AA-HD95A-TK

VAXmate™

Technical Reference Manual
Volume 2

digital

software

VAXmate™

Technical Reference Manual Volume 2

First Printing, February 1987

Printed in U.S.A.

Contents

Bibliography

Index

Tables

Figures

Preface

Audience

This manual provides reference material about the VAXmate workstation. It covers all programmable components, the firmware, and several MS-DOS related environments. The material and its presentation are directed to experienced programmers or software designers.

Manual Organization

This manual is divided into four parts and appendixes:

- Chapter 1 provides an overview of the VAXmate workstation and optional equipment.
- Chapters 2 through 13 introduce the VAXmate workstation programmable hardware devices. Each chapter discusses a single hardware programming task, such as video input/output (I/O), external interrupt processing, or serial communications and includes the following information:
  - A brief device description
  - A list of additional references
  - A description of the programmable hardware registers
  - A programming example
  - A discussion of the example

  The examples are written in the C programming language to reduce the size of the examples and focus on the task rather than the detail required by the language.
- Chapter 14 describes the power-up diagnostics and system startup.
- Chapter 15 describes the read-only memory basic input/output system (ROM BIOS).
- The appendixes contain additional information, including a bibliography of other useful publications.

Terminology

The following terms are used throughout this manual and are defined as follows:

Term	Definition	
Industry-standard	The computer industry recognizes two open architectures as industry standards, the IBM PC AT bus structure and the Microsoft disk operating system (MS-DOS). Moreover, supporting MS-DOS requires a defined set of ROM BIOS services. The term *industry-standard* refers to compatibility with these architectures.	
Reserved Available Unassigned	To avoid confusion and incompatibility, the use of certain items such as memory space, I/O space, interrupt vectors, and ROM BIOS parameters or return values must be clearly defined. These three categories define those items that do not have a specific use.	
	Reserved	In future hardware or software releases, DIGITAL may define a specific use for this item. Hardware or software applications that use this item may not work with future releases.
	Available	Hardware or software applications can use this item. DIGITAL has defined the specific use of this item as available for applications.
	Unassigned	Hardware or software applications can use this item. However, there remains some risk that DIGITAL may define a specific use for this item.

Federal Communications Commission Radio Frequency Interference

Class A Computing Devices

This equipment generates, uses, and may emit radio frequency energy. The equipment has been tested and found to comply with the limits for a Class A computing device pursuant to Sub-part J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference in which case the user at his own expense may be required to take measures to correct the interference.

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 methods:

- re-orient 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 and television technician for additional suggestions. The user may find the booklet, *How to Identify and Resolve Radio/TV Interference Problems*, prepared by the Federal Communications Commission helpful. This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock No. 004-000-00398-5.

> NOTE
> Shielded cables are provided for use with this device. Should any cables be replaced or added for any reason, these cables should be the same as, or with higher shielding capabilities, than those provided by Digital Equipment Corporation.

Chapter 14
System Startup

Overview

During system startup, the ROM firmware on the processor board runs diagnostic tests, initializes the video, memory, disk controller, and firmware data. Following diagnostic tests and initialization, the firmware tries to load the operating system from the diskette, hard disk, or network.

The ROM diagnostic tests isolate errors to a field-replaceable unit (processor board, I/O board, keyboard, drives, or DIGITAL options). The diagnostic tests have two modes, a 30-second powerup test and a 3-minute extended self-test.

Powerup Test

The powerup test automatically performs a brief check of the system hardware. It performs a processor board test, a keyboard test, an input/output (I/O) board test, a brief video check, and an internal I/O interface test. During the video test, a solid line flashes at the top of the screen. The firmware checks the presence of a diskette controller, a diskette drive, a hard disk controller, and a hard disk drive. If found, they are also tested. The last tests performed are on-board diagnostics for DIGITAL options, such as the modem board. Finally, the firmware initializes the hardware and firmware data. Figures 14-1 through 14-4 show the powerup test sequence.

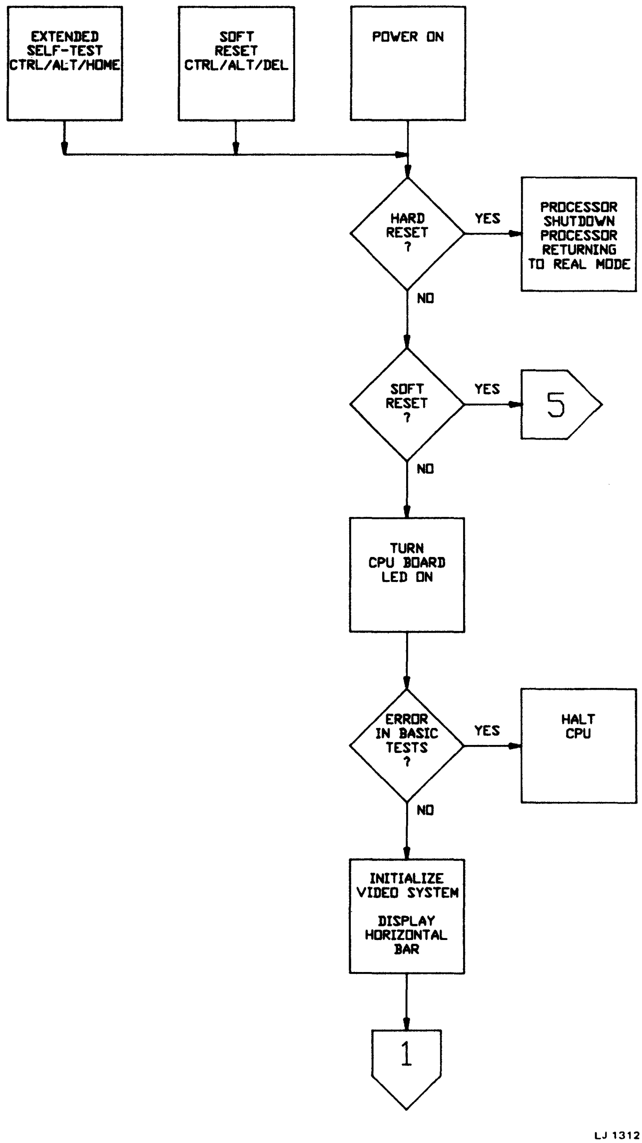

Figure 14-1 Test Sequence - Processor Board

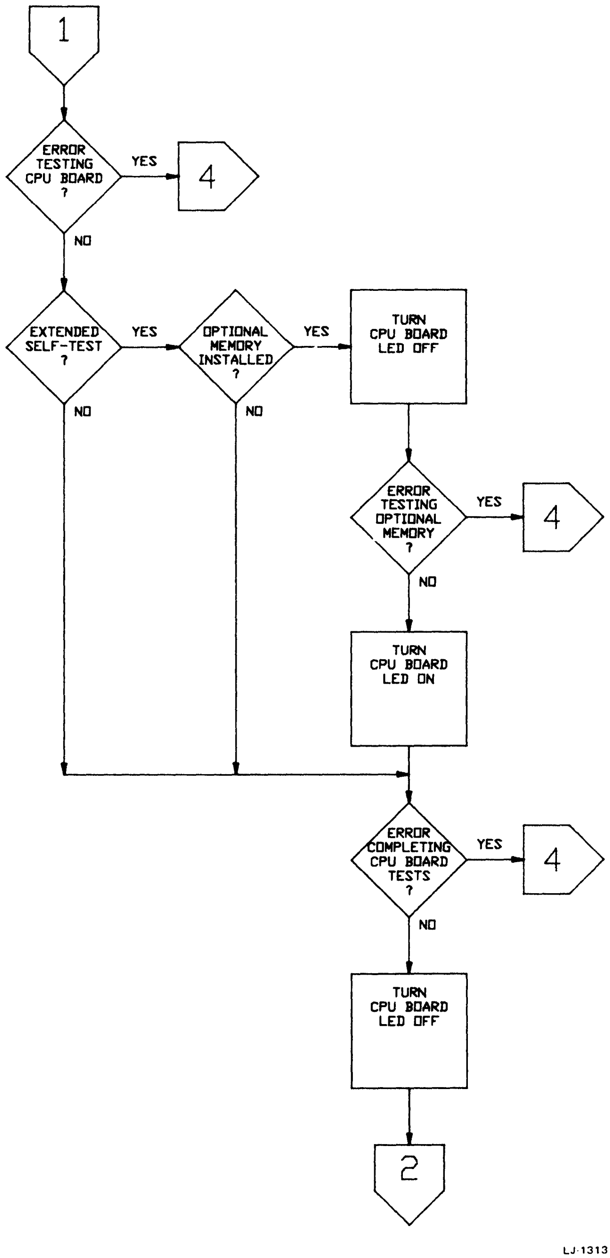

Figure 14-1 Test Sequence - Processor Board (cont.)

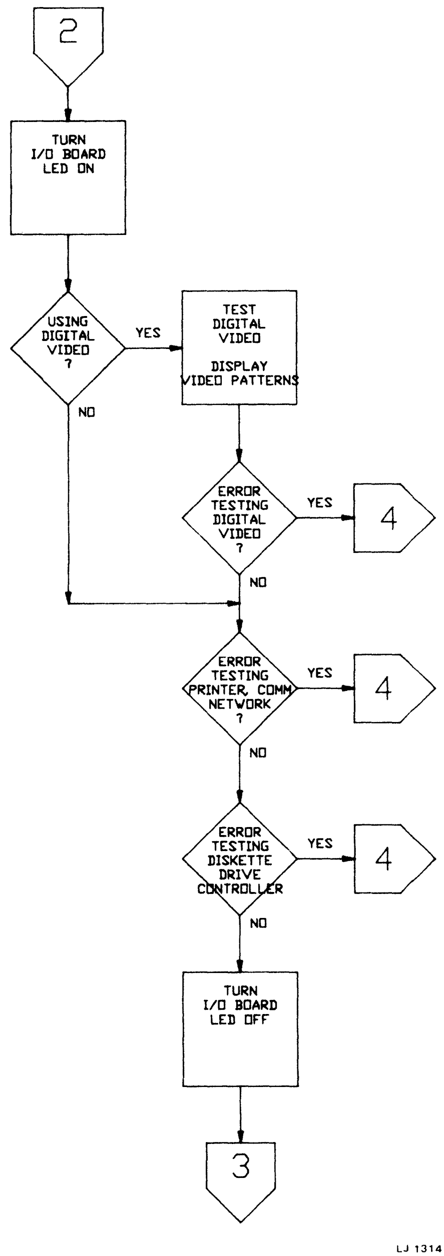

Figure 14-2 Test Sequence - I/O Board

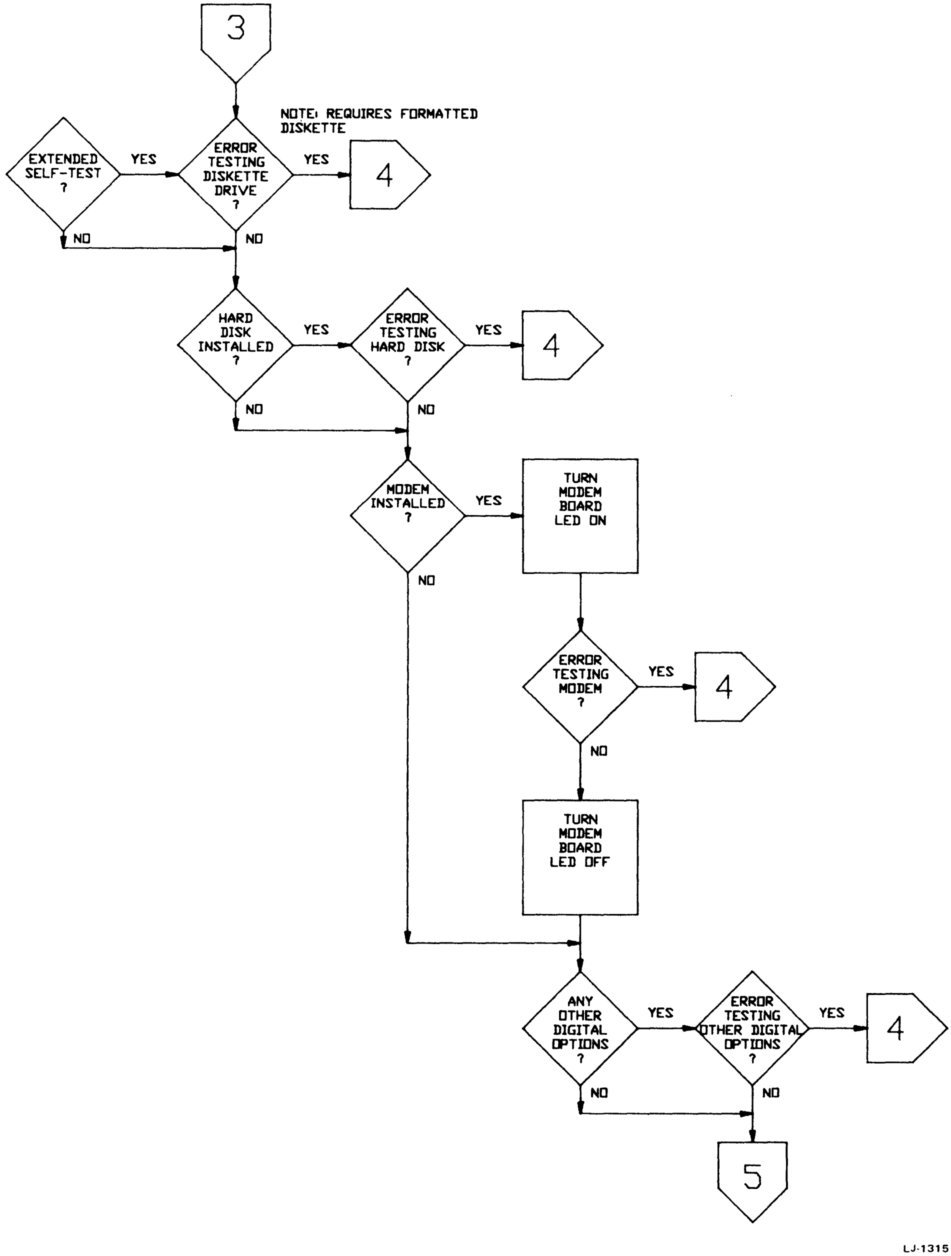

Figure 14-3 Test Sequence - Options

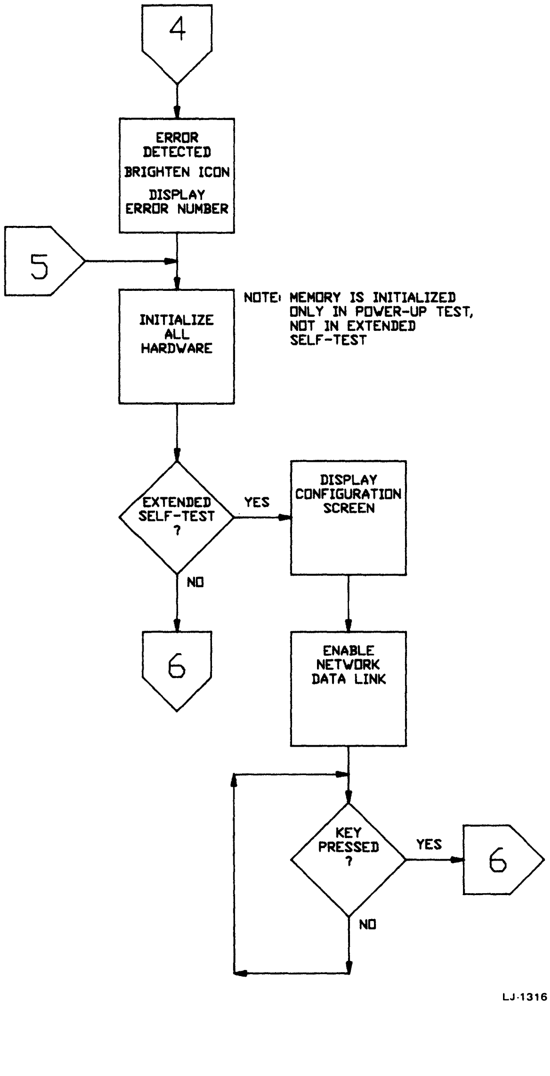

Figure 14-4 Test Sequence - Initialization and Bootstrap

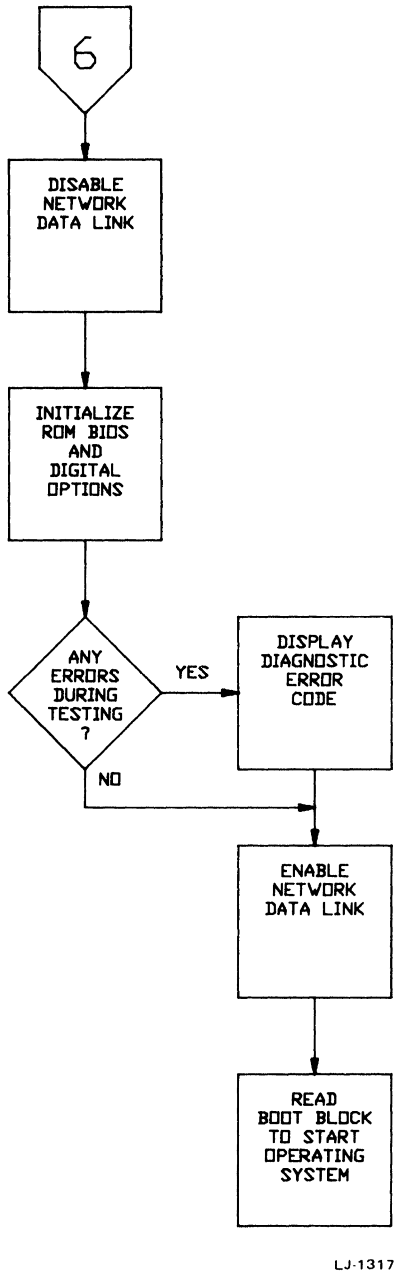

Figure 14-4 Test Sequence - Initialization and Bootstrap (cont.)

During the powerup test, the firmware diagnostic draws a horizontal bar on the screen. As tests complete, the firmware gradually shades in this bar. When the diagnostic detects an error, the filled-in area of the bar changes shade and two beeps sound. A failure value displays below the middle of the bar and remains for 5 seconds. When the bar disappears, the error value moves to the middle of the first line, where it remains until scrolled off the screen. Table 14-1 lists the error codes.

Table 14-1 VAXmate Powerup and Self-Test Error Codes

Code	FRU
00H-1FH	CPU Board
20H-3FH	I/O Board
40H-4FH	Diskette Drive A
50H-5FH	Option Memory
60H-6FH	Keyboard
70H-7FH	Hard Disk Controller
80H-8FH	Hard Disk
90H-9FH	Integral Modem
A0H-FFH	Reserved

Only a few test failures are severe enough to halt the system. These are in the basic tests. If a severe failure occurs, the processor board LED stays on, and two beeps sound.

The tests include:

- Memory access
- Data path validity
- Addressing
- ROM checksum
- Stack and vector area
- Refresh request
- CMOS shutdown byte

When testing of a field-replaceable unit (FRU) completes, the firmware turns off the LED on the FRU. The FRUs with LEDs include the processor board, the I/O board, the memory option board (parity errors only), and the modem option board.

NOTE
Depending on the type of failure, if the video initialization sequence fails, it is possible that the I/O board LED may remain on.

The processor board LED and the I/O board LED are visible through the top of the VAXmate cover. The processor board LED is red color. The I/O board LED is an amber color. The memory board option LED lights up only if the test detects a parity error.

The powerup test checks only DIGITAL supported hardware. For example, the test checks the DIGITAL modem option but does not check other vendor modem boards, unless the vendor adds option ROMs with a powerup test that conforms to a DIGITAL standard.

Initialization

After the powerup test completes, the firmware diagnostic performs an initialization sequence that consists of sizing the memory, initializing up to 15 Mbytes of memory, and initializing the hardware. Then, the firmware diagnostic passes control to the ROM BIOS, which initializes the firmware data, sets up all interrupt vectors, and attempts to load the operating system from the diskette, the hard disk (if installed), or the network.

Real Mode Versus Virtual Protected Mode

The processor can operate in two modes, real mode and virtual protected mode. The coding of programs is distinctly different for these two modes. When operating in real mode (the powerup mode), the processor can access only the first megabyte of physical memory. When operating in virtual protected mode, the processor can access all 16 Mbytes of the physical address space. The powerup test checks only the 640K system RAM and the DEC private RAM that reside in the first megabyte of physical address space.

To prevent parity errors, the memory above the first megabyte, including the 2 Mbytes memory option board, is initialized during power up. The memory above the first megabyte is tested in the extended self-test mode.

Extended Self-Test

Holding down the Ctrl and Alt keys, then pressing the Home key on the numeric keypad invokes the extended self-test. A bar goes across the screen and fills in as each subtest completes. After about 3 minutes (or more, depending on the options installed), a system configuration list displays on the screen.

In addition to more extensive tests, the extended self-test diagnostic performs the same series of tests as the powerup test. The extended self-test diagnostic handles errors in the same manner as the powerup diagnostics. Included in the extended self-test are tests for protected memory and the 80287 math coprocessor option (if present).

Some video failures do not allow the failure value to be written to the screen. If the monitor board fails, error reports are not displayed. When the tests complete, the video display may be absent or distorted. If the system has a third-party video card installed, the diagnostic bypasses all video tests.

After the video test, the self-test performs extensive internal loopback tests on the printer, communications, and mouse serial ports. Loopback connectors are not required. Following the serial port test, the firmware tests the real time clock.

Then, the firmware tests the diskette controller and drive. A double-sided, high-density formatted diskette is required for this test. (The test does not write on the diskette.) The self-test also reads the hard disk (if present) and checks any other DIGITAL options in the system.

To allow the system to recognize a newly installed option, execute the extended self-test. When the extended self-test completes, the firmware diplays the configuration list. If the newly installed option is one of those shown in Figure 14-5, the newly installed option should be displayed in the system configuration list.

After the self-test completes its subtests, memory is sized and the hardware is initialized. The firmware updates the CMOS configuration to the new system configuration and displays the system configuration list on the screen.

Configuration List

On completion of the self-test, the firmware diagnostic displays the system configuration list on the screen. The user can check the amount of memory available, the options installed, the keyboard version, battery backup (if available), the ROM date, and the Ethernet address. See Figure 14-5 for an example of a typical configuration. Because the ROM diagnostics do not include multinational translation tables, the configuration screen uses numeric values and option codes. If an option is not present, the position for that option shows a dashed line.

After checking the configuration list, the user presses any key to continue. Then the firmware passes control to the ROM BIOS, which initializes the firmware data and tries to load an operating system.

> **NOTE**
> If a third-party video board is installed, the configuration list may not be displayed. Thus, there may be no indication that the user must press a key to continue.

Configuration list	Explanation (does not appear on screen)
1024 Kb	Standard memory
2048 Kb	Optional memory
1 RX	Diskette drive
1 RD	Hard-disk drive option
Li	Lithium battery
BA500	Expansion box
80287	Math coprocessor option
PC50X-MA	Modem option
VSXXX-AA	Mouse
LK250 01	Keyboard version number
27256 MO/DA/YR	ROM and revision date
08-00-2B-02-78-78	Ethernet address

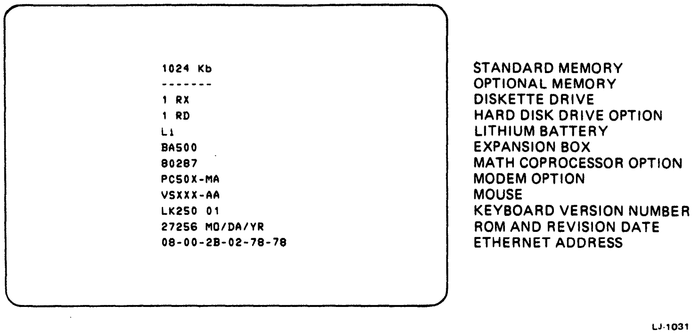

Figure 14-5 VAXmate Configuration Screen

Soft Reset

A soft reset, performed by pressing the Ctrl/Alt/Del key sequence, goes directly to the diagnostic initialization procedure. This initializes the hardware, sizes memory without initializing it, gets the status words, and sets up the CMOS RAM. (If the checksum is valid, the CMOS is not changed.) Then, the diagnostic passes control to the ROM BIOS initialization procedure. A soft reset does not display the configuration screen. The ROM BIOS tries to load the operating system from a diskette, from the hard disk (if present), or from the network.

Hard Reset

A hard reset, such as returning from virtual protected mode to real mode, resets only the processor. The firmware determines the reason for the reset by reading the shutdown byte, location 0FH, in the CMOS RAM. For example:

Shutdown Byte	Meaning
00H-03H	Execute diagnostic tests.
04H	CPU is retuning from CPU with operating system load request (INT 19H).
05H	Initialize the interrupt controller and begin execution at the specified address. The specified address is contained in two words, 0040:0067H for the instruction pointer and 0040:0069H for the code segment. These are industry-standard reserved locations.
09H	CPU is returning from a block move shutdown (used only by ROM BIOS INT 15H).
0AH	Begin execution at the specified address. The specified address is contained in two words, 0040:0067H for the instruction pointer and 0040:0069H for the code segment. These are industry-standard reserved locations. The interrupt controller is not initialized.
0BH-FFH	Execute diagnostic tests.

Examples of subprograms that can cause a hard reset are:

- Memory sizing routine
- Memory initialization routine
- Reset processor test
- Testing memory with physical addresses above 1 Mbyte
- MDrive utility
- Move block ROM BIOS call
- Third-party software

Hardware Jumper Configuration

The processor board has three jumpers to enable testing in one of three modes:

- Test hardware including diskette drive (factory configuration)
- Test hardware with no diskette drive
- Manufacturing mode (no diskette drive or keyboard)

Table 14-2 shows the jumper usage. Figure 14-6 shows the factory configuration of the jumpers on the processor board. This configuration tests the system with a diskette drive, a video monitor, and a keyboard.

Table 14-2 VAXmate Processor Board Jumpers

System/Mode	W1	W2	W3
Factory configuration	IN	OUT	IN
System without diskette drive	IN	OUT	OUT
Manufacturing mode	IN	IN	IN

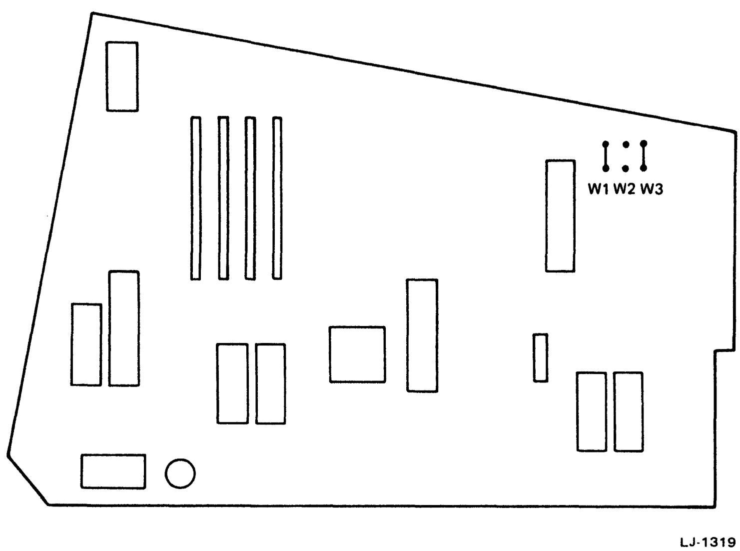

Figure 14-6 VAXmate Processor Board Jumper Configuration

Chapter 15
ROM BIOS

This chapter describes the interrupt services provided by the ROM BIOS. Table 15-1 lists, by hexadecimal value, all ROM BIOS interrupts. Function arguments, register use, and return values are described for each interrupt. Some functions use the CPU CARRY flag or the CPU ZERO flag as return values. Throughout this chapter, CF indicates the carry flag, and ZF indicates the zero flag.

Table 15-1 ROM BIOS Interrupt Vectors

INT	Usage	Description	Industry-Standard
02H	Hardware	Nonmaskable interrupt	Yes
05H	Software	Print screen function	Yes
08H	Hardware	Timer interrupt service	Yes
09H	Hardware	Keyboard interrupt service	Yes
0BH	Hardware	Serial port #2 interrupt service (modem option)	Yes
0CH	Hardware	Serial port #1 interrupt service (asynchronous)	Yes
0EH	Hardware	Diskette interrupt service	Yes
10H	Software	Video I/O	Yes
11H	Software	Return configuration	Yes
12H	Software	Return memory size	Yes
13H	Software	Diskette and Hard Disk I/O	Yes

Table 15-1 ROM BIOS Interrupt Vectors (cont.)

INT	Usage	Description	Industry-Standard
14H	Software	Asynchronous Communications I/O	Yes
15H	Software	Cassette I/O (Multitasking hooks)	Yes
16H	Software	Keyboard I/O	Yes
17H	Software	Printer output	Yes
18H	Software	Invoke network boot/Maintenance Operations Protocol (MOP)	No
19H	Software	Bootstrap	Yes
1AH	Software	Time of day	Yes
1BH	Software	Keyboard BREAK	Yes
1CH	Software	Timer tick vector	Yes
1DH	Pointer	Video parameter table	Yes
1EH	Pointer	Diskette parameter table	Yes
1FH	Pointer	Graphic mode character table (character codes 80H-FFH)	Yes
40H	Software	Interrupt 13H redirect when hard disk in use	Yes
41H	Pointer	Parameter table for hard disk 0	Yes
46H	Pointer	Parameter table for hard disk 1	Yes
4AH	Software	Real-time clock alarm	Yes
70H	Hardware	Real-time clock interrupt (IRQ8)	Yes
71H	Hardware	Redirect to interrupt 0AH - Old IRQ2 (IRQ9)	Yes
72H	Hardware	Ethernet controller (IRQ10)	No
73H	Hardware	Serial printer port (IRQ11)	No
74H	Hardware	Mouse port (IRQ12)	No
75H	Hardware	80287 error (IRQ13)	Yes
76H	Hardware	Hard disk controller (IRQ14)	Yes
77H	Hardware	Available (IRQ15)	Yes

Interrupt 02H: Nonmaskable Interrupt

Hardware Interrupt - Industry-Standard

Interrupt 02H handles the nonmaskable interrupt (NMI). An NMI is generated for either of two catastrophic events:

- Memory parity errors
- Input/output (I/O) bus parity errors

Interrupt 02H has no arguments, preserves all registers, and returns no values.

To process these inputs to the NMI, the following conditions must exist:

- The nonmaskable interrupt is so named because the 80286 CPU has no provisions for disabling the NMI. The VAXmate workstation provides for disconnecting the inputs to the NMI input line using the NMI mask register. The NMI mask register, a write only register, is accessed by writing bit 7 at I/O address 0070H. When bit 7 is 0, NMI inputs are enabled. This is the default condition after system startup. The I/O address 0070H is also used to access the real-time clock. For information about the real-time clock, see Chapter 5.
- Memory parity checking must be enabled. It is controlled by bit 2 at I/O address 0061H. When bit 2 is 0, memory parity checking is enabled. This is the default condition after system startup.

  Memory errors are confirmed by reading bit 7 of I/O address 0062H. When bit 7 is set (1), a memory error has occurred. To clear the error indication (bit 7), disable and reenable memory parity checking.
- I/O bus checking must be enabled. It is controlled by bit 3 at I/O address 0061H. When bit 3 is 0, I/O bus checking is enabled. This is the default condition after system startup.

  I/O bus errors are confirmed by reading bit 6 of I/O address 0062H. When bit 6 is set (1), an I/O bus error has occurred. To clear the error indication (bit 6), disable and reenable I/O bus checking.

Interrupt 05H: Print Screen

Software Interrupt - Industry-Standard

Interrupt 05H reproduces the ASCII characters, displayed on a video monitor, by printing them on the LPT1 printer. Either program execution (INT 05H) or keyboard interaction (pressing the Shift and Prt Sc keys) activates the printer.

Interrupt 05H has no arguments, preserves all registers, and returns no values.

In text mode (see Interrupt 10H), the character codes are sent to the printer. In graphic mode (see Interrupt 10H), interrupt 05H interprets the pixel pattern at each character location in the video display memory. If the interpretation produces a valid character code, it is sent to the printer.

The ROM BIOS does not support interpretation of the pixel pattern in graphic mode D2H (see Interrupt 10H).

> NOTE
> The user can execute the MS-DOS external command GRAPHICS. This terminate-and-stay-resident program takes over interrupt 05H. In graphic mode (including mode D2H), it reproduces pixel graphics at the printer. In text mode, it calls the ROM BIOS.
>
> If the printer is not using the same character set as the display, the printer incorrectly reproduces the screen.
>
> The output to LPT1 can be redirected to other printers, including a network printer.

Interrupt 08H: Clock Tick

Hardware Interrupt - Industry-Standard with DIGITAL Extensions

Interrupt 08H provides the ROM BIOS with hardware-interrupt services for the 8254-2 CLOCK1 output. CLOCK1 interrupts 18.206482 times per second, which is 1573040 times in a 24 hour period. At each CLOCK1 output, interrupt 08H maintains several internal counters, and then provides an application timing service by executing an INT 1CH instruction. Interrupt 08H has no arguments, preserves all registers, and returns no values.

Do not take over interrupt 08H to acquire clock services. Using this interrupt requires knowledge of the VAXmate workstation hardware, the ROM BIOS, and the operating system. If an application requires clock services, use the interrupt vector at 1CH (see Interrupt 1CH). Review functions 35H and 25H of MS-DOS interrupt 21H for the proper method to get and set interrupt vectors.

Interrupt 09H: Keyboard

Hardware Interrupt - Industry-Standard with DIGITAL Extensions

Interrupt 09H provides the ROM BIOS with hardware-interrupt services for the keyboard-interface controller. This interrupt service monitors the state of the keyboard-interface controller, reads scan codes from the keyboard-interface controller, and maintains the state of the keyboard LEDs. After reading a scan code, the interrupt service may translate a scan code or a combination of scan codes. Also, the interrupt service reacts to certain scan code combinations such as Ctrl/Alt/Del.

Interrupt 09H has no arguments, preserves all registers, and returns no values.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. The keyboard-interface controller is described in Chapter 8. For information about the ROM BIOS keyboard input service, see Interrupt 16H.

Interrupt 0BH: COM2 / Modem

Hardware Interrupt - Industry-Standard with DIGITAL Extensions

Interrupt 0BH provides the ROM BIOS with hardware-interrupt services for the optional integral modem or any asynchronous serial communications option that is configured as COM2. This interrupt service monitors the state of the serial communications protocol and line status. It also transmits and receives characters as required.

Interrupt 0BH has no arguments, preserves all registers, and returns no values.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. For information about the 8250A serial communications device, see Chapter 9. For more information about the ROM BIOS asynchronous communications service, see Interrupt 14H.

The integral modem is an optional device for the VAXmate workstation. For information about the integral modem, see the *Modem User's Guide.*

Interrupt 0CH: COM1 / Serial

Hardware Interrupt - Industry-Standard with DIGITAL Extensions

Interrupt 0CH provides the ROM BIOS with hardware-interrupt services for COM1 asynchronous serial communications port. This interrupt service monitors the state of the serial communications protocol and the line status. It also transmits and receives characters as required.

Interrupt 0CH has no arguments, preserves all registers, and returns no values.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. For information about the 8250A serial communications device, see Chapter 9. For information about the ROM BIOS asynchronous communications service, see Interrupt 14H.

Interrupt 0EH: Floppy Disk

Hardware Interrupt - Industry-Standard

Interrupt 0EH provides the ROM BIOS with hardware-interrupt services for the diskette drive controller. This interrupt service provides a *operation complete* indication from the diskette drive controller.

Interrupt 0EH has no arguments, preserves all registers, and returns no values.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. For information about the diskette drive controller, see Chapter 11. For more information about the ROM BIOS diskette I/O service, see Interrupts 13H, 40H, and 1EH.

Interrupt 10H: Video Input/Output

Software Interrupt - Industry-Standard with DIGITAL Extensions

Interrupt 10H provides access to several video management and display functions. These functions support the VAXmate graphic video system, industry-standard color graphic, and monochrome adapters.

> **NOTE**
> Although the VAXmate workstation supports color graphics, it comes with a monochrome graphics monitor. Colors are displayed as shades of gray or intensity levels.
>
> Use of an industry-standard color graphic adapter or a monochrome adapter requires an external monitor.

When using interrupt 10H functions, the following rules apply:

- In general, there are no validity tests performed on interrupt 10H arguments. Invalid functions or function arguments can destroy data or cause unpredictable results. The validity of arguments depends on the video mode in effect at the time.
- The VAXmate graphic video system has a feature that reduces image burning on the video monitor. When there has been no keyboard input and no video output for 30 minutes, the video output is disabled. Execution of any interrupt 10H function or pressing any key on the keyboard enables video output and initializes the screen-blanking counter to 30 minutes. Also, reading or writing the video RAM enables the video output (but not the screen-blanking counter).
- All graphic text operations are based on an 8 x 8 character cell size.
- The video hardware can operate in a graphic mode of 800 x 252 x 4 colors, which emulates DIGITAL terminals. This video mode (D2H), has the following limited ROM BIOS support:
  - Function 00H: Set the video mode
  - Function 0FH: Return the video state
  - Function D1H: Font RAM and color mapping support (Color mapping only)

  These are the only functions supported for video mode D2H. The display of graphics or graphic text must be accomplished directly by the application. For detailed information on direct programming of the VAXmate graphic video system, see Chapter 7.

The value in the AH register specifies the desired function. Most functions require additional information in other registers. The function-specific register usage is defined within the description of each function. Table 15-2 lists the available functions.

Table 15-2 Interrupt 10H: Video I/O Functions

Function	Description	DIGITAL Extended
AH = 00H	Set video mode	Yes
AH = 01H	Set cursor type	No
AH = 02H	Set cursor position	No
AH = 03H	Read cursor position	No
AH = 04H	Read light-pen position	No
AH = 05H	Select display page	No
AH = 06H	Scroll active page up	No
AH = 07H	Scroll active page down	No
AH = 08H	Read character and attribute at current cursor position	No
AH = 09H	Write character and attribute at current cursor position	No
AH = 0AH	Write character at current cursor position	No
AH = 0BH	Set color palette	No
AH = 0CH	Write pixel	No
AH = 0DH	Read pixel	No
AH = 0EH	TTY write character	No
AH = 0FH	Read current video state	Yes
AH = 10H	Reserved	No
AH = 11H	Reserved	No
AH = 12H	Reserved	No
AH = 13H	TTY write string	No
AH = D0H	Enable/disable 256 character graphic fonts	Yes
AH = D1H	Font RAM and color map support	Yes

Function 00H: Set Video Mode

Industry-Standard with DIGITAL Extensions

Parameters

AH = 00H
AL = One of the AL values listed in Table 15-3

Returns

Nothing

Table 15-3 list the video modes supported by the ROM BIOS.

Table 15-3 Video Modes

AL	Description
00H	40 X 25 monochrome text *
01H	40 X 25 color text *
02H	80 X 25 monochrome text *
03H	80 X 25 color text *
04H	320 X 200 X 4 color graphic
05H	320 X 200 monochrome graphic
06H	640 X 200 monochrome graphic
07H	80 X 25 monochrome (requires monochrome adapter)
D0H	640 X 400 X 2 color DIGITAL extended graphics
D1H	640 X 400 X 4 color DIGITAL extended graphics
D2H	800 X 252 X 4 color DIGITAL extended graphics

* In modes 0 and 2, monochrome means lack of a color burst signal at the composite video connector of an industry-standard color graphics adapter. This is the opposite of modes 1 and 3, which do produce a color burst signal at the composite video connector of an industry-standard color graphics adapter.

For the VAXmate workstation, there is no difference between modes 0 and 1 or between modes 2 and 3.

This function selects the video mode. Selecting a video mode configures the video controller and clears the display.

Video mode 07H is only valid when an industry-standard monochrome adapter is installed in an expansion slot. An industry-standard monochrome or color graphics adapter installed in an expansion slot is recognized by the ROM BIOS during the power-up sequence. On finding an industry-standard monochrome or color graphic adapter, the ROM BIOS disables the VAXmate graphic video system with the following consequences:

- If an industry-standard monochrome adapter is installed, video mode 07H becomes the only valid video mode. Attempts to use any other mode are forced to video mode 07H.
- If an industry-standard color graphic adapter is installed, video modes 07H, D0H, D1H, and D2H are not valid modes.

Changing between an industry-standard mode and a DIGITAL extended mode resets the color map to values appropriate for the mode. See function D1H. Changing from one industry-standard mode to another industry-standard mode does not affect the color map. Also, changing from one DIGITAL extended mode to another DIGITAL extended mode does not affect the color map.

For detailed information on direct programming of the VAXmate graphic video system, see Chapter 7.

Function 01H: Set Cursor Type

Industry-Standard

Parameters

AH = 01H
CH = The cursor start scan line
CL = The cursor end scan line

Returns

Nothing

Table 15-4 lists the allowed values for each of the ROM BIOS supported video modes.

Table 15-4 Mode-Dependent Values for Set Cursor Type

Modes	Reg	Range	Comments
00H, 01H, 02H, 03H			Text mode
	CH	00H-07H	Cursor start scan line
	CL	00H-07H	Cursor end scan line
07H			Monochrome text
	CH	00H-0DH	Cursor start scan line
	CL	00H-0DH	Cursor end scan line
05H, 06H, D0H, D1H, D2H			No cursor in graphic modes
	CH		Ignored
	CL		Ignored

This function selects, within the character cell, the size and placement of the cursor. When this function executes, it checks the current video mode. If the current video mode is one of the graphic modes, the initialization sequence is ignored.

The VAXmate graphic video system character cell height is 16 scan lines, but the allowable arguments are limited to the range 0-7. To maintain compatibility, the start value is multiplied by two, and the end value is multiplied by two and incremented.

A start or end value greater than 7 (13 for the monochrome adapter) disables the cursor. Also, an end value that is less than the start value disables the cursor. That is, the cursor becomes invisible. Although the cursor is invisible, subsequent commands that change the cursor position continue to be effective.

Function 02H: Set Cursor Position

Industry-Standard

Parameters

AH = 02H
DH = The row position
DL = The column position
BH = The page number

Returns

Nothing

This function sets the logical cursor position for any display page. Because the ROM BIOS maintains a logical cursor position for each display page, it is possible to change the cursor position for a display page that is not active. When the display page becomes active, the cursor is moved to the new position. If the indicated display page is the active page, the cursor is moved to the new position.

The unit of measurement is one character cell.

Graphic modes have the following limitations:

- The display page must be page zero.
- No cursor is displayed, but the cursor position is maintained.
- This function is not supported for graphic mode D2H.

Function 03H: Read Cursor Position

Industry-Standard

Parameters

AH = 03H
BH = The page number

Returns

CH = The cursor start scan line
CL = The cursor end scan line
DH = The row position
DL = The column position

This function returns the cursor position of the indicated display page and the current cursor type. It does not support video mode D2H.

Function 04H: Read Light-Pen Position

Industry-Standard

Parameters

AH = 04H	

Returns

AH = 00H	No input or the switch is not closed
AH = 01H	The light pen read BX = The pixel column (0-319 or 0-639) CH = The pixel scan line (0-199) DH = The row character position DL = The column character position

This function returns the position of the light pen.

To read a light-pen position successfully, the following conditions must exist:

- A video adapter that supports light pens must be installed.
- The read switch on the light pen must be closed.
- The light pen must have detected an input signal.

NOTE
The VAXmate graphic video system does not support light pens. When the VAXmate graphic video system is in use, the returned AH register contains zero.

Function 05H: Set Page Function

Industry-Standard

Parameters

AH = 05H
AL = The page number

Returns

Nothing

This function selects the active display page. The page is displayed and the cursor is positioned according to the cursor position for that page.

Valid page numbers depend on the video mode in effect.

Mode	Range
00H	00H through 07H
01H	00H through 07H
02H	00H through 03H
03H	00H through 03H
04H	Function call ignored
05H	Function call ignored
06H	Function call ignored
07H	Function call ignored
D0H	Function call ignored
D1H	Function call ignored
D2H	Function call ignored

Function 06H: Scroll Active Page Up
Function 07H: Scroll Active Page Down

Parameters

AH = 06H	Scroll Up
	AL = The number of rows (0 means blank the window) BH = The attribute byte CH = The row of upper-left corner of scroll window CL = The column of upper-left corner of scroll window DH = The row of lower-right corner of scroll window DL = The column of lower-right corner of scroll window
AH = 07H	Scroll Down
	AL = The number of rows (0 means blank the window) BH = The attribute byte CH = The row of upper-left corner of scroll window CL = The column of upper-left corner of scroll window DH = The row of lower-right corner of scroll window DL = The column of lower-right corner of scroll window

Returns

Nothing

These functions scroll data within a window on the screen. They work in text and graphic modes, but do not support video mode D2H.

For these functions, the AL, CH, CL, DH, and DL register values always refer to character positions. In graphic modes, the graphic data within a character cell area is treated as a single unit. The AL, CH, and DH registers contain character row values in the range 0 through 24. The CL and DL registers contain character column values in the range 0 through 39 or 0 through 79.

The scroll area is a window or rectangular area defined by two diagonal points. The two points are defined by the contents of the CX and DX registers. The CX register (CH and CL) defines the upper-left corner of the window. The DX register (DH and DL) defines the lower-right corner of the window.

The scroll up operation moves the rows one at a time so that row CH + 1 moves to row CH, row CH + 2 moves to row CH + 1, and so on. The scroll down operation moves the rows one at a time so that row DH - 1 moves to row DH, row DH - 2 moves to row DH - 1, and so on. When the last row is vacated, it is blanked. This process repeats until the specified number of rows are scrolled. If the specified number of rows is greater than or equal to the vertical size of the window, the entire window is cleared. Also, if the AL register equals 0, the entire window is cleared.

The contents of the CL and DL registers determine the horizontal position and width of the scrolled area. For example, if the CL register contains 20 and the DL register contains 40, only those columns and the data between them is scrolled.

When using this function, the key difference between text and graphic modes is the way the attribute byte is applied. In text modes, a space character is written to the data byte and the contents of the BH register are written to the attribute byte. In graphic modes, the contents of the BH register are written as graphic data one byte at a time. Thus, if the current graphic mode requires two bits of information for each pixel, the BH register must contain data for four pixels.

Only the active display page can be scrolled. Data scrolled out of the window is lost. It does not go into the adjacent page.

The cursor position remains the same after scrolling as it was before scrolling.

This function is not supported for video mode D2H.

Function 08H: Read Character and Attribute at Cursor Position

Industry-Standard

Parameters

AH = 08H
BH = The page number (text modes only)

Returns

AL = The character
AH = The attribute (text modes only)

In text modes, this function returns the character and attribute at the cursor location of the specified display page. A page other than the active display page can be specified. The cursor location, character, and attribute are extracted from the indicated display page data.

In graphic modes, there is only one page, so the page selection is ignored. Because there is no attribute byte, only the character value is returned. This is accomplished similarly to the interrupt 05H interpretation of graphic text. The ROM BIOS attempts to interpret the pixel pattern in the character cell at the current cursor location. The pixel pattern is matched with the bit patterns of the characters used for graphic text. Interrupt 1FH (0000:007CH) contains a pointer to the graphic text character set used in the comparison. If no match is found, the function returns a character value of 0.

Normally, interrupt 1FH points to a table containing 128 entries in the range 80H-FFH. However, interrupt 10H function D0H provides an extended mode where interrupt 1FH points to a table containing 256 entries in the range 00H-FFH.

This function is not supported for video mode D2H.

Function 09H: Write Character and Attribute at Cursor Position

Industry-Standard

Parameters

AH = 09H
AL = The character
BH = The page number (text modes only)
BL = The attribute byte (text modes) or color (graphic modes)

> If bit 7 is set (1), exclusive OR the current contents with the contents of BL and store the result. Normally, the contents of BL are stored.

CX = Number of times to write character and attribute

Returns

Nothing

This function writes a character and attribute at the current cursor position. The current cursor position is extracted from the page data of the page specified in register BH. The position of the cursor is not updated. That is, the cursor remains as it was when the function was called. Register CX specifies the number of times to repeat the operation. Each repetition advances the position one character location. A line wrap occurs at the end of a line. Counts that exceed the page size continue into the adjacent display page (if one exists).

In graphic modes, the character code in AL is an index into a table of graphic characters. If 256 character mode is not enabled, and the character code is less than 80H, the information is retrieved from the ROM. If 256 character mode is not enabled, and the character code is 80H or greater, the information is retrieved from the table pointed to by interrupt vector 1FH. If 256 character mode is enabled, interrupt vector 1FH points to the beginning of the entire 256 character table. For each pixel that is on in the pattern, the color selection in register BL is shifted into position and written to display memory. The number of bits used from register BL is mode dependent. For example, a 4-color mode uses the two least significant bits, and a 2-color mode uses the least significant bit. If bit 7 of the BL register is set to 1, each display memory field is exclusive *OR*ed with the color field in register BL, and the result is written to display memory.

This function is not supported for video mode D2H.

Function 0AH: Write Character at Cursor Position

Industry-Standard

Parameters

AH = 0AH
AL = The character
BH = The page number (text modes only)
BL = The color (graphic modes only)

If bit 7 is set (1), exclusive OR the current contents with the contents of BL and store the result. Normally the contents of BL are stored.

CX = Number of times to write character

Returns

Nothing

This function writes a character at the current cursor position. It is similar to function 09H except that in text modes, a new attribute is not written. The current cursor position is extracted from the page data of the page as specified in register BH. The position of the cursor is not updated. That is, the cursor remains as it was when the function was called. Register CX specifies the number of times to repeat the operation. Each repetition advances the position one character location. A line wrap occurs at the end of a line. Counts that exceed the page size continue into the adjacent display page (if one exists).

In graphic modes, the character code in AL is an index into a table of graphic characters. If 256 character mode is not enabled, and the character code is less than 80H, the information is retrieved from the ROM. If 256 character mode is not enabled, and the character code is 80H or greater, the information is retrieved from the table pointed to by interrupt vector 1FH. If 256 character mode is enabled, interrupt vector 1FH points to the beginning of the entire 256 character table. For each pixel that is on in the pattern, the color selection in register BL is shifted into position and written to display memory. The number of bits used from register BL is mode dependent. For example, a 4-color mode uses the two least significant bits. If bit 7 of the BL register is set to 1, the current bits of the specified pixel field are exclusive *OR*ed with the appropriate BL bits, and the results are written to the specified pixel field. Otherwise, the appropriate BL bits replace the current bits of the specified pixel field.

This function is not supported for video mode D2H.

Function 0BH: Set Color Palette

Industry-Standard

Parameters

AH = 0BH	
BH = 00H	BL bits 4-0 control the background color and the palette intensity bit
BH = 01H	BL bit 0 selects the color palette
BL = The background color or palette, depending on contents of BH	

Returns

Nothing

This function is not supported for video mode D2H.

If register BH equals 0, this function controls the palette intensity bit and, depending on the current video mode, sets the background or border color. If current video mode is 01H or 03H, it sets the border color (the VAXmate border color is always black.) Otherwise, it sets the background color.
In either case, bits 4-0 of the BL register are interpreted as follows:

BL Bit	Description
4	Intensity control of the palette colors
3	(I) Intensity control of the color
2	(R) Red contribution to the color
1	(G) Green contribution to the color
0	(B) Blue contribution to the color

> **NOTE**
> The VAXmate graphic video system does not control the border color. The border color is always black.

If register BH equals one, this function selects the color palette for graphic video mode 04H.

Pixel Field Value	Palette 0 Selected	Palette 1 Selected
01H	Green	Cyan
02H	Red	Magenta
03H	Yellow	White

Function 0CH: Write Pixel

Industry-Standard

Parameters

AH = 0CH
AL = The color value

If bit 7 is set (1), exclusive OR the current contents with the contents of AL and store the result. Normally the contents of AL are stored.

CX = The pixel column number
DX = The pixel row number

Returns

Nothing

This function sets a pixel field, specified by registers CX and DX, to the color specified in register AL. This function is ignored in text modes.

The bits used from register AL depend on the current graphic mode:

Graphic mode	Bits used	Number of colors
04H	7 and 1-0	4
05H	7 and 0	Monochrome
06H	7 and 0	Monochrome
D0H	7 and 0	Monochrome
D1H	7 and 1-0	4

If bit 7 of the AL register is set to 1, the current bits of the specified pixel field are exclusive *OR*ed with the appropriate AL bits, and the results are written to the specified pixel field. Otherwise, the appropriate AL bits replace the current bits of the specified pixel field.

This function is not supported for video mode D2H.

Function 0DH: Read Pixel

Industry-Standard

Parameters

AH = 0DH	
CX = The pixel column number	
DX = The pixel row number	

Returns

AL = The color value of the pixel	

This function returns the color of the pixel field specified by registers CX and DX.

The valid bits returned in the AL register depend on the current graphic mode.

Graphic mode	Bits used	Number of colors
04H	1-0	4
05H	0	Monochrome
06H	0	Monochrome
D0H	0	Monochrome
D1H	1-0	4

This function is not supported for video mode D2H.

Function 0EH: Write Character Using Terminal Emulation

Industry-Standard

Parameters

AH = 0EH
AL = The character
BL = The foreground color (graphic mode only)

Returns

Nothing

This function is sometimes known as Write TTY. It operates in text and graphics modes and accesses only the active display page.

Prior to any other operations, the character in AL is tested for one of four values:

- If the character is a carriage return (0DH), the cursor is moved to the start of the current line.
- If the character is backspace (08H), the cursor is moved backward one character position. If the cursor is at the beginning of the line, the character is ignored.
- If the character is a line feed (0AH), the cursor is moved to the same column position on the next line. If the cursor is on the last line, the screen is scrolled up one line. In this case, the cursor remains in the same location.
- If the character is a bell character (07H), a bell sound (beep) is issued from the speaker.

For all other values, the character is written to the current cursor position and the cursor is advanced to the next position in the line. If the cursor was at the last position on the line, it is positioned at the first location on the next line. If the cursor was at the last position on the last line, the screen is scrolled up one line, and the cursor is positioned at the start of an empty line.

In graphic modes, the character code in AL is an index into a table of graphic characters. If 256 character mode is not enabled, and the character code is less than 80H, the information is retrieved from the ROM. If 256 character mode is not enabled, and the character code is 80H or greater, the information is retrieved from the table pointed to by interrupt vector 1FH. If 256 character mode is enabled, interrupt vector 1FH points to the beginning of the entire 256 character table. For each pixel that is on in the pattern, the color selection in register BL is shifted into position and written to display memory. The number of bits used from register BL is mode dependent. For example, a 4-color mode uses the two least significant bits. If bit 7 of the BL register is set to 1, the current bits of the specified pixel field are exclusive *OR*ed with the appropriate BL bits, and the results are written to the specified pixel field. Otherwise, the appropriate BL bits replace the current bits of the specified pixel field.

This function is not supported for video mode D2H.

Function 0FH: Read Current Video State

Industry-Standard

Parameters

AH = 0FH

Returns

AL = The current video mode
AH = The number of columns
BH = The current page

This function returns the current state of the video system. In text and graphic modes, the value in the AH register is the width of the screen in character cells. The mode value returned in the AL register is defined as follows:

AL	Description
00H	40 x 25 monochrome text
01H	40 x 25 color text
02H	80 x 25 monochrome text
03H	80 x 25 color text
04H	320 x 200 x 4 color graphic
05H	320 x 200 monochrome graphic
06H	640 x 200 monochrome graphic
07H	80 x 25 monochrome (requires monochrome adapter)
D0H	640 x 400 x 2 color DIGITAL extended graphics
D1H	640 x 400 x 4 color DIGITAL extended graphics
D2H	800 x 252 x 4 color DIGITAL extended graphics

Function 13H: TTY Write String

Industry-Standard

Parameters

AH = 13H	
AL = 00H	The string pointed to by ES:BP is a set of contiguous character codes. The register BL contains the attribute that is applied as each character is written to the display page. The CX register specifies the number of characters to write. After the last character is written, the cursor is restored to the position it had before this function was executed.
AL = 01H	This subfunction is similar to AL = 0 except that the cursor is positioned after the last character in the string.
AL = 02H	The string pointed to by ES:BP is a set of contiguous byte pairs. Each byte pair contains a character code and an attribute. The first byte of the string is the character; the second is the attribute. The BL register is ignored. The CX register specifies the number of characters to write, not the length of the string. After the last character is written, the cursor is restored to the position it had before this function was executed.
AL = 03H	This subfunction is similar to AL = 2 except that the cursor is positioned after the last character in the string.

BH = Display the page to write

BL = The attribute (AL = 0 or AL = 1)

CX = The number of characters to write

DH = The row position of the first character

DL = The column position of the first character

ES:BP = The pointer to the start of the string to write

Returns

Nothing

This function writes a string of characters to the specified display page. It operates in text and graphics modes.

Prior to writing each character, it is tested for one of four values:

> **NOTE**
> Even though another page is designated in register BH, the following operations occur on the current display page.

- If the character is a carriage return (0DH), the cursor is moved to the start of the current line.
- If the character is backspace (08H), the cursor is moved backward one character position. If the cursor is at the beginning of the line, the character is ignored.
- If the character is a line feed (0AH), the cursor is moved to the same column position on the next line. If the cursor is on the last line, the screen is scrolled up one line. In this case, the cursor remains in the same location.
- If the character is a bell character (07H), a bell sound (beep) is issued from the speaker.

For all other values, the character is written to the current cursor position, and the cursor is advanced to the next position in the line. If the cursor was at the last position on the line, it is positioned at the first location on the next line. If the cursor was at the last position on the last line, the screen is scrolled up one line, and the cursor is positioned at the start of an empty line.

In graphic modes, the character code in AL is an index into a table of graphic characters. If 256 character mode is not enabled, and the character code is less than 80H, the information is retrieved from the ROM. If 256 character mode is not enabled, and the character code is 80H or greater, the information is retrieved from the table pointed to by interrupt vector 1FH. If 256 character mode is enabled, interrupt vector 1FH points to the beginning of the entire 256 character table. For each pixel that is on in the pattern, the color selection is shifted into position and written to display memory. The number of bits used is mode dependent. For example, a 4-color mode uses the two least significant bits. If bit 7 of the color selection is set to 1, the current bits of the specified pixel field are exclusive *OR*ed with the color selection bits, and the results are written to the specified pixel field. Otherwise, the appropriate BL bits replace the current bits of the specified pixel field.

This function is not supported for video mode D2H.

Function D0H: Enable/Disable 256 Character Graphic Font

DIGITAL Extension

Parameters

AH = D0H	
AL = 00H	Interrupt 1FH (0000:007CH) points to 128 graphic mode characters in the range 80H through FFH.
AL = 01H-FFH	Interrupt 1FH (0000:007CH) points to 256 graphic mode characters in the range 00H through FFH.

Returns

Nothing

Function D0H extends user-defined font tables. On power-up, the ROM BIOS accesses the ROM for character codes 00H through 7FH. The character codes 80H through FFH are accessed through interrupt 1FH (0000:007CH).

This function is not supported for video mode D2H.

Function D1H: Font RAM and Color Map Support

DIGITAL Extension

This function provides access to the extended hardware capabilities of the VAXmate graphic video system. Using this function, the font RAM or the color map can be read, written, or restored to the default condition.

Font RAM Functions

Parameters

AH	=	D1H	
AL	=	00H	The font RAM functions
CX	=	The number of character descriptions to transfer (0001H to 0100H)	
DL	=	The first character to transfer (00H to FFH)	
ES:BX	=	The pointer to the data buffer (at least CX * 16 bytes in size)	
DH	=	00H	Restore the defaults (ES:BX is ignored)
DH	=	01H	Copy the data at ES:BX to the font RAM
DH	=	02H	Copy the font RAM to the buffer at ES:BX

Returns

Nothing

In text modes only, the font RAM acts as a character generator ROM. This subfunction can restore the font RAM to default conditions. It can also read or write one or more sequential character descriptions in the font RAM.

This function is not available in graphic modes.

Each character description contains 16 bytes of data. Each byte of data represents a scan line in the character cell. The first byte of the character description is the top scan line (scan line 0) in the character cell. Within each byte, the most significant bit is the leftmost pixel. The character descriptions are arranged in order of increasing character code value. Reading 256 character descriptions from the font RAM requires a 4096 byte buffer.

Color Map Functions

Parameters

AH	=	D1H	
AL	=	01H	The color map functions
CX	=	The number of entries to transfer (01H to 10H)	
DL	=	The map address of the entry to transfer (00H to 0FH)	
ES:BX	=	The pointer to the data buffer (at least CX words in size)	
DH	=	00H	Restore the defaults (ES:BX is ignored)
DH	=	01H	Copy the data at ES:BX to the color map
DH	=	02H	Copy the color map to the buffer at ES:BX

Returns

Nothing

This subfunction can read or write one or more sequential values in the color map. Any of the 16 IRGB inputs can be mapped to any of the 16 outputs. The default condition is gray-scale outputs at power-up. The color map is a synonym for the video look-up table (VLT). For more information on the VLT, see Chapter 7.

The color map is arranged as 16 words of IRGB output data. Only the least significant 4 bits of data are output. When the video controller accesses video memory, the attributes or graphic data are used as an offset into the color map. The contents of that location in the color map are sent to the video output circuit. To calculate the offset accessed by any IRGB value, use the following bit values:

Bit value	Attribute
0	I (Intensity)
1	B (Blue)
2	G (Green)
3	R (Red)

Thus, an attribute of intensified red (IRGB = C0H) accesses location 09H of the 16 locations in the color map.

On power-up or system reset and when changing from a DIGITAL extended 4-color video mode (D1H or D2H) to an industry-standard video mode, the color map is initialized to the values in Table 15-5. The color map defined in Table 15-5 supports video modes 00H, 01H, 02H, 03H, 04H, 05H, 06H and D0H. When changing from any of these modes to video mode D1H or D2H, the color map is initialized to the values defined in Table 15-6.

Table 15-5 Default Color Map

Offset R G B I	Contents R G B I	Color	Intensity
0 0 0 0	0 0 0 0	Black	0
0 0 0 1	0 0 0 1	Gray	1
0 0 1 0	0 0 1 0	Blue	2
0 0 1 1	0 0 1 1	Light blue	3
0 1 0 0	0 1 0 0	Green	4
0 1 0 1	0 1 0 1	Light green	5
0 1 1 0	0 1 1 0	Cyan	6
0 1 1 1	1 1 1 0	White	14
1 0 0 0	1 0 0 0	Red	8
1 0 0 1	1 0 0 1	Light red	9
1 0 1 0	1 0 1 0	Magenta	10
1 0 1 1	1 0 1 1	Light magenta	11
1 1 0 0	1 1 0 0	Brown	12
1 1 0 1	1 1 0 1	Yellow	13
1 1 1 0	0 1 1 1	Light cyan	7
1 1 1 1	1 1 1 1	Intense white	15

Table 15-6 Color Map for Video Modes D1H and D2H

Offset R G B I	Contents R G B I	Color	Intensity
0 0 0 0	0 0 0 0	Black	0
0 0 0 1	1 0 0 0	Red	4
0 0 1 0	0 1 0 0	Green	8
0 0 1 1	0 1 1 1	Light cyan	7
0 1 0 0	Not Used		
0 1 0 1	Not Used		
0 1 1 0	Not Used		
0 1 1 1	Not Used		
1 0 0 0	Not Used		
1 0 0 1	Not Used		
1 0 1 0	Not Used		
1 0 1 1	Not Used		
1 1 0 0	Not Used		
1 1 0 1	Not Used		
1 1 1 0	Not Used		
1 1 1 1	Not Used		

Interrupt 11H: Read Configuration

Software Interrupt - Industry-Standard

Parameters

None

Returns

AX = Configuration data

15-14	This two bit field equals the number of parallel printer ports in the system. 00 = Zero parallel printer ports 01 = One parallel printer port 10 = Two parallel printer ports 11 = Three parallel printer ports
13	Unused
12	Game adapter 0 = Game adapter not installed 1 = Game adapter installed
11-9	This three-bit field equals the number of asynchronous serial ports in the system. The VAXmate workstation has an integral serial port (COM1) and reserves COM2 for the optional integral modem. The serial printer port is not included in this count. The maximum number supported is four. 000 = There are zero serial ports 001 = There is one serial port 010 = There are two serial ports 011 = There are three serial ports 100 = There are four serial ports
8	Unused
7-6	This two-bit field equals the number of diskette drives in the system minus one. This field is only valid when bit 0 equals 1. 00 = 1 diskette drive 01 = 2 diskette drives 10 = 3 diskette drives 11 = 4 diskette drives

5-4	Initial video mode (see Interrupt 10H) 00 = Unused 01 = 40 X 25 (Color Graphics Adapter) 10 = 80 X 25 (Color Graphics Adapter) 11 = 80 X 25 (Monochrome Adapter)
3-2	Unused
1	80287 0 = 80287 not installed 1 = 80287 installed
0	Diskette drive 0 = No diskette drives installed (bits 7-6 are invalid) 1 = At least 1 diskette drive installed (bits 7-6 are valid)

This function returns the system configuration information. If the expansion box and battery are present and the CMOS RAM has not lost power, the configuration data is extracted from the CMOS RAM. Otherwise, the configuration data is extracted from the power-up initialization data.

Additional configuration data is available through function D0H of interrupt 15H. This configuration data is specific to the VAXmate workstation.

Interrupt 12H: Return Memory Size

Software Interrupt - Industry-Standard

Parameters

None

Returns

AX = Memory size measured in 1K blocks

This interrupt returns the memory size as the number of contiguous 1K (1024) memory blocks. Only the low address memory (0000:0000H to 000B:FFFFH) is measured by this function. The VAXmate workstation always returns 640.

Interrupt 13H: Disk Input/Output (I/O)

Software Interrupt - Industry-Standard with DIGITAL Extensions

This interrupt provides a generalized disk I/O service for diskettes and hard disks. If a hard disk is not installed, interrupt 13H points to the diskette functions. If a hard disk is installed, interrupt 13H points to the hard disk functions, and interrupt 40H points to the diskette functions.

Bit 7 of the drive number distinguishes diskette and hard disk function requests. If bit 7 is set (1), the request is for a hard disk function. Thus, hard disks are assigned drive numbers equal to or greater than 80H. When a hard disk is installed, interrupt 13H compares the drive number to 80H. Requests with drive numbers less than 80H are revectored to interrupt 40H.

This revectoring information is provided only for clarity. Always use interrupt 13H for both diskette and hard disk functions.

> NOTE
> Most operating systems intercept and sometimes modify interrupt 13H requests. When developing or testing software, this fact is important. For example, several interrupt 13H functions warn against exceeding a physical page boundary during disk I/O. By translating a single I/O request into many small sized I/O requests, some operating systems eliminate page boundary problems.

The following is a list of the interrupt 13H hard disk functions:

Function Number	Description	DIGITAL Extended
00H	Initialize Entire Disk Subsystem	No
01H	Return Status Code Of Last I/O Request	No
02H	Read One Or More Disk Sectors	No
03H	Write One Or More Disk Sectors	No
04H	Verify One Or More Disk Sectors	No
05H	Format A Track	No
08H	Return Current Drive Parameters	No
09H	Initialize Drive Characteristics	No
0AH	Read Long	No
0BH	Write Long	No
0CH	Seek To Specific Cylinder	No
0DH	Hard Disk Reset	No
10H	Test Drive Ready	No
11H	Recalibrate Drive	No
14H	Execute Controller Internal Diagnostics	No
15H	Return Drive Type	No
D0H	Read Long 256 Byte Sector	Yes

The following is a list of the interrupt 13H diskette functions:

Function Number	Description	DIGITAL Extended
00H	Initialize Diskette Subsystem	No
01H	Return Status Code Of Last I/O Request	No
02H	Read One Or More Track Sectors	No
03H	Write One Or More Track Sectors	No
04H	Verify One Or More Track Sectors	No
05H	Format Track	No
15H	Return Drive Type	No
16H	Return Change Line Status	No
17H	Set Drive And Media Type For Format	No

Hard Disk Functions

The value in the AH register indicates the desired hard disk function. All hard disk functions require a drive number in the DL register. Because hard disk drive numbers start at 80H, hard disk 0 is 80H, and hard disk 1 is 81H.

Functions requiring a cylinder number expect a 10-bit value in the range of 0 to 1023. The low-order eight bits of the cylinder number are passed in the CH register. The two high-order bits of the cylinder number are passed in the two high-order bits of the CL register. At times, bits 4-0 of the the CL register contain a sector number. Some functions require a cylinder and sector number.

Except for the flags register, all registers not mentioned in the function description are preserved.

Hard Disk Errors

If CF is set (1), an error occurred, and the AH register contains the error code. Table 15-7 lists the hard disk error codes.

Table 15-7 Hard Disk Error Codes

Error Code	Description
FFH	Sense operation failed (not implemented)
E0H	Status error (error register = 0)
CCH	Write fault on selected drive
BBH	Undefined error occurred
AAH	Drive not ready
80H	Hardware failed to respond
40H	Seek operation failed
20H	Disk controller failed
11H	ECC corrected data error The ECC algorithm corrected a recoverable error. The data is probably valid, however the calling program must make that decision.
10H	ECC for data incorrect
0BH	Bad track flag detected (not implemented)
0AH	Bad sector flag detected
09H	Data extends too far (past 64K page boundary)
07H	Drive parameter activity failed
05H	Reset failed
04H	Sector not found
02H	Address mark not found
01H	Illegal I/O request (bad command)

Hard Disk Parameter Tables

A hard disk parameter table defines the physical characteristics of a hard disk. The values in the table are used by the hard disk driver to initialize the hard disk controller. Table 15-8 describes the contents of a hard disk parameter table.

Table 15-8 Hard Disk Parameter Table Description

Offset	Size	Description
00H	1 Word	Maximum number of cylinders on hard disk drive
02H	1 Byte	Maximum number of heads on hard disk drive
03H	1 Word	Not used
05H	1 Word	Cylinder number to start using write precompensation
07H	1 Byte	Not used
08H	1 Byte	Control byte sent to controller
		If bit 7 or bit 6 is set (1), disable retries If bit 3 is set (1), the hard disk has more than eight heads
09H	3 Bytes	Not used
0CH	1 Word	Landing zone
0EH	1 Byte	Number of sectors per track
0FH	1 Byte	Reserved for future use

The hard disk parameter tables are located in DIGITAL private RAM. During the power-up sequence, the disk type is extracted from CMOS RAM. If the disk type is unknown, the table contains all zeros. If the disk type is one of the 14 industry-standard types, the table is initialized from the hard disk data in the ROM BIOS. If the disk type is the DIGITAL extended type 0FH, the ROM BIOS expects the boot block to contain the parameters. The ROM BIOS initializes the table with data extracted from the boot block. (As part of its initialization process, the FDISK utility writes the parameters in the boot block.)

The interrupt vectors for interrupt 41H and 46H point to the hard disk parameter tables for hard disk 0 and hard disk 1, respectively. If hard disk 1 does not exist, the interrupt vector for interrupt 46H is reserved and undefined.

Function 00H: Initialize Entire Disk Subsystem

Industry-Standard

Parameters

AH = 00H	
DL = The drive number (80H or 81H)	

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function resets the diskette and hard disk controllers to their initial power-up state. The hard disk controller is initialized to the values in the hard disk parameter tables. Because all drives are marked as reset, the next drive specific I/O request recalibrates that drive.

To initialize only the hard disk controller, use hard disk function 0DH. To initialize only the diskette controller, use diskette function 00H.

Function 01H: Return Status Code of Last I/O Request

Industry-Standard

Parameters

AH = 01H
DL = The drive number (80H or 81H)

Returns

AH = 0
AL = The error code of the previous operation

This function returns, in the AL register, the error code of the last function call. If AL returns a 0, no previous error condition existed. Because calls to this function do not generate error conditions, successive calls return 0.

The AH register always returns 0.

For the hard disk error codes, see Table 15-7.

Function 02H: Read One or More Disk Sectors

Industry-Standard

Parameters

AH = 02H
AL = The number of sectors to read
CH = The cylinder number (lower 8 bits)
CL = The starting sector number (and bits 9-8 of cylinder)
DH = The head number
DL = The drive number (80H or 81H)
ES:BX = The buffer address

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function reads the indicated number of sectors and stores the data starting at the buffer address in ES:BX. Attempts to store data past a physical page boundary return an error. This can occur when the data size exceeds 10000H or when the BX offset plus the data size exceed 10000H.

To calculate the required buffer size, multiply the contents of the AL register by 512.

Function 03H: Write One Or More Disk Sectors

Industry-Standard

Parameters

AH = 03H
AL = The number of sectors to write
CH = The cylinder number (lower 8 bits)
CL = The starting sector number (and bits 9-8 of cylinder)
DH = The head number
DL = The drive number (80H or 81H)
ES:BX = The buffer address

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function writes the indicated number of sectors of data starting at the buffer address in ES:BX. Attempts to read data past a physical page boundary return an error. This can occur when the data size exceeds 10000H or when the BX offset plus the data size exceed 10000H.

To calculate the number of sectors in a buffer, divide the buffer size by 512. If the division produces a remainder, increment the sector count.

Function 04H: Verify One or More Disk Sectors

Industry-Standard

Parameters

AH = 04H
AL = The number of sectors to verify
CH = The cylinder number (lower 8 bits)
CL = The starting sector number (and bits 9-8 of cylinder)
DH = The head number
DL = The drive number (80H or 81H)

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition
	AH = The error code

This function verifies the indicated number of sectors. The data is not compared against data in memory. It is only verified for internal consistency. Thus, the verify command only checks for Error Correction Code (ECC) errors.

Function 05H: Format a Track

Industry-Standard

Parameters

AH = 05H
AL = The number of sectors per track
CH = The cylinder number (lower 8 bits)
CL = Bits 9-8 of cylinder number
DH = The head number
DL = The drive number (80H or 81H)
ES:BX = The sector interleave table address

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function formats the indicated track. It formats only the sectors described in the sector interleave table. The data field of the formatted sectors is initialized to zeros. Before formatting the track, the ROM BIOS initializes the controller to the values found in the hard disk parameter table.

ES:BX points to the sector interleave table, which contains an entry for each sector on the track. A table entry requires two bytes of data. Therefore, the expected buffer size is two times the number of sectors per track. The following list describes a single table entry:

Offset	Name	Description
00H	The sector status	00H = good sector 80H = bad sector
01H	The sector number	A sector number in the range of 1 to 17

Function 08H: Return Current Drive Parameters

Industry-Standard

Parameters

AH = 08H
DL = The drive number (80H or 81H)

Returns

CF = 0	Indicates a successful operation	
	DL =	The number of consecutive acknowledging drives
	DH =	The maximum usable head number for the requested drive
	CH =	Lower 8 bits of the maximum usable cylinder number for the requested drive
	CL =	The maximum usable sector number for the requested drive and two high bits of the cylinder number
CF = 1	Indicates an error condition	
	AH = The error code	

This function returns the number of consecutive, acknowledging, hard disk drives. For example, if the DL register contains 02H, hard disks 0 (80H) and 1 (81H) are present and respond to the controller.

If the function returns with CF set (1), only the AH register is valid and contains the error code.

The data returned in the DH, CH, and CL registers is only meaningful for the drive specified by the calling parameter in the DL register.

> **NOTE**
> If an invalid drive type has been specified for the selected drive or the selected drive is unformatted, this function can return invalid data. For the selected drive, check the parameter table. If the parameter table contains all zeros, the returned data is invalid. The interrupt vector at interrupt 41H points to the drive 0 parameter table. The interrupt vector at 46H points to the drive 1 parameter table.

Function 09H: Initialize Drive Characteristics

Industry-Standard

Parameters

AH = 09H
DL = The drive number (80H or 81H)

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function initializes the hard disk controller to the values in the appropriate hard disk parameter table. For drive 80H, the parameter table pointed to by interrupt 41H is used. For drive 81H, the parameter table pointed to by interrupt 46H is used.

Function 0AH: Read Long

Industry-Standard

Parameters

AH = 0AH
AL = The number of sectors to read
CH = The cylinder number (lower 8 bits)
CL = The starting sector number (and bits 9-8 of cylinder)
DH = The head number
DL = The drive number (80H or 81H)
ES:BX = The buffer address

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function is similar to function 02H, except that each sector of data is terminated by a 4-byte ECC field. This function reads the indicated number of sectors and stores the data starting at the buffer address in ES:BX. Attempts to store data past a physical page boundary return an error. This can occur when the data size exceeds 10000H or when the BX offset plus the data size exceed 10000H.

To calculate the required buffer size, multiply the contents of the AL register by 516. If the division produces a remainder, increment the sector count.

Function 0BH: Write Long

Industry-Standard

Parameters

AH = 0BH
AL = The number of sectors to write
CH = The cylinder number (lower 8 bits)
CL = The starting sector number (and bits 9-8 of cylinder)
DH = The head number
DL = The drive number (80H or 81H)
ES:BX = The buffer address

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function is similar to function 03H, except that each sector of data is terminated by a 4-byte ECC field. This function writes the indicated number of sectors of data starting at the buffer address in ES:BX. Attempts to read data past a physical page boundary return an error. This can occur when the data size exceeds 10000H or when the BX offset plus the data size exceed 10000H.

To calculate the number of sectors in a buffer, divide the buffer size by 516.

Function 0CH: Seek to Specific Cylinder

Industry-Standard

Parameters

AH = 0CH
CH = The cylinder number (lower 8 bits)
CL = Bits 9-8 of cylinder number
DH = The head number
DL = The drive number (80H or 81H)

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function positions the head of the selected drive. Illegal cylinder numbers produce an error, but no head movement occurs.

Before each invocation of this function, the target drive must be tested to determine if the drive is ready to accept another I/O command (see Interrupt 13H, Function 10H).

Function 0DH: Hard Disk Reset

Industry-Standard

Parameters

AH = 0DH	
DL = The drive number (80H or 81H)	

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function resets the hard disk controller to its initial power-up state. The hard disk controller is initialized to the values in the hard disk parameter tables. The diskette controller is not affected. Only the hard disk controller is reset.

Function 10H: Test Drive Ready

Industry-Standard

Parameters

AH = 10H
DL = The drive number (80H or 81H)

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

If this function does not return an error code, the hard disk is ready to accept I/O requests. For the hard disk error codes, see Table 15-7.

Function 11H: Recalibrate Drive

Industry-Standard

Parameters

AH = 11H	
DL = The drive number (80H or 81H)	

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition
	AH = The error code

This function moves the head of the selected hard disk to the home position (cylinder zero).

Function 14H: Execute Controller Internal Diagnostics

Industry-Standard

Parameters

AH = 14H
DL = The drive number (80H or 81H)

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition
	AH = The error code

This function performs a diagnostic test of the hard disk controller card circuitry. Any errors are reflected in the returned error code.

During testing, this function destroys the initialization state of the controller. On completion, the controller state is undefined. Use one of the hard disk functions (00H, 09H, or 0DH) to initialize the controller to a normal mode of operation.

Function 15H: Return Drive Type

Industry-Standard

Parameters

AH = 15H
DL = The drive number (80H or 81H)

Returns

CF = 0	Indicates a successful operation		
	AH = 00H	The drive is not present	
	AH = 03H	The hard disk is present	
		CX =	The number of 512 byte sectors (high-order 16-bits)
		DX =	The number of 512 byte sectors (low-order 16-bits
CF = 1	Indicates an error condition		
	AH = The error code		

This function returns the drive type of the indicated hard disk. If the returned AH register contains 03H, the CX and DX register pair contains the number of 512 byte sectors on the disk. The CX register is the high word of the pair.

Function D0H: Read Long 256 Byte Sector

DIGITAL Extension

Parameters

AH = D0H
AL = The number of sectors to read
CH = The cylinder number (lower 8 bits)
CL = The starting sector number (and bits 9-8 of cylinder)
DH = The head number
DL = The drive number (80H or 81H)
ES:BX = The buffer address

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function is similar to function 0AH, except that the sector size is 256 bytes instead of 512. Each sector of data is terminated by a 4-byte ECC field. This function reads the indicated number of sectors and stores the data starting at the buffer address in ES:BX. Attempts to store data past a physical page boundary return an error. This can occur when the data size exceeds 10000H or when the BX offset plus the data size exceed 10000H.

To calculate the required buffer size, multiply 260 by the contents of the AL register.

Diskette Functions

The value in the AH register indicates the desired diskette function. All diskette functions require a drive number in the DL register. The diskette drive numbers are 00H and 01H.

With the exception of the flags register, all registers not mentioned in the function description are preserved.

Diskette Errors

If CF is set (1), an error occurred, and the AH register contains the error code. Table 15-9 lists the diskette error codes.

Table 15-9 Diskette Error Codes

Error Code	Description
A0H	Combination of 80H and 20H error codes
80H	Hardware failed to respond
40H	Seek operation failed
20H	Disk controller failed
10H	CRC incorrect for data
09H	Direct Memory Access (DMA) overflowed 64K page boundary
08H	DMA controller failed to respond
06H	Disk change line *true*
04H	Sector not found
03H	Diskette write protected
02H	Sector address mark not found
01H	Illegal I/O request (bad command)

Diskette Parameter Tables

A diskette parameter table defines the physical characteristics of a diskette. The values in the table are used by the diskette driver to initialize the diskette controller. Table 15-10 describes the contents of a diskette parameter table. Each parameter in Table 15-10 is one byte long.

Table 15-10 Diskette Parameter Table Description

Offset	Bits	Description
00H	7-4	Step rate
		Each increase in the value of bits 7-4 decreases the step rate by 1 ms, so that zero equals 16 ms, one equals 15 ms, two equals 14 ms, and so on.
	3-0	Head unload time
		Each increase in the value of bits 3-0 increases the head unload time by 16 ms, so that zero equals 16 ms, one equals 32 ms, two equals 48 ms, and so on.
01H	7-1	Head load time
		Each increase in the value of bits 7-1 increases the head load time by 2 ms, so that zero equals 2 ms, one equals 4 ms, two equals 6 ms, and so on.
	0	Direct Memory Access (DMA) selection 0 = Do not use DMA mode 1 = Use DMA mode
02H	7-0	Clock ticks until the motor is turned off
03H	7-0	Sector size
		Each increase in value doubles the sector size, so that zero equals 128 bytes, one equals 256 bytes, two equals 512 bytes and so on. The default value is two (512 bytes).
04H	7-0	Sectors per track (8, 9, 10, or 15)
05H	7-0	Sector gap length (1BH)
06H	7-0	Data length (FFH)
07H	7-0	Format gap length (54H)
08H	7-0	Format fill byte (F6H)
09H	7-0	Head settle time in milliseconds
		If this value is less than 17 ms, the ROM BIOS uses 17 ms.
0AH	7-0	Motor start-up time in .125 second increments

The interrupt vector for interrupt 1EH points to the diskette parameter table. If a diskette drive does not exist, the interrupt vector for interrupt 1EH is reserved and undefined.

Function 00H: Initialize Diskette Subsystem

Industry-Standard

Parameters

AH = 00H
DL = The drive number (00H or 01H)

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function resets the diskette controller to its initial power-up state. the diskette controller is initialized to the values in the diskette parameter table. Because the diskette drive is marked as reset, the next diskette I/O request recalibrates that drive.

After hard disk function 00H resets the hard disk controller, it calls this function to reset the diskette controller.

Function 01H: Return Status Code of Last I/O Request

Industry-Standard

Parameters

AH = 01H
DL = The drive number (00H or 01H)

Returns

AH = 0
AL = The error code of the previous operation

This function returns, in the AL register, the error code of the last function call. If AL returns a 0, no previous error condition existed. Because calls to this function do not generate error conditions, successive calls return 0.

The AH register always returns 0.

For the diskette error codes, see Table 15-9.

Function 02H: Read One or More Track Sectors

Industry-Standard

Parameters

AH = 02H
AL = The number of sectors to read
CH = The track number
CL = The starting sector number
DH = The head number
DL = The drive number (00H or 01H)
ES:BX = The buffer address

Returns

AL = 00H		
CF = 0	Indicates a successful operation	
CF = 1	Indicates an error condition	
	AH = The error code	

This function reads the indicated number of sectors and stores the data starting at the buffer address in ES:BX. Requests to read more sectors than remain on the track return an error. Attempts to store data past a physical page boundary return an error. This can occur when the BX offset plus the data size exceed 10000H.

To calculate the required buffer size, multiply the contents of the AL register by 512.

The AL register always returns 00H.

Function 03H: Write One or More Track Sectors

Industry-Standard

Parameters

AH = 03H
AL = The number of sectors to write
CH = The track number
CL = The starting sector number
DH = The head number
DL = The drive number (00H or 01H)
ES:BX = The buffer address

Returns

AL = 00H
CF = 0 Indicates a successful operation
CF = 1 Indicates an error condition
AH = The error code

This function writes the indicated number of sectors starting at the buffer address in ES:BX. Requests to write more sectors than remain on the track return an error. Attempts to read data past a physical page boundary return an error. This can occur when the data size exceeds 10000H or when the BX offset plus the data size exceed 10000H.

To calculate the number of sectors in a buffer, divide the buffer size by 512. If the division produces a remainder, increment the sector count.

Function 04H: Verify One or More Track Sectors

Industry-Standard

Parameters

AH = 04H
AL = The number of sectors to verify
CH = The track number
CL = The starting sector number
DH = The head number
DL = The drive number (00H or 01H)
ES:BX = The buffer address

Returns

AL = 00H
CF = 0 Indicates a successful operation
CF = 1 Indicates an error condition
AH = The error code

This function verifies the indicated number of sectors. The data is not compared against data in memory. It is only verified for internal consistency. Thus, the verify command only checks for Cyclical Redundancy Check (CRC) errors. Requests to verify more sectors than remain on the track return errors.

Function 05H: Format a Track

Industry-Standard

Parameters

AH = 05H	
AL = The number of sectors to format	
CH = The track number	
DH = The head number	
DL = The drive number (00H or 01H)	
ES:BX = The track identification table address	

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function formats the indicated track. It formats only the sectors described in the track identification table. The data field of the formatted sectors is initialized to the diskette parameter table value, *sector fill.* Before formatting the track, the ROM BIOS initializes the diskette controller to the values found in the diskette parameter table.

ES:BX points to the track identification table, which contains an entry for each sector on the track. A table entry requires four bytes of data. Therefore, the expected buffer size is four times the number of sectors per track. The following list describes a single table entry.

Offset	Name	Description
00H	Track	0 to 39 for 48 tracks per inch (TPI) 0 to 79 for 96 TPI
01H	Head	0 = the back side of diskette 1 = the label side of diskette
02H	Sector number	1 to 8 for 48 TPI 1 to 9 for 48 TPI 1 to 15 for 96 TPI (high capacity)
03H	Sector size	Each increase in value doubles the sector size, so that 0 equals 128 bytes, 1 equals 256 bytes, 2 equals 512 bytes, and so on. The default value is two (512 bytes).

Function 15H: Return Drive Type

Industry-Standard

Parameters

AH = 15H	
DL = The drive number (00H or 01H)	

Returns

CF = 0	Indicates a successful operation	
	AH = 00H	The drive is not present
	AH = 02H	An RX33 drive with status change line
CF = 1	An error condition	
	AH = The error code	

This function returns the drive type of the indicated diskette drive.

Function 16H: Return Change Line Status

Industry-Standard

Parameters

AH = 16H
DL = The drive number (00H or 01H)

Returns

CF = 0	AH = 00H	The media has not changed
CF = 1	AH = 06H	The media could have changed

This function returns, for the indicated drive, the state of the diskette change line. A changed status indicates that the media may have been changed since the last I/O request to that drive. The change flag is set only after the media is changed and the drive door is closed. If the door is open or no media is present, a timeout error occurs.

Function 17H: Set Drive and Media Type for Format

Industry-Standard with DIGITAL Extensions

Parameters

AH = 17H	
AL = 02H	There is 48 tracks per inch (TPI) media in the RX33 drive
AL = 03H	There is 96 TPI high-capacity media in the RX33 drive
AL = 04H	There is 96 TPI low-capacity media in the RX33 drive (DIGITAL extension)
DL = The drive number (00H or 01H)	

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition AH = The error code

This function sets the diskette media and the drive type. It is called before function 05H to override an existing diskette format or to define the format for a blank diskette.

Interrupt 14H: Asynchronous Communications

Software Interrupt - Industry-Standard with DIGITAL Extensions

This interrupt provides an industry-standard software interface to the asynchronous communications ports. It also supports extended functionality:

- Buffered transmit
- Buffered receive
- Receive notification
- Flow control
- Line signal notification
- Modem signal control
- Modem change notification
- Break conditions
- Error handling (timeout or continuous loop)
- Additional baud rates

In accordance with industry-standard practice, the ROM BIOS code supports four serial ports, and the ROM BIOS data area maintains four base addresses. However, due to the limited number of interrupt controller inputs, some portions of extended function D0H are limited to ports 00H and 01H.

During power-up, the ROM BIOS looks for serial ports at I/O addresses 03F8H and 02F8H. The serial port at 03F8H is the integral serial port and is assigned to port 00H. If a serial port is found at I/O address 02F8H, it is assigned to port 01H. Normally, this is the optional integral modem.

The serial printer port is treated as a special case. It is assigned to serial port FFH and can be accessed like other serial ports. This information is provided for consistency only. Use interrupt 17H, the parallel printer output, for normal printer output. The ROM BIOS redirects parallel port 00H to serial port FFH.

The following is a list of the available functions:

AH	Description	Digital Extended
00H	Initialize the asynchronous port	No
01H	Send a character	No
02H	Receive a character	No
03H	Return asynchronous port status	No
D0H	Extended mode	Yes
D1H	Break control	Yes
D2H	Modem control	Yes
D3H	Retry On Timeout Error	Yes
D4H	Baud rate select	Yes

All registers not specified in the function register usage are preserved.

Function 00H: Initialize Asynchronous Port

Industry-Standard

Parameters

AH = 00H
AL = Initialization byte

Bits 7-5	Baud rate 000 = 110 001 = 150 010 = 300 011 = 600 100 = 1200 101 = 2400 110 = 4800 111 = 9600
Bits 4-3	Parity 00 = None 01 = Odd 10 = None 11 = Even
Bit 2	Stop bits 0 = 1 1 = 2
Bits 1-0	Data bits 00 = 5 01 = 6 10 = 7 11 = 8

DX = The port number (00H to 03H and FFH)

Returns

AH = The data status as defined in function 03H

This function initializes the specified port. The value returned in the AL register is interpreted the same as the value returned by function 03H.

Function 01H: Transmit Character

Industry-Standard

Parameters

AH = 01H
AL = Character to transmit
DX = The port number (00H to 03H and FFH)

Returns

AH = Data status as defined in function 03H

This function attempts to transmit a character to the specified port. Unless modem signal bypass is set (see Interrupt 14H, Function D2H), the following modem signals are required to complete a transmission:

VAXmate Workstation to External Device	External Device to VAXmate Workstation
Data Terminal Ready (DTR)	Data Set Ready (DSR)
Request To Send (RTS)	Clear To Send (CTS)

Buffer Mode Enabled

If buffered mode is enabled, continuous retry is disabled, and bit 7 is set in the returned AH register, then the transmit buffer was full and the character was not placed in the buffer.

The XON/XOFF characters defined in the communications control block (see Interrupt 14H, Function D0H) are transmitted independently and before all other characters in the buffer.

For more information on buffered mode, see function D0H.

Function 02H: Receive Character

Industry-Standard

Parameters

AH = 02H
DX = The port number (00H to 03H and FFH)

Returns

AL = Received character
AH = Data status as defined in function 03H

The function attempts to receive data from the specified port. Unless modem signal bypass is set (see Interrupt 14H, Function D2H), the following modem signals are required to complete a transmission.

VAXmate Workstation	External Device
Data Terminal Ready (DTR)	Data Set Ready (DSR)

Buffer Mode Enabled

If bit 7 is set in the returned AH register, the receive buffer is empty.

If bit 1 is set in the returned AH register, the receive buffer overflowed. The character stored in the buffer is the overflow character, as specified in the communications control block (CCB).

For more information on buffered mode, see function D0H.

Function 03H: Return Asynchronous Port Status

Industry-Standard

Parameters

AH = 03H
DX = The port number (00H to 03H and FFH)

Returns

AH = The data status (set bits indicate condition)

Bit 7 -	Timeout error
Bit 6 -	8250 transmit shift register empty (all data has been transmitted)
Bit 5 -	8250 transmit holding register empty (ready to accept another character for transmission)
Bit 4 -	Break detect
Bit 3 -	Framing error
Bit 2 -	Parity error
Bit 1 -	Overrun error
Bit 0 -	8250 receive buffer full (received character available)

AL = The modem status (set bits indicate condition)

Bit 7 -	Carrier detect
Bit 6 -	Ring indicate
Bit 5 -	Data set ready
Bit 4 -	Clear to send
Bit 3 -	Delta carrier detect *
Bit 2 -	Ring trailing edge
Bit 1 -	Delta data set ready *
Bit 0 -	Delta clear to send *

* If a delta bit is set, it indicates that between that last status request and this status request, the state of the indicated input has changed.

This function retrieves the current data and modem status of the specified port.

Buffer Mode Enabled

If the receive buffer is empty, the status returned is the current data and modem status. Otherwise, the data status reflects the status of the next character to be extracted from the buffer, and the modem status is the current modem status.

For more information on buffered mode, see function D0H.

Function D0H: Extended Mode

DIGITAL Extension

Parameters

AH = D0H

AL = FFH	Enable function
AL = 00H	Disable function

DX = The port number (00H to 01H and FFH)
ES:BX = The address of CCB (ignored when AL = 00H)

Returns

AL = 00H	Indicates a successful operation
AL = 01H	Indicates a nonexistent device
AL = 02H	Indicates that the first four CCB entries are 0 (enable only)
AL = 03H	Indicates that the buffer size is less than 4 (enable only)

This function is an extension to the industry-standard asynchronous communications functions. The following features are available:

- Notification on data status interrupt
- Notification on modem status interrupt
- Flow control
- Buffered communications
  - Notification on receive interrupt (requires receive buffering)
  - Notification on transmit interrupt (requires transmit buffering)

ES:BX points to the communication control block (CCB), which specifies the desired environment. Each port is allowed only one CCB. The CCB is assigned to the port indicated in the DX register. The communications control block is defined in Table 15-11.

While a CCB is enabled, the first four entries cannot be modified. Within reason, the remaining entries in the CCB and buffer structures can be modified dynamically.

The CCB and the CCB-buffer structure must reside in the same memory segment.

Table 15-11 Communications Control Block (CCB) Description

Offset	Name	Size	Description
00H	Line Vector	Double word	This is the address (segment:offset) of the received-data status interrupt service routine. This interrupt occurs on an overrun, parity, framing, or break error condition. A value of 0000:0000H disables this notification. While the CCB is enabled, this value must not be changed.
04H	Modem Vector	Double word	This is the address (segment:offset) of the modem status interrupt service routine. This interrupt occurs whenever clear-to-send, data-set-ready, ring-indicator, or received-line-signal-detector changes state. A value of 0000:0000H disables this notification. While the CCB is enabled, this value must not be changed.
08H	Rxbuff	Word	This is the offset of the receive-buffer structure. Table 15-12 describes the receive-buffer structures. A value of 0000H disables receive buffering for the indicated port. While the CCB is enabled, this value must not be changed.
0AH	Txbuff	Word	This is the offset of the transmit-buffer structure. Table 15-12 describes the transmit-buffer structures. A value of 0000H disables transmit buffering for the indicated port. While the CCB is enabled, this value must not be changed.
0CH	Xflag	Byte	This byte selects the XON/XOFF flow control option. It can be changed at any time without restriction. A value of FFH enables flow control. A value of 00H disables flow control. Flow control is only available when the buffered mode is active. Flow control should be disabled for 8-bit binary data transfers.
0DH	Status	Byte	This byte contains the current flow control state. If the high order nibble equals 00H, transmissions are disabled. If the high order nibble equals F0H, transmissions are enabled. The low-order nibble contains internal state information and must be preserved.

Table 15-11 Communications Control Block (CCB) Description (cont.)

Offset	Name	Size	Description
0EH	Overflow	Byte	This byte contains the character code that represents an overflow condition. Under an overflow (buffer full) condition, this value is written over the last character in the buffer. Because a new overflow character can be written over an old overflow character, this value should not be changed while the CCB is enabled.
0FH	Xonchr	Byte	This byte contains the character code used for XON in the flow control operation. Because the external device must cooperate in the change, this value should not be changed while the CCB is enabled.
10H	Xoffchr	Byte	This byte contains the character code used for XOFF in the flow control operation. Because the external device must cooperate in the change, this value should not be changed while the CCB is enabled.
11H	Xonpt	Word	This value defines the number of characters left in the receive buffer when XON is sent (low-water mark). This value should be changed only when the receive buffer is empty. However, it can be changed any time if synchronization of flow control is protected.
13H	Xoffpt	Word	This value defines the number of characters in the receive buffer when XOFF is sent (high water mark). This value should be changed only when the receive buffer is empty. However, it can be changed any time if synchronization of flow control is protected.
15H	CntlMask	Byte	If the value of this byte is nonzero, the received characters are *AND*ed with it. The *AND* operation takes place before the character is tested as a flow control character. This value can be changed at any time.

Each CCB can have two buffer structures associated with it, one for receive and one for transmit. The buffer structures contain pointers, counters, and status information. The maximum size of any buffer is 64 Kbytes. Table 15-12 describes the CCB buffer structure.

Table 15-12 CCB Buffer Structure Description

Offset	Name	Size	Description
00H	Vector	Double word	This is the address (segment:offset) of the receive or transmit interrupt service routine. A value of 0000:0000H disables this option. It can be changed at any time.
04H	Head	Double word	This is a pointer (segment:offset) to the next empty position in the buffer. The segment of the head pointer must be common to the tail, start, and end pointers.
08H	Tail	Word	This is a pointer (offset only) to the next available character in the buffer. It assumes the same segment as the head pointer.
0AH	Start	Word	This is a pointer (offset only) to the beginning of the buffer. It assumes the same segment as the head pointer.
0CH	End	Word	This is a pointer (offset only) to the end of the buffer. It assumes the same segment as the head pointer.
0EH	Count	Word	This is the number of characters in the buffer.

Buffering Enabled

Receive and transmit buffering is enabled or disabled by the contents of the CCB receive-buffer/transmit-buffer structure pointers. A nonzero pointer indicates a valid pointer to a structure, and the desire for buffering.

Function D0H does not initialize buffer pointers or counters. It only updates them. This allows a CCB to be enabled (AL = FFH) or disabled (AL = 00H) dynamically. Therefore, before a CCB is enabled for the first time, all buffer pointers must be initialized, and all counters must be zeroed.

The ROM BIOS expects the receive buffer and the transmit buffer to occupy distinct and separate locations.

When buffering is enabled, functions 00H, 01H, 02H, and 03H continue to operate in the same manner. However, some operations have minor side-effects on buffered ports:

- Transmit Buffering Enabled

  Function 01H buffers characters until they are transmitted. If continuous retry is disabled, a timeout error indicates a full buffer instead of a timeout.

  The XON and XOFF characters are handled independently and before any buffered characters.

- Receive Buffering Enabled

  Function 02H extracts characters from a buffer. If continuous retry is disabled, a timeout error indicates an empty buffer instead of a timeout.

  The data status returned by functions 00H, 01H, 02H, and 03H depends on the state of the receive buffer. If the receive buffer is empty, the returned data status is the current data status. Otherwise, the returned data status is associated with the next character to be extracted from the buffer and reflects the status when the character was placed in the buffer. The modem status is always the current modem status.

Notification Enabled

Due to the limited number of interrupt controller inputs, only ports 00H, 01H, and FFH can use notification.

When a CCB is enabled (AL = FFH), the ROM BIOS examines the LineVector and ModemVector pointers. On finding a nonzero pointer, the ROM BIOS enables the associated interrupt for the indicated port. The ROM BIOS then examines the receive-buffer/transmit-buffer structure pointers. On finding a nonzero buffer structure pointer, it is used to examine the Vector pointer. If it is nonzero, the associated receive or transmit interrupt is enabled. The interrupts remain enabled until the CCB is disabled (AL = 00H). The application must disable the CCB before exiting. Otherwise, the ROM BIOS assumes that it still owns the CCB and buffer locations. On the next interrupt, it uses them with undefined results.

All service routines are accessed using far calls. They must return to the ROM BIOS by a far return.

At the time of the call, CPU interrupts are disabled and the interrupt controller is waiting for an end-of-interrupt instruction. After the notified service routine returns control to the ROM BIOS, the interrupt controller is restored, and any additional asynchronous port interrupts are serviced. After all asynchronous interrupts are serviced, the CPU interrupt state is restored by an IRET instruction. To maintain a minimum system interrupt latency, keep the service routine as short as possible. Otherwise, handling high baud rates can create problems, such as missing a timer interrupt.

The service routines are called as follows:

- Receive Notification Enabled

  When the service routine is called, ZF is clear, the current data and modem status are in the AX register (see function 03H), and interrupts are disabled. The service routine returns ZF to indicate the action the ROM BIOS should take. If ZF is set (1), the ROM BIOS ignores the data and modem status in the AX register. If ZF is clear (0), the ROM BIOS stores the data and modem status in the AX register.

- Transmit Notification Enabled

  When the service routine is called, the AL register contains the current data status (same as AH in function 03H), and interrupts are disabled. This service routine is called when the transmit buffer is empty.

- Received Data Status Notification Enabled

  When the service routine is called, the AL register contains the current data status (same as AH in function 03H), and interrupts are disabled. This service routine is called when an overrun, parity, framing, or break interrupt error condition occurs.

- Modem Status Notification Enabled

  When the service routine is called, the AL register contains the current modem status (same as AL in function 03H), and interrupts are disabled. This service routine is called when a clear-to-send (CTS), data-set-ready (DSR), ring-indicator (RI), or received-line-signal-detector (RLSD) changes state.

Error Codes Returned

When enabling a CCB (AL = FFH), function D0H can return one of the following status codes in the AL register:

AL	Meaning
00H	A successful operation
01H	A nonexistent port was specified in the DX register
02H	No operation was specified (first 4 entries in CCB contain 0)
03H	There was an invalid buffer description. (The end pointer must be at least four more than the start pointer.)

When disabling a CCB (AL = 00H), function D0H can return one of the following status codes in the AL register:

AL	Meaning
00H	A successful operation
01H	There was a nonexistent port specified in the DX register

Function D1H: Send Break

DIGITAL Extension

Parameters

AH = D1H
DX = The port number (00H to 03H and FFH)

AL = FFH	Set the break condition
AL = 00H	Clear the break condition

Returns

AH = The data status as defined in function 03H
AL = The modem status as defined in function 03H

This function sets or clears the break condition. When AL equals FFH, the transmit data line is forced to the space state (break condition is set). When AL equals 00H, the transmit data line is returned to the mark state (break condition is cleared).

Function D2H: Set Modem Control

DIGITAL Extension

Parameters

AH = D2H
DX = The port number (00H to 03H)

AL = FFH	Set the modem signal bypass
AL = F0H	Clear the modem signal bypass
AL = 0FH	Read the modem control register
AL = 00H to 07H	Write modem control register

	Bit 0	Data Terminal Ready (DTR) 0 = DTR is low at the external connector 1 = DTR is high at the external connector
	Bit 1	Request to Send (RTS) 0 = RTS is low at the external connector 1 = RTS is high at the external connector
	Bit 2	Speed Select (SS) 0 = SS is low at the external connector 1 = SS is high at the external connector

Returns

AL = The contents of the modem control register

This function reads or writes the modem control register and sets or clears the modem signal bypass feature.

The modem signal bypass feature disables or enables ROM BIOS servicing of modem line state changes. If the modem signal bypass is cleared (AL = F0H), the ROM BIOS services the modem signals. If the modem signal bypass is set (AL = FFH), the ROM BIOS does not service the modem signals. When modem signal bypass is enabled, the modem signals are ignored and the serial port operates in a data-leads-only mode.

Because it provides a different means of handling modem line state changes, the modem signal bypass feature is not applicable when buffered communication is enabled. For further information on buffered communication, see function D0H.

Function D3H: Retry on Timeout Error

DIGITAL Extension

Parameters

AH = D3H	
AL = 01H	Return the current retry map in AL and CL
AL = 00H, 02H-FFH	Write AL and CL (bits 1-0) to the retry map CL (bits 1-0) = Port FFH retry map

Returns

AL = The current retry map (return map AL = 01H)
CL = The current retry map for port FFH

This function controls the ROM BIOS support of asynchronous port timeout errors. Each port is individually controlled by reading the current map and changing only the desired control bits. If the AL register equals 01H, the current retry map is returned in the AL register. Otherwise, the contents of the AL register are written to the retry map. Initially, all ports are set to return timeout errors. The retry map bit assignments are defined as follows.

Reg	Port	Bits	Usage
AL	03H	7-6	00 = return the timeout errors 11 = loop on the timeout errors
	02H	5-4	00 = return the timeout errors 11 = loop on the timeout errors
	01H	3-2	00 = return the timeout errors 11 = loop on the timeout errors
	00H	1-0	00 = return the timeout errors 11 = loop on the timeout errors
CL	FFH	7-2	Unused
		1-0	00 = return the timeout errors 01 = loop on timeout errors

Function D4H: Set Baud Rate

DIGITAL Extension

Parameters

AH = D4H
DX = The port number (00H to 03H)

AL = FFH	Return the current baud rate in AL
AL = 00H-FEH	Any value other than FFH sets the baud rate

Returns

AL = The current baud rate selection

Bits 7-5		Standard group	Extended group
	000 =	110	50
	001 =	150	75
	010 =	300	134.5
	011 =	600	1800
	100 =	1200	2000
	101 =	2400	3600
	110 =	4800	7200
	111 =	9600	19200

Bit 4 — Baud rate group select
- 0 = Select the baud rate from the standard group
- 1 = Select the baud rate from the extended group

Bit 3-1 — Not Used

Bit 0 — Split baud rate
- 0 = The baud rate is not split (Port 0 only)
- 1 = The baud rates are split (Port 0 only)

This function sets or reads the baud rate and selects split baud rates. Compared to function 00H, it provides an expanded set of baud rates. Excluding the split baud rate selection, this function is compatible with industry-standard serial port adapters.

Split baud rates are only supported on DIGITAL serial ports that have a split baud rate capability. When the baud rates are split, the receive baud rate is fixed at 1200 baud, and the transmit baud rate is set to the currently selected value. The VAXmate workstation integral COM1 port supports split baud rates.

Interrupt 15H: Cassette Input/Output

Software Interrupt - Industry-Standard with DIGITAL Extensions

This function provides support for multitasking and other functions associated with the 80286 CPU in virtual memory mode. Because there is no cassette hardware, the original cassette I/O functions respond as errors. An extended function returns a DIGITAL-specific configuration word in the BX register.

Any AH values not defined in the following list of Interrupt 15H functions return the error condition AH equals 86H and CF set (1).

AH	Function Name	DIGITAL Extended
00H-7FH	Returns error condition (AH = 86H, CF = 1)	No
80H	Open Device	No
81H	Close Device	No
82H	Termination	No
83H	Set Wait Interval	No
84H	Joystick Support (not supported)	No
85H	Service System Request Key	No
86H	Wait	No
87H	Move a Block of Memory	No
88H	Memory size above 1Mb	No
89H	Begin Virtual Mode	No
90H	Device is Busy	No
91H	Interrupt Completion Handler	No
D0H	Return DIGITAL Configuration Word	Yes

Function 80H: Open Device

Industry-Standard

Parameters

AH = 80H

Returns

AH = Undefined
CF = 0

This function is just a hook. It only returns the indicated values.

Function 81H: Close Device

Industry-Standard

Parameters

AH = 81H

Returns

AH = Undefined
CF = 0

This function is just a hook. It only returns the indicated values.

Function 82H: Termination

Industry-Standard

Parameters

AH = 82H

Returns

AH = Undefined
CF = 0

This function is just a hook. It only returns the indicated values.

Function 83H: Set a Wait Interval

Industry-Standard

Parameters

AH = 83H
CX = High 16-bits (number of microseconds delay time) *
DX = Low 16-bits (number of microseconds delay time) *
ES:BX = The pointer to caller-supplied flag byte

* Although this parameter is measured in microseconds, the minimum resolution is 976 μs. Requested values are rounded up to the next 976-μs increment.

Returns

CF = 1	The error condition
CF = 0	Interval timer started (application should monitor bit 7 of the caller-supplied flag byte)

This function does not wait until the time interval has elapsed before returning. It returns to the caller immediately. After the specified time interval has elapsed, this function sets bit 7 of a caller-supplied flag byte. The time interval is a 32-bit value measured in microseconds.

If an attempt is made to start a second interval before the first interval is completed, an error condition is returned (CF = 1). This function is mutually exclusive of function 86H.

Function 84H: Joystick Support

Not Supported

Parameters

AH = 84H

Returns

CF = 1 Indicates an error (always returns CF = 1)

> **NOTE**
> This function always returns an error (CF = 1). The VAXmate workstation does not support joysticks.

Function 85H: Service System Request Key

Industry-Standard

Parameters

AH = 85H

Returns

AH = Undefined
CF = 0

This function is just a hook. It only returns the indicated values.

Function 86H: Wait (No Return to User)

Industry-Standard

Parameters

AH = 86H
CX = High 16-bits (number of microseconds delay time) *
DX = Low 16-bits (number of microseconds delay time) *

* Although this parameter is measured in microseconds, the minimum resolution is 976 μs. Requested values are rounded up to the next 976-μs increment.

Returns

CF = 0	Indicates a successful operation
CF = 1	Indicates an error condition

This function waits until the specified interval has elapsed before control is returned. The time interval is a 32-bit value measured in microseconds.

This function is mutually exclusive of function 83H. If attempted while function 83H is active, it returns an error (CF = 1).

Function 87H: Move a Block of Memory

Industry-Standard

Parameters

AH = 87H
CX = The number of 16-bit words to move (8000H maximum)
ES:SI = The pointer to a table of caller-supplied GDT descriptors

Returns

AH = 00H	Indicates a successful operation
AH = 01H	Indicates a RAM parity error (error cleared)
AH = 02H	Indicates an exception interrupt error
AH = 03H	Indicates an address line 20 gating failure

This function moves a block of memory to or from the address space above 1 Mbyte. The completion status is returned in the AH register and is also stored in the real-time clock's CMOS RAM at offset 3CH. Table 15-13 describes the caller-supplied table of descriptors.

Table 15-13 Function 87H Descriptor Table

Offset	Size	Contents
00H	8 bytes	All 0
08H	8 bytes	GDT Descriptor These values are loaded into the GDTR with the LGDT instruction. The ROM BIOS supplies this descriptor. It is the 24-bit equivalent of the entry-time contents of ES:SI. Thus, this structure becomes the GDT.
10H	8 bytes	Data Segment Descriptor After entering virtual protected mode, these values are loaded into the DS register. The descriptor is supplied by the caller. This descriptor points to the source data block. The data block must begin at offset 0 in this segment.
18H	8 bytes	Extra Segment Descriptor After entering virtual protected mode, these values are loaded into the ES register. The descriptor is supplied by the caller. This descriptor points to the destination of the data block. The destination must begin at offset 0 in this segment.
20H	8 bytes	Code Segment Descriptor This is the code segment descriptor for the ROM BIOS code. The ROM BIOS supplies this descriptor.
28H	8 bytes	Stack Segment Descriptor This is the stack segment descriptor for the ROM BIOS code. This descriptor is filled in by the ROM BIOS. The ROM BIOS uses the caller stack segment and stack pointer. The caller is responsible for providing a minimum of 256 bytes of stack space.

Function 88H: Return Memory Size Above One Megabyte

Industry-Standard

Parameters

AH = 88H

Returns

AX = Starting from address 100000H, the number of contiguous 1 Kbyte blocks

This function returns the amount of contiguous memory above 1 Mbyte as determined during powerup.

Function 89H: Begin Virtual Mode

Industry-Standard

Parameters

AH = 89H
BH = The offset of interrupt level 1 in interrupt descriptor table (IRQ0-7)
BL = The offset of interrupt level 2 in interrupt descriptor table (IRQ8-15)
ES:SI = The pointer to a table of caller-supplied descriptors

Returns

AH = 00H	Indicates a successful operation
AH = FFH	Indicates an address line 20 gating failure

This function provides a method of entering the virtual protected mode of the 80286 CPU. Table 15-14 describes the caller-supplied table of descriptors. The caller must initialize all required tables. Within those tables are new locations for all 16 hardware interrupt vectors (15 plus the unused IRQ2). Control is returned to the caller at the instruction following the Interrupt 15H instruction that invoked this function. At that time, one of two conditions exists as follows:

- The CPU is in virtual protected mode, all interrupts are disabled, and the registers are initialized according to the descriptor tables. The AH register contains 00H.

  The master (BH) and slave (BL) peripheral interrupt controllers are programmed according to the descriptor tables pointed to by the BH and BL registers. Both peripheral-interrupt-controller mask registers contain FFH. That is, all interrupt inputs are disabled. The peripheral interrupt controllers are initialized to the following states:

  - Cascade mode is enabled
  - Slave identification equals 02H
  - Slave interrupts on IRQ2
  - Vector interval equals 8
  - Edge-triggered mode is enabled
  - 8086 mode is enabled
  - Normal EOI mode is enabled
  - Nonbuffered and not special-fully-nested modes are established
  - Interrupt descriptors pointed to by BH and BL are loaded into the respective ICW2 registers

- The CPU is in real mode and the registers are unchanged (except AH). The interrupt structure is the same as it was before the function was invoked. The AH register contains FFH, which indicates a failure in gating address line 20. The calling routine must be prepared to recover from this error condition.

Table 15-14 Function 89H Descriptor Table

Offset	Size	Contents
00H	8 bytes	All 0
08H	8 bytes	GDT Descriptor This caller-supplied descriptor is loaded into the GDTR with a LGDT instruction.
10H	8 bytes	IDT Descriptor This caller-supplied descriptor is loaded into the IDTR with a LIDT instruction
18H	8 bytes	Data Segment Descriptor This caller-supplied descriptor is loaded into the DS register.
20H	8 bytes	Extra Segment Descriptor This caller-supplied descriptor is loaded into the ES register.
28H	8 bytes	Stack Segment Descriptor This caller-supplied descriptor is loaded into the SS register.
30H	8 bytes	Code Segment Descriptor This caller-supplied descriptor is for the code that receives control at the successful conclusion of this function.
38H	8 bytes	Code Segment Descriptor This descriptor is supplied by the ROM BIOS and is used by the ROM BIOS while it operates in virtual protected mode.

Function 90H: Device Is Busy

Industry-Standard

Parameters

AH = 90H

Returns

AH = Undefined
CF = 0

This function is just a hook. It only returns the indicated values.

Function 91H: Interrupt Completion Handler

Industry-Standard

Parameters

AH = 91H

Returns

AH = Undefined
CF = 0

This function is just a hook. It only returns the indicated values.

Function D0H: Return DIGITAL Configuration Word

DIGITAL Extension

Parameters

AH = D0H

Returns

AH = 86H
CF = 1
BX = The DIGITAL configuration word

This function returns the DIGITAL-unique hardware configuration information in the BX register. The BX register has the following bit definitions:

Bit	Description
15	Unused
14	Modem option or COM2 present
13-10	Hard disk type 0000 = There is no hard disk 0001 = The hard disk is an RD*xx* type drive 0010-1110 = Reserved 1111 = Unknown
9	Hard disk controller present 0 = No hard disk controller 1 = Hard disk controller present
8	Expansion box 0 = No expansion box (no battery) 1 = Expansion box present (implies battery present)
7-5	Video type 000 = An industry-standard monochrome adapter 001 = An industry-standard color graphic adapter 010 = The VAXmate graphic video system 011-111 = Reserved

Bit	Description (DIGITAL Configuration Word - cont.)
4	LK250 keyboard 0 = LK250 keyboard not present 1 = LK250 keyboard present
3-2	Diskette drive type (drive 1) 00 = Not present 01 = RX31 (48-tpi drive) 10 = RX33 (96-tpi high capacity drive) 11 = Reserved
1-0	Diskette drive type (drive 0) 00 = Not present 01 = RX31 (48-tpi drive) 10 = RX33 (96-tpi high capacity drive) 11 = Reserved

Interrupt 16H: Keyboard Input

Software Interrupt - Industry-Standard with DIGITAL Extensions

This interrupt provides an interface to the LK250 keyboard. In addition to the industry-standard functions, it provides the following enhanced capabilities:

- Increase the size of the keyboard buffer
- Determine the number of characters in the keyboard buffer
- Real-time key notification
- Custom key mapping
- Selective disabling of various key conversion processes

Interrupt 16H supports the following functions:

Function	Description	DIGITAL Extended
00H	Keyboard Input	No
01H	Keyboard Status	No
02H	Keyboard State	No
D0H	Key Notification	Yes
D1H	Character Count	Yes
D2H	Keyboard Buffer	Yes
D3H	Extended Codes And Functions	Yes
D4H	Request Keyboard ID	Yes
D5H	Send To Keyboard	Yes
D6H	Keyboard Table Pointers	Yes

Table of Returned Scan Codes

Table 15-15 lists the scan codes returned for various conditions. The columns are marked as follows:

- The column marked "Key Pos" refers to the key positions shown in Figure 15-1.
- The column marked "D" indicates the scan code when a key is pressed.
- The column marked "R" indicates the scan code when a key is released.
- The columns marked "A" indicate the ASCII key value.
- The columns marked "S" indicate the scan code.

The symbol — indicates an invalid code that is ignored in the conversion process.

A *1 after an entry indicates that it is only available when DIGITAL extended codes are in effect and combination keys are disabled.

A *2 after an entry indicates that it is only available when DIGITAL extended codes are in effect and NumLock, Insert, and Scrl Lock are disabled.

A *3 after an entry indicates that it is only available when DIGITAL extended codes are in effect.

A *4 after an entry indicates that it is only available when Compose key pass through is in effect.

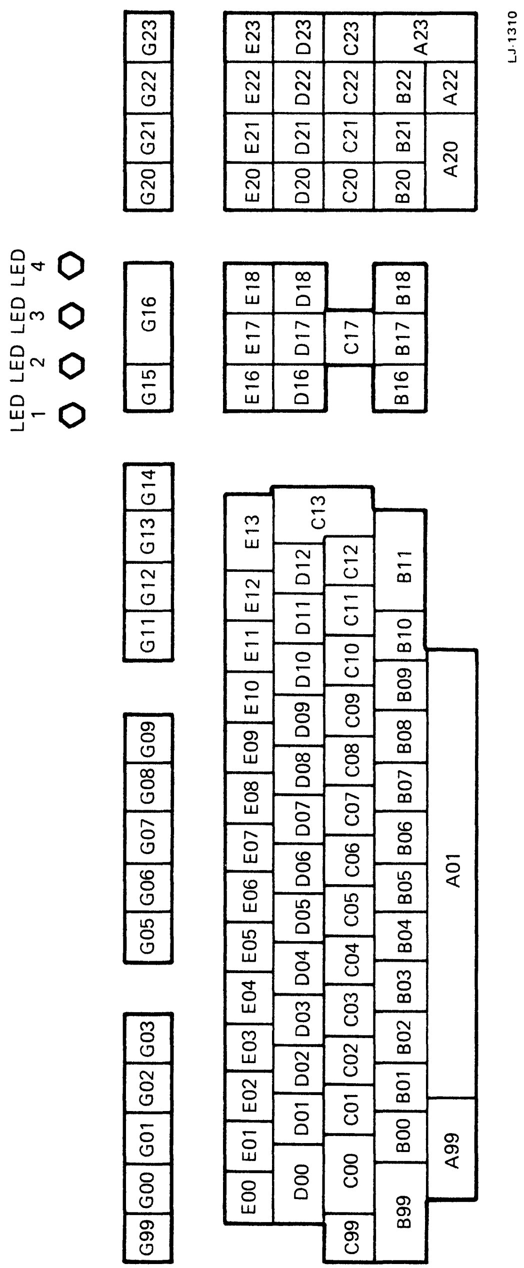

Figure 15-1: LK250 Keyboard Layout

Table 15-15 Keyboard Scan Codes Returned by the ROM BIOS

Key Pos	Scan Code D R	Alt A S	Lock A S	Ctrl A S	Normal A S	Num Lock A S	Shift A S
E20	01 81	-- --	1B 01	1B 01	1B 01	1B 01	1B 01
E01	02 82	00 78	31 02	-- --	31 02	31 02	21 02
E02	03 83	00 79	32 03	00 03	32 03	32 03	40 03
E03	04 84	00 7A	33 04	-- --	33 04	33 04	23 04
E04	05 85	00 7B	34 05	-- --	34 05	34 05	24 05
E05	06 86	00 7C	35 06	-- --	35 06	35 06	25 06
E06	07 87	00 7D	36 07	1E 07	36 07	36 07	5E 07
E07	08 88	00 7E	37 08	-- --	37 08	37 08	26 08
E08	09 89	00 7F	38 09	-- --	38 09	38 09	2A 09
E09	0A 8A	00 80	39 0A	-- --	39 0A	39 0A	28 0A
E10	0B 8B	00 81	30 0B	-- --	30 0B	30 0B	29 0B
E11	0C 8C	00 82	2D 0C	1F 0C	2D 0C	2D 0C	5F 0C
E12	0D 8D	00 83	3D 0D	-- --	3D 0D	3D 0D	2B 0D
E13	0E 8E	-- --	08 0E	7F 0E	08 0E	08 0E	08 0E
D00	0F 8F	-- --	09 0F	-- --	09 0F	09 0F	00 0F
D01	10 90	00 10	51 10	11 10	71 10	71 10	51 10
D02	11 91	00 11	57 11	17 11	77 11	77 11	57 11
D03	12 92	00 12	45 12	05 12	65 12	65 12	45 12
D04	13 93	00 13	52 13	12 13	72 13	72 13	52 13
D05	14 94	00 14	54 14	14 14	74 14	74 14	54 14
D06	15 95	00 15	59 15	19 15	79 15	79 15	59 15
D07	16 96	00 16	55 16	15 16	75 16	75 16	55 16
D08	17 97	00 17	49 17	09 17	69 17	69 17	49 17
D09	18 98	00 18	4F 18	0F 18	6F 18	6F 18	4F 18
D10	19 99	00 19	50 19	10 19	70 19	70 19	50 19
D11	1A 9A	-- --	5B 1A	1B 1A	5B 1A	5B 1A	7B 1A
D12	1B 9B	-- --	5D 1B	1D 1B	5D 1B	5D 1B	7D 1B
C13	1C 9C	-- --	0D 1C	0A 1C	0D 1C	0D 1C	0D 1C
C99	1D 9D	-- --	-- --	-- --	-- --	-- --	-- --
C01	1E 9E	00 1E	41 1E	01 1E	61 1E	61 1E	41 1E

Table 15-15 Keyboard Scan Codes Returned by the ROM BIOS (cont.)

Key Pos	Scan Code D R	Alt A S	Lock A S	Ctrl A S	Normal A S	Num Lock A S	Shift A S	
C02	1F 9F	00 1F	53 1F	13 1F	73 1F	73 1F	53 1F	
C03	20 A0	00 20	44 20	04 20	64 20	64 20	44 20	
C04	21 A1	00 21	46 21	06 21	66 21	66 21	46 21	
C05	22 A2	00 22	47 22	07 22	67 22	67 22	47 22	
C06	23 A3	00 23	48 23	08 23	68 23	68 23	48 23	
C07	24 A4	00 24	4A 24	0A 24	6A 24	6A 24	4A 24	
C08	25 A5	00 25	4B 25	0B 25	6B 25	6B 25	4B 25	
C09	26 A6	00 26	4C 26	0C 26	6C 26	6C 26	4C 26	
C10	27 A7	-- --	3B 27	-- --	3B 27	3B 27	3A 27	
C11	28 A8	-- --	27 28	-- --	27 28	27 28	22 28	
B00	29 A9	-- --	60 29	-- --	60 29	60 29	7E 29	
B99	2A AA	-- --	-- --	-- --	-- --	-- --	-- --	
C12	2B AB	-- --	5C 2B	1C 2B	5C 2B	5C 2B	7C 2B	
B01	2C AC	00 2C	5A 2C	1A 2C	7A 2C	7A 2C	5A 2C	
B02	2D AD	00 2D	58 2D	18 2D	78 2D	78 2D	58 2D	
B03	2E AE	00 2E	43 2E	03 2E	63 2E	63 2E	43 2E	
B04	2F AF	00 2F	56 2F	16 2F	76 2F	76 2F	56 2F	
B05	30 B0	00 30	42 30	02 30	62 30	62 30	42 30	
B06	31 B1	00 31	4E 31	0E 31	6E 31	6E 31	4E 31	
B07	32 B2	00 32	4D 32	0D 32	6D 32	6D 32	4D 32	
B08	33 B3	-- --	2C 33	-- --	2C 33	2C 33	3C 33	
B09	34 B4	-- --	2E 34	-- --	2E 34	2E 34	3E 34	
B10	35 B5	-- --	2F 35	-- --	2F 35	2F 35	3F 35	
B11	36 B6	-- --	-- --	-- --	-- --	-- --	-- --	
E23	37 B7	-- --	2A 37	00 72	2A 37	2A 37	2A 37	*1
A99	38 B8	-- --	-- --	-- --	-- --	-- --	-- --	
A01	39 B9	20 39	20 39	20 39	20 39	20 39	20 39	
C00	3A BA	-- --	-- --	-- --	-- --	-- --	-- --	
G99	3B BB	00 68	00 3B	00 5E	00 3B	00 3B	00 54	
G00	3C BC	00 69	00 3C	00 5F	00 3C	00 3C	00 55	

Table 15-15 Keyboard Scan Codes Returned by the ROM BIOS (cont.)

Key Pos	Scan Code D R	Alt A S	Lock A S	Ctrl A S	Normal A S	Num Lock A S	Shift A S
G01	3D BD	00 6A	00 3D	00 60	00 3D	00 3D	00 56
G02	3E BE	00 6B	00 3E	00 61	00 3E	00 3E	00 57
G03	3F BF	00 6C	00 3F	00 62	00 3F	00 3F	00 58
G05	40 C0	00 6D	00 40	00 63	00 40	00 40	00 59
G06	41 C1	00 6E	00 41	00 64	00 41	00 41	00 5A
G07	42 C2	00 6F	00 42	00 65	00 42	00 42	00 5B
G08	43 C3	00 70	00 43	00 66	00 43	00 43	00 5C
G09	44 C4	00 71	00 44	00 67	00 44	00 44	00 5D
E21	45 C5	-- --	00 45 *2	-- --	00 45 *2	-- --	00 45 *2
E22	46 C6	-- --	00 46 *2	-- --	00 46 *2	-- --	00 46 *2
D20	47 C7	-- --	00 47	00 77	00 47	37 47	37 47
D21	48 C8	-- --	00 48	-- --	00 48	38 48	38 48
D22	49 C9	-- --	00 49	00 84	00 49	39 49	39 49
D23	4A CA	-- --	2D 4A	-- --	2D 4A	2D 4A	2D 4A
C20	4B CB	-- --	00 4B	00 73	00 4B	34 4B	34 4B
C21	4C CC	-- --	00 4C *3	-- --	00 4C *3	35 4C	35 4C
C22	4D CD	-- --	00 4D	00 74	00 4D	36 4D	36 4D
C23	4E CE	-- --	2B 4E	-- --	2B 4E	2B 4E	2B 4E
B20	4F CF	-- --	00 4F	00 75	00 4F	31 4F	31 4F
B21	50 D0	-- --	00 50	-- --	00 50	32 50	32 50
B22	51 D1	-- --	00 51	00 76	00 51	33 51	33 51
A20	52 D2	-- --	00 52	-- --	00 52	30 52	30 52
A22	53 D3	-- --	00 53	-- --	00 53	2E 53	2E 53
G23	54 D4	-- --	00 98	00 B0	00 98	00 98	00 A4
E16	55 D5	-- --	00 85	-- --	00 85	00 85	00 85
E17	56 D6	-- --	00 86	00 C3	00 86	00 86	00 86
E18	57 D7	-- --	00 87	00 C1	00 87	00 87	00 87
D16	58 D8	-- --	00 88	-- --	00 88	00 88	00 88
D17	59 D9	-- --	00 89	00 C4	00 89	00 89	00 89
D18	5A DA	-- --	00 8A	00 C2	00 8A	00 8A	00 8A

Table 15-15 Keyboard Scan Codes Returned by the ROM BIOS (cont.)

Key Pos	Scan Code D R	Alt A S	Lock A S	Ctrl A S	Normal A S	Num Lock A S	Shift A S
C17	5B DB	-- --	00 8B	-- --	00 8B	00 8B	00 8B
B16	5C DC	-- --	00 8C	00 BF	00 8C	00 8C	00 8C
B18	5D DD	-- --	00 8D	00 C0	00 8D	00 8D	00 8D
B17	5E DE	-- --	00 8E	-- --	00 8E	00 8E	00 8E
G11	5F DF	00 B3	00 8F	00 A7	00 8F	00 8F	00 9B
G12	60 E0	00 B4	00 90	00 A8	00 90	00 90	00 9C
G13	61 E1	00 B5	00 91	00 A9	00 91	00 91	00 9D
G14	62 E2	00 B6	00 92	00 AA	00 92	00 92	00 9E
G15	63 E3	00 B7	00 93	00 AB	00 93	00 93	00 9F
G16	64 E4	00 B8	00 94	00 AC	00 94	00 94	00 A0
G20	65 E5	00 B9	00 95	00 AD	00 95	00 95	00 A1
G21	66 E6	00 BA	00 96	00 AE	00 96	00 96	00 A2
G22	67 E7	00 BB	00 97	00 AF	00 97	00 97	00 A3
E00	68 E8	00 BD *4	00 BD *4	00 BD *4	00 BD *4	00 BD *4	00 BD *4
A23	69 E9	00 BE	0D 9A	0A B2	0D 9A	00 9A	00 A6

Combination Keys

When detected by the ROM BIOS, certain key combinations invoke special functions. Detection of these key combinations occurs after key stroke notification (see function D0H) and before key buffering notification (see function D0H). If detected and acted upon (see function D3H), these key combinations are not stored in the keyboard buffer.

System Reset

The ROM BIOS recognizes the key combination Ctrl/Alt/Del as a system reset. When detected, interrupt 19H (Bootstrap) is executed.

System Request Key (Sys Req)

The ROM BIOS recognizes the key combination Alt/F20 as the system request key. When detected, interrupt 15H function 85H is executed.

The F20 key is also the Sys Req key, and sends the Sys Req key scan code. However, the system request function is executed only for the Alt/F20 key combination.

Extended Self-test

The ROM BIOS recognizes the key combination Ctrl/Alt/Home as the extended self-test key. When detected, the ROM BIOS invokes the extended self-test diagnostics.

Break

The ROM BIOS recognizes the key combination Ctrl/Break as break. When detected, the ROM BIOS stores 00H in the keyboard buffer and executes interrupt 1BH.

Pause

The ROM BIOS recognizes the key combination Ctrl/NumLock as a system pause. When detected, all noninterrupt driven tasks are suspended. The tasks are resumed by pressing any key except the following:

- NumLock
- Left-Shift
- Right-Shift
- Ctrl
- Alt
- Lock
- System Request (Alt/F20)
- Insert

Print Screen

The ROM BIOS recognizes the key combination Shift/Prt Sc as print screen. When detected, interrupt 05H is executed.

Automatic LED Control

The keyboard state can be changed by some functions and by user interaction. Therefore, during each keyboard function call, the ROM BIOS checks the keyboard state and updates the state of the LK250 LED indicators. Applications do not have to maintain the LK250 LED indicators.

Function 00H: Keyboard Input

Industry-Standard

Parameters

AH = 00H

Returns

AH = The scan code
AL = The ASCII key value

This function returns the next available character from the keyboard buffer. This function does not return until it has a character.

Function 01H: Keyboard Status

Industry-Standard

Parameters

AH = 01H

Returns

ZF = 1	The keyboard buffer is empty
ZF = 0	One or more characters in the keyboard buffer AH = The scan code (remains in buffer) AL = The ASCII key value (remains in buffer)

This function returns the status of the keyboard buffer. On return, ZF indicates the state of the buffer. If ZF is 1, the buffer is empty. If ZF is 0, the buffer contains one or more characters, and the next available character is returned (AH = scan code, AL = ASCII key value). The character remains in the keyboard buffer. That is, a subsequent function 00H call extracts the same scan code and ASCII key value.

Function 02H: Keyboard State

Industry-Standard

Parameters

AH = 02H

Returns

AL = The state of the modifier keys (set bit indicates the state is true)

Bit 7 - Insert
Bit 6 - Lock
Bit 5 - NumLock
Bit 4 - Scrol Lock
Bit 3 - Alternate
Bit 2 - Control
Bit 1 - Left Shift
Bit 0 - Right Shift

The function returns, in the AL register. the current state of the keyboard. A set (1) bit indicates that the corresponding state is true.

Function D0H: Key Notification

DIGITAL Extension

Parameters

AH = D0H	
AL = 00H	Disable key stroke and keyboard buffer notification
AL = 01H	Disable key stroke notification
AL = 02H	Disable keyboard buffer notification
AL = 81H	Return the pointer to the key stroke service routine in ES:BX
AL = 82H	Return the pointer to the keyboard buffer service routine in ES:BX
AL = FEH	Enable keyboard buffer notification
	ES:BX = The address of the keyboard buffer service routine
AL = FFH	Enable key stroke notification
	ES:BX = The address of the key stroke service routine

Returns

ES:BX = 0000:0000H	There is no active service routine (AL = 81H or AL = 82H)
ES:BX =	The pointer to the active service routine (AL = 81H or AL = 82H)

This function enables or disables key stroke or key buffering notification.

The service routines must preserve all registers except the AX and must use a far return to exit.

To determine if key stroke or keyboard buffer notification is in use, execute function D0H with AL = 81H or AL = 82H respectively. If the service routine is active, the returned ES:BX pair contains a pointer to the service routine. Otherwise, the returned ES:BX pair contains 0000:0000H.

Key Stroke Notification Enabled

Key Stroke Service Routine Parameters

AH = The keyboard state as defined in function 02H
AL = The scan code

Keystroke Service Routine Returns

AH = FAH   Keystroke should be ignored

AL = The scan code (original or revised)
AH = The keyboard state (original or revised)

The key stroke service routine is called each time a key is depressed or released.

On return from the service routine, the ROM BIOS examines the contents of the AL register. If the AL register contains FAH, the keystroke is ignored. Otherwise, the scan code in AL and the keyboard state in AH are treated as though the ROM BIOS had just established their values. Thus, an application can trap keys or map their values.

Key Buffering Notification Enabled

Key Stroke Service Routine Parameters

AH = The scan code
AL = The ASCII key value
BL = The keyboard state as defined in function 02H
ZF = 0

Keystroke Service Routine Returns

ZF = 1	Keystroke ignored (nothing stored in keyboard buffer) BL = The keyboard state (original or revised)
ZF = 0	The scan code in AH stored in keyboard buffer BL = The keyboard state (original or revised) AH = The scan code (original or revised) AL = The ASCII code (original or revised)

The key buffering service routine is called immediately before the character is placed in the keyboard buffer.

On return from the service routine, the ROM BIOS internal keyboard state is updated with the the contents of BL. The ROM BIOS then examines ZF. If ZF is set (1), the code returned in AX is ignored and is not stored in the keyboard buffer. If ZF is set (0), the code returned in AX is stored in the keyboard buffer.

Function D1H: Character Count

DIGITAL Extension

Parameters

AH = D1H

Returns

AX = The number of characters in the keyboard buffer

This function returns the number of characters remaining in the keyboard buffer. The maximum value that can be returned is the keyboard buffer size minus one. The default keyboard buffer is 16 characters long. However, the size of the buffer is not fixed. Function D2H increases or resets the size of the buffer.

Function D2H: Keyboard Buffer

DIGITAL Extension

Parameters

AH = D2H	
AL = 00H	Restore the keyboard buffer to the default location and size
AL = FFH	Establish a new keyboard buffer as defined by ES:BX and CX
CX = The new buffer size	
ES:BX = The pointer to start of new buffer	

Returns

Nothing

This function installs a new keyboard buffer or restores the default 16-character keyboard buffer.

When a keyboard buffer contains one entry less than its size, the keyboard buffer is full. To calculate the CX register value, add one to the desired capacity. For example, when the default 16-character keyboard buffer contains 15 characters, it is full. Each entry in the keyboard buffer requires 2 bytes, 1 for the scan code and 1 for the ASCII key value. To calculate the physical size of the buffer, double the value in the CX register.

Parameter values are not checked. Invalid or illogical buffer assignments cause unpredictable results. For example, do not use a buffer size of zero or a buffer that wraps around the end of a segment.

Function D3H: Extended Codes And Functions

DIGITAL Extension

Parameters

AH = D3H

AL = 00H Clear all bits (default state)

AL = Nonzero value (01H-FFH) is inclusive *OR*ed

Bit 7	Return the current bit usage in AL register This bit does not remain set. To return the current bit usage on successive calls, bit 7 must be set for each call.
Bit 6	Not Used
Bit 5	Enable Compose key pass through Although DIGITAL extended codes are disabled, the Compose key is placed in the keyboard buffer.
Bit 4	LK250 in DIGITAL extended mode This sends the DIGITAL extended mode command to the LK250, and ROM BIOS use of extended scan codes is enabled. Normally, the ten cursor edit pad keys return the scan codes of their equivalent numeric keypad keys.
Bit 3	Disable Shift/Lock override When the Lock key is in effect, the Shift key does not unshift alphabetic keys.
Bit 2	Disable combination keys This disables detection of the key combinations Shift/Prt Sc, Ctrl-Break, and Ctrl-NumLock. These key combinations no longer invoke special functions. They are treated as normal key sequences. The key combinations Ctrl/Alt/Del and Ctrl/Alt/Home are not affected by this command.

Parameters	(Function D3H: Extended Codes and Functions - cont.)
Bit 1	Disable Alt compose This disables the ability to generate any character (0 to 255) by holding down the Alt key and typing the decimal value on the numeric keypad with the keypad number keys. The keypad number keys are treated as normal keys.
Bit 0	Disable keypad state keys NumLock, Insert, and Scrl Lock no longer set states. Instead, they are treated as normal characters. That is, they are translated according to the translation tables and then stored in the keyboard buffer.

Returns	
AL = 00H	Indicates a successful operation
AL = 01H	The keyboard is busy (operation failed)
AL = 02H	There was no keyboard acknowledge (operation failed)
AL = The current bit usage (AL bit 7 = 1)	

This function enables or disables various scan code conversion functions. It can also return the current bit usage.

Bit flags in the AL register select various options. Successive selections are ORed together. To clear an individual bit, clear all bits (AL = 00H) and then select the desired bits.

Function D4H: Request Keyboard ID

DIGITAL Extension

Parameters

AH = D4H

Returns

AL = 00H	Indicates a successful operation	
	BL = The LK250 keyboard's firmware version number	
	BH = 01H	Industry-standard mode
	BH = 02H	DIGITAL extended mode
AL = 01H	The keyboard is busy (operation failed)	
AL = 02H	There was no response (operation failed)	

This function returns the LK250 keyboard identification.

Function D5H: Send to Keyboard

DIGITAL Extension

Parameters

AH = D5H
AL = The value to send

Returns

AL = 00H	Indicates a successful operation
AL = 01H	The keyboard is busy (operation failed)
AL = 02H	There was no keyboard acknowledge (operation failed)

This function sends commands or data to the LK250 keyboard. It provides only the means for sending. It does not regulate what is sent. For details on the commands or data, see Chapter 8.

This function sends a single byte at a time. Use successive calls for multibyte commands or data.

Function D6H: Keyboard Table Pointers

DIGITAL Extension

Parameters

AH = D6H

AL = 00H Return the table pointer CL in ES:BX

AL = Non-zero (set table pointer CL to ES:BX)

CL = The table pointer to define or return

00H =	Normal table
01H =	Ctrl table
02H =	Alt table
03H =	Shift table
04H =	NumLock table
05H =	Lock table
06H =	Alt/Ctrl table
07H =	Alt/Shift table
08H =	Ctrl/shift table
09H =	Alphabetic table
FFH =	Set all table pointers to default value (AL = non-zero, ES:BX is ignored)

ES:BX = The pointer to table

Returns

ES:BX = The pointer to table CL (AL = 00H)

This function provides control of the pointers to the tables used in the scan code translation process. The value of any table pointer can be set or returned. Because the table pointer is a double-word pointer, each table can have a different segment address.

Under default conditions, the Alt/Ctrl and Alt/Shift table pointers point to the Alt table, and the Ctrl/Shift table pointer points to the Ctrl table.

The translation tables convert a scan code into a pair of codes, a scan code and an ASCII key value. The table used for any given translation depends on the keyboard state at the time.

Keyboard Translation Table Formats And Usage

In the following description, the term "keypad keys" refers to the keys 0 through 9, the plus key, the minus key, and the period key located on the keypad. The alphabetic table determines if a key is an alphabetic character. Only alphabetic characters are shifted by the Lock key or unshifted by the combination of Lock and Shift. The rules for table usage and precedence are:

Keys in Effect	Tables Used for Keypad Keys	Tables Used for Alphabetic Keys	Tables Used for All Other Keys
Alt	Alt	Alt	Alt
Alt and Ctrl	Alt/Ctrl	Alt/Ctrl	Alt/Ctrl
Alt and Shift	Alt/Shift	Alt/Shift	Alt/Shift
Ctrl	Ctrl	Ctrl	Ctrl
Ctrl and Shift	Ctrl/Shift	Ctrl/Shift	Ctrl/Shift
Shift	Shift	Shift	Shift
Shift and Lock	NumLock	Normal	Shift
Shift and NumLock	Normal	Shift	Shift
NumLock	NumLock	Normal	Normal
Lock	Lock	Lock	Lock
None	Normal	Normal	Normal

Except for the alphabetic and NumLock tables, the format of the table contents are the same. Each table has 105 entries with an entry being a 2-byte pair. The first byte (low byte) is the ASCII key value (function 00H returns it in AL). The second byte (high byte) is the scan code (function 00H returns it in AH). To find the correct entry in a table, subtract one from the scan code, double the result, and add that to the table pointer.

A table entry FFFFH is interpreted as an invalid key and is ignored. In Table 15-15, invalid keys are shown as - -.

The NumLock table has 13 2-byte keypad entries. They are the keys 0 through 9, the plus key, the minus key, and the period key. To find the correct entry in a NumLock table, subtract 47H from the scan code, double the result, and add that to the table pointer.

The alphabetic table is 53 bytes in size. Each byte corresponds to one of the scan codes 01H (Esc) through 35H (forward slash). If an entry has a value of 0, that key is treated as a nonalphabetic key. The Shift key and the combination Shift/Lock shifts nonalphabetic keys, but Lock key does not. If an entry has a value of FFH, that key is treated as an alphabetic key. The Shift or Lock key shifts alphabetic keys, but the combination Shift/Lock does not. To find the correct entry in an alphabetic table, subtract one from the scan code and add the result to the table pointer.

Interrupt 17H: Printer Output

Software Interrupt - Industry-Standard with DIGITAL Extensions

This interrupt provides an industry-standard software interface to the parallel printer ports. It also supports extended functionality, such as:

- Redirecting any parallel port output to any serial port
- Setting or returning the printer type associated with any port
- Setting or returning the current retry map

In accordance with industry-standard practice, the ROM BIOS code supports four parallel ports, and the ROM BIOS data area maintains four base addresses. However, due to the limited number of interrupt controller inputs, only parallel port 00H can be interrupt driven.

Initially, parallel port 00H is redirected to serial port FFH, the integral serial printer port at I/O address 0CA0H. Serial port FFH is interrupt driven through hardware interrupt vector 73H. The default conditions for the serial printer port are 4800 baud, 8 data bits, no parity, and 1 stop bit. It is also set to use XON/XOFF protocol and receive buffering. For additional information on the serial printer port, see Interrupt 14H.

Parallel port 00H can be redirected to a physical parallel port.

After redirecting parallel port 00H, the ROM BIOS looks for three physical parallel ports at I/O addresses 0378H, 03BCH, and 0278H. The first port found is assigned to logical parallel port 01H. The second port found is assigned to logical parallel port 02H. The third port found is assigned to logical parallel port 03H.

The following is a list of the available functions:

AH	Description	Digital Extended
00H	Transmit character	No
01H	Initialize printer port	No
02H	Return port status	No
D0H	Redirect parallel output	Yes
D1H	Printer type	Yes
D2H	Parallel Retry On Timeout	Yes

Function 00H: Transmit Character

Industry-Standard

Parameters

AH = 00H
AL = The character to transmit
DX = The port number (00H to 03H)

Returns

AH = The port status (as specified in function 02H)

This function transmits a character to the specified printer port. It returns current port status in AH. Refer to function 02H for the bit definitions.

If a timeout error is returned, the character was not transmitted.

Function 01H: Initialize Printer

Industry-Standard

Parameters

AH = 01H
DX = The port number (00H to 03H)

Returns

AH = The port status (as specified in function 02H)

This function initializes the specified port and associated printer. It returns current port status in AH. Refer to function 02H for the bit definitions.

If specified port is a serial device, it is initialized to 4800 baud, 8 data bits, no parity, and 1 stop bit.

Function 02H: Return Printer Status

Industry-Standard

Parameters

AH = 02H
DX = The port number (00H to 03H)

Returns

AH = The port status (set bits indicate condition)

For parallel ports:

Bit 7 - Not busy
Bit 6 - Acknowledge
Bit 5 - Out of paper
Bit 4 - Selected
Bit 3 - I/O Error
Bit 2 - Not Used
Bit 1 - Not Used
Bit 0 - Time Out

For serial ports:

Bit 7 - Not busy, serial transmitter empty or done (or serial timeout)
Bit 6 - Not Used
Bit 5 - Not Used
Bit 4 - Modem signals DSR or CTS
Bit 3 - DSR, CTS, break, framing, parity or overrun error
Bit 2 - Not Used
Bit 1 - Not Used
Bit 0 - Serial timeout

This function returns, in the AH register, the current port and printer status.

Function D0H: Redirect Parallel Printer

DIGITAL Extension

Parameters

AH = D0H

AL = 00H	Return the redirection map in DH and DL
AL = FFH	Set the redirection map according to DH and DL

DH = Parallel device selection mask
DL = Parallel to serial mapping assignment

Returns

DX = Parallel device selection mask (AL = 00H)
DL = Parallel to serial mapping assignment (AL = 00H)

This function redirects any of the four parallel ports to any of the four serial ports or the integral serial printer port FFH.

When output is redirected to a serial port, the current operational conditions associated with that serial port remain in effect. For information regarding serial port communications protocol and signal requirements, see Interrupt 14H.

The DH and DL registers each have four 2-bit fields. If a 2-bit field in DH is set to 11 (binary), the corresponding 2-bit field in DL defines the target serial port. If the 2-bit field in DH is set to 00, the corresponding 2-bit field in DL has no meaning, and that port is set to its original parallel assignment. To redirect a port, read the current map, set the desired redirection bits, and set the new map.

The four 2-bit fields in DH and DL are aligned as follows:

Port	Bits
03H	7-6
02H	5-4
01H	3-2
00H	1-0

The 2-bit fields in DH are the parallel device selection masks and are defined as follows:

Value	Description
00	The parallel port was not redirected
01	The parallel port was redirected to the integral serial printer port
10	Reserved
11	The parallel port was redirected as defined in DL

The 2-bit fields in DL are the parallel to serial mapping assignments and are defined as follows:

Value	Description
00	Assigned to logical serial port 00H
01	Assigned to logical serial port 01H
10	Assigned to logical serial port 02H
11	Assigned to logical serial port 03H

Function D1H: Printer Type

DIGITAL Extension

Parameters

AH = D1H	
AL = 00H	Return the printer types in BX, CX, and DL
AL = FFH	Set the printer types according to BX, CX, and DL

Returns

BX = Parallel printer types
CX = Serial printer types
DL = Printer type at serial printer port (bits 3-0)

This function sets or returns a code that defines the type of printer attached to any port. This allows applications to tailor the output according to the defined printer type.

This function has no effect on how the ROM BIOS handles printers. This function provides a method for applications to maintain and share printer-type information.

The printer types returned in the BX, CX, and DL registers are in 4-bit fields and aligned as follows:

Port	Bits
03H	15-12
02H	11-8
01H	7-4
00H	3-0

The following list defines the printer type assigned to each of the possible values:

Binary Value	Parallel Type	Serial Type
0000	Unknown	Unknown
0001	Industry-standard graphic	Industry-standard graphic
0010	Reserved	LA50
0011	Reserved	LA75 (DIGITAL mode)
0100	Reserved	LA75 (Industry-standard mode)
0101	Reserved	LN03
0110	Reserved	Reserved
0111	Reserved	Reserved
1000	Reserved	Reserved
1001	Reserved	Reserved
1010	Reserved	Reserved
1011	Reserved	Reserved
1100	Reserved	Reserved
1101	Not used	Not used
1110	Not used	Not used

Function D2H: Parallel Port Retry

DIGITAL Extension

Parameters

AH = D2H

AL = 01H Return the current retry map in AL

AL = Any value other than 01H is written to the retry map

Returns

AL = The current retry map (return map AL = 01H)

This function controls the ROM BIOS support of parallel port timeout errors. Each port is individually controlled by reading the current map and changing only the desired control bits. If the AL register equals 01H, the current retry map is returned in the AL register. Otherwise, the contents of the AL register are written to the retry map. Initially, all ports are set to return timeout errors. The retry map bit assignments are defined as follows.

Port	Bits	Usage
03H	7-6	00 = Return the timeout errors 11 = Loop on the timeout errors
02H	5-4	00 = Return the timeout errors 11 = Loop on the timeout errors
01H	3-2	00 = Return the timeout errors 11 = Loop on the timeout errors
00H	1-0	00 = Return the timeout errors 11 = Loop on the timeout errors

Interrupt 18H: Basic

Software Interrupt - DIGITAL Extension

Parameters

None

Returns

Nothing

The interrupt attempts to boot from the network. If that fails, interrupt 19H is invoked.

> **NOTE**
> The VAXmate workstation does not have BASIC in ROM.

Interrupt 19H: Bootstrap

Software Interrupt - DIGITAL Extension

Parameters
None

Returns
Nothing

NOTE
Because this interrupt invokes other interrupts to accomplish the bootstrap, intercepted interrupts must be restored to their original values. To know when the intercepted interrupts must be restored, intercept Interrupt 19H. On intercepting Interrupt 19H, restore any intercepted interrupts (including Interrupt 19H) and invoke Interrupt 19H.

Interrupt 19H has the following boot logic:

1. Reset interrupt 1EH to the default diskette table. Set the boot device to diskette 0. Read the first sector from diskette 0 into 0000:7C00H.
2. The contents of the boot block are examined. If the first word is 0000H or the first ten words are all equal, the diskette is not considered bootable. The boot block is not tested for AA55H in the last word.
3. If the read is successful and the contents are correct, execute a far call to 0000:7C00H.
4. If the diskette is not present or the diskette is not bootable:
   a. The CMOS RAM at offset 0EH is read. If bit 3 is 0, boot the hard disk.
   b. If there is a hard disk present, read the boot block. If the boot block is a valid DIGITAL boot block, read the boot flag. Otherwise, if the boot block is a valid industry-standard boot block, try to boot it.
   c. If the boot flag indicates boot network first or the hard disk is not present:
      (1) Attempt to boot from network.
      (2) If that fails and hard disk is present, attempt to boot hard disk.
      (3) If that fails, go to the start of the process and try again.
   d. If the flag indicates boot hard disk first or boot block is not a DIGITAL boot block:

(1) Attempt to boot from the hard disk.

(2) If that fails, attempt to boot from network.

(3) If that fails, go to the start of the process and try again.

This process loops 22 times. If the system has not booted, it drops into a keyboard loop and waits for the key combination Ctrl/Alt/Del. On receiving that key combination, the ROM BIOS executes Interrupt 19H and restarts the boot process.

DIGITAL Hard Disk Boot Block

The hard disk boot sector is located at cylinder 0, head 0, sector 1. It consists of the following:

Offset	Description
0000H-019FH	The boot code
01A0H	The 16-byte disk parameter table This table is valid only if the DIGITAL signature contains 0DECH and bit 1 of BOOT FLAGS is set.
01B0H-01B9H	Reserved
01BAH	16 flag bits (unused bits are set to 0) Bit 1 - If set, parameter block is valid. Bit 0 - If set, attempt boot from network first.
01BCH	The DIGITAL signature A value of 0DECH indicates that the boot block contains valid DIGITAL data.
01BEH	The 32-word industry-standard partition table
01FEH	The industry-standard boot signature A value of AA55H indicates a valid boot block.

Interrupt 1AH: Time-of-day

Software Interrupt - Industry-Standard with DIGITAL Extensions

The functions in this interrupt read and set the system clock and read or set the real-time clock.

The system clock frequency is 18.20648 Hz or 1573040 ticks per day.

The industry-standard 24-hour overflow flag is invalid if more than 48 hours elapse between reads. The ROM BIOS provides a days-since-read counter that can count up to 255 days between reads.

> **NOTE**
> The system clock is read or written in timer ticks (1573040 ticks per day) and is a binary value. The real-time clock is read or written using time measures like month, hours, minutes, and seconds. However, the value is in binary coded decimal.

> **NOTE**
> Execution of interrupt 1AH, functions 00H, 01H or D0H, clears the 24-hour overflow flag and the days-since-read counter.

The following is a list of the available functions:

AH	Description	Digital Extended
00H	Read the system clock	No
01H	Set the system clock	No
02H	Read the real-time clock	No
03H	Set the real-time clock	No
04H	Return the RTC date	No
05H	Set the RTC date	No
06H	Set the alarm	No
07H	Cancel the alarm	No
D0H	Return the days-since-read counter	Yes

Function 00H: Read System Clock

Industry-Standard

Parameters

AH = 00H

Returns

AL = 00H 24-hour overflow has not occurred
AL = 01H 24-hour overflow occurred

CX = High-order 16 bits of elapsed-time
DX = Low-order 16 bits of elapsed-time

This function returns the system elapsed-time. The elapsed-time is a 32-bit value measured in timer ticks from the last time written. At power-up, the elapsed-time is set to 0.

Function 01H: Set System Clock

Industry-Standard

Parameters

AH = 01H
CX = High-order 16 bits of elapsed-time
DX = Low-order 16 bits of elapsed-time

Returns

Nothing

This function sets the system elapsed-time clock. The elapsed-time is a 32-bit value measured in timer ticks from the last time written. At power-up, the elapsed-time is set to 0.

Function 02H: Read Real-Time Clock

Industry-Standard

Parameters

AH = 02H

Returns

CH = Hours (BCD)
CL = Minutes (BCD)
DH = Seconds (BCD)

This function returns the time from the real-time clock. The returned values are in binary coded decimal.

> **NOTE**
> If an expansion box is installed, the battery-backed clock maintains the current date and time. Otherwise, the time is relative to the last time it was written since power-up (at power-up, the time is set to 0).

Function 03H: Set Real-Time Clock

Industry-Standard

Parameters

AH = 03H
CH = Hours (BCD)
CL = Minutes (BCD)
DH = Seconds (BCD)
DL = The daylight savings flag
    0 = Do not use daylight savings
    1 = Use daylight savings

Returns

Nothing

This function sets the real-time clock. The parameters are in binary coded decimal.

Function 04H: Return RTC Date

Industry-Standard

Parameters

AH = 04H

Returns

CH = Century (19 or 20 in BCD)
CL = Year (00 to 99 in BCD)
DH = Month (BCD)
DL = Day (BCD)

This function returns the real-time clock date. The parameters are in binary coded decimal.

Function 05H: Set RTC Date

Industry-Standard

Parameters

AH = 05H
CH = Century (19 or 20 in BCD)
CL = Year (00 to 99 in BCD)
DH = Month (BCD)
DL = Day (BCD)

Returns

Nothing

This function sets the real-time clock date. The parameters are in binary coded decimal.

Function 06H: Set Alarm

Industry-Standard

Parameters

AH = 06H
CH = Hours (BCD)
CL = Minutes (BCD)
DH = Seconds (BCD)

Returns

CF = 0	No previous alarm
CF = 1	The previous alarm was not canceled, this alarm not set

This function sets the real-time clock alarm. At the specified time of the day, an alarm interrupt is issued by the real-time clock. The ROM BIOS handles the alarm interrupt and executes interrupt 4AH. The application should set interrupt vector 4AH to point to the alarm service. The parameters are in binary coded decimal.

Function 07H: Cancel Alarm

Industry-Standard

Parameters

AH = 07H

Returns

Nothing

This function cancels the real-time clock alarm.

Function D0H: Return Days-Since-Read Counter

DIGITAL Extension

Parameters

AH = D0H

Returns

AL = The contents of days-since-read counter
CX = High-order 16 bits of elapsed-time
DX = Low-order 16 bits of elapsed-time

This function returns the contents of the days-since-read counter and the elapsed-time. The days-since-read counter indicates the number of 24-hour periods that have passed since any of Interrupt 1AH functions 00H, 01H, or D0H was executed. The elapsed-time is a 32-bit value measured in timer ticks.

This function is similar to function 00H, except that AL returns days-since-read instead of the 24-hour overflow flag. The 24-hour overflow flag indicates only that 24 hours or more have elapsed.

Executing this function clears the 24-hour overflow flag and the days-since-read counter.

Interrupt 1BH: Keyboard Break

Software Interrupt - Industry-Standard

Parameters

None

Returns

Nothing

Whenever the key combination Ctrl/Break is typed at the keyboard, the routine pointed to by this vector is executed by interrupt 09H. Use an IRET instruction to return control to interrupt 09H. The ROM BIOS initializes this vector to point to a ROM BIOS IRET.

Interrupt 1CH: Timer Tick

Software Interrupt - Industry-Standard

Parameters

None

Returns

Nothing

When a system clock interrupt occurs, the ROM BIOS executes Interrupt 1CH. Thus, the routine pointed to by this vector is executed. Use an IRET instruction to return control to the ROM BIOS. The ROM BIOS initializes this vector to point to a ROM BIOS IRET. For additional information, see Interrupt 08H.

Interrupt 1DH: Video Parameters

Pointer - Industry-Standard

This vector points to a table, not executable code.

This vector is not a true interrupt. This table is 16 bytes long and corresponds directly with the registers R0 through R15 of the 6845 video controller. For additional information on the video controller, see Chapter 7.

Interrupt 1EH: Diskette Parameter Tables

Pointer - Industry-Standard

This vector points to a table, not executable code.

A diskette parameter table defines the physical characteristics of a diskette. The values in the table are used by the diskette driver to initialize the diskette controller. Table 15-16 describes the contents of a diskette parameter table. Each parameter in Table 15-16 is one byte long.

The interrupt vector for interrupt 1EH points to the diskette parameter table. If a diskette drive does not exist, the interrupt vector for interrupt 1EH is reserved and undefined.

Table 15-16 Diskette Parameter Table Description

Offset	Bits	Description
00H	7-4	Step rate Each increase in the value of bits 7-4 decreases the step rate by 1 ms, so that zero equals 16 ms, one equals 15 ms, two equals 14 ms, and so on.
	3-0	Head unload time Each increase in the value of bits 3-0 increases the head unload time by 16 ms, so that zero equals 16 ms, one equals 32 ms, two equals 48 ms, and so on.
01H	7-1	Head load time Each increase in the value of bits 7-1 increases the head load time by 2 ms, so that zero equals 2 ms, one equals 4 ms, two equals 6 ms, and so on.
	0	Direct Memory Access (DMA) selection 0 = Do not use DMA 1 = Use DMA mode
02H	7-0	Clock ticks until motor is turned off

Table 15-16 Diskette Parameter Table Description (cont.)

Offset	Bits	Description
03H	7-0	Sector size Each increase in value doubles the sector size, so that zero equals 128 bytes, one equals 256 bytes, two equals 512 bytes, and so on. The default is value is two (512 bytes).
04H	7-0	Sectors per track (8, 9, 10, or 15)
05H	7-0	Sector gap length (1BH)
06H	7-0	Data length (FFH)
07H	7-0	Format gap length (54H)
08H	7-0	Format fill byte (F6H)
09H	7-0	Head settle time in milliseconds
0AH	7-0	Motor start-up time in .125 second increments

Interrupt 1FH: Graphics Character Table Pointer

Pointer - Industry-Standard with DIGITAL Extension

This vector points to a table, not executable code. It points to a character table for generation of character codes 80H through FFH. Interrupt 10H, function D0H extends the functionality of this pointer. If enabled, this vector points character table for generation of character codes 00H through FFH.

Interrupt 40H: Revector of Interrupt 13H

Software Interrupt - Industry-Standard

Normally, the diskette I/O is serviced through interrupt 13H. When a hard disk is installed, the hard disk I/O is serviced through interrupt 13H, and the diskette I/O service is revectored to Interrupt 40H. This revectoring information is provided only for clarity. Always use Interrupt 13H for both diskette and hard disk functions.

Interrupt 41H and 46H: Hard Disk Parameter Tables

Pointer - Industry-Standard

This vector points to a table, not executable code.

A hard disk parameter table defines the physical characteristics of a hard disk. The values in the table are used by the hard disk driver to initialize the hard disk controller. Table 15-17 describes the contents of a hard disk parameter table.

The hard disk parameter tables are located in DIGITAL private RAM. During the power-up sequence, the disk type is extracted from CMOS RAM. If the disk type is unknown, the table contains all zeros. If the disk type is one of the 14 industry-standard types, the table is initialized from the hard disk data in the ROM BIOS. If the disk type is the DIGITAL extended type 0FH, the ROM BIOS expects the boot block to contain the parameters. The ROM BIOS initializes the table with data extracted from the boot block. (As part of its initialization process, the FDISK utility writes the parameters in the boot block.)

The interrupt vectors for Interrupt 41H and 46H point to the hard disk parameter tables for hard disk 0 and hard disk 1, respectively. If hard disk 1 does not exist, the interrupt vector for Interrupt 46H is reserved and undefined.

Table 15-17 Hard Disk Parameter Table Description

Offset	Size	Description
00H	1 Word	Maximum number of cylinders on hard disk drive
02H	1 Byte	Maximum number of heads on hard disk drive
03H	1 Word	Not used
05H	1 Word	Cylinder number to start using write precompensation
07H	1 Byte	Not used
08H	1 Byte	Control byte sent to controller
		If bit 7 or bit 6 is set (1), disable retries
		If bit 3 is set (1), hard disk has more than eight heads
09H	3 Bytes	Not used
0CH	1 Word	Landing zone
0EH	1 Byte	Number of sectors per track
0FH	1 Byte	Reserved for future use

Interrupt 4AH: RTC Alarm

Software - Industry-Standard

When a real-time clock alarm occurs, the ROM BIOS executes Interrupt 4AH. Thus, the routine pointed to by this vector is executed. Use an IRET instruction to return control to the ROM BIOS. For additional information, see Interrupt 1AH (functions 06H and 07H).

Interrupt 70H: Real-Time Clock

Hardware Interrupt - Industry-Standard

Interrupt 70H provides the ROM BIOS with hardware-interrupt services for the real-time clock. This interrupt monitors the periodic interrupt and the alarm interrupt.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. For informtion about the real-time clock, see Chapter 5. For information about the ROM BIOS clock services, see Interrupt 1AH.

Interrupt 71H: Redirect to Interrupt 0AH

Hardware Interrupt - Industry-Standard

This interrupt redirects the IRQ9 hardware interrupt to Interrupt 0AH, which is the old IRQ2 hardware interrupt. IRQ9 is an available interrupt input and Interrupt 0AH is an available interrupt vector.

Interrupt 72H: Local Area Network Controller (LANCE)

Hardware Interrupt - DIGITAL Extension

Interrupt 72H provides the network software with hardware-interrupt services for the LANCE. This interrupt service provides an operation complete indication from the LANCE.

Interrupt 72H has no arguments, preserves all registers, and returns no values.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. For more information on the LANCE, see Chapter 13. For more information about network I/O services, see the *VAXmate Network Technical Reference Manual.*

Interrupt 73H: Serial Printer Port

Hardware Interrupt - Industry-Standard

Interrupt 73H provides the ROM BIOS with hardware-interrupt services for the asynchronous serial printer port. This interrupt service monitors the state of the serial communications protocol. It also transmits and receives characters as required.

Interrupt 73H has no arguments, preserves all registers, and returns no values.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. For information on the 8250A serial communications device, see Chapter 9. For information about the ROM BIOS asynchronous communications service, see Interrupt 14H.

Interrupt 74H: Mouse Port

Hardware Interrupt - DIGITAL Extension

Interrupt 74H provides the mouse driver with hardware interrupt services. The mouse driver interrupt handler is present only when the mouse driver has been loaded under MS-DOS.

Using this interrupt requires knowledge of the VAXmate hardware. For information on the mouse, see Chapter 10.

Interrupt 75H: 80287 Error

Hardware Interrupt - Industry-Standard

Interrupt 75H provides MS-DOS with hardware interrupt services for 80287 math coprocessor errors.

Interrupt 76H: Hard Disk

Hardware Interrupt - Industry-Standard

Interrupt 76H provides the ROM BIOS with hardware-interrupt services for the hard disk controller. This interrupt service provides a operation complete indication from the hard disk controller.

Interrupt 76H has no arguments, preserves all registers, and returns no values.

Using this interrupt requires knowledge of the VAXmate hardware, the ROM BIOS, and the operating system. For information on the hard disk controller see Chapter 12. For more information about the ROM BIOS hard disk I/O service, see Interrupts 13H, 14H, and 46H.

Interrupt 77H: Available (IRQ15)

Hardware Interrupt - Industry-Standard

Interrupt 77H is available for undefined uses related to IRQ15.

Chapter 16
Programming the VAXmate Under MS-DOS

Overview

Microsoft Disk Operating System (MS-DOS) is the operating system for microcomputers using Intel 8086 and 8088 microprocessors. An operating system, a program that controls the overall operation of a computer, provides an environment within the computer that enables the user to easily perform operations such as:

- Program execution
- File management
- Resource management
- Programming
- Device handling

The MS-DOS operating system provides functions for commonly-used operations and I/O operations that are hardware independent. Therefore, a user can write an application program to run under the MS-DOS operating system without a detailed knowledge of the computer hardware. Such a program runs on any computer that can run the MS-DOS operating system, as long as the computer has the appropriate peripherals. MS-DOS provides all the logical operations necessary for writing to and reading from disk storage devices.

MS-DOS Operating System Versions

The MS-DOS operating system has evolved through a number of versions. Microsoft's Version 3.10 is the base for DIGITAL's VAXmate Version 3.10. DIGITAL added many new features, modifications, and utilities to Microsoft's base version. Some new features are:

- ANSI, which replaces the normal console device with an ANSI escape sequence parser.
- MDRIVE, which enables a user to use RAM as a fast logical disk drive.
- FDISK, which manages hard disks.
- International support in MS-DOS Interrupt 21H Function 38H
- FONT, which enables a user to load file-based text font sets.
- GRAFTABL, which enables a user to load file-based graphics font sets.
- GRAPHICS, which enables a user to print screen images on DIGITAL and industry standard printers.
- KEYB, which enables a user to load file-based keyboard maps.
- LCOUNTRY, which enables a user to load file-based country data sets.
- SORT, which coordinates the sort set with the current font set.

Programs written to run under IBM's DOS Version 3.10 and later will run under DIGITAL's version of MS-DOS.

Programs written for the Rainbow that use only generic MS-DOS Interrupt 21H calls and do not make ROM calls directly can be run under MS-DOS Version 3.10 for the VAXmate. An example is the Microsoft C Compiler and Linker.

MS-DOS Version 3.10 runs only in the real mode of the 80286 processor.

Loading MS-DOS Operating System

MS-DOS Memory Map

Chapter 2 contains an MS-DOS memory map.

MS-DOS Interrupt 21H Digital Specific Functions

Function 30H Get MS-DOS OEM Number

Function 30H returns the OEM serial number in register BH. This number is 16H for DIGITAL's version of MS-DOS, and 00H for IBM's DOS.

Function 38H Get/Set Country Code

To accommodate the VAXmate's ability to load file-based character fonts and keyboard maps, MS-DOS saves the current font set and keyboard map in a text string.

To retrieve the pointer to the data area containing the file name of the current font file and keyboard map, do an INT 21H with the following values in the specified registers:

Parameters

AH =	38H	
	AL =	0FFH
	BX =	8000H
	DS:DX =	points to a double word address that is loaded with the address of the beginning of the current country-specific table

Returns

DS:DX =	32-bit pointer to the MS-DOS date and time structure containing the current font file and keyboard map

After retrieving the double word pointer to the date and time structure, information about the current font file and keyboard map can be retrieved, changed or set.

At offset -1AH of the date and time structure, a 14-byte buffer contains the name of the current keyboard file. When a new keyboard file is loaded, this string must be replaced with the complete file name of the newly-loaded keyboard file, padded with zeros.

At offset -34H of the date and time structure, a 14-byte buffer contains the name of the current text font file. When a new font file is loaded, this string must be replaced with the complete file name of the newly-loaded font file, padded with zeros.

Figure 16-1 shows the MS-DOS date and time structure.

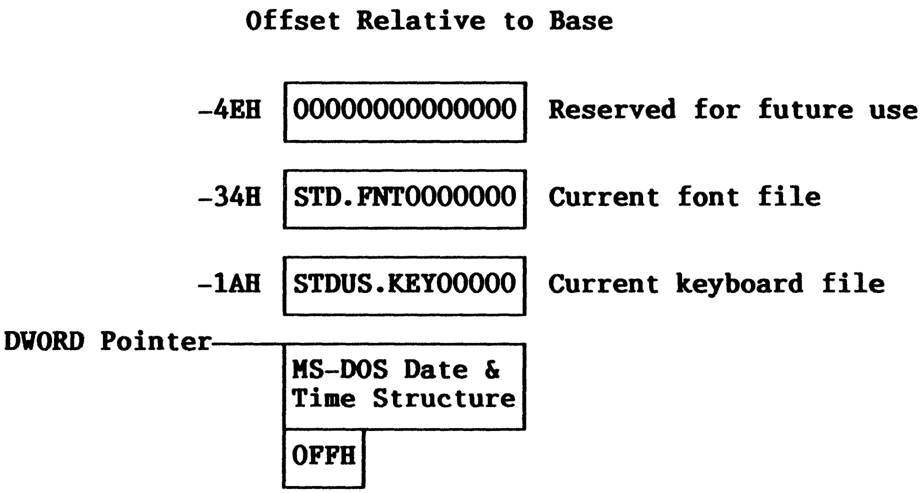

Figure 16-1 MS-DOS Date and Time Structure

Loadable MS-DOS Device Drivers

ANSI.SYS

ANSI.SYS is an MS-DOS loadable device driver that replaces the standard console device driver. An ANSI escape sequence is a series of characters (beginning with an escape character or keystroke) that defines the following functions to the MS-DOS operating system:

- Cursor control functions
- Erase functions
- Set graphics rendition function
- Set mode function
- Reset mode function
- Keyboard key reassignment function

ESC is the one byte ASCII code 1BH.

The size of the MS-DOS operating system increases by the size of ANSI.SYS.

For more information about the ANSI X3.64-1979 standard, see **Information Systems-Codes**, which is available through the Order Department of the American National Standards Institute, 1430 Broadway, New York, NY, 10018.

Installing ANSI.SYS

To install the ANSI.SYS device driver, insert the following line in a CONFIG.SYS file:

```
DEVICE=ANSI.SYS
```

Cursor Control Functions

Cursor control functions affect the position of the cursor on the screen. Table 16-1 contains the escape sequences to position the cursor on the screen. The # indicates a string of decimal digits. For example, ESC [10A moves the cursor up 10 rows. Row 1 refers to the first line of the screen; row 2 refers to the second line of the screen, etc.

Table 16-1 Cursor Control Functions

Sequence	Function
ESC [#;#H	This sequence moves the cursor to the specified position. The first # specifies the row. The second # specifies the column. The default value is 1. If no parameters are specified, the cursor moves to the upper left hand corner of the screen. Some error checking is done. If illegal values are specified, garbage is displayed on the screen.
ESC [#A	This sequence moves the cursor up one or more rows without changing the column position. The value of # determines the number of rows moved. The default value is 1. If the cursor is already on the top row or reaches the top row during this operation, this sequence is ignored.
ESC [#B	This sequence moves the cursor down one or more rows without changing the column position. The value of # determines the number of rows moved. The default value is 1. If the cursor is already on the bottom row or reaches the bottom row during this operation, this sequence is ignored.
ESC [#C	This sequence moves the cursor forward one or more columns without changing the row position. The value of # determines the number of columns moved. The default value is 1. If the cursor is already in the right-most column or reaches it during this operation, this sequence is ignored.
ESC [#D	This sequence moves the cursor back one or more columns without changing the row position. The value of # determines the number of columns moved. The default value is 1. If the cursor is already in the left-most column or reaches it during this operation, this sequence is ignored.
ESC [#;#f	This sequence moves the cursor to the position specified by #;#. The first # specifies the row number. The second # specifies the column. The default value for both is 1. If no parameters are specified, the cursor moves to the upper left hand corner of the screen.

Table 16-1 **Cursor Control Functions (cont.)**

Sequence	Function
ESC [#;#R	This sequence reports the current cursor position through the standard input device. The first # specifies the current row. The second # specifies the current column. This is not a command, but a "Cursor Position Report." This sequence is received after issuing the "Device Status Report" command.
ESC [6n	This sequence is the Device Status Report. ANSI.SYS outputs a "Cursor Position Report" when it receives this escape sequence.
ESC [s	This sequence saves the current cursor position.
ESC [u	This sequence restores the cursor to the value it had when ANSI.SYS received the last "Save Cursor Position" escape sequence. ESC [s is the "Save Cursor Position."

Erase Functions

Erase functions erase characters from part or all of the screen. Table 16-2 contains the escape sequences to erase text from the screen.

Table 16-2 **Erase Functions**

Sequence	Function
ESC [2J	This sequence erases the entire screen. The cursor moves to the upper left hand corner of the screen.
ESC [K	This sequence erases from the cursor to the end of the line, including the cursor position.

Set Graphics Rendition

The set graphics rendition function invokes the graphics rendition specified by the parameters. All of the following characters are rendered according to the parameters until the next set graphics rendition. Table 16-3 contains the escape sequence and parameters to set graphics rendition.

Table 16-3 Set Graphics Rendition Function

Sequence	Function		
ESC [#;...;#m	Invokes the graphics rendition specified by the parameters.		
	Parameter	Meaning	Notes
	0	All attributes off	Normal white on black
	1	Bold on	High intensity
	4	Underscore on	Monochrome adapter only
	5	Blink on	
	7	Reverse video on	
	8	Canceled on	
	30	Black foreground	
	31	Red foreground	
	32	Green foreground	
	33	Yellow foreground	
	34	Blue foreground	
	35	Magenta foreground	

Table 16-3 Set Graphics Rendition Function (cont.)

Sequence	Function		
	Parameter	Meaning	Notes
	36	Cyan foreground	
	37	White foreground	
	40	Black background	
	41	Red background	
	42	Green background	
	43	Yellow background	
	44	Blue background	
	45	Magenta background	
	46	Cyan background	
	47	White background	

Set Mode Function

The set mode function sets screen width and screen display. Table 16-4 contains the escape sequence and parameters to set the mode.

Table 16-4 Set Mode Function

Sequence	Function
ESC [=#h	This sequence invokes the screen width or type specified by the parameter.
ESC [?#h	
	The ? and = are interchangeable in this command.
	ESC [=h assumes a parameter value of zero.
	Parameter — Meaning
	0 — 40x25 black and white
	1 — 40x25 color
	2 — 80x25 black and white
	3 — 80x25 color
	4 — 320x200 color
	5 — 320x200 black and white
	6 — 640x200 black and white
	7 — Wrap at end of line

Reset Mode Function

The reset mode function resets the screen width and screen display. Table 16-5 contains the escape sequence and parameters to reset the mode.

Table 16-5 Reset Mode Function

Sequence	Function	
ESC [=#l	This sequence resets the screen width or type specified by the parameter.	
ESC [?#l		
	The ? and = are interchangeable in this command.	
	ESC [=l assumes a parameter value of zero.	
	Parameter	Meaning
	0	40x25 black and white
	1	40x25 color
	2	80x25 black and white
	3	80x25 color
	4	320x200 color
	5	320x200 black and white
	6	640x200 black and white
	7	No wrap at end of line. Type past end of line is lost.

Keyboard Key Reassignment Function

The keyboard key reassignment function intercepts a key and redefines it. Table 16-6 contains the escape sequence to redefine the meaning of the intercepted keyboard key.

Table 16-6 Keyboard Key Reassignment Function

Sequence	Function
ESC [#;#;...#p	
ESC ["string"p	
ESC [#;"string";#;#;"string";...p	The first ASCII code in the sequence defines which code is being mapped. The remaining numbers define the sequence of ASCII codes generated when this key is intercepted. If the first code in this sequence is 0, the first and second codes make up an extended ASCII redefinition.

Mouse Driver

With the MS-DOS operating system, DIGITAL provides a mouse driver, MOUSE.SYS or MOUSE.COM. The mouse driver provides the following features:

- Automatic tracking of motion and button events
- Optional, automatic cursor management for text and graphic video modes
- Synchronous and asynchronous handling of mouse related events

The mouse driver provides the standard functions listed in Table 16-7 and the extended functions listed in Table 16-8. Assuming that the mouse driver is present in memory, executing software interrupt 33H invokes the mouse driver. The contents of register AX specify the desired function. The mouse functions are defined later in the chapter.

Table 16-7 Standard Mouse Driver Functions

Function	Description
0000H	Mouse initialization
0001H	Show cursor
0002H	Hide cursor
0003H	Get mouse position and button status
0004H	Set mouse cursor position
0005H	Get button press information
0006H	Get button release information
0007H	Set minimum and maximum X-axis position
0008H	Set minimum and maximum Y-axis position
0009H	Define graphics cursor
000AH	Define text cursor
000BH	Read mouse motion counters
000CH	Define event handler
000DH	Light pen emulation mode on
000EH	Light pen emulation mode off
000FH	Set mouse motion/pixel ratio
0010H	Conditional hide cursor
0013H	Set speed threshold

Table 16-8 Extended Mouse Driver Functions

Function	Description
001CH	Get driver version
0024H	Get configuration
0025H	Set configuration
0026H-002BH	Reserved
004DH	Reserved
006DH	Reserved

Detecting the Mouse Driver

To determine if the mouse driver is present in memory, load the interrupt vector at interrupt 51H, add 0000:0010H to that memory address, and examine the resulting address. If the mouse driver is present, that address and the following locations should contain the following text string:

DIGITAL/LOGITECH MOUSE DRIVER V *X.xx*<LF><CR>

Where *X* represents the release number and *xx* represents the sub-release number. The string is terminated by a linefeed and carriage return.

Video Support

To display a cursor, the mouse driver logic must contain information regarding the video system and the available modes. Table 16-9 defines the supported video systems and modes.

To provide the correct cursor at all times, the mouse driver monitors calls to interrupt 10H (video I/O). To monitor interrupt 10H, the mouse driver intercepts interrupt 10H function calls. Therefore, applications that intercept interrupt 10H function calls must invoke the mouse driver through the previous interrupt vector.

Additionally, the mouse driver guarantees consistency of the screen contents only if the screen is updated through interrupt 10H. Programs that write directly to video memory should invoke the mouse driver as follows:

1. Before writing directly to video memory, invoke function 0002H (hide cursor) or function 0010H (conditional hide cursor).
2. After writing directly to video memory, invoke function 0001H (show cursor).

Table 16-9 Video Systems and Modes Supported by MOUSE.SYS

Video Mode	Size	DIGITAL Video System	Color Graphics Adapter	Enhanced Graphics Adapter
Text (monochrome)	40 X 25	Yes	Yes	Yes
	80 X 25	Yes	Yes	Yes
Text (color)	40 X 25	Yes	Yes	Yes
	80 X 25	Yes	Yes	Yes
Graphics (monochrome)	320 X 200	Yes	Yes	Yes
	640 X 200	Yes	Yes	Yes
	640 X 350	No	No	Yes
	640 X 400	Yes	No	No
Graphics (4-color)	320 X 200	Yes	Yes	Yes
	640 X 400	Yes	No	No
	800 X 252	Yes	No	No
Graphics (16-color)	640 X 350	No	No	Yes

Function 0000H: Mouse Initialization

Industry-Standard

Parameters

AX = 0000H

Returns

AX = 0000H	Mouse hardware or driver not installed
AX = FFFFH	Mouse hardware and driver installed

BX = The number of logical buttons (see function 25H)

This function resets the driver and hides the mouse cursor. The hide-cursor count is set to -1. To display the mouse cursor, execute a show-cursor command (function 0001H).

This function also restores the following parameters to the default value:

Parameter	Default Value
Cursor position	Center of screen
Hide-cursor counter	-1 (mouse cursor hidden)
Graphics cursor	Arrow
Text cursor	Inverted box
Interrupt call mask	Disabled (equals 0)
Light pen emulation	Enabled
Horizontal mouse encode count to pixel ratio	8 to 8
Vertical mouse encode count to pixel ratio	16 to 8
Horizontal minimum and maximum cursor position	0 to 639
Vertical minimum and maximum cursor position	0 to 199

Function 0001H: Show Cursor

Industry-Standard

Parameters
AX = 0001H

Returns
Nothing

The show-cursor function increments a software counter. After incrementing the counter, the show-cursor function tests the software counter contents. If the software counter equals 0, the show-cursor function displays the mouse cursor.

Function 0002H: Hide Cursor

Industry-Standard

Parameters
AX = 0002H

Returns
Nothing

The hide-cursor function disables (hides) the mouse cursor and decrements a software counter. For the show-cursor command, the mouse driver displays the mouse cursor only if the software counter equals 0. To redisplay the mouse cursor, execute a show-cursor command for each hide-cursor command executed since the mouse cursor was last visible.

Function 0003H: Get Mouse Position and Button Status

Industry-Standard

Parameters

AX = 0003H

Returns

BX =	Button status	
	Bits 15-3	Always 0
	Bit 2	Middle button 0 = Button released 1 = Button pressed
	Bit 1	Right button 0 = Button released 1 = Button pressed
	Bit 0	Left button 0 = Button released 1 = Button pressed
CX =	The horizontal cursor position	
DX =	The vertical cursor position	

The cursor position is a signed 16-bit positive value. When the cursor position is outside of the pixel range permitted by the current video mode or at the limits specified by functions 0007H and 0008H, the mouse cursor is not displayed. The returned cursor position is always within the pixel range permitted by the current video mode or the limits specified by functions 0007H and 0008H.

Function 0004H: Set Mouse Cursor Position

Industry-Standard

Parameters

AX = 0004H
CX = Desired horizontal cursor position
DX = Desired vertical cursor position

Returns

Nothing

The cursor position is a signed 16-bit positive value. Negative cursor position values can cause unpredictable mouse-cursor behavior. When the cursor position is outside of the pixel range permitted by the current video mode or at the limits specified by functions 0007H and 0008H, the mouse cursor is not displayed. The values specified in registers CX and DX must be within the limits specified by functions 0007H and 0008H.

Function 0005H: Get Button Press Information

Industry-Standard

Parameters

AX =	0005H
BX =	Button 0 = Return status of left button 1 = Return status of right button 2 = Return status of middle button

Returns

AX =	Button status	
	Bits 15-3	Always 0
	Bit 2	Middle button 0 = Button released 1 = Button pressed
	Bit 1	Right button 0 = Button released 1 = Button pressed
	Bit 0	Left button 0 = Button released 1 = Button pressed
BX =	The number of times the button has been pressed since its status was checked (in the range 0 - 32767)	
CX =	The horizontal cursor position the last time the button was pressed	
DX =	The vertical cursor position the last time the button was pressed	

The cursor position is a signed 16-bit positive value. When the cursor position is outside of the pixel range permitted by the current video mode or at the limits specified by functions 0007H and 0008H, the mouse cursor is not displayed. The returned cursor position is always within the pixel range permitted by the current video mode or the limits specified by functions 0007H and 0008H.

Function 0006H: Get Button Release Information

Industry-Standard

Parameters

AX =	0006H	
BX =	Button	
	0 =	Return status of left button
	1 =	Return status of right button
	2 =	Return status of middle button

Returns

AX =	Button status	
	Bits 15-3	Always 0
	Bit 2	Middle button 0 = Button released 1 = Button pressed
	Bit 1	Right button 0 = Button released 1 = Button pressed
	Bit 0	Left button 0 = Button released 1 = Button pressed
BX =	The number of times the button has been released since its status was checked (in the range 0 - 32767)	
CX =	The horizontal cursor position the last time the button was released	
DX =	The vertical cursor position the last time the button was released	

The cursor position is a signed 16-bit positive value. When the cursor position is outside of the pixel range permitted by the current video mode or at the limits specified by functions 0007H and 0008H, the mouse cursor is not displayed. The returned cursor position is always within the pixel range permitted by the current video mode or the limits specified by functions 0007H and 0008H.

Function 0007H: Set Minimum and Maximum X-Axis Position

Industry-Standard

Parameters

AX = 0007H
CX = Minimum horizontal cursor position
DX = Maximum horizontal cursor position

Returns

Nothing

The cursor position is a signed 16-bit positive value. Negative cursor position values can cause unpredictable mouse-cursor behavior. When the cursor position is outside of the pixel range permitted by the current video mode or at the limits specified by functions 0007H and 0008H, the mouse cursor is not displayed. For functions that return cursor position information, the returned cursor position is always within the pixel range permitted by the current video mode or the limits specified by functions 0007H and 0008H.

> NOTE
> When an application starts running, it is the responsibility of the application to set the desired horizontal and vertical limits. If an application changes video modes, it is the responsibility of the application to set the desired horizontal and vertical limits.

If the cursor is outside the limits defined by registers CX and DX, the cursor is moved to a position that is at the new limit.

Function 0008H: Set Minimum and Maximum Y-Axis Position

Industry-Standard

Parameters

AX =	0008H
CX =	Minimum vertical cursor position
DX =	Maximum vertical cursor position

Returns

Nothing

The cursor position is a signed 16-bit positive value. Negative cursor position values can cause unpredictable mouse-cursor behavior. When the cursor position is outside of the pixel range permitted by the current video mode or at the limits specified by functions 0007H and 0008H, the mouse cursor is not displayed. For functions that return cursor position information, the returned cursor position is always within the pixel range permitted by the current video mode or the limits specified by functions 0007H and 0008H.

> **NOTE**
> When an application starts running, it is the responsibility of the application to set the desired horizontal and vertical limits. If an application changes video modes, it is the responsibility of the application to set the desired horizontal and vertical limits.

If the cursor is outside the limits defined by registers CX and DX, the cursor is moved to a position that is at the new limit.

Function 0009H: Define Graphics Cursor

Industry-Standard

Parameters

AX = 0009H
BX = Horizontal focal point of cursor (hot spot)
CX = Vertical focal point of cursor (hot spot)
DX = Pointer to the screen and cursor masks (assumes DS:DX)

Returns

Nothing

The graphics cursor is defined by two arrays of bit masks called the *screen mask* and the *cursor mask*. Each array has a fixed size of 16 words. The two arrays are contigous, with the cursor mask following the screen mask. The following C structure declares storage for a graphics cursor:

```
typedef struct
{
  unsigned int screen_mask[16];
  unsigned int cursor_mask[16];
} GRAPHICS_CURSOR;

GRAPHICS_CURSOR gc;                            /* ds:dx = &gc */
```

The screen mask determines which cursor mask bits are background (0 value screen mask bits) or foreground (1 value screen mask bits). The screen mask is *ANDed* with the screen contents.

The cursor mask determines shape/color of the cursor. After the screen mask is *AND*ed with the screen contents, the cursor mask is *XORed* with the the results.

The horizontal and vertical focal points of the cursor are signed 16-bit values in the range, -16 to 16. The focal point of the default cursor is the upper left corner of the cursor (the hot spot).

The following list of screen mask and cursor mask bit values define the resulting value of the screen pixel:

Screen Mask Bit	Cursor Mask Bit	Resulting Screen Pixel
0	0	0
0	1	1
1	0	Unchanged
1	1	Inverted

Function 000AH: Define Text Cusor

Industry-Standard

Parameters

AX =	000AH
BX =	Select hardware or software text cursor 0 = Software text cursor selected CX = Screen mask DX = Cursor mask 1 = Hardware text cursor selected CX = Start scan line DX = End scan line

Returns

Nothing

The mouse driver supports two kinds of text cursor, a hardware cursor and a software cursor. The hardware cursor is the same as the cursor defined in interrupt 10H (video I/O). The software cursor is a character or character attribute that modifies the character cell at the cursor location. The software cursor is defined by the *screen mask* and *cursor mask* in the CX and DX registers respectively. The format of the two masks is as follows:

Bit	Description
15	Blink 0 = Nonblinking character 1 = Blinking character
14-12	Background color
11	Intensity 0 = Medium intensity 1 = High intensity
10-8	Foreground color
7-0	Character code

After the screen mask is *AND*ed with the screen contents, the cursor mask is *XOR*ed with the results.

Function 000BH: Read Mouse Motion Counters

Industry-Standard

Parameters
AX = 000BH

Returns
CX = Horizontal count DX = Vertical count

This function returns the incremental distance traveled by the mouse since the last time this function was called. After this function loads the software counters into registers CX and DX, it clears the software counters.

The unit of measurement for the values in registers CX and DX are mouse encoder counts and are in the range -32768 to 32767 (overflow is ignored). A positive horizontal count indicates motion to the right. A positive vertical count indicates motion towards the bottom of the screen.

The mouse encoder resolution is 200 counts per inch of travel. For additional information about the mouse, see Chapter 10.

Function 000CH: Define Event Handler

Industry-Standard

Parameters

AX = 000CH	
CX = 0000H	Disables mouse event handling
CX = Event mask	Specifies which events invoke the event handler
ES:DX = Address of the event handler	

Returns

Nothing

This function enables or disables application handling of specific events. If one or more bits are set in CX, event handling is enabled. The set bits in CX (call mask) specify which events invoke the event handler. The CX bits have the following meaning:

Bit	Event that Invokes the Event Handler
15-7	Not used
6	Releasing the middle button
5	Pressing the middle button
4	Releasing the right button
3	Pressing the right button
2	Releasing the left button
1	Pressing the left button
0	Change in cursor position

When an enabled event occurs, the mouse driver calls the event handler with the following parameters:

Register	Contents
AX	Event mask The event mask has the same bit assignments as the event mask in CX. However, A set bit in AX indicates that the event occurred.
BX	Button status (same as defined in function 0003H)
CX	Horizontal cursor position (same as defined in function 0003H)
DX	Vertical cursor position (same as defined in function 0003H)

The returned cursor position is always within the pixel range permitted by the current video mode or the limits specified by functions 0007H and 0008H.

To return control to the mouse driver, the event handler must use a far return.

Function 000DH: Enable Light-Pen Emulation

Industry-Standard

Parameters
AX = 000DH

Returns
Nothing

This function enables light-pen emulation. When light-pen emulation is enabled, calls to the interrupt 10H (video I/O) light-pen functions return the position of the mouse cursor at the last pen down. Pressing both the left and right mouse buttons simulates pressing the light pen button. When the left and right mouse buttons are released, the light pen is off the screen.

Function 000EH: Disable Light-Pen Emulation

Industry-Standard

Parameters
AX = 000EH

Returns
Nothing

This function disables light-pen emulation. When light-pen emulation is disabled, calls to interrupt 10H (video I/O) light-pen functions return only information about the light pen.

> **NOTE**
> The DIGITAL video system does not support the use of light pens. To use a light pen, an industry-standard video adapter that supports light pens must be installed. For additional information about the ROM BIOS light-pen functions, see the description of interrupt 10H (video I/O) in Chapter 15.

Function 000FH: Set Mouse Motion/Pixel Ratio

Industry-Standard

Parameters

AX =	000FH
CX =	The number of encoder counts of horizontal mouse motion to 8 pixels of horizontal cursor motion (range equals 1 - 32767).
DX =	The number of encoder counts of vertical mouse motion to 8 pixels of vertical cursor motion (range equals 1 - 32767).

Returns

Nothing

This function defines the ratio of mouse motion to cursor motion. The default ratio (see function 0000H) is 8 encoder counts for 8 pixels of horizontal motion and 16 encoder counts for 8 pixels of vertical motion. The default ratio requires approximately 3.2 inches of mouse motion to move the cursor from border to border (horizontally or vertically).

Function 0010H: Conditional Hide Cursor

Industry-Standard

Parameters

AX = 0010H
CX = Left margin (smallest x-axis screen coordinate)
DX = Top margin (smallest y-axis screen coordinate)
SI = Right margin (largest x-axis screen coordinate)
DI = Bottom margin (largest y-axis screen coordinate)

Returns

Nothing

This function defines a rectangular protected region on the screen. If the mouse cursor enters this region, the mouse cursor is disabled automatically. Executing function 0001H (show cursor) releases the protected region and displays the cursor. Before screen updates are performed, call this function or function 0002H (hide cursor).

Function 0013H: Set Speed Threshold

Industry-Standard

Parameters

AX = 0013H
DX = Speed threshold

Returns

Nothing

This function sets a mouse speed threshold. When the speed of the mouse motion exceeds the specified threshold, the ratio of cursor motion to mouse motion is doubled. The speed threshold is measured in encoder counts per second. Function 0000H initializes the speed threshold to 64 encoder counts per second.

For additional information about the mouse, see Chapter 10 Mouse Information.

Function 001CH: Get Driver Version

DIGITAL Extension

Parameters

AX = 001CH

Returns

BX = Version code
CX = Release number

This function returns the version code and release number of the mouse driver. Each register returns two ASCII codes. The BX register returns 'SS' (serial standard) or 'DE' (DIGITAL). The CX register returns two ASCII digits that represent the release number. For example, '21' equals release number 2.10.

Function 0024H: Get Configuration

DIGITAL Extension

Parameters

AX = 0024H	
ES:DX = Address of the configuration table	

Returns

AX = FFFFH	Successful operation (any other value indicates failure)
	Number of physical buttons updated
	Number of logical buttons updated

If the logical number of buttons equals 2, the mouse driver translates the middle button as a combination of the left and right buttons. The table pointed to by ES:DX is 32 bytes long and has the following organization:

Offset	Size	Description
0000H	12 bytes	Reserved
000CH	2 bytes	Number of physical buttons
000EH	2 bytes	Number of logical buttons
0010H	16 bytes	Reserved

Function 0025H: Set Configuration

DIGITAL Extension

Parameters

AX = 0025H
BX = 0004H
CX = The number of logical buttons (2 or 3)

Returns

AX = FFFFH	Successful operation (any other value indicates failure)

If the logical number of buttons equals 2, the mouse driver translates the middle button as a combination of the left and right buttons.

Enhanced Graphics Adapter (EGA) Functions

When the mouse driver detects an enhanced graphics adapter (EGA), the mouse driver installs extensions to interrupt 10H (video I/O). These extensions provide the mouse driver and the application with a means of communicating the current state of the EGA write-only registers. Table 16-10 lists the extensions to the interrupt 10H functions.

Table 16-10 Extensions to Interrupt 10H EGA Functions

Function	Description
F0H	Read EGA Register
F1H	Write EGA Register
F2H	Read EGA Register Group
F3H	Write EGA Register Group
F4H	Read EGA Register List
F5H	Write EGA Register List
FAH	EGA Functions Installed

EGA registers are referred to by a group number and a register number. Table 16-11 lists the group numbers and the registers associated with each group.

Table 16-11 EGA Register Groups and Associated Registers

Group	Register	Port	Description
00H	00H-18H	03D4H	CRT controller
08H	00H-04H	03C4H	Sequencer
10H	00H-08H	03CEH	Graphics controller
18H	00H-13H	03C0H	Attribute controller
20H	00H	03C2H	Output register
28H	00H	03DAH	Feature control register
30H	00H	03CCH	Graphics 1 position
38H	00H	03CAH	Graphics 2 position

Function F0H: Read EGA Register

DIGITAL Extension

Parameters

AH = F0H
BX = Register number
DX = The group number

Returns

BL = The contents of the indicated register

Function F1H: Write EGA Register

DIGITAL Extension

Parameters

AH = F1H
BL = Register number
BH = The value to write
DX = The group number

Returns

Nothing

Function F2H: Read EGA Register Group

DIGITAL Extension

Parameters

AH = F2H
ES:BX = Buffer address
CH = Starting register number
CL = Number of registers
DX = Group number

Returns

Nothing

This function reads the contents of the specified EGA registers. The EGA registers are from the group indicated by register DX. The contents of the EGA registers are stored in the buffer pointed to by ES:BX. Register CH, an index into the group, specifies the first register contents read. Register CL specifies the number of registers to write.

The buffer pointed to by ES:BX is an array of 8-bit values. Register CL specifies the number of entries in the buffer. Each entry in the buffer is the value to write to the corresponding register.

Function F3H: Write EGA Register Group

DIGITAL Extension

Parameters

AH = F3H
ES:BX = Buffer address
CH = Starting register number
CL = Number of registers
DX = Group number

Returns

Nothing

This function writes the specified EGA registers. The EGA registers are from the group indicated by register DX. Registers ES:BX point to a buffer that contains the corresponding values. Register CH, an index into the register group, specifies the first register written. Register CL specifies the number of registers to write.

Function F4H: Read EGA Register List

DIGITAL Extension

Parameters

AH = F4H ES:BX = Address of the register list CX = Number of entries in the list

Returns

Nothing

This function transfers the contents of the indicated EGA register to the corresponding value byte in the list. The caller must supply the group and register numbers. Each entry is 4 bytes long and contains the following:

Offset	Size	Description
00H	Word	Group number
02H	Byte	Register number
03H	Byte	Value

Function F5H: Write EGA Register List

DIGITAL Extension

Parameters

AH = F5H
ES:BX = Address of the register list
CX = Number of entries in the list

Returns

Nothing

This function transfers the value byte of each entry to the indicated EGA register. Each entry is 4 bytes long and contains the following:

Offset	Size	Description
00H	Word	Group number
02H	Byte	Register number
03H	Byte	Value

Function FAH: EGA Functions Installed

DIGITAL Extension

Parameters

AH = FAH
BX = 0000H

Returns

BX = 0000H	EGA functions not installed
BX = 0001H-FFFFH	EGA functions installed

MS-DOS Media ID Tables

Hard Disk Support Through FDISK

FDISK is the MS-DOS fixed disk management utility. It is responsible for initializing the disk, setting up the partition information, and setting up the boot information. FDISK loads the tables pointed to by the ROM BIOS Interrupt 41H and Interrupt 46H. FDISK was designed so that any hard disk type can be loaded at these vector pointers. Table 16-12 contains the hard disk types predefined within the FDISK program.

Table 16-12 Hard Disk Types

Type	Cylinders	Heads	Precomp	Control	Landing	Sectors
RD32	820	6	None*	0	910	17
RD31	615	4	256	0	669	17
1	306	4	128	0	305	17
2	615	4	300	0	615	17
3	615	6	300	0	615	17
4	940	8	512	0	940	17
5	940	6	512	0	940	17
6	615	4	None*	0	615	17
7	462	8	256	0	511	17
8	733	5	None*	0	733	17
9	900	15	None*	0	901	17
10	820	3	None*	0	820	17
11	855	5	None*	0	855	17
12	855	7	None*	0	855	17
13	306	8	128	0	319	17
14	733	7	None*	0	733	17
16	612	4	0	0	663	17
17	977	5	300	0	977	17
18	977	7	None*	0	977	17
19	1024	7	512	0	1023	17
20	733	5	300	0	732	17
21	733	7	300	0	732	17
22	733	5	300	0	733	17
23	306	4	0	0	336	17

* To specify no precompensation, the register contents should be FFFFH.

Disk Parameters

MS-DOS for the VAXmate supports many types of disks. Table 16-13 contains the BIOS parameter block data for the most frequently used and supported disk types:

Table 16-13 BIOS Parameter Block Data

	A	B	C	D	E	F	G	H
Bytes/Sector	512	512	512	512	512	512	512	512
Sector/Cluster	1	1	2	1	2	1	4	1
Reserved Sector	1	1	1	1	1	20	1	1
Number of FATs	2	2	2	2	2	2	2	1
Dir Entries	224	64	112	64	112	96	512	48
Sectors/disk	2400	360	720	320	640	800	41667	128
Media byte	F9H	FCH	FDH	FEH	FFH	FAH	F8H	FEH
Sectors/FAT	7	2	2	1	1	3	41	1
Sectors/Track	15	9	9	8	8	10	17	-
Heads/drive	2	1	2	1	2	1	4	-
Hidden Sectors	0	0	0	0	0	0	17	0
Tracks/disk	80	40	40	40	40	80	614	-

Disk A is a 96-TPI high capacity VAXmate workstation diskette. This disk type is supported by FORMAT, MS-DOS read and write operations, and DISKCOPY.

Disk B is a 48-TPI, single-sided, 9-sector-per-track diskette. This disk type is supported by FORMAT, MS-DOS read and write operations, and DISKCOPY.

DISK C is a 48-TPI, double-sided, 9-sector-per-track diskette. This disk type is supported by FORMAT, MS-DOS read and write operations, and DISKCOPY.

DISK D is a 48-TPI, single-sided, 8-sector-per-track diskette This disk type is supported by FORMAT, MS-DOS read and write operations, and DISKCOPY.

DISK E is a 48-TPI, double-sided, 8-sector-per-track diskette. This disk type is supported by FORMAT, MS-DOS read and write operations, and DISKCOPY.

DISK F is a Rainbow RX50 diskette. This disk type is supported by MS-DOS read and write operations and DISKCOPY.

DISK G is an RD31 fixed disk that has one 20-Mbyte partition. The sector-to-cluster ratio becomes eight sectors to one cluster for any fixed disk type F8H that has less than 32,681 sectors per image. If disk type F8H has 32,680 or less sectors per image, it uses a 12-bit FAT. If disk type F8H has 32,681 or more sectors per image, it uses a 16-bit FAT. This disk type is supported by FDISK, FORMAT, and MS-DOS read and write operations.

DISK H is a Mdrive disk that is a minimum of 64 Kbytes. Each increment of 64 Kbytes of Mdrive increases the allowable root directory entries by the increment size. For example, a 256 Kbyte Mdrive will have 48 (number of directory entries for one Mdrive) * 4 (4 * 64 Kbytes = 256 Kbytes) = 192 directory entries. This disk is supported by MS-DOS read and write operations.

MS-DOS International Support

FONT and GRAFTABL

A font file contains the size and shape description of the characters in a character set. The ability to load new fonts from disk allows the VAXmate workstation to display characters from a character set that is appropriate for the environment. Because of the VAXmate workstation's advanced video features, fonts are stored in two types of files, depending on the video mode. Font files with a .FNT file extension support the text video modes. Font files with a .GRF file extension support the graphics video modes.

The differences between the two types of font files are the character cell size, the file size, where they are stored in memory, and the utilities used to load them.

FONT.COM

FONT.COM is an MS-DOS utility that loads disk-based font files. At boot time, MS-DOS loads the default font file STD.FNT (stored in the ROM BIOS). STD.FNT can be replaced at any time with another disk-based font file in the proper format. FONT.COM searches the current directory, the root directory, the path, and any appended directories for the file. It is not necessary to reboot the system after FONT.COM loads a new font file. In text mode, a newly-loaded font file affects data already displayed on the screen.

GRAFTABL.COM

GRAFTABL.COM is a terminate and stay resident program that loads disk-based font files. These fonts can be displayed only when the VAXmate workstation is in a graphics mode. A newly-loaded graphics font file affects data already displayed on the screen. If no font file is specified on the command line, GRAFTABL.COM attempts to load the font file that corresponds to the current font file for text mode. If the current font file for text mode is a font file other than STD.FNT, GRAFTABL.COM searches the current directory, the root directory, the path, and any appended directories for the file. If GRAFTABL.COM cannot find the corresponding font file for graphics mode, or the file is not a valid font file, an error message is displayed, and no font file for graphics mode is loaded.

The ROM BIOS uses the information pointed to by the vector at Interrupt 1FH (7CH). This vector can point to either characters 0-FF or 80-FF.

GRAFTABL.COM uses Interrupt 10H, Function D0H to tell the ROM BIOS how many characters the pointer at Interrupt 1FH vector points to. To load STD.GRF, Interrupt 1FH vector points to characters 80 - FF. Otherwise, Interrupt 1FH vector points to characters 0 - FF.

Description of Fonts

Each .FNT character cell is 8x16 and is represented by 16 bytes of data. The total data representing 256 characters is 4096 (4 K) bytes.

Each .GRF character cell is 8x8 and is represented by 8 bytes of data. The total data representing 256 characters is 2048 (2 K) bytes.

How FONT.COM Affects KEYB.COM and SORT.EXE

FONT.COM affects how KEYB.COM is used. For example, if KEYB.COM loads a keyboard map that does not correspond to the current text font, the keyboard is incorrectly mapped. When a key is pressed, an unexpected character is displayed. This also affects the operation of SORT.EXE, because it sorts according to the current text font file.

Font File Structures

For proper loading, FONT.COM requires font files to be in the format specified in Table 16-14. There are no other restrictions for the user when creating a .FNT file.

Table 16-14 .FNT File Structure

Bytes	Contents	Description
0-3	FO$N	File identification label. The first four bytes must contain the ASCII characters "FO$N".
4	Total bytes	Total bytes per character. Must be 16.
5	Column	Number of columns per character. Must be 8.
6	Row	Number of rows per one font. Must be 16.
7-8	Total Characters	Total number of characters in a file. Must be 256.
9-10	Start Character	Starting character location to load first character description. Must be 0.
11-12	End Character	Ending character location to load last character description. Must be 255.
13-131	Reserved	
132-4228	Character description	16 bytes per character * 256 characters = 4096 bytes.
4229-4232	FO$N	File identification label.

For proper loading, GRAFTABL.COM requires font files to be in the format specified in Table 16-15. There are no other restrictions for the user when creating a .GRF file.

Table 16-15 .GRF File Structure

Bytes	Contents	Description
0-3	FO$N	File identification label. The first four bytes must contain the ASCII characters "FO$N".
4	Total bytes	Total bytes per character. Must be 8.
5	Column	Number of columns per character. Must be 8.
6	Row	Number of rows per character. Must be 8.
7-8	Total Characters	Total number of characters in a file. Must be 256.
9-10	Start Character	Starting character location to load first character description. Must be 0.
11-12	End Character	Ending character location to load last character description. Must be 255.
13-131	Reserved	
132-2180	Character description	8 bytes per character * 256 characters = 2048 bytes.
2181-2184	FO$N	File identification label.

Loading Font Files

To load a font file in the same way FONT.COM loads one, do an INT 10H with the following values in the specified registers:

Parameters

AH =	0D1H	
	AL =	0
	CX =	Number of characters to be loaded (256)
	DH =	01H
	DL =	First character to be loaded (0)
	ES:BX =	Address of the character description buffer

Returns

Nothing

KEYB

KEYB.COM is a terminate and stay resident program that loads disk-based keyboard map files. KEYB.COM can load a keyboard map file any time during an MS-DOS session, and the system does not have to be rebooted. KEYB.COM searches the current directory, the root directory, the path, and any appended directories for the file.

Keyboard Remapping

A keyboard map file contains an ASCII code and a scan code for every keyboard key. When a key is pressed, a scan code is generated. The firmware checks the current keyboard state, and indexes the correct table. Table 16-16 lists the keyboard tables.

Table 16-16 Keyboard Table

Table	Description
Base Table	Used when a key is pressed and no other key is down (caps lock is off).
Ctrl Table	Used when the Ctrl key is held down and another key is pressed.
Alt Table	Used when the Alt key is held down and another key is pressed.
Shift Table	Used when the Shift key is held down and another key is pressed.
NumLock Table	Used when the NumLock key has been activated and another key is pressed. The NumLock table only contains entries for scan codes 71 through 83.
Caps Table	Used when the Caps key has been activated and another key is pressed.
Alt/Ctrl Table	Used when the Alt and Ctrl keys are held down and another key is pressed.
Alt/Shift Table	Used when the Alt and Shift keys are held down and another key is pressed.
Ctrl/Shift Table	Used when the Ctrl and Shift keys are held down and another key is pressed.
Alpha ID Table	Indicates whether the scan code (1-35H) is an alpha key or a non alpha key. This is used when the Caps and Shift keys are held down and another key is pressed.

NOTE
When starting the VAXmate workstation, the Alt/Ctrl and the Alt/Shift pointers point to the Alt table. The Ctrl/Shift pointer points to the Ctrl table.

Each entry in the table is a word. The high byte contains a scan code, and the low byte contains an ASCII value. The ROM BIOS calculates the offset to the appropriate word in the table. It then sends the ASCII value in the low byte to the MS-DOS operating system. To make a key send out another value, change the value in the low byte of the word. The high byte can also be changed.

The table entries are arranged in order of the dependent scan code (1 through 105) for the keys. For more information on keyboards and keyboard mapping, see Chapter 8.

The Alpha-ID table is 53 bytes in size. Each byte is associated with the keys 1 through 35H, respectively. If an entry is set to zero, that key is treated as a non alpha key (Shift does not reverse Lock). If an entry is set to 0FFH, that key is treated as an alpha key (Shift reverses Lock). The table is indexed by the received scan code. KEYB issues the following interrupt to set the keyboard map file:

Interrupt 16H

Parameters

AH =	D6H	
	AL =	1 ; has to be a non-zero number
	CL =	table to be set 0 = Base (normal) table 1 = Ctrl table 2 = Alt table 3 = Shift table 4 = NumLock table 5 = Lock table 6 = Alt/Ctrl table 7 = Alt/Shift table 8 = Ctrl/Shift table 9 = Alpha-ID table

ES:BX must point to the caller-defined/supplied table.

Creating Keyboard Map Tables for International Countries

To support a new country, a keyboard map file must be created for:

- Industry standard character set (STD)
- DIGITAL Multinational character set (MCS)
- International Standards Organization character set (ISO)
- New country's 7-bit National Replacement character set (NRC)

These tables differ in that not all characters in the four character sets are located in the same position. For example, "a" grave is located at position 85H in STD, and at location 0E0H in MCS and ISO. When creating new tables, start with the default character set in the ROM BIOS (see Chapter 9) and make the necessary changes for the country.

When creating the Base table, Shift table, and Caps table, the scan codes should remain the same for all the keys. The character code should change for those characters located in a new position on the keyboard. For example, on the French LK250 keyboard, the "M" character is located where the ";" character is on the US keyboard. Therefore, the table entry for ";" in the US table contains the character code for "M" in the French table.

The NumLock table should remain unchanged.

The Alt table should resemble the default table in the ROM BIOS. The values returned for a character should be the same regardless of where the character is located on the keyboard. For example, the "a" on the French keyboard is located where the "q" is on the US keyboard, but an Alt/a on the French keyboard still produces the same value as an Alt/a on the US keyboard. Therefore, the table entry has to change to return the value returned on the US keyboard. For characters that are not on the US keyboard, the entry in the table should be set to undefined (0FFFFH) in the table.

The Alt/Shift and Alt/Ctrl tables should be identical to the Alt table. If a keyboard contains characters that are accessed by pressing Alt/Ctrl/key, the value in the table should be identical to the value that would be placed in the base table if the character was on the keyboard. For example, on the French keyboard, pressing Alt/Ctrl/$ generates the "]" character, so the entry for "$" in the Alt/Ctrl table should be 1B5Dh, which is the same value in the default set, base table for the "]" character.

The Ctrl table and the Ctrl/Shift table should be identical. The entries for dependent scan codes above 35H should be identical to the default entries in the ROM BIOS. The entries for dependent scan codes 1 through 35H should be the same as in the default table in the ROM BIOS. The entries for Ctrl/2 through Ctrl/8, Ctrl/Backspace, and Ctrl/Return should be:

Ctrl/2	entry: 0300H
Ctrl/3	entry: 041BH
Ctrl/4	entry: 051CH
Ctrl/5	entry: 061DH
Ctrl/6	entry: 071EH
Ctrl/7	entry: 081FH
Ctrl/8	entry: 097FH
Ctrl/Backspace	entry: 0E7FH
Ctrl/Return	entry: 1C0AH

The other table entries for 1 through 35H not specified should be undefined (value 0FFFFH).

How Compose Sequences Are Recognized

For compose sequences to work, every key that is pressed must be captured before it is sent to the user. Firmware interrupt 16H, Function 0D0H, Subfunction 0FEH "Enable Notify Before Buffering" notifies a routine before the key stroke is placed in the keyboard buffer.

When KEYB.COM runs with a valid keyboard map file, the compose routine is installed in memory as the notify routine. Whenever a key is pressed, the compose routine checks to determine whether it is part of a compose sequence (which includes dead diacritical keys).

Before the compose routine is installed, firmware interrupt 16H, Function 0D0H, Subfunction 82H "Return Segment:Offset of any Current Buffer Notify Routine" is called to get the address of the buffer routine that currently exists (if any).

If a buffer routine exists, the address of that routine is saved, and the buffer-notify routine is replaced with the compose routine. After the compose routine is done and before it does a far return, the compose routine checks the address it saved. If the Segment:Offset is zero, no routine existed, and the compose routine does a far return. Otherwise, the compose routine does a far jump to the buffer-notify routine that existed before the compose routine replaced it. When that routine does a far exit, it returns to whatever called the compose routine.

How Dead Diacritical Keys Are Recognized

A scan code of 0EFH indicates that a key is a dead diacritical key. When the scan code is received, the compose routine treats the ASCII code (low byte) as the first key in the compose sequence. The next key pressed is used as the second key in the compose sequence.

Format and Use of the Compose Sequence Pointer Table

The compose sequence pointer table contains 96 words. Each word contains a pointer (byte offset from the beginning of the file) to its corresponding compose sequence translation table. The 60H entries refer to ASCII values (character codes) 20H through 7FH. For example, the first word contains the pointer to the translation table for the space character (ASCII value 20H). If the pointer is zero, no translation table exists for that ASCII value. A compose sequence is made up of three keys: the compose key followed by two other keys in the range 20H through 7FH. The second key in the sequence indexes this table and obtains the pointer to the translation table. The third key in the sequence is then checked to determine if it exists in the translation table.

Format and Use of the Compose Sequence Translation Table

The compose sequence translation table can be a maximum of 1024 words. This area contains the translation tables for the legal compose sequences. It is possible to have as many as 96 translation tables (one each for ASCII values 20H through 7FH). The first byte in a translation table contains the number of entries in the table. The size of the table is 2 * number of entries + 1. The entries consist of 2 bytes:

- The first byte is the third character (ASCII value) of the compose sequence.
- The second byte is the compose character (ASCII value) to be returned as a result of the compose sequence.

Changing to STDUS.KEY and Back Again

Pressing Ctrl/Alt/F2 replaces the current keyboard map file with the default keyboard map file STDUS.KEY. Pressing Ctrl/Alt/F3 replaces the current keyboard map file with the last map file loaded into memory. This feature is activated when KEYB.COM is run with an external map file (in other words, MCSUS.KEY).

Keyboard Map File Structure

KEYB.COM requires keyboard map files to be in a specific format for proper loading. Table 16-17 shows the keyboard map file structure.

Table 16-17 Keyboard Map File Structure

Bytes	Contents
0-7	File identification string. The first four bytes must contain "KE$Y" for KEYB.COM to load the file. The next 124 bytes of the header record are reserved.
8	This byte is used to enable/disable the additional key codes associated with the LK250 keyboard. This byte is passed in the AL register to Interrupt 16H Function D3H. Bit 0 - Disable keypad state keys. The keypad keys NumLock and INSERT no longer set the keystate flags in the ROM BIOS area. They are stored in the keypad buffer as normal keys according to the table translation process. Bit 1 - Disable "Alt compose". This disables the generating of key scan codes using the Alt key and the keypad number keys. The keypad number keys are treated as normal keys.

Table 16-17 Keyboard Map File Structure (cont.)

Bytes	Contents
	Bit 2 - Disable all combination keys except for Ctrl/Alt/Del and Ctrl/Alt/Home. This disables all special detection of the key combinations that invoke special functions and treats them as normal key sequences. The disabled combinations are Shift/Prt Sc, Ctrl/Break, and Ctrl/NumLock. Bit 3 - Disables the ability to temporarily override the Lock key with a Shift key to unshift a key. Bit 4 - Guarantees that the LK250 keyboard is in DIGITAL mode. It sends a command to the LK250 keyboard to ensure it is in DIGITAL mode as well. It enables the use of DIGITAL extended scan codes sent by the LK250 keyboard. If it is not enabled, the 10-key keypad keys return scan codes of their equivalent numeric keypad keys as obtained from the current "normal" table through the table pointer. Bit 5 - Enable Compose key pass through. This means that even though the DIGITAL-extended codes are disabled, the Compose key is placed in the keyboard buffer.
9	Reserved; must always equal zero.
10-127	Other header information (not currently used).
128-337	Alt Table (105 keys x 2 bytes/entry = 210 bytes)
338-547	Ctrl Table (105 keys x 2 bytes/entry = 210 bytes)
548-757	Base Table (105 keys x 2 bytes/entry = 210 bytes)
758-783	NumLock Table (13 keys x 2 bytes/entry = 26 bytes)
784-993	Shift Table (105 keys x 2 bytes/entry = 210 bytes)
994-1203	Caps Table (105 keys x 2 bytes/entry = 210 bytes)
1204-1413	Alt/Ctrl Table (105 keys x 2 bytes/entry = 210 bytes)
1414-1623	Alt/Shift Table (105 keys x 2 bytes/entry = 210 bytes)
1624-1833	Ctrl/Shift Table (105 keys x 2 bytes/entry = 210 bytes)
1834-1886	Alpha-ID Table (53 Keys x 1 byte/entry = 53 bytes)
1887	Not used
1888-2079	Compose Sequence Pointer Table (96 words = 192 bytes)
2080-4147	Compose Sequence Translation Table (maximum of 2048 bytes)

LCOUNTRY

LCOUNTRY.EXE is an MS-DOS utility that installs and overlays country-specific information into the MS-DOS operating system. MS-DOS uses the information when it displays the date, the time, currency symbols, decimal separators, and performs case conversions on file names. The exact usage is country- and character-set dependent. LCOUNTRY.EXE can be executed manually by the user, or automatically, if the command is contained in the AUTOEXEC.BAT file.
When seaching for a file, LCOUNTRY.EXE searches the current directory, the root directory, the path, and any appended directories.

LCOUNTRY.EXE does not check to ensure that the font, keyboard, and country-specific data match.

Each LCOUNTRY file must have the .COU file extension.

Each .COU file represents a character set and can contain a multiple number of countries.

Country File Structure

The .COU file is a module that contains data that must be ORGed at 00H. There is no executable code in the module. The data overlays the previous data resident in the MS-DOS operating system.

The module size (number of bytes that overlay the resident table) cannot exceed 700 bytes. This is a combination of both the country-specific data structures and the case conversion tables.

The MS-DOS operating system organizes the data into structures that correlate to the country codes (each 18H bytes long). The block size is the first byte of the structure. This byte is always 18H or 0FFH, which indicates there are no more structures. Any other values are considered errors.

The second byte is the country code, which is the same as the international telephone number prefix for the country.

In the .COU file, the offset value is relative to the beginning of the module, which is ORGed at 00H. LCOUNTRY.EXE takes these offsets and adjusts them according to where they are loaded in the MS-DOS operating system. The next assembly language program section describes the file format.

```
BLOCK_SIZE       DB  18H    ;This is 18H for each data structure except
                            ;for the last, which will contain a OFFH to
                            ;indicate the structures have ended.
COUNTRY_CODE     DW  1      ;This is the value scanned to see if a hit
                            ;has occurred. This is an industry standard
                            ;US sample.
Date_tim_format  DW  0      ;0-USA, 1-EUR, 2-JAP
Currency_sym     DB '$'     ;Currency Symbol
                 DB  0
                 DB  0
                 DB  0
                 DB  0
Thous_sep        DB ','     ;Thousands separator
                 DB  0
Decimal_sep      DB '.'     ;Decimal separator
                 DB  0
Date_sep         DB '-'     ;Date separator
                 DB  0
Time_sep         DB ':'     ;Time separator
                 DB  0
Bit_field        DB  0      ;Bit values
                            ;Bit 0 = 0 if currency symbol first
                            ;      = 1 if currency symbol last
                            ;Bit 1 = 0 if No space after currency symbol
                            ;      = 1 if space after currency symbol
Currency_cents   DB  2      ;Number of places after currency decimal point
Time_24          DB  0      ;0 if 12-hour time; 1 if 24-hour time
Case_Convert_Tab DW  offset ;Address/Offset of case mapping tables for a
                            ;particular country. If tables of several
                            ;countries are the same, the offset may be the
                            ;same. Every structure, however, must have a
                            ;pointer to the case conversion table.
                 DW  0      ;Offset to case mapping routine
Data_sep         DB ','     ;Data list separator character
                 DB  0
```

Case Conversion Tables

The case conversion tables contain the lowercase character and its associated uppercase character. This table is scanned by the MS-DOS operating system; if a hit occurs, the uppercase character is substituted.

Characters are arranged as pairs in the table, and the number of pairs varies. The first word of the table is neither a lowercase nor an uppercase pair, but the length of the table. Do not include this word in the table length calculation.

Table —— **Offset in country table**

02	00
a	A

Order as seen in memory

> NOTE
> All references to hexadecimal digits in Table 16-18 and the SORT tables are taken from the DIGITAL Multinational Character set. The case conversion table cannot contain an entry for a lowercase character that replaces the backslash character (ASCII 5CH). If this is allowed, COMMAND.COM cannot find any external commands to execute. If the backslash character is replaced by an uppercase character, there is no effect on the operation of COMMAND.COM.

Table 16-18 lists the characters that can cause problems for COMMAND.COM if they are replaced by a lowercase character.

Table 16-18 Characters Causing Problems for COMMAND.COM

Name	ASCII	Character
Asterisk	2AH	*
Slash	2FH	/
Colon	3AH	:
Semicolon	3BH	;
Equal	3DH	=
Question mark	3FH	?
Backslash	5CH	\
Vertical bar	7CH		

SORT

SORT.EXE is an MS-DOS utility that sorts character sets according to a predefined sorting order. When executed, SORT.EXE checks for the name of the current text font file. Then, it searches for a file with the same name, but with the file extension .SRT. If this file is found in the current directory, the root directory, the path, or any appended directories, SORT.EXE reads it into memory and uses those values for sorting.

Format for Sorting Order

A sort file is 256 bytes long, one byte for each character in the character set. The first byte in the sort table is the sort order for ASCII 0, the next byte is the sort order for ASCII 1, and so on. When sorting, all letters are collated with ASCII characters A-Z (code 41H through 5AH). Lowercase is translated to uppercase, and international characters are translated to their English equivalents.

Creating Sort Tables for Character Sets

When creating a sort table for a character set, try to keep the same order as the ASCII character set. In other words, the control characters (ASCII 0 through 31) should be first. The control characters should be followed by the symbols and numbers. The letters should come next, with each letter followed by its corresponding accented characters (if any). When more than one accented character exists for a letter, the order of the accented characters should be the same as the order in the character set. For example, in DIGITAL MCS "A" grave precedes "A" acute, and "A" acute precedes "A" circumflex. All three of these characters follow "A" and precede "B".

All upper and lowercase characters should have equivalent sort orders, if the sort is to be case insensitive. Otherwise, the lowercase characters should sort after the uppercase characters.

After all the letters and their accented characters, any leftover characters should follow in the order they appear in the character set.

The sort orders for Tables 16-19 through 16-23 are read from left to right and from top to bottom. All values in these tables are hexadecimal values.

Table 16-19 Sort Order for Industry Standard Character Set (STD)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	60	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
70	50	51	52	53	54	55	56	57	58	59	5A	7B	7C	7D	7E	7F
80	43	55	45	41	41	41	41	43	45	45	45	49	49	49	41	41
90	45	41	41	4F	4F	4F	55	55	59	4F	55	24	24	24	24	24
A0	41	49	4F	55	4E	4E	A6	A7	3F	A9	AA	AB	AC	21	22	22
B0	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E0	53	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Table 16-20 Sort Order for Digital Multinational Character Set (MCS)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	40	41	47	48	4A	4B	50	51	52	53	58	59	5A	5B	5C	5F
50	65	66	67	68	6A	6B	70	71	72	73	75	79	7A	7B	7C	7D
60	7E	41	47	48	4A	4B	50	51	52	53	58	59	5A	5B	5C	5F
70	65	66	67	68	6A	6B	70	71	72	73	75	B6	B7	B8	B9	BA
80	BB	BC	BD	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA
90	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA
A0	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA
B0	EB	EC	ED	EE	EF	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA
C0	42	43	44	45	46	78	76	49	4C	4D	4E	4F	54	55	56	57
D0	FB	5D	60	61	62	63	64	5E	77	6C	6D	6E	6F	74	FC	69
E0	42	43	44	45	46	78	76	49	4C	4D	4E	4F	54	55	56	57
F0	FD	5D	60	61	62	63	64	5E	77	6C	6D	6E	6F	74	FE	FF

Table 16-21 Sort Order for International Standards Organization Character Set (ISO)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	40	41	47	48	4A	4B	50	51	52	53	58	59	5A	5B	5C	5E
50	64	65	66	67	69	6A	6F	70	71	72	75	7B	7C	7D	7E	7F
60	80	41	47	48	4A	4B	50	51	52	53	58	59	5A	5B	5C	5E
70	64	65	66	67	69	6A	6F	70	71	72	75	BB	BC	BD	BE	BF
80	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
90	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
A0	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
B0	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
C0	42	43	44	45	46	78	76	49	4C	4D	4E	4F	54	55	56	57
D0	79	5D	5F	60	61	62	63	2A	77	6B	6C	6D	6E	73	7A	68
E0	42	43	44	45	46	78	76	49	4C	4D	4E	4F	54	55	56	57
F0	79	5D	5F	60	61	62	63	2F	77	6B	6C	6D	6E	73	7A	74

Table 16-22 Sort Order for French 7-Bit National Replacement Character Set (FR7)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	40	42	43	45	46	49	4A	4B	4C	4D	4E	4F	50	51	52
50	53	54	55	56	57	58	5A	5B	5C	5D	5E	5F	44	60	61	62
60	63	40	42	43	45	46	49	4A	4B	4C	4D	4E	4F	50	51	52
70	53	54	55	56	57	58	5A	5B	5C	5D	5E	48	59	47	7E	7F
80	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E0	53	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Table 16-23 Sort Order for German 7-Bit National Replacement Character Set (DE7)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	40	41	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50
50	52	53	54	55	57	58	5A	5B	5C	5D	5E	42	51	59	5F	60
60	61	41	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50
70	52	53	54	55	57	58	5A	5B	5C	5D	5E	42	51	59	56	7F
80	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E0	53	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Chapter 17
MS-Windows on the VAXmate

Introduction

This chapter defines unique features of the DIGITAL adaptation of MS-Windows for execution on the VAXmate. The intended audience is the programmer who writes applications for MS-Windows and wants to use extensions to the Application Programming Interface which DIGITAL provides. This chapter assumes the reader is familiar with the MS-Windows environment and has the MS-Windows Software Development Kit. This chapter is an addendum to the manuals provided with the MS-Windows Software Development Kit. The following documents are part of that kit:

- *MS-Windows Software Development Kit Programmer's Reference*
- *MS-Windows Software Development Kit Programmer's Utility Guide*
- *MS-Windows Software Development Kit Programming Guide*

Overview

The adaptation, sometimes referred to as the OEM layer of MS-Windows, consists of the machine-dependent support modules and special device drivers required to communicate with the machine's particular hardware.

The machine-dependent support modules allow MS-Windows to control the VAXmate hardware, such as the system's display screen, keyboard, mouse, and communications resources.

The adaptation, or OEM layer, is just one of three components or layers of the MS-Windows environment. It is the lowest level component, dealing only with the hardware. Above this layer is the Windows layer which consists of:

- Kernel - The Kernel provides for tasking, memory management, and the loading of modules and resources. Tasking is non preemptive with a prioritized round-robin scheduler.
- User - User is the windowing manager. It manages windows, provides for orderly painting, and provides the user interface (tiled windows, pop-up windows, dialog boxes, menus, icons, cursors and scroll bars). It also manages events from the mouse, keyboard and timer. It is responsible for sending messages to applications.
- GDI - GDI is the Graphics Device Interface for MS-Windows. It maintains the graphics device state. It provides regions and clipping, display attributes, display objects and display primitives.

Above the Windows layer is the application layer. Applications must use the MS-Windows Applications Programming Interface to run in the MS-Windows environment.

DIGITAL's extensions to the OEM layer include routines that are callable from MS-Windows applications. These extensions allow applications to use some of the advanced features of the VAXmate's LK250 keyboard and ethernet communications.

NOTE
Symbolic constants used in this section of the reference manual are documented in either the WINDOWS.H include file or the DECWIN.H file listing included at the end of this chapter.

Keyboard Driver for the LK250 Keyboard

The MS-Windows keyboard driver for the VAXmate's LK250 configures the keyboard in DIGITAL Extended Scan Code mode. In this mode the keyboard generates unique scan codes for each key. In general, applications running in the DIGITAL windows environment behave, in regard to the keyboard, basically the same as they would on an industry-standard machine running MS-Windows. This mode also allows for extensions in the keyboard architecture under MS-Windows so applications can uniquely identify each key on the keyboard.

This section describes the following:

- The behavior of the Numeric and Edit keypads on the LK250.
- The behavior of the LEDs on the LK250.
- Compose handling.
- Extension to the keyboard device driver for MS-Windows.
- Key mappings for each LK250 country keyboard.
- Character mappings between the MS-Windows ANSI and OEM character sets.

Numeric and Edit Keypads

The normal or default action of the numeric keypad is to execute cursor or editing functions. The Numlock key toggles the state of the numeric keypad (and associated LED) between two states which generate two separate sets of output. For example, if the Numlock key is toggled 'off' (LED off), striking the key with the '4' legend on top (left arrow in blue on front) generates a VK_LEFT virtual key message. If the Numlock key is toggled 'on' (LED on) striking the key generates a VK_NUMPAD4 virtual key message.

Some VK messages generated by the numeric keypad with Numlock off can be generated on the edit keypad. For example, the edit/cursor keypad left arrow key generates a VK_LEFT, too. One that cannot be generated is the numeric keypad's 5 key when Numlock is off. See the TOASCII tables below for further details.

In addition, if Numlock processing is enabled (see following section on the DecSetNumlockMode routine), Shift may be used to modify the action of the numeric keypad. Holding a shift key down while using the numeric keypad temporarily causes the keypad to return the messages associated with the other state of the Numlock key. For example, if Numlock processing is enabled and Numlock is on and the '4' key on the keypad is pressed, a VK_NUMPAD4 message is returned. If Shift is held down and the '4' is pressed again, a VK_LEFT message is returned. The keys affected by this temporary state switch are the 10-digit keys, and the Del key.

> **NOTE**
> Holding down Shift and pressing the PF4/*/Prt Sc key produces a print screen function regardless of whether or not Numlock is enabled and regardless of the Numlock state.

An application that requires the use of the Numlock key for some purpose of it's own, (i.e., PF2 key for the VT220 emulator) and does not want Numlock toggling can call the keyboard driver to disable Numlock processing. The Numlock key will generate a VK_OEM_PF2 message and NOT a VK_OEM_NUMBER message. The numeric keypad's state and LED will be unaffected by hitting the Numlock/PF2 key or using a shift key. Using this extension allows the application to receive unique virtual key codes for the entire keyboard. This is not the default mode for keyboard processing. Also, there are certain restrictions when using this mode. See the section on MS-Windows Keyboard Extensions for details.

Keyboard LEDs for the VAXmate LK250

Although there are four LEDs on the LK250, only the Caps Lock LED and the Numlock LED are supported in DIGITAL's MS-Windows adaptation for the VAXmate. The Scroll Lock has no meaning and is always OFF. The SPECIAL (Industry-Standard/DIGITAL) LED is always OFF while running applications designed explicitly for MS-Windows. The LED will be ON when a standard application has the keyboard input focus. While in MS-Windows, the user is prevented from toggling the keyboard in and out of Industry-Standard/DIGITAL keyboard mode (the Alt/F17 key sequence).

VAXmate Compose Handling

Compose sequences may be handled by the MS-Windows keyboard driver or by the application. The default is for the MS-Windows keyboard driver to handle compose. An application using the basic MS-Windows message routines (GetMessage, TranslateMessage and DispatchMessage) will receive "composed" output as described below.

Pressing either the Compose (compose character) key (E00) or a dead diacritical key initiates a compose sequence. If the compose sequence is started with the Compose key then the next two keys define the character to be composed. When the application sends the virtual Compose key back to the keyboard driver for translation (TranslateMessage), the compose sequence is initiated and the key is returned as a WM_KEYDOWN and WM_KEYUP message with the virtual key value of VK_OEM_COMPOSE. The second key in the sequence must be translated in order to continue the compose sequence. It generates a WM_DEADCHAR message with the character value of the dead key. The third key in the sequence must be translated in order to complete the compose sequence. If the key completes a valid compose sequence, it is translated to the appropriate ASCII value and passed to the application as a WM_CHAR message. If invalid, it is passed to the application as a WM_DEADCHAR with the character value of the last key pressed. Also, the keyboard bell sounds indicating an invalid compose sequence.

Two-key compose sequences are initiated with a dead diacritical key. When the application sends it back to the keyboard driver for translation (Translate-Message), the compose sequence is initiated and the key is returned as a WM_DEADCHAR message with the character value of the dead key. The second key must be translated in order to complete the compose sequence. If the key completes a valid compose sequence it is translated to the appropriate ASCII value and passed to the application as a WM_CHAR message. If invalid, it is passed to the application as a WM_DEADCHAR with the character value of the last key pressed. Also, the keyboard bell sounds indicating an invalid compose sequence.

Compose sequences may be aborted by hitting the BACKSPACE key as long as the application passes it to the Windows TranslateMessage routine, which generates a WM_DEADCHAR message rather than a WM_CHAR message for the BACKSPACE key when it is used to abort a compose sequence.

The default set of compose sequences supported are those that produce output in the ISO Latin-1 Character Set.

An application wishing to receive output in the DIGITAL Multinational Character Set instead of ISO Character Set may do so by calling the DecSetComposeState routine as described in the DIGITAL Windows Keyboard Extensions section below. This set provides the OE ligature, both upper and lower case, and the uppercase Y with umlaut, which the ISO Latin-1 set does not. In addition, two compose sequence results are remapped. The lowercase y with umlaut is remapped from FDh to FFh. The international currency symbol is remapped from A8h to A4h. Characters in the ISO Latin-1 set, which are not in the DIGITAL Multinational Character Set, are considered invalid in this mode.

There are certain restrictions when changing the compose mode. See the section on MS-Windows Keyboard Extensions for details.

Reserved Keys Under MS-Windows

Use of F17, F18, F19, and F20 with the Alt key are reserved by DIGITAL. Application programs under MS-Windows must not employ these four key sequences. When not running MS-Windows, Alt/F17 switches the keyboard between DIGITAL-extended mode and compatible mode. Alt/F20 produces the SYSREQ function. Alt/F18 and Alt/F19 are undefined but reserved for future use.

DIGITAL MS-Windows Keyboard Extensions

The DIGITAL adaptation of the keyboard driver provides three routines to handle keyboard user preference features. The first sets the state of the Shift key into Caps Lock mode or Shift Lock modes. The second sets the keyclick volume. The third enables or disables autorepeat. These routines are called by the Control Panel application to allow user selection of these features. The user's selections are saved in the WIN.INI file by the control panel. The MS-Windows keyboard driver reads the WIN.INI file during its keyboard enable routine and sets the keyboard preference features.

These routines may also be called by any other MS-Windows application. If called, the WIN.INI file should be updated to reflect the current user preference state. For more information about the WIN.INI file, see the *VAXmate User's Guide.*

In addition, the keyboard driver provides three application-callable routines. The first returns the current nationality of the keyboard. The second selects DIGITAL Multinational Character Set compose processing or ISO Latin-1 Character Set compose processing. The third selects compatible or extended Numlock key processing.

The six routines are documented in the following sections.

DecSetLockState (lock)

This routine sets the sense in which the Lock key is interpreted.

Parameters

lock	is an integer value specifying the action where: 0 = DEC_CAPSLOCK (default) 1 = DEC_SHIFTLOCK

Returns

Nothing

When you type a key with DEC_CAPSLOCK selected, the uppercase letter is used for the alphabetic keys, but the lower character on the numeric/symbolic keys is used. To clear the lock function momentarily, press the Shift key.

When you type a key with DEC_SHIFTLOCK selected, the uppercase letter is used for the alphabetic keys, and the top character on the numeric/symbol keys is used.

DecSetKClickVol (vol)

This routine sets the volume associated with keyclick.

Parameters

vol	is an integer value specifying the action where: 0 = DEC_NOSOUND 1 = DEC_SOFT 2 = DEC_INTERMED (default) 3 = DEC_LOUD

Returns

Nothing

DecSetAutorep (repeat)

This routine sets autorepeat on or off.

Parameters

repeat	is an integer value specifying the action where: 0 = DEC_AUTOREPOFF 1 = DEC_AUTOREPON(default)

Returns

Nothing

DecGetKbdCountry ( ) : Result

This routine returns the keyboard's nationality.

Parameters

none

Returns

Result	is an integer value identifying the country keyboard. These values are defined in DECWIN.H. DEC_USA is the U.S. keyboard. DEC_BRITAIN is the British keyboard. DEC_FRANCE is the French keyboard. DEC_WEST_GERMAN is the German keyboard. DEC_ITALY is the Italian keyboard. DEC_SPAIN is the Spanish keyboard. DEC_SWEDEN is the Swedish keyboard. DEC_FINLAND is the Finish keyboard. DEC_NORWAY is the Norwegian keyboard. DEC_DENMARK is the Danish keyboard. DEC_CANADA is the Canadian keyboard. DEC_SWISS_GERMAN is the Swiss German keyboard. DEC_SWISS_FRENCH is the Swiss French keyboard.

DecSetComposeState (compose_mode)

Parameters

compose_mode	is an integer value specifying the action where: 0 = DEC_ISO_COMP (default) 1 = DEC_MULTINAT_COMP

Returns

Nothing

> **IMPORTANT**
> An application using DecSetComposeState to change from default handling must call the routine to set the non-default state whenever it receives the keyboard input focus, and must reset it to the default when it loses the keyboard input focus. If it does not, other applications (all of which share the keyboard) that do not understand the non-default modes will not function properly.

This routine sets the sense in which the compose sequences are processed and the mapping of returned values.

By default, the legal set of compose characters are those characters in the ISO Latin-1 Character Set. The character translation is the byte value of the character's position in the ISO Latin-1 set.

Optionally, the legal set of compose characters can be set to the DIGITAL Multinational Character Set. The character translation is the byte value of the character's position in the DIGITAL Multinational Character Set.

DecSetNumlockMode (numlock_mode)

Parameters

numlock_mode	is an integer value specifying the action where: 0 = DEC_Numlock (default) 1 = DEC_NONumlock

Returns

Nothing

> **IMPORTANT**
> An application using DecSetNumlockMode to change from default handling must call the routine to set the non-default state whenever it receives the keyboard input focus, and must reset it to the default when it loses the keyboard input focus. If it does not, other applications (all of which share the keyboard) which do not understand the non-default modes will not function properly.

This routine sets the sense in which the Numlock key is processed.

If numlock_mode is 0 (that is, industry-standard-compatible), the Numlock key toggles the state of the numeric keypad and subsequent output. The Numlock key generates a VK_OEM_NUMBER virtual key code.

The application's .DEF file must contain an import statement for each of the routines it uses as follows:

```
IMPORTS
Keyboard.DecSetLockState
Keyboard.DecSetKClickVol
Keyboard.DecSetAutorep
Keyboard.DecGetKbdCountry
Keyboard.DecSetComposeState
Keyboard.DecSetNumlockMode
```

The application must declare the following for each routine it uses:

```
extern int FAR PASCAL DecSetLockState (int);
extern int FAR PASCAL DecSetKClickVol (int);
extern int FAR PASCAL DecSetAutorep (int);
extern int FAR PASCAL DecGetKbdCountry ();
extern int FAR PASCAL DecSetComposeState (int);
extern int FAR PASCAL DecSetNumlockMode (int);
```

When Numlock mode is set to no-Numlock, the current state of Numlock and Numlock LED are saved and the LED is turned OFF. Toggling the Numlock key always generates a VK_OEM_PF2 virtual key code. The state of the numeric keypad is equivalent to Numlock being ON. This mode allows for unique key identification between all keys on the numeric and edit keypads of the LK250 keyboard. When Numlock mode is reset to industry-standard compatible, the previous state is restored.

Windows Keyboard Processing Anomalies

Applications programmers should be aware of the WINDOWS software anomalies described in the following sections.

Repeating Key Allowed to Change Focus

When two or more copies of the same Windows applications program are loaded, it is possible to change the input focus of a repeating key. To create the condition, two or more copies of the applications program must have autorepeat enabled.

Select the first copy by moving the cursor to its window and pressing the left mouse button, which gives the first copy the input focus. Press and hold down a key (for example, the 'A'). After the required delay, the window displays multiple instances of the 'A' key. While the key is automatically repeating, move the mouse cursor to another copy of the application and press the left mouse button. This gives the second copy the input focus. When the focus shifts to the second copy, the repeating key follows the input focus, and the second copy displays the repeating key.

If the program monitors only the translated (WM_CHAR or WM_SYSCHAR) messages, the problem is not apparent. However, if the program monitors the KEYDOWN and KEYUP messages, there are at least two problems as follows:

1. The first copy of the program does not receive a KEYUP message.
2. The second copy of the program receives a KEYDOWN message with a previous state of keydown. The second copy should have received a KEYDOWN message with a previous state of keyup.

Illogical Set of Keyboard Messages

The following keyboard operations produce an illogical set of messages:

- Ensure that the NumLock LED is on.
- Press and hold down the left shift key.
- Press and release the '1' key on the numeric pad.
- Release the left shift key.

Table 17-1 contains the keyboard messages transmitted by MS-Windows.

Table 17-1 Keyboard Messages Transmitted by MS-Windows

MS-Windows Message Type	Scan Code	Prev Key State	Virtual Key Name	Comments
WM_KEYDOWN	2AH	Up	VK_SHIFT	Left shift
WM_KEYUP	36H	Down	VK_SHIFT	Illogical message (right shift)
WM_KEYDOWN	4FH	Up	VK_END	Keypad '1'
WM_KEYDOWN	36H	Up	VK_SHIFT	Illogical message (right shift)
WM_KEYUP	36H	Down	VK_SHIFT	Illogical message (right shfit)
WM_KEYUP	4FH	Down	VK_END	Keypad '1'
WM_KEYDOWN	36H	Up	VK_SHIFT	Illogical message (right shift)
WM_KEYUP	2AH	Down	VK_SHIFT	Left shift

If the right shift key is used instead of the left shift key, the number of messages are the same. However, the two messages with a scan code of 2AH are changed to 36H.

The keys on the numeric keypad which exhibit this behavior are zero (0) through nine (9), minus sign (VK_SUBTRACT), plus sign (VK_ADD) and period (.), which can be either VK_DECIMAL or VK_DELETE.

Key Mappings for VAXmate's LK250

In Tables 17-2 through 17-13:

Keypos	Refers to the keyboard layout numbering scheme used in Figure 17-1.
Keycap	Refers to the legend in black (and/or blue) on the LK250 key.
Virtkey	Refers to the keyname associated with that key; these virtkeys may be in any of the three categories: standard, extended (preceded by an asterisk), or OEM specific (using the convention of VK_OEM_KEYNAME).
TOASCII Translation Table	Refers to the possible virtual key translations based on the state of the Shift, Ctrl and Alt keys. The keyboard driver's TOASCII entry point is ultimately called when an application makes the TranslateMessage function call.
Unshift	Refers to the default translated output for the unshifted keystroke (TOASCII translation table).
Shift	Refers to the default translated output for the shifted keystroke (TOASCII translation table).
Ctrl	Refers to the default translated output for the keystroke when pressed with control key held down (TOASCII translation table).
Ctrl/Alt	Refers to the default translated output for the keystroke when pressed with the Ctrl and Alt keys held down (TOASCII translation table); this value is the 'extra' output for this key.
Ctrl/Alt/Shift	Refers to the default translated output for the keystroke when pressed with the Ctrl and Alt and left Shift keys held down (TOASCII translation table); this value is the shifted 'extra' output for this key.

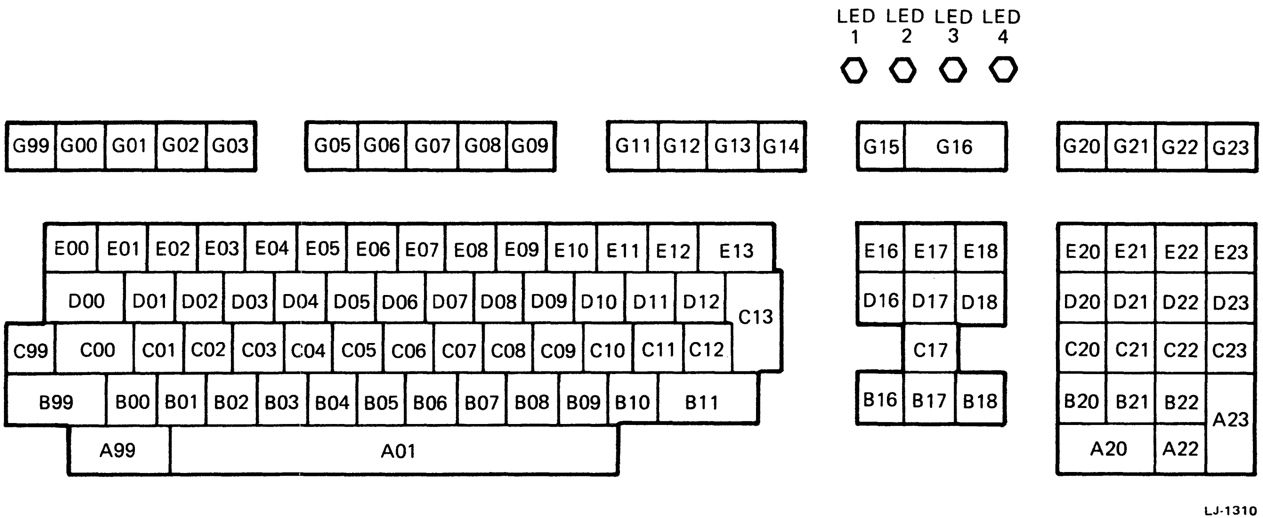

Figure 17-1 Keyboard Position Labels

Table 17-2 US to ASCII Translation Table - Main Key Array

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 !	'1'	1	!			
E02	2 @	'2'	2	@	NUL		
E03	3 # UKpound	'3'	3	#	ESC	UK pound	
E04	4 $	'4'	4	$	FS		
E05	5 %	'5'	5	%	GS		
E06	6 ^	'6'	6	^	RS		
E07	7 &	'7'	7	&	US		
E08	8 *	'8'	8	*	DEL		
E09	9 (	'9'	9	(			
E10	0)	'0'	0	)			
E11	- _	VK_OEM_MINUS	-	_			
E12	= +	VK_OEM_PLUS	=	+			
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'y'	y	Y	EM		

Table 17-2 US to ASCII Translation Table - Main Key Array (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	[{	VK_OEM_4	[	{	ESC		
D12	] }	VK_OEM_6	]	}	GS		
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	; :	VK_OEM_1	;	:			
C11	' "	VK_OEM_7	'	"			
C12	\ |	VK_OEM_5	\	|	FS		
C13	RETURN	VK_RETURN	CR	CR	LF		

Table 17-2 US to ASCII Translation Table - Main Key Array (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B99	SHIFT	VK_SHIFT					
B00	‘ ~	VK_OEM_3	‘	~	RS		
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, <	VK_OEM_COMMA	,	<			
B09	. >	VK_OEM_PERIOD	.	>			
B10	/ ?	VK_OEM_2	/	?	US		
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

Table 17-2 US to ASCII Translation Table - Edit Key Array

Keypos	Keycap	Virtkey
E16	FIND	VK_HOME
E17	INSERT	VK_INSERT
E18	REMOVE	VK_DELETE
D16	SELECT	*VK_END
D17	PREV	VK_PRIOR
D18	NEXT	VK_NEXT
C17	up arrow	VK_UP
B16	left arrow	VK_LEFT
B17	down arrow	VK_DOWN
B18	right arrow	VK_RIGHT

Table 17-2 US to ASCII Translation Table - Keypad Array

Keypos	Keycap	Virtkey
E20	PF1	VK_ESCAPE = VK_OEM_PF1
E21	PF2	VK_OEM_NUMBER or in no-Numlock mode VK_OEM_PF2
E22	PF3	VK_OEM_SCROLL = VK_OEM_PF3, with CTRL produces VK_CANCEL**
E23	PF4	*VK_MULTIPLY = VK_OEM_PF4

NOTES

The VK_OEM_PF1 (VK_ESCAPE) key, produces a 27 (ESC) for unshifted and shifted key presses if translated; VK_MULTIPLY generates a '*' for translation of the unshifted keypress (shifted VK_MULTIPLY produces a print screen function).

** VK_CANCEL, when translated by means of a call to TranslateMessage, produces output of 03h.

Keypos	Keycap	Virtkey w/Numlock on or Numlock disabled	Virtkey w/Numlock off
D20	7	*VK_NUMPAD7	VK_HOME
D21	8	*VK_NUMPAD8	VK_UP
D22	9	*VK_NUMPAD9	VK_PRIOR
D23	-	*VK_SUBTRACT	*VK_SUBTRACT
C20	4	*VK_NUMPAD4	VK_LEFT
C21	5	*VK_NUMPAD5	*VK_CLEAR
C22	6	*VK_NUMPAD6	VK_RIGHT
C23	,	*VK_ADD	*VK_ADD
B20	1	*VK_NUMPAD1	*VK_END
B21	2	*VK_NUMPAD2	VK_DOWN
B22	3	*VK_NUMPAD3	VK_NEXT
A20	0	*VK_NUMPAD0	VK_INSERT
A22	.	*VK_DECIMAL	VK_DELETE
A23	ENTER	*VK_EXECUTE	*VK_EXECUTE

NOTE

Translation of VK_NUMPAD0-9 produces '0' through '9'. Similarly, translation of VK_SUBTRACT produces '-', VK_ADD produces '+', VK_DECIMAL produces '.', and VK_EXECUTE produces a carriage return (13 decimal).

TOP ROW FUNCTION KEYS

Keypos	Keycap	Virtkey	
G99	F1	VK_F1	
G00	F2	VK_F2	
G01	F3	VK_F3	
G02	F4	VK_F4	
G03	F5	VK_F5	
G05	F6	VK_F6	
G06	F7	VK_F7	
G07	F8	VK_F8	
G08	F9	VK_F9	
G09	F1	VK_F10	
G11	F11	*VK_F11	
G12	F12	*VK_F12	
G13	F13	*VK_F13	
G14	F14	*VK_F14	
G15	HELP	*VK_F15	
G16	DO	*VK_F16	
G20	F17	VK_OEM_F17	
G21	F18	VK_OEM_F18	
G22	F19	VK_OEM_F19	
G23	F20	VK_OEM_F20	; with Alt, produces SYSREQ output

Table 17-3 Danish to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 !	'1'	1	!			
E02	2 " @	'2'	2	"	NUL		@
E03	3 # section	'3'	3	#	ESC		section
E04	4 $ currency	'4'	4	$	FS		currency
E05	5 %	'5'	5	%	GS		
E06	6 & ^	'6'	6	&	RS		^
E07	7 / &	'7'	7	/	US		&
E08	8 (*	'8'	8	(	DEL		*
E09	9) (	'9'	9	)			(
E10	0 =)	'0'	0	=			)
E11	+ ? - _	VK_OEM_PLUS	+	?		-	_
E12	’ ‘ = +	VK_OEM_3	dead ’	dead ‘		=	+
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-3 Danish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	A ring [{	VK_OEM_6	a ring	A ring		[	{
D12	" ^] }	VK_OEM_7	dead "	dead ^		]	}
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	AE ; :	VK_OEM_4	ae	AE		;	:
C11	O slash ' "	VK_OEM_5	o slash	O slash		'	"
C12	' * ` ~	VK_OEM_2	'	*		`	dead ~
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \ |	VK_OEM_1	<	>		\	|
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		

Table 17-3 Danish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ; <	VK_OEM_COMMA	,	;			<
B09	. : >	VK_OEM_PERIOD	.	:			>
B10	- _ / ?	VK_OEM_MINUS	-	_		/	?
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-4 Finnish to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 ! U umlaut	'1'	1	!		u umlaut	U umlaut
E02	2 " @	'2'	2	"	NUL		@
E03	3 UKpound #	'3'	3	UKpound	ESC		#
E04	4 $ section	'4'	4	$	FS		section
E05	5 % degree	'5'	5	%	GS	degree	
E06	6 & ^	'6'	6	&	RS		^
E07	7 / &	'7'	7	/	US		&
E08	8 (*	'8'	8	(	DEL		*
E09	9) (	'9'	9	)			(
E10	0 =)	'0'	0	=			)
E11	+ ? - _	VK_OEM_PLUS	+	?		-	_
E12	’ ‘ = +	VK_OEM_3	dead ’	dead ‘		=	+
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-4 Finnish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	A ring [{	VK_OEM_6	a ring	A ring		[	{
D12	" ^] }	VK_OEM_7	dead "	dead ^		]	}
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	O umlaut ; :	VK_OEM_4	o umlaut	O umlaut		;	:
C11	A umlaut ' "	VK_OEM_5	a umlaut	A umlaut		'	"
C12	' * ` ~	VK_OEM_2	'	*		`	dead ~
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \ |	VK_OEM_1	<	>		\	|
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		

Table 17-4 Finnish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ; <	VK_OEM_COMMA	,	;			<
B09	. : >	VK_OEM_PERIOD	.	:			>
B10	- _ / ?	VK_OEM_MINUS	-	_		/	?
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-5 French to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	& 1 ~	'1'	&	1		dead ~	
E02	e acute 2 @	'2'	e acute	2	NUL	@	
E03	" 3 #	'3'	"	3	ESC	#	
E04	' 4 `	'4'	'	4	FS	dead `	
E05	(5	'5'	(	5	GS		
E06	section 6 ^	'6'	section	6	RS	^	
E07	e grave 7	'7'	e grave	7	US		
E08	! 8 {	'8'	!	8	DEL	{	
E09	c cedil 9 }	'9'	c cedil	9		}	
E10	a grave 0	'0'	a grave	0			
E11	) degree	VK_OEM_4	)	degree			
E12	- _	VK_OEM_MINUS	-	_			
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	A	'A'	a	A	SOH		
D02	Z	'Z'	z	Z	SUB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-5 French to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	^ " [	VK_OEM_7	dead ^	dead "		[	
D12	$ *]	VK_OEM_1	$	*		]	
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	Q	'Q'	q	Q	DC1		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	M	'M'	m	M	CR		
C11	u grave %	VK_OEM_5	u grave	%			
C12	mu UK pound	VK_OEM_6	mu	UK pound			
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \	VK_OEM_3	<	>		\	
B01	W	'W'	w	W	ETB		
B02	X	'X'	x	X	CAN		

Table 17-5 French to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SOH		
B07	, ?	VK_OEM_COMMA	,	?			
B08	; .	VK_OEM_PERIOD	;	.			
B09	: /	VK_OEM_2	:	/			
B10	= +	VK_OEM_PLUS	=	+			
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-6 French Canadian and Bilingual Canadian to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 !	'1'	1	!			
E02	2 " @	'2'	2	"	NUL		@
E03	3 / #	'3'	3	/	ESC		#
E04	4 $	'4'	4	$	FS		
E05	5 %	'5'	5	%	GS		
E06	6 ? ^	'6'	6	?	RS		^
E07	7 &	'7'	7	&	US		
E08	8 *	'8'	8	*	DEL		
E09	9 (	'9'	9	(			
E10	0)	'0'	0	)			
E11	- _	VK_OEM_MINUS	-	_			
E12	= +	VK_OEM_PLUS	=	+			
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-6 French Canadian and Bilingual Canadian to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	c Cedil [{	VK_OEM_4	c cedilla	C cedilla		[	{
D12	# @] }	VK_OEM_6	#	@		]	}
C99	CTRL	VK_CONTRO					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	; :	VK_OEM_1	;	:			
C11	` ^ ' ~	VK_OEM_7	dead `	dead ^		'	"
C12	\		VK_OEM_5	\					
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > ` ~	VK_OEM_3	<	>		`	~
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		

Table 17-6 French Canadian and Bilingual Canadian to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ' <	VK_OEM_COMMA	,	'			<
B09	. ~ >	VK_OEM_PERIOD	.	dead ~			>
B10	E acute / ?	VK_OEM_2	e acute	E acute		/	?
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-7 German to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 ! ~	'1'	1	!		dead ~	
E02	2 " @	'2'	2	"	NUL	@	
E03	3 section	'3'	3	section	ESC		
E04	4 $	'4'	4	$	FS		
E05	5 %	'5'	5	%	GS		
E06	6 &	'6'	6	&	RS		
E07	7 /	'7'	7	/	US		
E08	8 ({	'8'	8	(	DEL	{	
E09	9) }	'9'	9	)		}	
E10	0 =	'0'	0	=			
E11	sharp ?	VK_OEM_2	sharp	?			
E12	' ‘	VK_OEM_3	dead '	dead ‘			
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Z	'Z'	z	Z	SUB		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-7 German to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	U umlaut [	VK_OEM_6	u umlaut	U umlaut		[	
D12	+ *]	VK_OEM_PLUS	+	*		]	
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	O umlaut	VK_OEM_5	o umlaut	O umlaut			
C11	A umlaut	VK_OEM_4	a umlaut	A umlaut			
C12	# ^	VK_OEM_7	#	dead ^			
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \	VK_OEM_1	<	>		\	
B01	Y	'Y'	y	Y	EM		
B02	X	'X'	x	X	CAN		

Table 17-7 German to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ;	VK_OEM_COMMA	,	;			
B09	. :	VK_OEM_PERIOD	.	:			
B10	- _	VK_OEM_MINUS	-	_			
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-8 Italian to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 ! ~	'1'	1	!		dead ~	
E02	2 " ‘	'2'	2	"	NUL	dead ‘	
E03	3 UKpound cCedil	'3'	3	UKpound	ESC	c cedilla	
E04	4 $	'4'	4	$	FS	degree	
E05	5 %	'5'	5	%	GS		
E06	6 &	'6'	6	&	RS		
E07	7 /	'7'	7	/	US		
E08	8 ({	'8'	8	(	DEL	{	
E09	9) }	'9'	9	)		}	
E10	0 =	'0'	0	=			
E11	' ?	VK_OEM_2	'	?			
E12	i grave ^	VK_OEM_7	i grave	dead ^			
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-8 Italian to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	eGrav eAcu [	VK_OEM_6	e grave	e acute		[	
D12	+ *]	VK_OEM_PLUS	+	*		]	
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	o grave @	VK_OEM_5	o grave	@			
C11	a grave #	VK_OEM_4	a grave	#			
C12	u grave section	VK_OEM_1	u grave	section			
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \	VK_OEM_3	<	>		\	
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		

Table 17-8 Italian to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ;	VK_OEM_COMMA	,	;			
B09	. :	VK_OEM_PERIOD	.	:			
B10	- _	VK_OEM_MINUS	-	_			
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-9 Norwegian to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 !	'1'	1	!			
E02	2 " @	'2'	2	"	NUL		@
E03	3 UKpound #	'3'	3	UK pound	ESC		#
E04	4 $	'4'	4	$	FS		
E05	5 %	'5'	5	%	GS		
E06	6 & ^	'6'	6	&	RS		^
E07	7 / &	'7'	7	/	US		&
E08	8 (*	'8'	8	(	DEL		*
E09	9) (	'9'	9	)			(
E10	0 =)	'0'	0	=			)
E11	+ ? - _	VK_OEM_PLUS	+	?		-	_
E12	’ ‘ = +	VK_OEM_3	dead ’	dead ‘		=	+
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-9 Norwegian to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	A ring [{	VK_OEM_6	a ring	A ring		[	{
D12	" ^] }	VK_OEM_7	dead "	dead ^		]	}
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	O slash ; :	VK_OEM_5	o slash	O slash		;	:
C11	AE ' "	VK_OEM_4	ae	AE		'	"
C12	' * ` ~	VK_OEM_2	'	*		`	dead ~
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \ |	VK_OEM_1	<	>		\	|
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		

Table 17-9 Norwegian to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ; <	VK_OEM_COMMA	,	;			<
B09	. : >	VK_OEM_PERIOD	.	:			>
B10	- _ / ?	VK_OEM_MINUS	-	_		/	?
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-10 Spanish to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 inverted ! ~	'1'	1	inverted !		dead ~	
E02	2 inverted ? @	'2'	2	inverted ?	NUL	@	
E03	3 # UKpound	'3'	3	#	ESC	UKpound	
E04	4 $ aUndersc	'4'	4	$	FS	a underscore	
E05	5 % oUndersc	'5'	5	%	GS	o underscore	
E06	6 /	'6'	6	/	RS		
E07	7 &	'7'	7	&	US		
E08	8 *	'8'	8	*	DEL		
E09	9 ({	'9'	9	(		{	
E10	0) }	'0'	0	)		}	
E11	- _	VK_OEM_MINUS	-	_			
E12	= +	VK_OEM_PLUS	=	+			
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-10 Spanish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	' " [	VK_OEM_4	dead '	dead "		[	
D12	‘ ^]	VK_OEM_3	dead ‘	dead ^		]	
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	N tilde	VK_OEM_5	n tilde	N tilde			
C11	; :	VK_OEM_1	;	:			
C12	c cedilla	VK_OEM_2	c cedilla				
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \	VK_OEM_6	<	>		\	
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		

Table 17-10 Spanish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ?	VK_OEM_COMMA	,	?			
B09	. !	VK_OEM_PERIOD	.	!			
B10	' "	VK_OEM_7	'	"			
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-11 Swedish to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 ! U umlaut	'1'	1	!		u umlaut	U umlaut
E02	2 " @	'2'	2	"	NUL		@
E03	3 UKpound #	'3'	3	UKpound	ESC		#
E04	4 $ E acute	'4'	4	$	FS	e acute	E acute
E05	5 % degree	'5'	5	%	GS	degree	
E06	6 & ^	'6'	6	&	RS		^
E07	7 / &	'7'	7	/	US		&
E08	8 (*	'8'	8	(	DEL		*
E09	9) (	'9'	9	)			(
E10	0 =)	'0'	0	=			)
E11	+ ? - _	VK_OEM_PLUS	+	?		-	_
E12	' ` = +	VK_OEM_3	dead '	dead `		=	+
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Y	'Y'	y	Y	EM		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-11 Swedish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	A ring [{	VK_OEM_6	a ring	A ring		[	{
D12	" ^] }	VK_OEM_7	dead "	dead ^		]	}
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	O umlaut ; :	VK_OEM_4	o umlaut	O umlaut		;	:
C11	A umlaut ' "	VK_OEM_5	a umlaut	A umlaut		'	"
C12	' * ` ~	VK_OEM_2	'	*		`	dead ~
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \ |	VK_OEM_1	<	>		\	|
B01	Z	'Z'	z	Z	SUB		
B02	X	'X'	x	X	CAN		

Table 17-11 Swedish to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ; <	VK_OEM_COMMA	,	;			<
B09	. : >	VK_OEM_PERIOD	.	:			>
B10	- _ / ?	VK_OEM_MINUS	-	_		/	?
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-12 Swiss French to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 +	'1'	1	+			
E02	2 " @	'2'	2	"	NUL	@	
E03	3 * #	'3'	3	*	ESC	#	
E04	4 cCedil degree	'4'	4	c cedilla	FS	degree	
E05	5 % section	'5'	5	%	GS	section	
E06	6 &	'6'	6	&	RS		
E07	7 / |	'7'	7	/	US	|	
E08	8 (	'8'	8	(	DEL		
E09	9)	'9'	9	)			
E10	0 =	'0'	0	+			
E11	' ? `	VK_OEM_7	dead '	?		'	
E12	^ ` ~	VK_OEM_PLUS	dead ^	dead `		dead ~	
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Z	'Z'	z	Z	SUB		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-12 Swiss French to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	eGrav uUmlaut [	VK_OEM_5	e grave	u umlaut		[	
D12	" !]	VK_OEM_6	dead "	!		]	
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	eAcu oUmlaut {	VK_OEM_1	e acute	o umlaut		{	
C11	aGra aUmlaut }	VK_OEM_2	a grave	a umlaut		}	
C12	$ UKpound	VK_OEM_4	$	UKpound			
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \	VK_OEM_3	<	>		\	
B01	Y	'Y'	y	Y	EM		
B02	X	'X'	x	X	CAN		

Table 17-12 Swiss French to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ;	VK_OEM_COMMA	,	;			
B09	. :	VK_OEM_PERIOD	.	:			
B10	- _	VK_OEM_MINUS	-	_			
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

Table 17-13 Swiss German to ASCII Translation Table

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
E00	Compose	VK_OEM_COMPOSE					
E01	1 +	'1'	1	+			
E02	2 " @	'2'	2	"	NUL	@	
E03	3 * #	'3'	3	*	ESC	#	
E04	4 cCedil degree	'4'	4	c cedilla	FS	degree	
E05	5 % section	'5'	5	%	GS	section	
E06	6 &	'6'	6	&	RS		
E07	7 / |	'7'	7	/	US	|	
E08	8 (	'8'	8	(	DEL		
E09	9)	'9'	9	)			
E10	0 =	'0'	0	+			
E11	' ? `	VK_OEM_7	dead '	?		'	
E12	^ ` ~	VK_OEM_PLUS	dead ^	dead `		dead ~	
E13	<X	VK_BACK	BS	BS	DEL		
D00	TAB	VK_TAB	HT	HT	HT		
D01	Q	'Q'	q	Q	DC1		
D02	W	'W'	w	W	ETB		
D03	E	'E'	e	E	ENQ		
D04	R	'R'	r	R	DC2		
D05	T	'T'	t	T	DC4		
D06	Z	'Z'	z	Z	SUB		
D07	U	'U'	u	U	NAK		
D08	I	'I'	i	I	HT		

Table 17-13 Swiss German to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
D09	O	'O'	o	O	SI		
D10	P	'P'	p	P	DLE		
D11	uUmlaut eGrav [	VK_OEM_5	u umlaut	e grave		[	
D12	" !]	VK_OEM_6	dead "	!		]	
C99	CTRL	VK_CONTROL					
C00	LOCK	VK_CAPITAL					
C01	A	'A'	a	A	SOH		
C02	S	'S'	s	S	DC3		
C03	D	'D'	d	D	EOT		
C04	F	'F'	f	F	ACK		
C05	G	'G'	g	G	BEL		
C06	H	'H'	h	H	BS		
C07	J	'J'	j	J	LF		
C08	K	'K'	k	K	VT		
C09	L	'L'	l	L	FF		
C10	oUmlaut eAcu {	VK_OEM_1	o umlaut	e acute		{	
C11	aUmlaut aGra }	VK_OEM_2	a umlaut	a grave		}	
C12	$ UKpound	VK_OEM_4	$	UKpound			
C13	RETURN	VK_RETURN	CR	CR	LF		
B99	SHIFT	VK_SHIFT					
B00	< > \	VK_OEM_3	<	>		\	
B01	Y	'Y'	y	Y	EM		
B02	X	'X'	x	X	CAN		

Table 17-13 Swiss German to ASCII Translation Table (cont.)

Keypos	Keycap	Virtkey	Unshift	Shift	Ctrl	Ctrl/Alt	Ctrl/Alt/Shift
B03	C	'C'	c	C	ETX		
B04	V	'V'	v	V	SYN		
B05	B	'B'	b	B	STX		
B06	N	'N'	n	N	SO		
B07	M	'M'	m	M	CR		
B08	, ;	VK_OEM_COMMA	,	;			
B09	. :	VK_OEM_PERIOD	.	:			
B10	- _	VK_OEM_MINUS	-	_			
B11	Shift	VK_SHIFT					
A99	Alt	VK_MENU					
A01-09	Space bar	VK_SPACE	blank	blank	NUL		

The remainder of the keyboard layout is the same as the U.S. version.

U.K. to ASCII Translation

U.K. keyboard mappings are identical to U.S., except that the VK_3 shifted output is the U.K. pound sign instead of the hash mark, and the 'extra' value (Ctrl/Alt/3) is the hash mark instead of the U.K. pound sign.

AnsiToOem, OemToAnsi

The keyboard driver supports two MS-Windows entry points AnsiToOem and OemToAnsi. These entry points translate character byte values between the OEM character set, which on the VAXmate is equivalent to the IBM PC character set, and the ANSI character set, also known as the ISO Latin-1.

ANSI to OEM Table

ANSI characters in the range 20h to 7Eh are the same as those in the OEM character set. ANSI characters from 00h to 1Fh and from 7Fh to A0h are non-printable control codes. Therefore, there is no translation from the ANSI character set to the OEM character set for these characters.

Table 17-14 documents the translation from characters A0h through FFh in the ANSI set to the OEM set.

Table 17-14 Translation of ANSI Set to OEM Set

ANSI Char	Description	OEM Char	Description (if different from ANSI)
A0H	no-break space (NBSP)	20H	space
A1H	inverted exclamation point	ADH	
A2H	cent sign	9BH	
A3H	Pound Sterling sign	9CH	
A4H	currency sign	0FH	
A5H	yen sign	9DH	
A6H	broken bar	7CH	
A7H	section sign	15H	
A8H	diaeresis	22H	quotation mark
A9H	copyright sign	63H	lowercase c
AAH	feminine ordinal indicator	A6H	
ABH	left angle quotation mark	AEH	
ACH	logical NOT	AAH	
ADH	hyphen	2DH	minus sign
AEH	registered trade mark	72H	lowercase r
AFH	macron	5FH	low line
B0H	ring above, degree sign	F8H	
B1H	plus-minus sign	F1H	
B2H	2 (superscript)	FDH	
B3H	3 (superscript)	33H	digit 3
B4H	acute accent	27H	apostrophe
B5H	Greek small mu, micro sign	E6H	

Table 17-14 Translation of ANSI Set to OEM Set (cont.)

ANSI Char	Description	OEM Char	Description (if different from ANSI)
B6H	pilcrow sign, paragraph sign	14H	
B7H	middle dot	FAH	
B8H	cedilla	2CH	comma
B9H	1 (superscript)	31H	digit 1
BAH	masculine ordinal indicator	A7H	
BBH	right angle quotation mark	AFH	
BCH	1/4	ACH	
BDH	1/2	ABH	
BEH	3/4	5FH	low line
BFH	inverted question mark	A8H	
C0H	A grave uppercase	41H	uppercase A
C1H	A acute uppercase	41H	uppercase A
C2H	A circumflex uppercase	41H	uppercase A
C3H	A tilde uppercase	41H	uppercase A
C4H	A diaeresis uppercase	8EH	
C5H	A ring uppercase	8FH	
C6H	AE diphthong uppercase	92H	
C7H	C cedilla uppercase	80H	
C8H	E grave uppercase	45H	uppercase E
C9H	E acute uppercase	90H	
CAH	E circumflex uppercase	45H	uppercase E
CBH	E diaeresis uppercase	45H	uppercase E
CCH	I grave uppercase	49H	uppercase I
CDH	I acute uppercase	49H	uppercase I
CEH	I circumflex uppercase	49H	uppercase I
CFH	I diaeresis uppercase	49H	uppercase I
D0H	Icelandic Eth uppercase	44H	uppercase D
D1H	N tilde uppercase	A5H	
D2H	O grave uppercase	4FH	uppercase O
D3H	O acute uppercase	4FH	uppercase O
D4H	O circumflex uppercase	4FH	uppercase O
D5H	O tilde uppercase	4FH	uppercase O
D6H	O diaeresis uppercase	99H	
D7H	multiplication sign	78H	lowercase x
D8H	O with oblique stroke uppercase	4FH	uppercase O
D9H	U grave uppercase	55H	uppercase U

Table 17-14 Translation of ANSI Set to OEM Set (cont.)

ANSI Char	Description	OEM Char	Description (if different from ANSI)
DAH	U acute uppercase	55H	uppercase U
DBH	U circumflex uppercase	55H	uppercase U
DCH	U diaeresis uppercase	9AH	
DDH	Y acute uppercase	59H	uppercase Y
DEH	Icelandic Thorn uppercase	5FH	low line
DFH	German sharp S lowercase	E1H	
E0H	a grave lowercase	85H	
E1H	a acute lowercase	A0H	
E2H	a circumflex lowercase	83H	
E3H	a tilde lowercase	61H	lowercase a
E4H	a diaeresis lowercase	84H	
E5H	a ring lowercase	86H	
E6H	ae diphthong lowercase	91H	
E7H	c cedilla lowercase	87H	
E8H	e grave lowercase	8AH	
E9H	e acute lowercase	82H	
EAH	e circumflex lowercase	88H	
EBH	e diaeresis lowercase	89H	
ECH	i grave lowercase	8DH	
EDH	i acute lowercase	A1H	
EEH	i circumflex lowercase	8CH	
EFH	i diaeresis lowercase	8BH	
F0H	Icelandic Eth lowercase	64H	lowercase d
F1H	n tilde lowercase	A4H	
F2H	o grave lowercase	95H	
F3H	o acute lowercase	A2H	
F4H	o circumflex lowercase	93H	
F5H	o tilde lowercase	6FH	lowercase o
F6H	o diaeresis lowercase	94H	
F7H	division sign	F6H	
F8H	o with oblique stroke lowercase	6FH	lowercase o
F9H	u grave lowercase	97H	
FAH	u acute lowercase	A3H	
FBH	u circumflex lowercase	96H	
FCH	u diaeresis lowercase	81H	
FDH	y acute lowercase	79H	lowercase y
FEH	Icelandic Thorn lowercase	5FH	low line
FFH	y diaeresis lowercase	98H	

OEM to ANSI Table

OEM characters in the range 20h to 7Eh are the same as those in the ANSI character set. Therefore, there is no translation. OEM characters from 00h to 1Fh are also not translated. Therefore, they are equivalent to the corresponding control code in the ANSI set.

Table 17-15 documents the translation from characters 80h through FFh in the OEM set to the ANSI set.

Table 17-15 Translation of OEM Set to ANSI Set

OEM Char	Description	ANSI Char	Description (if different from OEM)
80H	C cedilla uppercase	C7H	
81H	u diaeresis lowercase	FCH	
82H	e acute lowercase	E9H	
83H	a circumflex lowercase	E2H	
84H	a diaeresis lowercase	E4H	
85H	a grave lowercase	E0H	
86H	a ring lowercase	E5H	
87H	c cedilla lowercase	E7H	
88H	e circumflex lowercase	EAH	
89H	e diaeresis lowercase	EBH	
8AH	e grave lowercase	E8H	
8BH	i diaeresis lowercase	EFH	
8CH	i circumflex lowercase	EEH	
8DH	i grave lowercase	ECH	
8EH	A diaeresis uppercase	C4H	
8FH	A ring uppercase	C5H	
90H	E acute uppercase	C9H	
91H	ae diphthong lowercase	E6H	
92H	AE diphthong uppercase	C6H	
93H	o circumflex lowercase	F4H	
94H	o diaeresis lowercase	F6H	
95H	o grave lowercase	F2H	
96H	u circumflex lowercase	FBH	
97H	u grave lowercase	F9H	
98H	y diaeresis lowercase	FFH	
99H	O diaeresis uppercase	D6H	
9AH	U diaeresis uppercase	DCH	
9BH	cent	A2H	
9CH	Pound Sterling sign	A3H	
9DH	yen	A5H	
9EH	Peseta sign or point sign (Pt)	70H	lowercase p
9FH	Function sign (curved f)	66H	lowercase f

Table 17-15 Translation of OEM Set to ANSI Set (cont.)

OEM Char	Description	ANSI Char	Description (if different from OEM)
A0H	a acute lowercase	E1H	
A1H	i acute lowercase	EDH	
A2H	o acute lowercase	F3H	
A3H	u acute lowercase	FAH	
A4H	n tilde lowercase	F1H	
A5H	N tilde uppercase	D1H	
A6H	feminine ordinal indicator	AAH	
A7H	masculine ordinal indicator	BAH	
A8H	inverted question mark	BFH	
A9H	Reverse logical NOT sign	5FH	low line
AAH	logical NOT sign	ACH	logical NOT
ABH	1/2	BDH	
ACH	1/4	BCH	
ADH	inverted exclamation point	A1H	
AEH	left angle quotation mark	ABH	
AFH	right angle quotation mark	BBH	
B0H	graphic character	20H	space
B1H	graphic character	20H	space
B2H	graphic character	20H	space
B3H	graphic character	7CH	vertical line
B4H	graphic character	2BH	plus sign
B5H	graphic character	2BH	plus sign
B6H	graphic character	2BH	plus sign
B7H	graphic character	2BH	plus sign
B8H	graphic character	2BH	plus sign
B9H	graphic character	2BH	plus sign
BAH	graphic character	7CH	vertical line
BBH	graphic character	2BH	plus sign
BCH	graphic character	2BH	plus sign
BDH	graphic character	2BH	plus sign
BEH	graphic character	2BH	plus sign
BFH	graphic character	2BH	plus sign
C0H	graphic character	2BH	plus sign
C1H	graphic character	2BH	plus sign
C2H	graphic character	2BH	plus sign
C3H	graphic character	2BH	plus sign
C4H	graphic character	2DH	minus sign

Table 17-15 Translation of OEM Set to ANSI Set (cont.)

OEM Char	Description	ANSI Char	Description (if different from OEM)
C5H	graphic character	2BH	plus sign
C6H	graphic character	2BH	plus sign
C7H	graphic character	2BH	plus sign
C8H	graphic character	2BH	plus sign
C9H	graphic character	2BH	plus sign
CAH	graphic character	2BH	plus sign
CBH	graphic character	2BH	plus sign
CCH	graphic character	2BH	plus sign
CDH	graphic character	3DH	equal sign
CEH	graphic character	2BH	plus sign
CFH	graphic character	2BH	plus sign
D0H	graphic character	2BH	plus sign
D1H	graphic character	2BH	plus sign
D2H	graphic character	2BH	plus sign
D3H	graphic character	2BH	plus sign
D4H	graphic character	2BH	plus sign
D5H	graphic character	2BH	plus sign
D6H	graphic character	2BH	plus sign
D7H	graphic character	2BH	plus sign
D8H	graphic character	2BH	plus sign
D9H	graphic character	2BH	plus sign
DAH	graphic character	2BH	plus sign
DBH	graphic character	20H	space
DCH	graphic character	20H	space
DDH	graphic character	20H	space
DEH	graphic character	20H	space
DFH	graphic character	20H	space
E0H	Alpha	5FH	low line
E1H	Beta	DFH	
E2H	Gamma	5FH	low line
E3H	Pi	B6H	pilcrow sign, paragraph sign
E4H	sigma uppercase	5FH	low line
E5H	sigma lowercase	5FH	low line
E6H	mu lowercase	B5H	
E7H	tau lowercase	5FH	low line
E8H	phi uppercase	5FH	low line
E9H	theta uppercase	5FH	low line
EAH	omega uppercase	5FH	low line

Table 17-15 Translation of OEM Set to ANSI Set (cont.)

OEM Char	Description	ANSI Char	Description (if different from OEM)
EBH	delta lowercase	5FH	low line
ECH	infinity sign	5FH	low line
EDH	math empty set or phi lowercase	5FH	low line
EEH	math own sign	5FH	low line
EFH	math intersection sign	5FH	low line
F0H	math equivalence sign	5FH	low line
F1H	plus-minus sign	B1H	
F2H	greater than or equal sign	5FH	low line
F3H	less than or equal sign	5FH	low line
F4H	math integral upper part	5FH	low line
F5H	math integral lower part	5FH	low line
F6H	divide sign	F7H	
F7H	math roughly equals sign	5FH	low line
F8H	degree sign	B0H	
F9H	bold dot	B7H	middle dot
FAH	middle dot	B7H	
FBH	square root	5FH	low line
FCH	n superscript	6EH	lowercase n
FDH	2 superscript	B2H	
FEH	black box (or diaeresis)	A8H	diaresis
FFH	space	20H	

Mouse

The VAXmate uses the DIGITAL 3-button mouse. Movement of the mouse and/or button transitions on the mouse results in standard MS-Windows messages.

Communications

Communications on the VAXmate under MS-Windows may take place via an asynchronous serial communications device, parallel device or over ethernet with the support of the DIGITAL LAT driver.

Full asynchronous serial and parallel communications device support is provided as defined in the *MS-Windows Software Development Kit Programmer's Reference Manual.* In addition, DIGITAL adaptation allows for the LPTx ports which are redirected to the DIGITAL Serial Printer Port (SPP) to be accessed. Redirection of an LPTx port to a network device is also allowed. Redirection of an LPTx port to a COMx port is not allowed.

LAT support is an enhancement to DIGITAL's MS-Windows product. Support is in two forms. First, there is a custom application interface to the DIGITAL LAT driver. Second, there is a mapping of the MS-Windows RS232 asynchronous serial communications interface to LAT functions.

The Windows LAT support driver supports up to four sessions or circuits. This is due to the fact that only four LAT Control Blocks (LCBs) are available for use in the MS-Windows communications driver. Any attempt to open a fifth session or circuit results in an out-of-memory initialization error return code.

LAT Support Through the Windows Asynchronous Serial Communications Interface

The RS232 communications functions in the standard Windows adaptation can optionally be mapped to LAT functions. It has certain restrictions and is under the user's control. The LAT support driver is transparent to applications which are aware of the Windows asynchronous communications interface.

In order to provide LAT via the asynchronous serial communications interface, the asynchronous serial communication devices must be logically mapped to LAT services. The user of Windows is required to associate the desired LAT service name with one of the asynchronous serial communication devices, namely COM1 or COM2. If the communications device is mapped to a LAT service, then applications which use the standard asynchronous serial communications routines supplied under Windows will have the calls automatically redirected to the LAT support driver. The user's mapping of asynchronous devices to LAT services is supported in the Control Panel. The user's selection is saved in the WIN.INI file. See the section on the Control Panel application in the *VAXmate User's Guide.*

Each MS-Windows supplied and defined serial communication routine is mapped to an appropriate LAT function.

Not all error return codes are 100% meaningful when mapping asynchronous communications functions to LAT functions. The most meaningful error defined by Windows was chosen to indicate errors in the LAT support driver.

In the following section, each Windows communications function is followed by a one sentence description of its purpose. (More detail may be obtained from the *MS-Windows Programmer's Reference Manual.*) Each description is followed by the routine's functionality when redirected to LAT.

OpenComm

This routine handles the opening of a communications device. A LAT session is opened if:

1. The LAT driver has been loaded. If not, the OPEN request is handled by the asynchronous communications driver.
2. The device ID is 0 or 1. If not, the OPEN request is handled by the asynchronous communications driver.
3. The mode of transmission is computer to computer. Therefore, the FDTRFLOW, FRTSFLOW, FOUTXCTSFLOW, and FOUTXDSRFLOW flags in the serial communications DCB must be 0. If not, the open call is handled by the serial communications open routine.
4. The transmit/receive byte size is valid. If less than 4 or greater than 8, an illegal byte size (IE_BYTESIZE) error is returned.
5. There is an available LCB. If not, the open call is handed off to the serial communications driver.
6. A service name is present in the WIN.INI file for the serial communications device being opened. If not, the open call is handed off to the serial communications open routine.
7. The LAT service is available. If the LAT service is unavailable, an invalid or unsupported ID error is returned (IE_BADID).
8. The serial communications DCB is copied into the LCB in case the application performs a GetCommState call.

All subsequent Windows communications functions are handled by the LAT support routine if a successful open-under-LAT was previously performed. Otherwise, the call is handled by the asynchronous communications driver.

WriteComm

This Windows routine handles write operations to the communications device. The following actions are performed:

1. Check for a LAT session fail or stop. If so, the break-event bit, EV_BREAK, is set in the communications event word. The communications event word can be read by the GetCommEventMask or SetCommEventMask functions. The communications error code CE_BREAK is also returned indicating the LAT session failure or stop.
2. The transmit/receive byte size passed in the serial communications Device Control Block (DCB) during the OpenComm call is used to mask unwanted data bits.
3. The character passed is transmitted. If unable to transmit, a transmit queue full error (CE_TXFULL) is logged and can be retrieved by doing a GetCommError call.

TransmitCommChar

This LAT support routine transmits a character immediately, just as any other character. Therefore, its functionality is identical to the WriteComm routine.

ReadComm

This routine reads the communications device. The following actions are performed:

1. Check for a LAT session fail or stop. If so, no characters are returned for the read. Rather, the break-event bit, EV_BREAK, is set in the communications event word. Also, the communications error code CE_BREAK is returned indicating the LAT session failure or stop.
2. The end of file flag is checked. If set, the EOF character is returned.
3. A character is read. If a character is available, steps 4-7 are performed.
4. The transmit/receive byte size passed in the serial communications Device Control Block (DCB) during the OpenComm call is used to mask unwanted data bits.
5. If the strip receive null flag (FNULL) passed in the DCB is set and the character received is a null, no character is returned to the caller.
6. If the binary flag (FBINARY) in the DCB is not set, check for the EOF character as passed in the DCB (EOFCHAR). If character read is the EOF, it is returned to the caller and the EOF status flag (FEOF) is set. The FEOF status flag may be retrieved by doing a GetCommError call.
7. The event character (EVTCHAR) passed in the DCB is checked against the read character. If equal, the event is logged and can be retrieved by calling SetCommEventMask or GetCommEventMask.

CloseComm

This routine closes the communication device. The following actions are performed:

1. The LAT session is unconditionally closed.
2. Data structures allocated to the session are freed.

SetCommState

This routine sets parameters in the serial communications DCB. The following actions are performed:

1. The FDTRFLOW, FRTSFLOW, FOUTXCTSFLOW, and FOUTXDSRFLOW flags in the serial communications DCB must all be reset to indicate computer-to-computer transmission. If not, the LAT session is closed and a call is made to the asynchronous communications driver's OPEN routine. Otherwise, steps 2-3 are performed.
2. The transmit/receive byte size is checked. If less than 4 or greater than 8, an illegal byte size (IE_BYTESIZE) error is returned.
3. The serial communications DCB is copied into the LCB.

GetCommState

This routine fills a buffer with the serial communications DCB.

EscapeCommFunction

This LAT support routine performs extended communication functions. It does nothing except exit with the current device error word.

SetCommBreak

This LAT support routine puts the communications device in a break state. It performs the following function:

1. SendCommBreak sends a break to the host and exits with the current device error word. If the break cannot be sent, the CE_TXFULL bit is set in the communications error word.

ClearCommBreak

This LAT support routine clears the communication device's break state. It does nothing except exit with the current device error word.

SetCommEventMask

This LAT support routine enables and retrieves the event mask. Its functionality is identical to the asynchronous communications driver.

GetCommEventMask

This LAT support routine returns and clears the event mask. Its functionality is identical to the asynchronous communications driver.

FlushComm

This LAT support routine flushes characters from the transmit or receive queue. Its functionality is identical to the asynchronous communications driver.

GetCommError

This LAT support routine fills a communications status buffer and returns the communications error word if an error occurred since the last GetCommError. Its functionality is identical to the asynchronous communications driver. The only flag which may ever be set in STFLAGS in the status buffer is the EOF character flag (FEOF). The only communications device error bit which may ever be set in the communications error word are CE_TXFULL and CE_BREAK.

The Windows LAT interface ignores RS232 specific parameters which are part of the LAT Device Control Block data structure (e.g., baud rate, parity, stop bits, etc.).

Custom LAT Application Interface Under Windows

Applications that are aware of LAT may use custom functions provided under DIGITAL's adaptation of MS-Windows. These functions are a direct interface to LAT from the application. Parameters passed and returned are specific to LAT. The interface does not attempt to emulate the asynchronous communications interface provided under MS-Windows.

There are eight custom application interface functions provided to support the interface to LAT. They are described in detail below.

OpenLat (lpServiceName, lpNodeName, lpPortName) : Latid

This routine opens a session to a LAT-supported service and assigns a LATID handle to it. The routine allocates the data structures for the LAT session including space for the receive and transmit queues.

Parameters

lpServiceName	Is a long pointer to a null-terminated string that contains the requested service name. It may be 1-18 characters in length.
lpNodeName	Is a long pointer to a null-terminated string that contains the requested node name. This parameter may be NULL or up to 18 characters in length.
lpPortName	Is a long pointer to a null-terminated string that contains the requested port name. This parameter may be NULL or up to 18 characters in length.

Returns

Latid	Is an integer value identifying the opened communication device. If Latid is negative an initialization error occurred.
IE_LATINSTALL	Is returned if the LAT driver was not installed.
IE_LATSERVICE	Is returned if the requested service is unavailable.
IE_LATMEMORY	Is returned if unable to allocate memory for the LAT data structures.
IE_LATSESSIONS	Is returned if no sessions are available.
IE_LATCIRCSESS	Is returned if no sessions are available on the circuit.
IE_LATVIRTCIRC	If no more virtual circuit blocks are available.
IE_LATBUFFER	Is returned if there is a data buffer specification error (internal LAT driver error).

CloseLat (Latid) : Result

The routine closes the LAT session specified by the Latid and frees all the data structures associated with the session.

Parameters

Latid	is an integer value identifying the LAT session to be closed.

Returns

Result	is an integer value specifying the result of the routine. = 0 (CE_LATOK) if the session was closed. = a negative number if there was an error. = CE_LATID if there is no session associated with Latid. = CE_LATSTOP if the LAT circuit failed or was stopped.

ReadLat (Latid) : Result

This routine attempts to read a character from the receive queue for a session as specified by Latid.

Parameters

Latid is an integer value identifying the LAT session to be read from.

Returns

The low order 8 bits of Result contain the read character.

= a negative value if no character was read or there was an error.

= CE_LATID if there is no session associated with Latid.

= CE_LATSTOP if the LAT circuit failed or was stopped.

= CE_LATNOCHAR if no character was available.

WriteLat (Latid, ch) : Result

This routine writes a character to the transmit queue for the session specified by Latid.

Parameters

Latid	is an integer value identifying the LAT session to which the character is queued.
ch	is the 8-bit value of the character to write to the transmit queue.

Returns

Result	= 0 (CE_LATOK) if the write was successful.
	= a negative value if the write was unsuccessful.
	= CE_LATID if there is no session associated with Latid.
	= CE_LATTXQUE if unable to queue the character.
	= CE_LATSTOP if the LAT circuit failed or was stopped.

GetLatStatus (Latid) : Result

This routine is used to get the status of the LAT session specified by Latid.

Parameters

Latid	is an integer value identifying the LAT session to get status from.

Returns

Result	is an unsigned integer value whose bits, when set, indicate LAT status. The bits set can be any combination of the following:	
	ST_LATREC	Receive data is available.
	ST_LATTXQUE	Unable to queue transmit data.
	ST_LATSESINACT	Lat session is not active.
	ST_TXEMPTY	Transmit buffer is empty.
Result	= a negative value if the status call was in error.	
	= CE_LATID if there is no session associated with Latid.	
	= CE_LATSTOP if the LAT circuit failed or was stopped.	

SendLatBreak (Latid) : Result

This routine causes the LAT driver to send a break to the host.

Parameters

Latid	is an integer value identifying the LAT session over which the break is sent.

Returns

Result	= 0 (CE_LATOK) if the break was sent. = a negative value if the break was not sent. = CE_LATID if there is no session associated with Latid. = CE_LATSTOP if the LAT circuit failed or was stopped. = CE_LATBRK if unable to transmit the break because a buffer or transmit credit is not available.

InquireLatServices ( ) : LResult

This routine asks the LAT driver to return the maximum number of service name entries in its service table. The number of actual service names available may be less. It also resets the service name counter so that the first GetLatService call returns the first name in the service table. If the LAT service table has overflowed the caller is informed.

Parameters

None

Returns

LResult	is a long (32) bit integer. The high word of LResult indicates error return codes. The low word of LResult is the maximum number of services in the LAT driver's table.
High word	= IE_LATINSTALL if the LAT driver is not installed. = IE_LATOVERFLOW if the LAT driver's service table overflowed. = zero if no errors are returned. = IE_LATOVERFLOW or 0, the low word of LResult is the maximum number of services in the LAT driver's table. Otherwise, the low word is undefined.

GetLatService (lpServiceName) : Result

This routine asks the LAT driver for the next service name in its table. It fills the buffer passed with the service name. An InquireLatServices call must be made before the first call to GetLatService.

Parameters

lpServiceName	is a long pointer to a character string buffer containing a null terminated string. This string is the LAT service name. The actual service name may be up to 16 characters in length. Therefore, allocate a buffer of at least 17 bytes.

Returns

Result	is the number of service names remaining in the LAT driver's table. = 0 indicates the last name in the list is being returned. = a negative number if there was an error. = IE_LATINSTALL if the LAT driver was not installed. = IE_LATNOSERVNAME if no service name is being returned because 1) the InquireLatServices function was not called, or 2) the end of the service name table was reached on a previous GetLatService call.

IMPORTANT
Result may be decremented by more than one from a previous call. This is due to the LAT driver's filtering out of duplicate service names. Note too that because of this feature zero may never be returned. Therefore, programs must loop while Result not equal IE_LATNOSERVNAME.

If the Windows LAT interface detects a circuit failure or stop (CE_LATSTOP), the application's virtual connection to the communication device is closed. The Windows LAT interface driver invalidates the applications LAT session ID. It also automatically deallocates any data structures associated with the LAT session. The application may make a CloseLat function call, but is not required.

The application's .DEF file must contain an import statement for each of the custom LAT routines it uses as follows (these statements are in DECWIN.H):

```
IMPORTS
comm.OpenLat
comm.CloseLat
comm.ReadLat
comm.WriteLat
comm.GetLatStatus
comm.SendLatBreak
comm.InquireLatServices
comm.GetLatService
```

The application must declare the following for each routine it uses:

```
extern int  FAR PASCAL OpenLat (LPSTR, LPSTR, LPSTR);
extern int  FAR PASCAL WriteLat (int, char);
extern int  FAR PASCAL GetLatStatus (int);
extern int  FAR PASCAL ReadLat (int);
extern int  FAR PASCAL CloseLat (int);
extern int  FAR PASCAL SendLatBreak (int);
extern long FAR PASCAL InquireLatServices ();
extern int  FAR PASCAL GetLatService (LPSTR);
```

Display on the VAXmate

The VAXmate video controller when running under MS-Windows is configured to operate in the 640x400 2-color graphics mode. This mode has twice the vertical resolution as the industry-standard color graphics adapter. This mode allows for smoother looking graphics and the use of a higher quality font.

A custom font is supplied for the display resolution in order to support DIGITAL's VT220 Terminal Emulator, which runs under MS-Windows. Other applications may also use this font.

The DIGITAL Terminal Emulation font has an 8x14 and a 6x9 (width x height) character cell for single-high/single-wide characters and a 16x14 and 12x9 character cell for double-wide characters. Fonts are provided for double-wide/double-high top and double-wide/double-high bottom. When combined, character cell sizes of 16x28 and 12x18 are realized. The character cell size selection allows for 24 text lines of display in a full screen window which contains a caption area and a horizontal scroll bar.

Each font has a unique face name so that they can be enumerated and distinguished in size. The face names are listed below along with the character cell description and cell size.

DECTerm	Single-high/single-wide character, cell size 8x14
DECTerm Small	Single-high/single-wide character, cell size 6x9
DECTerm Dbl-Wide	Single-high/double-wide character, cell size 16x14
DECTerm Small Dbl-Wide	Single-high/double-wide character, cell size 12x9
DECTerm Dbl-Size Upper	Top half of double-high/double-wide character, cell size 16x14
DECTerm Dbl-Size Lower	Bottom half of double-high/double-wide character, cell size 16x14
DECTerm Small Dbl-Size Upper	Top half of double-high/double-wide character, cell size 12x9
DECTerm Small Dbl-Size Lower	Bottom half of double-high/double-wide character, cell size 12x9

See the DECWIN.H listing for the symbolic constants that should be used by applications when accessing the character set.

There is one character set provided in the font. The character set in the font is a superset of the ANSI Character Set. It is designated by the character set ID DECTERM_CHARSET in the DECWIN.H file. The ANSI Character Set has characters in positions 21h-7Eh and A1h-FFh. The DIGITAL Terminal Emulation fonts character set provides all those characters in the same positions. Included are the newly ISO approved times and divide signs in positions D7h and F7h, respectively.

Three characters, which are in the DIGITAL Multinational character set but which are not represented in the ANSI Character Set for Windows, occupy positions 9Dh-9Fh in the DIGITAL Terminal Emulation Font. The three characters are the upper and lower case oe ligature and the lower case y-umlaut.

The reverse question mark, used to represent communications errors, occupies position 9Ch.

Characters in positions 60h-7Eh in the DIGITAL Special Graphics Character Set (also known as the VT100 line drawing set) occupy positions 00h-1Eh in the DIGITAL Terminal Emulation Font.

> NOTE
> The ANSI Character Set for Windows is equivalent to the ISO Latin-1 character set.

Standard Applications Support

MS-Windows provides support for standard MS-DOS applications. These applications are not designed to run in the MS-Windows environment. The old applications driver is responsible for managing the invocation and operation of the standard applications on a per task basis. All system resources are managed so that the standard application may co-exist within the MS-Windows environment and with other MS-Windows applications.

The old application support module is essentially the same as the support provided in the standard version with a few exceptions.

- Keyboard handling
- ANSI Support
- Video modes handled
- Interrupt 11 Support
- Interrupt 12 Support
- Interrupt 15 Support
- Memory requirements
- Unique Icons

Keyboard Handling

Keyboard Handling Inside an MS-Windows Window

If the standard application is running within an MS-Windows window, the application has access to Interrupt 16h functions. While the old applications driver is being enabled, the old interrupt vector is read and saved. A new vector is set for interrupt 16h which points to a routine within the old applications driver. In this manner, Interrupt 16h functions are intercepted by the old applications driver and filtered appropriately. The following functions are supported.

Normal Functions

Fetch Next Character Input From Keyboard

Parameters

AH = 0

Returns

AH = Scan Code
AL = ASCII Character

Test For Character Available

Parameters

AH = 1

Returns

AH = Scan Code
AL = ASCII Character
ZF = 0 Character is available
ZF = 1 No Character is available

Return Current Shift Status Flags

Parameters

AH = 2

Returns

Current shift status flags
AL = Contents of Keyboard Status Flag

Extended Functions:

Enter DEC mode

Parameters

AH = D5
AL = AC

Returns

Nothing

Exit DEC Mode

Parameters

AH = D5
AL = AD

Returns

Nothing

Enable/Disable Additional Key Codes

Parameters

AH = D3	Enable/disable additional key codes associated with LK250 keyboard.
AL = #	Each bit with enable/disable special functions, all are processed.

Extended functions not supported:

Key Notification

Parameters

AH = D0

Returns

Nothing

Character Count

Parameters

AH = D1

Returns

Nothing

Keyboard Buffer

Parameters

AH = D2

Returns

Nothing

Request Keyboard ID

Parameters

AH = D4

Returns

Nothing

Get/Set Table Pointer

Parameters

AH = D6

Returns

Nothing

If the keyboard is in DIGITAL-extended mode, the numeric keypad returns numeric values at all times. The scan codes returned are compatible with those returned while outside of MS-Windows when the keyboard is in DIGITAL-extended mode.

Function D5h, which sends a command byte to the keyboard, is allowed if the command is to enter or exit DIGITAL-extended mode (ie, ACh or ADh). All other commands via D5 return to the caller. Function D3h, which sets ROM BIOS keyboard states, is allowed only for bit 0 (ie, set/reset numpad states).

Functions such as Ctrl/C and Ctrl/S are supported as they normally are in an intrinsic MS-DOS environment. Ctrl/P is not supported by the MS-Windows old applications driver.

Keyboard Handling Outside an MS-Windows Window

While managing a standard application that runs outside of an MS-Windows window, the old applications driver takes over interrupt vector 9, the keyboard interrupt. This vector is taken over so key strokes may be monitored for program switch and screen exchange. The key strokes are Alt/Tab or Alt/Enter for program switch. Alt/Prt Sc is monitored for screen exchange. If these keys are not pressed, a call is made to the previous interrupt 9 handler.

When a standard application is run, the keyboard is set to industry-standard mode. When the application is exited, the keyboard is put back into DIGITAL-unique mode. This is accomplished from routines within the old applications driver, which are called when there is a change in the keyboard input focus.

While the application is running, the user may wish to temporarily suspend the standard application and return to MS-Windows without terminating the standard application. If the PIF file associated with the standard application allows for program switching, then the keyboard state is remembered along with the video state and video memory. Both are restored when the standard application is reactivated.

ANSI Support Inside an MS-Windows Window

The MS- Windows old applications driver supports most ANSI escape sequences that are supported by ANSI.SYS for standard applications that run in an MS-Windows window. The following is a list of functions which are not supported:

- Cursor Position Report
- Set Mode
- Keyboard Reassignment
- Set Graphics Rendition
  - Faint on
  - Italic on
  - Rapid blink on
  - Subscript
  - Superscript

Video Modes Handled Inside an MS-Windows Window

Standard applications running in an MS-Windows window may access certain ROM BIOS video functions provided by the old applications driver (WINOLDAP). Table 17-16 contains the Interrupt 10h functions available and indicates how they can be used.

Table 17-16 **Interrupt 10H Functions**

Function Name	Function #	Windows Response
Set mode	0	Ignored
Set cursor type	1	Emulated by WINOLDAP
Set cursor position	2	Emulated by WINOLDAP
Get cursor position	3	BH= Active page (ignore this) DH,DL = Row,col CH,CL = Cursor mode
Get light pen position	4	AX = 0 (no light pen)
Set active page	5	Ignored
Scroll active page up	6	Emulated by WINOLDAP
Scroll active page down	7	Emulated by WINOLDAP
Get character and attribute at cursor	8	BH is ignored AH,AL = attribute,character
Write character and attribute at cursor	9	Emulated by WINOLDAP
Write character string at cursor	10	Emulated by WINOLDAP
Set color palette	11	Ignored
Write dot	12	Ignored
Read dot	13	Ignored
Write TTY	14	Emulated by WINOLDAP
Get video state	15	AL = mode (always 7, monochrome) AH = MaxCol (80 columns) BH = Current active display page (always 0)
Set palette reg EGA	16	Ignored
Char gen EGA	17	Ignored

Table 17-16 Interrupt 10H Functions (cont.)

Function Name	Function #	Windows Response
Alternate select EGA	18	Ignored
Write string EGA	19	All ignored AL = 0 BL attribute for all characters, string is CX characters and the cursor is not moved AL = 1 BL attribute for all characters, string is CX characters and the cursor is moved AL = 2 String is CX characters, attrib pairs and the cursor is not moved AL = 3 String is CX characters, attrib pairs and the cursor is moved
TopView get video buffer address	FE	Returns a pointer to the buffer address
TopView update video buffer	FF	The display is updated

Associated with the old applications driver is a screen grabber that captures text and graphics video. It supports both industry-standard video modes and DIGITAL-unique video modes. Table 17-17 contains the supported video modes.

Table 17-17 Supported Video Modes

Mode	Description
Mode 0	40x25 Black/White
Mode 1	40x25 Color
Mode 2	80x25 Black/White
Mode 3	80x25 Color
Mode 4	320x200 Color
Mode 5	320x200 Black/White
Mode 6	640x200 Black/White
Mode 7	80x25 IBM monochrome
Mode D0h	640x400 Black/White (DIGITAL-unique)
Mode D1h	640x400 Color (DIGITAL-unique)
Mode D2h	800x250 Color (DIGITAL-unique)

Interrupt 11h Support

This interrupt returns the equipment available on the system to the caller. This interrupt is managed for applications running in an MS-Windows window. While the interrupt is not intercepted, the location, 40:10h is read and then ORed with bits that always indicate the system is running with a 80x25 BW card. The ORing in effect takes over the interrupt because the Interrupt 11h call reads the location. When the standard application exits, the original state of 40:10h is restored.

Interrupt 12h Support

Interrupt 12h calls are filtered for standard applications running in an MS-Windows window by the old applications driver. This function returns the memory size to the application. This is performed on a task basis and really does not indicate total memory, but the amount of memory available to the task, that is, the size of applications partition.

Interrupt 15h Support

Interrupt 15h is revectored to a handler within the old applications driver for standard applications running in MS-Windows. All functions normally supported by interrupt 15h are passed on to the firmware. Two Topview calls are emulated in the handler; one tells the caller that Topview is present while the other indicates it is Topview Version 1. The DIGITAL-extended D0h function ANDs the value passed back from the real Interrupt 15h call to simulate a IBM monochrome adapter is present. This maintains consistency with the Get Mode Interrupt 10h call and the Interrupt 11h equipment check.

Unique Icons

When a standard application is run, the user may want a more visually appealing and descriptive icon associated with the application. For most standard applications, the first three letters of the application name appear in a white box as the icon. This is a generic icon. However, the user can create an icon (using ICONEDIT.EXE) and save the icon (of the form AppName.ICO) somewhere on the path. This icon is associated with the application of the same name. The icon is seen if the application can switch to/from MS-Windows by pressing the Alt/Tab keys or if the application is loaded (runs as an icon). If the icon (.ICO file) is not found on the path, the three letter functionality is used as before.

For the generic icon, small dots appear near the bottom of the box to denote multiple instances. For unique icons, dots do not appear for multiple instances.

Printers

Full GDI support for the LN03PLUS (with cartridge) and the LA75 Companion printers is supplied. The LA75 is supported in both the DIGITAL mode and STD (industry-standard) mode. Printing from MS-Windows may either be local or remote over the ethernet.

All the printer drivers support the ISO Latin-1 character set. The LA50, LN03, LN03PLUS, and LA75DEC use the fallback representations for characters not in its ROM font.

The printer drivers also support all of the character sets native to the printer. These include various National Replacement Character (NRC) sets and DIGITAL-unique character sets. Applications may select these character sets using standard MS-Windows functions passing the constants listed in the DECWIN.H file listing. Table 17-18 indicates the character sets supported by each printer.

Table 17-18 Character Set Supported by Each Printer

Character Set	LA50	LN03	LN03PLUS	LA75DEC	LA75STD
ISO Latin-1	X	X	X	X	X
Industry Standard STD	X				
United Kingdom NRC	X	X	X	X	
French NRC	X	X	X	X	
German NRC	X	X	X	X	
Italian NRC	X	X	X	X	
Danish NRC	X	X	X	X	
Norwegian NRC	X	X	X	X	
Spanish NRC	X	X	X	X	
Swedish NRC	X	X	X	X	
Japanese (JIS Roman) NRC	X	X	X	X	
Japanese Katakana Graphic	X	X			
Finnish NRC	X	X	X	X	
French Canadian NRC	X	X	X	X	
Dutch NRC		X	X		
Swiss NRC		X	X		
Portuguese NRC					
DIGITAL Multinational	X	X	X	X	
DIGITAL Special (VT100)	X	X	X	X	
DIGITAL Technical		X	X		
DIGITAL Publishing					

Refer to the DECWIN.H file for constants that define these character sets.

DECWIN.H File Listing

A C programming language include file that documents constants and routine declarations follows. Values of symbolic constants used in earlier sections are documented in this include file.

```
/*  DECWIN.H

    This collection of constants and routine declarations details
    information specific to DIGITAL's Adaptation of MS-Windows for
    the VAXmate.

    ----------------------------------------------------------------------

                              Copyright (c) 1986 by
                  Digital Equipment Corporation, Maynard, Mass.

    This software is furnished under a license and may be used and  copied
    only  in  accordance  with  the  terms  of  such  license and with the
    inclusion of the above copyright notice.  This software or  any  other
    copies  thereof may not be provided or otherwise made available to any
    other person.  No title to and ownership of  the  software  is  hereby
    transferred.

    The information in this software is subject to change  without  notice
    and  should  not  be  construed  as  a commitment by Digital Equipment
    Corporation.

    DIGITAL assumes no responsibility for the use or  reliability  of  its
    software on equipment which is not supplied by DIGITAL.
    ----------------------------------------------------------------------

*/

/**************************************************************************
*                                                                         *
*       Version 1.01.11       07/21/86                                    *
*       DECWIN.H  Application layer include file for DIGITAL extensions *
*       for MS-Windows version 1.01                                       *
*                                                                         *
**************************************************************************/


/*
   These three routines and the following constants are used to switch
   keyboard states according to user preference. They handle the state
```

```
   of the LOCK key (caps/shift), keyclick volume, and the autorepeat
   on/off state.
*/
extern int FAR PASCAL DecSetLockState();
extern int FAR PASCAL DecSetKClickVol();
extern int FAR PASCAL DecSetAutorep();

#define  DEC_CAPSLOCK      0    /*  lock sense = capslock   */
#define  DEC_SHIFTLOCK     1    /*  lock sense = shift      */

#define  DEC_NOSOUND       0     /* values for keyclick volume */
#define  DEC_SOFT          1
#define  DEC_INTERMED      2
#define  DEC_LOUD          3

#define  DEC_AUTOREPOFF    0     /* state of autorepeat     */
#define  DEC_AUTOREPON     1

/* DecSetComposeState routine allows an application to switch between
   ISO mode compose sequences (the default) and DIGITAL Multinational
   Compose sequences. An application using this routine to receive
   DIGITAL Multinational sequences should call it with the
DEC_MULTINAT_COMP value upon getting the keyboard input focus, and
   MUST CALL IT AGAIN with the DEC_ISO_COMP value upon losing keyboard
   input focus. If the second call is not made, other applications will
   be receiving DIGITAL Multinational sequences when they are expecting
   ISO.
*/

extern int FAR PASCAL DecSetComposeState();

#define DEC_ISO_COMP      0              /* select ISO compose sequences *
#define DEC_MULTINAT_COMP 1              /* select DEC Multinational seqs


/* DecSetNumlockMode routine allows an application to switch between
   Numlock interpretation VK_OEM_NUMBER of the PF2 key on the numeric
   keypad and a unique interpretation VK_OEM_PF2 of the key. The default
   is Numlock enabled. An application desiring this functionality should
   call this routine with the DEC_NONumlock value upon getting the
   keyboard input focus. The application MUST RE-ENABLE Numlock PROCESSI
   upon losing the keyboard input focus. If the second call is not made,
   other applications will get unexpected keyboard output.
*/
extern int FAR PASCAL DecSetNumlockMode();

#define DEC_Numlock     0               /* enable Numlock interpretation
```

```
   #define DEC_NONumlock   1              /* disable Numlock interpretation
*/


   /* Decfonts used for terminal emulation.
      The following constants are used to select a specific font.

      The following character set constant is used in selecting any
      DECTERM.FON specific font. Each font variation in the set has a unique
      face name.
   */

   #define DECTERM_CHARSET              1

   /* DIGITAL Standard Terminal Character Set */

   #define DECTERM_NORMAL               "DECTerm"
   #define DECTERM_SMALL                "DECTerm Small"

   /* DIGITAL Double Wide Terminal Character Set */

   #define DECTERM_WIDE                 "DECTerm Dbl-Wide"
   #define DECTERM_SMALL_WIDE           "DECTerm Small Dbl-Wide"

   /* DIGITAL Double High Terminal Character Set (Top and Bottom halves) */

   #define DECTERM_DBL_TOP              "DECTerm Dbl-Size Upper"
   #define DECTERM_DBL_BOTTOM           "DECTerm Dbl-Size Lower"
   #define DECTERM_SMALL_DBL_TOP        "DECTerm Small Dbl-Size Upper"
   #define DECTERM_SMALL_DBL_BOTTOM     "DECTerm Small Dbl-Size Lower"


   /*  DecGetKbdCountry returns the nationality of the current keyboard
       specified in the list below. */

   extern int FAR PASCAL DecGetKbdCountry();

   /* Return codes from DecGetKbdCountry */

   #define DEC_USA              1           /* U.S. keyboard */
   #define DEC_BRITAIN          2           /* British keyboard */
   #define DEC_FRANCE           3           /* French keyboard */
   #define DEC_WEST_GERMANY     4           /* German keyboard */
   #define DEC_ITALY            5           /* Italian keyboard */
   #define DEC_SPAIN            6           /* Spanish keyboard */
   #define DEC_SWEDEN           7           /* Swedish keyboard */
   #define DEC_FINLAND          8           /* Finish keyboard */
```

```
#define DEC_NORWAY            9           /* Norwegian keyboard */
#define DEC_DENMARK          10           /* Danish keyboard */
#define DEC_CANADA           11           /* Canadian keyboard */
#define DEC_SWISS_GERMAN     12           /* Swiss German keyboard */
#define DEC_SWISS_FRENCH     13           /* Swiss French keyboard */


/* Non-standard virtual keys defined in MS-Windows version 1.01 */

#define VK_OEM_NUMBER    0x90          /* Numlock          */
#define VK_OEM_SCROLL    0x91          /* ScrollLock       */
#define VK_OEM_1         0xBA          /* ';' for US       */
#define VK_OEM_PLUS      0xBB          /* '+' any country  */
#define VK_OEM_COMMA     0xBC          /* ',' any country  */
#define VK_OEM_MINUS     0xBD          /* '-' any country  */
#define VK_OEM_PERIOD    0xBE          /* '.' any country  */
#define VK_OEM_2         0xBF          /* '/?' for US      */
#define VK_OEM_3         0xC0          /* '`~' for US      */
#define VK_OEM_4         0xDB          /* '[{' for US      */
#define VK_OEM_5         0xDC          /* '\|' for US      */
#define VK_OEM_6         0xDD          /* ']}' for US      */
#define VK_OEM_7         0xDE          /* ''"' for US      */
#define VK_OEM_8         0xDF          /* Not assigned     */

/* DIGITAL defined virtual keys */
#define VK_OEM_PF1       VK_ESCAPE
#define VK_OEM_PF2       0xE2
#define VK_OEM_PF3       VK_OEM_SCROLL
#define VK_OEM_PF4       VK_MULTIPLY

#define VK_OEM_F17       0xE3
#define VK_OEM_F18       0xE4
#define VK_OEM_F19       0xE5
#define VK_OEM_F20       0xE6

#define VK_OEM_COMPOSE    0x92


/*  The following definitions are for the Windows LAT interface. */

/*  Windows LAT interface functions */

extern int  FAR PASCAL OpenLat (LPSTR, LPSTR, LPSTR);
extern int  FAR PASCAL WriteLat (int, char);
extern int  FAR PASCAL GetLatStatus (int);
extern int  FAR PASCAL ReadLat (int);
extern int  FAR PASCAL CloseLat (int);
```

```
   extern int  FAR PASCAL SendLatBreak (int);
   extern long FAR PASCAL InquireLatServices ();
   extern int  FAR PASCAL GetLatService (LPSTR);

   /* LAT initialization error and service table error return codes */

   #define IE_LATINSTALL   -1      /* Lat driver not installed */
   #define IE_LATSERVICE   -2      /* Service not in table or name error */
   #define IE_LATVIRTCIRC  -3      /* No more virtual circuit blocks
                                      available */
   #define IE_LATSESSIONS  -4      /* No more sessions available */
   #define IE_LATMEMORY    -5      /* All SCBs allocated, no memory
                                      available */
   #define IE_LATBUFFER    -6      /* Data buffer specification error */
   #define IE_LATCIRCSESS  -7      /* No more sessions available on this
                                      circuit */
   #define IE_LATNOSERVNAME -8     /* No service name returned, have not
called the LAT service table reset
function or have reached the end of the
table */
   #define IE_LATOVERFLOW  -9      /* LAT service table overflow */

   /* LAT function return codes
      All error codes are negative 16 bit integers */

   #define CE_LATOK         0      /* General function success return value */
   #define CE_LATID        -1      /* Invalid LAT session ID */
   #define CE_LATTXQUE     -2      /* Unable to queue character for
                                      transmission */
   #define CE_LATNOCHAR    -3      /* No character available on a read
                                      request */
   #define CE_LATSTOP      -4      /* LAT circuit failed or stopped */

   #define CE_LATBRK       -5      /* Unable to send break to host */

   /* LAT status word bit definitions */

   #define ST_LATREC       0x0001  /* Receive data is available. */
   #define ST_LATTXQUE     0x0002  /* Unable to queue transmit data. */
   #define ST_LATSESINACT  0x0004  /* Lat session is not active. */
   #define ST_TXEMPTY      0x0020  /* Transmit buffer is empty. */


   /* The following constants define new character set definitions for
DIGITAL's MS-Windows printer drivers.

      The terms GL and GR are used below.  Character sets in GL fall in the
```

```
    range of 20h to 7Fh.  Character sets in GR fall in the range of A0h
    FFh.
  */

  #define UK_NRC              224  /* United Kingdom NRC in GL */
  #define FRENCH_NRC          225  /* French NRC in GL */
  #define GERMAN_NRC          226  /* German NRC in GL */
  #define ITALIAN_NRC         227  /* Italian NRC in GL */
  #define DANISH_NRC          228  /* Danish NRC in GL */
  #define NORWEGIAN_NRC       229  /* Norwegian NRC in GL */
  #define SPANISH_NRC         230  /* Spanish NRC in GL */
  #define SWEDISH_NRC         231  /* Swedish NRC in GL */
  #define JIS_ROMAN_NRC       232  /* Japanese (JIS Roman) NRC in GL */
  #define KATAKANA_NRC        233  /* Japanese Katakana Graphic Char Set i
*/
  #define FINNISH_NRC         234  /* Finnish NRC in GL */
  #define FR_CANADIAN_NRC     235  /* French Canadian NRC in GL */
  #define DUTCH_NRC           236  /* Dutch NRC in GL */
  #define SWISS_NRC           237  /* Swiss NRC in GL */
  #define PORTUGUESE_NRC      238  /* Portuguese NRC in GL */
  #define DEC_MCS             240  /* ASCII in GL, DIGITAL Multinational i
*/
  #define DEC_SPC_GRAPHICS    241  /* DIGITAL Special (VT100) Graphics in
*/
  #define DEC_TECHNICAL       242  /* DIGITAL Technical Character Set */
  #define DEC_PUBLISHING      243  /* DIGITAL Publishing Character Set */
```

Chapter 18
VAXmate Network Software

Introduction

The VAXmate network software allows VAXmate workstations to be nodes in a DIGITAL local area network. The software provides users and applications with a Microsoft MS-Network compatible environment and a DIGITAL DECnet compatible environment.

This chapter describes the VAXmate-specific programmer interfaces into the network environment. It also discusses the relationship among the various components that implement the network interfaces. Some of the described interfaces are not recommended for use by application software. Their description is provided only for completeness.

The discussion of the network software is restricted to the VAXmate client node. VAXmate client nodes are those VAXmate workstations that sit on a user's desk and utilize the services of VAX/VMS servers and VAXmate servers.

This chapter assumes the reader is familiar with the Microsoft MS-Network V1.0 implementation and has documentation on that implementation. To learn more about Ethernet and the DIGITAL Network Architecture, refer to the documentation list at the end of this introduction.

Figure 18-1 shows the basic components that comprise the VAXmate client network system.

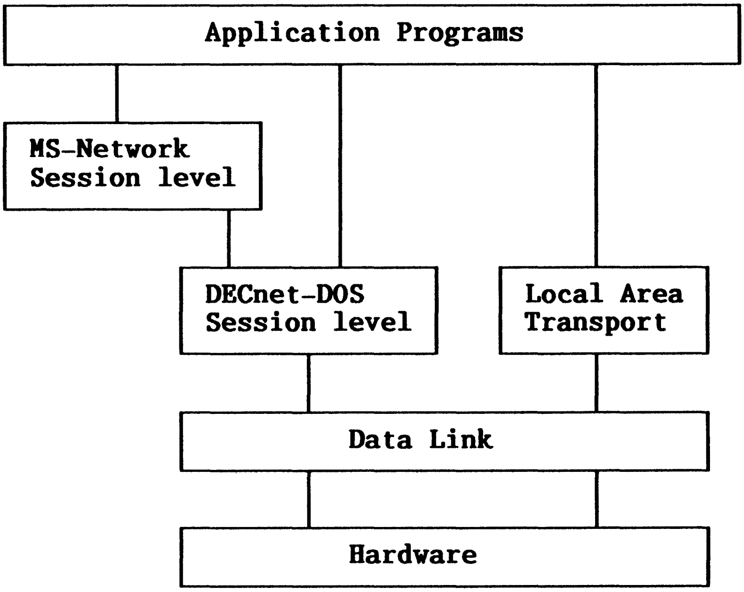

Figure 18-1 VAXmate Network Components

Each item in Figure 18-1 is discussed in the following section.

Hardware

The center of the VAXmate ThinWire Ethernet network hardware is the DIGITAL Local Area Network Controller integrated circuit; referred to as the LANCE chip. This chip and its support circuits connect the VAXmate workstation to the ThinWire Ethernet local area network. The LANCE chip internal registers are described in this manual.

VAXmate workstations contain different versions of network hardware, which behave differently on large networks that have high levels of traffic. The network hardware that is different requires special treatment by software. Digital Equipment Corporation recommends that you not attempt to directly program the hardware interface. It is recommended that applications use the MS-Network session level interface, DECnet-DOS session level interface, and the Data Link interface to access the network.

For more information about the VAXmate network hardware, see chapter 13.

Datalink

The Datalink is implemented in VAXmate ROM code. The datalink interface, which is used by all upper levels of the VAXmate network, provides a hardware independent interface for accessing the network. The Datalink is the lowest supported interface through which applications access the ThinWire network and it is capable of managing multiple network streams. Applications can access the datalink concurrently with the DECnet-DOS and MS-Network sessions.

The VAXmate ROM code also contains a subset of the DECnet Maintenance Operation Protocol (MOP). This protocol allows the booting of a VAXmate workstation from a host machine located on the network and provides loopback functions to support network trouble shooting.

Local Area Transport

The Local Area Transport (LAT) lets the VT220 and VT240 terminal emulators and MS-Windows serial communication applications to access VAX/VMS systems using the ThinWire Ethernet network. The LAT module implements the DIGITAL proprietary LAT protocol for Ethernet networks. With this protocol, applications that normally used serial communication lines into VAX/VMS hosts can be written to use the ThinWire Ethernet.

For many VAX/VMS communications applications the LAT interface eliminates the need for a serial line or a modem between the VAXmate workstation and its host.

DECnet-DOS Session Level

DECnet-DOS is the center of the RAM resident network software. Application programs can directly access all DECnet-DOS services. In particular, transparent task-to-task communications can be implemented between applications running on different network nodes.

For a full description of DECnet-DOS, its application visible interfaces, and its supporting utilities, refer to *DECnet-DOS Programmer Reference Manual* and *DECnet-DOS User Guide.*

MS-Network Session

The MS-Network Session is a DIGITAL-developed emulation of the Microsoft MS-Network session level interface. The MS-Network session level interface is lets application programs created to run with Microsoft MS-Network execute.

The emulation uses the DECnet-DOS programming interface to access the network. The MS-Network protocol is treated as an application level protocol in the DECnet environment.

The MS-DOS REDIRECTOR interfaces to the MS-Network Session Level emulator for network file access and network printing.

Documentation List

For further reading on DECnet-DOS refer to:

- *DECnet-DOS Programmers Reference Manual*
- *DECnet-DOS Users Guide*

For further information on Microsoft MS-Network software or the Session Level interface refer to:

- *Server/Redirector File Sharing Protocol* (Microsoft Corporation), which describes in detail the Server Message Block protocol that is used by the Redirector for accessing remote file and print services.
- *Microsoft Network Session Layer Interface* (Microsoft Corporation), which describes the interface between the session and higher layers of the Microsoft Network.
- MS-Network Version 1.0 documentation

For Further information on the DIGITAL Network Architecture refer to the following Digital DECnet publications:

- *DNA General Description*
- *DNA Session Control Functional Specification*
- *DNA Routing Functional Specification*
- *DNA Maintenance Operations Functional Specification*
- *DNA Network Management Functional Specification*
- *DNA Data Access Protocol (DAP) Functional Specification*
- *The Ethernet, A Local Area Network, Datalink Layer and Physical Layer Specification*

Datalink

The datalink layer is that portion of the DNA architecture that lies between the routing layer and the physical network hardware. The purpose of the datalink is to provide a hardware independent set of services for use by higher levels of network software and special application programs. Software that directly accesses the datalink interface is called a client of the datalink. In the VAXmate DECnet implementation, the routing layer is a client of the datalink. The Local Area Transport software also is a client of the datalink. Multiple clients can access the datalink services simultaneously.

The VAXmate workstation implementation of the datalink has two modules:

- The datalink module, which provides the client interfaces described in this chapter. This datalink layer is independent of the underlying Ethernet hardware used to implement the actual physical network.
- The port driver module, which provides an unsupported interface to the underlying hardware. The port driver is specific to the Ethernet hardware implementation.

The VAXmate datalink, in combination with the port driver, provides a complete program interface for accessing the ThinWire Ethernet.

The datalink is independant of the operating system service. No operating system services are accessed from within the datalink. Most of the VAXmate datalink code is resident in the same system ROM as the ROM BIOS and the self-test diagnostic code. A small portion of the VAXmate datalink is implemented by the DLL.EXE program, which is a terminate and stay resident module that runs as part of the VAXmate workstation network startup procedure. It initializes the datalink and allocates RAM resident variable and buffer storage.

An application interacts with the datalink through a portal. A portal is comprised of the state variables and data structures shared by a specific application and the datalink for the reception or transmission of information over the network. Applications open and close portals as a part of their interaction with the datalink.

The datalink layer is responsible for multiplexing messages received from the ThinWire Ethernet to the correct program client of the datalink. The multiplexing is done first by address and then by protocol type. The protocol type field is the last 2 bytes of the Ethernet datalink header in non-802.3 compatible mode. IEEE 802.3 compatible mode is treated as a protocol type.

The type of multiplexing is determined on a per portal basis and is specified when the portal is opened.

If a portal is opened in IEEE 802.3 compatible mode, the multiplexing is done based on the address. Only one 802.3 compatible portal can be enabled at a time. The length field must be less than or equal to the maximum Ethernet length of 1518(decimal).

Future versions of the VAXmate datalink may implement extensions to the 802.3 mode described in this chapter. Applications that use the 802.3 mode described here may not work in future network environments.

All non-802.3 protocols are identified by protocol type values that are larger than the maximum Ethernet packet length of 1518(decimal).

For the VAXmate workstation in the DIGITAL Ethernet environment, the multiplexing is done on both address and protocol type. The protocol type value must be larger than the maximum Ethernet packet length of 1518(decimal).

In either case, the address is the station ID for the VAXmate workstation, or the multicast address of one of the portals enabled for reception of multicast messages.

Common Definition Formats

Throughout this chapter, C language constructs are used to describe the data structures associated with accessing the services of the datalink.

The following terminology applies to the structures:

- int is an unsigned 16 bit integer value.
- uchar is an unsigned 8 bit value.
- farptr is a double word pointer.

Datalink Communication Block

Client software must define a Datalink Communication Block (DCB) as part of the datalink interface definition.

The following C programming language structure describes the DCB. For many datalink accesses, actual use of fields described here may not conform with expected meaning of the field names.

```
struct dcb
{
  int portal_id;                 /* The portal ID for this request */
  uchar source_address[6];       /* The source address */
  uchar destination_address[6]; /* The destination address */
  farptr *bh;                    /* double word pointer to the buffer header */
  int bl;                        /* Buffer length */
  int operation;                 /* Used by each function differently */
  uchar pad;                     /* Pad flag used on open */
  uchar mode;                    /* Mode flag used on open */
  farptr *line_state();          /* pointer to line state change routine */
  farptr *rcv_callback();        /* pointer to received data routine */
  farptr *xmit_callback();       /* pointer to transmitted data routine */
  char max_outstanding;          /* Number of outstanding transmits/receives */
  uchar ptype[2];                /* Protocol type */
  int buffers_lost;              /* Number of buffers lost */
};
```

Multicast Address Format

Figure 18-2 describes the format of an Ethernet multicast address. A multicast address is six bytes long. The Xs in the diagram represent address-specific bits.

```
      Byte 0          Byte 1    Byte 2    Byte 3    Byte 4    Byte 5
.---------------.--------.--------.--------.--------.--------.
|7|6|5|4|3|2|1|0|76543210|76543210|76543210|76543210|76543210|
|---------------|--------|--------|--------|--------|--------|
|x|x|x|x|x|x|x|1|xxxxxxxx|xxxxxxxx|xxxxxxxx|xxxxxxxx|xxxxxxxx|
|_______________|________|________|________|________|________|
```

Figure 18-2 Multicast Address Format

The special case Broadcast Address is specified by a multicast address, where all address bits equal 1.

Byte 0, Bit 0 distinguishes a multicast address from a physical address.

```
Byte 0 <0>   = 1    This is a multicast address.
             = 0    This is a physical address.
```

For example, 08 is a physical address, and 09 is a multicast address.

Software Capabilities

The datalink layer is that portion of the ISO/DNA architecture that is responsible for the multiplexing of messages from the Ethernet port driver to clients of the datalink. This multiplexing is done on the destination address and protocol type fields of the Ethernet message.

User Call-Back Routines

The datalink invokes call-back routines that are specified in the DCB.

Call-back routines are required application-specific subroutines that are dispatched to by the datalink. The routine is invoked by a FAR CALL. Within the call-back routine, the client should not attempt to use any MS-DOS facilities and the client should not enable or disable interrupts. The call-back routine must end with a FAR RETURN. The datalink ensures that a call-back routine is not called again until the previous call is complete with a FAR RETURN.

The routines are called by the datalink when an event occurs that indicates a change in the Ethernet hardware. The event can be either an error, a receive buffer filled, a transmit buffer sent, or a line state change. The datalink call-back routines are described in the next sections.

Each call-back routine involves a data structure called the the User Call Back block (UCB). The address of the UCB is passed to the client in the ES:BX register.

The following C structure describes the UCB.

```
structure ucb
{
 int    portal_id;      /* Portal ID from the request */
 uchar  destination[6];    /* Destination address from the buffer header */
 uchar  source[6];         /* Source address from the buffer header */
 farptr *buffer;        /* Pointer to the client buffer being returned */
 int    bl;             /* NULL or byte length of received message */
 uchar  buffer_status;  /* COMPLETE or  ERROR */
 uchar  buffer_reason;  /* Completion error reason or LINE_STATE_CHANGE */
};
```

The UCB fields are further described below:

- **ucb.portal_id** is the portal ID of the request that led to this call-back.
- **ucb.destination** corresponds to the first 6 bytes of the Ethernet frame and to the destination field of the Ethernet packet.
- **ucb.source** is the second 6 bytes of the Ethernet frame and corresponds to the Ethernet address of the originating node.
- **ucb.bl** is the count of client data bytes actually in the buffer on a receive request completion. Field **ucb.bl** has no meaning on a transmitted message completion.
- **ucb.buffer** is the address of the data portion of the transmit or receive buffer that is being returned to the client.
- **ucb.buffer_status** indicates the status of the buffer and can be one of the following values:
  - COMPLETE - Value = 1 - The request was completed.
  - ERROR - Value = 3 - The request encountered an error and could not be completed.
- **ucb.buffer_reason** is a further explanation of the buffer status field and attempts to tell the upper layers why the operation was aborted.

  For a transmit complete call-back routine with an ERROR status, ucb.buffer_reason is a byte of bit flags with the following meaning. Bit set means true.

```
 7   6   5    4      3      2      1      0
.---.---.---.-----.------.-----.-----.-------.
| Reserved  |LCAR | LCOL | EXC | BUF | UNDER |
|___|___|___|_____|______|_____|_____|_______|
```

Bit 7-5:	Reserved for future use.
Bit 4:	The LCAR flag means the hardware lost the carrier signal during the transmission.
Bit 3:	The LCOL flag means the hardware detected a collision which was after the slot time during the transmission.
Bit 2:	The EXC flag means that too many collisions occurred during the transmission. The hardware could not transmit the packet and aborted the transmission.
Bit 1:	The BUF flag means the hardware could not access the bus during the transmission.
Bit 0:	The UNDER flag means the hardware experienced an underflow error during the transmission of the packet.

For a line state change call-back, field **ucb.buffer_reason** means:

LINE_STATE_CHANGED - Value = 3 - The line state is changed from ON to some other state and therefore all connections are aborted.

The receive call-back routine UCB does not return ERROR status. All receive errors are handled within the Ethernet hardware and datalink layer.

Line State Change Call-Back Routine

The line-state-change call back routine specified in the DCB is used to notify the client of a state change on the Ethernet channel. Notification consists of executing the call-back routine pointed at by the line_state field of the DCB structure. If this field contains a zero value, the datalink does not notify the client application of a line state change.

When the datalink calls the line-state-change call-back routine, the ES:BX register pair points at the UCB. See the section "User Call Back Routines" for more information on line-state-change call-backs.

Within the line-state-change call back routine, the client should not use any MS-DOS facilities or enable or disable interrupts. The call-back routine must end with a FAR RETURN. The datalink ensures that a call-back routine is not called again until the previous call is completed with a FAR RETURN.

Receive

There is not an explicit receive function. Opening a datalink portal automatically sets the interface up for receiving information. The receive call-back routine is the mechanism by which the client is notified of a message received on its behalf. Notification consists of executing the call-back routine pointed at by the rcv_callback field of the DCB structure.

When the receive call-back routine is called by the datalink, the ES:BX register pair points at the UCB. See the section "User call-back Routines" for more information on receive call-backs.

Within the receive call-back routine, the client should not attempt to use any MS-DOS facilities or enable or disable interrupts. The receive call-back routine must end with a FAR RETURN. The datalink ensures that a call-back routine is not called again until the previous call is completed with a FAR RETURN.

The client should deallocate its receive buffers before exiting.

NOTE
If datalink clients do not return (dll_deallocate) all datalink transmit and receive buffers before exiting, the datalink runs out of buffers and does not function properly. This leads to serious network problems.

Transmit

The transmit function DLL_TRANSMIT is discussed in the section "Datalink Functions" in this chapter.

Datalink Functions

Client programs request services from the datalink through Interrupt 6DH. After initialization, each time the datalink is accessed, the client program must pass the address of the DCB in the form of a far pointer in registers ES:BX. The client program must specify the function code in the AH register. Upon return from a datalink request, the AX register contains the status of the request.

Table 18-1 describes the function codes passed in register AH to call the datalink routines.

Table 18-1 Interrupt 6D: Datalink Functions

Function	Value in Register AH	Description
DLL_INIT	00H	Initialize datalink and port driver
DLL_OPEN	01H	Open a datalink portal
DLL_CLOSE	02H	Close a datalink portal
DLL_ENABLE_MUL	03H	Enable a multicast address
DLL_DISABLE_MUL	04H	Disable a multicast address
DLL_TRANSMIT	05H	Transmit an Ethernet message
DLL_REQUEST_XMIT	06H	Request a transmit buffer for an Ethernet message
DLL_DEALLOCATE	07H	Deallocate buffer
DLL_READ_CHAN	08H	Read the channel status
DLL_READ_PLIST	09H	Read the datalink portal list
DLL_READ_PORTAL	0AH	Read information about a datalink portal
DLL_READ_COUNT	0BH	Read and/or clear counters
DLL_NETWORK_BOOT	0CH	Request to boot from a network server
DLL_ENABLE_CHAN	0DH	Enable the Ethernet channel
DLL_DISABLE_CHAN	0EH	Disable the Ethernet channel
DLL_START_MOP	0FH	Start MOP/Send a System ID message
DLL_STOP_MOP	10H	Stop MOP (documented in MOP section)
DLL_READECPARM	11H	Read the address of the DECPARM string
DLL_SETDECPARM	12H	Set the address of the DECPARM string
DLL_EXT_LOOPBACK	13H	Loopback a message through the Ethernet hardware

Datalink Return Codes

The datalink always returns with status from a client request for a service. The status is returned in the AX register. The return value overwrites the function code that was in AH. Table 18-2 describes the return code values returned to the client software.

Table 18-2 Datalink Return Codes

Symbol Name	Value in Register AX	Description
SUCCESS	00H	Successful completion.
INITIALIZATION_FAILURE	01H	The hardware failed to initialize.
CHANNEL_NOT_OFF	02H	The channel state was not off. Set it off to execute this function.
STATE_OFF	03H	The channel state is off. Set it on to execute this function.
ADDRESS_NOT_SET	04H	The address of the Ethernet hardware is not set. Set it before starting the port.
NO_HARDWARE	05H	Reading from the Ethernet hardware meaning that there is none available, indicating a hardware malfunction if there is hardware.
BUFFER_TOO_SMALL	06H	On read counters, the buffer that holds the counters is too small to fit all the counter values.
NONE_AVAILABLE	07H	There are no more buffers available.
NO_RESOURCES	08H	There are no more resources available for this request.
PROMISCUOUS_ RECEIVER_ACTIVE	09H	A promiscuous receiver is currently active, therefore the requested function is illegal.

Table 18-2 Datalink Return Codes (cont.)

Symbol Name	Value in Register AX	Description
NON_EXCLUSIVE	0AH	Client attempted to enable promiscuous mode, but a promiscuous mode receiver is already active, or another portal has enabled a protocol or multicast. Client attempted to enable 802.3 compatible mode, but another portal is currently in 802.3 compatible mode.
UNRECOGNIZED_PORTAL	0BH	The portal ID returned was not recognized.
PROTOCOL_TYPE_IN_USE	0CH	The protocol type the client attempted to enable is in use for another datalink portal.
NOT_MULTICAST	0DH	The multicast address supplied to enable_multicast is not a valid multicast address.
OUTSTANDING_CALLS	0EH	Outstanding transmit or receive calls are pending, therefore the client may not close the datalink portal.
NO_RECEIVE_BAD	0FH	The client specified reception of bad frames to the datalink layer and the hardware does not support the function.
NONE_OUTSTANDING	10H	There are no receive buffers to be aborted.
NO_EVENTS	11H	There are no events in the event queue to be read.
STATE_BROKEN	12H	The port driver could not initialize the port hardware.

Table 18-2 Datalink Return Codes (cont.)

Symbol Name	Value in Register AX	Description
QUOTA_EXCEEDED	13H	The user buffer quota is exceeded. No more buffers can be allocated to this portal.
ALREADY_INITED	14H	The datalink layer is already initialized. This is an INIT failure error, but the datalink is still usable if the state is ON.
LB_FAILURE	15H	The hardware failed to loopback data, indicating an Ethernet hardware failure.

Function 00H: Initialization (dll_init)

Datalink intialization is automatically done when the DLL.EXE module is loaded as a part of the network start up procedure. Under most circumstances, an application should not attempt to initialize the datalink. This service is documented here for completeness. You should excercise caution if you decide to replace DLL.EXE and initialize the datalink yourself. DIGITAL does not support the datalink interface if it is initialized by the application. Future versions of the VAXmate workstation and the network software may require different initialization parameters than those described here.

The datalink initialization function is invoked once at startup time to initialize the datalink and port driver modules. Parameters to the initialization call are the port driver and datalink buffer pool blocks, the maximum number of portals allowed, the maximum number of multicasts allowed, and the maximum number of receive buffers allowed to be queued for a single portal.

Parameters

AH =	00H
ES:BX =	Port driver buffer pool
CH =	Number of Port driver buffers
CL =	Number of transmit and receive rings as power of 2
DI =	Maximum number of open portals
DH =	Maximum number of multicasts allowed per portal

Returns

AX =	0000H — SUCCESS
	0001H — INITIALIZATION_FAILURE
	0014H — ALREADY_INITED

Use the following formulas to calculate the amount of buffer space required for the port driver and datalink.

```
BUFFER_pool_size = pd_size + dll_size
```

Where:

```
dll_size = (MAX_DLL_PORTALS*(SIZE PLIST)) +
           (MAX_DLL_PORTALS*(SIZE DLL_PDB)) +
           (MAX_DLL_PORTALS*MAX_MULTICAST*SIZE OF MULTICAST)

           MAX_DLL_PORTALS    = DI  Recommended value is 8
           SIZE PLIST         = 5   Size of internal data structure
           SIZE DLL_PDB       = 24  Size of internal data structure
           MAX_MULTICAST      = DH  Recommended value is 8
           SIZE OF MULTICAST  = 6   Ethernet constant
```

```
pd_size = ((MAX_ETHER_PACK + (SIZE BHEAD))*TOTAL_BUFFERS)+
          (2^NUM_BUFFER_RINGS/2 * SIZE RECEIVE) +
          (2^NUM_BUFFER_RINGS/2 * SIZE TRANSMIT) + JUSTIFY

          MAX_ETHER_PACK   = 1518  1514 byte of data + 4 bytes CRC
          SIZE BHEAD       = 9     Size of internal data structure
          NUM_BUFFER_RINGS = CL    Recommended value is 5
          SIZE RECEIVE     = 8     Size of hardware receive buffer ring
          SIZE TRANSMIT    = 8     Size of hardware transmit buffer ring
          JUSTIFY          = 8     Number of bytes for LANCE buffer alignment
```

Table 18-3 describes the recommended values for the datalink parameters.

Table 18-3 Recommended Values for Datalink Parameters

Symbol Name	Value	Meaning
MAX_PROTOCOLS	8	The maximum number of protocol types that may be enabled at one time
MAX_MULTICAST	8	The maximum number of multicast addresses that may be enabled at one time
MIN_Ethernet_PACKET	64	The minimum Ethernet packet size including the datalink header and checksum (60 data+4 CRC bytes)
MAX_Ethernet_PACKET	1518	The maximum Ethernet packet size including datalink header and checksum (1514 data+4 bytes CRC)
MAX_PORTALS	8	The maximum number of datalink portals allowed to be opened at any time

Function 01H: Open a Datalink Portal (dll_open)

The open function opens a portal so the client can transmit and receive frames from the Ethernet port driver. This routine, which is called with a DCB, expects the 'pad' flag to be set to PAD if padding is to be used and NO_PAD if it is not.

If the mode flag is PROMISCUOUS_MODE, no other portals are allowed to open.

Parameters

AH =	01H	
ES:BX =	far pointer to DCB	
	dcb.pad =	PAD or NO_PAD.
	dcb.mode =	802_COMPATABLE or Ethernet or PROMISCUOUS_MODE.
	dcb.ptype =	Protocol type to enable if not promiscuous.
	dcb.line_state =	Address of line state change routine or 0.
	dcb.rcv_callback =	Address of receive call-back routine .
	dcb.xmit_callback =	Address of transmit call-back routine.
	dcb.max_outstanding =	Maximum number of outstanding buffers.

Returns

AX =	0000H – SUCCESS with dcb.portal_id as the portal ID
	0003H – STATE_OFF
	0008H – NO_RESOURCES
	0009H – PROMISCUOUS_RECEIVER_ACTIVE
	000CH – PROTOCOL_TYPE_IN_USE
	000AH – NON_EXCLUSIVE

The DCB fields and return values are described in the following section:

- **dcb.pad** indicates whether frames are to be padded out to minimum Ethernet frame size or not.

NOPAD	value = 0	Do not pad messages out to minimum Ethernet length.
PAD	value = 1	Pad the Ethernet messages on this portal out to minimum length if they are less then the Ethernet minimum size. Padding must be agreed upon by both sides or else the remote side does not know to de-pad the packet.

- **dcb.mode** indicates the mode in which the portal is to be opened. Certain events and routines act differently in IEEE 802.3 compatible mode than they do in Ethernet compatible mode. The major difference is that in IEEE 802.3 compatible mode, the protocol-type field is set to the actual amount of user data in the frame. The header and CRC bytes are not included in this byte count. Protocol-type multiplexing cannot occur because there is no protocol-type field.

  - 802_COMPATABLE - value = 0 - Open a datalink portal in 802.3 compatible mode.
  - Ethernet - value = 1 - Open a datalink portal in Ethernet compatible mode.
  - PROMISCUOUS_MODE - value 2 - Open a datalink portal in promiscuous mode.

- **dcb.ptype** indicates the protocol type to enable on this portal.
- **dcb.linestate** indicates the address of the line-state-change routine. If the client does not want to be notified of line-state changes, the client should specify a 0 for this double word routine pointer.
- **dcb.rcv_callback** indicates the address of the routine the datalink calls when a buffer comes in for the client. A valid address must be specified.
- **dcb.xmit_callback** indicates the address of the routine the datalink calls when a buffer that this portal has queued for transmission either aborts due to an error, or is successfully transmitted.
- **dcb.max_outstanding** indicates the portals quota for the number of buffers, including transmit and receive buffers, that can be outstanding at any one time. If the client specifies a quota of zero, a default quota of one receive and one transmit buffer is given. If the client specifies a quota larger than the number of buffers allocated on the dll_init call and no other datalink clients exist, the number of buffers used in dll_init are allocated to the caller. This means the client can effectively use all of the datalink buffer space. If other datalink clients already exist and the caller asks for more buffers than are available, the caller is given the buffers that remain.

- **SUCCESS** indicates that a portal is open and therefore dcb.portal_id is set to a valid portal ID.
- **NO_RESOURCES** indicates that there were not enough resources to open another portal.
- **STATE_OFF** indicates that the channel is currently off or broken, and therefore no opens are allowed.
- **PROMISCUOUS_RECEIVER_ACTIVE** indicates that there is a promiscuous receiver active and therefore no other portals may be opened at this time.
- **PROTOCOL_TYPE_IN_USE** indicates that the protocol type you tried to enable was already in use by another portal.
- **NON_EXCLUSIVE** indicates on of the following:
  - You tried to enable promiscuous mode with a promiscuous receiver active.
  - You tried to enable another 802.3 compatible portal when there was already an 802.3 mode compatible portal enabled.

Function 02H: Close a Datalink Portal (dll_close)

The close function closes an open Ethernet portal and releases all of its resources. A portal cannot be closed unless all outstanding transmit and receive requests are completed.

Parameters

AH =	02H	
ES:BX =	far pointer to DCB	
	dcb.portal_id =	Portal ID from open

Returns

AX =	0000H – SUCCESS 000BH – UNRECOGNIZED_PORTAL 000EH – OUTSTANDING_CALLS

The DCB field and return values are described below:

- **dcb.portal_id** is the portal ID from the open call that you want to close.
- **SUCCESS** indicates that the portal is successfully closed. Any references to this portal return the UNRECOGNIZED_PORTAL error.
- **UNRECOGNIZED_PORTAL** indicates that the portal ID is not a valid open datalink portal ID.
- **OUTSTANDING_CALLS** indicates that the portal could not be closed because there are incomplete transmit or receive requests on this portal.

Function 03H: Enable Multicast Addresses (dll_enable_mul)

The enable multicast request is made to enable reception of frames addressed to a specific group address (a multicast address) for this portal.

Parameters

AH =	03H	
ES:BX =	far pointer to DCB	
	dcb.portal_id =	The portal ID from the open call
	dcb.source_address =	multicast address to be enabled

Returns

AX =	0000H – SUCCESS 000DH – NOT_MULTICAST 0008H – NO_RESOURCES 000BH – UNRECOGNIZED_PORTAL 0009H – PROMISCUOUS_RECEIVER_ACTIVE 0003H – STATE_OFF

The DCB fields and the return values are described below:

- **dcb.portal_id** is a valid datalink portal ID on which this multicast address is to be multiplexed.
- **dcb.source_address** is the multicast address to start receiving messages for.

NOTE
If you specify the broadcast address, FF-FF-FF-FF-FF-FFH, no error is returned. This address is not displayed as part of the Read Portal Status function. However, the Broadcast Address is always enabled for this portal.

- **SUCCESS** indicates that the multicast address is enabled for this portal.
- **NOT_MULTICAST** indicates that the address specified in the source address was not a valid multicast address.
- **NO_RESOURCES** indicates that there are currently not enough resources to filter another multicast address for this portal.
- **UNRECOGNIZED_PORTAL** indicates that the portal ID specified in the DCB was not a valid open datalink portal.

- **PROMISCUOUS_RECEIVER_ACTIVE** indicates that there is currently a promiscuous receiver, and, therefore, no protocol types or multicast addresses are enabled.
- **STATE_OFF** indicates that the channel state was set to BROKEN or OFF. The channel state must be ON to filter multicast addresses.

Function 04H: Disable Multicast Addresses (dll_disable_mul)

The disable multicast function request indicates that a client no longer wants to receive frames for the specified multicast address through this portal.

Parameters

AH =	04H	
ES:BX =	far pointer to DCB	
	dcb.portal_id =	Portal ID from open call
	dcb.source_address =	Multicast address to be disabled

Returns

AX =	0000H – SUCCESS 000BH – UNRECOGNIZED_PORTAL

The DCB fields and the return values are described below:

- **dcb.portal_id** is a valid portal ID returned by the open function on which the multicast address is to be disabled.
- **dcb.source_address** is the multicast address to be disabled.
- **SUCCESS** indicates that the multicast address is no longer multiplexed for this portal.
- **UNRECOGNIZED_PORTAL** indicates that the portal id did not correspond to a valid open datalink portal.

Function 05H: Transmit (dll_transmit)

The transmit function queues a frame for transmission. The client is notified at the call back address when the transmission is completed. Notification consists of executing the call-back routine pointed at by the xmit_callback field of the DCB structure. When the transmit call-back routine is called by the datalink, the ES:BX register pair points at the UCB block. See the section "User call-back Routines" for more information on transmit call-back routines.

Within the transmit call-back routine, the client should not attempt to use any MS-DOS facilities or enable or disable interrupts. The transmit call-back routine must end with a FAR RETURN. The datalink ensures that a call-back routine is not called again until the previous call has completed with a FAR RETURN.

No abort transmit function is provided because transmission always succeeds or fails within a short period of time.

> **NOTE**
> If datalink clients do not return (dll_deallocate) all datalink transmit and receive buffers before exiting, the datalink runs out of buffers and does not function properly. This leads to serious network problems.

A client application can supply its own buffer for transmission or request a buffer from the datalink using the Request Transmit Buffer function. Client supplied buffers do not have to be deallocated to the datalink before exiting.

Parameters

AH =	05H	
ES:BX =	far pointer to DCB	
	dcb.portal_id =	The portal ID from the open call
	dcb.destination_address =	Address of remote node
	dcb.ptype =	Protocol type or NULL
	dcb.bh =	Pointer to the buffer to be sent
	dcb.bl =	The buffer length excluding the datalink header

Returns

AX =	0000H – SUCCESS 0008H – NO_RESOURCES 000BH – UNRECOGNIZED_PORTAL 0003H – STATE_OFF
ES:BX =	Far pointer to User call-back Block. See the section "User call-back Routines" for further information.

The DCB fields and the return values are described below:

- **dcb.portal_id** is the portal ID returned by the open call.
- **dcb.destination_address** is the 6 byte address to which this transmission is to be sent. This address can be a physical address, a multicast address, or the broadcast address.
- **dcb.ptype** is the protocol type field in Ethernet compatible mode. It is ignored in the IEEE 802.3 compatible mode. In IEEE 802.3 mode the type field is used for the frame length.
- **dcb.bh** is the pointer to a buffer to be transmitted.
- **dcb.bl** is the length of the client data in the buffer. The length does not include the datalink header length.
- **SUCCESS** indicates that the buffer is successfully queued for transmission. The client is notified at the transmit complete call back routine supplied as part of the calling sequence.
- **NO_RESOURCES** indicates that there are currently not enough resources to queue the buffer for transmission.
- **UNRECOGNIZED_PORTAL** indicates that the portal id supplied is not a valid open datalink portal.
- **STATE_OFF** indicates that the Ethernet channel on which this portal has been opened is currently not turned on.

Function 06H: Request Transmit Buffer Function (dll_request_xmit)

The request transmit buffer function allocates a datalink transmit buffer. Once a buffer is allocated, the client owns the buffer until it returns the buffer via the deallocate buffer function. The client should return all allocated buffers to the datalink before exiting.

> **NOTE**
> If datalink clients do not return (dll_deallocate) all datalink transmit and receive buffers before exiting, the datalink runs out of buffers and does not function properly. This leads to serious network problems.

A client application can supply its own buffer for transmission. The client does not need to request a transmit buffer from the datalink layer. Client supplied buffers do not have to be deallocated to the datalink before exiting.

Parameters

AH =	06H	
ES:BX =	far pointer to DCB	
	dcb.portal_id =	The portal ID from the open call

Returns

AX =	0000H – SUCCESS 0008H – NO_RESOURCES 000BH – UNRECOGNIZED_PORTAL 0003H – STATE_OFF
dcb.bh =	The address of the buffer into which a client can place a message

The DCB fields and the return values are described below:

- **dcb.portal_id** is the portal ID to which the buffer is to be allocated.
- **dcb.bh** is the address of the buffer in which the data is to be placed when transmitting a frame.
- **SUCCESS** indicates that a buffer address is returned to the user.
- **NO_RESOURCES** indicates that there are currently not enough resources to process this request.
- **UNRECOGNIZED_PORTAL** indicates that the portal ID provided is not a valid datalink portal ID.
- **STATE_OFF** indicates that the channel is currently not turned on.

Function 07H: Deallocate Buffer (dll_deallocate)

The deallocate buffer request is made to return a data buffer to the datalink layer. The buffers returned are transmit buffers and receive buffers.

> **NOTE**
> If datalink clients do not return (dll_deallocate) all buffers before exiting, the datalink runs out of buffers and does not function properly. This leads to serious network problems.

Parameters

AH =	07H	
ES:BX =	far pointer to DCB	
	dcb.portal_id =	Portal ID returned by open
	dcb.bh =	Address of buffer to free up

Returns

AX =	0000H – SUCCESS
	000BH – UNRECOGNIZED_PORTAL

The DCB fields and the return values are described below:

- **dcb.portal_id** is the portal ID on which receive and transmit buffers are to be returned.
- **dcb.bh** is the address of the data buffer the client wants to return to the datalink buffer pool. The buffer should be one that was previously given to the client as part of a receive call-back sequence or the request-transmit buffer (dll_request_xmit) call.
- **SUCCESS** indicates that the function completed successfully and all buffers are returned.
- **UNRECOGNIZED_PORTAL** indicates that the portal ID supplied was not a valid open datalink portal.

Function 08H: Read Channel Status (dll_read_chan)

The read channel function is the network management interface to read information about a specified channel from the port driver. A channel corresponds to a physical Ethernet port. This is equivalent to a hardware controller. The VAXmate workstation has only one hardware Ethernet port.

Parameters

AH =	08H
ES:BX =	far pointer to DCB

Returns

AX =	0000H – SUCCESS with:	
	dcb.source_address =	Physical address
	dcb.destination_address =	Hardware address
	dcb.operation =	STATE(ON,OFF,INIT,BROKEN)
	dcb.mode =	Reserved
	dcb.pad =	Hardware interrupt vector number
	dcb.max_outstanding =	MOP Status

The DCB fields and the return values are described below:

- SUCCESS indicates that the function completed successfully, and the following locations in the DCB have been set.
  - **dcb.source_address** is set to the current physical channel address which is the address the channel is currently using.
  - **dcb.destination_address** is set to the hardware address associated with the channel. This may or may not be the same as the physical address.
  - **dcb.operation** is set to the current state of the datalink and can be one of the following:

State	Value	Meaning
OFF	0	The datalink state is now off.
ON	1	The datalink state is now on.
INIT	2	The port hardware is initializing.
BROKEN	3	The datalink failed to initialize.

- dcb.mode is reserved.
- dcb.pad is the interrupt vector which the Ethernet hardware is using to get its interrupts.
- dcb.max_outstanding is a mask which indicates the current status of the Maintenance Operations Protocol module, MOP. It has the following form:

```
 MOP STATUS MASK

 7   6   5   4   3   2      1         0

.---.---.---.---.---.---.---------.----------.
| Reserved                | CONSOLE | LOOPBACK |
|___|___|___|___|___|___|_________|__________|
```

- LOOPBACK indicates that the LOOPBACK server is enabled when a ONE is in this field.
- CONSOLE indicates that the remote console is enabled when a one in this bit.

Function 09H: Read the Portal List (dll_read_plist)

This is the network management function used to obtain a list of open portal IDs. It returns them in the buffer specified as its argument. If there is not enough space in the buffer, a partial list is returned, along with an error.

Parameters

AH =	09H	
ES:BX =	far pointer to DCB	
	dcb.bh =	Buffer to receive portal ID list
	dcb.bl =	Buffer length in 16 bit words

Returns

AX =	0000H — SUCCESS 0006H — BUFFER_TOO_SMALL
dcb.bl =	Number of portals returned on SUCCESS
dcb.operation =	Current datalink state

The DCB fields and the return values are described below:

- **dcb.bh** is the address of the buffer to receive portal IDs. Each portal ID occupies one word in the buffer.
- **dcb.bl** is the number of words held by the buffer. Note that this is not the number of bytes. It is actually the number of portal IDs that can be placed into the buffer.
- **dcb.operation** is set to the current state of the datalink and can be one of the following:

State	Value	Meaning
OFF	0	The datalink state is now off.
ON	1	The datalink state is now on.
INIT	2	The port hardware is initializing.
BROKEN	3	The datalink failed to initialize.

- **dcb.operation** is the current datalink state.
- **SUCCESS** indicates that the function was successfully completed and the buffer now has the list of portal IDs.
- **BUFFER_TOO_SMALL** indicates that there were more portal IDs than the buffer_length argument allowed for. An incomplete list is returned.

Functions 0AH: Read the Portal Status (dll_read_portal)

The read portal function reads portal data base information for a given portal.

Parameters

AH =	0AH	
ES:BX =	far pointer to DCB	
	dcb.portal_id =	Portal ID from open call
	dcb.bh =	Address of portal status buffer
	dcb.bl =	Size of portal status buffer

Returns

AX =	0000H – SUCCESS
	000BH – UNRECOGNIZED_PORTAL
	0006H – BUFFER_TOO_SMALL
dcb.bl =	number of multicast addresses

The DCB fields and the return values are described below:

- **dcb.portal_id** is the portal id whose data base you want to read from.
- **dcb.bh** is a pointer to the portal status buffer. When the function call returns, this buffer has all the information about the portal. The format of the read portal information buffer is outlined below.
- **dcb.bl** is the length of the buffer in which the portal status is to be returned. If there is not enough space in the buffer, a partial list is returned along with the BUFFER_TOO_SMALL error.
- **SUCCESS** indicates that the routine has read the data base successfully and the information is returned.
- **UNRECOGNIZED_PORTAL** indicates that the portal ID was not recognized by the datalink layer as being that of a valid portal.
- **BUFFER_TOO_SMALL** indicates that the buffer was too small so only a partial list was returned.

The format of the buffer that the dll_read_portal function returns to the caller is:

```
struct  portal_status
{
        int     lost_buffers;
        uchar   enabled_protocol_type[2];
        uchar   enabled_multicasts[MAXMULTICAST][6];
};
```

Where:

lost_buffers	Is a two-byte count of the client buffers that are lost.
enabled_protocol_type	Is a two-byte description of enabled protocol type for this portal.
enabled_multicasts	Is a return list of enabled multicast addresses. The size of the list is always MAXMULTICAST multiplied by 6 bytes per multicast address. The enabled multicast addresses appear first in the list. The rest of the entries are all zero.

Function 0BH: Read the Datalink Counters (dll_read_count)

The read counters function reads the system counters and optionally clears them all.

Parameters

AH =	0BH	
ES:BX =	far pointer to DCB	
	dcb.operation =	[CLEAR ! NULL]
	dcb.bh =	Address of the buffer to hold the datalink counters

Returns

AX =	0000H — SUCCESS (never returns error)

The DCB fields and the return values are described below:

- **dcb.operation** tells the datalink layer that after the counters are read, all the counters should either be cleared (CLEAR=1) or not cleared (NULL=0).
- **dcb.bh** points to a buffer which is to hold the datalink counters.
- **SUCCESS** indicates that the specified counters were read and placed into the client supplied buffer.

The datalink layer maintains several counters for diagnosing network related problems. The datalink counters structure is listed below:

```
struct datalink_counters
{
   int   seconds_since_zeroed;
   long  bytes_received;
   long  bytes_transmitted;
   long  frames_received;
   long  frames_sent;
   long  multicast_bytes_received;
   long  multicast_frames_received;
   long  blocks_sent_initially_deffered;
   long  blocks_sent_single_collision;
   long  blocks_sent_multiple_collisions;
   uint  send_failures;
   uint  send_failure_mask;
   uint  receive_failures;
   uint  receive_failure_mask;
```

```
    uint  unrecognized_frame_destination;
    uint  data_overrun;
    uint  system_buffer_unavailable;
    uint  user_buffer_unavailable;
    uint  collision_detect_check_failed;
};
```

The datalink counters fields are described below:

- **seconds_since_zeroed** (16 bits) counter is reserved for future use.
- **bytes_received** (32 bits) counter indicates the total number of bytes received by the port hardware successfully.
- **bytes_transmitted** (32 bits) counter indicates the total number of bytes transmitted by the port hardware excluding the datalink header bytes. It does not include bytes caused by datalink layer retransmission of messages.
- **frames_received** (32 bits) counter indicates the number of frames received by the datalink.
- **frames_sent** (32 bits) counter indicates the number of frames transmitted by the datalink.
- **multicast_bytes_received** (32 bits) counter counts the number of frames received that were addressed to multicast addresses.
- **multicast_frames_received** (32 bits) counter counts the number of frames received from multicast addresses.
- **blocks_sent_initially_deferred** (32 bits) counter is the total number of times that a frame was deferred on its first attempt. This counter is maintained by the port driver. This counter is not available for all hardware.
- **blocks_sent_single_collision** (32 bits) counter is the total number of times that a frame was successfully sent on the second attempt after its first collision. this counter is maintained by the port driver.
- **blocks_sent_multiple_collisions** (32 bits) counter is the total number of times a frame was transmitted successfully after experiencing multiple collisions on the Ethernet. This counter is maintained by the port driver. This counter is not available for all hardware.
- **send_failures** (16 bits) counter is the total number of times a transmit failed to occur. The reason for the failure is specified by the send_failure_mask field.
- **send_failure_mask** (16 bits) is a bit mask that determines what types of errors occurred that causes the transmit failure counter to increment.

Bit	Reason	Description
15-6	Reserved for furture use	
5	Remote failure to defer	A collision was detected after the slot time for the Ethernet wire.
4	Frame too long	The frame to be transmitted was to long to fit into the maximum length Ethernet message.
3	Open Circuit	There is a short or open circuit on the Ethernet wire.
2	Short Circuit	There is a short or open circuit on the Ethernet wire.
1	Carrier check failed	The hardware could not detect the Carrier on the line, so no transmission was made.
0	Excessive collisions	A node experienced more than the allowed number of collisions and aborted the transmission.

- **receive_failures** (16 bits) counter indicates the total number of frames received with some form of data error. The reason for the failure is specified in the Receive Failure Mask field.
- **receive_failure_mask** (16 bits) is a bit mask which contains the reason(s) that the receive failure counter has been incremented.

Bit	Reason	Description
15-3	Reserved for future use	
2	Frame too long	The length of the receive data was greater than the maximum Ethernet message size.
1	Framing error	Both an odd number of bits was received and a CRC error occured.
0	Block Check (CRC) Error	A CRC error was detected in the receive data.

- **unrecognized_frame_destination** (16 bits) counter indicates the total number of times a frame was discarded because there was no portal with the protocol type (non-802 compatible mode) or multicast address enabled. This includes frames received for the physical address, broadcast address, or a multicast address.
- **data_overrun** (16 bits) counter is the total number of times the hardware lost a frame because it could not keep up with the data rate. This counter is maintained by the port driver. This counter is not available for all hardware.
- **system_buffer_unavailable** (16 bits) counter is the total number of times a frame was discarded because the datalink had insufficient internal buffers to receive a message from the hardware. This counter is maintained by the port driver.

- **user_buffer_unavailable** (16 bits) counter is the total number of times a frame was discarded because a client application buffer was not available to store the message.
- **collision_detect_check_failed** (16 bits) indicates the approximate number of times that collision detect was not sensed after a transmission. This counter is maintained by the port driver.

Function 0CH: Network Boot Request (dll_network_boot)

This function is called to request a remote boot of this node from another node on the network. It does not return unless the boot request fails to find a suitable boot server to boot the node. If the boot request succeeds, the datalink state is left ON and the Maintenance Operation Protocol, MOP, portal ID is opened and enabled. If the boot request fails, the datalink and port driver are left in the OFF state and need to be initialized.

Further information on network booting is presented in the MOP section of this chapter.

Parameters

AH =	0CH
ES:BX =	Far pointer to DCB

Returns

AX =	0000H — SUCCESS (never returns error)

Function 0DH: Enabling a Channel Function (ddl_enable_chan)

The enable channel function is called by the network management software to turn on an Ethernet channel, as well as set the station ID used by the Ethernet hardware.

Parameters

AH =	0DH
ES:BX =	far pointer to DCB
	dcb.source_address = address to set

Returns

AX =	0000H – SUCCESS 0004H – ADDRESS_NOT_SET 000DH – NOT_MULTICAST

The return fields are described below:

- **SUCCESS** indicates that the channel is now enabled. A channel that has been enabled is not necessarily usable. The caller can determine the actual state of the channel by issuing the read_channel function.
- **ADDRESS_NOT_SET** indicates that the physical address for the channel is not set yet, and therefore the channel cannot be enabled.
- **NOT_MULTICAST** indicates that the address that you attempted to enable as your station ID was a multicast address, and it should be a physical address. If this error occurs, and an address was previously set, the previous address is still recognized as the station ID.

Function 0EH: Disabling a Channel (dll_disable_chan)

The disable channel function is invoked to put the channel into the OFF state.

Parameters

AH =	0EH
ES:BX =	Far pointer to DCB

Returns

AX =	0000H — SUCCESS (never returns error)

The return field is **SUCCESS**, which indicates that the channel is disabled.

Function 11H: Read Decparm String Address (dll_readecparm)

This function is invoked to read the current DECPARM string address for the datalink. The address is a pointer to the null terminated ASCII string that describes the MS-DOS path to the DECnet data files. This path is where DECnet-DOS looks for its data files.

Parameters

AH =	11H
ES:BX =	Far pointer to DCB

Returns

AX =	0000H – SUCCESS (never returns error)	
	dcb.bh =	Address of DECPARM string

Function 12H: Set Decparm String Address (dll_setdecparm)

This function is invoked to set the DECPARM string address for the datalink. This address points at null terminated ASCII string that describes the MS-DOS path to the DECnet-DOS data files. This path is where the DECnet driver looks for its data files.

> **NOTE**
> If a client application modifies the DECPARM string address while DECnet-DOS is loaded in the VAXmate workstation, the network may fail.

Parameters

AH =	12H	
ES:BX =	Far pointer to DCB	
	DCB.BH =	Address of DECPARM string

Returns

AX =	0000H — SUCCESS (never returns error)

Function 13H: External Loopback (dll_ext_loopback)

This function invokes an external loopback test for diagnostic purposes. The function requires no arguments. If the hardware successfully loops the message, the SUCCESS return is given. If the message is not looped or the function is not supported, the LB_FAILURE return is given. Loopback testing can be accomplished using a Loopback Terminator.

Parameters

AH =	13H
ES:BX =	far pointer to DCB

Returns

AX =	0000H – SUCCESS
	0002H – CHANNEL_NOT_OFF
	0015H – LB_FAILURE

Maintenance Operation Functions

The VAXmate workstation implements a subset of the DIGITAL Network Architecture Maintenance Operations capability, which is referred to as the Maintenance Operations Protocol (MOP).

MOP services are used by network management software to assist in network configuration determination and network problem diagnosis. The VAXmate MOP implementation also provides for loading a system image from another node on the Ethernet. This is provided to facilitate loading operating systems or standalone applications from a host node.

The MOP functions implemented in the VAXmate workstation are:

- Loop Services
- Console Server Identify Self
- Remote Read Counters
- Network Boot Request

These functions are described in the following sections:

Loop Services

Loop services allow a remote node to loop a message through the client node. The Loop Server functions are provided to allow the VAXmate workstation to respond to MOP Loop Test requests from another system on the Ethernet. These requests can be serviced at any time. The only requirement is that the MOP process not be stopped. Servicing a MOP Loop Test request will not affect the operation of other network software on the VAXmate workstation.

For further information on loop services refer to *Digital Network Architecture Maintenance Operations Functional Specification.*

Console Server Identify Self

The Console Server Identify Self periodically sends the system identification message and sends the system identification message to a requesting system. This function is not visible to application software and is provided to allow this VAXmate workstation to respond to MOP Request ID messages from another system on the Ethernet. This request can be serviced at any time. The only requirement is that the MOP process not be stopped. Servicing a MOP Request ID message will not affect the operation of other network software on the VAXmate workstation.

The VAXmate system periodically transmits an unsolicited system ID message to the MOP multicast address. The message is transmitted at approximately 9 minute intervals. The messages are not transmitted if a portal is active in promiscuous mode.

The VAXmate system ID message conforms to the Digital Network Architecture MOP system ID message. The VAXmate workstation specific fields are the Comm field with a value of 25(decimal), and the System Processor Type field with a value of 7.

For further information on System ID message refer to *Digital Network Architecture Maintenance Operations Functional Specification.*

Remote Read Counters

Remote Read Counters send Ethernet Data Link counters to the requesting system. This function is provided to allow this VAXmate workstation to respond to the MOP Request Counters message from another system on the Ethernet. This request can be serviced at any time. The only requirement is that the MOP process not be stopped. Servicing a MOP Request Counters message will not affect the operation of other network software executing in the VAXmate workstation.

The Counters are counts of events maintained by the Data Link. For further information on Data Link counters refer to *Digital Network Architecture Maintenance Operations Functional Specification.*

Network Boot Request

Network Boot Request requests a remote boot of this node, as part of the VAXmate boot procedure. The default VAXmate boot sequence is the following:

1. Attempt to boot from the floppy drive.
2. Attempt to boot from the optional Hard Disk.
3. Attempt to boot from the network.
4. Repeat the sequence.

A VAXmate network boot can only be initiated by powering on the system or by a system reset (such as pressing Ctrl/Alt/Del). Another system on the Ethernet cannot force a remote boot or a system reset of a VAXmate workstation.

A successful network boot procedure is as follows.

1. Send a MOP Request Program Load multicast message. Multicast address AB-00-00-01-00-00.
2. Nodes on the Ethernet that can perform program loads check their network databases against the Ethernet address of the requesting node. If the address and an associated Load File specifier is found in the data base, the load request can be serviced.

The nodes that can service the request then volunteer to load the VAXmate workstation. They volunteer by sending a MOP Assistance Volunteer message to the Ethernet address of the VAXmate system.

3. The VAXmate workstation sends a directed MOP Request Program Load message to the physical Ethernet address of the first system it receives an Assistance Volunteer message from.
4. The VAXmate system image is now transferred into the workstation.

For VAX/VMS systems, target node Ethernet addresses and associated load files are specified using the Network Control Program, NCP.

For further information on Remote Boot procedures refer to *Digital Network Architecture Maintenance Operations Functional Specification* and VAX/VMS DECnet manuals covering network management topics.

NOTE
Currently VAX/VMS systems require that the system image to be loaded into a remote node be in a file format compatible with RSX/11-S down line loadable images.

Data Link Interface to the MOP Process

The Data Link interface supports two MOP functions.

Function 0FH: Mop Start and Send System ID (dll_start_mop)

This function is called to send a MOP system ID message. The first time the routine is called, it will also start the MOP Loopback server if it is not already running. This function returns no errors.

Parameters

AH =	0FH
ES:BX =	far pointer to DCB

Returns

AX =	0000H — SUCCESS (never returns error)

Function 10H: Mop Stop (dll_mop_stop)

This function is called to halt the operation of MOP and disable all protocol and multicast message processing associated with the MOP process.

MOP STOP must be performed to enable promiscuous mode or take over any of the MOP functions.

The MOP software continues to transmit periodic system ID messages if the MOP Remote Console Protocol Type is not being used by a client application portal. The Digital Network Architecture requires a system ID message transmission every 10 minutes or less. If the client application uses the MOP Remote Console Protocol Type and MOP is stopped, the application must implement periodic system ID message function.

Parameters

AH =	10H
ES:BX =	far pointer to DCB

Returns

AX =	0000H — SUCCESS (never returns error)

Sample Datalink Session

This is an example of a code fragment that performs the following functions:

1. Calls the enable channel function to turn the channel on and enable an address.
2. Calls the datalink open routine to open a datalink portal, enable a protocol type and specify a line state change call-back for the portal.
3. Enables multicast addresses.
4. Requests transmit buffers.
5. Transmits frames to the port driver.
6. Closes the portal.

This example assumes Microsoft C making use of the nontransportable construct of far pointers.

```
        struct DCB dcb_blk_1;                         /* Working DCB */

mop_mon_sysid()
{
        unsigned char hardware_address[6];      /* Hardware address */
        unsigned open_mode,pad_mode,ptypev[2];
        unsigned char multicast_address[6];     /* Multicast to enable */
        extern int line_state_change_far();
        extern int receive_complete_far();
        extern int transmit_complete_far();
        unsigned char cchar;
        struct BUFFERS
        {
                unsigned char used;
                unsigned char far *buffer;
                unsigned char dest[6],src[6];
        } buffers;

/* First thing to do is read the channel state to get our current hardware */
/* address.                                                                 */

        read_channel_state(&dcb_blk_1);
        for (j=0;j<6;j++) hardware_address[j]=dcb_blk_1.dest_address[j];

        printf ("\nThe current hardware address is: ");
        print_address(hardware_address);

        buffer_state=dcb_blk_1.operation;
        printf ("\nThe port state is now: ");
```

```
        give_state(buffer_state);

/* Now enable the channel using the hardware address from the port driver */

        if (give_error(enable_channel(&dcb_blk_1,hardware_address)) !=
SUCCESS)
        {
                return -1;
        }

        dll_disable_mop(&dcb_blk_1);            /* Stop MOP from running */

/* Now that we have enabled the channel successfully, open a portal in     */
/* PADDED Ethernet MODE.                                                    */

        pad_mode=PAD;
        open_mode=Ethernet;
        ptypev[0]=0x60;                         /* Start of MOP Ptype */
        ptypev[1]=0x02;                         /* End of MOP Ptype */

        multicast_address[0]=0xab;
        multicast_address[1]=0x00;
        multicast_address[2]=0x00;
        multicast_address[3]=0x02;
        multicast_address[4]=0x00;
        multicast_address[5]=0x00;

        if ((give_error(dll_open(&dcb_blk_1,
                                pad_mode,open_mode,ptypev,
                                line_state_change_far,
                                transmit_complete_far,
                                receive_complete_far,
                                segrg.cs,-1))) != SUCCESS) return -1;

/* We now have an open portal, so lets enable a multicast address          */

/* We will now send a system ID message (NOT USING THE DATALINK FUNCTION)  */
/* By building the message and transmitting it.                            */

        give_error(dll_enable_multicast(&dcb_blk_1,&multicast_address[0]));
        send_system_id(&multicast_address[0]);

        while (1)
        {
                if (buffers.used == 1) dump_system_id();
/* Free up buffer for datalink layer use.                                  */
                deallocate_buffer(&dcb_blk_1,buffers.buffer);
```

```
/* This will break out of the loop when the first character is typed */

                if (kbd16(&cchar) == 0) break;
        }

        dll_close(&dcb_blk_1);          /* Close the portal */
        dll_enable_mop(&dcb_blk_1);     /* And start MOP */
}

send_system_id(destination,dcb)
        uchar destination[];
{
        int i;
        unsigned char far *buffer;
        if (allocate_transmit_buffer(&dcb_blk_1) == SUCCESS)
        {
                for (i=0;i<6;i++) dcb_blk_1.dest_address[i]=destination[i]
                buffer=dcb_blk_1.bh_address;
                pokeb(buffer_off,buffer_seg,SYSTEM_ID); /* Save opcode */
                pokew(buffer_off+1,buffer_seg,1);       /* Receipt number
                pokew(buffer_off+3,buffer_seg,1);       /* Maint version *
                pokeb(buffer_off+5,buffer_seg,3);       /* length */
                pokeb(buffer_off+6,buffer_seg,3);       /* version */
                pokeb(buffer_off+7,buffer_seg,1);       /* 1. */
                pokeb(buffer_off+8,buffer_seg,0);       /* 0 */
                pokew(buffer_off+9,buffer_seg,2);       /* Funcions availa
*/
                pokeb(buffer_off+11,buffer_seg,2);      /* length */
                pokew(buffer_off+12,buffer_seg,0x41);   /* ... */

                xmit_callback=-1;
                transmit_frame(&dcb_blk_1,buffer,15);
                while (xmit_callback != 0) {}
                deallocate_buffer(&dcb_blk_1,buffer);
        }
        else
        {
                printf ("\nBuffer allocation error in send_systemid");
        }

}

dump_system_id()
{
        struct mop_header far *ptr_mop_msg;
```

```
        ptr_mop_msg=(struct mop_header far *)buffers.bh_buffer;
        if (ptr_mop_msg->msg_type == SYSTEM_ID)
        {
                ptr_mop_data=(unsigned char far *)ptr_mop_msg;
                printf ("\n------ System ID Message -----");
                display_nodes(buffer_number);
                info_type=-1;
                i=sizeof(struct mop_header);
                while (info_type != 0)
                {
                        info_type=ptr_mop_data[i]+(ptr_mop_data[i+1]<<8);
                        i=i+2;
                        if (info_type == 0) break;
                        switch (info_type)
                        {
                                case 1:
                                        i++;
                                        printf ("\nMaintenance Ver: %d.%d.%d",
                                                ptr_mop_data[i++],
                                                ptr_mop_data[i++],
                                                ptr_mop_data[i++]);
                                        break;

                                case 2:
                                        i++;
                                        printf ("\nFunctions: ");
                                        i2=ptr_mop_data[i]+
                                          (ptr_mop_data[i+1]<<8);
                                        printf ("%x ",i2);
                                        i=i+2;
                                        if ((i2 &0x1) == 0x1) printf ("LOOP
");
                                        if ((i2 &0x2) == 0x2) printf ("DUMP
");
                                        if ((i2 & 0x4) == 0x4) printf ("PRIM
");
                                        if ((i2 & 0x8) == 0x8) printf ("MBL
");
                                        if ((i2&0x10) == 0x10) printf ("BOOT
");
                                        if ((i2 & 0x20) == 0x20) printf ("CC
");
                                        if ((i2 & 0x40) == 0x40) printf
("DLC");
                                        if ((i2& 0x80) == 0x80) printf ("CCR
");
```

```
                                        break;

                                case 7:
                                        printf ("\nHardware address: ");
                                        i2=ptr_mop_data[i++];
                                        for (i1=0;i1<i2;i1++)
                                        {
                                           printf ("%2x-",ptr_mop_data[i1+i]
                                        }
                                        i=i+i2;
                                        break;
                                default:

                                        i2=ptr_mop_data[i++];
                                        i=i+i2;
                                        break;
                        }
                }
        }
}

#define TRANSMIT 0x0500

transmit_frame(dcb_blk,buffer_addr,length)
        unsigned int far *buffer_addr;
        int length;
        struct dcb *dcb_blk;
{
        dcb_blk->bl=length;
        dcb_blk->bh_address=buffer_addr;
        return do_dll_call(TRANSMIT,dcb_blk);
}


#define READ_CHANNEL 0x0800

read_channel_state(dcb_blk)
        struct dcb *dcb_blk;
{
        return do_dll_call(READ_CHANNEL,dcb_blk);
}


#define ENABLE_CHANNEL 0x0d00

enable_channel(dcb_blk,address)
        uchar address[];
```

```
        struct dcb *dcb_blk;
{
        int i;
        for (i=0;i<6;i++) dcb_blk->source_address[i]=address[i];
        return do_dll_call(ENABLE_CHANNEL,dcb_blk);
}

#define ENABLE_MOP 0x0f00
#define DISABLE_MOP 0x1000

dll_enable_mop(dcb_blk)
        struct dcb *dcb_blk;
{
        return do_dll_call(ENABLE_MOP,dcb_blk);
}

dll_disable_mop(dcb_blk)
        struct dcb *dcb_blk;
{
        return do_dll_call(DISABLE_MOP,dcb_blk);
}


#define OPEN 0x0100
#define CLOSE 0x0200

dll_open (dcb_blk,pad,mode,ptypev,
          line_state,transmit_complete,receive_complete,
          cs,max_out)
        uchar pad,mode,ptypev[],max_out;
        uint line_state,transmit_complete,receive_complete,cs;
        struct dcb *dcb_blk;
{
        int i;
        dcb_blk->pad=pad;
        dcb_blk->mode=mode;
        for (i=0;i<2;i++) dcb_blk->ptype[i]=ptypev[i];
        dcb_blk->line_state_off=line_state;
        dcb_blk->rcv_callback_off=receive_complete;
        dcb_blk->xmit_callback_off=transmit_complete;
        dcb_blk->line_state_seg=cs;
        dcb_blk->rcv_callback_seg=cs;
        dcb_blk->xmit_callback_seg=cs;
        return do_dll_call(OPEN,dcb_blk);
}
dll_close(dcb_blk)
        struct dcb *dcb_blk;
```

```
{
        return do_dll_call(CLOSE,dcb_blk);
}

#define ENABLE_MULTICAST 0x0300
#define DISABLE_MULTICAST 0x0400

dll_enable_multicast(dcb_blk,multi_address)
        uchar multi_address[];
        struct dcb *dcb_blk;
{
        int i;
        for (i=0;i<6;i++) dcb_blk->source_address[i]=multi_address[i];
        return do_dll_call(ENABLE_MULTICAST,dcb_blk);
}

dll_disable_multicast(dcb_blk,multi_address)
        uchar multi_address[];
        struct dcb *dcb_blk;
{
        int i;
        for (i=0;i<6;i++) dcb_blk->source_address[i]=multi_address[i];
        return do_dll_call(DISABLE_MULTICAST,dcb_blk);
}


#define TRANSMIT 0x0500


transmit_frame(dcb_blk,buffer_addr,length)
        unsigned int far *buffer_addr;
        int length;
        struct dcb *dcb_blk;
{
        dcb_blk->bl=length;
        dcb_blk->bh_address=buffer_addr;
        return do_dll_call(TRANSMIT,dcb_blk);
}


#define ALLOCATE_TRANSMIT_BUFFER 0x0600
#define DEALLOCATE_BUFFER 0x0700

allocate_transmit_buffer(dcb_blk)
        struct dcb *dcb_blk;
{
        return do_dll_call(ALLOCATE_TRANSMIT_BUFFER,dcb_blk);
```

```
}
deallocate_buffer(dcb_blk,buffer_addr)
        unsigned int far *buffer_addr;
        struct dcb *dcb_blk;
{
        dcb_blk->bh_address=buffer_addr;
        return do_dll_call(DEALLOCATE_BUFFER,dcb_blk);
}
```

Local Area Transport

The LAT Ethernet protocol is a DIGITAL proprietary protocol. This section only describes the functional interfaces for use by VAXmate applications. It does not describe the LAT protocol.

The Local Area Transport, LAT, module provides a means for terminal emulators and other programs to communicate with a VAX/VMS host system. This communication is done over the ThinWire Ethernet. The LAT subsystem is designed to eliminate the need for serial communications cabling between a VAXmate workstation and a VAX/VMS host system.

The LAT software is an MS-DOS terminate and stay resident module, LAT.EXE. An application accesses the LAT functions by issuing software interrupts through INT 6AH. The LAT software uses the Data Link interfaces to access the network.

The LAT software does not interfere with other network uses of the workstation. In particular, applications can simultaneously use the LAT services and the MS-Network and DECnet-DOS interfaces. The VAXmate LAT interface supports multiple virtual terminals and other related services, such as simple data transfers with appropriate software. When coupled to a terminal emulator, the LAT software can support VAX/VMS interactive terminal sessions in a fashion similar to a LAT terminal server.

The LAT subsystem provides a service directory facility that supports VAX cluster services. In a VAX cluster, the user logs onto a service offered by one or more cooperating DECnet nodes as opposed to logging onto a specific node. This allows logging on to occur independent of whether a particular node is functioning or overloaded. In addition, in most VAX clusters the individual nodes are themselves a service. This facilitates logging on to a particular node in the cluster. VAX/VMS LAT software running on the VAXmate workstation automatically selects the cluster node with the lightest load. This provides a form of load balancing for interactive jobs.

The following sections describe:

- LAT services
- LAT data structures
- LAT functions
- Sample terminal program

LAT Services

This section describes various LAT services including:

- LAT command line
- Service directory
- Sessions and slots
- Session start
- Data exchange
- Flow control
- call-back routines
- Closing a session

LAT Command Line

Assuming a path is defined to LAT.EXE or lies within the current default subdirectory, the LAT software is invoked either from a batch file or by the user typing LAT at the MS-DOS prompt. A second invocation has no affect and does not install a second copy of the software. If there is insufficient memory present, LAT exits with an error code of 8 returned to MS-DOS. No error message is issued to the user.

Three command line switches are provided. If you leave out a switch or specify a -1 switch value, the default is used. There is no provision for command lines that exceed one line in length.

The command line switches are:

- /D:nn

  This switch is used to increase the default size of the LAT service directory. The value nn is an unsigned integer representing the number of entries in addition to the default of 10 entries. Each additional entry causes an extra 47 bytes of memory to be allocated rounded to the nearest 16-byte paragraph. An entry is assumed to be one service offered by one node.

  The maximum number of entries is 1054 (1044 + default). This reserves a total of approximately 49,072 bytes for the service table. If the overflow call-back is not enabled, the only effect of a service table overflow is that new services are not added to the table.

  The default size of the directory is 10 entries = 470 bytes.

- /G: 1, 2, 3, 32...

  This switch is used to reduce the overhead of servicing multicast service announcement messages, or to prevent the service table from filling with unwanted service names. This control uses LAT group codes to disable selectively the processing of multicast messages. Group code control is only advised when the number of services at a site routinely exceeds available memory on the workstation.

  Each number represents a LAT address group code. Groups are numbered from 0 to 255. If the /G: switch is used, only the specified codes are enabled.

  The default is all group codes enabled.

  For example, to enable group codes 0,2,3 and 54, invoke the LAT software by typing:

  ```
  LAT /G:0,2,3,54
  ```

- /R:n

  This switch is used to set the number of retransmits permitted for a circuit. The default is eight retransmits allowed before the circuit is stopped. The minimum is four and maximum is 255.

Service Directory

The VAXmate LAT software listens to the Ethernet for messages from host systems offering LAT services. These are multicast Ethernet messages that identify a LAT server. The frequency at which a server identification message is transmitted is a function of the VAX/VMS configuration.

When the VAXmate LAT software receives a LAT multicast message, the sending service is added to the service table maintained by the software. If the service table is full, the service name is not kept, and an appropriate error message is returned by a LAT status call. An application can also enable a call-back notification when this error occurs.

Under certain conditions, a LAT server can become unreachable. The VAXmate LAT software detects this condition when the number of retansmits to the server reaches the maximum specified by the /R switch. If this occurs, the service table entry for the server is marked as unreachable. Subsequent requests to read the table do not return this server name. The server name again becomes readable when the server receives a multicast LAT service message.

A LAT application program can read the service table through the 6AH software interrupt.

You can specify that a preferred LAT server be entered into the service table at LAT.EXE startup time. This forces a preferred LAT server into the tables and eliminates waiting for the multicast message to arrive.

At startup time, the LAT software attempts to read the DECNODE.DAT file. Within DECNODE.DAT, you can specify preferred LAT service names, which are entered into the service table. If the number of preferred service names exceed the size of the startup service table size, the LAT software automatically increases the size of the table to accommodate the number of entries.

You can specify or delete preferred nodes using the DECnet-DOS NCP utility.

LAT Sessions and Slots

A LAT session is conducted between an application and a LAT server. The VAXmate LAT software creates one virtual circuit between itself and a particular LAT server. A VAXmate application or multiple VAXmate applications that want to communicate with that server, do so over the single virtual circuit.

The actual LAT data structure definitions use the term SLOT as an alternative for the term SESSION. A LAT slot is a unit of data being transmitted or received as a part of the session. Slots are data or buffers of data that are being exchanged between an application and the LAT server.

The VAXmate LAT software supports up to 4 virtual circuits and a maximum of 10 slots.

Session Start

To start or open a session, the application passes a pointer to a data structure, LAT's Session Control Block, (SCB). The LAT SCB contains the name of the desired host service. The first open call to a particular service name creates a virtual circuit. All subsequent opens to the same service name use the same virtual circuit. The open call returns a handle (8-bit integer) to refer to the session in subsequent functions.

The LAT Session Control Block, provided by the application, is used by the LAT software to store data and control its flow between the VAXmate workstation and the host node.

> **NOTE**
> It is the responsibility of the application to pass valid SCBs and handles across the LAT interface. Failure to pass valid SCBs and handles can produce unpredictable results.

Data Exchange

After the session is established, data received for the application program is made available one character at a time. Characters are obtained using the Read Data function.

The LAT software can transmit multiple characters in a packet to the host. However, application programs can only transmit one character at a time to the LAT interface.

Flow Control

Flow control is handled automatically by the protocol.

LAT Call-Back Routines

Application programs can specify an address to be called when certain conditions occur. This is termed a call-back routine. These routines are specified in the SCB.

The following considerations apply to all LAT call-back routines.

- call-back routines are accessed by means of a 'far call' and must end with a 'far return'.
- At the time of the call, interrupts are enabled.
- MS-DOS may be interrupted at Call Back time. Do not execute MS-DOS functions from within a call-back routine.
- A call-back on receive is done once for each received data buffer. It is not done for each character received.
- A call-back on transmit is executed when the transmit buffer is completely emptied by the circuit logic.

Closing the LAT Session

When a session is terminated and no other sessions remain on the virtual circuit, the LAT driver shuts down the virtual circuit.

> NOTE
> Failure to explicitly close the LAT session can produce unpredictable results. The LAT software needs to know when the application has finished a session.

Data Structures

The following data structures are used to communicate information between an application and the LAT software.

The LAT Session Control Block

The LAT SCB structure is set up by the application. It is used to control data exchange across the LAT interface. The SCB is pointed at by the ES:BX register pair when a Open Session request is issued. After the Open Session, ES:BX should not point at the SCB; the session handle is sufficient to access the LAT services. After a session is opened, the SCB cannot be moved.

The LAT SCB is described below:

```
SCB                 STRUC

service             DB   18 DUP (0)
node                DB   18 DUP (0)
port                DB   18 DUP (0)
session_stopped     DD    0       ; Address of a call-back routine
table_overflow      DD    0       ; Address of a call-back routine
transmit_notify     DD    0       ; Address of a call-back routine
receive_notify      DD    0       ; Address of a call-back routine
session_status      DW    0
slot_state          DW    0
local_credits       DB    0
vcb_pntr            DD    0       ; Pointer to Virtual Circuit Block
                                  ; vcb_offset, vcb_segment.
back_slot           DW    0
forward_slot        DW    0
rem_slot_id         DB    0
loc_slot_id         DB    0
slot_byte_count     DB    0
remote_credits      DB    0
tx_slot_data        DB    255 Dup(0); Transmit buffer -
                                    ; Contains the actual transmit data.
num_slots           DB    4         ; Number of entries on Slot_ptr_table.
                                    ; Four is the recommended number.
num_occupied        DB    0
next_rx_slot        DB    0
cur_buf_slot        DB    0
Rx_Slot_Pntr        DW    0

Slot_ptr_table      DW    OFFSET slot_1 ; Start of table of 5 session
                    DW    OFFSET slot_2 ; buffer offsets.
                    DW    OFFSET slot_3
                    DW    OFFSET slot_4
                    :     :      :
                    DW    OFFSET slot_n ; Four entries is the recommended
                                        ; table size.
```

```
slot_1          DB    259 DUP(0)     ; Session buffer 1
slot_2          DB    259 DUP(0)     ; Session buffer 2
slot_3          DB    259 DUP(0)     ; Session buffer 3
slot_4          DB    259 DUP(0)     ; Session buffer 4
:               :     :   :
slot_n          DB    259 DUP(0)     ; Four buffers are recommended.

SCB             ENDS
```

The SCB structure is further described in the following section:

- **Service**, which is initialized by the application, is the name of the service requested. This is in ASCII format terminated by a null byte (0). The LAT software converts lower case to upper case.
- **Node** is RESERVED for future use.
- **Port** is RESERVED for future use.
- **Session_Stopped**, which is initialized by the application, is the address of a routine to call when the session has been stopped.

  If the field contains 0, no call-back notification is given.

  The LAT call-back routine is entered with the data in Table 18-4.

Table 18-4 LAT Call-Back Routine

Register	Description
AH = 1	Stop session received.
AH = 2	Stop message received. AL = Host reason code.
AH = 3	Circuit has failed due to excess retransmits.
AH = 4	Illegal buffer (slot) has been received. Type as follows: AL = 1 An unknown SLOT_TYPE value is received. AL = 2 A non-zero SRC_SLOT_ID in a received Stop slot. AL = 3 A zero SRC_SLOT_ID in a Start slot. AL = 4 A Start slot received in the Run state without an intervening Stop slot. AL = 5 A Reject slot received in the Run state. AL = 6 A Data_a or Data_b slot arrives which contains data (consumed a remote credit), but no user buffer is available (no credit was extended).

Table 18-4 LAT Call-Back Routine (cont.)

Register	Description	
	AL = 7	A Run slot with a zero SRC_SLOT_ID.
	AL = 8	A start slot with an invalid service class is received.
	AL = 9	The Attention_slot must-be-zero field is not zero while the LAT software is in the run state.
AH = 5	Illegal message has been received. Type as follows:	
	AL = 1	RESERVED
	AL = 2	An unknown MSG_TYPE in a received message.
	AL = 3	A non-zero SRC_CIR_ID in a received Stop message.
	AL = 4	A zero SRC_CIR_ID in a Start or Run message.
	All other AL values are reserved for future use.	
AH = 6	User requested disconnect.	
All other AH values are reserved for future use.		

- Table_overflow, which is initialized by the application, is the address of a routine to call when the LAT service table overflows.

  If the field contains 0, no call-back notification is given.

  This call-back routine has no data.

- Transmit_notify, which is initialized by the application, is the address of the Call Back routine to call to signal a transmit completion. This call-back is invoked when the transmit buffer is completely emptied. A value of zero disables this option. It can be changed at any time, providing the specified routine is already in place and ready for use.

  If the field contains 0, no call-back notification is given.

- Receive_notify, which is initialized by the application, is the address of the Call Back routine to call to signal a receive has occured. This call-back is invoked when the receive buffer is completely full. A value of zero disables this option. It may be changed at any time, providing the specified routine is already in place and ready for use.

  If the field contains 0, no call-back notification is given.

- Session_status is the current status of the session. See the section "Session Status Word Definition" for a complete description.

- Slot_state is the protocol engine state of this session and is used by the LAT software (0 = halted).

- Local_credits, which you should initialize to zero, is used by the lat software.

- VCB_pntr is a Long Pointer to Virtual Circuit Block used for this session and is used by the LAT software. The actual format of the pointer is:

```
VCB_offset    DW   0
VCB_segment   DW   0
```

- Back_Slot is an index to the back SCB on this circuit and is used by the LAT software.
- Forward_Slot is an index to the forward SCB on this circuit and is used by the LAT software.
- Rem_Slot_Id is used by the LAT software.
- Loc_Slot_Id is used by the LAT software.
- Slot_Byte_Count is the number of Tx_slot_data bytes to be transmitted and used by the LAT software.
- Remote_Credits is used by the LAT software.
- Tx_Slot_Data is the transmit buffer that contains actual data to be transmitted to the host.
- Num_Slots, which is initialized by the application, is the number of receive data slot buffers.
- Num_Occupied MUST BE ZERO and is used by the LAT software.
- Next_Rx_Slot MUST BE ZERO and is used by the LAT software.
- Cur_Buf_Slot MUST BE ZERO and is used by the LAT software.
- Rx_Slot_Pntr, which is initialized by the application, equals (Offset of Slot_1 entry) + 4 and is used by the LAT software.
- Slot_Ptr_Table is initialized by the application. Each entry points to a slot_x field in the following data area. Slot_Ptr_Table is used by the LAT software.
- Slot_1 through Slot_n are the receive data buffer areas and are used by the LAT software. These data buffers must completely reside in the same data segment as the SCB.

  The application programmer specifies the number of receive data buffers. Each receive data buffer consumes 259 bytes for the actual buffer and 2 bytes for Slot_Ptr_Table entry. The minimum recommended number of receive data buffers is two. For most configurations, specifing four is adequate.

NOTE
The programmer should refer to the Call Back section for further information on call-back routines. All pointers are in the form of offset - segment.

Session Status Word Definition

This is the field labeled session_status in the SCB.

Status is reported in the form of one byte of bit flags followed by one byte of explanation. Bit set (= 1) means the condition is true.

Status Word - Check for circuit and session state PRIOR to status call.

Status Word - First Byte - Contains bit flags as described below:

Bit 7-5	Reserved
Bit 4	Host sent a stop slot (stop session command)
Bit 3	Circuit Failure, reason in second byte
Bit 2-1	Reserved
Bit 0	Transmit buffer busy

Status Word - Second Byte - Contains reason number code on circuit or session failure as described below:

Value	01H = Stop slot received.
	02H = Stop message received.
	03H = Circuit has failed due to excess retransmits.
	04H = Illegal slot has been received.
	05H = Illegal message has been received.
	06H = User has requested disconnect.

LAT Functions

The LAT services are accessed through INT 6AH. The AH register contains the function code for the requested service. Each access requires that FFH is in the DH register.

> **NOTE**
> Application software must save and use the session handle returned from the open session service. Failure to use the correct handle can cause the VAXmate software environment to hang.

Table 18-5 lists the available LAT functions, which are described in the sections following the table.

Table 18-5 Interrupt 6AH: LAT Functions

Function	Description
AH = 03H	Get status
AH = D0H	Open session
AH = D0H	Close LAT session
AH = 02H	Read data
AH = 01H	Send data
AH = D5H	Get next service name
AH = D6H	LAT service table reset
AH = 0D1H	Send break signal

Function 03H: LAT Get Status

Parameters

AH =	03H	
DH =	FFH	
DL =	xxx	Where xxx is the session handle returned from the open session call

Returns

AH =	Status byte (Set bits indicate condition)	
	Bit 7-6	Reserved
	Bit 5	Transmit buffer empty
	Bit 4	Reserved
	Bit 3	Session in start state
	Bit 2	Session not active
	Bit 1	Unable to queue transmit data
	Bit 0	Receive data available
ES:BX =	Reserved	
ES:DX =	Reserved	

Function D0H: Open Session

This service creates buffers and starts the LAT session.

Parameters

AH =	D0H
AL =	FFH
ES:BX =	Long pointer to LAT Session Control Block, SCB.
DH =	FFH

Returns

AH =	00H Success, or	
	7-5	Reserved
	4	No more sessions available. A maximum of 5 sessions per Virtual Circuit.
	3	Data buffer specification error.
	2	No more sessions available.
	1	No more virtual circuit blocks.
	0	Service not in table or name error.
DL =	Session handle for subsequent service requests over this session connection.	

If a virtual circuit to the selected service is not active, a virtual circuit to the node offering the service is created in addition to the requested session.

Application software must save and use the session handle returned from the OPEN SESSION service. Failure to use the correct handle can cause unpredictable results in the VAXmate software environment.

Function D0H: Close LAT Session

Parameters

AH =	D0H	
AL =	00H	
DH =	FFH	
DL =	xxx	Where xxx is the session handle returned from open session call.

Returns

AX =	0000H	no error
AX =	0001H	No such active session
AX =	0002H	Session not in running state. Retry again after a short delay.

Before closing the session, the application should confirm that all data has been transmitted. Any receive buffers not empty from this session are freed.

Function 02H Read Data

Parameters

AH =	02H	
DH =	FFH	
DL =	xxx	Where xxx is the session handle returned from open session call.

Returns

AH =	Bit pattern as below. (Set bits indicate condition)	
	7	No character read
	6-0	Reserved
AL =	Received Character	

Function 01H: Send Data

Parameters

AL =	Character to be sent	
AH =	01H	
DH =	FFH	
DL =	xxx	Where xxx is the session handle returned from the open session call.

Returns

AH =	00H success, or	
	Bit 7	Unable to queue character
	6-0	Reserved

Function D5H: Get Next LAT Service Name

This service is used by the application to read the entries in the service table. To read the entire table, the application issues successive requests. The LAT software does not report duplicate services or services that are unavailable because the network node is not currently reachable.

Parameters

AH =	D5H
ES:BX =	Long pointer to buffer for the returned service name. The buffer must be at least 17 bytes long.
DH =	FFH

Returns

AH =	00H Success, or FFFFH end of table - no service name available.
ES:BX =	Long pointer to service name terminated by a zero byte.

Function D6H: LAT Service Table Reset

In addition to clearing the LAT service table, this function forces the Get Next LAT Service request to return the first entry in the service table.

Parameters

AH =	D6H
DH =	FFH

Returns

AX	Number of services entered into the service table. This number varies with time as the service table fills.
BX	FFFFH Service table has overflowed.

Function D1H: Send Break Signal

This service is analagous to sending a break signal through a modem.

Parameters

AH =	D1H	
DH =	FFH	
DL =	xxx	Where xxx is the session handle

Returns

AX =	0000H	Success, or
	8000H	Unable to send break signal

Sample Terminal Program

The following is an example of a simple terminal program that can operate on a VAXmate workstation or compatible computer.

```
TITLE   A simple terminal to test the LAT driver.
PAGE    60,132
NAME    term

;****************************************************************************
;*                                                                          *
;*  Copyright (c) 1985, 1986                                                *
;*  by DIGITAL Equipment Corporation, Maynard, Mass.                        *
;*                                                                          *
;*  This software is furnished under a license and may be used and  copied  *
;*  only  in  accordance  with  the  terms  of  such  license and with the  *
;*  inclusion of the above copyright notice.  This software or  any  other  *
;*  copies  thereof may not be provided or otherwise made available to any  *
;*  other person.  No title to and ownership of  the  software  is  hereby  *
;*  transferred.                                                            *
;*                                                                          *
;*  The information in this software is subject to change  without  notice  *
;*  and  should  not  be  construed  as  a commitment by DIGITAL Equipment  *
;*  Corporation.                                                            *
;*                                                                          *
;*  DIGITAL assumes no responsibility for the use or  reliability  of  its  *
;*  software on equipment which is not supplied by DIGITAL.                 *
;*                                                                          *
;****************************************************************************

cr      EQU     13      ; Carriage return
tab     EQU      9      ; Tab
lf      EQU     10      ; Line Feed
lat_int EQU    6AH      ; LAT INTerrupt

;*********************************************************************
;*                                                                   *
;*      A simple terminal program using the PC  LAT Driver           *
;*                                                                   *
;*      To build:                                                    *
;*      MASM TERM;                                                   *
;*      LINK TERM;                                                   *
;*      EXE2BIN TERM TERM.COM                                        *
;*                                                                   *
;*      To invoke, after LAT is loaded:                              *
;*      TERM service_name                                            *
```

```
;*                                                                     *
;***********************************************************************

; Dummy segment used to determine if LAT Driver has been installed.

page0   SEGMENT AT      0

ORG     lat_int*4       ; Location of LAT INT in page zero.

lat_entry       DD      0

page0   ENDS

cseg    SEGMENTPUBLIC 'codeseg'

ASSUME  CS:cseg,DS:cseg,ES:cseg,SS:NOTHING

  ORG     100h          ; Origin for .COM file

main     PROC    NEAR

start:

  MOV     DX,OFFSET hello_message ; Issue greeting message to user.
  MOV     AH,9                    ;  Function = write string.
  INT     21h                     ;  From MS-DOS

  CLD                             ; Set to auto-increment.
  MOV     SI,80H                  ; Location of command line.
  LODSB                           ; First byte = count in AL

  CMP     AL,16                   ; Greater then 16?
  JBE     st_03
  JMP     service_error           ;  Yes, this is an error.

st_03:

  CMP     AL,1                    ; Less than 1?
  JA      st_02                   ;  No, we are fine.

  JMP     service_error

st_02:

  MOV     CL,AL                    ; Count in CX
  XOR     CH,CH
  MOV     DI,OFFSET Service        ; Destination in ES:DI
```

```
  XOR    CH,CH                          ; Zero out CH

copy_loop:

  LODSB                                 ; Load first characters.
  CMP    AL,20H                         ; Space?
  JE     st_04                          ;  Spaces shall be evaporated!

  STOSB                                 ; No, save the character.

st_04:

  LOOP  copy_loop                      ; Copy the service name into the scb.
  MOV   AL,CH                          ;  Terminate the string with a zero byte.
  STOSB                                ;  ..

  CALL  check_installation             ; Check to see if LAT is installed.
  JZ    st_01                          ;  Yes, proceed.

  MOV   DX,OFFSET no_lat_message ; Load pointer to no lat message.
  MOV   AH,9                           ;  Function = write string.
  INT   21h                            ;  Call MS-DOS
  JMP   error_exit

st_01:

  CALL  lat_initialization             ; Execute the LAT init call.

main_loop:

  XOR   AX,AX                         ;Zero AX and reset flags
  MOV   AH,01h                        ;Keyboard poll
  INT   16h                           ;From the BIOS
  JZ    rxd                           ;No character at Kbd, check serial port

;Retrieve character from keyboard buffer

  MOV  AH,00h                     ;Function, read character
  INT  16h                        ;From the BIOS

;Check it for an F1 = Exit from terminal program

  CMP  AX,3B00h                    ;F1 Key ?
  JNE  yx_01                      ; No, continue.

; Explicit stop now implemented!
```

```
  MOV  AX,0D000h             ; Function, close session.
  MOV  DX,WORD PTR handle     ; Session handle in DX.
  INT  lat_int

  JMP  exit

yx_01:
  CMP  AX,3C00h              ;F2 key ?
  JNE  yx_02                ; No, continue

; Send a break signal
  MOV  AH,0D1H               ;
  MOV  DX,WORD PTR handle     ;
  INT  lat_int              ; Send break signal

yx_02:
  CMP  AL,08h               ; Backspace?
  JNE  ml_01                ;  No, continue.
  MOV  AL,07Fh               ; Yes, map to delete.

ml_01:

;Send out the character

  MOV  CX,100               ; We will try and xmit 100 times.
  MOV  AH,01h               ;Function, port_write

txd:

  MOV  DX,WORD PTR handle     ; Use handle given by LAT
  PUSH AX                   ;Preserve Ax destroyed by LAT INT
  INT  lat_int               ;Send the character via the BIOS
  TEST AH,80h               ;Test for character sent

  POP  AX                   ;Restore Ax
  JZ   rxd                  ;Check for another character
  LOOP txd                  ;Try again if character not sent

;See if there is a character received

rxd:

  TEST session_status,1000b   ; Circuit stopped?
  JNZ  circuit_dead

  MOV  AH,03h               ; Status
```

```
  MOV  DX,WORD PTR handle     ; Port
  PUSH ES
  INT  lat_int               ; Get status
  POP  ES

  TEST AH,01h               ; Character available?
  JZ   main_loop             ; No character, poll keyboard again

; Read the character from the LAT buffer.

  MOV  DX,WORD PTR handle     ;Use handle to LAT session
  MOV  AH,02h               ;Function, port_read
  INT  lat_int               ;From the BIOS
  TEST AH,80h               ;Test for character received
  JNZ  main_loop             ; If so, try keyboard!

  OR   AL,AL                ; Null?
  JZ   main_loop             ; If so, don't display

; CMP  AL,09h               ; Tab?

  AND  AL,07Fh               ;Mask out bit 8
  MOV  AH,0Eh               ;function, write TTY
  XOR  BX,BX                ;Display page = 0
  INT  10h                  ;BIOS write_teletype call
  JMP  main_loop             ;All done, poll keyboard again

exit:

  MOV  AX,4C00h
  INT  21h                  ; Normal MS-DOS exit

service_error:

  MOV  DX,OFFSET bad_service_mess ; Bad service message.
  MOV  AH,9                      ;  Function = write string.
  INT  21h                       ;  Call MS-DOS.

error_exit:

  MOV  AX,4C01h                   ; Error level = 1
  INT  21h                       ;  Call MS-DOS.

main   ENDP

lat_initialization     PROC   NEAR
```

```
  MOV  DX,OFF00h              ; This INT is for LAT
  MOV  BX,OFFSET scb          ; ES:BX points to lccb
  MOV  AX,0D0FFh              ; Extended function
  INT  lat_int                ;  Invoke LAT
  OR   AH,AH                 ;  Any errors?
  JZ   li_go                 ;  AH = zero, no errors.

  TEST AH,1                  ; Service not in directory?
  JZ   li_01                 ;  No.

  MOV  DX,OFFSET no_service_message  ; Issue not in directory message.
  MOV  AH,9                        ;  Function = write string.
  INT  21h                         ;  Call MS-DOS

li_01: ; Use this general message for all other failures for now.

  MOV  DX,OFFSET init_failure ; Send failure message to user.
  MOV  AH,9                  ;  Function = write string.
  INT  21h                   ;  Call MS-DOS
  JMP  SHORT error_exit        ;  Exit with error.

li_go:

  MOV  WORD PTR handle,DX     ; Save handle to session

li_exit:

  RET

lat_initialization     ENDP

circuit_dead   PROC   NEAR

  MOV  DX,OFFSET dead_message ; Load offset to circuit dead message.
  MOV  AH,9                  ;  Function = write string.
  INT  21H                   ;  Call MS-DOS
  JMP  SHORT exit              ;  Exit.

circuit_dead   ENDP

PAGE

;*********************************************************************
;*                                                                   *
;*    P R O C E E D U R E  c h e c k _ i n s t a l l a t i o n      *
;*                                                                   *
```

```
;*      Entry: Nothing                                                   *
;*      Exit:  Z flag set = LAT installed.                               *
;*                                                                       *
;************************************************************************

check_installation      PROC    NEAR

  PUSH AX                       ; Preserve registers.
  PUSH CX                       ; .
  PUSH SI                       ; .
  PUSH DI                       ; .
  PUSH ES                       ; . .

  CLD                           ; Set the direction flag to forward.
  XOR   AX,AX                    ; Set ES to page zero.
  MOV   ES,AX                    ; . .

ASSUME  ES:page0

  LES   DI,DWORD PTR lat_entry ; ES:DI => lat_int entry

ASSUME  ES:NOTHING

  MOV   CX,3                    ; Compare 3 bytes
  SUB    DI,3                    ;  Starting at entry -3
  MOV    SI,OFFSET lat_string    ; Local string for compare.

REPZ     CMPSB                   ; Compare it!

  POP   ES                      ; Restore registers.
  POP   DI                      ; .
  POP   SI                      ; .
  POP   CX                      ; .
  POP   AX                      ; ..

  RET

lat_string      DB   'LAT'

check_installation      ENDP

PAGE

dead_message       DB cr,lf,'Circuit disconnected!',cr,lf,'$'
hello_message      DB cr,lf,'LAT test terminal now connecting.',cr,lf,lf,'$'
init_failure       DB cr,lf,'Initialization call failed!',cr,lf,lf,'$'
no_lat_message     DB cr,lf,'LAT Driver not installed!',cr,lf,lf,'$'
```

```
no_service_message DB cr,lf,'Requested service not in directory!',cr,lf,lf,'
bad_service_mess   DB cr,lf,'Bad Service Name!',cr,lf,lf,'$'

handle          DW      0       ; Handle for LAT session

;
; SCB = Session Control Block. Structure used by client application
;        to arrange for data exchange.
;

scb             LABEL  WORD

Service          DB  18 DUP (0) ; Requested service.
Node             DB  18 DUP (0) ; Reserved for future use.
Port             DB  18 DUP (0) ; Reserved for future use.

; ****************************************************************
; The following four call-back addresses must be initialized to 0
; if call-backs are not desired for each condition.

Session_Stopped  DD     0        ; Session stopped notification routine.
Table_overflow   DD     0        ; Service table overflow notification routir
Transmit_notify  DD     0        ; Routine to call when slot is transmitted.
Receive_notify   DD     0        ; Routine to call when a slot is received.

session_status   DW     0        ; Status word
slot_state       DW     0        ; Used by LAT Driver - initialize to 0
local_credits    DB     0        ; Used by LAT Driver - initialize to 0


vcb_offset       DW     0        ; Used by LAT Driver. Pointer to LAT
vcb_segment      DW     0        ; Driver's internal circuit block.

back_slot        DW     0        ; Used by LAT Driver - initialize to 0
forward_slot     DW     0        ; Used by LAT Driver - initialize to 0

; Transmit slot buffer - Contains actual transmit slot.

rem_slot_id      DB     0        ; Used by LAT Driver - initialize to 0
loc_slot_id      DB     0        ; Used by LAT Driver - initialize to 0
slot_byte_count  DB     0        ; Used by LAT Driver - initialize to 0
remote_credits   DB     0        ; Used by LAT Driver - initialize to 0


tx_slot_data     Db 255 Dup(0)  ; Transmit slot data buffer.
;
; Transmit data area
```

```
;
; >>>>>>>>> The following variable is initialized by the client application!!
;
Num_slots         DB    4  ; Number of receive data slot buffers
                           ; in this structure. Value of 4 is suggested.

Num_occupied      DB    0  ; Number of occupied slots.
Next_rx_slot      DB    0  ; Index - Next slot to be used for receive slot.
Cur_buf_slot      DB    0  ; Index - Current slot sending characters to client.
;
;>>> The following variable must be initialized by the client application!
;
Rx_Slot_Pntr      DW    OFFSET Slot_1+4 ; Offset of the first character
                                        ;  to be taken by client.
;
;>>> The following table of pointers must be initialized by the client
;     application!

Slot_Ptr_table    LABEL  WORD

                 DW      OFFSET slot_1
                 DW      OFFSET slot_2
                 DW      OFFSET slot_3
                 DW      OFFSET slot_4

;>>>>>> The following data definitions are the actual receive data buffers.

slot_1           DB      259 DUP(0)
slot_2           DB      259 DUP(0)
slot_3           DB      259 DUP(0)
slot_4           DB      259 DUP(0)

CSEG   ENDS

       END    start
```

Session

This section describes the VAXmate MS-Network Session Level interface. This is the recommended interface for applications that want to use network services provided by the Microsoft MS-Network environment. DIGITAL has added a number of extensions to the standard MS-Network interface to support node name and node address manipulation.

Application programs access the services of the MS-Network Session Level through INT 2AH. The Session Level interface is implemented by a terminate and stay resident emulation module named SESSION.EXE. This emulation software uses DECnet-DOS as the network transport layer of the Microsoft network architecture. Most session services are mapped into DECnet-DOS services for subsequent processing by the network.

The standard unit of communication between applications using the session level interface is the message. A program on one computer sends messages to a program on another computer. The sender is notified if the message is not received by the target application. The communication channel over which the messages are sent is called a virtual circuit. Each program refers to the other by a name. A virtual circuit is a communication channel between two named programs. The session layer software translates program names into network addresses and creates and maintains the virtual circuits between communicating applications.

In addition to name and message services, the session level provides a datagram service. Datagrams are small packets of data that are sent to other programs. Unlike messages, delivery of datagrams is not guaranteed. The sender is not notified of delivery or non-delivery of the datagram. The session module implements the datagram service by directly accessing the datalink.

Figure 18-3 shows the general flow and interfaces between each of the layers that are involved in implementing the session interface.

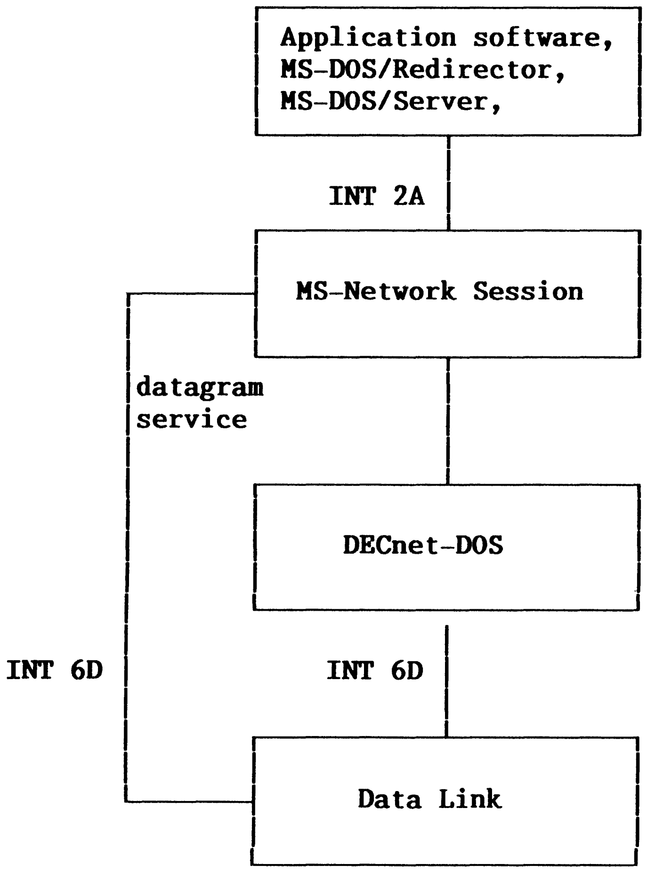

Figure 18-3 Session Interface Implementation

The VAXmate MS-DOS operating system redirects file and print I/O over the network using the INT 2A interface provided by the MS-Network Session software. The operating system module responsible for distinguishing between local and network file access is called the Redirector. The Redirector manages the entire remote file access transaction. The Redirector communicates over the network with file and print server software located on another VAXmate workstation or on a VAX/VMS system. The Actual file access protocol used by the Redirector is called the Server Message Block Protocol, SMB protocol. The SMB is just one of many protocols that can be used for network transactions through the session level interface.

Software Capabilities

An application accesses the MS-Network Session Level services through the INT 2A interface. The INT 2A is issued with register AH = 1. This identifies the access as an MS-Network Session Level request. As part of the access, the application must define a Session Control Block, SCB for each request. The fields in this data structure direct the session level and return status to the application. The application points at the SCB using the ES:BX register pair.

The VAXmate workstation software provides a set of DIGITAL-specific extensions to the session level interface for manipulating node names and node addresses. The DIGITAL-specific services are accessed through the INT 2A interface. The INT 2A is issued with register AH = DCH. This identifies the access as a DIGITAL extension. As part of the access, the application must define a DIGITAL Session Control Block, DSCB, for each request.

All INT 2A invocations that do not have AH set to 0, 1 or DCH are passed on to the INT 2A service routine that was present before the Session Level software was loaded.

All MS-network functions can be performed synchronously. Certain functions can be performed both synchronously or asynchronously. When an application issues the synchronous form of a command, execution of the application stops until the session level interface completes the request. When the asynchronous form of a command is used, the application can test, or poll, for completion or request that a routine be called upon completion. Completion routines are called Asynchronous Notification Routines (ANR).

In addition to the INT 2A interface the session software supports an installation check function through the INT 2F interface. This request is supported for compatibilty with industry standard versions of the MS-Network session interface.

MS-Network Session Control Block

An application accesses the MS-Network Session Level services through the INT 2A interface. The INT 2A is issued with register AH = 1 identifying this request as an MS-Network Session Level request. As part of the access, the application points at the Session Control Block, SCB, using the ES:BX register pair.

An SCB is defined for each INT 2A invocation. This allows the application program to post multiple simultaneous session level requests.

The following is a C language structure description of the Session Control Block, SCB. This data structure is used for all MS-Network compatible session accesses. Session Control Block data structure is described as follows:

```
struct    scb
{
    byte   scb_command;    /* function code */
    byte   scb_error;      /* error code */
    byte   scb_vcid;       /* virtual circuit identifier */
    byte   scb_num;        /* name number (for datagram) */
    long   scb_baddr;      /* buffer address */
    int    scb_length;     /* data buffer length in byte */
    byte   scb_rname[16];  /* remote name */
    byte   scb_lname[16];  /* local name */
    byte   scb_rto;        /* receive timeout */
    byte   scb_sto;        /* send timeout */
    long   scb_async;      /* address of ANR */
    byte   scb_res1;       /* reserved */
    byte   scb_done;       /* function pending flag */
    byte   scb_res2[14];   /* reserved */
};
```

Table 18-6 describes the Session Control Block fields.

Table 18-6 Session Control Block Fields

Field Name	Size	Description
scb_command	1 byte	Contains the function request code. For functions that support the asynchronous mode, the high order bit of the byte is set to indicate asynchronous operation of the request.
scb_error	1 byte	Contains either an error or a 'command pending' flag if the request is not completed. Do not poll this field for request completion (see scb_done). scb_error = FFH Reserved. Currently command pending for asynchronous notification. 0 = Success non-zero = Error
scb_vcid	1 byte	Contains the virtual circuit identifier. The Call and Listen functions return this value, which must be filled in prior to issuing Send or Receive requests. The virtual circuit identifier, which is in the range 1 to 31, is used by Send and Receive to identify the virtual circuit to use for sending or receiving.
scb_num	1 byte	Used as a part of datagram support. The Add Name function returns the value, which is used by Send Datagram, Receive Datagram, and Send Broadcast. The VAXmate workstation implementation of the session level always sets scb_num to one. All other values are reserved for future use. It has a potential range of 1 to 254.

Table 18-6 Session Control Block Fields (cont.)

Field Name	Size	Description
scb_baddr	4 bytes	Contains the address of the data to be transferred. This field is in double-word format (DD segment:offset).
scb_length	2 bytes	Contains the length, in bytes, of the data to be transferred.
scb_rname	16 bytes	Contains the remote network name, which must be upper case. All 16 bytes must be used. Session supports a maximum of 72 simultaneous network names, including the name of this workstation.
scb_lname	16 bytes	Contains the local network name, which must be upper case. All 16 bytes must be used. The session level supports a maximum of 72 simultaneous network names.
scb_rto	1 byte	Contains the receive timeout for a virtual circuit. The value represents the number of 500 ms. ticks, and is set by a Call or Listen function and holds for the life of the circuit. A zero results in the use of the transport level default. In this implementation, the default is zero ticks.
scb_sto	1 byte	Contains the send timeout for a virtual circuit. The value represents the number of 500 ms. ticks, and is set by a Call or Listen function and holds for the life of the circuit. A zero results in the use of the transport level default. In this implementation, the default is zero ticks.
scb_async	4 bytes	Contains the address of the Asynchronous Notification Routine, ANR. This field is in double-word format (DD segment:offset). If this field is zero and the asynchronous form of the function is used, then scb_done must be polled to determine completion. The ANR routine is called at interrupt time with interrupts masked off. Upon completion of the ANR, the application must issue an IRET instruction.
scb_res	1 byte	Is a reserved field, which must be initialized to 0.
scb_done	1 byte	Is a status field filled in by session. The value 0FFH means the function is not yet complete. Another value indicates completion.
scb_res2	14 bytes	Is a reserved field, which must be initialized to 0.

DIGITAL-Specific Session Control Block

The VAXmate workstation software provides a set of DIGITAL-specific extensions to the session level interface for manipulating node names and addresses. The INT 2A is issued with the register AH = DCH. Each request includes a DIGITAL Session Control Block, DSCB, pointed at by the ES:BX register pair.

The DIGITAL session level functions are synchronous. There is no asynchronous support. The DIGITAL Session Control Block, DSCB, structure is described below:

```
struct   dscb
{
   unsigned char    dscb_cmd;       /* function number */
   unsigned char    dscb_err;       /* return code */
   unsigned char    dscb_index;     /* index number of node entry */
   unsigned char    dscb_name[16];/* node name padded with spaces */
   unsigned int     dscb_num;       /* node number */
};
```

Table 18-7 describes the DIGITAL session control block fields.

Table 18-7 DIGTIAL Session Control Block Fields

Field Name	Size	Description
dscb_cmd	1 byte	Holds the function code for the requested DIGITAL-specific service.
dscb_err	1 byte	Contains the status of the completed request.
dscb_index	1 byte	Is the index of the node name into the internal name table maintained by session. This index is used in certain services to allow the application to cycle through all the known node names. Index values are in the range zero to (number of names)-(1). The table can contain a maximum of 72 names. The first name in the table is the name of this workstation.
dscb_name	16 bytes	Is a 16-byte node name. All 16 bytes are used for the name. For names shorter than 16 bytes, the name is padded with space characters in the high order bytes.
dscb_num	2 bytes	Is the DECnet area number and node number associated with the name found in dscb_name. The format of this field is: area number = bit 15 through bit 10. node number = bit 9 through 0.

Synchronous Requests

For synchronous requests, application execution is suspended until all network Activity associated with the request is completed.

The flow of control for a typical synchronous request is:

1. The application makes an INT 2A synchronous request.
2. Control passes to the session-level software.
3. If appropriate for this request, a series of calls are made to the DECnet-DOS transport layer.

   When appropriate, session makes both synchronous and asynchronous calls to the transport software. Depending on the request made by the application to session, any mix of synchronous and asynchronous requests can be made to the transport software.
4. Upon completion of all transport functions, the SCB is updated and scb_done is set.
5. Control is returned to the application.

Asynchronous Requests

The asynchronous form of a command is specified by setting the high-order bit of the command function code. Software issuing an asynchronous request regains control immediately after the request is issued. In most cases the request will be satisfied at a later time. To determine completion of the request, the application must specify an Asynchronous Notification Routine as a part of the request or poll the scb_done field.

The session level software only invokes an ANR for requests involving a non-zero scb_async field. A value of zero in the scb_async field means the application must poll the the scb_done field.

While an asynchronous request is outstanding, the SCB for that request must remain unchanged. The session level software uses the SCB for control and status reporting.

The flow of control for a typical asynchronous request is:

1. The application makes an INT 2A asynchronous request.
2. Control passes to the session level software.
3. If appropriate for this request, a series of calls are made to the DECnet-DOS transport layer.

   When appropriate, session makes both synchronous and asynchronous calls to the transport software. Depending on the request made by the application to session, any mix of synchronous and asynchronous requests can be made to the transport software.

4. Control is returned to the application.
5. Upon completion of all asynchronous transport functions, the SCB is updated, scb_done is set, and if specified, the applications ANR is invoked.

Asynchronous Notification Routine

The Asynchronous Notification Routine, ANR, is specified as a part of the Session Control Block for each request. The SCB field scb_async contains a zero value or a valid pointer to the ANR. An ANR is invoked only upon completion of asynchronous session requests.

When an asynchronous request completes, the session software starts execution of the ANR, which is entered with interrupts off. Because the asynchronous notification may have interrupted processing by MS-DOS, the ANR cannot issue any MS-DOS calls. It is recommended the ANR not enable interrupts, and complete its processing quickly. The ANR is exited by executing an IRET instruction.

Network Addressing

The standard MS-Network supports a 20-byte network address. The session software provides the transport level with the network address of the messages destination. The transport level uses this address to route the message to its destination.

The VAXmate workstation implementation of the MS-Network session level uses DECnet addresses in place of the standard 20-byte addresses. The session software maps a network name into a DECnet node address, which consists of an area number and node number.

For more information on DECnet node numbers, addressing in a DECnet network, and user control over node names and numbers, refer to:

- The VAXmate system administration documentation
- *DECnet-DOS Programmers Reference Manual*
- *DECnet-DOS User's Guide*
- *DNA General Description* and other manuals in the DIGITAL Network Architecture series. For the titles of this series, see the "Introduction" in this chapter.

Session Level Services

The VAXmate session level interface accessed through INT 2A provides two distinct sets of services. The first is the MS-Network compatible functions specified with AH = 01H. The second is a set of DIGITAL-specific functions specified with AH = DCH.

Table 18-8 lists the services offered by the session level interface. Table 18-9 lists the DIGITAL-specific session level extensions.

Table 18-8 Interrupt 2A: MS-Network Compatible Services

Synchronous	Asynchronous	Service
10H	90H	CALL
11H	91H	LISTEN
12H	92H	HANGUP
14H	94H	SEND
15H	95H	RECEIVE
16H	96H	RECEIVE ANY
17H	97H	SEND DOUBLE
20H	A0H	SEND DATAGRAM
21H	A1H	RECEIVE DATAGRAM
22H	A2H	SEND BROADCAST
23H	A3H	RECEIVE BROADCAST
30H	B0H	ADD NAME
31H	B1H	DELETE NAME
32H		RESET
33H	B3H	STATUS
34H	B4H	NAME STATUS
35H		CANCEL

All other function codes are reserved for future use.

Table 18-9 Interrupt 2A: DIGITAL-Specific Session Extensions

Function Code	Service	Description
00H	decfunccheck	extension check
01H	decfuncadd	add node entry
02H	decfuncdelnum	delete node by number
03H	decfuncdelname	delete node by name
04H	decfuncreadnum	read node by number
05H	decfuncreadname	read node by name
06H	decfuncreadindex	read node by index
07H	decfuncdelall	delete all nodes

All other function codes are reserved for future use.

MS-Network Compatible Session Level Services

An MS-Network session level access is performed by setting register AH = 1, pointing ES:BX at a session control block (SCB) and doing an INT 2AH.

When control is returned to the application from the session level, the AH and AL registers contain the return status. Status is also returned in the appropriate field of the SCB.

For a synchronous request:

Upon Return	AX = non-zero	Error has occurred. AH = 01H AL = error code scb_error = error code
	AX = 00H	Success scb_error = 0

For an asynchronous request:

Upon Return	AX = non-zero	Error has occurred. AH = 01H AL = error code scb_error = error code
	AX = 00H	Function successfully started. SCB fields pending completion of request. Upon completion of request, SCB fields updated, and error code, if any, in scb_error.

MS-Network Session Level Return Codes

Table 18-10 lists the possible MS-Network session level interface error codes returned in scb_error, and the service requests that can generate these errors.

Table 18-10 Error Codes Returned by Session

Error Code	Description	Service Request
00H	Success	All services
01H	Illegal buffer length	SEND RECEIVE STATUS NAME STATUS SEND DATAGRAM RECEIVE DATAGRAM RECEIVE ANY SEND BROADCAST RECEIVE BROADCAST
03H	Illegal command	All services
05H	Command timed out	Any asynchronous command
06H	Message incomplete, issue another command	RECEIVE ANY RECEIVE STATUS
08H	Session number out of range	SEND RECEIVE HANGUP
09H	No resource available	All services
0AH	Session closed	RECEIVE SEND RECEIVE ANY HANGUP
0BH	Command canceled	STATUS ADD NAME DELETE NAME CALL LISTEN SEND SEND DOUBLE RECEIVE RECEIVE ANY SEND DATAGRAM RECEIVE DATAGRAM SEND BROADCAST RECEIVE BROADCAST

Table 18-10 Error Codes Returned by Session (cont.)

Error Code	Description	Service Request
0DH	Duplicate name	ADD NAME
0EH	Name table full	ADD NAME
11H	Local session table full	CALL LISTEN
12H	Session Open rejected	CALL
13H	Illegal name number	NAME STATUS RECEIVE DATAGRAM SEND DATAGRAM SEND BROADCAST RECEIVE BROADCAST
14H	No call name	CALL
15H	Name not found or no valid name	DELETE NAME
16H	Name in use	ADD NAME
18H	Session ended abnormally	SEND RECEIVE HANGUP
19H	Name conflict detected	ADD NAME
21H	Interface busy	All services
22H	Too many commands outstanding, retry later	All services
23H	Reserved	All services
24H	Command completed while cancel occurring	CANCEL
26H	Command not valid to cancel	CANCEL
31H	Internal error, can result from an invalid address file	CALL LISTEN SEND DATAGRAM RECEIVE DATAGRAM

Table 18-10 Error Codes Returned by Session (cont.)

Error Code	Description	Service Request
32H	Transport not installed	STATUS CANCEL LISTEN CALL SEND SEND DOUBLE RECEIVE RECEIVE ANY HANGUP SEND DATAGRAM RECEIVE DATAGRAM SEND BROADCAST RECEIVE BROADCAST
4XH	Network Error X may have any value	All services
FFH	Asynchronous command is not yet finished	All services

The DECnet-DOS transport can generate error codes that do not correspond to the session level errors described in this list. These errors and their codes are described in the *DECnet-DOS Programmer's Reference Manual.*

Transport error codes that cannot be directly mapped to a session level error code, are passed directly through the interface to the application. Such transport error codes are mapped into the following format.

DECnet-DOS error code + the number 80H

To determine the error represented by the error code being returned from the DECnet-DOS transport, subtract 80H from the number returned in scb_error. The error description can then be found in the *DECnet-DOS Programmer's Reference Manual.*

Function 00H and Function B800H: Check for Presence of MS-Network Session

Two functions are provided to support this request. The first is the recommended function accessed through the INT 2A interface. The second function is accessed through the INT 2F interface and is present only for compatibility with industry-standard versions of the MS-Network session interface.

The INT 2A function allows an application to determine the presence of the Session software.

The check is performed by setting register AH = 0 and doing an INT 2A. If the network software is installed, upon return AH will have a non-zero value.

Parameters

AH =	00H
ES:BX =	not applicable

Returns

AH =	non-zero	Session is present
	00H	Session not present

The INT 2F function is provided for compatibility with other vendor's versions of the MS-Network session level interface.

The check is performed by setting register AX = B800H and doing an INT 2F. If the network software is installed, upon return AL will have a value of 1.

Parameters

AX =	B800H
ES:BX =	not applicable

Returns

AL =	01H	Session is present
	00H	Session not present
BX =	08H	Always

Function 35H: Cancel (synchronous)

This service allows an application to cancel a pending request. The request to cancel is identified by pointing at its SCB.

Data may be lost when a command is canceled. If the canceled command is a receive, then data is lost only for that command, and the virtual circuit remains active and usable.

Canceling a request is a form of completing the request. The canceled SCB will be updated. A request that normally invoked an Asynchronous Notification Routine will cause that routine to be invoked.

The following commands can be canceled.

- Listen
- Receive
- Receive Any
- Receive Datagram
- Receive Broadcast

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	35H
	scb_error =	00H
	scb_baddr =	far pointer to SCB to cancel

Returns

scb_error =	00H	Success
	21H	Interface busy
	22H	Too many outstanding commands, retry later
	24H	Command completed during cancel operation
	26H	Command not valid to cancel
	32H	Transport not installed
	4xH	Network error, x may be any value

Function 32H: Reset (synchronous)

Reset the session software. All data, virtual circuits, and status are lost. This resets the entire session level, not just one virtual circuit. The session software is not completely reset to its initial state on startup.

The parameters scb_vcid and scb_num are reserved for future use. They should be set to zero by the application. If they are zero, then default values will be used in future session implementations.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	32H
	scb_error =	00H
	scb_vcid =	RESERVED, Must Be zero
	scb_num =	RESERVED, Must Be Zero

Returns

scb_error =	00H	Success
	4xH	Network error, x may be any value

Function 33H: Status (synchronous)
Function B3H: Status (asynchronous)

Returns overall transport status information for this VAXmate workstation. Information is loaded into the buffer supplied by the application and pointed to by scb_baddr.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	33H Synchronous B3H Asynchronous
	scb_error =	00H
	scb_length =	Length, in bytes, of the data to be transferred
	scb_rname =	RESERVED
	scb_baddr =	Far pointer to status buffer
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	05H	Command timed out
	06H	Message incomplete
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value

The data area pointed to by the scb_baddr contains the status buffer as described in table 18-11.

Table 18-11 Session Status Buffer

Field Name	Size (bytes)	Description
SSB_HID1	6	Ethernet address of this workstation, low order byte first
SSB_JMPR1	1	Jumper status: reserved, returns zero
SSB_HRD	1	Hardware status: reserved, returns 128
SSB_SVER	2	Session software version, in BCD format
SSB_DUR	2	Duration of reporting period
SSB_CRC	2	Number of CRC errors

SSB_ALIGN	2	Number of alignment errors

Table 18-11 Session Status Buffer (cont.)

Field Name	Size (bytes)	Description
SSB_COL	2	Number of collisions detected
SSB_ABORT	2	Number of aborted transmissions
SSB_NSENT	4	Number of successfully transmitted packets
SSB_NRECD	4	Number of successfully received packets
SSB_RETRAN	2	Number of retransmissions
SSB_NRSRC	2	Number of times the receiver exhausted its resources
SSB_RES4	8	Reserved
SSB_RES5	2	Reserved
SSB_RES6	2	Reserved
SSB_RES7	2	Maximum number of free command blocks
SSB_RES8	4	Reserved
SSB_RES9	2	Reserved
SSB_RES10	2	Reserved
SSB_RES11	2	Reserved
SSB_MAXMSG	2	Maximum message data size, which is 4096 for this implementation of session
SSB_NNAMES	2	Number of names in the immediately following list, (value is is always 1 for this implementation)
SSB_NAM0	16	Name of this workstation
SSB_NUM0	1	RESERVED for future use
SSB_NAMSTAT0	1	Status of this VAXmate workstation, 4 = active, 5 = inactive
: :		
: :		
SSB_NAMn	16	Name n, not present in this implemenation
SSB_NUMn	1	RESERVED for future use
SSB_NAMSTATn	1	Status of name n, 4 = active, 5 = inactive, not present in this implemenation

SSB__MAXMSG, maximum message data size is the recommended maximum message size for applications that are communicating with VAXmate and VAX/VMS file and print servers. For this implementation, maximum message size is 4096 bytes. However, the session interface can send and receive messages as large as 65536 bytes.

This implementation of the session level only returns SSB__NAM, SSB__NUM and SSB__NAMSTAT status for this workstation. Consequently, the value of SSB__NNAMES always equals 1.

Function 30H: Add Name (synchronous)
Function B0H: Add Name (asynchronous)

This function is provided for compatibility with industry-standard implementations of the MS-Network session level interface, and to facilitate the execution of MS-Network compatible application software. This function does not actually add a name to the name and node tables. To add a name or node number, use the DIGITAL-specific function Add A Node, decfuncadd, described later in this manual.

The name cannot start with the character '*' or 00H or FFH and should not start with the strings "MSNET" or "IBM". For non-VAXmate workstation implementation of the session level, this call makes a name known to the network software.

It is recommended, but not required, that names be unique across a network. The node numbers must be unique across the network for the network is to function properly.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	30H synchronous B0H Asynchronous
	scb_error =	00H
	scb_lname =	Name to be added
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	0DH	Duplicate name in this workstation
	0EH	Name table full
	15H	Name not found or Not a valid name
	16H	Name in use
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	4xH	Network error, x may be any value
scb_num =	reserved for future use	

Function 31H: Delete Name (synchronous)
Function B1H: Delete Name (asynchronous)

This function is provided for compatibility with industry standard implementations of the MS-Network session level interface. It is provided to facilitate the execution of MS-Network compatible application software. This function does not actually delete a name from the name and node tables. To actually delete a name or node number from the network tables use the DIGITAL-specific functions Delete Entry Given Node Number, Delete Entry Given Node Name, or Delete All Node Entries.

For non-VAXmate workstation session level interfaces, this function removes a name from the network. A name may not be deleted if a virtual circuit is in use that has the name as an endpoint.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	31H synchronous B1H asynchronous
	scb_error =	00H
	scb_lname =	name to be deleted
	scb_async =	address of ANR or zero for asynchronous form of request. Not required for synchronous request

Returns

scb_error =	00H	Success
	15H	Name not found or Not a valid name
	16H	Name in use
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	4xH	Network error, x may be any value

Function 34H: Name Status (synchronous)
Function B4H: Name Status (asynchronous)

This command returns information about a specific name and its associated Virtual Circuit.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	34H synchronous B4H asynchronous
	scb_error =	00H
	scb_length =	Length of buffer pointed at by scb_baddr
	scb_baddr =	Far pointer to status buffer
	scb_lname =	Name to return status on
	scb_async =	Address of anr or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	4xH	Network error, x may be any value

The scb_baddr field points to a area of length scb_length which contains the following:

SB_NUM	Name number of name being reported on
SB_NRA	Number of virtual circuits associated with this name
SB_VCN	Number of receive datagram and receive broadcast commands outstanding
SB_NVC	Number of receive any commands outstanding

The remaining information is returned about each virtual circuit (36 bytes each)

SB_I_VCID	1 byte	Virtual circuit id#
SB_I_STATE	1 byte	State of the virtual circuit
SB_I_LNAME	16 byte	Local name
SB_I_RNAME	16 byte	Remote name
SB_I_NRC	1 byte	Number of outstanding receive commands
SB_I_NSC	1 byte	Number of outstanding send commands

SB_I_STATE

Values:
3 = Normal
4 = Hang-up pending

Function 10H: Call (synchronous)
Function 90H: Call (asynchronous)

Call creates a virtual circuit between this node and another node that issued a listen. This node must identify the node it is calling in the scb_rname field. The listening node, which does not identify the calling node, posts a listen and waits to be called. The scb_rname must correspond to a scb_lname in an outstanding listen command on some machine on the MS-Network network.

The node named in the scb_rname field must be in the VAXmate workstation list of known nodes. The node list is updated with entries when the network is started by the user using one of the network management utilities or by the application invoking the DIGITAL-specific add node functions described in this section. If the user uses the DECnet-DOS NCP utility to add a node to the network tables, the user must remember to specify the MS-NET switch as a part of the name and number definition. Otherwise, the node name is not added to the list maintained by the session software.

The fields scb_rto and scb_sto do not take affect until the virtual circuit is established by the DECnet-DOS transport software. If a virtual circuit is not established within approximately one minute, a timeout error occurs.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	10H synchronous 90H Asynchronous
	scb_error =	00H
	scb_rto =	Number of 500 ms time ticks for receive timeout
	scb_sto =	Number of 500 ms time ticks for transmit timeout
	scb_lname =	Name of this node
	scb_rname =	Name of target node
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request

Returns

scb_error =	00H	Success
	05H	request timed out
	06H	Message incomplete, issue another request
	0BH	Command canceled
	14H	No call name
	15H	Name not found or Not a valid name
	16H	Name in use
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value
scb_vcid =	Virtual Circuit ID	

Function 11H: Listen (synchronous)
Function 91H: Listen (asynchronous)

Listen waits for a call from any node that specifically wants to communicate with this node. The listen request completes normally when a call with scb_rname that matches the listen scb_lname, is made somewhere on the network. Upon completion, this node is informed of the name of the caller and the Virtual Circuit ID of the circuit communication can now proceed

The fields scb_rto and scb_sto do not take affect until the virtual circuit is established by the DECnet-DOS transport software.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	11H synchronous 91H asynchronous
	scb_error =	00H
	scb_rto =	Number of 500 ms time ticks for receive timeout
	scb_sto =	Number of 500 ms time ticks for transmit timeout
	scb_lname =	Name of this node
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	09H	No resources available
	0BH	Command canceled
	11H	Local session table full
	15H	Name not found or Not a valid name
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value
scb_vcid =		Virtual Circuit ID of the Virtual Circuit the nodes can now communicate over
scb_rname =		Name of node that issued CALL to this node

Function 12H: Hangup (synchronous)
Function 92H: Hangup (asynchronous)

Hangup ends a virtual circuit. Any pending receive commands are terminated. Any pending send commands will complete before the hangup completes.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	12H synchronous 92H asynchronous
	scb_error =	00H
	scb_vcid =	Virtual Circuit ID of the Virtual Circuit to hang-up
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request

Returns

scb_error =	00H	Success
	08H	Invalid Virtual Circuit ID
	0AH	Session closed
	18H	Session ended abnormally
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Not installed
	4xH	Network error, x may be any value

Function 14H: Send (synchronous)
Function 94H: Send (asynchronous)

Send data on a virtual circuit. More than one send command can be outstanding. The commands are processed in FIFO order. Each send may specify a message length of 0 to 65536 bytes. The recommended maximum message length is the value of the field SSB_MAXMSG that is returned by the Status function. The length specified by SSB_MAXMSSG is guaranteed to be accepted by DIGITAL-developed file and print servers.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	14H synchronous 94H asynchronous
	scb_error =	00H
	scb_vcid =	Virtual Circuit ID of the Virtual Circuit to send the data over
	scb_length =	Length of buffer pointed at by scb_baddr. 0 to 65535 bytes
	scb_baddr =	Address of buffer to send
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	05H	Command timed out
	08H	Session number out of range
	0AH	Session closed
	0BH	Command canceled
	18H	Session ended abnormally
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value

Function 17H: Send Double (synchronous)
Function 97H: Send Double (asynchronous)

Send two buffers of data on a virtual circuit. This has the effect of concatenating successive buffers into a single message on the virtual circuit.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	17H synchronous 97H asynchronous
	scb_error =	00H
	scb_vcid =	Virtual Circuit ID of the Virtual Circuit to send the data over
	scb_length =	Length of first buffer pointed at by scb_baddr, 0 to 65535 bytes
	scb_baddr =	Address of first buffer to send.
	scb_rname =	The length and address of the second buffer to send Length = first 2 bytes, low order byte first Address = next four bytes Offset = first two bytes Segment= next two bytes
	scb_async =	Address of ANR or zero for asynchronous form of request Not required for synchronous request

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	05H	Command timed out
	08H	Session number out of range
	0AH	Session closed
	0BH	Command canceled
	18H	Session ended abnormally
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value

Function 15H: Receive (synchronous)
Function 95H: Receive (asynchronous)

This service allows you to receive data on a virtual circuit. If an application has receive and receive any requests outstanding at the same time, the received messages are posted in the order of receive followed by Receive Any. This allows a specific receive to take precedence over a general receive.

If multiple Receive commands are outstanding, they are processed in FIFO order. When the Receive completes, scb_length is updated to the actual message length. If the message transmitted is larger than the available buffer space in scb_baddr, then the message-incomplete error is returned in scb_error. The application can retrieve the next portion of the message by issuing another Receive request.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	15h synchronous 95h asynchronous
	scb_error =	00h
	scb_vcid =	Virtual Circuit ID of the Virtual Circuit to receive the data over
	scb_length =	Length of the buffer pointed at by scb_baddr
	scb_baddr =	Address of buffer to receive into.
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	05H	Command timed out
	06H	Message incomplete, issue another receive request
	08H	Incorrect Virtual Circuit ID
	0AH	Session closed
	0BH	Command canceled
	18H	Session ended abnormally
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value
scb_length =		The actual length of the message received

Function 16H: Receive Any (synchronous)
Function 96H: Receive Any (asynchronous)

Receive Any receives the next message on any virtual circuit associated with this VAXmate node. If an application has Receive and Receive Any requests outstanding at the same time, the received messages are posted in the order of Receive followed by Receive Any. This allows a specific receive to take precedence over a general receive.

If multiple Receive Any commands are outstanding, they are processed in FIFO order.

When the Receive Any completes, scb_length is updated to the actual message length. If the message transmitted is larger than the available buffer space in scb_baddr, then the message-incomplete error is returned in scb_error. The application can retrieve the next portion of the message by issuing another Receive or Receive Any request.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	16H synchronous 96H asynchronous
	scb_error =	00H
	scb_num =	RESERVED: must be one for future compatibility
	scb_length =	Length of the buffer pointed at by scb_baddr
	scb_baddr =	Address of buffer to receive into.
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	05H	Command timed out
	06H	Message incomplete, issue another receive request.
	0AH	Session closed
	0BH	Command canceled
	13H	Illegal scb_num, RESERVED for future use.
	18H	Session ended abnormally
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value
scb_vcid =		Virtual Circuit ID of the Virtual Circuit data was received over.
scb_length =		The actual length of the message received.
scb_rname =		Name of remote node that sent the message.

Datagram Commands

Datagrams are short packets of data sent to one or more nodes. The network does not guarantee the delivery of a datagram. Datagram lengths must range from 46 to 512 bytes. Because the session software does not pad a datagram message to the minimum Ethernet packet length, the minimum length packet you can transmit is 46 bytes.

For a node to receive a datagram, a datagram receive must be outstanding at the time a datagram is sent.

The actual order in which datagrams are received is not guaranteed to be the same order in which they were transmitted.

Function 20H: Send Datagram (synchronous)
Function A0H: Send Datagram (asynchronous)

Send a datagram to a specific node. The network does not verify that everyone (or anyone) actually received the datagram. The ordering of datagrams is not guaranteed.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	20H synchronous A0H asynchronous
	scb_error =	00H
	scb_num =	RESERVED: must be one for future compatibility
	scb_length =	Length in bytes of the buffer pointed at by scb_baddr in the range 46 to 512
	scb_baddr =	Address of buffer to transmit from
	scb_rname =	Name of remote node to send datagram to
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	0BH	Command canceled
	14H	No Call name
	15H	Name not found or not a valid name.
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value

Function 21H: Receive Datagram (synchronous)
Function A1H: Receive Datagram (asynchronous)

When the Receive Datagram completes, scb_length is updated to the actual datagram length. If the datagram transmitted is larger than the available buffer space in scb_baddr, then the message-incomplete error is returned in scb_error. The application can retrieve the next portion of the datagram by issuing another Receive Datagram request.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	21H synchronous A1H asynchronous
	scb_error =	00H
	scb_num =	RESERVED: must be one for future compatibility.
	scb_length =	Length in bytes of the buffer pointed at by scb_baddr
	scb_baddr =	Address of buffer to receive into
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	06H	Message incomplete, issue another receive request
	0BH	Command canceled
	13H	RESERVED Illegal name number
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value
scb_length =		Actual length of datagram received.
scb_rname =		Name of remote node datagram was received from.

Function 22H: Send Broadcast (synchronous)
Function A2H: Send Broadcast (asynchronous)

Send Broadcast sends a broadcast datagram. A datagram is sent to all machines on the local network that have an outstanding Receive Broadcast command. If the machine performing the send also has an outstanding Receive Broadcast command, it will receive its own datagram.

As with the Send Datagram request, the network does not verify that everyone (or anyone) actually received the broadcast datagram. The ordering of broadcast datagrams is not guaranteed.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	22H synchronous A2H Asynchronous form = A2H
	scb_error =	00H
	scb_num =	RESERVED: must be one for future compatibility
	scb_length =	Length in bytes of the buffer pointed at by scb_baddr in the range 46 to 512
	scb_baddr =	Address of buffer to transmit from
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	0BH	Command canceled
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value

Function 23H: Receive Broadcast (synchronous)
Function A3H: Receive Broadcast (asynchronous)

Receive Broadcast requests the receive of a datagram sent using the send broadcast command. If a receive broadcast command is not outstanding at the time a broadcast datagram is sent, then it will not be received.

Parameters

AH =	01H	
ES:BX =	Far pointer to SCB	
	scb_command =	23H synchronous A3H asynchronous
	scb_error =	00H
	scb_length =	Length in bytes of the buffer pointed at by scb_baddr
	scb_baddr =	Address of buffer to receive into
	scb_async =	Address of ANR or zero for asynchronous form of request. Not required for synchronous request.

Returns

scb_error =	00H	Success
	01H	Illegal buffer length
	06H	Message incomplete, issue another receive request
	0BH	Command canceled
	13H	RESERVED Illegal name number
	19H	Name conflict detected
	21H	Interface busy
	22H	Too many outstanding requests, retry later
	32H	Transport not installed
	4xH	Network error, x may be any value
scb_length =		Actual length of datagram received.
scb_rname =		Name of remote node datagram was received from.

DIGITAL-Specific Session Level Services

The DIGITAL-specific functions support the manipulation of network name and node number entries that are found in the memory resident tables.

The memory resident tables are called the volatile database. The permanent database consists of the disk resident node tables. The permanent database is loaded into memory at network startup time. To update the permanent database, the VAXmate workstation user should use one of the network management utilities described in the VAXmate system administration documentation.

The DIGITAL-specific session level functions are:

- Digital Function Check
- Add an entry into node table
- Delete entry by node number
- Delete entry by node name
- Read entry by node number
- Read entry by node name
- Read entry by index into the table (0-71)
- Delete all entries

A DIGITAL-specific session request is performed by setting register AH = DCH, resetting dscb_err to zero, pointing ES:BX to the address of DIGITAL Session Control Block, and doing an INT 2AH function call.

The values returned in dscb_err are:

00 Function completed successfully
01 Table full, cannot add another entry
02 Duplicate name, node name is currently used by another entry
03 Duplicate number, node number is currently used by another entry
04 No entry with given node name
05 No entry with given node number
06 No entry with given index
07 Index given is out of range
08 Illegal function number
09 Out of resource, currently there is no stack space available to perform the function, try again later
0A Cannot delete own node entry

Function 00H: DIGITAL Function Check (decfunccheck)

Function check is used to confirm the presence of the DIGITAL-developed session level module. In addition, this function returns status on the availability of internal resources for supporting further network services. The far pointer in the field dscb_name points to a string containing path and file name of the DECNODE.DAT permanent database file.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	00H
	dscb_err =	00H
	dscb_name =	first double word contains far pointer to DECNODE.DAT path string.

Returns

dscb_err =	00H	Success
	09H	Out of resource

Function 01H: Add a Node (decfuncadd)

This function adds a node name and its node number to the memory resident list of nodes known to the session software. A node must appear in this list before any communication with that node is possible. The node names and node numbers must be unique within this VAXmate workstation.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	01H
	dscb_err =	00H
	dscb_name =	16 byte node name padded with spaces
	dscb_num =	node number

Returns

dscb_err =	00H	Success
	01H	Table full
	02H	Duplicate node name
	03H	Duplicate node number
	09H	Out of resource

Function 02H: Delete Entry Given the Node Number (decfuncdelnum)

This service removes a node name and number from the memory resident list of known nodes. The node name and node number of this workstation cannot be deleted with this function. To remove the node name and node number of this workstation, the Delete All Node Entries function must be used.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	02H
	dscb_err =	00H
	dscb_num =	node number

Returns

dscb_err =	00H	Success
	05H	Node number not found
	09H	Out of resource
	0AH	Cannot delete own node entry

Function 03H: Delete Entry Given Node Name (decfuncdelname)

This service removes a node name and number from the memory resident list of known nodes. The node name and node number of this workstation cannot be deleted with this function. To remove the node name and node number of this workstation, the Delete All Node Entries function must be used.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	03H
	dscb_err =	00H
	dscb_name =	16 byte node name padded in high-order bytes with spaces

Returns

dscb_err =	00H	Success
	04H	Node name not found
	09H	Out of resource
	0AH	Cannot delete own node entry

Function 04H: Read Node Entry Given Node Number (decfuncreadnum)

This service allows an application to use the node number to determine the node name and its position in the memory resident list of known nodes.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	04H
	dscb_err =	00H
	dscb_num =	node number

Returns

dscb_err =	00H	Success
	05H	No entry with given node number
	09H	Out of resource
dscb_index	Position in internal node table	
dscb_name	16-byte node name padded in the high-order bytes with spaces	

Function 05H: Read Node Entry Given Node Name (decfuncreadname)

This service allows an application to use the node name to determine the node number and its position in the memory resident list of known nodes.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	05H
	dscb_err =	00H
	dscb_name =	16-byte node name padded in the high order bytes with spaces

Returns

dscb_err =	00H	Success
	04H	No entry with given node name
	09H	Out of resource
dscb_index =		Position in internal node table
dscb_num =		Node number

Function 06H: Read Node Entry Given Index (decfuncreadindex)

This services returns the node name and node number for a given index position in the memory resident list of nodes.

The value dscb_index ranges from 0 to 71. Seventy-two is the maximum number of node name and number pairs contained in the session internal database.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	06H
	dscb_err =	00H
	dscb_index =	Position in internal node table

Returns

dscb_err	00H	Success
	06H	No entry at the given index
	07H	Index out of range
	09H	Out of resource
dscb_name =		16 byte node name padded in the high order bytes with spaces
dscb_num =		Node number

Function 07H: Delete All Node Entries (decfuncdelall)

This service deletes all entries in the memory resident node list of the session level. Note that the node name and node number of this workstation will also be deleted.

Parameters

AH =	DCH	
ES:BX =	Far pointer to DSCB	
	dscb_cmd =	07H
	dscb_err =	00H

Returns

dscb_err =	00H	Success
	09H	Out of resource

Server Message Block (SMB) Protocol

The Server Message Block (SMB) protocol allows the VAXmate workstations running the MS-DOS operating system and the VAXmate network software to access and share files stored on a server.

Functions and data are passed between a server and a workstation with a Server Message Block (SMB). The SMB data structure and functions are described in:

- *Server/Redirector File Sharing Protocol*
  Microsoft Corporation and Intel Corporation
  Intel Part Number 136329-001
- *IBM Personal Computer Seminar Proceedings,*
  Volume 2, Number 8, October 1984
  IBM Corportation

IMPORTANT
DIGITAL's implementation of the SMB protocol is based on the guidelines in the above documents and other applicable industry standards. DIGITAL is not responsible for inaccuracies or errors in those documents. DIGITAL's implementation of the SMB protocol is subject to change according to changes in industry standards.

The SMB protocol allows for extended functions, which are identified by the function code FFH. The actual function code is stored in the reserved field SMB_REH.

DIGITAL's implementation of the SMB protocol includes one extended SMB function, the Get Current Date and Time function, which is described in the next section.

Extended Function D0H: Get Current Date and Time

Parameters

None

Returns

10 signed 16-bit word parameters. The return parameters are:

- Year
- Month (for example, January = 1)
- Day of month
- Day of week (for example, Sunday = 0)
- Hour (0 -24)
- Minutes (0-59)
- Seconds (0-59)
- Milliseconds (0-999)
- Timezone in minutes west (or east) of Greenwich (for example, Eastern = 300, Switzerland = -60)
- Current local daylight savings correction, in minutes (usually +60 or -60)

This function returns the local date and time. The workstation must adjust the time by the returned time zone offset to produce coordintated universal time, UTC, or Greenwich mean time, (GMT). If the server and the workstation are in different time zones, note that the returned time zone is that of the server.

The following C language definitions describe the date and time information returned by the server:

```
#define date_year       smb_vwv[0]
#define date_month      smb_vwv[1]
#define date_day        smb_vwv[2]
#define date_week       smb_vwv[3]
#define date_hour       smb_vwv[4]
#define date_minute     smb_vwv[5]
#define date_second     smb_vwv[6]
#define date_mills      smb_vwv[7]
#define date_zone       smb_vwv[8]
#define date_savings    smb_vwv[9]
```

Appendix A
Support Code for Examples

This appendix describes several subroutines and include files that support the program examples, but are not specific to a particular hardware example.

File: SUPPORT.ASM

This file contains assembly language subroutines that could not be written in the C programming language.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; declare some C compiler compatible segment types                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
_TEXT   SEGMENT  BYTE PUBLIC 'CODE'
_TEXT   ENDS
CONST   SEGMENT  WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
DGROUP  GROUP CONST, _BSS, _DATA
        ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP

EXTRN _rtc_int_hand:NEAR
EXTRN _kyb_int_hand:NEAR
EXTRN _mouse_int:NEAR
EXTRN _fdc_int_hand:NEAR
EXTRN _hdc_int_hand:NEAR
EXTRN _com1_int:NEAR
EXTRN _modem_int:NEAR
EXTRN _printer_int:NEAR
```

```
_TEXT SEGMENT

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; At interrupt time, the current data segment is unknown. The following ;
; statement provides storage for the interrupt time data  segment.       ;
; Later, this storage is initialized to the value of the interrupt time ;
; data segment.                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

it_ds:  DW      0                       ; place to store data segment


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; _ini_it_ds  - C callable routine to store the interrupt time data     ;
;               segment so that the correct data segment can be used at ;
;               interrupt time                                           ;
;                                                                        ;
;       parameters: ds register                                          ;
;             uses: cs:it_ds (1 word of R/W storage in cs segment)      ;
;          returns: nothing                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     PUBLIC _ini_it_ds
_ini_it_ds  PROC NEAR
    mov     word ptr cs:it_ds, ds       ; save it for later
    ret
_ini_it_ds  ENDP
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The following routines provide entry points for each of the hardware  ;
; interrupt vectors. At the entry point, the ax register is saved and a ;
; pointer to the interrupt handler is loaded into ax. Then a jump is    ;
; made to comhand. The comhand routine saves additional registers,      ;
; calls the interrupt handler through ax, and on return from the        ;
; handler, it restores the registers including ax.                      ;
;                                                                       ;
;       parameters: none                                                ;
;             uses: none   (see comhand, it restores the ax register)   ;
;          returns: nothing                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    PUBLIC  _hdw_int_08
_hdw_int_08 PROC NEAR                   ; re-vector INT 08H
    iret
_hdw_int_08 ENDP

    PUBLIC  _hdw_int_09
_hdw_int_09 PROC NEAR                   ; re-vector INT 09H
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_kyb_int_hand
    jmp     short comhand
_hdw_int_09 ENDP

    PUBLIC  _hdw_int_0a
_hdw_int_0a PROC NEAR                   ; re-vector INT 0AH
    iret
_hdw_int_0a ENDP

    PUBLIC  _hdw_int_0b
_hdw_int_0b PROC NEAR                   ; re-vector INT 0BH
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_modem_int
    jmp     short comhand
_hdw_int_0b ENDP

    PUBLIC  _hdw_int_0c
_hdw_int_0c PROC NEAR                   ; re-vector INT 0CH
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_com1_int
    jmp     short comhand
_hdw_int_0c ENDP
```

```
    PUBLIC  _hdw_int_0d
_hdw_int_0d PROC NEAR                   ; re-vector INT 0DH
    iret
_hdw_int_0d ENDP

    PUBLIC  _hdw_int_0e
_hdw_int_0e PROC NEAR                   ; re-vector INT 0EH
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_fdc_int_hand
    jmp     short comhand
_hdw_int_0e ENDP

    PUBLIC  _hdw_int_0f
_hdw_int_0f PROC NEAR                   ; re-vector INT 0FH
    iret
_hdw_int_0f ENDP

    PUBLIC  _hdw_int_70
_hdw_int_70 PROC NEAR                   ; re-vector INT 70H
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_rtc_int_hand
    jmp     short comhand
_hdw_int_70 ENDP

    PUBLIC  _hdw_int_71
_hdw_int_71 PROC NEAR                   ; re-vector INT 71H
    iret
_hdw_int_71 ENDP

    PUBLIC  _hdw_int_72
_hdw_int_72 PROC NEAR                   ; re-vector INT 72H
    iret
_hdw_int_72 ENDP

    PUBLIC  _hdw_int_73
_hdw_int_73 PROC NEAR                   ; re-vector INT 73H
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_printer_int
    jmp     short comhand
_hdw_int_73 ENDP
```

```
    PUBLIC  _hdw_int_74
_hdw_int_74 PROC NEAR                   ; re-vector INT 74H
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_mouse_int
    jmp     short comhand
_hdw_int_74 ENDP

    PUBLIC  _hdw_int_75
_hdw_int_75 PROC NEAR                   ; re-vector INT 75H
    iret
_hdw_int_75 ENDP

    PUBLIC  _hdw_int_76
_hdw_int_76 PROC NEAR                   ; re-vector INT 76H
    push    ax                          ; save ax
    mov     ax, offset _TEXT:_hdc_int_hand
    jmp     short comhand
_hdw_int_76 ENDP

    PUBLIC  _hdw_int_77
_hdw_int_77 PROC NEAR                   ; re-vector INT 77H
    iret
_hdw_int_77 ENDP
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The following routine is a common intermediate interrupt handler.          ;
; It saves and restores volatile registers and calls the handler.            ;
;                                                                             ;
;       parameters: ax = pointer to interrupt handler                         ;
;             uses: none (ax was previously saved and is restored here)       ;
;          returns: nothing                                                   ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    PUBLIC  comhand
comhand     PROC NEAR                       ; common handler
    push    bx                              ; save bx
    push    cx                              ; save cx
    push    dx                              ; save dx
    push    es                              ; save es
    push    ds                              ; save ds
    mov     ds, word ptr cs:it_ds           ; retrieve IT data segment
    call    ax                              ; call the C function
    pop     ds                              ; restore ds
    pop     es                              ; restore es
    pop     dx                              ; restore dx
    pop     cx                              ; restore cx
    pop     bx                              ; restore bx
    pop     ax                              ; restore ax
    iret                                    ; return from interrupt
comhand     ENDP
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The following routine is a C compatible function that does WORD I/O.  ;
;                                                                        ;
;       parameters: stack contains the port and value                    ;
;             uses: dx and ax (which are saved and restored)             ;
;          returns: nothing                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    PUBLIC  _outw

_outw       PROC NEAR

    push    bp                          ; save bp
    mov     bp,sp                       ; set up frame pointer
    push    dx                          ; save dx
    push    ax                          ; save ax

    mov     dx, WORD PTR [bp+4]         ; C port parameter
    mov     ax, WORD PTR [bp+6]         ; C word value
    out     dx, ax                      ; output word to port

    pop     ax                          ; restore ax
    pop     dx                          ; restore dx
    pop     bp                          ; restore bp
    ret

_outw   ENDP
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; _int_off - C callable routine to disable CPU interrupt                 ;
;                                                                        ;
;       parameters: none                                                 ;
;             uses: ax                                                   ;
;          returns: state of IF flag when this routine was called        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     PUBLIC _int_off
_int_off PROC NEAR

    pushf                                ; push flags
    pop     ax                           ; pop flags to ax
    and     ax,0200H                     ; isolate IF bit
    cli                                  ; interrupts off
    ret                                  ; return IF state to caller

_int_off ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; _int_on - C callable routine to enable CPU interrupt                   ;
;                                                                        ;
;       parameters: [sp + 2] = state of IF when _int_off was called      ;
;             uses: nothing                                              ;
;          returns: nothing                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     PUBLIC _int_on
_int_on PROC NEAR

    push    bp                           ; save frame pointer
    mov     bp,sp                        ; setup new frame pointer
    test    word ptr [bp + 4], 0ffffH    ; test previous IF state
    pop     bp                           ; restore frame pointer
    jz      int_on1                      ; don't set interrupt if zero
    sti                                  ; interrupts on

int_on1:
    ret                                  ; return to caller

_int_on ENDP

_TEXT ENDS
END
```

File: EXAMPLE.H

The include file, EXAMPLE.H, contains the structure declaration, MESSAGE. The MESSAGE structure is used in each of the hardware example's menus. Additionally, this file contains some common constant definitions and some function return types.

```
/**********************************************************************/
/* define MESSAGE structure type                                      */
/**********************************************************************/

typedef struct
{
    int row;                                    /* row location of text */
    int col;                                    /* col location of text */
    char *mess;                               /* the message to display */
} MESSAGE;

/**********************************************************************/
/* define some widely used constants                                  */
/**********************************************************************/
#define ON      -1
#define TRUE    -1
#define OFF     0
#define FALSE   0

/**********************************************************************/
/* declare some return types                                          */
/**********************************************************************/

void int_on();
int  int_off();
int  get_key();
int  rb_out();
int  rb_in();
void init_rb();
int cursor_off();
void shw_date();
void shw_time();
```

File: KYB.H

The include file, KYB.H, defines the names of the LK250 function keys as the value returned by the keyboard interrupt handler. These are the same values returned by the ROM BIOS.

```
/************************************************************************/
/* kyb.h - define function key names as value returned by ROM BIOS     */
/*        Note: keyboard demo returns same VALUES as the ROM BIOS       */
/*              All function keys return NULL byte, then this value     */
/************************************************************************/

#define F1  0x3b                                   /* Function key F1  */
#define F2  0x3c                                   /* Function key F2  */
#define F3  0x3d                                   /* Function key F3  */
#define F4  0x3e                                   /* Function key F4  */
#define F5  0x3f                                   /* Function key F5  */
#define F6  0x40                                   /* Function key F6  */
#define F7  0x41                                   /* Function key F7  */
#define F8  0x42                                   /* Function key F8  */
#define F9  0x43                                   /* Function key F9  */
#define F10 0x44                                   /* Function key F10 */
#define F11 0x8f                                   /* Function key F11 */
#define F12 0x90                                   /* Function key F12 */
#define F13 0x91                                   /* Function key F13 */
#define F14 0x92                                   /* Function key F14 */
#define F15 0x93                                   /* Function key F15 */
#define F16 0x94                                   /* Function key F16 */
#define F17 0x95                                   /* Function key F17 */
#define F18 0x96                                   /* Function key F18 */
#define F19 0x97                                   /* Function key F19 */
#define F20 0x98                                   /* Function key F20 */
```

File: RB.H

The include file, RB.H, declares the ring buffer control structure, RING_BUFF. In the example programs, the ring buffer control structure is used for all serial communications devices.

```
/*********************************************************************/
/* define a ring buffer control structure                            */
/*********************************************************************/

typedef struct
{
  char *pbs;                              /* pointer to buffer start */
  char *pbe;                            /* pointer to buffer end + 1 */
  char *pbh;                               /* pointer to buffer head */
  char *pbt;                               /* pointer to buffer tail */
  int  size;                                          /* buffer size */
  int  count;                         /* # of characters in buffer */
  int  high_lim;                         /* buffer near full limit */
  int  low_lim;                         /* buffer near empty limit */
} RING_BUFF;
```

File: VECTORS.C

The file VECTORS.C contains two C functions, *iv_init* and *iv_rest*. The function, iv_init, initializes the indicated interrupt vector with the address of an example interrupt handler. It does this by saving the current interrupt handler address and installing an interrupt handler address from one of those in the assembly language source file SUPPORT.ASM. The function, iv_rest, restores the previously saved interrupt handler address.

```
/**********************************************************************/
/* declare the external interrupt vector entry points                 */
/**********************************************************************/

extern int hdw_int_70();                              /* real time clock */
extern int hdw_int_71();               /* redirect to int OAH, old irq2 */
extern int hdw_int_72();                         /* Ethernet controller */
extern int hdw_int_73();                         /* serial printer port */
extern int hdw_int_74();                                  /* mouse port */
extern int hdw_int_75();                                 /* 80287 error */
extern int hdw_int_76();                        /* hard disk controller */
extern int hdw_int_77();                                  /* unassigned */
extern int hdw_int_08();                        /* 8254 counter/timer */
extern int hdw_int_09();                                    /* keyboard */
extern int hdw_int_0a();                  /* reserved - slave controller */
extern int hdw_int_0b();              /* optional modem/COM2 serial port */
extern int hdw_int_0c();                            /* COM1 serial port */
extern int hdw_int_0d();                                  /* unassigned */
extern int hdw_int_0e();                   /* diskette drive controller */
extern int hdw_int_0f();             /* optional LPT1 parallel printer port */
```

```
/*********************************************************************/
/* declare storage for old vectors and define an array of new vectors  */
/*********************************************************************/

int (far *old_iv[16])();          /* array of far pointers to functions */
int (far *new_iv[16])() =         /* array of far pointers to functions */
{
    hdw_int_70,
    hdw_int_71,
    hdw_int_72,
    hdw_int_73,
    hdw_int_74,
    hdw_int_75,
    hdw_int_76,
    hdw_int_77,
    hdw_int_08,
    hdw_int_09,
    hdw_int_0a,
    hdw_int_0b,
    hdw_int_0c,
    hdw_int_0d,
    hdw_int_0e,
    hdw_int_0f
};
```

```
/*******************************************************************/
/* iv_init() - Based on the interrupt number, this routine saves the    */
/*             old vector and installs a new vector. It is assumed     */
/*             that the new vector does not link to the old vector.    */
/*******************************************************************/

iv_init(int_num)                          /* interrupt vector initialize */

int int_num;                              /* interrupt number for vector */

{

/* This declaration declares a 32-bit pointer that points to */
/* another 32-bit pointer (the interrupt vector) that points */
/* to a function (the interrupt handler), which returns an */
/* integer value (because a pointer to void is illegal) */

int (far * far *piv)();  /* far pointer to a far pointer to a function */
int i_flag;                                          /* CPU IF state */

  piv = (int (far * far *)())((long)int_num << 2);/* pointer to vector */
  if(int_num > 0x0f) int_num -= 0x70;        /* adust index into array */
  old_iv[int_num] = *piv;                            /* save old vector */
  i_flag = int_off();                        /* no interrupts allowed */
  *piv = new_iv[int_num];                        /* install new vector */
  int_on(i_flag);                                  /* allow interrupts */
}
```

```
/*********************************************************************/
/* iv_rest() - Based on the interrupt number, this routine restores    */
/*             the previously saved vector.                            */
/*********************************************************************/

iv_rest(int_num)                              /* interrupt vector restore */

int int_num;                                /* interrupt number for vector */

{

/* This declaration declares a 32-bit pointer that points to */
/* another 32-bit pointer (the interrupt vector) that points */
/* to a function (the interrupt handler), which returns an */
/* integer value (because a pointer to void is illegal) */

int (far * far *piv)();  /* far pointer to a far pointer to a function */
int i_flag;                                              /* CPU IF state */

  piv = (int (far * far *)())((long)int_num << 2);/* pointer to vector */
  if(int_num > 0x0f) int_num -= 0x70;       /* adjust index into array */
  i_flag = int_off();                         /* no interrupts allowed */
  *piv = old_iv[int_num];                        /* restore old vector */
  int_on(i_flag);                                  /* allow interrupts */
}
```

File: RB.C

The file, RB.C, contains two C functions, *rb_in* and *rb_out*. The function, rb_in, stores characters in a ring buffer. the function, rb_out, retrieves characters from a ring buffer. In the example programs, ring buffers are used for all serial communications devices.

```
#include "rb.h"

/***********************************************************************/
/* rb_out() - get character from ring buffer                          */
/***********************************************************************/

int rb_out(prb, pc)

register RING_BUFF *prb;                    /* pointer to ring buff struct */
char    *pc;                                   /* put retrieved char here */

{

int     intr_flg;

  if(prb->count)                          /* any characters in buffer ? */
  {
    *pc = *prb->pbt++;                     /* get character from buffer */
    if(prb->pbt == prb->pbe)                  /* time to wrap pointer ? */
      prb->pbt = prb->pbs;                   /* wrap to start of buffer */
    intr_flg = int_off();                      /* no interrupts allowed */
    prb->count--;                                    /* decrement count */
    int_on(intr_flg);                             /* allow interruppts */
    if(prb->count == prb->low_lim)              /* buffer near empty ? */
      return(0);                              /* indicate restart flow */
    else return(1);                                   /* normal process */
  }
  else return(-1);                                  /* nothing available */
}
```

```
/*************************************************************************/
/* rb_in() - put charater in to ring buffer                              */
/*************************************************************************/

int rb_in(prb, c)

register RING_BUFF *prb;                  /* pointer to ring buff struct */
char c;                                   /* character to put in buffer */

{
int     intr_flg;

  if(prb->count == prb->size)               /* buffer absolutely full ? */
  {
    return(-1);                                    /* can't do anything */
  }
  else
  {
    *prb->pbh = c;                           /* put character in buffer */
    if(++prb->pbh == prb->pbe)               /* advance ptr, wrap ptr ? */
      prb->pbh = prb->pbs;                   /* wrap to start of buffer */
    intr_flg = int_off();                      /* no interrupts allowed */
    prb->count++;                                    /* increment count */
    int_on(intr_flg);                               /* allow interrupts */
    if(prb->count >= prb->high_lim)             /* buffer almost full ? */
      return(0);                                /* indicate almost full */
    else return(1);                            /* allow more to come in */
  }
}
```

File: DEMO.C

The file, DEMO.C, contains several functions that support the example programs. The primary function, *demo*, displays the main menu and drives all of the example programs.

```
#include "video.h"
#include "kyb.h"
#include "example.h"

char glb_attr = 0x07;                          /* default attribute byte */

/**********************************************************************/
/* disp_str() - at specified location, display a string of text      */
/**********************************************************************/

disp_str(row, col, pc)

int row;                                          /* row of start location */
int col;                                       /* column of start location */
char *pc;            /* pointer to beginning of null terminated string */

{

  while(*pc) disp_t(row, col++, *pc++, glb_attr);  /* display the line */
}

/**********************************************************************/
/* disp_menu() - display a menu from an array of MESSAGE              */
/**********************************************************************/

disp_menu(pm)

MESSAGE *pm;              /* pointer to an array of MESSAGE structures */

{
  clear_vid_mem();                                    /* clear the screen */
  cursor_off(0, 0);                                /* turn the cursor off */
  shw_date();                                         /* display the date */
  shw_time();                                         /* display the time */
  for( ; pm->mess; pm++)               /* do until null message detected */
    disp_str(pm->row, pm->col, pm->mess, glb_attr);  /* display string */
}
```

```
/*******************************************************************/
/* get_fkey() - get a function key and return it's value         */
/*******************************************************************/

unsigned char get_fkey()
{

unsigned char c;                    /* temporary storage for key value */

    while(1)                                        /* until break out */
    {
        if(get_key(&c) >= 0 && c == 0) break;        /* function key ? */
        chk_dt();               /* update date and time while waiting ? */
    }
    while(1)                                          /* until return */
    {
        if(get_key(&c) >= 0) return(c);         /* return function key */
        chk_dt();               /* update date and time while waiting ? */
    }
}

/*******************************************************************/
/* get_keys() - get string of characters from keyboard input buffer  */
/*******************************************************************/

get_keys(row, col, pc)           /* get string of char from input buf */

int row;            /* row location to start displaying keyboard input */
int col;         /* column location to start displaying keyboard input */
char *pc;                            /* where to store keyboard input */

{
char c;                                          /* temporary storage */
int tcol = col;                              /* remember the start column */
```

```
  *pc = ' ';                                    /* clear the  first location */
  while(1)                      /* do forever, actually until a return key */
  {
    cursor_on(row, tcol);                 /* position cursor and turn it on */
    while(1)                    /* do forever, actually until we have a key */
    {
      if(get_key(&c) < 0)                  /* get scan code from input buff */
        if(c == 0x00) beep();                    /* no function keys please */
        else break;                                 /* otherwise, break out */
      chk_dt();                 /* update date and time while waiting ? */
    }
    while(get_key(&c) < 0)                       /* get char from input buff */
      chk_dt();                 /* update date and time while waiting ? */
    if(c == 0x0d)                                            /* return key ? */
    {
      *pc = 0x00;                  /* terminate input string with a null */
      cursor_off(0, 0);                                  /* turn cursor off */
      return;                                           /* return to caller */
    }
    else if(c == 0x08)                                       /* back space ? */
    {
      if(tcol > col)                                      /* room to back up ? */
      {
        *(--pc) = ' ';  /* remove last character and decrement pointer */
        disp_t(row, --tcol, *pc, glb_attr);            /* redisplay line */
      }
      else beep();                           /* if can't back up, then beep */
    }
    else if(c == 0x00)                         /* function or modified key ? */
    {
      while(get_key(&c) < 0)                         /* dump key that follows */
        chk_dt();               /* update date and time while waiting ? */
      beep();                                  /* let them know we dumped it */
    }
    else
    {
      *pc++ = c;                               /* concatenate to input string */
      disp_t(row, tcol++, c, glb_attr);                  /* redisplay line */
    }
  }
}
```

```
/************************************************************************/
/* chk_dt() - See if time to display date or time                      */
/************************************************************************/

chk_dt()
{
extern int time_flag;  /* located in real time clock interrupt handler */
extern int date_flag;  /* located in real time clock interrupt handler */
int intr_flag;                   /* temporary storage for CPU IF state */

    if(time_flag)                     /* time flag set by RTC handler ? */
    {
        shw_time();                                /* display the time */
        intr_flag = int_off();                /* no interrupts please */
        time_flag = 0;                         /* clear the time flag */
        int_on(intr_flag);                         /* allow interrupts */
    }
    if(date_flag)                     /* date flag set by RTC handler ? */
    {
        shw_date();                                   /* show the date */
        intr_flag = int_off();                /* no interrupts please */
        date_flag = 0;                         /* clear the date flag */
        int_on(intr_flag);                         /* allow interrupts */
    }
}
```

```
/*************************************************************************/
/* main() - execute all examples from main menu                          */
/*************************************************************************/

main()
{
static MESSAGE mmain[] =                                 /* opening menu */
{
  { 3, 24, "VAXmate Hardware Programming Example" },
  { 5, 24, "F1.  CMOS / Real Time Clock" },
  { 6, 24, "F2.  8254 Timer & Speaker" },
  { 7, 24, "F3.  Video System" },
  { 8, 24, "F4.  Keyboard" },
  { 9, 24, "F5.  Serial Communications" },
  { 10, 24, "F6.  Mouse" },
  { 11, 24, "F7.  Diskette Drive" },
  { 12, 24, "F8.  Hard Disk" },
  { 13, 24, "F10. Exit From Demo" },
  { 14, 24, "F11. Warm Boot" },
  { 0,  0,  0 },
};

static MESSAGE caution[] =                               /* caution menu */
{
  { 3, 30, "***** WARNING *****" },
  { 5, 31, "THIS DISK EXAMPLE" },
  { 6, 30, "CAN DESTROY THE DATA" },
  { 7, 33, "ON YOUR DISK" },
  { 10, 33, "F1.  Continue" },
  { 11, 33, "F10. Main Menu" },
  { 0,  0,  0 },
};

int intr_flag;                       /* temporary storage for CPU IF state */
int key;                                /* temporary storage for input key */
```

```
intr_flag = int_off();   /* no interrupts while I take over hardware */
set_mode(3);               /* set the video mode to ROM BIOS default */
clear_vid_mem();                                 /* clear the screen */
ini_it_ds();    /* initialize pointer to interrupt time data segment */
pic_init();           /* initialize peripheral interupt controllers */
dma_init();                             /* initialize dma controller */
rtc_init();                              /* intialize real time clock */
kyb_init();                                   /* initialize keyboard */
dma_open(2);          /* open dma channel 2 for diskette controller */
fdc_init();                         /* initalize diskette controller */
hdc_init();                        /* initialize hard disk controller */
int_on(intr_flag);                               /* allow interrupts */
while(1)
{
  disp_menu(mmain);                           /* display the main menu */
  switch(get_fkey())         /* get function key for menu selection */
  {
    case F1:                        /* run real time clock example ? */
      rtc();
      break;

    case F2:                         /* run timer/speaker example ? */
      tim_spk();
      break;

    case F3:                                 /* run video example ? */
      video();
      break;

    case F4:                            /* run keyboard example ? */
      kyb_exm();
      break;

    case F5:
      so();                               /* run serial example ? */
      break;

    case F6:
      mouse() ;                            /* run mouse example ? */
      break;
```

```
    case F7:                        /* run diskette controller example ? */
      for(key = 0; (key != F1) && (key  != F10); )          /* abort ? */
      {
        disp_menu(caution);                 /* display caution message */
        key = get_fkey();                          /* get a function key */
      }
      if(key == F1) fdc();                      /* proceed to example ? */
      break;

    case F8:                       /* run hard disk controller example ? */
      for(key = 0; (key != F1) && (key  != F10); )          /* abort ? */
      {
        disp_menu(caution);                 /* display caution message */
        key = get_fkey();                            /* get function key */
      }
      if(key == F1) hdc();                      /* proceed to example ? */
      break;

    case F10:                                        /* exit from demo ? */
      intr_flag = int_off();                    /* no interrupts please */
      rtc_rest();                 /* restore old real time clock vector */
      kyb_rest();                        /* restore old keyboard vector */
      fdc_rest();                 /* restore diskette contoller vector */
      hdc_rest();               /* restore hard disk controller vector */
      clear_vid_mem();                               /* clear the screen */
      cursor_on(0, 0);     /* move cursor to top left and turn it on */

      /* restore normal MSDOS state of interrupts */
      /* master's slave input (IRQ2) always on after reset */
      imask(0, 0, 1);                                      /* timer on */
      imask(0, 1, 1);                                   /* keyboard on */
      imask(0, 6, 1);                         /* diskette controller on */
      imask(1, 3, 1);                        /* serial printer port on */
      imask(1, 6, 1);                          /* hard disk controller */
      int_on(intr_flag);                             /* allow interrupts */
      exit(0);                                          /* normal exit */

    case F11:                                           /* warm boot ? */
      sys_reset();
      break;
    }
  }
}
```

```
sys_reset()
{

int far *warm_boot = (int far *)0x00400072;
int (far *p_reset)() = (int (far *)())0xffff0000;

  *warm_boot = 0x1234;                               /* indicate warm boot */
  (*p_reset)();                                          /* call boot code */
                                                       /* and never return */
}
```

Appendix B
80286 Instruction Set

Instruction	Operation
AAA	ASCII adjust for addition
AAD	ASCII adjust for division
AAM	ASCII adjust for multiply
AAS	ASCII adjust for subtraction
ADC	Add byte or word with carry
ADD	Add byte or word
AND	*AND* byte or word
ARPL	Adjust RPL field of selector
BOUND	Detect values outside prescribed range
CALL	Call procedure
CBW	Convert byte into word
CLC	Clear carry flag
CLD	Clear direction flag
CLI	Clear interrupt enable flag
CLTS	Clear task switched flag
CMC	Complement carry flag
CMP	Compare byte or word
CMPS	Compare byte or word string
CWD	Convert word to double word
DAA	Decimal adjust for addition
DAS	Decimal adjust for subtraction
DEC	Decrement byte or word by 1
DIV	Unsigned divide by byte or word
ENTER	Make stack frame for procedure parameters
ESC	Escape to extension processor
HLT	Halt until interrupt or reset
IDIV	Integer divide byte or word (signed)
IMUL	Integer multiply byte or word (signed)
IN	Input byte or word
INC	Increment byte or word by 1

Instruction	Operation (80286 Instruction Set - cont.)
INS	Input byte or word string
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return
JA/JNBE	Jump if above/not below or equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above or equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JCXZ	Jump if register CX is 0
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less or equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater or equal
JLE/JNG	Jump if less or equal/not greater
JMP	Jump
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
LAHF	Load AH register from flags
LAR	Load access rights byte
LDS	Load double-word pointer to DS and word register
LEA	Load effective address
LEAVE	Restore stack for procedure exit
LES	Load double-word pointer to ES and word register
LGDT/LIDT	Load global/interrupt descriptor table
LLDT	Load local descriptor table register
LMSW	Load machine status word
LOCK	Lock bus during next instruction
LODS	Load byte or word string
LOOP	Loop with CX as a counter
LOOPE/LOOPZ	Loop while equal/zero and CX not equal to 0
LOOPNE/LOOPNZ	Loop while not equal/not zero and CX not equal to 0
LSL	Load segment limit
LTR	Load task register
MOV	Move byte or word
MOVS	Move byte or word string
MUL	Multiply byte or word unsigned

Instruction	Operation (80286 Instruction Set - cont.)
NEG	Negative byte or word
NOP	No operation
NOT	*NOT* byte or word
OR	*Inclusive-OR* byte or word
OUT	Output byte or word
OUTS	Output bytes or word string
POP	Pop word off stack
POPA	Pop all registers from stack
POPF	Pop flags off stack
PUSH	Push word onto stack
PUSHA	Push all registers onto stack
PUSHF	Push flags onto stack
RCL	Rotate left through carry byte or word
RCR	Rotate right through carry byte or word
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
RET	Return from subroutine
ROL	Rotate left byte or word
ROR	Rotate right byte or word
SAHF	Store AH register in flags
SAR	Shift arithmetic right byte or word
SBB	Subtract byte or word with borrow
SCAS	Scan byte or word string
SGDT/SIDT	Store global/interrupt descriptor table register
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SLDT	Store local descriptor table register
SMSW	Store machine status word
STC	Set carry flag
STD	Set direction flag
STI	Set interrupt enable flag
STOS	Store byte or word string
STR	Store task register
SUB	Subtract byte or word
TEST	Logical *AND* of operands (only sets flags)
VERR/VERW	Verify a segment for read or write
WAIT	Wait for BUSY not active
XCHG	Exchange byte or word
XLAT	Translate byte (from look-up table)
XOR	*Exclusive-OR* byte or word

Appendix C
VT220 and VT240 Terminal Emulators

This appendix is divided into three parts:

VT220 Emulator	Discusses the differences in characteristics and functionality between the VAXmate workstation's VT220 emulator and DIGITAL's VT220 terminal.
VT240 Emulator	Discusses the differences in characteristics and functionality between the VAXmate workstation's VT240 emulator and DIGITAL's VT240 terminal.
Tables	Contains the DEC multinational, ISO Latin-1, and special graphics character set tables.

Wherever the words VT220 or VT240 emulator are used, it means the VT220 or VT240 terminal emulator for the VAXmate workstation.

For more information on the DIGITAL VT220 and VT240 terminals, see the:

- *VT220 Series Programmer's Reference Manual*
- *VT240 Series Programmer's Reference Manual*

It is assumed that you have read the *VAXmate User's Guide.*

VT220 Emulator and VT220 Terminal Differences

This part of the appendix describes the VT220 emulator and VT220 terminal differences.

Saving and Restoring Set-Up Selections

The VT220 emulator supports saving and recalling Set-Up selections from user-specified files. The VT220 terminal supports the saving and recalling of only one set of Set-Up selections.

Video Differences

Scrolling

There is no smooth scroll/jump scroll option in the VT220 emulator. The VT220 emulator is always in jump scroll.

Blinking Characters Remapped

The VT220 emulator does not support blinking characters. Use the Display Set-Up screen to select how the blink attribute is displayed. The settings are:

- Normal video (default)
- Reverse video
- Underscore

No Control Representation Mode

The VT220 emulator does not support either a Set-Up selection or the function of control representation mode.

Font Selection

The VT220 emulator supports a Display Set-Up selection of font sizes that the VT220 terminal does not. The settings are:

- Normal (default)
- Small
- Automatic

Communications Differences

LAT Protocol Support (Network Terminal Services)

Normally a terminal can only connect to a host using a serial port. The VT220 emulator supports a Communications Set-Up selection of Network Terminal Services, which provides for ThinWire ethernet connection using a LAT protocol.

No Split Baud Rate

The VT220 emulator does not support split baud rate. It transmits and receives at the same baud rate, as specified in the Speed selection in Communications Set-Up.

Session Logging

The VT220 emulator supports an Action Set-Up selection of session logging, in which the characters received from the host are written to a file. The VT220 terminal does not.

Autotyping Characters

The VT220 emulator supports an Action Set-Up selection of autotyping, in which characters read from a file are sent to the host. The VT220 terminal does not.

Keyboard Differences

Keyboard LEDs

The LK250 keyboard has four LEDs, but the VT220 emulator supports only the Lock LED. The Lock light behaves in the same way for both a VT220 terminal and the VT220 emulator. The other LEDs (NumLock, Hold, and Special) are always off for the VT220 emulator.

In the VT220 emulator, when Lock is turned on in the Control Panel, pressing the Shift key while you press a letter key, produces a lowercase letter. Conversely, when Lock is off, pressing the Shift key while pressing a letter key, produces an uppercase letter.

Alternate Characters

Alternate character keys are only available through the MS-Windows Control Panel country settings. For more information, see the *VAXmate User's Guide.*

Keyclick

The user cannot change keyclick volume in the VT220 emulator Set-Up. Keyclick volume can be selected by using the MS-Windows Control Panel. For more information, see the *VAXmate User's Guide.*

In addition, the VT220 emulator keys always click if the volume has not been set to off by the Control Panel.

Autorepeat Selection

Autorepeat settings (on or off) cannot be selected with Set-Up in the VT220 emulator. Use the MS-Windows Control Panel to make this selection. For more information, see the *VAXmate User's Guide.*

Character Sets

DEC MCS to ISO Latin-1 8-bit Transition

The VT220 terminal does not support the ISO Latin-1 8-bit character set, but the VT220 emulator does.

The user-preference supplemental character set is selected in General Set-Up to be either the DEC multinational character set or the ISO Latin-1 character set.

The user-preference character sets, DEC MCS and ISO Latin-1 8-bit, are found in the tables at the end of this appendix.

The factory default user preference character set is the DEC multinational character set.

Language Selection

The VT220 terminal supports a Set-Up selection for the national language to be used with the National Replacement character set (NRC). The VT220 emulator does not.

This selection depends upon the country keyboard selection reported in the Preferences menu in the MS-Windows Control Panel. The country keyboard is determined during MS-Windows configuration Setup.

Compose Sequences

The VT220 terminal supports only DEC MCS (multinational character sets) Compose sequences. The VT220 emulator supports both Compose sequences and ISO Latin-1 Compose sequences.

The set of Compose sequences in effect at any given time is determined by selecting of Multinational character set in General Set-Up.

Enabling or disabling the warning bell from the Keyboard Set-Up screen does not effect the Compose sequence error bell, which is always enabled.

Additional VT220 Emulator Escape Sequences

This section lists the additional escape sequences accepted and returned by the VT220 emulator. For more information about escape sequences and character sets, see the tables at the end of this appendix.

Assign User-Preference Supplemental Character Set (DECAUPSS)

To assign a user-preference supplemental character set, use the following escape sequences:

```
DEC Supplemental:   DCS   0    !    u    %    5     ST
                    9/0  3/0  2/1  7/5  2/5  3/5   9/12


ISO Latin-1 Supplemental:  DCS   1    !    u    A    ST
                           9/0  3/1  2/1  7/5  4/1  9/12
```

Request User-Preference Supplemental Character Set (DECRQUPSS)

To request a user-preference supplemental character set, use the following escape sequence:

```
CSI   &   u
9/11  2/6 7/5
```

When this sequence is received, the VT220 emulator returns the DECAUPSS sequence indicating the currently selected user-preference character set.

Select User-Preference Supplemental Coded Character Set (SCS)

To select the user-preference supplemental coded character set, use the following escape sequence:

```
ESC   Ig   <
1/11  2/?  3/12
```

The escape sequence designates the currently selected user-preference character set into the G-set indicated by Ig.

```
   Ig    | G-set
---------+--------
 ( 2/8   |  G0
 ) 2/9   |  G1
 * 2/10  |  G2
 + 2/11  |  G3
```

Select DEC Supplemental Coded Character Set (SCS)

To select the DEC supplemental coded character set, use the following escape sequence:

```
 ESC   Ig   %    5
1/11  2/?  2/5  3/5
```

The escape sequence designates the DEC supplemental character set into the G-set indicated by Ig.

```
   Ig     | G-set
 ---------+--------
  ( 2/8   |  G0
  ) 2/9   |  G1
  * 2/10  |  G2
  + 2/11  |  G3
```

Select ISO Latin-1 Supplemental Coded Character Set (SCS)

To select the ISO Latin-1 supplemental coded character set, use the following escape sequence:

```
ESC   Ig   A
1/11  2/?  4/1
```

The escape sequence designates the ISO Latin-1 character set into the G-set indicated in Ig.

```
   Ig     | G-set
 ---------+--------
  - 2/13  |  G1
  . 2/14  |  G2
  / 2/15  |  G3
```

Primary Device Attribute (DA)

The VT220 emulator responds to the primary Device Attributes (DA) request with the additional parameter:

14 This parameter supports the 8-bit Interface Architecture.

In a typical exchange between the host and the VT220 emulator, the VT220 emulator responds to the service code and attribute inquiries with the following sequence:

```
CSI ? 62; 1; 2; 3; 4; 6; 7; 8; 9; 14; c
```

Secondary Device Attribute (DA)

In a typical exchange between the host and the VT220 emulator, the VT220 emulator responds to the terminal identification code, firmware version level, and hardware options inquiries with the following sequence:

```
CSI > 20; Pv; Po c
```

Announcing ANSI Conformance Levels

To announce is to indicate which subset of code extension facilities are going to be employed by subsequent information interchange until the occurrence of another announcer function.

The following escape sequence is taken from ANSI standard x3.134.1:

```
ESC   SP   F*
1/11  2/0  4/?
```

```
        F*    | Identifies
  ------------+---------------
     L  4/12  | Level 1
     M  4/13  | Level 2
     N  4/14  | Level 3
```

Levels 1 and 2 assume that ASCII characters are designated into G0 and invoked into GL, and that the ISO Latin-1 Supplemental Set is designated into G1 and invoked into GR.

Level 3 assumes that ASCII characters are designated into G0 and invoked into GL.

Printing

Printer Options

The VT220 terminal supports a Set-Up selection of printer settings, such as stop bits and speed, but the VT220 emulator does not. The printer speed and stop bits are adjusted by using the MS-Windows Control Panel.

Print Terminator

The VT220 terminal supports Set-Up selection of print terminator, but the VT220 emulator does not. Every VT220 emulator print job ends with a form feed, except for the output in printer controller mode.

Print Size

The VT220 emulator supports a Set-Up selection for print size, but the VT220 terminal does not. The settings are:

- Normal (default)
- Compressed

VT240 Emulator and VT240 Terminal Differences

This part of the appendix describes the VT240 emulator and VT240 terminal differences.

Saving and Restoring Set-Up Selections

The VT240 emulator supports saving and recalling Set-Up selection from user-specified files. The VT240 terminal only supports the saving and recalling of one set of Set-Up selections.

Video Differences

Video Modes

The VT240 terminal operates in only one video mode, an 800 x 240 Text & Graphics video mode. The VT240 emulator operates in either of two video modes:

- Fast Text Only
- Text & Graphics

The Text & Graphics video mode is an 800 x 252 video bitmap. The 12 extra scan lines are used to display error messages and status information.

Automatic Video Mode Switching

The Fast Text Only video mode cannot display sixel graphics, ReGIS graphics, or Dynamic Redefinable character sets (DRCS). When either sixel graphics, ReGIS, or DRCS escape sequences are received, the VT240 emulator automatically switches to the Text & Graphics video mode.

Scrolling

There is no smooth/jump scroll option in the VT240 emulator. The VT240 emulator is always in jump scroll.

No Control Representation Mode

The VT240 emulator does not support either a Set-Up selection or the function of control representation mode.

132 Column Text

The Fast Text Only video mode is an 80 x 25 character display mode. The 25th line displays error messages and status information. When operating in 132 column mode using the Fast Text Only video mode, only 80 of the 132 columns can be seen at one time.

Pressing:

Shift/right arrow	Displays columns 52-132
Shift/left arrow	Displays columns 1-80

Underlined Characters
The VT240 emulator cannot display underlined characters in the Fast Text Only video mode. Characters intended to be underlined in this video mode are displayed with the bold attribute.

If true underlined characters are desired, select Text & Graphics video mode from the Display Set-Up screen.

Line Attributes
When using the VT240 emulator in the Text & Graphics mode, all line attributes display as they would on a VT240 Terminal.

When using the VT240 emulator in Fast Text Only mode, some line attributes display differently for double width and double height/double width.

Double Width Lines for Fast Text Only
When using double width characters in the Fast Text Only mode, they are displayed as that character followed by a space. For example:

```
t e s t
```

Double Height/Double Width Lines for Fast Text Only
When using double height/double width characters in the Fast Text Only mode, they are displayed as that character followed by a space with a blank line inserted before the next line of characters.

For example, entries of the word "test" on two separate lines would display as:

```
t e s t

t e s t
```

Because double height takes up two lines, the text is on the first line followed by a blank line on the second line.

Communications Differences

LAT Protocol Support (Network Terminal Services)

Normally a terminal can only connect to a host using a serial port. The VT240 emulator supports a Communications Set-Up selection of Network Terminal Services that provides for LAT connection through a LAT protocol.

Session Logging

The VT240 emulator supports session logging, in which characters received from the host are written to a file. The VT240 terminal does not.

Autotyping Characters

The VT240 emulator supports autotyping, in which characters read from a file are sent to the host. The VT240 terminal does not.

Keyboard Differences

Keyboard LEDs

The LK250 keyboard has four LEDs, but the VT240 emulator supports only the Hold Screen and Lock LEDs. The Hold and Lock LEDs behave in the same way for both a VT240 terminal and the VT240 emulator. The other LEDs (NumLock and Special) are always off for the VT240 emulator.

Alternate Characters

Alternate character keys are available on the VT240 emulator as they are on the VT240 terminal.

To obtain alternate character key outputs, hold down the Ctrl key while pressing the Alt key, then press the alternate character key.

No "Printer to Host" Mode

The VT240 Terminal supports a feature called Printer to Host mode, this mode is not implemented in the VT240 emulator.

Character Sets

DEC MCS to ISO Latin-1 8-bit Transition

The VT240 terminal does not support the ISO Latin-1 8-bit character set, but the VT240 emulator does.

The user-preference supplemental character set is selected in General Set-Up to be either the DEC multinational character set or the ISO Latin-1 character set.

The user-preference character sets, DEC MCS and ISO Latin-1 8-bit, are found in the tables at the end of this appendix. The factory default user preference character set is the DEC supplemental character set.

Compose Sequences

The Compose sequences are handled by the MS-DOS operating system using the KEYB program. If KEYB is loaded the user can use Compose or dead diacritical sequences to create characters that do not exist as standard keys on the keyboard. Compose mode is entered from the keyboard by pressing the Compose key or a dead diacritical key. For more information on KEYB, see Chapter 17 of this manual.

Additional VT240 Emulator Escape Sequences

This section lists the additional escape sequences accepted and received by the VT240 emulator. For more information about escape sequences and character sets, see the tables at the end of this appendix.

User-Preference Supplemental Character Set (DECAUPSS)

To assign user-preference supplemental character sets, use the following escape sequences:

```
DEC Supplemental:    DCS   0    !    u    %    5    ST
                     9/0  3/0  2/1  7/5  2/5  3/5  9/12


ISO Latin-1 Supplemental:  DCS   1    !    u    A    ST
                           9/0  3/1  2/1  7/5  4/1  9/12
```

Request User-Preference Supplemental Character Set (DECRQUPSS)

To request a user-preference supplemental character set, use the following escape sequence:

```
CSI   &    u
9/11  2/6  7/5
```

When this sequence is received, the VT240 emulator returns the DECAUPSS sequence indicating the currently selected user-preference character set.

Select User-Preference Supplemental Coded Character Set (SCS)

To select the user-preference supplemental coded character set, use the following escape sequence:

```
ESC   Ig   <
1/11  2/?  3/12
```

The escape sequence designates the currently selected user-preference character set into the G-set indicated by Ig.

Ig	G-set
(2/8	G0
) 2/9	G1
* 2/10	G2
+ 2/11	G3

Select DEC Supplemental Coded Character Set (SCS)

To select the DEC supplemental coded character set, use the following escape sequence:

```
ESC   Ig   %    5
1/11  2/?  2/5  3/5
```

The escape sequence designates the DEC supplemental character set into the G-set indicated by Ig.

```
  Ig     | G-set
---------+--------
 ( 2/8   |  G0
 ) 2/9   |  G1
 * 2/10  |  G2
 + 2/11  |  G3
```

Select ISO Latin-1 Supplemental Coded Character Set (SCS)

To select the ISO Latin-1 supplemental character set, use the following escape sequence:

```
ESC   Ig   A
1/11  2/?  4/1
```

The escape sequence designates the ISO Latin-1 character set into the G-set indicated in Ig.

```
  Ig     | G-set
---------+--------
 - 2/13  |  G1
 . 2/14  |  G2
 / 2/15  |  G3
```

Primary Device Attribute (DA)

The VT240 emulator responds to the primary Device Attributes (DA) request with the additional parameter:

14 This parameter supports the 8-bit Interface Architecture.

In a typical exchange between the host and the VT240 emulator, the VT240 emulator responds to the service code and attribute inquiries with the following sequence:

```
CSI ? 62; 1; 2; 3; 4; 6; 7; 8; 9; 14; c
```

Secondary Device Attribute (DA)

In a typical exchange between the host and the VT240 emulator, the VT240 emulator responds to the terminal identification code, firmware version level, and hardware options inquiries with the following sequence:

```
CSI > 21; Pv; Po c
```

Announcing ANSI Conformance Levels

To announce is to indicate which subset of code extension facilities are going to be employed by subsequent information interchange until the occurrence of another announcer function.

The following escape sequence is taken from ANSI standard x3.134.1:

```
ESC   SP   F*
1/11  2/0  4/?
```

```
       F*     | Identifies
  ------------+---------------
     L  4/12  | Level 1
     M  4/13  | Level 2
     N  4/14  | Level 3
```

Levels 1 and 2 assume that ASCII is designated into G0 and invoked into GL, and that the ISO Latin-1 supplemental set is designated into G1 and invoked into GR.

Level 3 assumes that ASCII is designated into G0 and invoked into GL.

This page is intentionally blank.

Table C-1 DEC MCS - ASCII Graphics Set (0-7)

COLUMN		0		1		2		3		4		5		6		7	
BITS b8 b7 b6 b5		0 0 0 0		0 0 0 1		0 0 1 0		0 0 1 1		0 1 0 0		0 1 0 1		0 1 1 0		0 1 1 1	
ROW	b4 b3 b2 b1																
0	0 0 0 0	NUL	0 0 0	DLE	20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	`	140 96 60	p	160 112 70
1	0 0 0 1	SOH	1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	a	141 97 61	q	161 113 71
2	0 0 1 0	STX	2 2 2	DC2	22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	b	142 98 62	r	162 114 72
3	0 0 1 1	ETX	3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	c	143 99 63	s	163 115 73
4	0 1 0 0	EOT	4 4 4	DC4	24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	d	144 100 64	t	164 116 74
5	0 1 0 1	ENQ	5 5 5	NAK	25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	e	145 101 65	u	165 117 75
6	0 1 1 0	ACK	6 6 6	SYN	26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	f	146 102 66	v	166 118 76
7	0 1 1 1	BEL	7 7 7	ETB	27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	g	147 103 67	w	167 119 77
8	1 0 0 0	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	h	150 104 68	x	170 120 78
9	1 0 0 1	HT	11 9 9	EM	31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	i	151 105 69	y	171 121 79
10	1 0 1 0	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	j	152 106 6A	z	172 122 7A
11	1 0 1 1	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	k	153 107 6B	{	173 123 7B
12	1 1 0 0	FF	14 12 C	FS	34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	l	154 108 6C	|	174 124 7C
13	1 1 0 1	CR	15 13 D	GS	35 29 1D	-	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	m	155 109 6D	}	175 125 7D
14	1 1 1 0	SO	16 14 E	RS	36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	n	156 110 6E	~	176 126 7E
15	1 1 1 1	SI	17 15 F	US	37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	_	137 95 5F	o	157 111 6F	DEL	177 127 7F

C1 CODES (columns 0–1) — GR CODES (DEC SUPPLEMENTAL GRAPHICS) (columns 2–7)

KEY

ASCII CHARACTER		
ESC	1/11	COLUMN/ROW
	33	OCTAL
	27	DECIMAL
	1B	HEX

LJ-0839

Table C-2 DEC MCS - Supplemental Graphics Set (8-15)

8		9		10		11		12		13		14		15		COLUMN	
1 0 0 0		1 0 0 1		1 0 1 0		1 0 1 1		1 1 0 0		1 1 0 1		1 1 1 0		1 1 1 1		b8 b7 **BITS** b6 b5 b4 b3 b2 b1	ROW
	200 128 80	DCS	220 144 90		240 160 A0	°	260 176 B0	À	300 192 C0		320 208 D0	à	340 224 E0		360 240 F0	0 0 0 0	0
	201 129 81	PU1	221 145 91	¡	241 161 A1	±	261 177 B1	Á	301 193 C1	Ñ	321 209 D1	á	341 225 E1	ñ	361 241 F1	0 0 0 1	1
	202 130 82	PU2	222 146 92	¢	242 162 A2	²	262 178 B2	Â	302 194 C2	Ò	322 210 D2	â	342 226 E2	ò	362 242 F2	0 0 1 0	2
	203 131 83	STS	223 147 93	£	243 163 A3	³	263 179 B3	Ã	303 195 C3	Ó	323 211 D3	ã	343 227 E3	ó	363 243 F3	0 0 1 1	3
IND	204 132 84	CCH	224 148 94		244 164 A4		264 180 B4	Ä	304 196 C4	Ô	324 212 D4	ä	344 228 E4	ô	364 244 F4	0 1 0 0	4
NEL	205 133 85	MW	225 149 95	¥	245 165 A5	µ	265 181 B5	Å	305 197 C5	Õ	325 213 D5	å	345 229 E5	õ	365 245 F5	0 1 0 1	5
SSA	206 134 86	SPA	226 150 96		246 166 A6	¶	266 182 B6	Æ	306 198 C6	Ö	326 214 D6	æ	346 230 E6	ö	366 246 F6	0 1 1 0	6
ESA	207 135 87	EPA	227 151 97	§	247 167 A7	·	267 183 B7	Ç	307 199 C7	Œ	327 215 D7	ç	347 231 E7	œ	367 247 F7	0 1 1 1	7
HTS	210 136 88		230 152 98	¤	250 168 A8		270 184 B8	È	310 200 C8	Ø	330 216 D8	è	350 232 E8	ø	370 248 F8	1 0 0 0	8
HTJ	211 137 89		231 153 99	©	251 169 A9	¹	271 185 B9	É	311 201 C9	Ù	331 217 D9	é	351 233 E9	ù	371 249 F9	1 0 0 1	9
VTS	212 138 8A		232 154 9A	ª	252 170 AA	º	272 186 BA	Ê	312 202 CA	Ú	332 218 DA	ê	352 234 EA	ú	372 250 FA	1 0 1 0	10
PLD	213 139 8B	CSI	233 155 9B	«	253 171 AB	»	273 187 BB	Ë	313 203 CB	Û	333 219 DB	ë	353 235 EB	û	373 251 FB	1 0 1 1	11
PLU	214 140 8C	ST	234 156 9C		254 172 AC	¼	274 188 BC	Ì	314 204 CC	Ü	334 220 DC	ì	354 236 EC	ü	374 252 FC	1 1 0 0	12
RI	215 141 8D	OSC	235 157 9D		255 173 AD	½	275 189 BD	Í	315 205 CD	Ÿ	335 221 DD	í	355 237 ED	ÿ	375 253 FD	1 1 0 1	13
SS2	216 142 8E	PM	236 158 9E		256 174 AE		276 190 BE	Î	316 206 CE		336 222 DE	î	356 238 EE		376 254 FE	1 1 1 0	14
SS3	217 143 8F	APC	237 159 9F		257 175 AF	¿	277 191 BF	Ï	317 207 CF	ß	337 223 DF	ï	357 239 EF		377 255 FF	1 1 1 1	15

C0 CODES — GL CODES (ASCII GRAPHIC)

LJ-0840

Table C-3 ISO Latin-1 Character Set (0-7)

B8 B7 B6 B5		0 0 0 0		0 0 0 1		0 0 1 0		0 0 1 1		0 1 0 0		0 1 0 1		0 1 1 0		0 1 1 1	
BITS B4 B3 B2 B1	ROW	COLUMN 0		1		2		3		4		5		6		7	
0 0 0 0	0	NUL	0 0 0	DLE	20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	`	140 96 60	p	160 112 70
0 0 0 1	1	SOH	1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	a	141 97 61	q	161 113 71
0 0 1 0	2	STX	2 2 2	DC2	22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	b	142 98 62	r	162 114 72
0 0 1 1	3	ETX	3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	c	143 99 63	s	163 115 73
0 1 0 0	4	EOT	4 4 4	DC4	24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	d	144 100 64	t	164 116 74
0 1 0 1	5	ENQ	5 5 5	NAK	25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	e	145 101 65	u	165 117 75
0 1 1 0	6	ACK	6 6 6	SYN	26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	f	146 102 66	v	166 118 76
0 1 1 1	7	BEL	7 7 7	ETB	27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	g	147 103 67	w	167 119 77
1 0 0 0	8	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	h	150 104 68	x	170 120 78
1 0 0 1	9	HT	11 9 9	EM	31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	i	151 105 69	y	171 121 79
1 0 1 0	10	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	j	152 106 6A	z	172 122 7A
1 0 1 1	11	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	k	153 107 6B	{	173 123 7B
1 1 0 0	12	FF	14 12 C	FS	34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	l	154 108 6C	|	174 124 7C
1 1 0 1	13	CR	15 13 D	GS	35 29 1D	-	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	m	155 109 6D	}	175 125 7D
1 1 1 0	14	SO	16 14 E	RS	36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	n	156 110 6E	~	176 126 7E
1 1 1 1	15	SI	17 15 F	US	37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	_	137 95 5F	o	157 111 6F	DEL	177 127 7F

KEY

ASCII CHARACTER			
	ESC	1 11	COLUMN ROW
		33	OCTAL
		27	DECIMAL
		1B	HEX

LJ-0865

Table C-4 ISO Latin-1 Character Set (8-15)

BITS B8 B7 B6 B5		1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
B4 B3 B2 B1	ROW	8	9	10	11	12	13	14	15
0 0 0 0	0	200 128 80	DCS 220 144 90	NBSP 240 160 A0	° 260 176 B0	À 300 192 C0	Đ 320 208 D0	à 340 224 E0	ð 360 240 F0
0 0 0 1	1	201 129 81	PU1 221 145 91	¡ 241 161 A1	± 261 177 B1	Á 301 193 C1	Ñ 321 209 D1	á 341 225 E1	ñ 361 241 F1
0 0 1 0	2	202 130 82	PU2 222 146 92	¢ 242 162 A2	2 262 178 B2	Â 302 194 C2	Ò 322 210 D2	â 342 226 E2	ò 362 242 F2
0 0 1 1	3	203 131 83	STS 223 147 93	£ 243 163 A3	3 263 179 B3	Ã 303 195 C3	Ó 323 211 D3	ã 343 227 E3	ó 363 243 F3
0 1 0 0	4	IND 204 132 84	CCH 224 148 94	¤ 244 164 A4	´ 264 180 B4	Ä 304 196 C4	Ô 324 212 D4	ä 344 228 E4	ô 364 244 F4
0 1 0 1	5	NEL 205 133 85	MW 225 149 95	¥ 245 165 A5	µ 265 181 B5	Å 305 197 C5	Õ 325 213 D5	å 345 229 E5	õ 365 245 F5
0 1 1 0	6	SSA 206 134 86	SPA 226 150 96	¦ 246 166 A6	¶ 266 182 B6	Æ 306 198 C6	Ö 326 214 D6	æ 346 230 E6	ö 366 246 F6
0 1 1 1	7	ESA 207 135 87	EPA 227 151 97	§ 247 167 A7	· 267 183 B7	Ç 307 199 C7	x 327 215 D7	ç 347 231 E7	÷ 367 247 F7
1 0 0 0	8	HTS 210 136 88	230 152 98	¨ 250 168 A8	¸ 270 184 B8	È 310 200 C8	Ø 330 216 D8	è 350 232 E8	ø 370 248 F8
1 0 0 1	9	HTJ 211 137 89	231 153 99	© 251 169 A9	1 271 185 B9	É 311 201 C9	Ù 331 217 D9	é 351 233 E9	ù 371 249 F9
1 0 1 0	10	VTS 212 138 8A	232 154 9A	ª 252 170 AA	º 272 186 BA	Ê 312 202 CA	Ú 332 218 DA	ê 352 234 EA	ú 372 250 FA
1 0 1 1	11	PLD 213 139 8B	CSI 233 155 9B	« 253 171 AB	» 273 187 BB	Ë 313 203 CB	Û 333 219 DB	ë 353 235 EB	û 373 251 FB
1 1 0 0	12	PLU 214 140 8C	ST 234 156 9C	¬ 254 172 AC	1/4 274 188 BC	Ì 314 204 CC	Ü 334 220 DC	ì 354 236 EC	ü 374 252 FC
1 1 0 1	13	RI 215 141 8D	OSC 235 157 9D	SHY 255 173 AD	1/2 275 189 BD	Í 315 205 CD	Ý 335 221 DD	í 355 237 ED	ý 375 253 FD
1 1 1 0	14	SS2 216 142 8E	PM 236 158 9E	® 256 174 AE	3/4 276 190 BE	Î 316 206 CE	Þ 336 222 DE	î 356 238 EE	þ 376 254 FE
1 1 1 1	15	SS3 217 143 8F	APC 237 159 9F	¯ 257 175 AF	¿ 277 191 BF	Ï 317 207 CF	ß 337 223 DF	ï 357 239 EF	ÿ 377 255 FF

LJ 0866

Table C-5 DEC Special Graphics Character Set

ROW	BITS	COLUMN 0		1		2		3		4		5		6		7	
	B7 B6 B5	0 0 0		0 0 1		0 1 0		0 1 1		1 0 0		1 0 1		1 1 0		1 1 1	
ROW	B4 B3 B2 B1																
0	0 0 0 0	NUL	0 0 0		20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	◆	140 96 60	⎺ SCAN 3	160 112 70
1	0 0 0 1		1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	▒	141 97 61	⎻ SCAN 5	161 113 71
2	0 0 1 0		2 2 2		22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	␉	142 98 62	─ SCAN 7	162 114 72
3	0 0 1 1		3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	␌	143 99 63	⎼ SCAN 9	163 115 73
4	0 1 0 0		4 4 4		24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	␍	144 100 64	├	164 116 74
5	0 1 0 1		5 5 5		25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	␊	145 101 65	┤	165 117 75
6	0 1 1 0		6 6 6		26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	°	146 102 66	┴	166 118 76
7	0 1 1 1		7 7 7		27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	±	147 103 67	┬	167 119 77
8	1 0 0 0	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	␤	150 104 68	│	170 120 78
9	1 0 0 1	HT	11 9 9		31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	␋	151 105 69	≤	171 121 79
10	1 0 1 0	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	┘	152 106 6A	≥	172 122 7A
11	1 0 1 1	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	┐	153 107 6B	π	173 123 7B
12	1 1 0 0	FF	14 12 C		34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	┌	154 108 6C	≠	174 124 7C
13	1 1 0 1	CR	15 13 D		35 29 1D	-	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	└	155 109 6D	£	175 125 7D
14	1 1 1 0	SO	16 14 E		36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	┼	156 110 6E	·	176 126 7E
15	1 1 1 1	SI	17 15 F		37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	(BLANK)	137 95 5F	⎽ SCAN 1	157 111 6F	DEL	177 127 7F

KEY

ASCII CHARACTER	ESC	1/11	COLUMN/ROW
		33	OCTAL
		27	DECIMAL
		1B	HEX

(shaded) HIGHLIGHTS DIFFERENCES FROM ASCII

Bibliography

MOS Microprocessors and Peripherals 1985 Data Book (Order Number 04426A)
Advanced Micro Devices
901 Thompson Place
P.O. Box 3453
Sunnyvale, CA 94088

8/16-Bit Multi-Chip Microcomputer Data Book
Hitachi America, Ltd.
Semiconductor and IC Division
2210 O'Toole Avenue
San Jose, CA 95131
1-408-942-1500

IBM Personal Computer Seminar Proceedings
Volume 2, Number 8, October 8, 1984
IBM Corporation
Editor, IBM Personal Computer Seminar Proceedings
4629
Post Office Box 1328
Boca Raton, FL 33432

Microsystem Components Handbook (Publication Number 230843)
Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130
1-800-548-4725

Server/Redirector File Sharing Protocol (Publication Number 136329-001)
Microsoft Corporation and Intel Corporation
Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130
1-800-548-4725

MS-Windows Software Development Kit Programmer's Reference
MS-Windows Software Development Kit Programmer's Utility Guide
MS-Windows Software Development Kit Programming Guide
Microsoft Corporation
16011 NE 36th Way
Box 97017
Redmond, WA 98073-9717

8-Bit Microprocessor & Peripheral Data
Motorola, Inc.
MOS Integrated Circuits Group
Microprocessor Division
3501 Ed Bluestein Blvd.
Austin, TX 78721
1-512-928-6800

Series 8000 Microprocessor Family Handbook
National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, CA 95051
1-408-737-5000

Microcomputer Products
NEC Electronics
401 Ellis Street
Mountain View, CA 94043
1-415-960-6000

Microprocessor Data Manual 1986
Signetics Corporation
811 E. Arques Avenue
P.O. Box 3409
Sunnyvale, CA 94088-3409
1-408-991-2000

1984 Data Communications Products Handbook
1986 Storage Management Products Handbook
Western Digital Corporation
Marketing Communications Manager
2445 McCabe Way
Irvine, CA 92714
1-714-863-0102

DECnet-DOS Programmer's Reference Manual (AA-EB46B-TV)

DECnet-DOS User's Guide (AA-EB45B-TV)

DECnet-VAXmate Programmer's Reference Manual (AA-GV34A-TH)

DECnet-VAXmate User's Guide (AA-GV36A-TH)

DECnet Digital Network Architecture Phase IV General Description (AA-N149A-TC)

DECnet Digital Network Architecture Phase IV Control Functional Specification (AA-K182A-TK)

DECnet Digital Network Architecture Phase IV Routing Functional Specification (AA-X435A-TK)

DECnet Digital Network Architecture Phase IV Maintenance Operations Functional Specification (AA-X437A-TK)

DECnet Digital Network Architecture Phase IV Network Management Functional Specification (AA-X436A-TK)

DECnet Digital Network Architecture Phase IV Data Access Protocol (DAP) Functional Specification (AA-K177A-TK)

The Ethernet, A Local Area Network, Datalink Layer and Physical Layer Specification, Version 2.0 (AA-K759B-TK)
Digital Equipment Corporation, Intel Corporation, Xerox Corporation
November 1982

VT220 Series Programmer's Reference Manual (EK-VT220-RM)

VT240 Series Programmer's Reference Manual (EK-VT240-RM)

VAXmate User's Guide (Q6A93-GZ)

To order DIGITAL documentation:

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

Index

D

E

F

H

I

J

K

L

M

N

O

P

R

T

W

Technical Reference Manual
Volume 2
AA-HD95A-TK

Reader's Comments

Your comments on this manual will help improve our product quality and usefulness.

Please indicate the type of reader you most closely represent.

☐ First-time user ☐ Programmer ☐ Experienced user

☐ Application user ☐ Other (please specify)________________________

How would you rate this manual for:

	Excellent	Good	Fair	Poor
Completeness of Information	☐	☐	☐	☐
Accuracy of Information	☐	☐	☐	☐
Easy to Read/Use	☐	☐	☐	☐
Usefulness of Examples	☐	☐	☐	☐
Number of Examples	☐	☐	☐	☐
Illustrations	☐	☐	☐	☐
Table of Contents	☐	☐	☐	☐
Index	☐	☐	☐	☐
Format	☐	☐	☐	☐
Binding Style	☐	☐	☐	☐
Print Quality	☐	☐	☐	☐

Did you find any errors in this manual? Please specify by page and paragraph.

Incorrect information: ____

____

Information left out: ____

____

Hard to understand: ____

____

What suggestions do you have for improving this manual? Attach a second sheet if necessary.

____

____

____

Name______________________ Title______________________

Company______________________ Dept.______________________

Street______________________ City______________________

State/Country______________________ Postal/Zip Code______________________

Telephone______________________ Date______________________

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MRO1-2 L12
MARLBOROUGH, MA 01752

Do Not Tear - Fold Here

Cut Along Dotted Line

Technical Reference Manual
Volume 2
AA-HD95A-TK

Reader's Comments

Your comments on this manual will help improve our product quality and usefulness.

Please indicate the type of reader you most closely represent.

☐ First-time user ☐ Programmer ☐ Experienced user

☐ Application user ☐ Other (please specify)____________________

How would you rate this manual for:

	Excellent	**Good**	**Fair**	**Poor**
Completeness of Information	☐	☐	☐	☐
Accuracy of Information	☐	☐	☐	☐
Easy to Read/Use	☐	☐	☐	☐
Usefulness of Examples	☐	☐	☐	☐
Number of Examples	☐	☐	☐	☐
Illustrations	☐	☐	☐	☐
Table of Contents	☐	☐	☐	☐
Index	☐	☐	☐	☐
Format	☐	☐	☐	☐
Binding Style	☐	☐	☐	☐
Print Quality	☐	☐	☐	☐

Did you find any errors in this manual? Please specify by page and paragraph.

Incorrect information: ____________________

Information left out: ____________________

Hard to understand: ____________________

What suggestions do you have for improving this manual? Attach a second sheet if necessary.

Name__________ Title__________

Company__________ Dept.__________

Street__________ City__________

State/Country__________ Postal/Zip Code__________

Telephone__________ Date__________

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MRO1-2 L12
MARLBOROUGH, MA 01752

Do Not Tear - Fold Here

Cut Along Dotted Line