer

Return
None

Function 05H prints the character in DL on the standard printer device. If CONTROL-C has been typed at console input, Interrupt 23H is issued.

```
Macro Definition:   print-char    macro   character
                                  mov     dl,character
                                  mov     ah,05H
                                  int     21H
                                  endm
```

Example

The following program prints a walking test pattern on the printer. It stops if CONTROL-C is pressed.

```
line-num     db        0
             .
func-05H:    mov       cx,60          ;print 60 lines
start-line:  mov       bl,33          ;first printable ASCII
                                      ;character (!)
             add       bl,line-num    ;to offset ne character
             push      cx             ;save number-of-lines counter
             mov       cx,80          ;loop counter for line
print-it:    print-char bl            ;THIS FUNCTION
             inc       bl             ;move to next ASCII character
             cmp       bl,126         ;last printable ASCII
                                      ;character ( ˜)
             jl        no-reset       ;not there yet
             mov       bl,33          ;start over with (!)
```

```
no-reset:   loop        print-it        ;print another character
            print-char 13               ;carriage return
            print-char 10               ;line feed
            inc         line-num        ;to offset 1st char. of line
            pop         cx              ;restore #-of-lines counter
            loop        start-line;     ;print another line
```

Direct Console I/0 (Function 06H)

Call
AH = 06H
DL
  See text

Return
AL
  If DL = FFH (255) before call, then Zero flag not set means AL has character from keyboard.
  Zero flag set means there was not a character to get, and AL = 0

The processing depends on the value in DL when the function is called:

DL is FFH (255) - If a character has been typed at the keyboard, it is returned in AL and the Zero flag is 0; if a character has not been typed, the Zero flag is 1.
DL is not FFH - The character in DL is displayed.

This function does **not** check for CONTROL-C.

```
Macro Definition:  dir-console-io macro   switch
                                  mov     dl,switch
                                  mov     ah,06H
                                  int     21H
                                  endm
```

Example

The following program sets the system clock to 0 and continuously displays the time. When any character is typed, the display stops changing; when any character is typed again, the clock is reset to 0 and the display starts again:

```
time        db  "00:00:00.00",13,10,"$"  ;see Function 09H
;                                        ;for explanation of $
ten         db  10
            .
            .
func-06H:   set-time  0,0,0,0           ;see Function 2DH
read-clock: get-time                    ;see Function 2CH
            convert  ch,ten,time        ;see end of chapter
            convert  cl,ten,time[3]     ;see end of chapter
            convert  dh,ten,time[6]     ;see end of chapter
            convert  dl,ten,time[9]     ;see end of chapter
            display  time               ;see Function 09H
            dir-console-io  FFH         ;THIS FUNCTION
            jne      stop               ;yes, stop timer
            jmp      read-clock         ;no, keep timer
                                        ;running
stop:       read-kbd                    ;see Function 08H
            jmp      func-06H           ;start over
```

Direct Console Input (Function 07H)

Call
AH = 07H

Return
AL
  Character from keyboard

Function 07H waits for a character to be typed, then returns it in AL. This function does not echo the character or check for CONTROL-C. (For a keyboard input function that echoes or checks for CONTROL-C, see Functions 01H or 08H.)

```
Macro Definition:  dir-console-input  macro
                                      mov     ah,07H
                                      int     21H
                                      endm
```

Example

The following program prompts for a password (8 characters maximum) and places the characters into a string without echoing them:

```
password    db       8 dup(?)
prompt      db       "Password: $"   ;see Function 09H for
                                     ;explanation of $
            .
            .
func-07H:   display prompt           ;see Function 09H
            mov      cx,8            ;maximum length of password
            xor      bx,bx           ;so BL can be used as index
get-pass:   dir-console-input        ;THIS FUNCTION
            cmp      al,0DH          ;was it a CR?
            je       continue        ;yes, all done
            mov      password[bx],al ;no, put character in string
            inc      bx              ;bump index
            loop     get-pass        ;get another character
continue:   .                        ;BX has length of password+1
            .
```

Read Keyboard (Function 08H)

Call
AH = 08H

Return
AL
  Character from keyboard

Function 08H waits for a character to be typed, then returns it in AL. If CONTROL-C is pressed, Interrupt 23H is executed. This function does not echo the character. (For a keyboard input function that echoes the character or does not check for CONTROL-C, see Functions 01H or 07H.)

```
Macro Definition:  read-kbd    macro
                               mov     ah,08H
                               int     21H
                               endm
```

Example

The following program prompts for a password (8 characters maximum) and places the characters into a string without echoing them:

```
password   db      8 dup(?)
prompt     db      "Password: $"  ;see Function 09H
                                  ;for explanation of $
           .

           .
func-08H:  display prompt         ;see Function 09H
           mov     cx,8           ;maximum length of password
           xor     bx,bx          ;BL can be an index
get-pass:  read-kbd               ;THIS FUNCTION
           cmp     al,0DH         ;was it a CR?
           je      continue       ;yes, all done
           mov     password[bx],al ;no, put char. in string
           inc     bx             ;bump index
           loop    get-pass       ;get another character
continue:  .                      ;BX has length of password+1
           .
```

Display String (Function 09H)

Call
AH = 09H
DS:DX
  String to be displayed

Return
None

DX must contain the offset (from the segment address in DS) of a string that ends with “$”. The string is displayed (the $ is not displayed).

```
Macro Definition:   display   macro   string
                              mov     dx,offset string
                              mov     ah,09H
                              int     21H
                              endm
```

Example

The following program displays the hexadecimal code of the key that is typed:

```
table       db        “0123456789ABCDEF”
sixteen     db        16
result      db        “ - 00H”,13,10,“$”     ;see text for
                                             ;explanation of $
            .
            .
func-09H:   read-kbd-and-echo                ;see Function 01H
            convert   al, sixteen, result[3] ;see end of chapter
            display   result                 ;THIS FUNCTION
            jmp       func-09H               ;do it again
```

Buffered Keyboard Input (Function 0AH)

Call
AH = 0AH
DS:DX
  Input buffer

Return
None

DX must contain the offset (from the segment address in DS) of an input buffer of the following form:

Byte	Contents
1	Maximum number of characters in buffer, including the CR (you must set this value).
2	Actual number of characters typed, not counting the CR (the function sets this value).
3-h	Buffer; must be at least as long as the number in byte 1.

This function waits for characters to be typed. Characters are read from the keyboard and placed in the buffer beginning at the third byte until RETURN is typed. If the buffer fills to one less than the maximum, additional characters typed are ignored and ASCII 7 (BEL) is sent to the display until RETURN is pressed. The string can be edited as it is being entered. If CONTROL-C is typed, Interrupt 23H is issued.

The second byte of the buffer is set to the number of characters entered (not counting the CR).

```
Macro Definition:  get-string   macro   limit,string
                                mov     dx,offset string
                                mov     string,limit
                                mov     ah,0AH
                                int     21H
                                endm
```

Example

The following program gets a 16-byte (maximum) string from the keyboard and fills a 24-line by 80-character screen with it:

```
buffer           label   byte
max-length       db      ?                    ;maximum length
chars-entered    db      ?                    ;number of chars.
string           db      17 dup (?)           ;16 chars + CR
strings-per-line dw      0                    ;how many strings
                                              ;fit on line
crlf             db      13,10,”$“
                 .
                 .
func-0AH:        get-string 17,buffer         ;THIS FUNCTION
                 xor     bx,bx                ;so byte can be
                                              ;used as index
                 mov     bl,chars-entered     ;get string length
                 mov     buffer[bx+2],”$“     ;see Function 09H
                 mov     al,50H               ;columns per line
                 cbw
                 div     chars-entered        ;times string fits
                                              ;on line
                 xor     ah,ah                ;clear remainder
                 mov     strings-per-line,ax  ;save col. counter
                 mov     cx,24                ;row counter
display-screen:  push    cx                   ;save it
                 mov     cx, strings-per-line ;get col. counter
display-line:    display string               ;see Function 09H
                 loop    display-line
                 display crlf                 ;see Function 09H
                 pop     cx                   ;get line counter
                 loop    display-screen       ;display 1 more line
```

Check Keyboard Status (Function 0BH)

```
Call
AH = 0BH

Return
AL
  255 (FFH) = characters in type-ahead
  buffer
  0 = no characters in type-ahead
      buffer
```

Checks whether there are characters in the type-ahead buffer. If so, AL returns FFH (255); if not, AL returns 0. If CONTROL-C is in the buffer, Interrupt 23H is executed.

```
Macro Definition:   check-kbd-status  macro
                                      mov     ah,0BH
                                      int     21H
                                      endm
```

Example
The following program continuously displays the time until any key is pressed.

```
time        db      "00:00:00.00",13,10,"$"
ten         db      10
            .
            .
func-0BH:   get-time                  ;see Function 2CH
            convert ch,ten,time       ;see end of chapter
            convert cl,ten,time[3]    ;see end of chapter
            convert dh,ten,time[6]    ;see end of chapter
            convert dl,ten,time[9]    ;see end of chapter
            display time              ;see Function 09H
            check-kbd-status          ;THIS FUNCTION
            cmp     al, FFH           ;has a key been typed?
            je      all-done          ;yes, go home
            jmp     func-0BH          ;no, keep displaying
                                      ;time
```

Flush Buffer, Read Keyboard (Function 0CH)

Call
AH = 0CH
AL
1, 6, 7, 8, or 0AH = The corresponding function is called.
Any other value = no further processing.

Return
AL
0 = Type-ahead buffer was flushed; no other
processing performed.

The keyboard type-ahead buffer is emptied. Further processing depends on the value in AL when the function is called:

> 1, 6, 7, 8, or 0AH - The corresponding MS-DOS function is executed.
>
> Any other value - No further processing; AL returns 0.

Macro Definition:

```
flush-and-read-kbd macro    switch
                   mov      al,switch
                   mov      ah,0CH
                   int      21H
                   endm
```

Example

The following program both displays and prints characters as they are typed. If RETURN is pressed, the program sends Carriage Return-Line Feed to both the display and the printer.

```
func-0CH: flush-and-read-kbd 1       ;THIS FUNCTION
          print-char   al            ;see Function 05H
          cmp          al,0DH        ;is it a CR?
          jne          func-0CH      ;no, print it
          print-char   10            ;see Function 05H
          display-char 10            ;see Function 02H
          jmp          func-0CH      ;get another character
```

Disk Reset (Function 0DH)

Call
AH = 0DH

Return
None

Function 0DH is used to ensure that the internal buffer cache matches the disks in the drives. This function writes out dirty buffers (buffers that have been modified), and marks all buffers in the internal cache as free.

Function 0DH flushes all file buffers. It does not update directory entries; you must close files that have changed to update their directory entries (see Function 10H, Close File). This function need not be called before a disk change if all files that changed were closed. It is generally used to force a known state of the system; CONTROL-C interrupt handlers should call this function.

```
Macro Definition:  disk-reset    macro   disk
                                 mov     ah,0DH
                                 int     21H
                                 endm
```

Example

```
        mov     ah,0DH
        int     21H
;There are no errors returned by this call.
```

Select Disk (Function 0EH)

Call
AH = 0EH
DL
  Drive number
  (0 = A:, 1 = B:, etc.)

Return
AL
  Number of logical drives

The drive specified in DL (0 = A:, 1 = B:, etc.) is selected as the default disk. The number of drives is returned in AL.

```
Macro Definition:  select-disk   macro   disk
                                 mov     dl,disk[-64]
                                 mov     ah, 0EH
                                 int     21H
                                 endm
```

Example
The following program selects the drive not currently selected in a 2-drive system:

```
func-0EH:  current-disk          ;see Function 19H
           cmp      al,00H       ;drive A: selected?
           je       select-b     ;yes, select B
           select-disk "A"       ;THIS FUNCTION
           jmp      continue
select-b:  select-disk "B"       ;THIS FUNCTION
Continue:  .
           .
```

Open File (Function 0FH)

Call
AH = 0FH
DS:DX
Unopened FCB

Return
AL
0 = Directory entry found
255 (FFH) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened File Control Block (FCB). The disk directory is searched for the named file.
If a directory entry for the file is found, AL returns 0 and the FCB is filled as follows:

> If the drive code was 0 (default disk), it is changed to the actual disk used (1 = A:, 2 = B:, etc.). This lets you change the default disk without interfering with subsequent operations on this file.
> The Current Block field (offset 0CH) is set to zero.
> The Record Size (offset 0EH) is set to the system default of 128.
> The File Size (offset 10H), Date of Last Write (offset 14H), and Time of Last Write (offset 16H) are set from the directory entry.

Before performing a sequential disk operation on the file, you must set the Current Record field (offset 20H). Before performing a random disk operation on the file, you must set the Relative Record field (offset 21H). If the default record size (128 bytes) is not correct, set it to the correct length.

If a directory entry for the file is not found, AL returns FFH (255).

```
Macro Definition:   open          macro    fcb
                                  mov      dx,offset fcb
                                  mov      ah,0FH
                                  int      21H
                                  endm
```

Example
The following program prints the file named TEXTFILE.ASC that is on the disk in drive B:. If a partial record is in the buffer at end-of-file, the routine that prints the partial record prints characters until it encounters an end-of-file mark (ASCII 26, or CONTROL-Z):

```
fcb          db        2,”TEXTFILEASC”
             db        25 dup (?)
buffer       db        128 dup (?)
             .
func-0FH:    set-dta  buffer           ;see Function 1AH
             open      fcb             ;THIS FUNCTION
read-line:   read-seq  fcb             ;see Function 14H
             cmp       al,02H          ;end of file?
             je        all-done        ;yes, go home
             cmp       al,00H          ;more to come?
             jg        check-more      ;no, check for partial
                                       ;record
             mov       cx,128          ;yes, print the buffer
             xor       si,si           ;set index to 0
print-it:    print-char  buffer[si]    ;see Function 05H
             inc       si              ;bump index
             loop      print-it        ;print next character
             jmp       read-line       ;read another record
check-more:  cmp       al,03H          ;part. record to print?
             jne       all-done        ;no
             mov       cx,128          ;yes, print it
             xor       si,si           ;set index to 0
find-eof:    cmp       buffer[si],26   ;end-of-file mark?
             je        all-done        ;yes
             print-char buffer[si]     ;see Function 05H
             inc       si              ;bump index to next
                                       ;character
             loop      find-eof
all-done:    close     fcb             ;see Function 10H
```

Close File (Function 10H)

Call
AH = 10 H
DS:DX
  Opened FCB

Return
AL
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (to the segment address in DS) of an opened FCB. The disk directory is searched for the file named in the FCB. This function must be called after a file is changed to update the directory entry.
If a directory entry for the file is found, the location of the file is compared with the corresponding entries in the FCB. The directory entry is updated, if necessary, to match the FCB, and AL returns 0.
If a directory entry for the file is not found, AL returns FFH (255).

```
Macro Definition:  close      macro   fcb
                              mov     dx,offset fcb
                              mov     ah,10H
                              int     21H
                              endm
```

Example
The following program checks the first byte of the file named MOD1.-BAS in drive B: to see if it is FFH, and prints a message if it is:

```
message    db       ”Not saved in ASCII format“,13,10,”$“
fcb        db       2,”MOD1   BAS“
           db       25 dup (?)
buffer     db       128 dup (?)
           .
           .
func-10H:  set-dta buffer           ;see Function 1AH
           open     fcb             ;see Function 0FH
           read-seq fcb             ;see Function 14H
```

```
            cmp      buffer,FFH     ;is first byte FFH?
            jne      all-done       ;no
            display  message        ;see Function 09H
all-done:   close    fcb            ;THIS FUNCTION
```

Search for First Entry (Function 11H)

Call
AH = 11H
DS:DX
  Unopened FCB

Return
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened FCB. The disk directory is searched for the first matching name. The name can have the ? wild card character to match any character. To search for hidden or system files, DX must point to the first byte of the extended FCB prefix.

If a directory entry for the filename in the FCB is found, AL returns 0 and an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.

If a directory entry for the filename in the FCB is not found, AL returns FFH (255).

Notes:

If an extended FCB is used, the following search pattern is used:

1. If the FCB attribute is zero, only normal file entries are found. Entries for volume label, sub-directories, hidden, and system files will not be returned.
2. If the attribute field is set for hidden or system files, or directory entries, it is to be considered as an inclusive search. All normal file entries plus all entries matching the specified attributes are returned. To look at all directory entries except the volume label, the attribute byte may be set to hidden + system + directory (all 3 bits on).

3. If the attribute field is set for the volume label, it is considered an exclusive search, and only the volume label entry is returned.

```
Macro Definition:  search-first   macro   fcb
                                  mov     dx,offset fcb
                                  mov     ah,11H
                                  int     21H
                                  endm
```

Example
The following program verifies the existence of a file named REPORT.ASM on the disk in drive B::

```
yes          db        ”FILE EXISTS.$“
no           db        ”FILE DOES NOT EXIST.$“
fcb          db        2,”REPORT ASM“
             db        25 dup (?)
buffer       db        128 dup (?)
             .
func-11H:    set-dta   buffer            ;see Function 1AH
             search-first fcb            ;THIS FUNCTION
             cmp       al,FFH            ;directory entry found?
             je        not-there         ;no
             display   yes               ;see Function 09H
             jmp       continue
not-there:   display   no                ;see Function 09H
continue:    display   crlf              ;see Function 09H
             .
             .
```

Search for Next Entry (Function 12H)

Call
AH = 12H
DS:DX
  Unopened FCB

Return
AL
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an FCB previously specified in a call to Function 11H. Function 12H is used after Function 11H (Search for First Entry) to find additional directory entries that match a filename that contains wild card characters. The disk directory is searched for the next matching name. The name can have the ? wild card character to match any character. To search for hidden or system files, DX must point to the first byte of the extended FCB prefix.

If a directory entry for the filename in the FCB is found, AL returns 0 and an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.

If a directory entry for the filename in the FCB is not found, AL returns FFH (255).

```
Macro Definition:  search-next  macro   fcb
                                mov     dx,offset fcb
                                mov     ah,12H
                                int     21H
                                endm
```

Example

The following program displays the number of files on the disk in drive B:

```
message   db    ”No files“,10,13,”$“
files     db    0
ten       db    10
fcb       db    2,”???????????“
          db    25  dup (?)
buffer    db    128 dup (?)
```

```
             .
             .
func-12H:    set-dta buffer          ;see Function 1AH
             search-first fcb        ;see Function 11H
             cmp      al,FFH         ;directory entry found?
             je       all-done       ;no, no files on disk
             inc      files          ;yes, increment file
                                     ;counter
search-dir:  search-next fcb         ;THIS FUNCTION
             cmp      al,FFH         ;directory entry found?
             je       done           ;no
             inc      files          ;yes, increment file
                                     ;counter
             jmp      search-dir     ;check again
done:        convert files,ten,message ;see end of chapter
all-done:    display  message        ;see Function 09H
```

Delete File (Function 13H)

Call
AH = 13H
DS:DX
  Unopened FCB

Return
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened FCB. The directory is searched for a matching filename. The filename in the FCB can contain the ? wild card character to match any character.
If a matching directory entry is found, it is deleted from the directory. If the ? wild card character is used in the filename, all matching directory entries are deleted. AL returns 0.
If no matching directory entry is found, AL returns FFH (255).

```
Macro Definition:  delete      macro   fcb
                               mov     dx,offset fcb
                               mov     ah,13H
                               int     21H
                               endm
```

Example
The following program deletes each file on the disk in drive B: that was last written before December 31, 1982:

```
year      dw     1982
month     db     12
day       db     31
files     db     0
ten       db     10
message   db     ”NO FILES DELETED.”,13,10,”$“
                                   ;see Function 09H for
                                   ;explanation of $
fcb       db     2,”???????????“
          db     25 dup (?)
```

```
buffer      db      128 dup (?)
            .
            .
func-13H:   set-dta  buffer            ;see Function 1AH
            search-first fcb           ;see Function 11H
            cmp     al,FFH             ;directory entry found?
            je      all-done           ;no, no files on disk
compare:    convert-date buffer        ;see end of chapter
            cmp     cx,year            ;next several lines
            jg      next               ;check date in directory
            cmp     dl,month           ;entry against date
            jg      next               ;above & check next file
            cmp     dh,day             ;if date in directory
            jge     next               ;entry isn't earlier.
            delete  buffer             ;THIS FUNCTION
            inc     files              ;bump deleted-files
                                       ;counter
next:       search-next fcb            ;see Function 12H
            cmp     al,00H             ;directory entry found?
            je      compare            ;yes, check date
            cmp     files,0            ;any files deleted?
            je      all-done           ;no, display NO FILES
                                       ;message.
            convert files,ten,message  ;see end of chapter
all-done:   display  message           ;see Function 09H
```

Sequential Read (Function 14H)

Call
AH = 14H
DS:DX
  Opened FCB

Return
Al
  0 = Read completed successfully
  1 = EOF
  2 = DTA too small
  3 = EOF, partial record

DX must contain the offset (from the segment address in DS) of an opened FCB. The record pointed to by the current block (offset 0CH) and Current Record (offset 20H) fields is loaded at the Disk Transfer Address, then the Current Block and Current Record fields are incremented.
The record size is set to the value at offset 0EH in the FCB.
AL returns a code that describes the processing:

Code	Meaning
0	Read completed successfully.
1	End-of-file, no data in the record.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.
3	End-of-file; a partial record was read and padded to the record length with zeros.

```
Macro Definition:   read-seq      macro   fcb
                                  mov     dx,offset fcb
                                  mov     ah,14H
                                  int     21H
                                  endm
```

Example
The following program displays the file named TEXTFILE.ASC that is on the disk in drive B:; its function is similar to the MS-DOS TYPE command. If a partial record is in the buffer at end of file, the routine that displays the partial record displays characters until it encounters an end-of-file mark (ASCII 26, or CONTROL-Z):

```
fcb          db         2,”TEXTFILEASC“
             db         25 dup (?)
buffer       db         128 dup (?),”$“
             .
             .
func-14H:    set-dta  buffer          ;see Function 1AH
             open       fcb           ;see Function 0FH
read-line:   read-seq fc              ;THIS FUNCTION
             cmp        al,02H        ;end-of-file?
             je         all-done      ;yes
             cmp        al,02H        ;end-of-file with partial
                                      ;record?
             jg         check-more    ;yes
             display buffer           ;see Function 09H
             jmp        read-line     ;get another record
check-more:  cmp        al,03H        ;partial record in buffer?
             jne        all-done      ;no, go home
             xor        si,si         ;set index to 0
find-eof:    cmp        buffer[si],26 ;is character EOF?
             je         all-done      ;yes, no more to display
             display-char buffer[si]  ;see Function 02H
             inc        si            ;bump index to next
                                      ;character
             jmp        find-eof      ;check next character
all-done     close      fcb           ;see Function 10H
```

Sequential Write (Function 15H)

Call
AH = 15H
DS:DX
  Opened FCB

Return
AL
  00H = Write completed successfully
  01H = Disk full
  02H = DTA too small

DX must contain the offset (from the segment address in DS) of an opened FCB. The record pointed to by Current Block (offset 0CH) and Current Record (offset 20H) fields is written from the Disk Transfer Address, then the current block and current record fields are incremented.

The record size is set to the Value at offset 0EH in the FCB. If the Record Size is less than a sector, the data at the Disk Transfer Address is written to a buffer; the buffer is written to disk when it contains a full sector of data, or the file is closed, or a Reset Disk system call (Function 0DH) is issued.

AL returns a code that describes the processing:

Code	Meaning
0	Transfer completed successfully.
1	Disk full; write canceled.
2	Not enough room at the Disk Transfer Address to write one record; write canceled

Macro Definition:

```
write-seq   macro   fcb
            mov     dx,offset fcb
            mov     ah,15H
            int     21H
            endm
```

Example

The following program creates a file named DIR.TMP on the disk in drive B: that contains the disk number (0 = A:, 1 = B:, etc.) and filename from each directory entry on the disk:

```
record-size  equu         14              ;offset of Record Size
                                          ;field in FCB
             .
             .
fcb1         db           2,"DIR  TMP"
             db           25 dup (?)
fcb2         db           2,"???????????"
             db           25 dup (?)
buffer       db           128 dup (?)
             .
             .
func-15H:    set-dta      buffer          ;see Function 1AH
             search-first fcb2            ;see Function 11H
             cmp          al,FFH          ;directory entry found?
             je           all-done        ;no, no files on disk
             create       fcb1            ;see Function 16H
             mov          fcb1[record-size],12
                                          ;set record size to 12
write-it:    write-seq    fcb1            ;THIS FUNCTION
             search-next  fcb2            ;see Function 12H
             cmp          al,FFH          ;directory entry found?
             je           all-done        ;no, go home
             jmp          write-it        ;yes, write the record
all-done:    close        fcb1            ;see Function 10H
```

Create File (Function 16H)

Call
AH = 16H
DS:DX
  Unopened FCB

Return
AL
  00H = Empty directory found
  FFH (255) = No empty directory
    available

DX must contain the offset (from the segment address in DS) of an unopened FCB. The directory is searched for an empty entry or an existing entry for the specified filename.

If an empty directory entry is found, it is initialized to a zero-length file, the Open File system call (Function 0FH) is called, and AL returns 0. You can create a hidden file by using an extended FCB with the attribute byte (offset FCB-1) set to 2.

If an entry is found for the specified filename, all data in the file is released, making a zero-length file, and the Open File system call (Function 0FH) is issued for the filename (in other words, if you try to create a file that already exists, the existing file is erased, and a new, empty file is created).

If an empty directory entry is not found and there is no entry for the specified filename, AL returns FFH (255).

Macro Definition:
```
create  macro   fcb
        mov     dx,offset fcb
        mov     ah,16H
        int     21H
        endm
```

Example

The following program creates a file named DIR.TMP on the disk in drive B: that contains the disk number (0 = A:, 1 = B:, etc.) and filename from each directory entry on the disk:

```
record-size  equ         14                   ;offset of Record Size
                                                  ;field of FCB
             .
             .
fcb1         db          2,"DIR TMP"
             db          25 dup (?)
fcb2         db          2,"???????????"
             db          25 dup (?)
buffer       db          128 dup (?)
             .
             .
func-16H:    set-dta     buffer               ;see Function 1AH
             search-first fcb2                ;see Function 11H
             cmp         al,FFH               ;directory entry found?
             je          all-done             ;no, no files on disk
             create      fcb1                 ;THIS FUNCTION
             mov         fcb1[record-size],12
                                              ;set record size to 12
write-it:    write-seq fcb1                   ;see Function 15H
             search-next fcb2                 ;see Function 12H
             cmp         al,FFH               ;directory entry found?
             je          all-done             ;no, go home
             jmp         write-it             ;yes, write the record
all-done:    close       fcb1                 ;see Function 10H
```

Rename File (Function 17H)

Call
AH = 17H
DS:DX
  Modified FCB

Return
AL
  00H = Directory entry found
  EFH (255) = No directory entry
  found or destination already exists

DX must contain the offset (from the segment address in DS) of an FCB with the drive number and filename filled in, followed by a second filename at offset 11H. The disk directory is searched for an entry that matches the first filename, which can contain the ? wild card character.

If a matching directory entry is found, the filename in the directory entry is changed to match the second filename in the modified FCB (the two filenames cannot be the same name). If the ? wild card character is used in the second filename, the corresponding characters in the filename of the directory entry are not changed. AL returns 0.

If a matching directory entry is not found or an entry is found for the second filename, AL returns FFH (255).

```
Macro Definition:  rename    macro   fcb,newname
                             mov     dx,offset fcb
                             mov     ah,17H
                             int     21H
                             endm
```

Example

The following program prompts for the name of a file and a new name, then renames the file:

```
fcb                 db    37 dup (?)
prompt1             db    "Filename: $"
prompt2             db    "New name: $"
reply               db    17 dup(?)
crlf                db    13,10,"$"
                    .
                    .
```

```
func-17H:  display    promptl          ;see Function 09H
           get-string 15,reply         ;see Function 0AH
           display    crlf             ;see Function 09H
           parse      reply[2],fcb     ;see Function 29H
           display    prompt2          ;see Function 09H
           get-string  15,reply        ;see Function 0AH
           display    crlf             ;see Function 09 H
           parse      reply[2],fcb[16]
                                       ;see Function 29H
           rename     fcb              ;THIS FUNCTION
```

Current Disk (Function 19H)

Call
AH = 19H

Return
AL
Currently selected drive
(0 = A, 1 = B, etc.)

AL returns the currently selected drive (0 = A:, 1 = B:, etc.).

```
Macro Definition:  current-disk  macro
                                 mov     ah,19H
                                 int     21H
                                 endm
```

Example
The following program displays the currently selected (default) drive in a 2-drive system:

```
message    db "Current disk is $"  ;see Function 09H
                                   ;for explanation of $
crlf       db      13,10,"$"
           .
           .
func-19H:  display  message        ;see Function 09H
           current-disk            ;THIS FUNCTION
           cmp     al,00H          ;is it disk A?
           jne     disk-b          ;no, it's disk B:
           display-char "A"        ;see Function 02H
           jmp     all-done
disk-b:    display-char "B"        ;see Function 02H
all-done:  display  crlf           ;see Function 09H
```

Set Disk Transfer Address (Function 1AH)

Call
AH = 1AH
DS:DX
  Disk Transfer Address

Return
None

DX must contain the offset (from the segment address in DS) of the Disk Transfer Address. Disk transfers cannot wrap around from the end of the segment to the beginning, nor can they overflow into another segment.

NOTE

> If you do not set the Disk Transfer Address, MS-DOS defaults to offset 80H in the Program Segment Prefix.

```
Macro Definition:   set-dta       macro   buffer
                                  mov     dx,offset buffer
                                  mov     ah,1AH
                                  int     21H
                                  endm
```

Example
The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. The file contains 26 records; each record is 28 bytes long:

```
record-size     equu    14              ;offset of Record Size
                                        ;field of FCB
relative-record equ     33              ;offset of Relative Record
                                        ;field of FCB
                .
                .
```

```
fcb        db        2,"ALPHABETDAT"
           db        25 dup (?)
buffer     db        34 dup (?),"$"
prompt     db        "Enter letter: $"
crlf       db        13,10,"$"
           .
           .
func-1AH:  set-dta   buffer            ;THIS FUNCTION
           open      fcb               ;see Function 0FH
           mov       fcb[record-size],28 ;set record size
get-char:  display   prompt            ;see Function 09H
           read-kbd-and-echo           ;see Function 01H
           cmp       al,0DH            ;just a CR?
           je        all-done          ;yes, go home
           sub       al,41H            ;convert ASCII
                                       ;code to record #
           mov       fcb[relative-record],al
                                       ;set relative record
           display   crlf              ;see Function 09H
           read-ran  fcb               ;see Function 21H
           display   buffer            ;see Function 09H
           display   crlf              ;see Function 09H
           jmp       get-char          ;get another character
all-done:  close     fcb               ;see Function 10H
```

Random Read (Function 21H)

Call
AH = 21H
DS:DX
  Opened FCB

Return
AL
  00H = Read completed successfully
  01H = EOF
  02H = DTA too small
  03H = EOF, partial record

DX must contain the offset (from the segment address in DS) of an opened FCB. The Current Block (offset 0CH) and Current Record (offset 20H) fields are set to agree with the Relative Record field (offset 21H), then the record addressed by these fields is loaded at the Disk Transfer Address.
AL returns a code that describes the processing:

Code Meaning

0 Read completed successfully.

1 End-of-file; no data in the record.

2 Not enough room at the Disk Transfer Address to read one record; read canceled.

3 End-of-file; a partial record was read and padded to the record length with zeros.

```
Macro Definition:  read-ran  macro   fcb
                             mov     dx,offset fcb
                             mov     ah,21H
                             int     21H
                             endm
```

Example

The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. The file contains 26 records; each record is 28 bytes long:

```
record-size     equ     14              ;offset of Record Size
                                        ;field of FCB
relative-record equu    33              ;offset of Relative Record
                                        ;field of FCB
                .
                .
fcb         db      2,“ALPHABETDAT”
            db      25  dup (?)
buffer      db      34 dup (?),“$”
prompt      db      “Enter letter: $”
crlf        db      13,10,“$”
            .
            .
func-21H:   set-dta   buffer            ;see Function 1AH
            open      fcb               ;see Function 0FH
            mov       fcb[record-size],28 ;set record size
get-char:   display   prompt            ;see Function 09H
            read-kbd-and-echo           ;see Function 01H
            cmp       al,0DH            ;just a CR?
            je        all-done          ;yes, go home
            sub       al,41H            ;convert ASCII code
                                        ;to record #
            mov       fcb[relative-record],al ;set relative
                                        ;record
            display   crlf              ;see Function 09H
            read-ran  fcb               ;THIS FUNCTION
            display   buffer            ;see Function 09H
            display   crlf              ;see Function 09H
            jmp       get-char          ;get another char.
all-done:   close     fcb               ;see Function 10H
```

Random Write (Function 22H)

Call
AH = 22H
DS:DX
  Opened FCB

Return
AL
  00H = Write completed successfully
  01H = Disk full
  02H = DTA too small

DX must contain the offset from the segment address in DS of an opened FCB. The Current Block (offset 0CH) and Current Record (offset 20H) fields are set to agree with the Relative Record field (offset 21H), then the record addressed by these fields is written from the Disk Transfer Address. If the record size is smaller than a sector (512 bytes), the records are buffered until a sector is ready to write. AL returns a code that describes the processing:

Code	Meaning
0	Write completed successfully.
1	Disk is full.
2	Not enough room at the Disk Transfer Address to write one record; write canceled.

```
Macro Definition:   write-ran     macro   fcb
                                  mov     dx,offset fcb
                                  mov     ah,22H
                                  int     21H
                                  endm
```

Example

The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. After displaying the record, it prompts the user to enter a changed record. If the user types a new record, it is written to the file; if the user just presses RETURN, the record is not replaced. The file contains 26 records; each record is 28 bytes long:

```
record-size      equu     14              ;offset of Record Size
                                          ;field of FCB
relative-record  equ      33              ;offset of Relative Record
                                          ;field of FCB
               .
               .
fcb            db     2,"ALPHABETDAT"
               db     25 dup (?)
buffer         db     26 dup (?),13,10,"$"
prompt1        db     "Enter letter: $"
prompt2        db     "New record (RETURN for no change): $"
crlf           db     13,10,"$"
reply          db     28 dup (32)
blanks         db     26 dup (32)
               .
               .
func-22H:      set-dta   buffer           ;see Function 1AH
               open      fcb              ;see Function 0FH
               mov       fcb[record-size],32 ;set record size
get-char:      display   prompt1          ;see Function 09H
               read-kbd-and-echo          ;see Function 01H
               cmp       al,0DH           ;just a CR?
               je        all-done         ;yes, go home
               sub       al,41H           ;convert ASCII
                                          ;code to record #
               mov       fcb[relative-record],al
                                          ;set relative record
               display   crlf             ;see Function 09H
               read-ran  fcb              ;THIS FUNCTION
               display   buffer           ;see Function 09H
               display   crlf             ;see Function 09H
               display   prompt2          ;see Function 09H
               get-string 27,reply        ;see Function 0AH
               display   crlf             ;see Function 09H
               cmp       reply[1],0       ;was anything typed
                                          ;besides CR?
               je        get-char         ;no
                                          ;get another char.
               xor       bx,bx            ;to load a byte
               mov       bl,reply[1]      ;use reply length as
                                          ;counter
               move-string blanks,buffer,26 ;see chapter end
               move-string reply[2],buffer,bx ;see chapter end
               write-ran fcb              ;THIS FUNCTION
               jmp       get-char         ;get another character
all-done:      close     fcb              ;see Function 10H
```

File Size (Function 23H)

Call
AH = 23H
DS:DX
  Unopened FCB

Return
AL
  00H = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened FCB. You must set the Record Size field (offset 0EH) to the proper value before calling this function. The disk directory is searched for the first matching entry.

If a matching directory entry is found, the Relative Record field (offset 21H) is set to the number of records in the file, calculated from the total file size in the directory entry (offset 1CH) and the Record Size field of the FCB (offset 0EH). AL returns 00.

If no matching directory is found, AL returns FFH (255).

NOTE

> If the value of the Record Size field of the FCB (offset 0EH) doesn't match the actual number of characters in a record, this function does not return the correct file size. If the default record size (128) is not correct, you must set the Record Size field to the correct value before using this function.

```
Macro Definition:  file-size      macro   fcb
                                  mov     dx,offset fcb
                                  mov     ah,23H
                                  int     21H
                                  endm
```

Example

The following program prompts for the name of a file, opens the file to fill in the Record Size field of the FCB, issues a File Size system call, and displays the file size and number of records in hexadecimal:

```
fcb          db        37 dup (?)
prompt       db        “File name: $”
msg1         db        “Record length:   “,13,10,“$”
msg2         db        “Records:  “,13,10,“$”
crlf         db        13,10,“$”
reply        db        17 dup (?)
sixteen      db        16
             .
             .
func-23H:    display   prompt              ;see Function 09H
             get-string 17,reply           ;see Function 0AH
             cmp       reply[1],0          ;just a CR?
             jne       get-length          ;no, keep going
             jmp       all-done            ;yes, go home
get-length:  display   crlf                ;see Function 09H
             parse     reply[2],fcb        ;see Function 29H
             open      fcb                 ;see Function 0FH
             file-size fcb                 ;THIS FUNCTION
             mov       si,33               ;offset to Relative
                                           ;Record field
             mov       di,9                ;reply in msg-2
convert-it:  cmp       fcb[si],0           ;digit to convert?
             je        show-it             ;no, prepare message
             convert   fcb[si],sixteen,msg-2[di]
             inc       si                  ;bump n-o-r index
             inc       di                  ;bump message index
             jmp       convert-it          ;check for a digit
show-it:     convert   fcb[14],sixteen,msg-1[15]
             display   msg-1               ;see Function 09H
             display   msg-2               ;see Function 09H
             jmp       func-23H            ;get a filename
all-done:    close     fcb                 ;see Function 10H
```

Set Relative Record (Function 24H)

Call
AH = 24H
DS:DX
  Opened FCB

Return
None

DX must contain the offset (from the segment address in DS) of an opened FCB. The Relative Record field (offset 21H) is set to the same file address as the Current Block (offset 0CH) and Current Record (offset 20H) fields.

```
Macro Definition:  set-relative-record  macro   fcb
                                        mov     dx,offset fcb
                                        mov     ah,24H
                                        int     21H
                                        endm
```

Example

The following program copies a file using the Random Block Read and Random Block Write system calls. It speeds the copy by setting the record length equal to the file size and the record count to 1, and using a buffer of 32K bytes. It positions the file pointer by setting the Current Record field (offset 20H) to 1 and using Set Relative Record to make the Relative Record field (offset 21H) point to the same record as the combination of the Current Block (offset 0CH) and Current Record (offset 20H) fields:

```
current-record  equu    32              ;offset of Current Record
                                        ;field of FCB
file-size       equ     16              ;offset of File Size
                                        ;field of FCB
                 .
                 .
fcb         db      37 dup (?)
filename    db      17 dup (?)
prompt1     db      "File to copy: $"   ;see Function 09H for
prompt2     db      "Name of copy: $" ;explanation of $
crlf        db      13,10,"$"
```

```
file-length  dw        ?
buffer       db        32767 dup (?)
             .
             .
func-24H:    set-dta   buffer            ;see Function 1AH
             display   prompt1           ;see Function 09H
             get-string 15, filename     ;see Function 0AH
             display   crlf              ;see Function 09H
             parse     filename[2],fcb   ;see Function 29H
             open      fcb               ;see Function 0FH
             mov       fcb[current-record],0 ;set Current Record
                                         ;field
             set-relative-record fcb     ;THIS FUNCTION
             mov       ax,word ptr fcb[file-size] ;get file size
             mov       file-length,ax    ;save it for
                                         ;ran-block-write
             ran-block-read fcb,1,ax     ;see Function 27H
             display   prompt2           ;see Function 09H
             get-string  15,filename     ;see Function 0AH
             display   crlf              ;see Function 09H
             parse     filename[2],fcb   ;see Function 29H
             create    fcb               ;see Function 16H
             mov       fcb[current-record],0 ;set Current Record
                                         ;field
             set-relative-record fcb     ;THIS FUNCTION
             mov       ax,file-length    ;get original file
                                         ;length
             ran-block-write fcb,1,ax    ;see Function 28H
             close     fcb               ;see Function 10H
```

Set Vector (Function 25H)

Call
AH = 25H
AL
  Interrupt number
DS:DX
  Interrupt-handling routine

Return
None

Function 25H should be used to set a particular interrupt vector. The operating system can then manage the interrupts on a per-process basis. Note that programs should **never** set interrupt vectors by writing them directly in the low memory vector table.
DX must contain the offset (to the segment address in DS) of an interrupt-handling routine. AL must contain the number of the interrupt handled by the routine. The address in the vector table for the specified interrupt is set to DS:DX.

Macro Definition:

```
set-vector  macro   interrupt,seg-addr,off-addr
            mov     al,interrupt
            push    ds
            mov     ax,seg-addr
            mov     ds,ax
            mov     dx,off-addr
            mov     ah,25H
            int     21H
            pop     ds
            endm
```

Example

```
lds     dx,intvector
mov     ah,25H
mov     al,intnumber
int     21H
;There are no errors returned
```

Random Block Read (Function 27H)

Call
AH = 27H
DS:DX
  Opened FCB
CX
  Number of blocks to read

Return
AL
  00H = Read completed successfully
  01H = EOF
  02H = End of segment
  03H = EOF, partial record
CX
  Number of blocks read

DX must contain the offset (to the segment address in DS) of an opened FCB. CX must contain the number of records to read; if it contains 0, the function returns without reading any records (no operation). The specified number of records - calculated from the Record Size field (offset 0EH) - is read starting at the record specified by the Relative Record field (offset 21H). The records are placed at the Disk Transfer Address.
AL returns a code that describes the processing:

Code	Meaning
0	Read completed successfully.
1	End-of-file; no data in the record.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.
3	End-of-file; a partial record was read and padded to the record length with zeros.

CX returns the number of records read; the Current Block (offset 0CH), Current Record (offset 20H), and Relative Record (offset 21H) fields are set to address the next record.

```
Macro Definition: ran-block-read macro   fcb,count,rec-size
                                 mov     dx,offset fcb
                                 mov     cx,count
                                 mov     word ptr fcb[14],rec-size
                                 mov     ah,27H
                                 int     21H
                                 endm
```

Example

The following program copies a file using the Random Block Read system call. It speeds the copy by specifying a record count of 1 and a record length equal to the file size, and using a buffer of 32 K bytes; the file is read as a single record (compare to the sample program for Function 28H that specifies a record **length** of 1 and a record **count** equal to the file size):

```
current-record  equ  32  ;offset of Current Record field
file-size       equ  16  ;offset of File Size field

           .
           .
fcb        db     37 dup (?)
filename   db     17 dup(?)
prompt1    db     "File to copy: $"   ;see Function 09H for
prompt2    db     "Name of copy: $"   ;explanation of $
crlf       db     13,10,"$"
file-length dw    ?
buffer     db     32767 dup(?)
           .
           .
func-27H:  set-dta  buffer                ;see Function 1AH
           display  prompt1               ;see Function 09H
           get-string15,filename          ;see Function 0AH
           display  crlf                  ;see Function 09H
           parse    filename[2],fcb       ;see Function 29H
           open     fcb                   ;see Function 0FH
           mov      fcb[current-record],0 ;set Current
                                          ;Record field
           set-relative-record fcb        ;see Function 24H
           mov      ax,word ptr fcb[file-size]
                                          ;get file size
           mov      file-length,ax        ;save it for
                                          ;ran-block-write
           ran-block-read  fcb,1,ax       ;THIS FUNCTION
```

```
display   prompt2             ;see Function 09H
get-string15,filename         ;see Function 0AH
display   crlf                ;see Function 09H
parse     filename[2],fcb     ;see Function 29H
create    fcb                 ;see Function 16H
mov       fcb[current-record],0
                              ;set Current Record
                              ;field
set-relative-record fcb       ;see Function 24H
mov       ax, file-length     ;get original file
                              ;size
ran-block-write   fcb,1,ax    ;see Function 28H
close     fcb                 ;see Function 10H
```

Random Block Write (Function 28H)

Call
AH = 28H
DS:DX
  Opened FCB
CX
  Number of blocks to write
  (0 = set File Size field)

Return
AL
  00H = Write completed successfully
  01H = Disk full
  02H = End of segment
CX
  Number of blocks written

DX must contain the offset (to the segment address in DS) of an opened FCB; CX must contain either the number of records to write or 0. The specified number of records (calculated from the Record Size field, offset 0EH) is written from the Disk Transfer Address. The records are written to the file starting at the record specified in the Relative Record field (offset 21H) of the FCB. If CX is 0, no records are written, but the File Size field of the directory entry (offset 1CH) is set to the number of records specified by the Relative Record field of the FCB (offset 21H); allocation units are allocated or released, as required.

AL returns a code that describes the processing:

Code	Meaning
0	Write completed successfully.
1	Disk full. No records written.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.

CX returns the number of records written; the current block (offset 0CH), Current Record (offset 20H), and Relative Record (offset 21H) fields are set to address the next record.

```
Macro Definition:  ran-block-write  macro  fcb,count,rec-size
                                    mov    dx,offset fcb
                                    mov    cx,count
                                    mov    word ptr fcb[14],
                                           rec-size
                                    mov    ah,28H
                                    int    21H
                                    endm
```

Example

The following program copies a file using the Random Block Read and Random Block Write system calls. It speeds the copy by specifying a record count equal to the file size and a record length of 1, and using a buffer of 32K bytes; the file is copied quickly with one disk access each to read and write (compare to the sample program of Function 27H, that specifies a record **count** of 1 and a record **length** equal to file size):

```
current-record equu    32              ;offset of Current Record field
file-size      equ     16              ;offset of File Size field
               .
               .
fcb            db      37 dup (?)
filename       db      17 dup(?)
prompt1        db      "File to copy: $"  ;see Function 09H for
prompt2        db      "Name of copy: $" ;explanation of $
crlf           db      13,10,"$"
num-recs       dw      ?
buffer         db      32767 dup(?)
               .
               .
func-28H:      set-dta    buffer               ;see Function 1AH
               display    prompt1              ;see Function 09H
               get-string 15, filename         ;see Function 0AH
               display    crlf                 ;see Function 09H
               parse      filename[2],fcb      ;see Function 29H
               open       fcb                  ;see Function 0FH
               mov        fcb[current-record],0
                                               ;set Current Record
                                               ;field
               set-relative-record fcb         ;see Function 24H
               mov        ax, word ptr fcb[file-size]
                                               ;get file size
```

```
mov       num-recs,ax          ;save it for
                               ;ran-block-write
ran-block-read fcb,num-recs,1 ;THIS FUNCTION
display   prompt2              ;see Function 09H
get-string15,filename          ;see Function 0AH
display   crlf                 ;see Function 09H
parse     filename[2],fcb      ;see Function 29H
create    fcb                  ;see Function 16H
mov       fcb[current-record],0 ;set Current
                               ;Record field
set-relative-record fcb        ;see Function 24H
mov       ax, file-length      ;get size of original
ran-block-write fcb,num-recs,1 ;see Function 28H
close     fcb                  ;see Function 10H
```

Parse File Name (Function 29H)

Call
AH = 29H
AL
Controls parsing (see text)
DS:SI
String to parse
ES:DI
Unopened FCB

Return
AL
00H = No wild card characters
01H = Wild-card characters used
FFH (255) = Drive letter invalid
DS:SI
First byte past string that was parsed
ES:DI
Unopened FCB

SI must contain the offset (to the segment address in DS) of a string (command line) to parse; DI must contain the offset (to the segment address in ES) of an unopened FCB. The string is parsed for a filename of the form d:filename.ext; if one is found, a corresponding unopened FCB is created at ES:DI.
Bits 0-3 of AL control the parsing and processing. Bits 4-7 are ignored:

Bit	Value	Meaning
0	0	All parsing stops if a file separator is encountered.
	1	Leading separators are ignored.
1	0	The drive number in the FCB is set to 0 (default drive) if the string does not contain a drive number.
	1	The drive number in the FCB is not changed if the string does not contain a drive number.
2	1	The filename in the FCB is not changed if the string does not contain a filename.
	0	The filename in the FCB is set to 8 blanks if the string does not contain a filename.
3	1	The extension in the FCB is not changed if the string does not contain an extension.
	0	The extension in the FCB is set to 3 blanks if the string does not contain an extension.

If the filename or extension includes an asterisk (*), all remaining characters in the name or extension are set to question mark (?).

Filename separators:

: . ; , = + / “ [ ] \ < > | space tab

Filename terminators include all the filename separators plus any control character. A filename cannot contain a filename terminator; if one is encountered, parsing stops.

If the string contains a valid filename:

1. AL returns 1 if the filename or extension contains a wild card character (* or ?); AL returns 0 if neither the filename nor extension contains a wild card character.
2. DS:SI point to the first character following the string that was parsed.
   ES:DI point to the first byte of the unopened FCB.

If the drive letter is invalid, AL returns FFH (255). If the string does not contain a valid filename, ES:DI+1 points to a blank (ASCII 20H).

Macro Definition:

```
parse       macro   string,fcb
            mov     si,offset string
            mov     di,offset fcb
            push    es
            push    ds
            pop     es
            mov     al,0FH ;bits 0, 1, 2, 3 on
            mov     ah,29H
            int     21H
            pop     es
            endm
```

Example

The following program verifies the existence of the file named in reply to the prompt:

```
fcb       db      37 dup (?)
prompt    db      “Filename: $”
reply     db      17 dup(?)
yes       db      “FILE EXISTS”,13,10,“$”
```

```
no          db        "FILE DOES NOT EXIST",13,10,"$"
            .
            .
func-29H:   display   prompt          ;see Function 09H
            get-string15,reply        ;see Function 0AH
            parse     reply[2],fcb    ;THIS FUNCTION
            search-first fcb          ;see Function 11H
            cmp       al,FFH          ;dir. entry found?
            je        not-there       ;no
            display   yes             ;see Function 09H
            jmp       continue
not-there:  display   no
continue:   .
            .
```

Get Date (Function 2AH)

Call
AH = 2AH

Return
CX
  Year (1980 - 2099)
DH
  Month (1 - 12)
DL
  Day (1 - 31)
AL
  Day of week (0=Sun., 6=Sat.)

This function returns the current date set in the operating system as binary numbers in CX and DX:

CX	Year (1980-2099)
DH	Month (1 = January, 2 = February, etc.)
DL	Day (1-31)
AL	Day of week (0 = Sunday, 1 = Monday, etc.)

```
Macro Definition:   get-date      macro
                                  mov      ah,2AH
                                  int      21H
                                  endm
```

Example

The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date:

```
month       db       31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func-2AH:   get-date                 ;see above
            inc      dl              ;increment day
            xor      bx,bx           ;so BL can be used as index
            mov      bl,dh           ;move month to index register
            dec      bx              ;month table starts with 0
            cmp      dl,month[bx]    ;past end of month?
            jle      month-ok        ;no, set the new date
            mov      dl,1            ;yes, set day to 1
```

```
          .  inc      dh            ;and increment month
             cmp      dh,12         ;past end of year?


             jle      month-ok      ;no, set the new date
             mov      dh,1          ;yes, set the month to 1
             inc      cx            ;increment year
month-ok:    set-date cx,dh,dl      ;THIS FUNCTION
```

Set Date (Function 2BH)

Call
AH = 2BH
CX
  Year (1980 - 2099)
DH
  Month (1 - 12)
DL
  Day (1 - 31)

Return
AL
  00H = Date was valid
  FFH (255) = Date was invalid

Registers CX and DX must contain a valid date in binary:

CX	Year (1980–2099)
DH	Month (1 = January, 2 = February, etc.)
DL	Day (1–31)

If the date is valid, the date is set and AL returns 0. If the date is not valid, the function is canceled and AL returns FFH (255).

```
Macro Definition:   set-date      macro   year,month,day
                                  mov     cx,year
                                  mov     dh,month
                                  mov     dl,day
                                  mov     ah,2BH
                                  int     21H
                                  endm
```

Example

The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date:

```
month       db      31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func-2BH:   get-date                ;see Function 2AH
            inc     dl              ;increment day
            xor     bx,bx           ;so BL can be used as index
```

```
          mov     bl,dh          ;move month to index register
          dec     bx             ;month table starts with 0
          cmp     dl,month[bx]   ;past end of month?
          jle     month-ok       ;no, set the new date
          mov     dl,1           ;yes, set day to 1
          inc     dh             ;and increment month
          cmp     dh,12          ;past end of year?
          jle     month-ok       ;no, set the new date
          mov     dh,1           ;yes, set the month to 1
          inc     cx             ;increment year
month-ok: set-date cx,dh,dl      ;THIS FUNCTION
```

Get Time (Function 2CH)

Call
AH = 2CH

Return
CH
  Hour (0 - 23)
CL
  Minutes (0 - 59)
DH
  Seconds (0 - 59)
DL
  Hundredths (0 - 99)

This function returns the current time set in the operating system as binary numbers in CX and DX:

CH	Hour (0–23)
CL	Minutes (0–59)
DH	Seconds (0–59)
DL	Hundredths of a second (0–99)

```
Macro Definition: get-time         macro
                                   mov      ah,2CH
                                   int      21H
                                   endm
```

Example

The following program continuously displays the time until any key is pressed:

```
time        db        "00:00:00.00",13,10,"$"
ten         db        10
            .
            .
func-2CH:   get-time                       ;THIS FUNCTION
            convert   ch,ten,time          ;see end of chapter
            convert   cl,ten,time[3]       ;see end of chapter
            convert   dh,ten,time[6]       ;see end of chapter
            convert   dl,ten,time[9]       ;see end of chapter
            display   time                 ;see Function 09H
            check-kbd-status               ;see Function 0BH
            cmp       al,FFH               ;has a key been pressed?
            je        all-done             ;yes, terminate
            jmp       func-2CH             ;no, display time
```

Set Time (Function 2DH)

Call
AH = 2DH
CH
  Hour (0 - 23)
CL
  Minutes (0 - 59)
DH
  Seconds (0 - 59)
DL
  Hundredths (0 - 99)

Return
AL
  00H = Time was valid
  FFH (255) = Time was invalid

Registers CX and DX must contain a valid time in binary:

CH	Hour (0-23)
CL	Minutes (0-59)
DH	Seconds (0-59)
DL	Hundredths of a second (0-99)

If the time is valid, the time is set and AL returns 0. If the time is not valid, the function is canceled and AL returns FFH (255).

Macro Definition:

```
set-time macro  hour,minutes,seconds,hundredths
         mov    ch,hour
         mov    cl,minutes
         mov    dh,seconds
         mov    dl,hundredths
         mov    ah,2DH
         int    21H
         endm
```

Example

The following program sets the system clock to 0 and continuously displays the time. When a character is typed, the display freezes; when another character is typed, the clock is reset to 0 and the display starts again:

```
time        db       "00:00:00.00",13,10,"$"
ten         db       10
                .

                .
func-2DH:   set-time 0,0,0,0          ;THIS FUNCTION
read-clock: get-time                  ;see Function 2CH
            convert  ch,ten,time      ;see end of chapter
            convert  cl,ten,time[3]   ;see end of chapter
            convert  dh,ten,time[6]   ;see end of chapter
            convert  dl,ten,time[9]   ;see end of chapter
            display  time             ;see Function 09H
            dir-console-io FFH        ;see Function 06H
            cmp      al,00H           ;was a char. typed?
            jne      stop             ;yes, stop the timer
            jmp      read-clock       ;no keep timer on
stop:       read-kbd                  ;see Function 08H
            jmp      func-2DH         ;keep displaying time
```

Set/Reset Verify Flag (Function 2EH)

```
Call
AH = 2EH
AL
  00H = Do not verify
  01H = Verify

Return
None
```

AL must be either 1 (verify after each disk write) or 0 (write without verifying). MS-DOS checks this flag each time it writes to a disk. The flag is normally off; you may wish to turn it on when writing critical data to disk. Because disk errors are rare and verification slows writing, you will probably want to leave it off at other times.

```
Macro Definition:   verify        macro   switch
                                  mov     al,switch
                                  mov     ah,2EH
                                  int     21H
                                  endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:, verifying each write. It uses a buffer of 32K bytes:

```
on          equu    1
off         equ     0
            .
            .
prompt      db      "Source in A, target in B",13,10
            db      "Any key to start. $"
start       dw      0
buffer      db      64 dup (512 dup(?)) ;64 sectors
            .
            .
func-2DH:   display prompt              ;see Function 09H
            read-kbd                    ;see Function 08H
            verify on                   ;THIS FUNCTION
            mov     cx,5                ;coby 64 sectors
                                        ;5 times
```

```
copy:       push      cx                  ;save counter
            abs-disk-read 0,buffer,64,start
                                          ;see Interrupt 25H
            abs-disk-write 1,buffer,64,start
                                          ;see Interrupt 26H
            add       start,64            ;do next 64 sectors
            pop       cx                  ;restore counter
            loop      copy                ;do it again
            verify    off                 ;THIS FUNCTION

disk-read 0,buffer,64,start               ;see Interrupt 25H
            abs-disk-write 1,buffer,64,start
                                          ;see Interrupt 26H
            add       start,64            ;do next 64 sectors
            pop       cx                  ;restore counter
            loop      copy                ;do it again
            verify    off
```

Get Disk Transfer Address (Function 2FH)

Call
AH = 2FH

Return
ES:BX
Points to Disk Transfer Address

Function 2FH returns the DMA transfer address.

Error returns:
None.

Example

```
        mov     ah,2FH
        int     21H
                ;es:bx has current DMA transfer address
```

Get DOS Version Number (Function 30H)

Call
AH = 30H

Return
AL
  Major version number
AH
  Minor version number

This function returns the MS-DOS version number. On return, AL.AH will be the two-part version designation; i.e., for MS-DOS 1.28, AL would be 1 and AH would be 28. For pre-1.28, DOS AL = 0. Note that version 1.1 is the same as 1.10, not the same as 1.01.

Error returns:
None.

Example

```
mov     ah,30
int     21H
       ; al is the major version number
       ; ah is the minor version number
       ; bh is the OEM number
       ; bl:cx is the (24 bit) user number
```

Keep Process (Function 31H)

Call
AH = 31H
AL
  Exit code
DX
  Memory size, in paragraphs

Return
None

This call terminates the current process and attempts to set the initial allocation block to a specific size in paragraphs. It will not free up any other allocation blocks belonging to that process. The exit code passed in AX is retrievable by the parent via Function 4DH.
This method is preferred over Interrupt 27H and has the advantage of allowing more than 64K to be kept.

Error returns:
None.

Example

```
mov     al, exitcode
mov     dx, parasize
mov     ah, 31H
int     21H
```

CONTROL-C Check (Function 33H)

```
Call
AH = 33H
AL
  Function
  00H =Request current state
  01H = Set state
DL (if setting)
  00H = Off
  01H = On

Return
DL
  00H = Off
  01H = On
```

MS-DOS ordinarily checks for a CONTROL-C on the controlling device only when doing function call operations 01H-0CH to that device. Function 33H allows the user to expand this checking to include any system call. For example, with the CONTROL-C trapping off, all disk I/0 will proceed without interruption; with CONTROL-C trapping on, the CONTROL-C interrupt is given at the system call that initiates the disk operation.

NOTE

> Programs that wish to use calls 06H or 07H to read CONTROL-Cs as data must ensure that the CONTROL-C check is off.

```
Error return:
AL = FF
     The function passed in AL was not in the range 0:1.
```

Example

```
mov    dl,val
mov    ah,33H
mov    al,func
```

```
int         21H
            ; If al was 0, then dl has the current value
            ;of the CONTROL-C check
```

Get Interrupt Vector (Function 35H)

Call
AH = 35H
AL
  Interrupt number

Return
ES:BX
  Pointer to interrupt routine

This function returns the interrupt vector associated with an interrupt. Note that programs should **never** get an interrupt vector by reading the low memory vector table directly.

Error returns:
None.

Example

```
mov     ah,35H
mov     al,interrupt
int     21H
    ; es:bx now has long pointer to interrupt routine
```

Get Disk Free Space (Function 36H)

Call
AH = 36H
DL
  Drive (0 = Default,
  1 = A, etc.)

Return
BX
  Available clusters
DX
  Clusters per drive
CX
  Bytes per sector
AX
  FFFF if drive number is invalid;
  otherwise sectors per cluster

This function returns free space on disk along with additional information about the disk.

Error returns:
AX = FFFF
  The drive number given in DL was invalid.

Example

```
mov     ah,36H
mov     dl,Drive         ;0 = default, A = 1
int     21H
    ; bx = Number of free allocation units on drive
    ; dx = Total number of allocation units on drive
    ; cx = Bytes per sector
    ; ax = Sectors per allocation unit
```

Return Country-Dependent Information (Function 38H)

Call
AH = 38H
DS:DX
  Pointer to 32-byte memory area
AL
  Function code. In MS-DOS 2.0,
  must be 0

Return
Carry set:
AX
  2 = file not found
Carry not set:
  DX:DS filled in with country data

The value passed in AL is either 0 (for current country) or a country code. Country codes are typically the international telephone prefix code for the country.
If DX = -1, then the call sets the current country (as returned by the AL = 0 call) to the country code in AL. If the country code is not found, the current country is not changed.

NOTE

> Applications must assume 32 bytes of information. This means the buffer pointed to by DS:DX must be able to accommodate 32 bytes.

This function is fully supported only in versions of MS-DOS 2.01 and higher. It exists in MS-DOS 2.0, but is not fully implemented.
This function returns, in the block of memory pointed to by DS:DX, the following information pertinent to international applications:

WORD Date/time format
5 BYTE ASCIZ string currency symbol
2 BYTE ASCIZ string thousands separator
2 BYTE ASCIZ string decimal separator
2 BYTE ASCIZ string date separator
2 BYTE ASCIZ string time separator
1 BYTE Bit field
1 BYTE Currency places
1 BYTE time format
DWORD Case Mapping call
2 BYTE ASCIZ string data list separator

The format of most of these entries is ASCIZ (a NUL terminated ASCII string), but a fixed size is allocated for each field for easy indexing into the table.
The date/time format has the following values:

0 - USA standard	h:m:s	m/d/y
1 - Europe standard	h:m:s	d/m/y
2 - Japan standard	y/m/d	h:m:s

The bit field contains 8 bit values. Any bit not currently defined must be assumed to have a random value.

Bit 0 = 0 If currency symbol precedes the currency amount.
= 1 If currency symbol comes after the currency amount.
Bit 1 = 0 If the currency symbol immediately precedes the currency amount.
= 1 If there is a space between the currency symbol and the amount.

The time format has the following values:

0 - 12 hour time
1 - 24 hour time

The currency places field indicates the number of places which appear after the decimal point on currency amounts.
The Case Mapping call is a FAR procedure which will perform country specific lower-to-uppercase mapping on character values from 80H to FFH. It is called with the character to be mapped in AL. It returns the correct upper case code for that character, if any, in AL. AL and the FLAGS are the only registers altered. It is allowable to pass this routine codes below 80H; however nothing is done to characters in this range. In the case where there is no mapping, AL is not altered.

Error returns:
AX
2 = file not found
The country passed in AL was not found (no table for specified country).

Example

```
lds     dx, blk
mov     ah, 38H
mov     al, Country-code
int     21H
   ;AX = Country code of country returned
```

Create Sub-Directory (Function 39H)

Call
AH = 39H
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  3 = path not found
  5 = access denied
Carry not set:
  No error

Given a pointer to an ASCIZ name, this function creates a new directory entry at the end.

Error returns:
AX
 3 = path not found
  The path specified was invalid or not found.
 5 = access denied
  The directory could not be created (no room in parent directory), the directory/file already existed or a device name was specified.

Example

```
lds     dx, name
mov     ah, 39H
int     21H
```

Remove a Directory Entry (Function 3AH)

Call
AH = 3AH
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  3 = path not found
  5 = access denied
  16 = current directory
Carry not set:
  No error

Function 3AH is given an ASCIZ name of a directory. That directory is removed from its parent directory.

Error returns:
AX
  3 = path not found
    The path specified was invalid or not found.
  5 = access denied
    The path specified was not empty, not a directory, the root directory, or contained invalid information.
16 = current directory
    The path specified was the current directory on a drive.

Example

```
    lds     dx, name
    mov     ah, 3AH
    int     21H
```

Change the Current Directory (Function 3BH)

Call
AH = 3BH
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  3 = path not found
Carry not set:
  No error

Function 3BH is given the ASCIZ name of the directory which is to become the current directory. If any member of the specified pathname does not exist, then the current directory is unchanged. Otherwise, the current directory is set to the string.

Error returns:
AX
  3 = path not found
    The path specified in DS:DX either indicated a file or the path was invalid.

Example

```
lds     dx, name
mov     ah, 3BH
int     21H
```

Create a File (Function 3CH)

Call
AH = 3CH
DS:DX
  Pointer to pathname
CX
  File attribute

Return
Carry set:
AX
  5 = access denied
  3 = path not found
  4 = too many open files
Carry not set:
  AX is handle number

Function 3CH creates a new file or truncates an old file to zero length in preparation for writing. If the file did not exist, then the file is created in the appropriate directory and the file is given the attribute found in CX. The file handle returned has been opened for read/write access.

Error returns:
AX
5 = access denied
    The attributes specified in CX contained one that could not be created (directory, volume ID), a file already existed with a more inclusive set of attributes, a directory existed with the same name, or the path was not found.
3 = path not found
    The path specified had a syntax error.
4 = too many open files
    The file was created with the specified attributes, but there were no free handles available for the process, or the internal system tables were full.

Example

```
lds         dx, name
mov         ah, 3CH
mov         cx, attribute
int         21H
  ; ax now has the handle
```

Open a File (Function 3DH)

Call
AH = 3DH
AL
  Access
  0 = File opened for reading
  1 = File opened for writing
  2 = File opened for both
  reading and writing

Return
Carry set:
AX
  12 = invalid access
  2 = file not found
  5 = access denied
  4 = too many open files
Carry not set:
  AX is handle number

Function 3DH associates a 16-bit file handle with a file.
The following values are allowed:

ACCESS	**Function**
0	file is opened for reading
1	file is opened for writing
2	file is opened for both reading and writing.

DS:DX point to an ASCIZ name of the file to be opened.

The read/write pointer is set at the first byte of the file and the record size of the file is 1 byte. The returned file handle must be used for subsequent I/O to the file.

Error returns:

AX

12 = invalid access

The access specified in AL was not in the range 0:2.

2 = file not found

The path specified was invalid or not found.

5 = access denied

The user attempted to open a directory or volume-id, or open a read-only file for writing.

4 = too many open files

There were no free handles available in the current process or the internal system tables were full.

Example

```
lds     dx, name
mov     ah, 3DH
mov     al, access
int     21H
   ; ax has error or file handle
   ; If successful open
```

Close a File Handle (Function 3EH)

Call
AH = 3EH
BX
  File handle

Return
Carry set:
AX
  6 = invalid handle
Carry not set:
  No error

In BX is passed a file handle (like that returned by Functions 3DH, 3CH, or 45H), Function 3EH closes the associated file. Internal buffers are flushed.

Error return:
AX
  6 = invalid handle
    The handle passed in BX was not currently open.

Example

```
        mov     bx, handle
        mov     ah, 3EH
        int     21H
```

Read From File/Device (Function 3FH)

Call
AH = 3FH
DS:DX
  Pointer to buffer
CX
  Bytes to read
BX
  File handle

Return
Carry set:
AX
  Number of bytes read
  6 = invalid handle
  5 = error set:
Carry not set:
  AX = number of bytes read

Function 3FH transfers count bytes from a file into a buffer location. It is not guaranteed that all count bytes will be read; for example, reading from the keyboard will read at most one line of text. If the returned value is zero, then the program has tried to read from the end of file.

All I/O is done using normalized pointers; no segment wraparound will occur.

Error returns:
AX
  6 = invalid handle
    The handle passed in BX was not currently open.
  5 = access denied
    The handle passed in BX was opened in a mode that did not allow reading.

Example

```
lds     dx, buf
mov     cx, count
mov     bx, handle
mov     ah, 3FH
int     21H
   ; ax has number of bytes read
```

Write to a File or Device (Function 40H)

Call
AH = 40H
DS:DX
  Pointer to buffer
CX
  Bytes to write
BX
  File handle

Return
Carry set:
AX
  Number of bytes written
  6 = invalid handle
  5 = access denied
Carry not set:
AX = number of bytes written

Function 40H transfers count bytes from a buffer into a file. It should be regarded as an error if the number of bytes written is not the same as the number requested.
The write system call with a count of zero (CX = 0) will set the file size to the current position. Allocation units are allocated or released as required.
All I/O is done using normalized pointers; no segment wraparound will occur.

Error returns:
AX
  6 = invalid handle
    The handle passed in BX was not currently open.
  5 = access denied
    The handle was not opened in a mode that allowed writing.

Example
```
    lds     dx, buf
    mov     cx, count
    mov     bx, handle
    mov     ah, 40H
    int     21H
      ;ax has number of bytes written
```

Delete a Directory Entry (Function 41H)

Call
AH = 41H
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  2 = file not found
  5 = access denied
Carry not set:
  No error

Function 41H removes a directory entry associated with a filename.

Error returns:
AX
  2 = file not found
    The path specified was invalid or not found.
  5 = access denied
    The path specified was a directory or read-only.

Example

```
lds          dx, name
mov          ah, 41H
int          21H
```

Move File Pointer (Function 42H)

Call
AH = 42H
CX:DX
Distance to move, in bytes
AL
Method of moving:
(see text)
BX
File handle

Return
Carry set:
AX
6 = invalid handle
1 = invalid function
Carry not set:
DX:AX = new pointer location

Function 42H moves the read/write pointer according to one of the following methods:

Method	**Function**
0	The pointer is moved to offset bytes from the beginning of the file.
1	The pointer is moved to the current location plus offset.
2	The pointer is moved to the end of file plus offset.

Offset should be regarded as a 32-bit integer with CX occupying the most significant 16 bits.

Error returns:
AX
6 = invalid handle
The handle passed in BX was not currently open.
1 = invalid function
The function passed in AL was not in the range 0:2.

Example

```
mov     dx, offsetlow
mov     cx, offsethigh
mov     al, method
mov     bx, handle
mov     ah, 42H
int     21H
  ; dx:ax has the new location of the pointer
```

Change Attributes (Function 43H)

Call
AH = 43H
DS:DX
  Pointer to pathname
CX (if AL = 01)
  Attribute to be set
AL
  Function
  01 Set to CX
  00 Return in CX

Return
Carry set:
AX
  3 = path not found
  5 = access denied
  1 = invalid function
Carry not set:
  CX attributes (if AL = 00)

Given an ASCIZ name, Function 42H will set/get the attributes of the file to those given in CX.
A function code is passed in AL:

AL	**Function**
0	Return the attributes of the file in CX.
1	Set the attributes of the file to those in CX.

Error returns:
AX
3 = path not found
  The path specified was invalid.
5 = access denied
  The attributes specified in CX contained one that could not be changed (directory, volume ID).
1 = invalid function
  The function passed in AL was not in the range 0:1.

Example

```
lds     dx, name
mov     cx, attribute
mov     al, func
int     ah, 43H
int     21H
```

I/O Control for Devices (Function 44H)

Call
AH = 44H
BX
  Handle
BL
  Drive (for calls AL = 4, 5
  0 = default, 1 = A, etc.)
DS:DX
  Data or buffer
CX
  Bytes to read or write
AL
  Function code; see text
Return
Carry set:
AX
  6 = invalid handle
  1 = invalid function
  13 = invalid data
  5 = access denied
Carry not set:
AL = 2,3,4,5
AX = Count transferred
AL = 6,7
  00 = Not ready
  FF = Ready

Function 44H sets or gets device information associated with an open handle, or send/receives a control string to a device handle or device. The following values are allowed for function:

Request	Function
0	Get device information (returned in DX)
1	Set device information (as determined by DX)
2	Read CX number of bytes into DS:DX from device control channel.
3	Write CX number of bytes from DS:DX to device control channel.
4	Same as 2 only drive number in BL 0=default,A:=1,B:=2,...
5	Same as 3 only drive number in BL 0=default,A:=1,B:=2,...
6	Get input status
7	Get output status

This function can be used to get information about device channels. Calls can be made on regular files, but only calls 0,6 and 7 are defined in that case (AL=0,6,7). All other calls return an invalid function error.

Calls AL=0 and AL=1

The bits of DX are defined as follows for calls AL=0 and AL=1. Note that the upper byte MUST be zero on a set call.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	CTRL	Reserved						ISDEV	EOF	RAW	SPECL	ISCLK	ISNUL	ISCOT	ISCIN

ISDEV = 1 if this channel is a device
= 0 if this channel is a disk file (Bits 8-15 = 0 in this case)

If ISDEV = 1
EOF = 0 if End Of File on input
RAW = 1 if this device is in Raw mode
= 0 if this device is cooked
ISCLK = 1 if this device is the clock device
ISNUL = 1 if this device is the null device
ISCOT = 1 if this device is the console output
ISCIN = 1 if this device is the console input
SPECL = 1 if this device is special

CTRL = 0 if this device can not do control strings via calls AL=2 and AL=3.
CTRL = 1 if this device can process control strings via calls AL=2 and AL=3.
NOTE that this bit cannot be set.

If ISDEV = 0
EOF = 0 if channel has been written
Bits 0-5 are the block device number for the channel (0 = A:, 1 = B:, ...)
Bits 15,8-13,4 are reserved and should not be altered.

Calls 2..5:

These four calls allow arbitrary control strings to be sent or received from a device. The call syntax is the same as the read and write calls, except for 4 and 5, which take a drive number in BL instead of a handle in BX.

An invalid function error is returned if the CTRL bit (see above) is 0.

An access denied is returned by calls AL=4,5 if the drive number is invalid.

Calls 6,7:

These two calls allow the user to check if a file handle is ready for input or output. Status of handles open to a device is the intended use of these calls, but status of a handle open to a disk file is allowed, and is defined as follows:

Input:

Always ready (AL=FF) until EOF reached, then always not ready (AL=0) unless current position changed via LSEEK.

Output:

Always ready (even if disk full).

IMPORTANT

The status is defined at the time the system is CALLED. On future versions, by the time control is returned to the user from the system, the status returned may NOT correctly reflect the true current state of the device or file.

Error returns:

AX

6 = invalid handle

The handle passed in BX was not currently open.

1 = invalid function

The function passed in AL was not in the range 0:7.

13 = invalid data

5 = access denied (calls AL=4..7)

```
Example
    mov        bx, Handle
(or mov        bl, drive      for calls AL=4,5
                              0=default,A:=1...)
    mov        dx, Data
(or lds        dx, buf        and
    mov        cx, count      for calls AL=2,3,4,5)
    mov        ah, 44H
    mov        al, func
    int        21H
   ; For calls AL=2,3,4,5 AX is the number of bytes
   ; transferred (same as READ and WRITE).
   ; For calls AL=6,7 AL is status returned, AL=0 if
   ; status is not ready, AL=0FFH otherwise.
```

Duplicate a File Handle (Function 45H)

Call
AH = 45H
BX
  File handle

Return
Carry set:
AX
  6 = invalid handle
  4 = too many open files
Carry not set:
AX = new file handle

Function 45H takes an already opened file handle and returns a new handle that refers to the same file at the same position.

Error returns:
AX
  6 = Invalid handle
    The handle passed in BX was not currently open.
  4 = too many open files
    There were no free handles available in the current process or the internal system tables were full.

Example

```
mov     bx, fh
mov     ah, 45H
int     21H
   ; ax has the returned handle
```

Force a Duplicat of a Handle (Function 46H)

Call
AH = 46H
BX
  Existing file handle
CX
  New file handle

Return
Carry set:
AX
  6 = invalid handle
  4 = too many open files
Carry not set:
  No error

Function 46H takes an already opened file handle and returns a new handle that refers to the same file at the same position. If there was already a file open on handle CX, it is closed first.

Error returns:
AX
  6 = invalid handle
    The handle passed in BX was not currently open.
  4 = too many open files
    There were no free handles available in the current process or the internal system tables were full.

Example

```
mov    bx, fh
mov    cx, newfh
mov    ah, 46H
int    21H
```

Return Text of Current Directory (Function 47H)

Call
AH = 47 H
DS:SI
  Pointer to 64-byte memory area
DL
  Drive number

Return
Carry set:
AX
  15 = invalid drive
Carry not set:
  No error

Function 47H returns the current directory for a particular drive. The directory is root-relative and does not contain the drive specifier or leading path separator. The drive code passed in DL is 0=default, 1=A:, 2=B:, etc.

Error returns:
AX
15 = invalid drive
  The drive specified in DL was invalid.

Example

```
    mov     ah, 47H
    lds     si,area
    mov     dl,drive
    int     21H
      ; ds:si is a pointer to 64 byte area that
      ; contains drive current directory.
```

Allocate Memory (Function 48H)

Call
AH = 48H
BX
  Size of memory to be allocated

Return
Carry set:
AX
  8 = not enough memory
  7 = arena trashed
BX
  Maximum size that could be allocated
Carry not set:
AX:0
  Pointer to the allocated memory

Function 48H returns a pointer to a free block of memory that has the requested size in paragraphs.

Error return:
AX
  8 = not enough memory
    The largest available free block is smaller than that requested or there is no free block.
  7 = arena trashed
    The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.

Example

```
    mov     bx,size
    mov     ah,48H
    int     21H
      ; ax:0 is pointer to allocated memory
      ; if alloc fails, bx is the largest block available
```

Free Allocated Memory (Function 49H)

Call
AH = 49H
ES
  Segment address of memory
  area to be freed

Return
Carry set:
AX
  9 = invalid block
  7 = arena trashed
Carry not set:
  No error

Function 49H returns a piece of memory to the system pool that was allocated by Function Request 48H.

Error return:
AX
  9 = invalid block
    The block passed in ES is not one allocated via Function Request 48H.
  7 = arena trashed
    The internal consistency of the memory arena has been destroyed. This is duė to a user program changing memory that does not belong to it.

Example

```
    mov     es,block
    mov     ah,49H
    int     21H
```

Modify Allocated Memory Blocks (Function 4AH)

Call
AH = 4AH
ES
  Segment address of memory area
BX
  Requested memory area size

Return
Carry set:
AX
  9 = invalid block
  7 = arena trashed
  8 = not enough memory
BX
  Maximum size possible
Carry not set:
  No error

Function 4AH will attempt to grow/shrink an allocated block of memory.

Error return:
AX
  9 = invalid block
    The block passed in ES is not one allocated via this function.
  7 = arena trashed
    The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.
  8 = not enough memory
    There was not enough free memory after the specified block to satisfy the grow request.

Example

```
    mov     es,block
    mov     bx,newsize
    mov     ah,4AH
    int     21H
       ; if setblock fails for growing, BX will have the
       ; maximum size possible
```

Load and Execute a program (Function 4BH)

Call
AH = 4BH
DS:DX
Pointer to pathname
ES:BX
Pointer to parameter block
AL
00 = Load and execute program
03 = Load program

Return
Carry set:
AX
1 = invalid function
10 = bad environment
11 = bad format
8 = not enough memory
2 = file not found
Carry not set:
No error

This function allows a program to load another program into memory and (default) begin execution of it. DS:DX points to the ASCIZ name of the file to be loaded. ES:BX points to a parameter block for the load.
A function code is passed in AL:

AL Function

0 Load and execute the program. A program header is established for the program and the terminate and CONTROL-C addresses are set to the instruction after the EXEC system call.

3 Load (do not create) the program header, and do not begin execution. This is useful in loading program overlays.

For each value of AL, the block has the following format:

AL = 0 –> load/execute program

WORD segment address of environment.
DWORD pointer to command line at 80H
DWORD pointer to default FCB to be passed at 5CH
DWORD pointer to default FCB to be passed at 6CH

AL = 3 –> load overlay

WORD segment address where file will be loaded.
WORD relocation factor to be applied to the image.

Note that all open files of a process are duplicated in the child process after an EXEC. This is extremely powerful; the parent process has control over the meanings of stdin, stdout, stderr, stdaux and stdprn. The parent could, for example, write a series of records to a file, open the file as standard input, open a listing file as standard output and then EXEC a sort program that takes its input from stdin and writes to stdout.

Also inherited (or passed from the parent) is an “environment”. This is a block of text strings (less than 32K bytes total) that convey various configurations parameters. The format of the environment is as follows:

(paragraph boundary)

BYTE ASCIZ string 1
BYTE ASCIZ string 2
. . .
BYTE ASCIZ string n
BYTE of zero

Typically the environment strings have the form:
parameter = value
For example, COMMAND.COM might pass its execution search path as:
PATH = A:XBIN;B:XBASICXLIB

A zero value of the environment address causes the child process to inherit the parent's environment unchanged.

Error returns:
AX
1 = invalid function
The function passed in AL was not 0, 1 or 3.
10 = bad environment
The environment was larger than 32Kb.
11 = bad format
The file pointed to by DS:DX was an EXE format file and contained information that was internally inconsistent.
8 = not enough memory
There was not enough memory for the process to be created.
2 = file not found
The path specified was invalid or not found.

Example

```
lds     dx, name
les     bx, blk
mov     ah, 4BH
mov     al, func
int     21H
```

Terminate a Process (Function 4CH)

Call
AH = 4CH
AL
  Return code

Return
None

Function 4CH terminates the current process and transfers control to the invoking process. In addition, a return code may be sent. All files open at the time are closed.

This method is preferred over all others (Interrupt 20H, JMP 0) and has the advantage that CS:0 does not have to point to the Program Header Prefix.

Error returns:
  None.

Example

```
        mov     al, code
        mov     ah, 4CH
        int     21H
```

Retrieve the Return Code of a Child (Function 4DH)

Call
AH = 4DH

Return
AX
  Exit Code

Function 4DH returns the Exit code specified by a child process. It returns this Exit code only once. The low byte of this code is that sent by the Exit routine. The high byte is one of the following:

0 = Terminate/abort
1 = CONTROL-C
2 = Hard error
3 = Terminate and stay resident

Error returns:
  None.

Example

```
        mov     ah, 4DH
        int     21H
          ; ax has the exit code
```

Find Match File (Function 4EH)

Call
AH = 4EH
DS:DX
  Pointer to pathname
CX
  Search attributes

Return
Carry set:
AX
  2 = file not found
  18 = no more files
Carry not set:
  no error

Function 4EH takes a pathname with wild card characters in the last component (passed in DS:DX), a set of attributes (passed in CX) and attempts to find all files that match the pathname and have a subset of the required attributes. A datablock at the current DMA is written that contains information in the following form:

```
find-buf-reserved  DB   21  DUP (?); Reserved*
find-buf-attr      DB   ?   ; attribute found
find-buf-time      DW   ?   ; time
find-buf-date      DW   ?   ; date
find-buf-size-l    DW   ?   ; low(size)
find-buf-size-h    DW   ?   ; high(size)
find-buf-pname     DB   13  DUP (?) ; packed name
find-buf    ENDS
```

*Reserved for MS-DOS use on subsequent find-nexts

To obtain the subsequent matches of the pathname, see the description of Function 4FH.

Error returns:
AX
 2 = file not found
    The path specified in DS:DX was an invalid path.
18 = no more files
    There were no files matching this specification.

Example

```
        mov     ah, 4EH
        lds     dx, pathname
        mov     cx, attr
        int     21H
            ; dma address has datablock
```

Step Through a Directory Matching Files (Function 4FH)

Call
AH = 4FH

Return
Carry set:
AX
  18 = no more files
Carry not set:
  No error

Function 4FH finds the next matching entry in a directory. The current DMA address must point at a block returned by Function 4EH (see Function 4EH).

Error returns:
AX
18 = no more files
  There are no more files matching this pattern.

Example

```
   ; dma points at area returned by Function 4FH
mov     ah, 4FH
int     21H
   ; next entry is at dma
```

Return Current Setting of Verify After Write Flag (Function 54H)

Call
AH = 54H

Return
AL
  Current verify flag value

The current value of the verify flag is returned in AL.

Error returns:
  None.

Example

```
        mov     ah, 54H
        int     21H
          ; al is the current verify flag value
```

Move a Directory Entry (Function 56H)

Call
AH = 56H
DS:DX
  Pointer to pathname of
  existing file
ES:DI
  Pointer to new pathname

Return
Carry set:
AX
   2 = file not found
  17 = not same device
   5 = access denied
Carry not set:
  No error

Function 56H attempts to rename a file into another path. The paths must be on the same device.

Error returns:
AX
 2 = file not found
  The file name specifed by DS:DX was not found.
17 = not same device
  The source and destination are on different drives.
 5 = access denied
  The path specified in DS:DX was a directory or the file specified by ES:DI exists or the destination directory entry could not be created.

Example

```
        lds     dx, source
        les     di, dest
        mov     ah, 56H
        int     21H
```

Get/Set Date/Time of File (Function 57H)

Call
AH = 57H
AL
  00 = get date and time
  01 = set date and time
BX
  File handle
CX (if AL = 01)
  Time to be set
DX (if AL = 01)
  Date to be set

Return
Carry set:
AX
  1 = invalid function
  6 = invalid handle
Carry not set:
No error
CX/DX set if function 0

Function 57H returns or sets the last-write time for a handle. These times are not recorded until the file is closed.
A function code is passed in AL:

AL	Function
0	Return the time/date of the handle in CX/DX
1	Set the time/date of the handle to CX/DX

Error returns:
AX
 1 = invalid function
  The function passed in AL was not in the range 0:1.
 6 = invalid handle
  The handle passed in BX was not currently open.

Example

```
mov     ah, 57H
mov     al, func
mov     bx, handle
      ; if al = 1 then next two are mandatory
mov     cx, time
mov     dx, date
int     21H
    ;  if al = 0 then cx/dx has the last write time/date
    ;  for the handle.
```

1.8 MACRO DEFINITIONS FOR MS-DOS SYSTEM CALL EXAMPLES

NOTE

These macro definitions apply to system call examples 00H through 57H.

```
.xlist
;
;******************
;
; Interrupts
;******************
;

                                             ;ABS-DISK-READ
;abs-disk-read macro disk,buffer,num-sectors,first-sector
               mov     al,disk
               mov     bx,offset buffer
               mov     cx,num-sectors
               mov     dx,first-sector
               int     37                    ;interrupt 37
               popf
               endm
;
                                             ;ABS-DISK-WRITE
abs-disk-write macro disk,buffer,num-sectors,first-sector
               mov     al,disk
               mov     bx,offset buffer
               mov     cx,num-sectors
               mov     dx,first-sector
               int     38                    ;interrupt 38
               popf
               endm
;
stay-resident macro last-instruc             ;STAY-RESIDENT
               mov     dx,offset last-instruc
               inc     dx
               int     39                    ;interrupt 39
               endm
;
;******************
;
; Functions
;******************
;
;
read-kbd-and-echo macro                      ;READ-KBD-AND-ECHO
               mov     ah,1                  ;function 1
               int     33
               endm
;
display-char macro character                 ;DISPLAY-CHAR
               mov     dl,character
               mov     ah,2                  ;function 2
```

```
                int     33
                endm
;
aux-input macro                         ;AUX-INPUT
                mov     ah,3            ;function 3
                int     33
                endm
;
aux-output macro                        ;AUX-OUTPUT
                mov     ah,4            ;function 4
                int     33
                endm
;;page
print-char macro        character       ;PRINT-CHAR
                mov     dl,character
                mov     ah,5            ;function 5
                int     33
                endm
;
dir-console-io macro switch             ;DIR-CONSOLE-IO
                mov     dl,switch
                mov     ah,6            ;function 6
                int     33
                endm
;
dir-console-input macro                 ;DIR-CONSOLE-INPUT
                mov     ah,7            ;function 7
                int     33
                endm
;
read-kbd        macro                   ;READ-KBD
                mov     ah,8            ;function 8
                int     33
                endm
;
display         macro   string          ;DISPLAY
                mov     dx,offset string
                mov     ah,9            ;function 9
                int     33
                endm
;
get-string macro        limit,string    ;GET-STRING
                mov     String,limit
                mov     dx,offset string
                mov     ah,10           ;function 10
                int     33
                endm
;
check-kbd-status macro                  ;CHECK-KBD-STATUS
                mov     ah,11           ;function 11
                int     33
                endm
;
```

```
flush-and-read-kbd  macro switch  ;FLUSH-AND-READ-KBD
            mov     al,switch
            mov     ah,12         ;function 12
            int     33
            endm
;
reset-disk macro                  ;RESET DISK
            mov     ah,13         ;function 13
            int     33
            endm
;;page
select-disk macro   disk          ;SELECT-DISK
            mov     dl,disk[-65]
            mov     ah,14         ;function 14
            int     33
            endm
;
open        macro   fcb           ;OPEN
            mov     dx,offset fcb
            mov     ah,15         ;function 15
            int     33
            endm
;
close       macro   fcb           ;CLOSE
            mov     dx,offset fcb
            mov     ah,16         ;function 16
            int     33
            endm
;
search-first macro  fcb           ;SEARCH-FIRST
            mov     dx,offset fcb
            mov     ah,17         ;Function 17
            int     33
            endm
;
search-next macro   fcb           ;SEARCH-NEXT
            mov     dx,offset fcb
            mov     ah,18         ;function 18
            int     33
            endm
;
delete      macro   fcb           ;DELETE
            mov     dx,offset fcb
            mov     ah,19         ;function 19
            int     33
            endm
;
read-seq    macro   fcb           ;READ-SEQ
            mov     dx,offset fcb
            mov     ah,20         ;function 20
            int     33
            endm
;
```

```
write-seq macro           fcb               ;WRITE-SEQ
                mov       dx,offset fcb
                mov       ah,21             ;function 21
                int       33
                endm
;
create          macro     fcb               ;CREATE
                mov       dx,offset fcb
                mov       ah,22             ;function 22
                int       33
                endm
;
rename          macro     fcb,newname       ;RENAME
                mov       dx,offset fcb
                mov       ah,23             ;function 23
                int       33
                endm
;
current-disk macro                          ;CURRENT-DISK
                mov       ah,25             ;function 25
                int       33
                endm
;
set-dta         macro     buffer            ;SET-DTA
                mov       dx,offset buffer
                mov       ah,26             ;function 26
                int       33
                endm
;
alloc-table macro                           ;ALLOC-TABLE
                mov       ah,27             ;function 27
                int       33
                endm
;
read-ran        macro     fcb               ;READ-RAN
                mov       dx,offset fcb
                mov       ah,33             ;function 33
                int       33
                endm
;
write-ran       macro     fcb               ;WRITE-RAN
                mov       dx,offset fcb
                mov       ah,34             ;function 34
                int       33
                endm
;
file-size       macro     fcb               ;FILE-SIZE
                mov       dx,offset fcb
                mov       ah,35             ;function 35
                int       33
                endm
;
```

```
set-relative-record macro fcb           ;SET-RELATIVE-RECORD
                mov     dx,offset fcb
                mov     ah,36           ;function 36
                int     33
                endm
;;page
set-vector macro interrupt,seg-addr,off-addr ;SET-VECTOR
                push
                mov     ax,seg-addr
                mov     ds,ax
                mov     dx,off-addr
                mov     al,interrupt
                mov     ah,37           ;function 37
                int     33
                endm
;
create-prog-seg macro seg-addr          ;CREATE-PROG-SEG
                mov     dx,seg-addr
                mov     ah,38           ;function 38
                int     33
                endm
;
ran-block-read macro fcb,count,rec-size ;RAN-BLOCK-READ
                mov     dx,offset fcb
                mov     cx,count
                mov     word ptr fcb[14],rec-size
                mov     ah,39           ;function 39
                int     33
                endm
;
ran-block-write macro fcb,count,rec-size ;RAN-BLOCK-WRITE
                mov     dx,offset fcb
                mov     cx,count
                mov     word ptr fcb[14],rec-size
                mov     ah,40           ;function 40
                int     33
                endm
;
parse           macro   filename,fcb    ;PARSE
                mov     si,offset filename
                mov     di,offset fcb
                push    es
                push    ds
                pop     es
                mov     al,15
                mov     ah,41           ;function 41
                int     33
                pop     es
                endm
;
get-date        macro                   ;GET-DATE
                mov     ah,42           ;function 42
                int     33
                endm
;;page
set-date        macro     year,month,day  ;SET-DATE
                mov       cx,year
                mov       dh,month
                mov       dl,day
                mov       ah,43           ;function 43
                int       33
                endm
;
get-time        macro                     ;GET-TIME
                mov       ah,44           ;function 44
                int       33
                endm
;

set-time        macro     hour,minutes,seconds,hundredths ;SET-TIME
                mov       ch,hour
                mov       cl,minutes
                mov       dh,seconds
                mov       dl,hundredths
                mov       ah,45           ;function 45
                int       33
                endm
;
verify          macro     switch          ;VERIFY
                mov       al,switch
                mov       ah,46           ;function 46
                int       33
                endm
;
;*******************
; General
;*******************
;
move-string macro source,destination,num-bytes ;MOVE-STRING
                push      es
                mov       ax,ds
                mov       es,ax
                assume    es:data
                mov       si,offset source
                mov       di,offset  destina-
                          tion
                mov       cx,num-bytes
            rep movs      es:destination,source
                assume    es:nothing
                pop       es
                endm
;
;
convert         macro     value,base,destination ;CONVERT
                local     table,start
                jmp       start
```

```
table           db          ”0123456789ABCDEF“
start:          mov         al,value
                xor         ah,ah
                xor         bx,bx
                div         base
                mov         bl,al
                mov         al,cs:table[bx]
                mov         destination,al
                mov         bl,ah
                mov         al,cs:table[bx]
                mov         destination[1],al
                endm
;;page
convert-to-binary macro string,number,value ;CONVERT-TO-BINARY
                local       ten,start,calc,mult,no-mult
                jmp         start
ten             db          10
start:          mov         value,0
                xor         cx,cx
                mov         cl,number
                xor         si,si
calc:           xor         ax,ax
                mov         al,string[si]
                sub         al,48
                cmp         cx,2
                jl          no-mult
                push        cx
                dec         cx
mult:           mul         cs:ten
                loop        mult
                pop         cx
no-mult:        add         value,ax
                inc         si
                loop        calc
                endm
;
convert-date macro          dir-entry
                mov         dx,word ptr dir-entry[25]
                mov         cl,5
                shr         dl,cl
                mov         dh,dir-entry[25]
                and         dh,1fh
                xor         cx,cx
                mov         cl,dir-entry[26]
                shr         cl,1
                add         cx,1980
                endm
;
```

1.9 EXTENDED EXAMPLE OF MS-DOS SYSTEM CALLS

```
title DISK DUMP
zero                    equ     0
disk-B                  equu    1
sectors-per-read        equ     9
cr                      equ     13
blank                   equ     32
period                  equ     46
tilde                   equ     126
     INCLUDE B:CALLS.EQU
;
subttl DATA SEGMENT
page +
data                    segment
;
input-buffer            db      9 dup(512 dup(?))
output-buffer           db      77 dup(” ”)
                        db      0DH,0AH,”$“
start-prompt            db      ”Start at sector: $“
sectors-prompt          db      ”Number of sectors: $“
continue-prompt         db      ”RETURN to continue $“
header                  db      ”Relative sector $“
end-string              db      0DH,0AH,0AH,07H,”ALL DONE$“
                                ;DELETE THIS
crlf                    db      0DH,0AH,”$“
table                   db      “0123456789ABCDEF$”
;
ten                     db      10
sixteen                 db      16
;
start-sector            dw      1
sector-num              label   byte
sector-number           dw      0
sectors-to-dump         dw      sectors-per-read
sectors-read            dw      0
;
buffer                  label   byte
max-length              db      0
current-length          db      0
digits                  db      5 dup(?)
;
data                    ends
;
subttl STACK SEGMENT
page +
stack                   seg-
                        ment    stack
                        dw      100 dup(?)
stack-top               label   word
stack                   ends
;
subttl MACROS
page +
;
```

```
    INCLUDE B:CALLS.MAC
;BLANK LINE
blank-line              macro   number
                        local   print-it
                        push    cx
                        call    clear-line
                        mov     cx,number
print-it:               display output-buffer
                        loop    print-it
                        pop     cx
                        endm
;
subttl ADDRESSABILITY
page +
code                    segment
                        assume  cs:code,ds:data,ss:stack
start:                  mov     ax,data
                        mov     ds,ax
                        mov     ax,stack
                        mov     ss,ax
                        mov     sp,offset stack-top
;
                        jmp     main-procedure
subttl PROCEDURES
page +
;
; PROCEDURES
; READ-DISK
read-disk               proc;
                        cmp     sectors-to-dump-zero
                        jle     done
                        mov     bx,offset input-buffer
                        mov     dx,start-sector
                        mov     al,disk-b
                        mov     cx,sectors-per-read
                        cmp     cx,sectors-to-dump
                        jle     get-sector
                        mov     cx,sectors-to-dump
get-sector:             push    cx
                        int     disk-read
                        popf
                        pop     cx
                        sub     sectors-to-dump,cx
                        add     start-sector,cx
                        mov     sectors-read,cx
                        xor     si,si
done:                   ret
read-disk               endp
;CLEAR-LINE
clear-line              proc;
                        push    cx
                        mov     cx,77
                        xor     bx,bx
move-blank:             mov     output-buffer[bx],“   ”
                        inc     bx
```

```
                    loop     move-blank
                    pop      cx
                    ret
clear-line          endp
;
;PUT-BLANK
put-blank           proc;
                    mov      output-buffer[di], "   "
                    inc      di
                    ret
put-blank           endp
;
;
;
setup               proc;
                    display  start-prompt
                    get-string 4,buffer
                    display  crlf
                    convert-to-binary digits,
                    current-length,start-sector
                    mov      ax,start-sector
                    mov      sector-number,ax
                    display  sectors-prompt
                    get-string 4,buffer
                    convert-to-binary digits,
                    current-length,sectors-to-dump
                    ret
setup               endp
;
;CONVERT-LINE
convert-line        proc;
                    push     cx
                    mov      di,9
                    mov      cx,16
convert-it          convert input-buffer[si],sixteen,
                    output-buffer[di]
                    inc      si
                    add      di,2
                    call     put-blank
                    loop     convert-it
                    sub      si,16
                    mov      cx,16
                    add      di,4
display-ascii:      mov      output-buffer[di],period
                    cmp      input-buffer[si],blank
                    jl       non-printable
                    cmp      input-buffer[si],tilde
                    jg       non-printable
printable:          mov      dl,input-buffer[si]
                    mov      output-buffer[di],dl
non-printable:      inc      si
                    inc      di
                    loop     display-ascii
                    pop      cx
                    ret
convert-line        endp
```

```
;
;DISPLAY-SCREEN
display-screen          proc;
                        push    cx
                        call    clear-line
;
                        mov     cx,17
;I WANT length header
                        dec     cx
;minus 1 in cx
                        xor     di,di
move-header:            mov     al,header[di]
                        mov     output-buffer[di],al
                        inc     di
                        loop    move-header   ;FIX THIS!
;
                        convert sector-num[1],sixteen,
                        output-buffer[di]
                        add     di,2
                        convert sector-num,sixteen,
                        output-buffer[di]
                        display output-buffer
                        blank-line 2
                        mov     cx,16
dump-it:                call    clear-line
                        call    convert-line
                        display output-buffer
                        loop    dump-it
                        blank-line 3
                        display continue-prompt
                        get-char-no-echo
                        display crlf
                        pop     cx
                        ret
display-screen          endp
;
;
; END PROCEDURES
subttl MAIN PROCEDURE
page +
main-procedure:         call    setup
check-done:             cmp     sectors-to-dump,zero
                        jng     all-done
                        call    read-disk
                        mov     cx,sectors-read
display-it:             call    display-screen
                        call    display-screen
                        inc     sector-number
                        loop    display-it
                        jmp     check-done
all-done:               display end-string
                        get-char-no-echo
code                    ends
                        end     start
```

CHAPTER 2
MS-DOS 2.0 DEVICE DRIVERS

2.1 WHAT IS A DEVICE DRIVER?

A device driver is a binary file with all of the code in it to manipulate the hardware and provide a consistent interface to MS-DOS. In addition, it has a special header at the beginning that identifies it as a device, defines the strategy and interrupt entry points, and describes various attributes of the device.

NOTE

> For device drivers, the file must not use the ORG 100H (like .COM files). Because it does not use the Program Segment Prefix, the device driver is simply loaded; therefore, the file must have an origin of zero (ORG 0 or no ORG statement).

There are two kinds of device drivers.

1. Character device drivers
2. Block device drivers

Character devices are designed to perform serial character I/O like CON, AUX, and PRN. These devices are named (i.e., CON, AUX, CLOCK, etc.), and users may open channels (handles or FCBs) to do I/O to them.
Block devices are the "disk drives" on the system. They can perform random I/O in pieces called blocks (usually the physical sector size). These devices are not named as the character devices are, and therefore cannot be opened directly. Instead they are identified via the drive letters (A:,B:,C:, etc.).

Block devices also have units. A single driver may be responsible for one or more disk drives. For example, block device driver ALPHA may be responsible for drives A:,B:,C: and D:. This means that it has four units (0-3) defined and, therefore, takes up four drive letters. The position of the driver in the list of all drivers determines which units correspond to which driver letters. If driver ALPHA is the first block driver in the device list, and it defines 4 units (0-3), then they will be A:,B:,C: and D:. If Beta is the second block driver and defines three units (0-2), then they will be E:,F: and G:, and so on. MS-DOS 2.0 is not limited to 16 block device units, as previous versions were. The theoretical limit is 63 (26 - 1), but it should be noted that after 26 the drive letters are unconventional (such as], \, and ^).

NOTE

> Character devices cannot define multiple units because they have only one name.

2.2 DEVICE HEADERS

A device header is required at the beginning of a device driver. A device header looks like this:

DWORD pointer to next device (Must be set to -1)
WORD attributes Bit 15 = 1 if char device 0 is blk if bit 15 is 1 Bit 0 = 1 if current sti device Bit 1 = 1 if current sto output Bit 2 = 1 if current NUL device Bit 3 = 1 if current CLOCK dev Bit 4 = 1 if special Bits 5 - 12 Reserved; must be set to 0 Bit 14 is the IOCTL bit Bit 13 is the NON IBM FORMAT bit
WORD pointer to device strategy entry point
WORD pointer to device interrupt entry point
8-BYTE character device name field Character devices set a device name. For block devices the first byte is the number of units.

Figure 2. Sample Device Header

Note that the device entry points are words. They must be offsets from the same segment number used to point to this table. For example, if XXX:YYY points to the start of this table, then XXX:strategy and XXX:interrupt are the entry points.

2.2.1 Pointer To Next Device Field

The pointer to the next device header field is a double word field (offset followed by segment) that is set by MS-DOS to point at the next driver in the system list at the time the device driver is loaded. It is important that this field be set to -1 prior to load (when it is on the disk as a file) unless there is more than one device driver in the file. If there is more than one driver in the file, the first word of the double word pointer should be the offset of the next driver's Device Header.

NOTE

> If there is more than one device driver in the .COM file, the **last** driver in the file must have the pointer to the next Device Header field set to -1.

2.2.2 Attribute Field

The attribute field is used to tell the system whether this device is a block or character device (bit 15). Most other bits are used to give selected character devices certain special treatment. (Note that these bits mean nothing on a block device). For example, assume that a user has a new device driver that he wants to be the standard input and output. Besides installing the driver, he must tell MS-DOS that he wants his new driver to override the current standard input and standard output (the CON device). This is accomplished by setting the attributes to the desired characteristics, so he would set bits 0 and 1 to 1 (note that they are separate!) Similarly, a new CLOCK device could be installed by setting that attribute. (Refer to section 2.7, "The CLOCK Device", in this chapter for more information.) Although there is a NUL device attribute, the NUL device cannot be reassigned. This attribute exists so that MS-DOS can determine if the NUL device is being used.

The NON IBM FORMAT bit applies only to block devices and affects the operation of the BUILD BPB (Bios Parameter Block) device call. (Refer to section 2.5.3 for further information on this call).

The other bit of interest is the IOCTL bit, which has meaning on character and block devices. This bit tells MS-DOS whether the device can handle control strings (via the IOCTL system call, Function 44H).

If a driver cannot process control strings, it should initially set this bit to 0. This tells MS-DOS to return an error if an attempt is made (via Function 44H) to send or receive control strings to this device. A device which can process control strings should initialize the IOCTL bit to 1. For drivers of this type, MS-DOS will make calls to the IOCTL INPUT and OUTPUT device functions to send and receive IOCTL strings.

The IOCTL functions allow data to be sent and received by the device for its own use (for example, to set baud rate, stop bits, and form length), instead of passing data over the device channel as does a normal read or write. The interpretation of the passed information is up to the device, but it **must not** be treated as a normal I/O request.

2.2.3 Strategy And Interrupt Routines

These two fields are the pointers to the entry points of the strategy and interrupt routines. They are word values, so they must be in the same segment as the Device Header.

2.2.4 Name Field

This is an 8-byte field that contains the name of a character device or the number of units of a block device. If it is a block device, the number of units can be put in the first byte. This is optional, because MS-DOS will fill in this location with the value returned by the driver's INIT code. Refer to Section 2.4, "Installation of Device Drivers" in this chapter for more information.

2.3 HOW TO CREATE A DEVICE DRIVER

In order to create a device driver that MS-DOS can install, you must write a binary file with a Device Header at the beginning of the file. Note that for device drivers, the code should not be originated at 100H, but rather at 0. The link field (pointer to next Device Header) should be -1, unless there is more than one device driver in the file. The attribute field and entry points must be set correctly.

If it is a character device, the name field should be filled in with the name of that character device. The name can be any legal 8-character filename.

MS-DOS always processes installable device drivers before handling the default devices, so to install a new CON device, simply name the device CON. Remember to set the standard input device and standard output device bits in the attribute word on a new CON device. The scan of the device list stops on the first match, so the installable device driver takes precedence.

NOTE

> Because MS-DOS can install the driver anywhere in memory, care must be taken in any far memory references. You should not expect that your driver will always be loaded in the same place every time.

2.4 INSTALLATION OF DEVICE DRIVERS

MS-DOS 2.0 allows new device drivers to be installed dynamically at boot time. This is accomplished by INIT code in the BIOS, which reads and processes the CONFIG.SYS file.
MS-DOS calls upon the device drivers to perform their function in the following manner:

> MS-DOS makes a far call to strategy entry, and passes (in a Request Header) the information describing the functions of the device driver.

This structure allows you to program an interrupt-driven device driver. For example, you may want to perform local buffering in a printer.

2.5 REQUEST HEADER

When MS-DOS calls a device driver to perform a function, it passes a Request Header in ES:BX to the strategy entry point. This is a fixed length header, followed by data pertinent to the operation being performed. Note that it is the device driver's responsibility to preserve the machine state (for example, save all registers on entry and restore them on exit). There is enough room on the stack when strategy or interrupt is called to do about 20 pushes. If more stack is needed, the driver should set up its own stack.
The following figure illustrates a Request Header.

REQUEST HEADER – >

BYTE length of record Length in bytes of this Request Header
BYTE unit code The subunit the operation is for (minor device) (no meaning on character devices)
BYTE command code
WORD status
8 bytes RESERVED

Figure 3. Request Header

2.5.1 Unit Code

The unit code field identifies which unit in your device driver the request is for. For example, if your device driver has 3 units defined, then the possible values of the unit code field would be 0, 1, and 2.

2.5.2 Command Code Field

The command code field in the Request header can have the following values:

Command Code	Function
0	INIT
1	MEDIA CHECK (Block only, NOP for character)
2	BUILD BPB “ “ “ “ “
3	IOCTL INPUT (Only called if device has IOCTL)
4	INPUT (read)
5	NON-DESTRUCTIVE INPUT NO WAIT (Char devs only)
6	INPUT STATUS “ “ “
7	INPUT FLUSH “ “ “
8	OUTPUT (write)
9	OUTPUT (write) with verify
10	OUTPUT STATUS “ “ “
11	OUTPUT FLUSH “ “ “
12	IOCTL OUTPUT (Only called if device has IOCTL)

2.5.3 MEDIA CHECK AND BUILD BPB

MEDIA CHECK and BUILD BPB are used with block devices only. MS-DOS calls MEDIA CHECK first for a drive unit. MS-DOS passes its current media descriptor byte (refer to the section "Media Descriptor Byte" later in this chapter). MEDIA CHECK returns one of the following results:

> Media Not Changed - current DPB and media byte are OK.
> Media Changed - Current DPB and media are wrong. MS-DOS invalidates any buffers for this unit and calls the device driver to build the BPB with media byte and buffer.
> Not Sure - If there are dirty buffers (buffers with changed data, not yet written to disk) for this unit, MS-DOS assumes the DPB and media byte are OK (media not changed). If nothing is dirty, MS-DOS assumes the media has changed. It invalidates any buffers for the unit, and calls the device driver to build the BPB with media byte and buffer.
> Error - If an error occurs, MS-DOS sets the error code accordingly.

MS-DOS will call BUILD BPB under the following conditions:

> If Media Changed is returned
>
> If Not Sure is returned, and there are no dirty buffers

The BUILD BPB call also gets a pointer to a one-sector buffer. What this buffer contains is determined by the NON IBM FORMAT bit in the attribute field. If the bit is zero (device is IBM format-compatible), then the buffer contains the first sector of the first FAT. The FAT ID byte is the first byte of this buffer. NOTE: The BPB must be the same, as far as location of the FAT is concerned, for all possible media because this first FAT sector must be read **before** the actual BPB is returned. If the NON IBM FORMAT bit is set, then the pointer points to one sector of scratch space (which may be used for anything).

2.5.4 Status Word

The following figure illustrates the status word in the Request Header.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E R R	RESERVED					B U S	D O N	ERROR CODE (bit 15 on)							

The Status word is zero on entry and is set by the driver interrupt routine on return.
Bit 8 is the done bit. When set, it means the operation is complete. For MS-DOS 2.0, the driver sets it to 1 when it exits.
Bit 15 is the error bit. If it is set, then the low 8 bits indicate the error. The errors are:

0	Write protect violation
1	Unknown Unit
2	Drive not ready
3	Unknown command
4	CRC error
5	Bad drive request structure length
6	Seek error
7	Unknown media
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read Fault
C	General failure

Bit 9 is the busy bit, which is set only by status calls.

For output on character devices: If bit 9 is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request, and a write request (if made) would start immediately.

For input on character devices with a buffer: If bit 9 is 1 on return, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the device buffer and a read would return quickly. It also indicates that something has been typed. MS-DOS assumes all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer should always return busy=0 so that MS-DOS will not continuously wait for something to get into a buffer that does not exist.

One of the functions defined for each device is INIT. This routine is called only once when the device is installed. The INIT routine returns a location (DS:DX), which is a pointer to the first free byte of memory after the device driver (similar to "Keep Process"). This pointer method can be used to delete initialization code that is only needed once, saving on space.

Block devices are installed the same way and also return a first free byte pointer as described above. Additional information is also returned:

The number of units is returned. This determines logical device names. If the current maximum logical device letter is F at the time of the install call, and the INIT routine returns 4 as the number of units, then they will have logical names G, H, I and J. This mapping is determined by the position of the driver in the device list, and by the number of units on the device (stored in the first byte of the device name field).

A pointer to a BPB (BIOS Paramter Block) pointer array is also returned. There is one table for each unit defined. These blocks will be used to build an internal DOS data structure for each of the units. The pointer passed to the DOS from the driver points to an array of n word pointers to BPBs, where n is the number of units defined. In this way, if all units are the same, all of the pointers can point to the same BPB, saving space. Note that this array must be protected (below the free pointer set by the return) since an internal DOS structure will be built starting at the byte pointed to by the free pointer. The sector size defined must be less than or equal to the maximum sector size defined at default BIOS INIT time. If it isn't, the install will fail.

The last thing that INIT of a block device must pass back is the media descriptor byte. This byte means nothing to MS-DOS, but is passed to devices so that they know what parameters MS-DOS is currently using for a particular drive unit.

Block devices may take several approaches; they may be **dumb** or **smart.** A dumb device defines a unit (and therefore an internal DOS structure) for each possible media drive combination. For example, unit 0 = drive 0 single side, unit 1 = drive 0 double side. For this approach, media descriptor bytes do not mean anything. A smart device allows multiple media per unit. In this case, the BPB table returned at INIT must define space large enough to accommodate the largest possible media supported. Smart drivers will use the media descriptor byte to pass information about what media is currently in a unit.

2.6 FUNCTION CALL PARAMETERS

All strategy routines are called with ES:BX pointing to the Request Header. The interrupt routines get the pointers to the Request Header from the queue that the strategy routines store them in. The command code in the Request Header tells the driver which function to perform.

NOTE

All DWORD pointers are stored offset first,
then segment.

2.6.1 INIT

Command code = 0

INIT – ES:BX – >

13–BYTE Request Header
BYTE # of units
DWORD break address
DWORD pointer to BPB array (Not set by character devices)

The number of units, break address, and BPB pointer are set by the driver. On entry, the DWORD that is to be set to the BPB array (on block devices) points to the character after the "=" on the line in CONFIG.SYS that loaded this device. This allows drivers to scan the CONFIG.SYS invocation line for arguments.

NOTE

> If there are multiple device drivers in a single .COM file, the ending address returned by the last INIT called will be the one MS-DOS uses. It is recommended that all of the device drivers in a single .COM file return the same ending address.

2.6.2 MEDIA CHECK

Command Code = 1
MEDIA CHECK - ES:BX -

13–BYTE Request Header
BYTE media descriptor from DPB
BYTE returned

In addition to setting the status word, the driver must set the return byte to one of the following:

-1	Media has been changed
0	Don't know if media has been changed
1	Media has not been changed

If the driver can return -1 or 1 (by having a door-lock or other interlock mechanism) MS-DOS performance is enhanced because MS-DOS does not need to reread the FAT for each directory access.

2.6.3 BUILD BPB (BIOS Paramter Block)

Command code = 2
BUILD BPB – ES:BX – >

13–BYTE Request Header
BYTE media descriptor from DPB
DWORD transfer address (Points to one sector worth of scratch space or first sector of FAT depending on the value of the NON IBM FORMAT bit)
DWORD pointer to BPB

If the NON IBM FORMAT bit of the device is set, then the DWORD transfer address points to a one sector buffer, which can be used for any purpose. If the NON IBM FORMAT bit is 0, then this buffer contains the first sector of the first FAT and the driver must not alter this buffer.

If IBM compatible format is used (NON IBM FORMAT BIT = 0), then the first sector of the first FAT must be located at the same sector on all possible media. This is because the FAT sector will be read BEFORE the media is actually determined. Use this mode if all you want is to read the FAT ID byte.

In addition to setting status word, the driver must set the Pointer to the BPB on return.

In order to allow for many different OEMs to read each other's disks, the following standard is suggested: The information relating to the BPB for a particular piece of media is kept in the boot sector for the media. In particular, the format of the boot sector is:

	3 BYTE near JUMP to boot code
	8 BYTES OEM name and version
B P B I V I B P B	WORD bytes per sector
	BYTE sectors per allocation unit
	WORD reserved sectors
	BYTE number of FATs
	WORD number of root dir entries
	WORD number of sectors in logical image
	BYTE media descriptor
	WORD number of FAT sectors
	WORD sectors per track
	WORD number of heads
	WORD number of hidden sectors

The three words at the end (sectors per track, number of heads, and number of hidden sectors) are optional. They are intended to help the BIOS understand the media. Sectors per track may be redundant (could be calculated from total size of the disk). Number of heads is useful for supporting different multi-head drives which have the same storage capacity, but different numbers of surfaces. Number of hidden sectors may be used to support drive-partitioning schemes.

2.6.4 Media Descriptor Byte

The last two digits of the FAT ID byte are called the media descriptor byte. Currently, the media descriptor byte has been defined for a few media types, including 5-1/4“ and 8“ standard disks. For more information, refer to Section 3.6, “MS-DOS Standard Disk Formats.” Although these media bytes map directly to FAT ID bytes (which are constrained to the 8 values F8-FF), media bytes can, in general, be any value in the range 0-FF.

2.6.5 READ OR WRITE

Command codes = 3,4,8,9, and 12

READ or WRITE - ES:BX (Including IOCTL) - >

13-BYTE Request Header
BYTE media descriptor from DPB
DWORD transfer address
WORD byte/sector count
WORD starting sector number (Ignored on character devices)

In addition to setting the status word, the driver must set the sector count to the actual number of sectors (or bytes) transferred. No error check is performed on an IOCTL I/O call. The driver **must** correctly set the return sector (byte) count to the actual number of bytes transferred.

THE FOLLOWING APPLIES TO BLOCK DEVICE DRIVERS:

Under certain circumstances the BIOS may be asked to perform a write operation of 64K bytes, which seems to be a "wrap around" of the transfer address in the BIOS I/O packet. This request arises due to an optimization added to the write code in MS-DOS. It will only manifest on user writes that are within a sector size of 64K bytes on files "growing" past the current EOF. **It is allowable for the BIOS to ignore the balance of the write that "wraps around" if it so chooses.** For example, a write of 10000H bytes worth of sectors with a transfer address of XXX:1 could ignore the last two bytes. A user program can never request an I/O of more than FFFFH bytes and cannot wrap around (even to 0) in the transfer segment. Therefore, in this case, the last two bytes can be ignored.

2.6.6 NON DESTRUCTIVE READ NO WAIT

Command code = 5

NON DESRUCTIVE READ NO WAIT - ES:BX - >

13–BYTE Request Header
BYTE read from device

If the character device returns busy bit = 0 (characters in buffer), then the next character that would be read is returned. This character is **not** removed from the input buffer (hence the term "Non Destructive Read"). Basically, this call allows MS-DOS to look ahead one input character.

2.6.7 STATUS

Command codes = 6 and 10

STATUS Calls - ES:BX - >

13-BYTE Request Header

All the driver must do is set the status word and the busy bit as follows:

> **For output on character devices:** If bit 9 is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request and a write request (if made) would start immediately.
> **For input on character devices with a buffer:** A return of 1 means, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the devices buffer and a read would return quickly. A return of 0 also indicates that the user has typed something. MS-DOS assumes that all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer should always return busy = 0 so that the DOS will not hang waiting for something to get into a buffer which doesn't exist.

2.6.8 FLUSH

Command codes = 7 and 11

FLUSH Calls - ES:BX - >

13-Byte Request Header

The FLUSH call tells the driver to flush (terminate) all pending requests. This call is used to flush the input queue on character devices.

2.7 THE CLOCK DEVICE

One of the most popular add-on boards is the real time clock board. To allow this board to be integrated into the system for TIME and DATE, there is a special device (determined by the attribute word), called the CLOCK device. The CLOCK device defines and performs functions like any other character device. Most functions will be: "set done bit, reset error bit, return." When a read or write to this device occurs, exactly 6 bytes are transferred. The first two bytes are a word, which is the count of days since 1-1-80. The third byte is minutes, the fourth, hours, the fifth, hundredths of seconds, and the sixth, seconds. Reading the CLOCK device gets the date and time; writing to it sets the date and time.

2.8 EXAMPLE DEVICE DRIVERS

The following examples illustrate a block device driver and a character device driver program.

2.8.1 Block Device Driver

```
;*************************** A BLOCK DEVICE *****************************

     TITLE 5 1/4" DISK DRIVER FOR SCP DISK-MASTER

;This driver is intended to drive up to four 5 1/4" drives
;hooked to the Seattle Computer Products DISK MASTER disk
;controller. All standard IBM PC formats are supported.

FALSE          EQU            0
TRUE           EQU            NOT FALSE

;The I/O port address of the DISK MASTER
DISK           EQU            0E0H
;DISK+0
;              1793           Command/Status
;DISK+1
;              1793           Track
;DISK+2
;              1793           Sector
;DISK+3
;              1793           Data
;DISK+4
;              Aux Command/Status
;DISK+5
;              Wait Sync

;Back side select bit
BACKBIT        EQU            04H
;5 1/4"                       select bit
SMALBIT        EQU            10H
;Double Density bit
DDBIT          EQU            08H

;Done bit in status register
DONEBIT        EQU            01H

; Use table below to select head step speed.
;Step times for 5" drives
; are double that shown in the table.
;
; Step value                  1771     1793
;
;              0              6ms      3ms
;              1              6ms      6ms
;               2               10ms    10ms
;               3               20ms    15ms
;
STPSPD          EQU             1

NUMERR          EQU             ERROUT-ERRIN

CR              EQU             0DH
LF              EQU             0AH
CODE            SEGMENT
ASSUME          CS:CODE,DS:NOTHING,ES:NOTHING,SS:NOTHING
;---------------------------------------------------------------
;
;               DEVICE          HEADER
;
DRVDEV          LABEL           WORD
                DW              -1,-1
                DW              0000    ;IBM format-compatible, Block
                DW              STRATEGY
                DW              DRV$IN
DRVMAX          DB              4

DRVTBL          LABEL           WORD
                DW              DRV$INIT
                DW              MEDIA$CHK
                DW              GET$BPB
                DW              CMDERR
                DW              DRV$READ
                DW              EXIT
                DW              EXIT
                DW              EXIT
                DW              DRV$WRIT
                DW              DRV$WRIT
                DW              EXIT
                DW              EXIT
                DW              EXIT

;---------------------------------------------------------------
;
;               STRATEGY
PTRSAV          DD              0

STRATP          PROC            FAR
STRATEGY:
                MOV             WORD PTR [PTRSAV],BX
                MOV             WORD PTR [PTRSAV+2],ES
                RET
STRATP          ENDP

;---------------------------------------------------------------
;
;               MAIN ENTRY
```

```
CMDLEN    =  0       ;LENGTH OF THIS COMMAND
UNIT      =  1       ;SUB UNIT SPECIFIER
CMDC      =  2       ;COMMAND CODE
STATUS    =  3       ;STATUS
MEDIA     = 13       ;MEDIA DESCRIPTOR
TRANS     = 14       ;TRANSFER ADDRESS
COUNT     = 18       ;COUNT OF BLOCKS OR CHARACTERS
START     = 20       ;FIRST BLOCK TO TRANSFER
DRV$IN:
          PUSH  SI
          PUSH  AX
          PUSH  CX
          PUSH  DX
          PUSH  DI
          PUSH  BP
          PUSH  DS
          PUSH  ES
          PUSH  BX

          LDS   BX,[PTRSAV]   ;GET POINTER TO I/O PACKET

          MOV   AL,BYTE PTR [BX].UNIT       ;AL = UNIT CODE
          MOV   AH,BYTE PTR [BX].MEDIA      ;AH = MEDIA DESCRIP
          MOV   CX,WORD PTR [BX].COUNT      ;CX = COUNT
          MOV   DX,WORD PTR [BX].START;     ;DX = START SECTOR
          PUSH  AX
          MOV   AL,BYTE PTR [BX].CMDC       ;Command code
          CMP   AL,11
          JA    CMDERRP                     ;Bad command
          CBW
          SHL   AX,1                        ;2 times command =
                                            ;word table index
          MOV   SI,OFFSET DRVTBL
          ADD   SI,AX                       ;Index into table
          POP   AX                          ;Get back media
                                            ;and unit

          LES   DI,DWORD PTR[BX].TRANS      ;ES:DI = TRANSFER
                                            ;ADDRESS

          PUSH  CS
          POP   DS
ASSUME    DS:CODE
          JMP   WORD PTR [SI]               ;GO DO COMMAND
;-------------------------------------------------------------------------------
;         EXIT - ALL ROUTINES RETURN THROUGH THIS PATH
;
ASSUME    DS:NOTHING
CMDERRP:
```

```
           POP    AX                            ;Clean stack
CMDERR:
           MOV    AL,3                          ;UNKNOWN COMMAND ERROR
           JMP    SHORT ERR$EXIT

ERR$CNT:   LDS    BX,[PTRSAV]
           SUB    WORD PTR [BX].COUNT,CX ;# OF SUCCESS. I/Os

ERR$EXIT:
;AL has error code
           MOV    AH,10000001B                          ;MARK ERROR RETURN
           JMP    SHORT ERR1

EXITP      PROC   FAR

EXIT:      MOV    AH,00000001B
ERR1:      LDS    BX, [PTRSAV]
           MOV    WORD PTR [BX].STATUS,AX ;MARK OPERATION COMPLETE

           POP    BX
           POP    ES
           POP    DS
           POP    BP
           POP    DI
           POP    DX
           POP    CX
           POP    AX
           POP    SI
           RET                                  ;RESTORE REGS AND RETURN
EXITP      ENDP

CURDRV     DB     -1

TRKTAB     DB     -1,-1,-1,-1

SECCNT     DW     0

DRVLIM     =      8                             ;Number of sectors on device
SECLIM     =      13                            ;MAXIMUM SECTOR
HDLIM      =      15                            ;MAXIMUM HEAD

;WARNING - preserve order of drive and curhd!

DRIVE      DB     0                             ;PHYSICAL DRIVE CODE
CURHD      DB     0                             ;CURRENT HEAD
CURSEC     DB     0                             ;CURRENT SECTOR
CURTRK     DW     0                             ;CURRENT TRACK

;
MEDIA$CHK:                                      ;Always indicates Don't know
ASSUME     DS:CODE
           TEST   AH,00000100B ;TEST IF MEDIA REMOVABLE
           JZ     MEDIA$EXT
```

```
                XOR     DI,DI                           ;SAY I DON'T KNOW
MEDIA$EXT:
                LDS     BX, [PTRSAV]
                MOV     WORD PTR [BX].TRANS,DI
                JMP     EXIT

BUILD$BPB:
ASSUME          DS:CODE
                MOV     AH,BYTE PTR ES: [DI]            ;GET FAT ID BYTE
                CALL    GETBP                           ;TRANSLATE
SETBPB:         LDS     BX,[PTRSAV]
                MOV     [BX].MEDIA,AH
                MOV     [BX].COUNT,DI
                MOV     [BX].COUNT+2,CS
                JMP     EXIT

BUILDBP:
ASSUME          DS:NOTHING
;AH is media byte on entry
;DI points to correct BPB on return
                PUSH    AX
                PUSH    CX
                PUSH    DX
                PUSH    BX
                MOV     CL,AH           ;SAVE MEDIA
                AND     CL,0F8H         ;NORMALIZE
                CMP     CL,0F8H ;COMPARE WITH GOOD MEDIA BYTE
                JZ      GOODID
                MOV     AH,0FEH         ;DEFAULT TO 8-SECTOR,
                                        ;SINGLE-SIDED
GOODID:
                MOV     AL,1            ;SET NUMBER OF FAT SECTORS
                MOV     BX,64*256+8     ;SET DIR ENTRIES AND SECTOR MAX
                MOV     CX,40*8         ;SET SIZE OF DRIVE
                MOV     DX,01*256+1     ;SET HEAD LIMIT & SEC/ALL UNIT
                MOV     DI,OFFSET DRVBPB
                TEST    AH,00000010B    ;TEST FOR 8 OR 9 SECTOR
                JNZ     HAS8            ;NZ = HAS 8 SECTORS
                INC     AL              ;INC NUMBER OF FAT SECTORS
                INC     BL              ;INC SECTOR MAX
                ADD     CX,40           ;INCREASE SIZE
HAS8:           TEST    AH,00000001B    ;TEST FOR 1 OR 2 HEADS
                JZ      HAS1            ;Z = 1 HEAD
                ADD     CX,CX           ;DOUBLE SIZE OF DISK
                MOV     BH,112          ;INCREASE # OF DIREC. ENTRIES
                INC     DH              ;INC SEC/ALL UNIT
                INC     DL              ;INC HEAD LIMIT
HAS1:           MOV     BYTE PTR [DI].2,DH
                MOV     BYTE PTR [DI].6,BH
                MOV     WORD PTR [DI].8,CX
                MOV     BYTE PTR [DI].10,AH
                MOV     BYTE PTR [DI].11,AL
                MOV     BYTE PTR [DI].13,BL
                MOV     BYTE PTR [DI].15,DL
                POP     BX
                POP     DC
                POP     CX
                POP     AX
                RET
;------------------------------------------------------------------------------
;
;               DISK I/O HANDLERS
;
;ENTRY:
;               AL = DRIVE NUMBER (0-3)
;               AH = MEDIA DESCRIPTOR
;               CX = SECTOR COUNT
;               DX = FIRST SECTOR
;               DS = CS
;               ES:DI = TRANSFER ADDRESS
;EXIT:
;               IF SUCCESSFUL CARRY FLAG = 0
;                 ELSE CF = 1 AND AL CONTAINS (MS-DOS) ERROR CODE,
                  CX # sectors NOT transferred
DRV$READ:
ASSUME          DS:CODE
                JCXZ            DSKOK
                CALL            SETUP
                JC              DSK$IO
                CALL            DISKRD
                JMP             SHORT DSK$IO

DRV$WRIT:
ASSUME          DS:CODE
                JCXZ            DSKOK
                CALL            SETUP
                JC              DSK$IO
                CALL            DISKWRT
ASSUME          DS:NOTHING
DSK$IO:         JNC             DSKOK
                JMP             ERR$CNT
DSKOK:          JMP             EXIT

SETUP:
ASSUME          DS:CODE
;Input same as above
;On output
; ES:DI = Trans addr
; DS:BX Points to BPB
; Carry set if error (AL is error code (MS-DOS))
; else
;               [DRIVE]         = Drive number (0-3)
;               [SECCNT]        = Sectors to transfer
;               [CURSEC]        = Sector number of start of I/O
;               [CURHD]         = Head number of start of I/O   ;SET
;               [CURTRK]        = Track # of start of I/O ;Seek performed
; All other registers destroyed
                XCHG    BX,DI                           ;ES:BX = TRANSFER ADDRESS
                CALL    GETBP                           ;DS:DI = PTR TO BPB
                MOV     SI,CX
                ADD     SI,DX
                CMP     SI,WORD PTR [DI].DRVLIM
                                                        ;COMPARE AGAINST DRIVE MAX
                JBE     INRANGE
                MOV     AL,8
                STC
                RET

INRANGE:
                MOV     [DRIVE],AL
                MOV     [SECCNT],CX                     ;SAVE SECTOR COUNT
                XCHG    AX,DX                           ;SET UP LOGICAL SECTOR
                                                        ;FOR DIVIDE
                XOR     DX,DX
                DIV     WORD PTR [DI].SECLIM ;DIVIDE BY SEC PER TRACK
                INC     DL
                MOV     [CURSEC],DL                     ;SAVE CURRENT SECTOR
                MOV     CX,WORD PTR [DI].HDLIM ;GET NUMBER OF HEADS
                XOR     DX,DX ;DIVIDE TRACKS BY HEADS PER CYLINDER
                DIV     CX
                MOV     [CURHD],DL                      ;SAVE CURRENT HEAD
                MOV     [CURTRK],AX                     ;SAVE CURRENT TRACK
SEEK:
                PUSH    BX                              ;Xaddr
                PUSH    DI                              ;BPB pointer
                CALL    CHKNEW                          ;Unload nead if change drives
                CALL    DRIVESEL
                MOV     BL,[DRIVE]
                XOR     BH,BH,                          ;BX drive index
                ADD     BX,OFFSET TRKTAB                ;Get current track
                MOV     AX, [CURTRK]
                MOV     DL,AL                           ;Save desired track
                XCHG    AL,DS:[BX]                      ;Make desired track current
                OUT     DISK+1,AL                       ;Tell Controller current track
                CMP     AL,DL                           ;At correct track?
                JZ      SEEKRET                         ;Done if yes
                MOV     BH,2                            ;Seek retry count
                CMP     AL,-1                           ;Position Known?
                JNZ     NOHOME                          ;If not home head
TRYSK:
                CALL    HOME
                JC      SEEKERR
NOHOME:
                MOV     AL,DL
                OUT     DISK+3,AL                       ;Desired track
                MOV     AL,1CH+STPSPD                   ;Seek
                CALL    DCOM
                AND     AL,98H                          ;Accept not rdy, seek, & CRC errors
                JZ      SEEKRET
                JS      SEEKERR                         ;No retries if not ready
```

```
            DEC     BH
            JNZ     TRYSK
SEEKERR:
            MOV     BL,[DRIVE]
            XOR     BH,BH                       ;BX drive index
            ADD     BX,OFFSET TRKTAB            ;Get current track
            MOV     BYTE PTR DS:[BX],-1         ;Make current track
                                                ;lunknown
            CALL    GETERRCD
            MOV     CX,[SECCNT]                 ;Nothing transferred
            POP     BX                          ;BPB Pointer
            POP     DI                          ;Xaddr
            RET

SEEKRET:
            POP     BX                          ;BPB pointer
            POP     DI                          ;Xaddr
            CLC
            RET

;-------------------------------------------------------------------------------
;
;           READ
;

DISKRD:
ASSUME      DS:CODE
            MOV     CX,[SECCNT]
RDLP:
            CALL    PRESET
            PUSH    BX
            MOV     BL,10                       ;Retry count
            MOV     DX,DISK+3                   ;Data port
RDAGN:
            MOV     AL,80H                      ;Read command
            CLI                                 ;Disable for 1793
            OUT     DISK,AL                     ;Output read command
            MOV     BP,DI                       ;Save address for retry
            JMP     SHORT RLOOPENTRY
RLOOP:
            STOSB
RLOOPENTRY:
            IN      AL,DISK+5                   ;Wait for DRQ or INTRQ
            SHR     AL,1
            IN      AL,DX                       ;Read data
            JNC     RLOOP
            STI                                 ;Ints OK now
            CALL    GETSTAT
            AND     AL,9CH
            JZ      RDPOP                       ;Ok
            MOV     DI,BP                       ;Get back transfer
            DEC     BL
            JNZ     RDAGN
            CMP     AL,10H                      ;Record not found?
```

```
                JNZ     GOT-CODE                        ;No
                MOV     AL,1                            ;Map it
GOT-CODE:
                CALL    GETERRCD
                POP     BX
                RET
RDPOP:
                POP     BX
                LOOP    RDLP
                CLC
                RET

;-------------------------------------------------------------------------------
;
;               WRITE
;

DISKWRT:
ASSUME          DS:CODE
                MOV     CX,[SECCNT]
                MOV     SI,DI
                PUSH    ES
                POP     DS
ASSUME          DS:NOTHING
WRLP:
                CALL    PRESET
                PUSH    BX
                MOV     BL,10                           ;Retry count
                MOV     DX,DISK+3                       ;Data port
WRAGN:
                MOV     AL,0A0H                         ;Write command
                CLI                                     ;Disable for 1793
                OUT     DISK,AL                         ;Output write command
                MOV     BP,SI                           ;Save address for retry
WRLOOP:
                IN      AL,DISK+5
                SHR     AL,1
                LODSB                                   ;Get data
                OUT     DX,AL                           ;Write data
                JNC     WRLOOP
                STI                                     ;Ints OK now
                DEC     SI
                CALL    GETSTAT
                AND     AL,0FCH
                JZ      WRPOP                           ;Ok
                MOV     SI,BP                           ;Get back transfer
                DEC     BL
                JNZ     WRAGN
                CALL    GETERRCD
                POP     BX
                RET

WRPOP:
                POP     BX
```

```
              LOOP   WRLP
              CLC
              RET

PRESET:
ASSUME        DS:NOTHING
              MOV    AL, [CURSEC]
              CMP    AL,CS:[BX].SECLIM
              JBE    GOTSEC
              MOV    DH,[CURHD]
              INC    DH
              CMP    DH,CS:[BX].HDLIM
              JB     SETHEAD                      ;Select new head
              CALL   STEP                         ;Go on to next track
              XOR    DH,DH                        ;Select head zero
SETHEAD:
              MOV    [CURHD],DH
              CALL   DRIVESEL
              MOV    AL,1                         ;First sector
              MOV    [CURSEC],AL                  ;Reset CURSEC
GOTSEC:
              OUT    DISK+2,AL                    ;Tell controller which sector
              INC    [CURSEC]                     ;We go on to next sector
              RET

STEP:
ASSUME        DS:NOTHING
              MOV    AL,58H+STPSPD                ;Step in w/ update, no verify
              CALL   DCOM
              PUSH   BX
              MOV    BL,[DRIVE]
              XOR    BH,BH                        ;BX drive index
              ADD    BX,OFFSET TRKTAB             ;Get current track
              INC    BYTE PTR CS:[BX]             ;Next track
              POP    BX
              RET

HOME:
ASSUME        DS:NOTHING
              MOV    BL,3
TRYHOM:
              MOV    AL,0CH+STPSPD                ;Restore with verify
              CALL   DCOM
              AND    AL,98H
              JZ     RET3
              JS     HOMERR                       ;No retries if not ready
              PUSH   AX                           ;Save real error code
              MOV    AL,58H+STPSPD                ;Step in w/ update no verify
              CALL   DCOM
              DEC    BL
              POP    AX                           ;Get back real error code
              JNZ    TRYHOM
HOMERR:
              STC
```

```
RET3:       RET

CHKNEW:
ASSUME      DS:NOTHING
            MOV    AL,[DRIVE]                  ;Get disk drive number
            MOV    AH,AL
            XCHG   AL,[CURDRV]                 ;Make new drive current.
            CMP    AL,AH                       ;Changing drives?
            JZ     RET1                        ;No
; If changing drives, unload head so the head load delay
;one-shot will fire again. Do it by seeking to the same
;track with the H bit reset.
;
            IN     AL,DISK+1                   ;Get current track number
            OUT    DISK+3,AL                   ;Make it the track to seek
            MOV    AL,10H                      ;Seek and unload head

DCOM:
ASSUME      DS:NOTHING
            OUT    DISK,AL
            PUSH   AX
            AAM                                ;Delay 10 microseconds
            POP    AX
GETSTAT:
            IN     AL,DISK+4
            TEST   AL,DONEBIT
            JZ     GETSTAT
            IN     AL,DISK
RET1:       RET

DRIVESEL:
ASSUME      DS:NOTHING
;Select the drive based on current info
;Only AL altered
            MOV    AL,[DRIVE]
            OR     AL,SMALBIT + DDBIT    ;5 1/4“ IBM PC disks
            CMP    [CURHD],0
            JZ     GOTHEAD
            OR     AL,BACKBIT                  ;Select side 1
GOTHEAD:
            OUT    DISK+4,AL                   ;Select drive and side
            RET

GETERRCD:
ASSUME      DS:NOTHING
            PUSH   CX
            PUSH   ES
            PUSH   DI
            PUSH   CS
            POP    ES                          ;Make ES the local segment
            MOV    CS:[LSTERR],AL              ;Terminate list w/ error code
            MOV    CX,NUMERR                   ;Number of error conditions
            MOV    DI,OFFSET ERRIN             ;Point to error conditions
            REPNE  SCASB
            MOV     AL,NUMERR-1[DI]                 ;Get translation
            STC                                     ;Flag error condition
            POP     DI
            POP     ES
            POP     CX
            RET                                     ;and return

;***********************************************************************************************
;           BPB FOR AN IBM FLOPPY DISK, VARIOUS PARAMETERS ARE
;           PATCHED BY GETBP TO REFLECT THE TYPE OF MEDIA
;           INSERTED
;           This is a nine sector single side BPB
DRVBPB:
            DW      512                             ;Physical sector size in bytes
            DB      1                               ;Sectors/allocation unit
            DW      1                               ;Reserved sectors for DOS
            DB      2                               ;# of allocation tables
            DW      64                              ;Number directory entries
            DW      9*40                            ;Number 512-byte sectors
            DB      11111100B                       ;Media descriptor
            DW      2                               ;Number of FAT sectors
            DW      9                               ;Sector limit
            DW      1                               ;Head limit

INITAB      DW      DRVBPB                          ;Up to four units
            DW      DRVBPB
            DW      DRVBPB
            DW      DRVBPB

ERRIN:      ;DISK ERRORS RETURNED FROM THE 1793 CONTROLER
            DB      80H                             ;NO RESPONSE
            DB      40H                             ;Write protect
            DB      20H                             ;Write Fault
            DB      10H                             ;SEEK error
            DB      8                               ;CRC error
            DB      1                               ;Mapped from 10H
                                                    ;(record not found) on READ
LSTERR      DB      0                               ;ALL OTHER ERRORS

ERROUT:     ;RETURNED ERROR CODES CORRESPONDING TO ABOVE
            DB      2                               ;NO RESPONSE
            DB      0                               ;WRITE ATTEMPT
                                                    ;ON WRITE-PROTECT DISK
            DB      0AH                             ;WRITE FAULT
            DB      6                               ;SEEK FAILURE
            DB      4                               ;BAD CRC
            DB      8                               ;SECTOR NOT FOUND
            DB      12                              ;GENERAL ERROR

DRV$INIT:
;
; Determine number of physical drives by reading CONFIG.SYS
;
```

```
ASSUME     DS:CODE
           PUSH   DS
           LDS    SI,[PTRSAV]
ASSUME     DS:NOTHING
           LDS    SI,DWORD PTR [SI.COUNT]       ;DS:SI points to ;CONFIG.SYS
SCAN-LOOP:
           CALL   SCAN-SWITCH
           MOV    AL,CL
           OR     AL,AL
           JZ     SCAN4
           CMP    AL,"s"
           JZ     SCAN4

WERROR:    POP    DS
ASSUME     DS:CODE
           MOV    DX,OFFSET ERRMSG2
WERROR2:   MOV    AH,9
           INT    21H
           XOR    AX,AX
           PUSH   AX                            ;No units
           JMP    SHORT ABORT

BADNDRV:
           POP    DS
           MOV    DX,OFFSET ERRMSG1
           JMP    WERROR2

SCAN4:
ASSUME     DS:NOTHING
;BX is number of floppies
           OR     BX,BX
           JZ     BADNDRV                       ;User error
           CMP    BX,4
           JA     BADNDRV                       ;User error
           POP    DS
ASSUME     DS:CODE
           PUSH   BX                            ;Save unit count
ABORT:     LDS    BX,[PTRSAV]
ASSUME     DS:NOTHING
           POP    AX
           MOV    BYTE PTR [BX].MEDIA,AL        ;Unit count
           MOV    [DRVMAX],AL
           MOV    WORD PTR [BX].TRANS,OFFSET DRV$INIT ;SET
                                                ;BREAK ADDRESS
           MOV    [BX].TRANS+2,CS
           MOV    WORD PTR [BX].COUNT,OFFSET INITAB
                                                ;SET POINTER TO BPB ARRAY
           MOV    [BX].COUNT+2,CS
           JMP    EXIT
;
; PUT SWITCH IN CL, VALUE IN BX
;
SCAN-SWITCH:
           XOR    BX,BX
           MOV    CX,BX
           LODSB
           CMP    AL,10
           JZ     NUMRET
           CMP    AL,“-”
           JZ     GOT-SWITCH
           CMP    AL,“/”
           JNZ    SCAN-SWITCH
GOT-SWITCH:
           CMP    BYTE PTR [SI+1],“:”
           JNZ    TERROR
           LODSB
           OR     AL,20H                    ; CONVERT TO LOWER CASE
           MOV    CL,AL                     ; GET SWITCH
           LODSB                            ; SKIP “:”
;
; GET NUMBER POINTED TO BY [SI]
;
; WIPES OUT AX,DX ONLY                      BX RETURNS NUMBER
;
GETNUM1:   LODSB
           SUB    AL,“0”
           JB     CHKRET
           CMP    AL,9
           JA     CHKRET
           CBW
           XCHG   AX,BX
           MOV    DX,10
           MUL    DX
           ADD    BX,AX
           JMP    GETNUM1

CHKRET:    ADD    AL,“0”
           CMP    AL,“ ”
           JBE    NUMRET
           CMP    AL,“-”
           JZ     NUMRET
           CMP    AL,“/”
           JZ     NUMRET
TERROR:
           POP    DS                        ; GET RID OF RETURN ADDRESS
           JMP    WERROR
NUMRET:    DEC    SI
           RET

ERRMSG1    DB     “SMLDRV: Bad number of drives”,13,10,“$”
ERRMSG2    DB     “SMLDRV: Invalid parameter”,13,10,“$”
CODE       ENDS
           END
```

2.8.2 Character Device Driver

The following program illustrates a character device driver program.

```
;****************** A CHARACTER DEVICE ******************

TITLE     VT52 CONSOLE FOR 2.0     (IBM)

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;
;         IBM ADDRESSES FOR I/O
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

          CR=13                ;CARRIAGE RETURN
          BACKSP=8             ;BACKSPACE
          ESC=1BH
          BRKADR=6CH           ;006C BREAK VECTOR ADDRESS
          ASNMAX=200           ;SIZE OF KEY ASSIGNMENT BUFFER

CODE      SEGMENT BYTE

     ASSUME CS:CODE,DS:NOTHING,ES:NOTHING
;-----------------------------------------------------------------------------------
;
;         C O N - CONSOLE DEVICE DRIVER
;
CONDEV:                                   ;HEADER FOR DEVICE "CON"
          DW    -1,-1
          DW    1000000000010011B         ;CON IN AND CON OUT
          DW    STRATEGY
          DW    ENTRY
          DB    'CON     '

;-----------------------------------------------------------------------------------
;
;         COMMAND JUMP TABLES
CONTBL:
          DW    CON$INIT
          DW    EXIT
          DW    EXIT
          DW    CMDERR
          DW    CON$READ
          DW    CON$RDND
          DW    EXIT
          DW    CON$FLSH
          DW    CON$WRIT
          DW    CON$WRIT
          DW    EXIT
          DW    EXIT

CMDTABL   DB    'A'
```

```
          DW    CUU                         ;cursor up
          DB    “B”
          DW    CUD                         ;cursor down
          DB    “C”
          DW    CUF                         ;cursor forward
          DB    “D”
          DW    CUB                         ;cursor back
          DB    “H”
          DW    CUH                         ;cursor position
          DB    “J”
          DW    ED                          ;erase display
          DB    “K”
          DW    EL                          ;erase line
          DB    “Y”
          DW    CUP                         ;cursor position
          DB    “j”
          DW    PSCP                        ;save cursor position
          DB    “k”
          DW    PRCP                        ;restore cursor position
          DB    “y”
          DW    RM                          ;reset mode
          DB    “x”
          DW    SM                          ;set mode
          DB    00

PAGE
;-------------------------------------------------------------
;
;         Device entry pont
;
CMDLEN    =     0                       ;LENGTH OF THIS COMMAND
UNIT      =     1                       ;SUB UNIT SPECIFIER
CMD       =     2                       ;COMMAND CODE
STATUS    =     3                       ;STATUS
MEDIA     =     13                      ;MEDIA DESCRIPTOR
TRANS     =     14                      ;TRANSFER ADDRESS
COUNT     =     18                      ;COUNT OF BLOCKS OR CHARACTERS
START     =     20                      ;FIRST BLOCK TO TRANSFER

PTRSAV    DD    0
STRATP    PROC  FAR

STRATEGY:
          MOV   WORD PTR CS:[PTRSAV],BX
          MOV   WORD PTR CS:[PTRSAV+2],ES
          RET

STRATP    ENDP

ENTRY:
          PUSH  SI
          PUSH  AX
          PUSH  CX
          PUSH  DX
```

```
                PUSH    DI
                PUSH    BP
                PUSH    DS
                PUSH    ES
                PUSH    BX

                LDS     BX,CS:[PTRSAV] ;GET POINTER TO I/O PACKET
                MOV     CX,WORD PTR DS:[BX].COUNT   ;CX = COUNT

                MOV     AL,BYTE PTR DS:[BX].CMD
                CBW
                MOV     SI,OFFSET CONTBL
                ADD     SI,AX
                ADD     SI,AX
                CMP     AL,11
                JA      CMDERR

                LES     DI,DWORD PTR DS:[BX].TRANS

                PUSH    CS
                POP     DS
                ASSUME    DS:CODE

                JMP     WORD PTR [SI]                                   ;GO DO COMMAND

PAGE
;===========================================================================
;=
;=              SUBROUTINES SHARED BY MULTIPLE DEVICES
;=
;===========================================================================
;---------------------------------------------------------------------------
;
;               EXIT - ALL ROUTINES RETURN THROUGH THIS PATH
;
BUS$EXIT:                                               ;DEVICE BUSY EXIT
                MOV     AH,00000011B
                JMP     SHORT ERR1

CMDERR:
                MOV     AL,3                            ;UNKNOWN COMMAND ERROR

ERR$EXIT:
                MOV     AH,10000001B                    ;MARK ERROR RETURN
                JMP     SHORT ERR1

EXITP           PROC    FAR

EXIT:           MOV     AH,00000001B
ERR1:           LDS     BX,CS:[PTRSAV]
                MOV     WORD PTR [BX].STATUS,AX         ;MARK
                                                        ;OPERATION COMPLETE
                POP     BX
                POP     ES
                POP     DS
                POP     BP
                POP     DI
                POP     DX
                POP     CX
                POP     AX
                POP     SI
                RET                                     ;RESTORE REGS AND RETURN
EXITP           ENDP
;-------------------------------------------------------------------------------
;
;               BREAK KEY HANDLING
;
BREAK:
                MOV     CS:ALTAH,3                      ;INDICATE BREAK KEY SET
INTRET:         IRET

PAGE
;
;               WARNING - Variables are very order dependent,
;                         so be careful when adding new ones!
;
WRAP            DB      0                               ; 0 = WRAP, 1 = NO WRAP
STATE           DW      S1
MODE            DB      3
MAXCOL          DB      79
COL             DB      0
ROW             DB      0
SAVCR           DW      0
ALTAH           DB      0                               ;Special key handling

;-------------------------------------------------------------------------------
;
;               CHROUT - WRITE OUT CHAR IN AL USING CURRENT ATTRIBUTE
;
ATTRW           LABEL   WORD
ATTR            DB      00000111B                       ;CHARACTER ATTRIBUTE
BPAGE           DB      0                               ;BASE PAGE
base            dw      0b800h

chrout:         cmp     al,13
                jnz     trylf
                mov     [col],0
                jmp     short setit

trylf:          cmp     al,10
                jz      lf
                cmp     al,7
                jnz     tryback
torom:
                mov     bx,[attrw]
                and     bl,7
                mov     ah,14
```

```
            int     10h
ret5:       ret
tryback:
            cmp     al,8
            jnz     outchr
            cmp     [col],0
            jz      ret5
            dec     [col]
            jmp     short setit

outchr:
            mov     bx,[attrw]
            mov     cx,1
            mov     ah,9
            int     10h
            inc     [col]
            mov     al,[col]
            cmp     al,[maxcol]
            jbe     setit
            cmp     [wrap],0
            jz      outchr1
            dec     [col]
            ret
outchr1:
            mov     [col],0
lf:         inc     [row]
            cmp     [row],24
            jb      setit
            mov     [row],23
            call    scroll

setit:      mov     dh,row
            mov     dl,col
            xor     bh,bh
            mov     ah,2
            int     10h
            ret

scroll:     call    getmod
            cmp     al,2
            jz      myscroll
            cmp     al,3
            jz      myscroll
            mov     al,10
            jmp     torom
myscroll:
            mov     bh,[attr]
            mov     bl,“ ”
            mov     bp,80
            mov     ax,[base]
            mov     es,ax
            mov     ds,ax
            xor     di,di
            mov     si,160
```

```
                mov     cx,23*80
                cld
                cmp     ax,0b800h
                jz      colorcard

                rep     movsw
                mov     ax,bx
                mov     cx,bp
                rep     stosw
sret:           push    cs
                pop     ds
                ret

colorcard:
                mov     dx,3dah
wait2:          in      al,dx
                test    al,8
                jz      wait2
                mov     al,25h
                mov     dx,3d8h
                out     dx,al                           ;turn off video
                rep     movsw
                mov     ax,bx
                mov     cx,bp
                rep     stosw
                mov     al,29h
                mov     dx,3d8h
                out     dx,al                           ;turn on video
                jmp     sret

GETMOD:         MOV     AH,15
                INT     16                              ;get column information
                MOV     BPAGE, BH
                DEC     AH
                MOV     WORD PTR MODE,AX
                RET
;-------------------------------------------------------------------------------
;
;               CONSOLE READ ROUTINE
;
CON$READ:
                JCXZ    CON$EXIT
CON$LOOP:
                PUSH    CX                              ;SAVE COUNT
                CALL    CHRIN                           ;GET CHAR IN AL
                POP     CX
                STOSB                                   ;STORE CHAR AT ES:DI
                LOOP    CON$LOOP
CON$EXIT:
                JMP     EXIT
;-------------------------------------------------------------------------------
;
;               INPUT SINGLE CHAR INTO AL
;
CHRIN:          XOR     AX,AX
```

```
            XCHG  AL,ALTAH                  ;GET CHARACTER & ZERO ALTAH
            OR    AL,AL
            JNZ   KEYRET

INAGN:      XOR   AH,AH
            INT   22
ALT10:
            OR    AX,AX                     ;Check for non-key after BREAK
            JZ    INAGN
            OR    AL,AL                     ;SPECIAL CASE?
            JNZ   KEYRET
            MOV   ALTAH,AH                    ;STORE SPECIAL KEY

KEYRET:     RET
;-----------------------------------------------------------------------------
;
;           KEYBOARD NON DESTRUCTIVE READ, NO WAIT
;
CON$RDND:
            MOV   AL,[ALTAH]
            OR    AL,AL
            JNZ   RDEXIT

RD1:        MOV   AH,1
            INT   22
            JZ    CONBUS
            OR    AX,AX
            JNZ   RDEXIT
            MOV   AH,0
            INT   22
            JMP   CON$RDND

RDEXIT:     LDS   BX,[PTRSAV]
            MOV   [BX].MEDIA,AL
EXVEC:      JMP   EXIT
CONBUS:     JMP   BUS$EXIT
;-----------------------------------------------------------------------------
;
;           KEYBOARD FLUSH ROUTINE
;
CON$FLSH:
            MOV   [ALTAH],0                 ;Clear out holding buffer

            PUSH  DS
            XOR   BP,BP
            MOV   DS,BP                     ;Select segment 0
            MOV   DS:BYTE PTR 41AH,1EH      ;Reset KB queue head
                                            ;pointer
            MOV   DS:BYTE PTR 41CH,1EH      ;Reset tail pointer
            POP   DS
            JMP   EXVEC
;-----------------------------------------------------------------------------
;
;           CONSOLE WRITE ROUTINE
;
CON$WRIT:
        JCXZ    EXVEC
        PUSH    CX
        MOV     AH,3                    ;SET CURRENT CURSOR POSITION
        XOR     BX,BX
        INT     16
        MOV     WORD PTR [COL],DX
        POP     CX

CON$LP: MOV     AL,ES:[DI]              ;GET CHAR
        INC     DI
        CALL    OUTC                    ;OUTPUT CHAR
        LOOP    CON$LP                  ;REPEAT UNTIL ALL THROUGH
        JMP     EXVEC

COUT:   STI
        PUSH    DS
        PUSH    CS
        POP     DS
        CALL    OUTC
        POP     DS
        IRET

OUTC:   PUSH    AX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    ES
        PUSH    BP
        CALL    VIDEO
        POP     BP
        POP     ES
        POP     DI
        POP     SI
        POP     DX
        POP     CX
        POP     AX
        RET

;-----------------------------------------------------------------------------
;
;       OUTPUT SINGLE CHAR IN AL TO VIDEO DEVICE
;
VIDEO:  MOV     SI,OFFSET STATE
        JMP     [SI]

S1:     CMP     AL,ESC                  ;ESCAPE SEQUENCE?
        JNZ     S1B
        MOV     WORD PTR [SI],OFFSET S2
        RET

S1B:    CALL    CHROUT
S1A:    MOV     WORD PTR [STATE],OFFSET S1
        RET
```

```
S2:       PUSH    AX
          CALL    GETMOD
          POP     AX
          MOV     BX,OFFSET CMDTABL-3
S7A:      ADD     BX,3
          CMP     BYTE PTR [BX],0
          JZ      S1A
          CMP     BYTE PTR [BX],AL
          JNZ     S7A
          JMP     WORD PTR [BX+1]

MOVCUR:   CMP     BYTE PTR [BX],AH
          JZ      SETCUR
          ADD     BYTE PTR [BX],AL
SETCUR:   MOV     DX,WORD PTR COL
          XOR     BX,BX
          MOV     AH,2
          INT     16
          JMP     S1A

CUP:      MOV     WORD PTR [SI],OFFSET CUP1
          RET
CUP1:     SUB     AL,32
          MOV     BYTE PTR [ROW],AL
          MOV     WORD PTR [SI],OFFSET CUP2
          RET
CUP2:     SUB     AL,32
          MOV     BYTE PTR [COL],AL
          JMP     SETCUR

SM:       MOV     WORD PTR [SI],OFFSET S1A
          RET

CUH:      MOV     WORD PTR COL,0
          JMP     SETCUR

CUF:      MOV     AH,MAXCOL
          MOV     AL,1
CUF1:     MOV     BX,OFFSET COL
          JMP     MOVCUR

CUB:      MOV     AX,00FFH
          JMP     CUF1

CUU:      MOV     AX,00FFH
CUU1:     MOV     BX,OFFSET ROW
          JMP     MOVCUR

CUD:      MOV     AX,23*256+1
          JMP     CUU1
```

```
PSCP:       MOV   AX,WORD PTR COL
            MOV   SAVCR,AX
            JMP   SETCUR

PRCP:       MOV   AX,SAVCR
            MOV   WORD PTR COL,AX
            JMP   SETCUR

ED:         CMP   BYTE PTR [ROW],24
            JAE   EL1

            MOV   CX,WORD PTR COL
            MOV   DH,24
            JMP   ERASE

EL1:        MOV   BYTE PTR [COL],0
EL:         MOV   CX,WORD PTR [COL]
EL2         MOV   DH,CH
ERASE:      MOV   DL,MAXCOL
            MOV   BH,ATTR
            MOV   AX,0600H
            INT   16
ED3:        JMP   SETCUR

RM:         MOV   WORD PTR [SI],OFFSET RM1
            RET
RM1:        XOR   CX,CX
            MOV   CH,24
            JMP   EL2

CON$INIT:
            int   11h
            and   al,00110000b
            cmp   al,00110000b
            jnz   iscolor
            mov   [base],0b000h               ;look for bw card
iscolor:
            cmp   al,00010000b                ;look for 40 col mode
            ja    setbrk
            mov   [mode],0
            mov   [maxcol],39

setbrk:
            XOR   BX,BX
            MOV   DS,BX
            MOV   BX,BRKADR
            MOV   WORD PTR [BX],OFFSET BREAK
            MOV   WORD PTR [BX+2],CS

            MOV   BX,29H*4
            MOV   WORD PTR [BX],OFFSET COUT
            MOV   WORD PTR [BX+2],CS
```

```
                LDS     BX,CS:[PTRSAV]
                MOV     WORD PTR [BX].TRANS,OFFSET CON$INIT
                                                        ;SET BREAK ADDRESS
                MOV     [BX].TRANS+2,CS
                JMP     EXIT

CODE            ENDS
                END
```

CHAPTER 3
MS-DOS TECHNICAL INFORMATION

3.1 MS-DOS INITIALIZATION

MS-DOS initialization consists of several steps. Typically, a ROM (Read Only Memory) bootstrap obtains control, and then reads the boot sector off the disk. The boot sector then reads the following files:

IO.SYS
MSDOS.SYS

Once these files are read, the boot process begins.

3.2 THE COMMAND PROCESSOR

The Command processor supplied with MS-DOS (file COMMAND.-COM.) consists of 3 parts:

1. **A resident part** resides in memory immediately following MSDOS.SYS and its data area. This part contains routines to process Interrupts 23H (CONTROL-C Exit Address), and 24H (Fatal Error Abort Address), as well as a routine to reload the transient part, if needed. All standard MS-DOS error handling is done within this part of COMMAND.-COM. This includes displaying error messages and processing the Abort, Retry, or Ignore messages.
2. **An initialization part** follows the resident part. During start-up, the initialization part is given control; in contains the AUTOEXEC file processor setup routine. The initialization part determines the segment address at which programs can be loaded. It is overlaid by the first program COMMAND.-COM loads because it is no longer needed.

3. **A transient part** is loaded at the high end of memory. This part contains all of the internal command processors and the batch file processor.
   The transient part of the command processor produces the system prompt (such as A >), reads the command from keyboard (or batch file) and causes it to be executed. For external commands, this part builds a command line and issues the EXEC system call (Function Request 4BH) to load and transfer control to the program.

3.3 MS-DOS DISK ALLOCATION

The MS-DOS area is formatted as follows:

Reserved area - variable size
First copy of file allocation table - variable size
Second copy of file allocation table - variable size (optional)
Additional copies of file allocation table - variable size (optional)
Root directory - variable size
File data area

Allocation of space for a file in the data area is not pre-allocated. The space is allocated one cluster at a time. A cluster consists of one or more consecutive sectors; all of the clusters for a file are "chained" together in the File Allocation Table (FAT). (Refer to Section 3.5, "File Allocation Table.") There is usually a second copy of the FAT kept, for consistency. Should the disk develop a bad sector in the middle of the first FAT, the second can be used. This avoids loss of data due to an unusable disk.

3.4 MS-DOS DISK DIRECTORY

FORMAT builds the root directory for all disks. Its location on disk and the maximum number of entries are dependent on the media. Since directories other than the root directory are regarded as files by MS-DOS, there is no limit to the number of files they may contain.
All directory entries are 32 bytes in length, and are in the following format (note that byte offsets are in hexadecimal):

0-7 Filename. Eight characters, left aligned and padded, if necessary, with blanks. The first byte of this field indicates the file status as follows:

- 00H The directory entry has never been used. This is used to limit the length of directory searches, for performance reasons.
- 2EH The entry is for a directory. If the second byte is also 2EH, then the cluster field contains the cluster number of this cirectory's parent directory (0000H if the parent directory is the root directory). Otherwise, bytes 01H through 0AH are all spaces, and the cluster field contains the cluster number of this directory.
- E5H The file was used, but it has been erased.

Any other character is the first character of a filename.

8-0A Filename extension.

0B File attribute. The attribute byte is mapped as follows (values are in hexadecimal):

- 01 File is marked read-only. An attempt to open the file for writing using the Open File system call (Function Request 3DH) results in an error code being returned. This value can be used along with other values below. Attempts to delete the file with the Delete File system call (13H) or Delete a Directory Entry (41H) will also fail.
- 02 Hidden file. The file is excluded from normal directory searches.
- 04 System file. The file is excluded from normal directory searches.
- 08 The entry contains the volume label in the first 11 bytes. The entry contains no other usable information (except date and time of creation), and may exist only in the root directory.

10 The entry defines a sub-directory, and is excluded from normal directory searches.

20 Archive bit. The bit is set to "on" whenever the file has been written to and closed.
Note: The system files (IO.SYS and MSDOS.SYS) are marked as read-only, hidden, and system files. Files can be marked hidden when they are created. Also, the read-only, hidden, system, and archive attributes may be changed through the Change Attributes system call (Function Request 43H).

0C-15 Reserved.

16-17 Time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

```
Offset 17H
| H | H | H | H | H | M | M | M |
  7               3   2

Offset 16H
| M | M | M | S | S | S | S | S |
          5   4               0
```

where:

H is the binary number of hours (0-23)
M is the binary number of minutes (0-59)
S is the binary number of two-second increments

18-19 Date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

```
Offset 19H
| Y | Y | Y | Y | Y | Y | Y | M |
  7                       1   0

Offset 18 H
| M | M | M | D | D | D | D | D |
          5   4               0
```

where:

Y	is 0-119 (1980-2099)
M	is 1-12
D	is 1-31

1A-1B Starting cluster; the cluster number of the first cluster in the file.

Note that the first cluster for data space on all disks is cluster 002.

The cluster number is stored with the least significant byte first.

NOTE

> Refer to Section 3.5.1, "How to Use the File Allocation Table," for details about converting cluster numbers to logical sector numbers.

1C-1F File size in bytes. The first word of this four-byte field is the low-order part of the size.

3.5 FILE ALLOCATION TABLE (FAT)

The following information is included for system programmers who wish to write installable device drivers. This section explains how MS-DOS uses the File Allocation Table to convert the clusters of a file to logical sector numbers. The driver is then responsible for locating the logical sector on disk. Programs must use the MS-DOS file management function calls for accessing files; programs that access the FAT are not guaranteed to be upwardly-compatible with future releases of MS-DOS.

The File Allocation Table is an array of 12-bit entries (1.5 bytes) for each cluster on the disk. The first two FAT entries map a portion of the directory; these FAT entries indicate the size and format of the disk.

The second and third bytes currently always contain FFH.

The third FAT entry, which starts at byte offset 4, begins the mapping of the data area (cluster 002). Files in the data area are not always written sequentially on the disk. The data area is allocated one cluster at a time, skipping over clusters already allocated. The first free cluster found will be the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by erasing files can be allocated for new files.

Each FAT entry contains three hexadecimal characters:

000 If the cluster is unused and available.

FF7 The cluster has a bad sector in it. MS-DOS will not allocate such a cluster. CHKDSK counts the number of bad clusters for its report. These bad clusters are not part of any allocation chain.

FF8-FFF Indicates the last cluster of a file.

XXX Any other characters that are the cluster number of the next cluster in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.

The File Allocation Table always begins on the first section after the reserved sectors. If the FAT is larger than one sector, the sectors are continguous. Two copies of the FAT are usually written for data integrity. The FAT is read into one of the MS-DOS buffers whenever needed (open, read, write, etc.). For performance reasons, this buffer is given a high priority to keep it in memory as long as possible.

3.5.1 How To Use The File Allocation Table

Use the directory entry to find the starting cluster of the file. Next, to locate each subsequent cluster of the file:

1. Multiply the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).
2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.
3. Use a MOV instruction to move the word at the calculated FAT offset into a register.
4. If the last cluster used was an even number, keep the low-order 12 bits of the register by ANDing it with FFF; otherwise, keep the high-order 12 bits by shifting the register right 4 bits with a SHR instruction.
5. If the resultant 12 bits are FF8H-FFFH, the file contains no more clusters. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, such as that used by Interrupts 25H and 26H and by DEBUG):

1. Subtract 2 from the cluster number.
2. Multiply the result by the number of sectors per cluster.
3. Add to this result the logical sector number of the beginning of the data area.

3.6 MS-DOS STANDARD DISK FORMATS

On an MS-DOS disk, the clusters are arranged on disk to minimize head movement for multi-sided media. All of the space on a track (or cylinder) is allocated before moving on to the next track. This is accomplished by using the sequential sectors on the lowest-numbered head, then all the sectors on the next head, and so on until all sectors on all heads of the track are used. The next sector to be used will be sector 1 on head 0 of the next track.
For disks, the following table can be used:

# Sides	Sectors/ Track	FAT size Sectors	Dir Sectors	Dir Entries	Sectors/ Cluster
1	8	1	4	64	1
2	8	1	7	112	2
1	9	2	4	64	1
2	9	2	7	112	2

Figure 4. 5-¼" Disk Format

The first byte of the FAT can sometimes be used to determine the format of the disk. The following 5-¼" formats have been defined for the IBM Personal Computer, based on values of the first byte of the FAT. The formats in Table 3.1 are considered to be the standard disk formats for MS-DOS.

Table 3.1 MS-DOS Standard Disk Formats

	5-1/4	5-1/4	5-1/4	5-1/4	8	8	8
No. sides	1	1	2	2	1	1	2
Tracks/side	40	40	40	40	77	77	77
Bytes/sector	512	512	512	512	128	128	1024
Sectors/track	8	9	8	9	26	26	8
Sectors/allocation unit	1	1	2	2	4	4	1
Reserved sectors	1	1	1	1	1	4	1
No. FATS	2	2	2	2	2	2	2
Root directory entries	64	64	112	112	68	68	192
No. sectors	320	360	640	720	2002	2002	616
Media Descriptor Byte	FE	FC	FF	FD	FE*	FD	FE*
Sectors for 1 FAT	1	2	1	2	6	6	2

* The two media descriptor bytes that are the same for 8" disks (FEH) is not a misprint. To establish whether a disk is single- or double-density, a read of a single-density address mark should be made. If an error occurs, the media is double-density.

CHAPTER 4
MS-DOS CONTROL BLOCKS AND WORK AREAS

4.1 TYPICAL MS-DOS MEMORY MAP

0000:0000	Interrupt vector table
XXXX:0000	IO.SYS - MS-DOS interface to hardware
XXXX:0000	MSDOS.SYS - MS-DOS interrupt handlers, service routines (Interrupt 21H functions)
	MS-DOS buffers, control areas, and installed device drivers
XXXX:0000	Resident part of COMMAND.COM - Interrupt handlers for Interrupts 22H (Terminate Address), 23H (CONTROL-C Exit Address), 24H (Fatal Error Abort Address) and code to reload the transient part
XXXX:0000	External command or utility - (.COM or .EXE file)
XXXX:0000	User stack for .COM files (256 bytes)
XXXX:0000	Transient part of COMMAND.COM - Command interpreter, internal commands, batch processor

1. Memory map addresses are in segment:offset format. For example, 0090:0000 is absolute address 0900H.
2. User memory is allocated from the lowest end of available memory that will meet the allocation request.

4.2 MS-DOS PROGRAM SEGMENT

When an external command is typed, or when you execute a program through the EXEC system call, MS-DOS determines the lowest available free memory address to use as the start of the program. This area is called the Program Segment.

The first 256 bytes of the Program Segment are set up by the EXEC system call for the program being loaded into memory. The program is then loaded following this block. An .EXE file with minalloc and maxalloc both set to zero is loaded as high as possible.

At offset 0 within the Program Segment, MS-DOS builds the Program Segment Prefix control block. The program returns form EXEC by one of four methods:

1. A long jump to offset 0 in the Program Segment Prefix
2. By issuing an INT 20H with CS:0 pointing at the PSP
3. By issuing an INT 21H with register AH = 0 with CS:0 pointing at the PSP, or 4CH and no restrictions on CS
4. By a long call to location 50H in the Program Segment Prefix with AH = 0 or Function Request 4CH

NOTE

> It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when terminating via any of these methods, except Function Request 4CH. For this reason, using Function Request 4CH is the preferred method.

All four methods result in transferring control to the program that issued the EXEC. During this returning process, Interrupts 22H, 23H, and 24H (Terminate Address, CONTROL-C Exit Address, and Fatal Error Abort Address) addresses are restored from the values saved in the Program Segment Prefix of the terminating program. Control is then given to the terminate address. If this is a program returning to COMMAND.COM, control transfers to its resident portion. If a batch file was in process, it is continued; otherwise, COMMAND.COM performs a checksum on the transient part, reloads it if necessary, then issues the system prompt and waits for you to type the next command.

When a program receives control, the following conditions are in effect:

For all programs:

The segment address of the passed environment is contained at offset 2CH in the Program Segment Prefix.

The environment is a series of ASCII strings (totaling less than 32K) in the form:

NAME = parameter

Each string is terminated by a byte of zeros, and the set of strings is terminated by another byte of zeros. The environment built by the command processor contains at least a COMSPEC = string (the parameters on COMSPEC define the path used by MS-DOS to locate COMMAND.COM on disk). The last PATH and PROMPT commands issued will also be in the environment, along with any environment strings defined with the MS-DOS SET command.

The environment that is passed is a copy of the invoking process environment. If your application uses a "keep process" concept, you should be aware that the copy of the environment passed to you is static. That is, it will not change even if subsequent SET, PATH, or PROMPT commands are issued.

Offset 50H in the Program Segment Prefix contains code to call the MS-DOS function dispatcher. By placing the desired function request number in AH a program can issue a far call to offset 50H to invoke an MS-DOS function, rather than issuing an Interrupt 21H. Since this is a **call** and not an interrupt, MS-DOS may place any code appropriate to making a system call at this position. This makes the process of calling the system portable.

The Disk Transfer Address (DTA) is set to 80H (default DTA in the Program Segment Prefix).

File control blocks at 5CH and 6CH are formatted from the first two parameters typed when the command was entered. If either parameter contained a pathname, then the corresponding FCB contains only the valid drive number. The filename field will not be valid.

An unformatted parameter area at 81H contains all the characters typed after the command (including leading and imbedded delimiters), with the byte at 80H set to the number of characters. If the <, >, or parameters were typed on the command line, they (and the filenames associated with them) will not appear in this area; redirection of standard input and output is transparent to applications.

Offset 6 (one word) contains the number of bytes available in the segment.

Register AX indicates whether or not the drive specifiers (entered with the first two parameters) are valid, as follows:

> Al = FF if the first parameter contained an invalid drive specifier (otherwise AL = 00)
> AH = FF if the second parameter contained an invalid drive specifier (otherwise AH = 00)

Offset 2 (one word) contains the segment address of the first byte of unavailable memory. Programs must not modify addresses beyond this point unless they were obtained by allocating memory via the Allocate Memory system call (Function Request 48H).

For Executable (EXE) programs:

DS and ES registers are set to point to the Program Segment Prefix.

CS,IP,SS, and SP registers are set to the values passed by MS-LINK.

For Executable (.COM) programs:

All four segment registers contain the segment address of the initial allocation block that starts with the Program Segment Prefix control block.

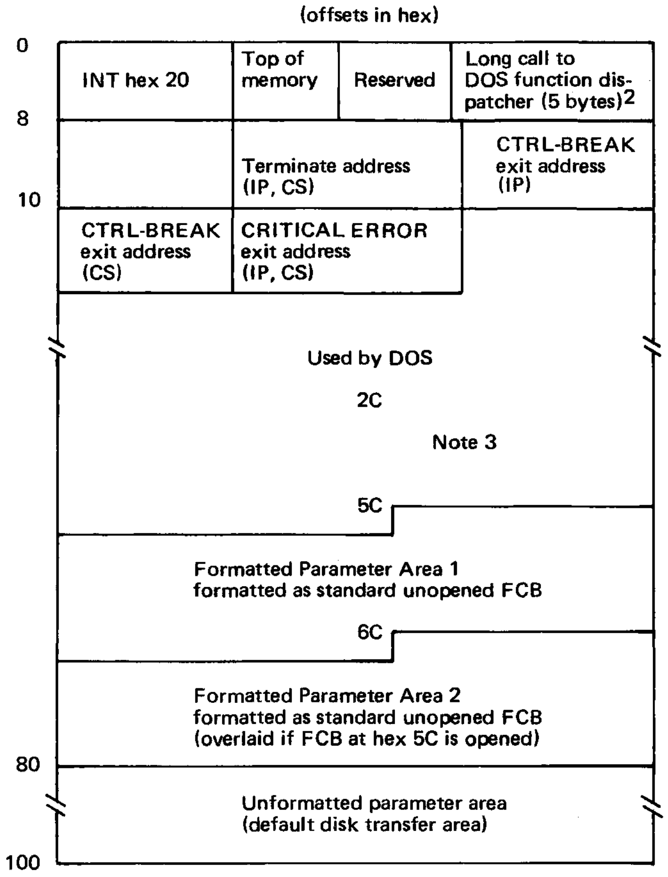
All of user memory is allocated to the program. If the program invokes another program through Function Request 4BH, it must first free some memory through the Set Block (4AH) function call, to provide space for the program being executed.

The Instruction Pointer (IP) is set to 100H.

The Stack Pointer register is set to the end of the program's segment. The segment size at offset 6 is reduced by 100H to allow for a stack of that size.

A word of zeros is placed on top of the stack. This is to allow a user program to exit to COMMAND.COM by doing a RET instruction last. This assumes, however, that the user has maintained his stack and code segments.

Figure 5. illustrates the format of the Program Segment Prefix. All offsets are in hexadecimal.


(offsets in hex)
0
8
10
80
100
INT hex 20
Top of memory
Reserved
Long call to DOS function dispatcher (5 bytes)2
Terminate address (IP, CS)
CTRL-BREAK exit address (IP)
CTRL-BREAK exit address (CS)
CRITICAL ERROR exit address (IP, CS)
Used by DOS
2C
Note 3
5C
Formatted Parameter Area 1 formatted as standard unopened FCB
6C
Formatted Parameter Area 2 formatted as standard unopened FCB (overlaid if FCB at hex 5C is opened)
Unformatted parameter area (default disk transfer area)

1. First segment of available memory is in segment (paragraph) form (for example, hex 1000 would represent 64K).
2. The word at offset 6 contains the number of bytes available in the segment.
3. Offset hex 2C contains the segment address of the environment.

Figure 5 Program Segment Prefix

IMPORTANT

Programs must not alter any part of the Program Segment Prefix below offset 5CH.

CHAPTER 5
EXE FILE STRUCTURE AND LOADING

NOTE

This chapter describes .EXE file structure and loading procedures for systems that use a version of MS-DOS that is lower than 2.0. For MS-DOS 2.0 and higher, use Function Request 4BH, Load and Execute a Program, to load (or load and execute) an .EXE file.

The .EXE files produced by MS-LINK consist of two parts:
Control and relocation information
The load module

The control and relocation information is at the beginning of the file in an area called the header. The load module immediately follows the header.
The header is formatted as follows. (Note that offsets are in hexadecimal.)

Offset	Contents
00-01	Must contain 4DH, 5AH.
02-03	Number of bytes contained in last page; this is useful in reading overlays.
04-05	Size of the file in 512-byte pages, including the header.
06-07	Number of relocation entries in table.

08-09 Size of the header in 16-byte paragraphs. This is used to locate the beginning of the load module in the file.

0A-0B Minimum number of 16-byte paragraphs required above the end of the loaded program.

0C-0D Maximum number of 16-byte paragraphs required above the end of the loaded program. If both minalloc and maxalloc are 0, then the program will be loaded as high as possible.

0E-0F Initial value to be loaded into stack segment before starting program execution. This must be adjusted by relocation.

10-11 Value to be loaded into the SP register before starting program execution.

12-13 Negative sum of all the words in the file.

14-15 Initial value to be loaded into the IP register before starting program execution.

16-17 Initial value to be loaded into the CS register before starting program execution. This must be adjusted by relocation.

18-19 Relative byte offset from beginning of run file to relocation table.

1A-1B The number of the overlay as generated by MS-LINK.

The relocation table follows the formatted area described above. This table consists of a variable number of relocation items. Each relocation item contains two fields: a two-byte offset value, followed by a two-byte segment value. These two fields contain the offset into the load module of a word which requires modification before the module is given control. The following steps describe this process:

1. The formatted part of the header is read into memory. Its size is 1BH.
2. A portion of memory is allocated depending on the size of the load module and the allocation numbers (0A-0B and 0C-0D). MS-DOS attempts to allocate FFFFH paragraphs. This will always fail, returning the size of the largest free block. If this block is smaller than minalloc and loadsize, then there will be no memory error. If this block is larger than maxalloc and loadsize, MS-DOS will allocate (maxalloc + loadsize). Otherwise, MS-DOS will allocate the largest free block of memory.
3. A Program Segment Prefix is built in the lowest part of the allocated memory.
4. The load module size is calculated by subtracting the header size from the file size. Offsets 04-05 and 08-09 can be used for this calculation. The actual size is downward-adjusted

based on the contents of offsets 02-03. Based on the setting of the high/low loader switch, an appropriate segment is determined at which to load the load module. This segment is called the start segment.

5. The load module is read into memory beginning with the start segment.
6. The relocation table items are read into a work area.
7. Each relocation table item segment value is added to the start segment value. This calculated segment, plus the relocation item offset value, points to a word in the load module to which is added the start segment value. The result is placed back into the word in the load module.
8. Once all relocation items have been processed, the SS and SP registers are set from the values in the header. Then, the start segment value is added to SS. The ES and DS registers are set to the segment address of the Program Segment Prefix. The start segment value is added to the header CS register value. The result, along with the header IP value, is the initial CS:IP to transfer to before starting execution of the program.

INDEX

MS-LIB Library Manager

MS-LIB
CONTENTS

INTRODUCTION

Features and Benefits

MS-LIB creates and modifies library files that are used with Microsoft's MS-LINK Linker Utility. MS-LIB can add object files to a library, delete modules from a library, or extract modules from a library and place the extracted modules into separate object files.

MS-LIB provides a means of creating either general or special libraries for a variety of programs or for specific programs only. With MS-LIB you can create a library for a language compiler, or you can create a library for one program only, which would permit very fast linking and possibly more efficient execution.

You can modify individual modules within a library by extracting the modules, making changes, then adding the modules to the library again. You can also replace an existing module with a different module or with a new version of an existing module.

The command scanner in MS-LIB is the same as the one used in Microsoft's MS-LINK, MS-Pascal, MS-FORTRAN, and other 16-bit Microsoft products. If you have used any of these products, using MS-LIB is familiar to you. Command syntax is straightforward, and MS-LIB prompts you for any of the commands it needs that you have not supplied. There are no surprises in the user interface.

Overview of MS-LIB Operation

MS-LIB performs two basic actions: it deletes modules from a library file, and it changes object files into modules and appends them to a library file. These two actions underlie five library manager functions:

> delete a module
> extract a module and place it in a separate object file
> append an object file as a module of a library
> replace a module in the library file with a new module
> create a library file

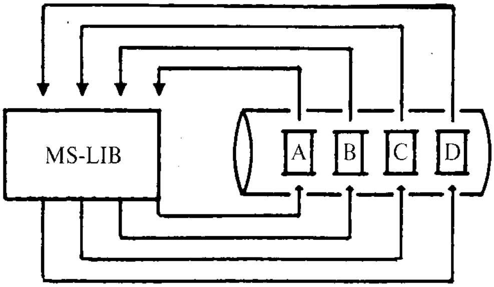
During each library session, MS-LIB first deletes or extracts modules, then appends new ones. In a single operation, MS-LIB reads each module into memory, checks it for consistency, and writes it back to the file. If you delete a module, MS-LIB reads in that module but does not write it back to the file. When MS-LIB writes back the next module to be retained, it places the module at the end of the last module written. This procedure effectively "closes up" the disk space to keep the library file from growing larger than necessary. When MS-LIB has read through the whole library file, it appends any new modules to the end of the file. Finally, MS-LIB creates the index, which MS-LINK uses to find modules and symbols in the library file, and outputs a cross reference listing of the PUBLIC symbols in the library, if you request such a listing. (Building the library index may take some extra time, up to 20 seconds in some cases.)

For example:

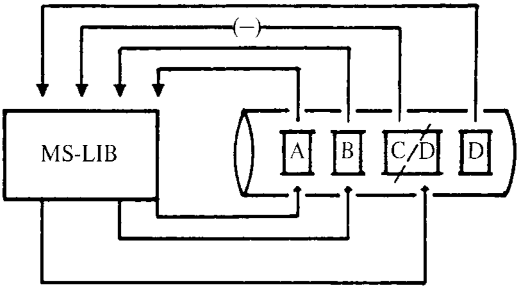
```
LIB PASCAL+HEAP-HEAP;
```

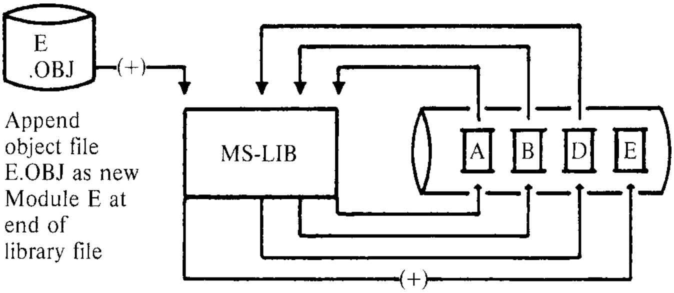
first deletes the library module HEAP from the library file, then adds the file HEAP.OBJ as the last module in the library. This order of execution prevents confusion in MS-LIB when a new version of a module replaces a version in the library file. Note that the replace function is simply the delete-append functions in succession. Also note that you can specify delete, append, or extract functions in any order; the order is insignificant to the MS-LIB command scanner.

Consistency
Check only



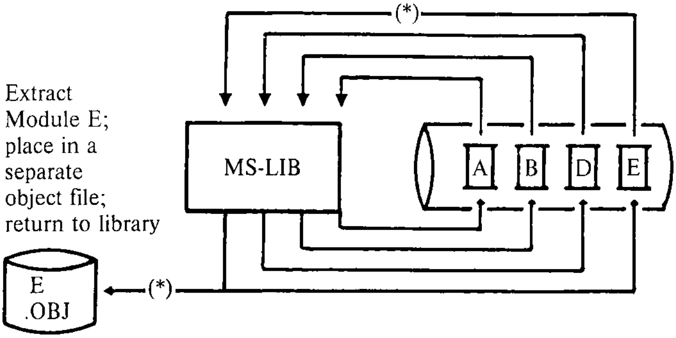
Delete
Module C;
Module D
written to
space of
Module C



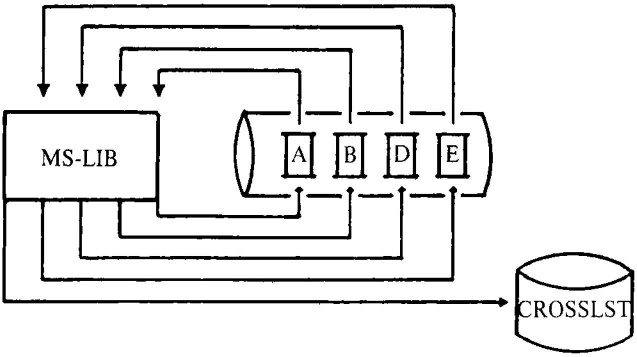


Append
object file
E.OBJ as new
Module E at
end of
library file

Extract
Module E;
place in a
separate
object file;
return to library



Consistency
Check, then
output a
cross
reference
listing of
PUBLIC
symbols



CHAPTER 1
RUNNING MS-LIB

Running MS-LIB requires two types of commands: a command to invoke MS-LIB and answers to command prompts. Usually you will enter all the commands to MS-LIB on the terminal keyboard. As an option, answers to the command prompts may be contained in a Response File. Some special command characters exist. Some are used as a required part of MS-LIB commands. Others assist you while entering MS-LIB commands.

1.1 INVOKING MS-LIB

MS-LIB may be invoked three ways. By the first method, you enter the commands as answers to individual prompts. By the second method, you enter all commands on the line used to invoke MS-LIB. By the third method, you create a Response File that contains all the necessary commands.

Summary of Methods to invoke MS-LIB

Method 1	LIB
Method 2	LIB <library> <operations>,<listing>
Method 3	LIB @ <filespec>

1.1.1 Method 1: LIB

Enter:

```
LIB
```

MS-LIB will be loaded into memory. Then, MS-LIB returns a series of three text prompts that appear one at a time. You answer the prompts as commands to MS-LIB to perform specific tasks.
The Command Prompts and Command Characters are summarized here. The Command Prompts and Command Characters are described fully in Sections 1.2 and 1.3.

Summary of Command Prompts

PROMPT	RESPONSES
Library file:	List filename of library to be manipulated (default: filename extension .LIB)
Operation:	List command character(s) followed by module name(s) or object filename(s) (default action: no changes. default object filename extension: .OBJ)
List file:	List filename for a cross reference listing file (default: NUL; no file)

Summary of Command Characters

Character	Action
+	Append an object file as the last module
-	Delete a module from the library
*	Extract a module and place in an object file
;	Use default responses to remaining prompts
&	Extend current physical line; repeat command prompt
Control-C	Abort library session.

1.1.2 Method 2: LIB <library> <operations>,<listing>

Enter:

```
LIB <library> <operations>,<listing>
```

The entries following LIB are responses to the command prompts. The **library** and **operations** fields and all operations entries must be separated by one of the command characters plus, minus, and asterisk (+, -, *). If a cross reference listing is wanted, the name of the file must be separated from the last operations entry by a comma.

where: **library** is the name of a library file. MS-LIB assumes that the filename extension is .OBJ, which you may override by specifying a different extension. If the filename given for the **library** fields does not exist, MS-LIB will prompt you:

```
Library file does not exist. Create?
```

Enter Yes (or any response beginning with Y) to create a new library file. Enter No (or any other response not beginning with Y) to abort the library session.

operations is deleting a module, appending an object file as a module, or extracting a module as an object file from the library file. Use the three command characters plus (+), minus (-), and asterisk (*) to direct MS-LIB what to do with each module or object file.

listing is the name of the file you want to receive the cross reference listing of PUBLIC symbols in the modules in the library. The list is compiled after all module manipulation has taken place.

To select the default for remaining field(s), you may enter the semicolon command character.

If you enter a Library filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you enter a Library filename followed immediately by a comma and a List filename, MS-LIB will perform its consistency check of the library file, then produce the cross reference listing file.

Example
LIB PASCAL-HEAP+HEAP;

This example causes MS-LIB to delete the module HEAP from the library file PASCAL.LIB, then append the object file HEAP.OBJ as the last module of PASCAL.LIB (the module will be named HEAP).
If you have many operations to perform during a library session, use the ampersand (&) command character to extend the line so that you can enter additional object filenames and module names. Be sure to always include one of the command characters for operations (+, -, *) before the name of each module or object filename.

Example

```
LIB PASCAL<CR>
```

causes MS-LIB to perform a consistency check of the library file PASCAL.LIB. No other action is performed.

Example

```
LIB PASCAL,PASCROSS.PUB
```

causes MS-LIB to perform a consitency check of the library file PASCAL.LIB, then output a cross reference listing file named PASCROSS.PUB.

1.1.3 Method 3: LIB @ <filespec>

Enter:

```
LIB @ <filespec>
```

where: **filespec** is the name of a Response File. A Response File contains answers to the MS-LIB prompts (summarized under method 1 for invoking and described fully in Section 1.2). Method 3 permits you to conduct the MS-LIB session without interactive (direct) user responses to the MS-LIB prompts.

IMPORTANT

> Before using method 3 to invoke MS-LIB, you must first create the Response File.

A Response File has text lines, one for each prompt. Responses must appear in the same order as the command prompts appear.

Use Command Characters in the Response File the same way as they are used for responses entered on the terminal keyboard.

When the library session begins, each prompt will be displayed in turn with the responses from the response file. If the response file does not contain answers for all the prompts, MS-LIB will use the default responses (no changes to the modules currently in the library file for Operation, and no cross reference listing file created).

If you enter a Library filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you enter a Library filename then only a carriage return of Operations then a comma and a List filename, MS-LIB will perform its consistency check of the library file, then produce the cross reference listing file.

Example:

```
PASCAL<CR>
+CURSOR+HEAP-HEAP*FOIBLES<CR>
CROSSLST<CR>
```

This Response File will cause MS-LIB to delete the module HEAP from the PASCAL.LIB library file, extract the module FOIBLES and place in an object file named FOIBLES.OBJ, then append the object files CURSOR.OBJ and HEAP.OBJ as the last two modules in the library. Then, MS-LIB will create a cross reference file named CROSSLST.

1.2 COMMAND PROMPTS

MS-LIB is commanded by entering responses to three text prompts. When you have entered your response to the current prompt, the next appears. When the last prompt has been answered, MS-LIB performs its library management functions without further command. When the library session is finished, MS-LIB exits to the operating system. When the operating system prompt is displayed, MS-LIB has finished the library session successfully. If the library session is unsuccessful, MS-LIB returns the appropriate error message.
MS-LIB prompts you for the name of the library file, the operation(s) you want to perform, and the name you want to give to a cross reference listing file, if any.

Library file:

Enter the name of the library file that you want to manipulate. MS-LIB assumes that the filename extension is .LIB. You can override this assumption by giving a filename extension when you enter the library filename. Because MS-LIB can manage only one library file at a time, only one filename is allowed in response to this prompt. Additional responses, except the semicolon command character, are ignored.
If you enter a library filename and follow it immediately with a semicolon command character, MS-LIB will perform a consistency check only, then return to the operating system. Any errors in the file will be reported.
If the filename you enter does not exist, MS-LIB returns the prompt:

```
Library file does not exist. Create?
```

You must enter either Yes or No, in either upper or lower (or mixed) case. Actually, MS-LIB checks the response of the letter Y as the first character. If any other character is entered first, MS-LIB terminates and returns to the operating system.

Operation:

Enter one of the three command characters for manipulating modules (+, -, *), followed immediately (no space) by the module name or the object filename. Plus sign appends an object file as the last module in the library file (see further discussion under the description of plus sign below). Minus sign deletes a module from the library file. Asterisk extracts a module from the library and places it in a separate object file with the filename taken from the module name and a filename extension .OBJ.

When you have a large number of modules to manipulate (more than can be typed on one line), enter an ampersand (&) as the last character on the line. MS-LIB will repeat the Operation prompt, which permits you to enter additional module names and object filenames.

MS-LIB allows you to enter operations on modules and object files in any order you want.

More information about order of execution and what MS-LIB does with each module is given in the descriptions of each Command Character.

List file:

If you want a cross reference list of the PUBLIC symbols in the modules in the library file after your manipulations, enter a filename in which you want MS-LIB to place the cross reference listing. If you do not enter a filename, no cross reference listing is generated (a NUL file).

The response to the List file prompt is a file specification. Therefore, you can specify, along with the filename, a drive (or device) designation and a filename extension. The List file is not given a default filename extension. If you want the file to have a filename extension, you must specify it when entering the filename.

The cross reference listing file contains two lists. The first list is an alphabetical listing of all PUBLIC symbols. Each symbol name is followed by the name of its module. The second list is an alphabetical list of the modules in the library. Under each module name is an alphabetical listing of the PUBLIC symbols in that module.

1.3 COMMAND CHARACTERS

MS-LIB provides six command characters: three of the command characters are required in responses to the Operation prompt; the other three command characters provide you additional helpful commands to MS-LIB.

+ The plus sign followed by an object filename appends the object file as the last module in the library named in response to the Library file prompt. When MS-LIB sees the plus sign, it assumes that the filename extension is .OBJ. You may override this assumption by specifying a different filename extension.

MS-LIB strips the drive designation and the extension from the object file specification, leaving only the filename. For example, if the object file to be appended as a module to a library is:

```
B:CURSOR.OBJ
```

a response to the Operation prompt of:

```
+B:CURSOR.OBJ
```

causes MS-LIB to strip off the B: and the .OBJ, leaving only CURSOR, which becomes a module named CURSOR in the library.

NOTE

The distinction between an object file and a module (or object module) is that the file possesses a drive designation (even if it is default drive) and a filename extension. Object modules possess neither of these.

- The minus sign followed by a module name deletes that module from the library file. MS-LIB then "closes up" the file space left empty by the deletion. This cleanup action keeps the library file from growing larger than necessary with empty space. Remember that new modules, even replacement modules are added to the end of the file, not stuffed into space vacated by deleting modules.

* The asterisk followed by a module name extracts that module from the library file and places it into a separate object file. The module will still exist in the library (extract means, essentially, copy the module to a separate object file). The module name is used as the filename. MS-LIB adds the default drive designation and the filename extension .OBJ. For example, if the module to be extracted is:

```
CURSOR
```

and the current default disk drive is A:, a reponse to the Operation prompt of:

```
*CURSOR
```

causes MS-LIB to extract the module named CURSOR from the library file and to set it up as an object file with the file specification of:

```
default drive:CURSOR.OBJ
```

(The drive designation and filename extension cannot be overridden. You can, however, rename the file, giving a new filename extension, and/or copy the file to a new disk drive, giving a new filename and/or filename extension.)

; Use a single semicolon (;) followed immediately by a carriage return at any time after responding to the first prompt (from Library file on) to select default responses to the remaining prompts. This feature saves time and overrides the need to answer additional prompts.

NOTE

Once the semicolon has been entered, you can no longer respond to any of the prompts for that library session. Therefore, do not use the semicolon to skip over some prompts. For this, use carriage return.

Example:

```
Library file: FUN <CR>
Operation: +CURSOR;<CR>
```

The remaining prompt will not appear, and MS-LIB will use the default value (no cross reference file).

& Use the ampersand to extend the current physical line. This command character will only be needed for the Operation prompt. MS-LIB can perform many functions during a single library session. The number of modules you can append is limited only by disk space. The number of modules you can replace or extract is also limited only by disk space. The number of modules you can delete is limited only by the number of modules in the library file. However, the line length for a response to any prompt is limited to the line length of your system. For a large number of responses to the Operation prompt, place an ampersand at the end of a line. MS-LIB will display the Operation prompt again, then enter more responses. You may use the ampersand character as many times as you need. For example:

```
Library file: FUN<CR>
Operation: +CURSOR-HEAP+HEAP*FOIBLES&
Operation: *INIT+ASSUME+RIDE;<CR>
```

MS-LIB will delete the module HEAP, extract the modules FOIBLES and INIT (creating two files, FOIBLES.OBJ and INIT.OBJ), then append the object files CURSOR, HEAP, ASSUME, and RIDE. Note, however, that MS-LIB allows you to enter your Operation responses in any order.

Control-C

Use Control-C at any time to abort the library session. If you enter an erroneous response, such as the wrong filename or module name, or an incorrectly spelled filename or module name, you must press CTRL-C to exit MS-LIB then reinvoke MS-LIB and start over. If the error has been typed but not entered, you may delete the erroneous characters, but for that line only.

CHAPTER 2
ERROR MESSAGES

<symbol> is a multiply defined PUBLIC. Proceed?
Cause: two modules define the same public symbol. The user is asked to confirm the removal of the definition of the old symbol. A No response leaves the library in an undetermined state.
Cure: Remove the PUBLIC declaration from one of the object modules and recompile or reassemble.

Allocate error on VM.TMP
Cause: out of space
Cannot create extract file
Cause: no room in directory for extract file
Cannot create list file
Cause: No room in directory for library file
Cannot nest response file
Cause: "@filespec" in response (or indirect) file
Cannot open VM.TMP
Cause: no room for VM.TMP in disk directory
Cannot write library file
Cause: Out of space
Close error on extract file
Cause: out of space
Error: An internal error has occurred.
Contact Microsoft, Inc.
Fatal Error: Cannot open input file
Cause: Mistyped object file name
Fatal Error: Module is not in the library
Cause: trying to delete a module that is not in the library
Input file read error
Cause: bad object module or faulty disk
Invalid object module/library
Cause: bad object and/or library
Library Disk is full
Cause: no more room on diskette
Listing file write error
Cause: out of space

No library file specified

Cause: no response to Library File prompt

Read error on VM.TMP

Cause: disk not ready for read

Symbol table capacity exceeded

Cause: too many public symbols (about 30K chars in symbols)

Too many object modules

Cause: more than 500 object modules

Too many public symbols

Cause: 1024 public symbols maximum

Write error on library/extract file

Cause: Out of space

Write error on VM.TMP

Cause: out of space

NCR

DEBUG Utility

DEBUG UTILITY
CONTENTS

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW OF DEBUG

The Microsoft DEBUG Utility (DEBUG) is a debugging program that provides a controlled testing environment for binary and executable object files. Note that EDLIN is used to alter source files; DEBUG is EDLIN's counterpart for binary files. DEBUG eliminates the need to reassemble a program to see if a problem has been fixed by a minor change. It allows you to alter the contents of a file or the contents of a CPU register, and then to immediately reexecute a program to check on the validity of the changes.

All DEBUG commands may be aborted at any time by pressing <CONTROL-C>. <CONTROL-S> suspends the display, so that you can read it before the output scrolls away. Entering any key other than <CONTROL-C> or <CONTROL-S> restarts the display. All of these commands are consistent with the control character functions available at the MS-DOS command level.

1.2 HOW TO START DEBUG

DEBUG may be started two ways. By the first method, you type all commands in response to the DEBUG prompt (a hyphen). By the second method, you type all commands on the line used to start DEBUG.

Summary of Methods to Start DEBUG

Method 1	DEBUG
Method 2	DEBUG [<filespec> [<arglist>]]

1.2.1 Method 1: DEBUG

To start DEBUG using method 1, type:

```
DEBUG
```

DEBUG responds with the hyphen (-) prompt, signaling that it is ready to accept your commands. Since no filename has been specified, current memory, disk sectors, or disk files can be worked on by using other commands.

Warnings

1. When DEBUG (Version 2.0) is started, it sets up a program header at offset 0 in the program work area. On previous versions of DEBUG, you could overwrite this header. You can still overwrite the default header if no <filespec> is given to DEBUG. If you are debugging a .COM or .EXE file, however, do not tamper with the program header below address 5CH, or DEBUG will terminate.
2. Do not restart a program after the "Program terminated normally" message is displayed. You must reload the program with the N and L commands for it to run properly.

1.2.2 Method 2: Command Line

To start DEBUG using a command line, type:

```
DEBUG [<filespec> [<arglist>]]
```

For example, if a <filespec> is specified, then the following is a typical command to start DEBUG:

```
DEBUG FILE.EXE
```

DEBUG then loads FILE.EXE into memory starting at 100 hexadecimal in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory.

An <arglist> may be specified if <filespec> is present. The <arglist> is a list of filename parameters and switches that are to be passed to the program <filespec> . Thus, when <filespec> is loaded into memory, it is loaded as if it had been started with the command:

```
<filespec>  <arglist>
```

Here, <filespec> is the file to be debugged, and the <arglist> is the rest of the command line that is used when <filespec> is invoked and loaded into memory.

CHAPTER 2 COMMANDS

2.1 COMMAND INFORMATION

Each DEBUG command consists of a single letter followed by one or more parameters. Additionally, the control characters and the special editing functions described in the **MS-DOS User's Guide,** apply inside DEBUG.

If a syntax error occurs in a DEBUG command, DEBUG reprints the command line and indicates the error with an up-arrow (^) and the word "error."

For example:

```
dcs:100 cs:110
      ^ error
```

Any combination of uppercase and lowercase letters may be used in commands and parameters.

The DEBUG commands are summarized in Table 2.1 and are described in detail, with examples, following the description of command parameters.

Table 2.1 DEBUG COMMANDS

DEBUG Command	Function
A[<address>]	Assemble
C<range> <address>	Compare
D[<range>]	Dump
E<address> [<list>]	Enter
F<range> <list>	Fill
G[=<address> [<address>. . .]]	Go
H<value> <value>	Hex
I<value>	Input
L[<address> [<drive> <record> <record>]]	Load
M<range> <address>	Move
N<filename> [<filename>]	Name
O<value> <byte>	Output
Q	Quit
R[<register-name>]	Register
S<range> <list>	Search
T[=<address>] [<value>]	Trace
U[<range>]	Unassemble
W[<address> [<drive> <record> <record>]]	Write

2.2 PARAMETERS

All DEBUG commands accept parameters, except the Quit command. Parameters may be separated by delimiters (spaces or commas), but a delimiter is required only between two consecutive hexadecimal values. Thus, the following commands are equivalent:

```
dcs:100 110
d cs:100 110
d,cs:100,110
```

PARAMETER DEFINITION

<drive> A one-digit hexadecimal value to indicate which drive a file will be loaded from or written to. The valid values are 0-3. These values designate the drives as follows: 0=A:, 1=B:, 2=C:, 3=D:.

<byte> A two-digit hexadecimal value to be placed in or read from an address or register.

<record> A 1- to 3-digit hexadecimal value used to indicate the logical record number on the disk and the number of disk sectors to be written or loaded. Logical records correspond to sectors. However, their numbering differs since they represent the entire disk space.

<value> A hexadecimal value up to four digits used to specify a port number or the number of times a command should repeat its functions.

<address> A two-part designation consisting of either an alphabetic segment register designation or a four-digit segment address plus an offset value. The segment designation or segment address may be omitted, in which case the default segment is used. DS is the default segment for all commands except G, L, T, U, and W, for which the default segment is CS. All numeric values are hexadecimal.

For example:

```
CS:0100
04BA:0100
```

The colon is required between a segment designation (whether numeric or alphabetic) and an offset.

<range< Two <address>es: e.g., <address> <address>; or one <address>, an L, and a <value>: e.g., <adress> L <value> where <value> is the number of lines the command should operate on, and LB0 is assumed. The last form cannot be used if another hex value follows the <range>, since the hex value would be interpreted as the second <address> of the <range>.

Examples:

```
CS:100 110
CS:100 L 10
CS:100
```

The following is illegal:

```
CS:100 CS:110
       ^ error
```

The limit for <range> is 10 000 hex. To specify a <value> of 10 000 hex within four digits, type 0000 (or 0).

<list> A series of <byte> values or of <string>s. <list> must be the last parameter on the command line.

Example:

```
fcs:100 42 45 52 54 41
```

<string> Any number of characters enclosed in quote marks. Quote marks may be either single (’) or double (“). If the delimiter quote marks must appear within a <string>, the quote marks must be doubled. For example, the following strings are legal:

```
’This is a “string” is okay.’
’This is a “string” is okay.’
```

However, this string is illegal:

```
’This is a ’string’ is not.’
```

Similarly, these strings are legal:

```
“This is a ’string’ is okay.”
“This is a ““string”” is okay.”
```

However, this string is illegal:

> “This is a “string” is not.”

Note that the double quote marks are not necessary in the following strings:

> “This is a ”string” is not necessary.”
> ’This is a ““string”” is not necessary.’

The ASCII values of the characters in the string are used as a <list> of byte values.

NAME Assemble

PURPOSE Assembles 8086/8087/8088 mnemonics directly into memory.

SYNTAX A[<address>]

COMMENTS If a syntax error is found, DEBUG responds with

```
^Error
```

and redisplays the current assembly address.

All numeric values are hexadecimal and must be entered as 1-4 characters. Prefix mnemonics must be specified in front of the opcode to which they refer. They may also be entered on a separate line.

The segment override mnemonics are CS:, DS:, ES:, and SS:. The mnemonic for the far return is RETF.

String manipulation mnemonics must explicitly state the string size. For example, use MOVSW to move word strings and MOVSB to move byte strings.

The assembler will automatically assemble short, near or far jumps and calls, depending on byte displacement to the destination address. These may be overridden with the NEAR or FAR prefix. For example:

```
0100:0500 JMP  502        ; a 2-byte short jump
0100:0502 JMP  NEAR 505   ; a 3-byte near jump
0100:505  JMP  FAR 50A    ; a 5-byte far jump
```

The NEAR prefix may be abbreviated to NE, but the FAR prefix cannot be abbreviated.

DEBUG cannot tell whether some operands refer to a word memory location or to a byte memory location. In this case, the data type must be explicitly stated with the prefix “WORD PTR” or “BYTE PTR”. Acceptable abbreviations are “WO” and “BY”. For example:

```
NEG   BYTE PTR [128]
DEC   WO [SI]
```

DEBUG also cannot tell whether an operand refers to a memory location or to an immediate operand. DEBUG uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
MOV    AX,21    ; Load AX with 21H
MOV    AX,[21]  ; Load AX with the
                ; contents
                ; of memory location 21H
```

Two popular pseudo-instructions are available with Assemble. The DB opcode will assemble byte values directly into memory. The DW opcode will assemble word values directly into memory. For example:

```
DB     1,2,3,4,“THIS IS AN EXAMPLE”
DB     ’THIS IS A QUOTE: “ ’
DB     “THIS IS A QUOTE: ’ ”

DW     1000,2000,3000,“BACH”
```

Assemble supports all forms of register indirect commands. For example:

```
ADD    BX,34[BP+2].[SI-1]
POP    [BP+DI]
PUSH   [SI]
```

All opcode synonyms are also supported. For example:

```
LOOPZ  100
LOOPE  100

JA     200
JNBE   200
```

For 8087 opcodes, the WAIT or FWAIT must be explicitly specified. For example:

```
FWAIT FADD ST,ST(3) ; This line will assemble
                    ; an FWAIT prefix
LD TBYTE PTR [BX]   ; This line will not
```

NAME Compare

PURPOSE Compares the portion of memory specified by <range> to a portion of the same size beginning at <address>.

SYNTAX C<range> <address>

COMMENTS If the two areas of memory are identical, there is no display and DEBUG returns with the MS-DOS prompt. If there **are** differences, they are displayed in this format:

```
<address1> <byte1> <byte2> <address2>
```

EXAMPLE The following commands have the same effect:

```
C100,1FF  300
    or
C100L100  300
```

Each command compares the block of memory from 100 to 1FFH with the block of memory from 300 to 3FFH.

NAME Dump

PURPOSE Displays the contents of the specified region of memory.

SYNTAX D[<range>]

COMMENTS If a range of addresses is specified, the contents of the range are displayed. If the D command is typed without parameters, 128 bytes are displayed at the first address (DS:100) after the address displayed by the previous Dump command.

The dump is displayed in two portions: a hexadecimal dump (each byte is shown in hexadecimal value) and an ASCII dump (the bytes are shown in ASCII characters). Nonprinting characters are denoted by a period (.) in the ASCII portion of the display. Each display line shows 16 bytes with a hyphen between the eighth and ninth bytes. At times, displays are split in this manual to fit them on the page. Each displayed line begins on a 16-byte boundary.

If you type the command:

```
dcs:100 110
```

DEBUG displays the dump in the following format:

```
04BA:0100 42 45 52 54 41 . . . 4E 44 TOM SAWYER
```

If you type the following command:

```
D
```

the display is formatted as described above. Each line of the display begins with an address, incremented by 16 from the address on the previous line. Each subsequent D (typed without parameters) displays the bytes immediately following those last displayed.

If you type the command:

```
DCS:100 L 20
```

the display is formatted as described above, but 20H bytes are displayed.
If then you type the command:

```
DCS:100 115
```

the display is formatted as described above, but all the bytes in the range of lines from 100H to 115H in the CS segment are displayed.

NAME Enter

PURPOSE Enters byte values into memory at the specified <address>.

SYNTAX E<address> [<list>]

COMMENTS If the optional <list> of values is typed, the replacement of byte values occurs automatically. (If an error occurs, no byte values are changed.)
If the <address> is typed without the optional <list>, DEBUG displays the address and its contents, then repeats the address on the next line and wait for your input. At this point, the Enter command waits for you to perform one of the following actions:

1. Replace a byte value with a value you type. Simply type the value after the current value. If the value typed in is not a legal hexadecimal value or if more than two digits are typed, the illegal or extra character is not echoed.
2. Press the <SPACE> bar to advance to the next byte. To change the value, simply type the new value as described in (1.) above. If you space beyond an 8-byte boundary, DEBUG starts a new display line with the address displayed at the beginning.
3. Type a hyphen (-) to return to the preceding byte. If you decide to change a byte behind the current position, typing the hyphen returns the current position to the previous byte. When the hyphen is typed, a new line is started with the address and its byte value displayed.
4. Press the <RETURN> key to terminate the Enter command. The <RETURN> key may be pressed at any byte position.

EXAMPLE

Assume that the following command is typed:

```
ECS:100
```

DEBUG displays:

```
04BA:0100   EB.-
```

To change this value to 41, type 41 as shown:

```
04BA:0100   EB.41-
```

To step through the subsequent bytes, press the <SPACE> bar to see:

```
04BA:0100    EB.41     10.     00.     BC.-
```

To change BC to 42:

```
04BA:0100    EB.41     10.     00.     BC.42-
```

Now, realizing that 10 should be 6F, type the hyphen as many times as needed to return to byte 0101 (value 10), then replace 10 with 6F:

```
04BA:0100     EB.41      10.      00.      BC.42-
04BA:0102     00.--
04BA:0101     10.6F-
```

Pressing the <RETURN> key ends the Enter command and returns to the DEBUG command level.

NAME Fill

PURPOSE Fills the addresses in the <range> with the values in the <list>.

SYNTAX F<range> <list>

COMMENTS If the <range> contains more bytes than the number of values in the <list>, the <list> will be used repeatedly until all bytes in the <range> are filled. If the <list> contains more values than the number of bytes in the <range>, the extra values in the <list> will be ignored. If any of the memory in the <range> is not valid (bad or nonexistent), the error will occur in all succeeding locations.

EXAMPLE Assume that the following command is typed:

```
F04BA:100 L 100 42 45 52 54 41
```

DEBUG fills memory locations 04BA:100 through 04BA:1FF with the bytes specified. The five values are repeated until all 100H bytes are filled.

NAME	Go
PURPOSE	Executes the program currently in memory.
SYNTAX	G [=<address> [<address>. . .]]

COMMENTS If only the Go command is typed, the program executes as if the program had run outside DEBUG.

If = <address> is set, execution begins at the address specified. The equal sign (=) is required, so that DEBUG can distinguish the start = <address> from the breakpoint <address>es.

With the other optional addresses set, execution stops at the first <address> encountered, regardless of that address' position in the list of addresses to halt execution or program branching. When program execution reaches a breakpoint, the registers, flags, and decoded instruction are displayed for the last instruction executed. (The result is the same as if you had typed the Register command for the breakpoint address.)

Up to ten breakpoints may be set. Breakpoints may be set only at addresses containing the first byte of an 8086 opcode. If more than ten breakpoints are set, DEBUG returns the BP Error message.

The user stack pointer must be valid and have 6 bytes available for this command. The G command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set, and the user flags, Code Segment register, and Instruction Pointer are pushed on the user stack. (Thus, if the user stack is not valid or is too small, the operating system may crash.) An interrupt code (0CCH) is placed at the specified breakpoint address(es).

When an instruction with the breakpoint code is encountered, all breakpoint addresses are restored to their original instructions. If execution is not halted at one of the breakpoints, the interrupt codes are not replaced with the original instructions.

EXAMPLE

Assume that the following command is typed:

```
GCS:7550
```

The program currently in memory executes up to the address 7550 in the CS segment. DEBUG then displays registers and flags, after which the Go command is terminated.

After a breakpoint has been encountered, if you type the Go command again, then the program executes just as if you had typed the filename at the MS-DOS command level. The only difference is that program execution begins at the instruction after the breakpoint rather than at the usual start address.

NAME Hex

PURPOSE Performs hexadecimal arithmetic on the two parameters specified.

SYNTAX H<value> <value>

COMMENTS First, DEBUG adds the two parameters, then subtracts the second parameter from the first. The results of the arithmetic are displayed on one line; first the sum, then the difference.

EXAMPLE Assume that the following command is typed:

```
H19F 10A
```

DEBUG performs the calculations and then displays the result:

```
02A9   0095
```

NAME Input

PURPOSE Inputs and displays one byte from the port specified by <value>.

SYNTAX I<value>

COMMENTS A 16-bit port address is allowed.

EXAMPLE Assume that you type the following command:

```
I2F8
```

Assume also that the byte at the port is 42H. DEBUG inputs the byte and displays the value:

```
42
```

NAME Load

PURPOSE Loads a file into memory.

SYNTAX L[<address> [<drive> <record> <record>]]

COMMENTS Set BX:CX to the number of bytes read. The file must have been named either when DEBUG was started or with the N command. Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C.

If the L command is typed without any parameters, DEBUG loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded. If the L command is typed with an address parameter, loading begins at the memory <address> specified. If L is typed with all parameters, absolute disk sectors are loaded, not a file. The <record>s are taken from the <drive> specified (the drive designation is numeric here-0=A:, 1=8:, 2=C:, etc.); DEBUG begins loading with the first <record> specified, and continues until the number of sectors specified in the second <record> have been loaded.

EXAMPLE Assume that the following commands are typed:

```
A>DEBUG
-NFILE.COM
```

Now, to load FILE.COM, type:

```
L
```

DEBUG loads the file and then displays the DEBUG prompt. Assume that you want to load only portions of a file or certain records from a disk. To do this, type:

```
L04BA:100 2 0F 6D
```

DEBUG then loads 109 (6D hex) records beginning with logical record number 15 into memory beginning at address 04BA:0100. When the records have been loaded, DEBUG simply returns the - prompt.

If the file has a .EXE extension, it is relocated to the load address specified in the header of the .EXE file: the <address> parameter is always ignored for .EXE files. The header itself is stripped off the .EXE file before it is loaded into memory. Thus the size of an .EXE file on disk will differ from its size in memory.

If the file named by the Name command or specified when DEBUG is started is a .HEX file, then typing the L command with no parameters causes DEBUG to load the file beginning at the address specified in the .HEX file. If the L command includes the option <address>, DEBUG adds the <address> specified in the L command to the address found in the .HEX file to determine the start address for loading the file.

NAME Move

PURPOSE Moves the block of memory specified by <range> to the location beginning at the <address> specified.

SYNTAX M<range> <address>

COMMENTS Overlapping moves (i.e., moves where part of the block overlaps some of the current addresses) are always performed without loss of data. Addresses that could be overwritten are moved first. The sequence for moves from higher addresses to lower addresses is to move the data beginning at the block's lowest address and then to work towards the highest. The sequence for moves from lower addresses to higher addresses is to move the data beginning at the block's highest address and to work towards the lowest.

Note that if the addresses in the block being moved will not have new data written to them, the data there before the move will remain. The M command copies the data from one area into another, in the sequence described, and writes over the new addresses. This is why the sequence of the move is important.

EXAMPLE Assume that you type:

```
MCS:100 110 CS:500
```

DEBUG first moves address CS:110 to address CS:510, then CS:10F to CS:50F, and so on until CS:100 is moved to CS:500. You should type the D command, using the <address> typed for the M command, to review the results of the move.

NAME Name

PURPOSE Sets filenames.

SYNTAX N<filename> [<filename> . . .]

COMMENTS The Name command performs two functions. First, Name is used to assign a filename for a later Load or Write command. Thus, if you start DEBUG without naming any file to be debugged, then the N<filename> command must be typed before a file can be loaded. Second, Name is used to assign filename parameters to the file being debugged. In this case, Name accepts a list of parameters that are used by the file being debugged.

These two functions overlap. Consider the following set of DEBUG commands:

```
-NFILE1.EXE
-L
-G
```

Because of the effects of the Name command, Name will perform the following steps:

1. (N)ame assigns the filename FILE1.EXE to the filename to be used in any later Load or Write commands.
2. (N)ame also assigns the filename FILE1.EXE to the first filename parameter used by any program that is later debugged.
3. (L)oad loads FILE1.EXE into memory.
4. (G)o causes FILE1.EXE to be executed with FILE1.EXE as the single filename parameter (that is, FILE1.EXE is executed as if FILE1.EXE had been typed at the command level).

A more useful chain of commands might look like this:

```
-NFILE1.EXE
-L
-NFILE2.DAT FILE3.DAT
-G
```

Here, Name sets FILE1.EXE as the filename for the subsequent Load command. The Load command loads FILE1.EXE into memory, and then the Name command is used again, this time to specify the parameters to be used by FILE1.EXE. Finally, when the Go command is executed, FILE1.EXE is executed as if FILE1 FILE2.DAT FILE3.DAT had been typed at the MS-DOS command level. Note that if a Write command were executed at this point, then FILE1.EXE - the file being debugged - would be saved with the name FILE2.DAT! To avoid such undesired results, you should always execute a Name command before either a Load or a Write.

There are four regions of memory that can be affected by the Name command:

CS:5C	FCB for file 1
CS:6C	FCB for file 2
CS:80	Count of characters
CS:81	All characters typed

A File Control Block (FCB) for the first filename parameter given to the Name command is set up at CS:5C. If a second filename parameter is typed, then an FCB is set up for it beginning at CS:6C. The number of characters typed in the Name command exclusive of the first character, "N") is given at location CS:80. The actual stream of characters given by the Name command (again, exclusive of the letter "N") begins at CS:81. Note that this stream of characters may contain switches and delimiters that would be legal in any command typed at the MS-DOS command level.

EXAMPLE

A typical use of the Name command is:

```
DEBUG PROG.COM
-NPARAM1 PARAM2/C
-G
-
```

In this case, the Go command executes the file in memory as if the following command line had been typed:

```
PROG PARAM1 PARAM2/C
```

Testing and debugging therefore reflect a normal runtime environment for PROG.COM.

NAME Output

PURPOSE Sends the <byte> specified to the output port specified by <value>.

SYNTAX 0<value> <byte>

COMMENTS A 16-bit port address is allowed.

EXAMPLE Type:

```
02F8 4F
```

DEBUG outputs the byte value 4F to output port 2F8.

NAME Quit

PURPOSE Terminates the DEBUG utility.

SYNTAX Q

COMMENTS The Q command takes no parameters and exits DEBUG without saving the file currently being operated on. You are returned to the MS-DOS command level.

EXAMPLE To end the debugging session, type:

```
Q<RETURN>
```

DEBUG has been terminated, and control returns to the MS-DOS command level.

NAME Register

PURPOSE Displays the contents of one or more CPU registers.

SYNTAX R[<register-name>]

COMMENTS If no <register-name> is typed, the R command dumps the register save area and displays the contents of all registers and flags.

If a register name is typed, the 16-byte value of that register is displayed in hexadecimal, and then a colon appears as a prompt. You then either type a <value> to change the register, or simply press the <RETURN> key if no change is wanted.

The only valid <register-name>s are:

AX
BX
CX
DX
SP

BP
SI
DI
DS
ES

SS
CS
IP
PC
F

(IP and PC both refer to the Instruction Pointer.)

Any other entry for <register-name> results in a BR Error message.

If F is entered as the <register-name>, DEBUG displays each flag with a two-character alphabetic code. To alter any flag, type the opposite two-letter code. The flags are either set or cleared.

The flags are listed below with their codes for SET and CLEAR:

FLAG NAME	SET	CLEAR
Overflow	OV	NV
Direction	DN Decrement	UP Increment
Interrupt	EI Enabled	DI Disabled
Sign	NG Negative	PL Plus
Zero	ZR	NZ
Auxiliary Carry	AC	NA
Parity	PE Even	PO Odd
Carry	CY	NC

Whenever you type the command RF, the flags are displayed in the order shown above in a row at the beginning of a line. At the end of the list of flags, DEBUG displays a hyphen (-). You may enter new flag values as alphabetic pairs. The new flag values can be entered in any order. You do not have to leave spaces between the flag entries. To exit the R command, press the <RETURN> key. Flags for which new values were not entered remain unchanged.

If more than one value is entered for a flag, DEBUG returns a DF Error message. If you enter a flag code other than those shown above, DEBUG returns a BF Error message. In both cases, the flags up to the error in the list are changed; flags at and after the error are not.

At startup, the segment registers are set to the bottom of free memory, the Instruction Pointer is set to 0100H, all flags are cleared, and the remaining registers are set to zero.

EXAMPLE

Type:

```
R
```

DEBUG displays all registers, flags, and the decoded instruction for the current location. If the location is CS:11A, then the display will look similar to this:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D
BP=0000 SI=005C DI=0000 DS=04BA ES=04BA
SS=04BA CS=04BA IP=011A
NV UP DI NG NZ AC PE NC
04BA:011A   CD21        INT     21
```

If you type:

```
RF
```

DEBUG will display the flags:

```
NV UP DI NG NZ AC PE NC - -
```

Now, type any valid flag designation, in any order, with or without spaces.

For example:

```
NV UP DI NG NZ AC PE NC - PLEICY<RETURN>
```

DEBUG responds only with the DEBUG prompt. To see the changes, type either the R or RF command:

```
RF
NV UP EI PL NZ AC PE CY - -
```

Press <RETURN> to leave the flags this way, or to specify different flag values.

NAME Search

PURPOSE Searches the <range< specified for the <list> of bytes specified.

SYNTAX S<range> <list>

COMMENTS The <list> may contain one or more bytes, each separated by a space or comma. If the <list> contains more than one byte, only the first address of the byte string is returned. If the <list> contains only one byte, all addresses of the byte in the <range> are displayed.

EXAMPLE If you type:

```
SCS:100 110 41
```

DEBUG will display a response similar to this:

```
04BA:0104
04BA:010D
-type:
```

NAME Trace

PURPOSE Executes one instruction and displays the contents of all registers and flags, and the decoded instruction.

SYNTAX T[=<address>] [<value>]

COMMENTS If the optional =<address> is typed, tracing occurs at the =<address> specified. The optional <value> causes DEBUG to execute and trace the number of steps specified by <value>.

The T command uses the hardware trace mode of the 8086 or 8088 microprocessor. Consequently, you may also trace instructions stored in ROM (Read Only Memory).

EXAMPLE TYPE:

```
T
```

DEBUG returns a display of the registers, flags, and decoded instruction for that one instruction. Assume that the current position is 04BA:011A; DEBUG might return the display:

```
AX=0E00 BX=00FF CS=0007 DX=01FF SP=039D
BP=0000 SI=005C DI=0000 DS=04BA ES=04BA
SS=04BA CS=04BA IP=011A
NV UP DI NG NZ AC PE NC
04BA:011A   CD21        INT     21
```

If you type

```
T=011A 10
```

DEBUG executes sixteen (10 hex) instructions beginning at 011A in the current segment, and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed. Then the display stops, and you can see the register and flag values for the last few instructions performed. Remember that <CONTROL-S> suspends the display at any point, so that you can study the registers and flags for any instruction.

NAME Unassemble

PURPOSE Disassembles bytes and displays the source statements that correspond to them, with addresses and byte values.

SYNTAX U[<range>]

COMMENTS The display of disassembled code looks like a listing for an assembled file. If you type the U command without parameters, 20 hexadecimal bytes are disassembled at the first address after that displayed by the previous Unassemble command. If you type the U command with the <range> parameter, then DEBUG disassembles all bytes in the range. If the <range> is given as an <address> only, then 20H bytes are disassembled instead of 80H.

EXAMPLE Type:

```
U04BA:100 L10
```

DEBUG disassembles 16 bytes beginning at address 04BA:0100:

```
04BA:0100 206472 AND  [SI+72],AH
04BA:0103 69     DB   69
04BA:0104 7665   JBE  016B
04BA:0106 207370 AND  [BP+DI+70],DH
04BA:0109 65     DB   65
04BA:010A 63     DB   63
04BA:010B 69     DB   69
04BA:010C 66     DB   66
04BA:010D 69     DB   69
04BA:010E 63     DB   63
04BA:010F 61     DB   61
```

If you type

```
004ba:0100 0108
```

The display will show:

```
04BA:0100 206472  AND  [SI+72],AH
04BA:0103 69      DB   69
04BA:0104 7665    JBE  016B
04BA:0106 207370  AND  [BP+DI+70],DH
```

If the bytes in some addresses are altered, the disassembler alters the instruction statements. The U command can be typed for the changed locations, the new instructions viewed, and the disassembled code used to edit the source file.

NAME Write

PURPOSE Wirtes the file being debugged to a disk file.

SYNTAX W[<address> [<drive> <record> <records>]]

COMMENTS If you type W with no parameters, BX:CX must already be set to the number of bytes to be written; the file is written beginning from CS:100. If the W command is typed with just an address, then the file is written beginning at that address. If a G or T command has been used, BX:CX must be reset before using the Write command without parameters. Note that if a file is loaded and modified, the name, length, and starting address are all set correctly to save the modified file (as long as the length has not changed).

The file must have been named either with the DEBUG invocation command or with the N command (refer to the Name command earlier in this manual). Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C.

If the W command is typed with parameters, the write begins from the memory address specified; the file is written to the <drive> specified (the drive designation is numeric here-0=A:, 1=B:, 2=C:, etc.); DEBUG writes the file beginning at the logical record number specified by the first <record>; DEBUG continues to write the file until the number of sectors specified in the second <record> have been written.

WARNING

Writing to absolute sectors is **EXTREMELY** dangerous because the process bypasses the file handler.

EXAMPLE Type:

```
W
```

DEBUG will write the file to disk and then display the DEBUG prompt. Two examples are shown below.

```
W
--
```

```
WCS:100 1 37 2B
```

DEBUG writes out the contents of memory, beginning with the address CS:100 to the disk in drive B:. The data written out starts in disk logical record number 37H and consists of 2BH records. When the write is complete, DEBUG displays the prompt:

```
WCS:100 1 37 2B
--
```

2.3 ERROR MESSAGES

During the DEBUG session, you may receive any of the following error messages. Each error terminates the DEBUG command under which it occurred, but does not terminate DEBUG itself.

ERROR CODE DEFINITION

BF Bad flag

You attempted to alter a flag, but the characters typed were not one of the acceptable pairs of flag values. See the Register command for the list of acceptable flag entries.

BP Too many breakpoints

You specified more than ten breakpoints as parameters to the G command. Retype the Go command with ten or fewer breakpoints.

BR Bad register

You typed the R command with an invalid register name. See the Register command for the list of valid register names.

DF Double flag

You typed two values for one flag. You may specify a flag value only once per RF command.