CMU-ITC-84-019

# MAXTOR Support Package for the IBM Personal Computer

June 27, 1984

L. K. Raper

Information Technology Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA, 15213

VNET: CMULKR at PGHVM1

IBM Internal Use Only

## ABSTRACT

This document describes a software package which supports the use of one or two MAXTOR XT-1140 Fixed Disk drive(s) attached to a Zobex 4HDC-62W Hard Disk Controller board with the IBM Personal Computer or Personal Computer/XT. The MAXTORs may be used in conjunction with standard IBM Fixed Disk drives or as a replacement for the IBM drives. Each MAXTOR has a formatted capacity of 116.9 Megabytes.

The MAXTOR Support software described here provides variable partitioning, selective or full initialization, transparent alternate sector retry, emulation of the standard BIOS fixed disk interface, a DOS device driver that also works with SRITEK coprocessor boards, and support for booting from a MAXTOR partition.

The MAXTOR Support Package has been carefully layered so that the functions provided are potentially available to multiple operating systems. Although all of the utility programs run only under PC-DOS (version 2.0 or later), additional drivers have been developed to allow the MAXTOR to be used with PC/IX, and XENIX or GENIX running in a SRITEK coprocessor board.

## ACKNOWLEDGEMENTS

Dave Hildebrandt of the IBM San Jose Research Laboratory graciously supplied the source for earlier MAXTOR support code he had developed that proved the feasibility of MAXTOR PC attachment and served as an example for this work.

Bryan Striemer's work in designing and developing a versatile PROM Programmer card for the IBM PC made it possible for us to burn a variety of ROMs to hold the MAXTOR boot support code.

John Howard has also implemented a PC/IX device driver for the MAXTOR which utilizes the same partitioning and alternate sector architecture and works in conjunction with the MAXTOR utilities described here.

Mike West, Jr was responsible for the coding and testing of the MAXFMT utility.

CONTENTS

# 1.0 INVENTORY LIST

The MAXTOR Support Package contains the following program files:

| | |
|---|---|
| MAXDD.SYS | The MAXTOR device driver. It must be installed in order to use any of the MAXTOR utilities. |
| MAXPART.EXE | The MAXTOR partitioning utility. This program is used to define, change, or delete MAXTOR partitions. MAXTOR partitions may or may not be accessible to PC-DOS. Any partition accessible to DOS will appear as a distinct device, requiring further initialization using the DOS FDISK program. |
| MAXINIT.EXE | The MAXTOR initialization utility. This program prepares a MAXTOR drive for subsequent use by formatting tracks, performing a surface analysis, assigning alternate sectors, etc. |
| MAXFMT.EXE | A utility program for formatting DOS partitions. This program creates the structures needed by PC-DOS to manage files in a DOS partition (boot record, FAT, root directory, etc). It has the same syntax as (and functions similar to) the native DOS FORMAT command. |
| MAXBOOT.SYS | This is the code which can be placed in an EPROM and used to boot from a MAXTOR. |
| CHECKSUM.EXE | A program used to compute and add in the checksum for the MAXBOOT.SYS module. |

Source code for the MAXTOR support is also available in the following files:

| | |
|---|---|
| MAXBIOS.ASM | Device support nucleus (read, write, verify, format) |
| MAXDD.ASM | DOS device driver |
| MAXUTILS.ASM | Common subroutines |
| MAXVEC.ASM | High performance interface for SRITEK coprocessor cards |
| MAXASR.ASM | Alternate Sector Retry |
| MAXBOOT.ASM | Boot support |
| MAXDOS2.DCL | Declarations relating to DOS device drivers |
| MAXZFC.DCL | Declarations relating to the ZOBEX file controller |

| | |
|---|---|
| MAXDATA.DCL | Global MAXTOR declarations |
| MAXPART.C | MAXTOR Partitioning Utility |
| MAXINIT.C | MAXTOR Initialization Utility |
| MAXSUBS.C | Common subroutines for MAXTOR utilities |
| MAXHDR.H | Common declarations for MAXTOR utilities |
| MAXFMT.C | MAXTOR DOS Formatter. |
| MAKEDD.BAT | BAT file to generate MAXDD.SYS |
| MAKEBOOT.BAT | BAT file to generate MAXBOOT.SYS |
| MAKEINIT.BAT | BAT file to generate MAXINIT.EXE |
| MAKEPART.BAT | BAT file to generate MAXPART.EXE |
| MAXDD.ARF | Automatic response file for MAKEDD.BAT |
| MAXBOOT.ARF | Automatic response file for MAKEBOOT.BAT |
| MAXINIT.ARF | Automatic response file for MAKEINIT.BAT |
| MAXPART.ARF | Automatic response file for MAKEPART.BAT |
| CHECKSUM.C | Checksum program for ROMable code. |

## 2.0 CONFIGURING THE ZOBEX HARD DISK CONTROLLER BOARD.

Follow the instructions in the ZOBEX User's Manual for the attachment of a large capacity drive with more than 8 heads. Briefly, this consists of moving the P1 jumper to the HS3 position above the small 6 inscribed on the board. This change is needed to permit the use of all 15 heads on the MAXTOR.

Set the eight DIP switches at U51 to reflect a base I/O address of hex 250. This setting was found to have fewer conflicts with add-on PC boards than the ZOBEX factory default setting of 300. The following switch settings result in an I/O address of 250:

| Switch | Setting |
|---|---|
| 1 | off |
| 2 | on |
| 3 | on |
| 4 | off |
| 5 | on |
| 6 | off |
| 7 | on |
| 8 | on |

If you find it necessary to move to an I/O address other than 250, you will need to change one declaration in the file MAXZFC.DCL and reassemble MAXBIOS.ASM. The new MAXBIOS.OBJ file must be relinked with MAXASR.OBJ, MAXDD.OBJ, and MAXUTILS.OBJ to produce a new MAXDD.SYS module.

## 3.0 INSTALLING THE MAXTOR DEVICE DRIVER.

Before you can use any of the MAXTOR utility programs, the MAXTOR device driver must be installed. MAXDD.SYS is a standard PC-DOS device driver, and requires an entry in a CONFIG.SYS file in the root directory of your boot device. If you plan to (eventually) boot from your MAXTOR drive, prepare a bootable diskette with a CONFIG.SYS file and a copy of MAXDD.SYS.

Add the following line anywhere in your CONFIG.SYS file:

```
device=maxdd.sys
```

If the MAXDD.SYS file is kept in another directory, you may specify the full path name following "device=".

After completing this step reboot your system to load the MAXTOR device driver.

# 4.0 PREPARING THE MAXTOR DRIVE FOR USE.

## 4.1 BACKGROUND

Before you can store data on the MAXTOR you must "initialize" it. Initialization consists of:

- Creating a "system area" which will hold partitioning information, alternate sector tables, and a pool of alternate sectors, in a reserved area of the drive (absolute cylinder number 918).

- Formatting some or all of the remaining usable space (your choice) with empty sectors and performing a surface analysis to determine which, if any, sectors are unusable and should be fetched from the alternate sector pool.

- Preparing partitions for use by the appropriate operating system. If you intend to use one or more MAXTOR partitions with PC-DOS you need to run the DOS FDISK program and either the DOS FORMAT command or the MAXFMT program (which of these is needed will be discussed later).

  The DOS FDISK utility must be used because each MAXTOR partition that you designate for use with DOS will appear as a separate drive to DOS, with a unique device address. For partitions which you will use with UNIX, run the UNIX utility **mkfs**.

The second (formatting) phase of this operation is time consuming. To format an entire MAXTOR and perform a complete surface analysis will probably take about 14 hours. You can shorten this time if you want (although this is not recommended) in one of two ways:

1. If you are not planning on using all of the space available on the MAXTOR immediately, you can initialize the portion you need for now, and do the remaining initialization later. The initialization can be done on a partition by partition basis, without affecting the contents of other partitions.

   To do this you will need to run the MAXPART utility to create the partitions you want prior to running the MAXINIT program.

2. You can deactivate the surface analysis, which will result in a formatting operation which is much faster, and can probably be done in a couple of hours. If you elect to do this, no alternate sectors will be assigned and no defective areas will be discovered. Undiscovered defective areas may result in loss of data at a later time.

Naturally, if you're interested in saving time, you could also combine these two approaches by doing only a partial initialization and also deactivating the surface scan. However the recommended procedure is to perform the full initialization and let it complete by running it overnight.

Each MAXTOR is shipped from the manufacturer with a defect map. You can use the defect map in conjunction with the initialization program to explicitly assign alternate sectors to the defective areas. This is recommended regardless of whether or not a complete surface analysis has been performed, because some defects cannot be discovered using the digital techniques incorporated in the initialization program. If you have suppressed the surface analysis to save time during initialization, this step is essential!

## 4.2 INITIALIZATION SEQUENCE

The following 4 steps apply to all operating systems (PC-DOS, PC/IX, XENIX, or GENIX), although you must execute the MAXINIT and MAXPART utilities with PC-DOS.

1. Run the MAXINIT program first. If your MAXTOR has never been used before, it will initialize the system area (where the Partition Table and Alternate Sector information is kept) and then prompt for further instructions. If you wish to completely initialize the entire drive, you can indicate that here, otherwise, terminate the program.

2. Run the MAXPART program to create one or more partitions.

3. If you initialized the entire MAXTOR drive in step 1 above, drop to step 4, otherwise run MAXINIT again and initialize the partitions you created in step 2.

4. Reboot your machine.

The following instructions apply only to PC-DOS partitions:

5. For each partition you defined in step 2 as accessible to PC-DOS (and initialized in steps 1 or 2), you will need to run the DOS FDISK program to specify how much of the space will actually be available to DOS. Use FDISK option 5 to select each MAXTOR DOS partition. These will appear as separate drives to FDISK. For each MAXTOR "drive," select option 1 (Create DOS partition). When FDISK asks, "Do you wish to use the entire fixed disk for DOS (Y/N)," always answer "N", whether you do or not. This way you can avoid having to reboot your machine after each execution of FDISK. You can then proceed to allocate some or all of the space to DOS, as you choose.

6. Reboot your machine.

7. For each DOS partition, run either the MAXFMT utility or the DOS FORMAT command. See "MAXFMT versus FORMAT" on page 17 and "Booting from a MAXTOR" on page 19 for a discussion of which utility is appropriate in each case.

# 5.0 THE MAXINIT PROGRAM

## 5.1 SYNTAX

| Command | Operands |
|---|---|
| maxinit | [physical drive number] |

Where "physical drive number" (1 or 2) specifies the MAXTOR drive to be used by the MAXINIT program. This argument is optional. Drive 1 is assumed if only one MAXTOR is attached to your system. If two drives are attached and you omit the argument, MAXINIT will ask you which drive to use.

## 5.2 OVERVIEW

The MAXINIT program offers a variety of initialization functions, which are listed following the display of the partition table when the program starts.

If you have not yet run the MAXPART utility to create a partition table, MAXINIT will initialize the MAXTOR system area and then ask you whether or not you wish to initialize the entire drive. If you reply **y** (for yes) it will ask whether on not a complete surface analysis should be done. A yes reply will keep MAXINIT busy for about 14 hours, whereas a no will initialize the drive, but bypass the defect detection phase, resulting in a much faster initialization.

If you only wish to perform a partial initialization, you should answer no to the first question and run the MAXPART utility to define one or more partitions. The next time you run MAXINIT the existence of a partition table (of any form) will trigger it to offer a number of selective initialization options. A brief discussion of each follows.

## 5.3 FUNCTIONS

### 5.3.1 INITIALIZE A SINGLE PARTITION

This is the main function of MAXINIT. You may select one or more individual partitions for initialization by entering the partition designator listed to the left of the partition. For example, to initialize partitions b and d, type

bd

and press the Enter key.

### 5.3.2 INITIALIZE SYSTEM AREA

This function formats and performs a surface analysis on the portion of the MAXTOR reserved for the Partition Table, the Alternate Sector Table, and the Alternate Sector Pool. If a Partition Table or Alternate Sector Table already exists, their contents are preserved, but all of the data areas in the Alternate Sector Pool are destroyed.

This function is performed automatically when MAXINIT starts if no Partition Table can be found. It should be performed at least one for each MAXTOR drive. If you used MAXPART to create a partition table, prior to running MAXINIT, you should select this function before doing any other initialization. (You may perform this function at any time, but all data already present in alternate sectors will be lost).

### 5.3.3 DISPLAY ALTERNATE SECTOR TABLE

If any alternate sector assignments are in effect, you may use this function to generate a list of all sectors for which alternates have been assigned. Alternate sectors are automatically assigned for any sectors discovered to have defects during a surface scan. In addition, sectors may have been included in the Alternate Sector Table as a result of an explicit action on your part (See **Declare Bad Block** below).

### 5.3.4 CLEAR ALTERNATE SECTOR TABLE

This function removes all entries from the Alternate Sector Table. Any sectors which have already been flagged as defective will no longer be eligible for automatic alternate sector retry by the MAXTOR device driver, and will result in an uncorrectable disk read or write error when referenced.

You will not ordinarily need to use this function. It is intended for situations where you might like to completely re-initialize your MAXTOR from scratch.

### 5.3.5 DEACTIVATE SURFACE ANALYSIS

If you wish to format the MAXTOR without any implicit defect analysis or alternate sector assignments, you may use this function to deactivate the normal surface analysis. This will dramatically decrease the time required for the initialization process. Once deactivated the surface scan function will remain deactivated until either you explicitly reactivate it or quit the MAXINIT program.

### 5.3.6 ACTIVATE SURFACE ANALYSIS

This function only is only available if you have explicitly deactivated the surface analysis. By default the MAXINIT program starts with the surface analysis function active.

### 5.3.7 DECLARE BAD BLOCK

This function is used to add an entry to the Alternate Sector Table and assign an alternate sector. Whenever a reference to the original sector occurs, the assigned alternate is automatically used in its place.

There are two main reasons why you might wish to declare a bad block.

1. Not all defective areas can be discovered using the digital techniques incorporated in the surface analysis. Consequently each MAXTOR drive is accompanied with a defect map prepared by the manufacturer. After you have otherwise completed the initialization of the MAXTOR, you may use this function to specify any additional defective areas. Defects may be specified by entering the cylinder number, track (or head) number, and byte offset of the defective areas, just as they are listed on the defect map. If you are not sure which items on the defect map are already represented in the Alternate Sector Table, add all of them. MAXINIT will filter out any duplicate entries.

2. If read or write errors develop after you have been using your MAXTOR you can use this function to add suspected bad sectors to the Alternate Sector Table and assign alternate sectors as replacements. Any

data already present in the sectors you designate is read out (if possible) and used to initialize the replacement sectors, so, in general, no further data loss will occur.

### 5.3.8 INITIALIZE ENTIRE USABLE AREA

This function will initialize all cylinders from 0 through 917. The Alternate Sector Table is automatically cleared prior to the initialization.

### 5.3.9 QUIT

This causes the MAXINIT program to exit normally. You may also exit the MAXINIT program prematurely by using the Control-Break keys. Any pending updates to the Alternate Sector Table are applied prior to exiting the program.

# 6.0 THE MAXPART PROGRAM

## 6.1 SYNTAX

| Command | Operands |
|---|---|
| maxpart | [physical drive number] |

Where "physical drive number" (1 or 2) specifies the MAXTOR drive to be used by the MAXPART program. This argument is optional. Drive 1 is assumed if only one MAXTOR is attached to your system. If two drives are attached and you omit the argument, MAXPART will ask you which drive to use.

## 6.2 A WORD ABOUT PARTITIONS

The MAXPART utility allows you to create, change, or destroy partitions on a MAXTOR drive. A partition is a somewhat arbitrary subdivision of the drive by cylinders. You can use partitions for the following purposes:

- To carve the space available on the MAXTOR into more manageable units, each of which has most of the properties of a distinct drive to the host operating system, and consequently may hold independent file systems. This is necessary for use with PC-DOS, since its File Allocation Table cannot address all of the space on the MAXTOR.

- To designate a region of the drive, when running the MAXINIT utility in order to perform a partial initialization.

- To compartmentalize the use of the MAXTOR between different operating systems. Device drivers for XENIX, GENIX, PC/IX and PC-DOS are available which understand the common partitioning scheme.

## 6.3 USING THE MAXPART PROGRAM

To define (or redefine) a partition simply move the reverse video cursor to the appropriate column and specify the cylinder number of the parti-

tion's origin and its length in cylinders. Up to 8 partitions may be specified for each MAXTOR drive. The "Partition Id" shown at the left is merely a reference tag and bears no particular relationship to the drive designators used by PC-DOS.

Within a particular field the **backspace** key may be used to undo your keystrokes. Pressing the **Escape** key will allow you to exit from the program without applying any of the changes you've made to the Partition Table.

You may change the size, origin or number of partitions at any time, but, in general, each time you do so, you will lose access to the data formerly stored there. Also, each time you update the Partition Table will necessitate a re-boot in order for the changes to take effect. Follow this by running the appropriate formatting utility for your host operating system (**mkfs** for UNIX systems, FDISK and MAXFMT or FORMAT for PC-DOS).

## 6.4 MAXTOR PARTITIONS AND PC-DOS

Each MAXTOR partition that you designate to be accessible to PC-DOS appears as a separate drive. You must use the DOS FDISK utility to create a "DOS Partition" on each of these logical drives. DOS's partitioning scheme operates within the space available in the logical drive, and is a subordinate level of partitioning. Stated another way: the standard PC-DOS Fixed Disk Architecture is supported within each MAXTOR PC-DOS partition.

To conform with the limitations of DOS, MAXPART restricts each PC-DOS partition to a maximum of 128 cylinders (approximately 16 megabytes) and does not allow DOS partitions to overlap each other.

# 7.0 THE MAXFMT PROGRAM

## 7.1 SYNTAX

| Command | Operands |
|---|---|
| maxfmt | d: [/s] [/v] |

Where "d:" is the drive identifier assigned by DOS to the MAXTOR partition you wish to format. See "Device Address Considerations" on page 23 for instructions on how to determine this value.

The /s and /v options are the same as those provided by the DOS FORMAT command. Refer to the DOS manual for further explanation.

## 7.2 MAXFMT VERSUS FORMAT

The MAXFMT program is ostensibly identical to the DOS FORMAT utility, except that it will only format MAXTOR DOS partitions. You cannot use it to format your diskettes or IBM fixed disk drives. In fact, if you are using the MAXTOR Boot ROM to boot DOS from your MAXTOR, you cannot use it to format the first two MAXTOR DOS partitions either. In all of these cases you must use the standard DOS FORMAT command instead.

The reason for the MAXFMT utility is that the DOS FORMAT program will not properly format partitions accessible only via an installed block device driver. If you are booting from a MAXTOR, then the device driver provides access to MAXTOR partitions 3 through 16 (potentially); if you are booting from an IBM fixed disk, the MAXTOR driver provides access to all of the MAXTOR partitions (potentially 1 through 16). The difference is that if you use the boot ROM to boot from the MAXTOR drive, the built-in DOS device driver (and NOT the MAXTOR device driver) supports access to the first two MAXTOR DOS partitions.

The MAXFMT utility (like the DOS FORMAT command) creates the underlying structures needed for the DOS file system. In particular it creates a boot record, two File Allocation Tables (FATs), and the root directory. It also performs a verify operation on each track to determine whether unusable areas exist, and if it finds any these are flagged as inaccessible in the File Allocation Tables. If requested, MAXFMT will also transfer DOS system files to the MAXTOR and/or write a volume label.

Note that the surface analysis and bad block reassignment functions of MAXINIT should guarantee that all tracks will successfully pass the verify operation of MAXFMT. To state this another way, the MAXFMT program should never discover unusable areas unless you have bypassed the surface analysis functions of MAXINIT or unless your MAXTOR has developed a problem. This latter case may be indicative of imminent hardware failures, so be suspicious if you notice this. (You can run the DOS CHKDSK program after MAXFMT to determine if any unusable areas were found).

One point to be aware of is that you cannot run the MAXFMT utility without first running the DOS FDISK program to define how much of the space available on the device can be used by DOS. This step is required because each MAXTOR "partition" appears as a complete fixed disk drive to DOS, and as such it may hold up to 4 "subpartitions". The operation of the DOS FDISK program is documented in the DOS manual.

## 8.0 BOOTING FROM A MAXTOR

The MAXBOOT.SYS program is an extension to the PC BIOS power-on initialization which was designed to reside in a ROM chip on a PC motherboard or adapter card and provide the functions necessary to boot a DOS partition on MAXTOR drive 1. The MAXBOOT ROM occupies 2K and may reside anywhere between addresses C8000 and F0000 when using an XT, or between addresses C8000 and F6000 when using an IBM PC. The PC Expansion Unit ROM upgrade is also required for PC systems.

If you plan to use the MAXTOR and one or more IBM fixed disks you cannot use the MAXBOOT ROM. With this configuration you must continue to boot from the IBM Fixed Disk, rather than the MAXTOR. The reason for this is that IBM does not provide an installable block device driver for the IBM Fixed Disk. This device is only supported via the built-in DOS fixed disk driver. If you boot from a MAXTOR, the MAXBOOT program causes the built-in DOS fixed disk driver to access the MAXTOR, which would leave the IBM Fixed Disk inaccessible. Consequently, you should use the MAXBOOT support only if your system does not also have an IBM Fixed Disk.

The MAXBOOT program always boots the first MAXTOR partition on drive 1 designated as accessible to PC-DOS. Once DOS is booted, up to one additional MAXTOR partition (on drive 1 only) can be supported by the DOS built-in fixed disk driver. Additional MAXTOR DOS partitions must be supported via the MAXTOR device driver (MAXDD.SYS).

In order to initialize a bootable DOS partition on the MAXTOR you should first prepare a bootable DOS diskette with a CONFIG.SYS file with an entry for MAXDD.SYS. Boot from this diskette in order to run the MAXINIT and MAXPART utilities. Use the standard DOS FDISK and FORMAT commands to complete the initialization of the boot partition and the next DOS partition on the same drive (if any). Subsequent DOS partitions must be initialized using FDISK and MAXFMT.

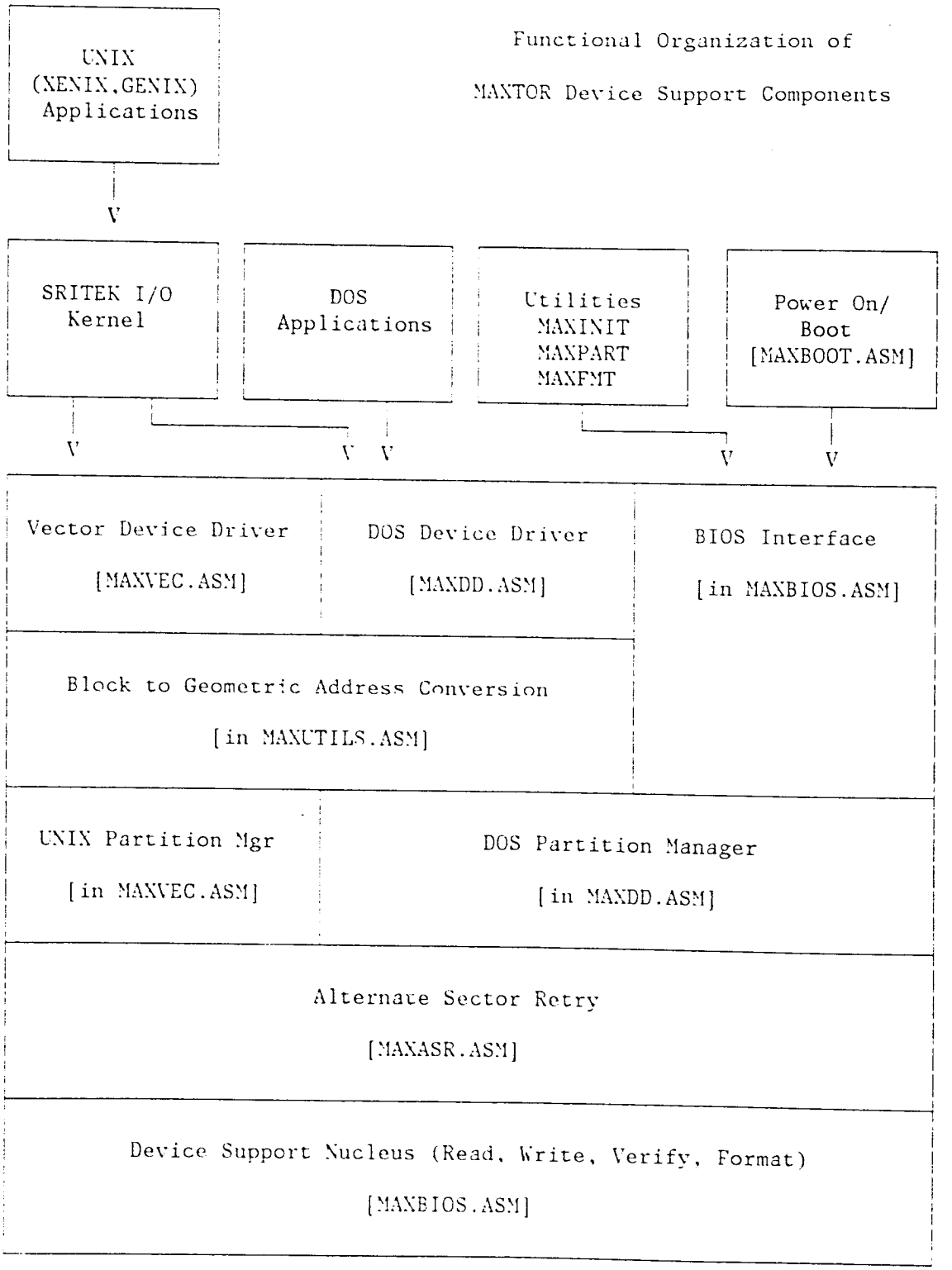
# A.0 PROGRAM ORGANIZATION

The diagram which follows should serve as a roadmap to anyone needing to modify the MAXTOR source code. The support functions have been carefully layered to maximize the applicability and transparency of the code in multiple operating system contexts. Furthermore each layer isolates those above it from the complexities and implementation specifics of the underlying layers. For example, none of the code above the lowest layer in the diagram is aware of the specifics of the Zobex File Controller. Similarly the alternate sector retry support is completely transparent to all of the code above it.

As a matter of packaging convenience, the source files do not correspond on a one-for-one basis to the depicted layers. Consequently the diagram indicates which source module contains each of the boxed components.

Functional Organization of

MAXTOR Device Support Components



# B.0 DEVICE ADDRESS CONSIDERATIONS

PC-DOS partitions on your MAXTOR are assigned a drive designator when DOS is booted. The designators are assigned sequentially, first to diskette drives, then any fixed disks known to the built-in DOS fixed disk driver, and then to installed block device drivers, according to the number of units each one supports. On a single diskette drive system, both A: and B: are reserved for the diskette driver, so the first fixed disk would have the designator C:.

Here's how to determine the drive designator of the first MAXTOR DOS partition:

- If your system has 1 or 2 diskette drives start with the letter C; if you have 3 diskette drives start with D, and for 4 start with E.
- Now, for each IBM fixed disk advance one letter.
- Advance one letter for each unit supported by block device drivers which are installed prior to the MAXTOR device driver.
- Whichever letter you have attained is the drive designator of the first MAXTOR DOS partition. Subsequent MAXTOR DOS partitions are assigned consecutive letters.

The standard BIOS programming interface (int 13h) is also available for access to the MAXTOR. This interface requires a device address rather than a drive designator. Device addresses are assigned as follows:

- 0-3 are used for diskette drives (up to 4 possible).
- 80 (hex) is the first fixed disk device.

  If you have no IBM fixed disk, this will be the first DOS partition listed in the MAXTOR partition table. If you have one or two IBM fixed disks, then 82 (hex) will be the device address of the first MAXTOR DOS partition.

  Notice that even if you only have a single IBM fixed disk device address 81 is skipped (reserved). This skipping of device 81 will be apparent if you run the DOS FDISK program. FDISK refers to fixed disks as drive 1, drive 2, drive 3, etc. These correspond directly to device addresses 80, 81, 82, etc. If your system has a MAXTOR and only a single IBM fixed disk, you will need to "skip" over drive 2, since this device will not exist.

  Since you may have up to 2 MAXTOR drives attached, and up to 8 DOS partitions on each MAXTOR, their device addresses may range from 82 through 91, or 80 through 8F (if you have no IBM fixed disks).

MAXTOR partitions which are not designated as accessible to PC-DOS are unaddressable using this device addressing scheme, except as follows. Two special device address are reserved for unpartitioned access to the MAXTOR. A0 refers to physical MAXTOR drive 1 and A1 refers to physical MAXTOR drive 2. Using these device addresses, and the BIOS programming interface, will permit unrestricted read or write access to any sector on either MAXTOR drive. These addresses are used by the MAXPART and MAXINIT utility programs for maintaining the partition table and the alternate sector table.

## C.1 IOCTL FUNCTIONS FOR THE DOS DEVICE DRIVER

These functions are available using int 21h, subfunction 44h (IOCTL).

### C.1.1 GET DRIVE PARAMETERS

This function will return device specific information for the designated MAXTOR drive, including its BIOS device address, the number of cylinders in its DOS FDISK partition, the cylinder number of the origin for the DOS FDISK partition, and the number of blocks accessible using the DOS absolute read and write functions (int 25h and int 26h).

The calling registers should be set as follows:

| | | |
|---|---|---|
| AX | 4404h | (Read bytes from control channel) |
| BL | Drive number | (0 = default, 1 = A, etc.) |
| CX | 7 | (number of bytes to return) |
| DX | Offset address of 7-byte return buffer | |
| DS | Segment address of 7-byte return buffer | |

If the carry flag is on after the int 21h, it indicates that the requested drive is not supported by the MAXTOR device driver and no information is placed in the buffer area. Notice that when booting from a MAXTOR, the boot partition and the next DOS partition on the same MAXTOR drive (if any), are supported directly by the built-in DOS fixed disk driver and not the MAXTOR device driver.

The layout of the information returned in the buffer when the carry flag is off, is as follows:

| offset | type | contents |
|---|---|---|
| 0 | byte | BIOS device address |
| 1 | word | number of cylinders in FDISK DOS partition |
| 3 | word | starting cylinder of FDISK DOS partition |
| 5 | word | number of blocks in FDISK DOS partition |

### C.1.2 SET ALTERNATE SECTOR RETRY ON OR OFF

This function will deactivate or reactivate the alternate sector retry logic. It is used primarily by the MAXINIT program.

The calling registers should be set as follows:

| | | |
|---|---|---|
| AX | 4405h | (Write bytes to control channel) |
| BL | MAXTOR drive | (0 = default, 1 = A, etc.) |
| CX | 1 | (number of bytes passed) |
| DX | Offset address of 1-byte buffer area | |
| DS | Segment address of 1-byte return area | |

If the carry flag is on after the int 21h, it indicates that the requested drive is not supported by the MAXTOR device driver and no action has occurred.

The layout of the information passed in the buffer is as follows:

| offset | type | contents |
|---|---|---|
| 0 | byte | 1 = Set Alternate Sector Retry Off |
| | | 0 = Set Alternate Sector Retry On |

## C.2 EXTENSIONS TO THE BIOS INTERFACE

The MAXTOR device driver emulates the existing fixed disk BIOS interface for MAXTOR PC-DOS partitions (which each have a unique device address), and supplies some new functions as well. The new functions are described here.

### C.2.1 FORMAT A MAXTOR TRACK

Input parameters should be assigned to registers as follows:

| | |
|---|---|
| CH | Cylinder number (low order 8 bits) |
| CL | Cylinder number (2 high bits as XX000000) |
| DH | Head (track) number |
| DL | Device address |
| BX | Offset address of Bad Block Array |
| ES | Segment address of Bad Block Array |

The Bad Block Array is a sequence of 17 bytes, each one corresponding to a sector (number from 1 through 17). A non-zero value in a byte indicates that the corresponding sector is defective.

The following calling sequence is used:

```
mov     AH,16h
int     13h
```

Following the operation the carry flag on to indicate a failure, or off for success. Status information is returned in the AX register.

## C.2.2 QUERY IF MAXTOR SUPPORT INSTALLED

This function may be used to determine whether the MAXTOR support software is installed (either as MAXBOOT.SYS or MAXDD.SYS), and if so, how many physical MAXTOR drives are attached.

The calling sequence is as follows:

```
mov     AH,17h
int     13h
```

If the carry flag is off the MAXTOR support is installed and the number of physical drives is returned in register AX.

Formatted by SCRIPT from file 'MAXDATA PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
                .xlist
;-------------------------------------------------------------------------------
;
;       Common Data Declarations for the MAXTOR Support Routines
;       Program Property of IBM
;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1
;       Version:        1.6 05/23/84
;       DOS Required:   2.0 or later
;       Classification: IBM Internal Use Only
;
;       (C) Copyright IBM Corporation, 1984
;       Developed for the Information Technology Center at
;       Carnegie-Mellon University
;
;-------------------------------------------------------------------------------
                .list

;       MAXTOR Device Constants
;
MAXDRIVES       equ     2               ;Total physical drives
MAXCYLS         equ     918             ;Total cylinders per drive
MAXHEADS        equ     15              ;Total heads per cylinder
MAXSECS         equ     17              ;Total sectors per head
MAXGAPSIZE      equ     16              ;Gap1 and Gap3 size for format
SECSIZE         equ     512             ;Sector size in bytes

;       Location of Partition Record
;
PARTRECCYL      equ     917             ;Partition record is kept
PARTRECHD       equ     0               ; at Cyl 917 Hd 0 Sec 0
PARTRECSEC      equ     0

;       Location of Alternate Sector Table
;
ASTCYL          equ     917             ;Alternate Sector Table is kept
ASTHD           equ     0               ; at Cyl 917 Hd 0 Sec 1
ASTSEC          equ     1
ASTSECCNT       equ     2

;       Alternate Sector Table
;
AST             segment at 0
ASTSignature    db      4 dup(?)        ;Signature ['AST',0]
ASTDatacyl      dw      ?               ;Cylinder with alternate sectors
ASTBBCount      dw      ?               ;Number of entries in ASTBBArray
ASTBBArray      equ     $               ;Bad Block Array starts here
  ASTbadcyl     dw      ?               ;  Cylinder of bad block
  ASTbadhd      db      ?               ;  Head of bad block
  ASTbadsec     db      ?               ;  Sector of bad block
AST             ends
```

```
;       BIOS Data Areas
;
BiosData        segment at 40h
                org     74h
StatusByte      db      ?               ;Status from last operation
  BIOS_TIMEOUT          equ     80h
  BIOS_CTLRFAILED       equ     20h
  BIOS_BADECC           equ     10h
  BIOS_BADTRACK         equ     0Bh
  BIOS_INITFAILED       equ     07h
  BIOS_RECNOTFOUND      equ     04h
  BIOS_NOADDRMARK       equ     02h
  BIOS_BADCOMMAND       equ     01h

                org     0A0h
OrigDiskReqOff  dw      ?               ;Entry point to original
OrigDiskReqSeg  dw      ?               ; BIOS Disk Request Handler
TerminateExit   dd      ?               ;User exit for forced abort
LogicalDisks    db      ?               ;Total MAXTOR logical disks
PhysicalDrives  db      ?               ;Number of MAXTORs attached
Count           db      ?               ;Sectors to read/write/verify
Drive           db      ?               ;Current Drive
Head            db      ?               ;Current Head
Sector          db      ?               ;Current Sector
Cyl             dw      ?               ;Current Cylinder
Partitioner     dd      ?               ;Exit for Partitioning Routine
MaxBase         db      ?               ;Address of 1st MAXTOR drive
  MAXBOOT       equ     80h             ;Value of MaxBase when booting
                                        ; from the MAXTOR [else 82h]
BiosData        ends
```

```
                .xlist
;-------------------------------------------------------------------------;
;                                                                         ;
;       PC-DOS Request Header Declarations for Device Drivers             ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1         ;
;       Version:        2.0 05/24/84                                      ;
;       DOS Required:   2.0 or later                                      ;
;       Classification: IBM Internal Use Only                             ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------;
                .list

;       Common Request Header
;
RH_length       equ     byte ptr DS:[SI+0]      ;Length of req header in bytes
RH_drive        equ     byte ptr DS:[SI+1]      ;Unit number
RH_command      equ     byte ptr DS:[SI+2]      ;Command code
RH_status       equ     word ptr DS:[SI+3]      ;Status word
RH_reserved     equ     byte ptr DS:[SI+5]      ;8 reserved bytes
RH_data         equ     byte ptr DS:[SI+13]     ;Start of data area

;       Request Header Fields for Read/Write Commands
;
RH_mediadesc    equ     byte ptr DS:[SI+13]     ;Media descriptor byte
RH_dtaoff       equ     word ptr DS:[SI+14]     ;Offset  of disk transfer area
RH_dtaseg       equ     word ptr DS:[SI+16]     ;Segment of disk transfer area
RH_blockcount   equ     word ptr DS:[SI+18]     ;Block count
RH_startblock   equ     word ptr DS:[SI+20]     ;Starting block number

;       Request Header Fields for Media Check Command
;
RH_medchkstatus equ     byte ptr DS:[SI+14]     ;Return value from Media Check

;       Request Header Fields for Build BPB Command
;
RH_bpboffptr    equ     word ptr DS:[SI+18]     ;Offset  pointer to BPB
RH_bpbsegptr    equ     word ptr DS:[SI+20]     ;Segment pointer to BPB

;       Request Header Fields for Initialize Command
;
RH_unitcount    equ     byte ptr DS:[SI+13]     ;Number of drives
RH_endoffptr    equ     word ptr DS:[SI+14]     ;Offset  addr of end of driver
RH_endsegptr    equ     word ptr DS:[SI+16]     ;Segment addr of end of driver
RH_bpbarrayoff  equ     word ptr DS:[SI+18]     ;Offset  addr of BPB array
RH_bpbarrayseg  equ     word ptr DS:[SI+20]     ;Segment addr of BPB array
```

```
;       Status Flags and Error Codes
;
RH_DONE         equ     0100h           ;Operation is complete
RH_IOERROR      equ     8000h           ;I/O error occurred
RH_NOTREADY     equ     02h             ;Device not ready
RH_BADCOMMAND   equ     03h             ;Unsupported command
RH_SEEKERROR    equ     06h             ;Sector address out of range
```

```
                .xlist
;-------------------------------------------------------------------------;
;                                                                         ;
;       Definitions for ZOBEX File Controller                             ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1         ;
;       Version:        1.6 05/23/84                                      ;
;       DOS Required:   2.0 or later                                      ;
;       Classification: IBM Internal Use Only                             ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------;
                .list

;       Register definitions
;
ZFC_BASE        equ     0250h           ;Base of I/O ports              In Out
ZFC_DATAREG     equ     ZFC_BASE+0      ;Data register                   Y  Y
ZFC_ERRORREG    equ     ZFC_BASE+1      ;Error register                  Y  N
ZFC_WPRECOMPREG equ     ZFC_BASE+1      ;Write precompensation register  N  Y
ZFC_SECCNTREG   equ     ZFC_BASE+2      ;Sector count register           Y  Y
ZFC_SECTORREG   equ     ZFC_BASE+3      ;Sector address register         Y  Y
ZFC_LOWCYLREG   equ     ZFC_BASE+4      ;Cylinder address - low byte     Y  Y
ZFC_HIGHCYLREG  equ     ZFC_BASE+5      ;Cylinder address - high byte    Y  Y
ZFC_SDHREG      equ     ZFC_BASE+6      ;Size/drive/head register        Y  Y
ZFC_CMDSTATREG  equ     ZFC_BASE+7      ;Command/status register         Y  Y
ZFC_EXTREG      equ     ZFC_BASE+8      ;External control/status register Y  Y

;       1 - Error register mask definitions:
;
ERR_NOADDRMARK  equ     01h             ;Data address mark not found
ERR_TRACKZERO   equ     02h             ;Track 0 error
ERR_CMDABORTED  equ     04h             ;Aborted command
ERR_IDNOTFOUND  equ     10h             ;ID not found
ERR_DATACRC     equ     40h             ;CRC error - data field
ERR_BADBLOCK    equ     80h             ;Bad block detected
ERR_UNUSED      equ     28h             ;Unused error bits
ERR_TIMEOUT     equ     28h             ;Used by BIOS to mean timeout occurred

;       6 - SDH register mask definitions:
;
SDH_ECCENABLE   equ     80h             ;Enable ECC

        ife SECSIZE-512                 ;SECSIZE is defined in maxdata.dcl
SDH_SECSIZE     equ     20h             ;512 byte sectors
        else
          ife SECSIZE-1024
SDH_SECSIZE     equ     40h             ;1024 byte sectors
```

```
          else
            ife SECSIZE-256
SDH_SECSIZE     equ     60h             ;256 byte sectors
            else
              ife SECSIZE-128
SDH_SECSIZE     equ     00h             ;128 byte sectors
              endif
            endif
          endif
        endif

;       7 - Command definitions:
;
CMD_RESTORE     equ     10h             ;Restore drive to track 0 command
;                                       ; [low order 4 bits is step rate]
CMD_READSEC     equ     20h             ;Read sector command
;                                       ; [01 bit on = retry disabled]
;                                       ; [04 bit on = transfer multiple secs]
;                                       ; [08 bit on = interrupt at end of cmd]
CMD_WRITESEC    equ     30h             ;Write sector command
;                                       ; [01 bit on = retry disabled]
;                                       ; [04 bit on = transfer multiple secs]
CMD_SCANID      equ     40h             ;Scan for next valid ID command
CMD_FORMAT      equ     50h             ;Format track command
CMD_SEEK        equ     70h             ;Seek command
;                                       ; [low order 4 bits is step rate]

;       7 - Status register mask definitions:
;
STA_BUSY        equ     80h             ;Busy
STA_DRIVEREADY  equ     40h             ;Drive ready
STA_WRITEFAULT  equ     20h             ;Write fault
STA_SEEKDONE    equ     10h             ;Seek complete
STA_DATAREQ     equ     08h             ;Data request
STA_CORRECTED   equ     04h             ;Corrected error
STA_WRITPROT    equ     02h             ;Drive write protected
STA_ERROR       equ     01h             ;Hard error

;       8 - External command/status register mask definitions:
;
EXT_SELENABLE   equ     04h             ;Drive select enable on
EXT_SELDISABLE  equ     00h             ;Drive select enable off
EXT_HIGHHEAD    equ     01h             ;High order head address bit

;       Miscellaneous Zobex Constants
;
BADBLOCKFLAG    equ     80h             ;Identifies bad blocks to CMD_FORMAT
```

Formatted by SCRIPT from file 'MAXASR PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
                page    56,132
                title   'Alternate Sector Retry for MAXTOR 1140 Hard Disk'

;-----------------------------------------------------------------------;
;                                                                       ;
;       Alternate Sector Retry for MAXTOR 1140 Hard Disk Support        ;
;       Program Property of IBM                                         ;
;                                                                       ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1       ;
;       Version:        2.2 06/11/84                                    ;
;       DOS Required:   None                                            ;
;       Classification: IBM Internal Use Only                           ;
;                                                                       ;
;       (C) Copyright IBM Corporation, 1984                             ;
;       Developed for the Information Technology Center at              ;
;       Carnegie-Mellon University                                      ;
;                                                                       ;
;-----------------------------------------------------------------------;

                include maxdata.dcl     ;Data seg for MAXTOR BIOS

                public  ObtainAST
                public  RecoveryViaAST

Cseg            segment public 'code'
                assume  CS:Cseg,SS:nothing

IBMCopyright    segment byte common 'IBMCopyright'
                db      ' (C) Copyright IBM Corporation, 1984 '
IBMCopyright    ends

                extrn   ReadSectors:near

AST1seg         dw      0               ;Segment address of AST for drive 0
AST1off         dw      0               ;Offset  address of AST for drive 0
AST2seg         dw      0               ;Segment address of AST for drive 1
AST2off         dw      0               ;Offset  address of AST for drive 1
ASTHeader       db      'AST',0         ;AST Signature value

;-----------------------------------------------------------------------;
;                                                                       ;
;       ObtainAST       Place AST at Specified Memory Address           ;
;                                                                       ;
;       Input   AL      Physical drive number for AST                   ;
;               DI      Offset  address for AST                         ;
;               ES      Segment address for AST                         ;
;               DS      BIOS data segment                               ;
;                                                                       ;
;       Output  none    AST read from drive, or null copy created       ;
;                                                                       ;
;-----------------------------------------------------------------------;
```

```
ObtainAST       assume  DS:nothing,ES:nothing
                proc    near
                push    AX              ;Save registers
                push    BX
                push    CX
                push    DX

                push    AX              ;Save drive number
                push    DI              ;Save AST offset pointer
                mov     AH,ASTSECCNT
                mov     BH,ASTHD
                mov     BL,ASTSEC
                mov     CX,ASTCYL
                call    ReadSectors     ;Try to read AST
                pop     DI              ;Restore AST offset pointer
                pop     BX              ;Drive number to BL

                mov     AX,word ptr ASTHeader
                cmp     ES:[DI],AX      ;Check AST signature
                jne     oa1             ;No good
                mov     AX,word ptr ASTHeader+2
                cmp     ES:[DI+2],AX
                je      oa2             ;AST is alright as is

oa1:            cld                     ;Initialize a new AST
                mov     AX,word ptr ASTHeader
                stosw
                mov     AX,word ptr ASTHeader+2
                stosw
                mov     ES:ASTBBCount[DI],0
                mov     ES:ASTDatacyl[DI],ASTCYL
                mov     CX,SECSIZE-4
                xor     AX,AX
                lea     DI,byte ptr ES:ASTBBArray[DI]
                rep     stosw

oa2:            cmp     BL,0            ;Drive 0?
                jne     oa3
                mov     AST1seg,ES
                mov     AST1off,DI
                jmp     short oa9

oa3:            cmp     BL,1            ;Drive 1?
                jne     oa9             ;No, should not occur
                mov     AST2seg,ES
                mov     AST2off,DI

oa9:            pop     DX              ;Restore regs
                pop     CX
                pop     BX
                pop     AX
```

```
                ret
ObtainAST       endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       RecoveryViaAST  Recovery using Alternate Sector Table           ;
;                                                                       ;
;       The primary rationale for the design of this routine was to     ;
;       minimize stack usage, allowing it to be invoked from within     ;
;       low level MAXBIOS routines.                                     ;
;                                                                       ;
;       Input   BX      Offset  address for retry operation             ;
;               CX      Segment address for retry operation             ;
;               DS      BIOS Data Segment                               ;
;                                                                       ;
;       Output  Carry   OFF if recovery successful                      ;
;                       ON  if recovery failed                          ;
;                                                                       ;
;-----------------------------------------------------------------------;

;       Local Constants for RecoveryViaAST
;
SecsPerHd       db      MAXSECS

;       Local Storage for RecoveryViaAST
;
rvarecursion    db      0
rvasavcnt       db      0
rvasavhd        db      0
rvasavsec       db      0
rvasavcyl       dw      0
rvasaves        dw      0
rvasavdi        dw      0
rvasavsi        dw      0
rvasavax        dw      0
rvasavdx        dw      0
rvaretry        dd      0

                assume  DS:BiosData,ES:nothing
RecoveryViaAST  proc    near
                cmp     rvarecursion,0  ;Recursive entry?
                je      rva0            ;No, continue
                jmp     rva8            ;Yes, exit with error

        rva0:   mov     rvarecursion,1  ;Lockout subsequent entry
                mov     rvasavax,AX     ;Save AX
                mov     rvasavdx,DX     ; and DX
                mov     word ptr rvaretry,BX    ;Save retry entry point
                mov     word ptr rvaretry+2,CX

                mov     AL,Count        ;Save current I/O parameters
                mov     rvasavcnt,AL
```

```
        mov     AL,Head
        mov     rvasavhd,AL
        mov     AL,Sector
        mov     rvasavsec,AL
        mov     AX,Cyl
        mov     rvasavcyl,AX
        mov     rvasaves,ES
        mov     rvasavdi,DI
        mov     rvasavsi,SI

        cmp     Drive,0         ;Drive 0?
        jne     rva1            ;No, check drive 1
        mov     AX,AST1seg      ;Yes, get pointers for first AST
        mov     DI,AST1off
        jmp     short rva2

rva1:   cmp     Drive,1         ;Any other drive but 1?
        je      rva1a           ;No, continue
        jmp     rva7            ;Yes, should not occur

rva1a:  mov     AX,AST2seg      ;Get pointers to second AST
        mov     DI,AST2off

rva2:   cmp     AX,0            ;Does appropriate AST exist?
        je      rva7            ;No, cannot recover
        mov     ES,AX           ;Yes, establish addressability

        assume  ES:AST
        mov     DX,ASTSEC+ASTSECCNT ;1st possible alt. sec (0-relative)
        mov     CX,ASTBBCount[DI]
        jcxz    rva7            ;Error, null AST
        mov     SI,ASTDatacyl[DI]
        mov     AX,rvasavcyl    ;Load search arguments
        mov     BL,rvasavhd
        mov     BH,rvasavsec
        inc     BH              ;Table has 1st sector as 1

rva3:   cmp     AX,ASTbadcyl[DI];Look for a match in AST
        jne     rva4
        cmp     BX,word ptr ASTbadhd[DI]
        je      rvahit          ;Found one!

rva4:   add     DI,4            ;Advance to next entry in ASTBBArray
        inc     DX              ;Bump alternate sector number
        loop    rva3            ;Check each possible alternate sector
        jmp     short rva7      ;No alternate sector assigned

rvahit: mov     AX,DX           ;Convert sector number to hd & sector
        div     SecsPerHd       ;AH = sector number, AL = head number

        mov     BH,AL           ;Set head number
        mov     BL,AH           ;Set Sector number
```

```
                mov     CX,SI           ;Set cylinder number
                mov     AL,Drive        ;Set drive number
                mov     AH,1            ;Indicate 1 sector only
                mov     DI,rvasavdi     ;Restore buffer offset  address
                mov     ES,rvasaves     ;Restore buffer segment address

                call    rvaretry        ;Far call to alternate sector retry
                jmp     short rvaret

        rva7:   mov     AX,rvasavax     ;Restore AX
                mov     DX,rvasavdx     ; and DX, which have correct error codes
                mov     SI,rvasavsi     ;Restore SI
                mov     DI,rvasavdi     ;Restore buffer offset  address
                mov     ES,rvasaves     ;Restore buffer segment address
                mov     rvarecursion,0  ;Allow reentry
                jmp     short rva8

        rvaret: mov     AL,rvasavcnt    ;Restore saved I/O parameters
                mov     Count,AL
                mov     AL,rvasavhd
                mov     Head,AL
                mov     AL,rvasavsec
                mov     Sector,AL
                mov     AX,rvasavcyl
                mov     Cyl,AX

                mov     SI,rvasavsi     ;Restore SI
                mov     DX,rvasavdx     ;Restore DX
                mov     rvarecursion,0  ;Allow reentry to this routine
                jmp     short rva9      ;Return with carry from retry operation

        rva8:   stc                     ;Indicate recovery failed

        rva9:   ret                     ;Return to caller
RecoveryViaAST  endp

Cseg            ends
                end
```

Formatted by SCRIPT from file 'MAXBIOS PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
                page    56,132
                title   'ITC BIOS Driver for MAXTOR 1140 Hard Disk'

;-----------------------------------------------------------------------------;
;                                                                             ;
;       BIOS Support Routines for the MAXTOR 1140 Hard Disk                   ;
;       Program Property of IBM                                               ;
;                                                                             ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1             ;
;       Version:        2.2 05/24/84                                          ;
;       DOS Required:   None                                                  ;
;       Classification: IBM Internal Use Only                                 ;
;                                                                             ;
;       (C) Copyright IBM Corporation, 1984                                   ;
;       Developed for the Information Technology Center at                    ;
;       Carnegie-Mellon University                                            ;
;                                                                             ;
;-----------------------------------------------------------------------------;

                include maxdata.dcl             ;BIOS data areas and constants
                include maxzfc.dcl              ;Zobex file controller info

                public  DeselectDrive
                public  ReadSectors
                public  RequestRouter
                public  ResetDisk
                public  SelectDrive
                public  SpecialRead
                public  VerifySectors
                public  WriteSectors
                public  EndOfMAXTORBIOS

Cseg            segment public 'code'
                assume  CS:Cseg,SS:nothing

IBMCopyright    segment byte common 'IBMCopyright'
                db      ' (C) Copyright IBM Corporation, 1984 '
IBMCopyright    ends

;       Local constants
;
WPRECOMPVAL     equ     230             ;1st cyl requiring write precompensation
STEPRATE        equ     0               ;Fastest possible step rate
SLOWRATE        equ     0Fh             ;Longest possible step rate

;       Operation Vector Table
;
LASTOP          equ     17h
OperationTable  label   word
                dw      ResetDisk       ;00h    Reset
                dw      GetStatus       ;01h    Status
                dw      ReadSectors     ;02h    Read Sectors
```

```
                dw      WriteSectors    ;03h    Write Sectors
                dw      VerifySectors   ;04h    Verify Sectors
                dw      UnimplementedOp ;05h    Format Track
                dw      UnimplementedOp ;06h    Format with Bad Sector Flags
                dw      UnimplementedOp ;07h    Format Drive from Track x
                dw      GetParameters   ;08h    Get Parameters
                dw      UnimplementedOp ;09h    Initialize Drive Characteristics
                dw      UnimplementedOp ;0Ah    Read Long  [Data + ECC]
                dw      UnimplementedOp ;0Bh    Write Long [Data + ECC]
                dw      UnimplementedOp ;0Ch    Seek
                dw      ResetDisk       ;0Dh    Alternate Reset
                dw      UnimplementedOp ;0Eh    Read Sector Buffer
                dw      UnimplementedOp ;0Fh    Write Sector Buffer
                dw      UnimplementedOp ;10h    Test Drive Ready
                dw      UnimplementedOp ;11h    Recalibrate
                dw      UnimplementedOp ;12h    Controller RAM Diagnostic
                dw      UnimplementedOp ;13h    Drive Diagnostic
                dw      UnimplementedOp ;14h    Controller Internal Diagnostic
                dw      UnimplementedOp ;15h    DASD Type

;       Following are operations unique to the MAXTOR
;
                dw      NativeFormat    ;16h    Format with interleave table
                dw      MaxInstalled    ;17h    Indicates this BIOS installed

BiosDataSeg     dw      seg BiosData

savebp          equ     word ptr [BP-0]
saveax          equ     word ptr [BP-2]
saveah          equ     byte ptr [BP-1]
savebx          equ     word ptr [BP-4]
savecx          equ     word ptr [BP-6]
savedx          equ     word ptr [BP-8]
savesi          equ     word ptr [BP-10]
savedi          equ     word ptr [BP-12]
saveds          equ     word ptr [BP-14]
savees          equ     word ptr [BP-16]

;-------------------------------------------------------------------------------
;                                                                              ;
;       RequestRouter   Request Router for MAXTOR BIOS support                 ;
;                                                                              ;
;       Input   AH      Operation code                                         ;
;               AL      Sector count                                           ;
;               CH      8 low bits of logical cylinder number                  ;
;               CL      2 high bits of cylinder plus 6 bit sector number       ;
;               DH      Head number                                            ;
;               DL      Drive number [80h-xxh]                                 ;
;               BX      buffer offset  address                                 ;
;               ES      buffer segment address                                 ;
;                                                                              ;
;       Output  Carry   ON=error OFF=no error                                  ;
```

```
;               AH      Error code if carry on                          ;
;               AL      Destroyed                                       ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
RequestRouter   proc    far
                push    DS              ;Save DS temporarily
                mov     DS,BiosDataSeg
                cmp     AH,8            ;Get Parms?
                je      r1              ;Yes, always intercept
                cmp     AH,0            ;Reset?
                je      r2              ;Yes, always intercept
                cmp     AH,17h          ;Query if MAXTOR Support Installed?
                je      r2              ;Yes, always intercept

                cmp     DL,MaxBase      ;One of our drives?
                jae     r2              ;Yes, handle request

        r0:     push    BP              ;No, just pass it thru to original BIOS
                mov     BP,SP
                push    OrigDiskReqSeg
                push    OrigDiskReqOff
                mov     DS,word ptr [BP+2]
                mov     BP,word ptr [BP+0]
                ret     4               ;Enter original BIOS Disk Request code

        r1:     cmp     DL,80h          ;In fixed disk range?
                jb      r0              ;No, pass it to original BIOS
                cmp     DL,MaxBase      ;One of ours?
                jae     r2              ;Yes, continue

                pushf                   ;Fake an iret frame for original BIOS
                call    dword ptr OrigDiskReqOff ;Pass to original BIOS
                jc      r1a
                mov     DL,LogicalDisks
                add     DL,2            ;Adjust number of drives value

        r1a:    pop     DS              ;Restore DS register
                ret     2               ;Return to caller with flags changed

        r2:     pop     DS              ;Restore original DS
                sti                     ;Enable interrupts
                push    BP
                mov     BP,SP           ;Establish frame pointer
                push    AX              ;Save AX for reference
                push    BX              ;Save registers during operation
                push    CX
                push    DX
                push    SI
                push    DI
                push    DS
                push    ES
```

```
        mov     DS,BiosDataSeg  ;Establish addressability to our data

        cmp     DL,MaxBase      ;Specific for MAXTOR?
        jae     r3              ;Yes, keep going
        mov     DL,0            ;No, use drive 0 while here
        jmp     short r4

r3:     sub     DL,MaxBase      ;Make drive number zero relative

r4      label   near
        mov     AH,AL           ;AH = count
        mov     AL,DL           ;AL = drive
        mov     DI,BX           ;DI = transfer offset
        mov     BH,DH           ;BH = head
        mov     BL,CL           ;BL = sector
        and     BL,03Fh         ;Remove high order 2 bits
        dec     BL              ;Make sector number zero relative

        xchg    CH,CL
        rept    6
        shr     CH,1            ;Align cylinder number
        endm

        cmp     saveah,17h      ;Installed query?
        je      r6              ;Yes, no partitioning
        cmp     saveah,8        ;Get Parms?
        jne     r5              ;No, continue
        xor     CX,CX           ;Yes, clear garbage in CX

r5:     clc                     ;In case Partitioner is an IRET,
        mov     DX,MAXCYLS      ; preset carry and DX
        pushf
        call    Partitioner     ;Apply partitioning if needed
        jc      re2             ;Partitioning error

        cmp     AL,PhysicalDrives
        jb      r6
        mov     AH,BIOS_TIMEOUT ;No such drive
        stc                     ;Set carry flag
        jmp     short re2       ;Return to caller

r6:     push    DX              ;Save output from Partitioner
        mov     DL,saveah       ;Use Operation as index
        cmp     DL,LASTOP       ;Is it in range?
        ja      re1             ;No, error
        xor     DH,DH
        shl     DL,1
        mov     SI,DX
        pop     DX              ;Restore DX from Partitioner
        call    SelectDrive     ;Enable drive select
        call    OperationTable[SI]
        call    DeselectDrive   ;Disable drive select
```

```
                jmp     short re2

        re1:    stc
                mov     AH,BIOS_BADCOMMAND      ;Indicate bad command

        re2:    pop     ES              ;Restore caller's registers
                pop     DS
                pop     DI
                pop     SI
                pop     DX
                pop     CX
                pop     BX
                pushf                   ;Save flags for a moment
                cmp     saveah,0        ;Reset?
                je      re3
                cmp     saveah,13       ;Alternate Reset?
                je      re3
                popf                    ;Restore flags
                pop     BP              ;Throw away caller's AX
                pop     BP              ;Restore caller's BP
                ret     2               ;Throw away saved flags and return

        re3:    popf                    ;Restore flags
                pop     AX              ;Restore caller's AX
                pop     BP              ;Restore caller's BP
                push    DS              ;Save caller's DS again
                mov     DS,BiosDataSeg  ;Restore addressability to BiosData
                jmp     r0              ;Now let original BIOS handle also
RequestRouter   endp

UnimplementedOp proc    near
                stc
                mov     AH,BIOS_BADCOMMAND      ;Indicate bad command
                ret
UnimplementedOp endp

;--------------------------------------------------------------------------
;                                                                         ;
;       GetParameters   Get configuration parameters and drive geometry info
;                                                                         ;
;       Input   AL      Drive number [0 thru PhysicalDrives-1]            ;
;               DX      Maximum accessible cylinders [from Partitioner]   ;
;               DS      BIOS Data Segment                                 ;
;                                                                         ;
;       Output  carry   ON=no such device, OFF=device exists              ;
;           [if carry then CX and DX = 0 and AH has error code]           ;
;               AX      Destroyed                                         ;
;               CH      Maximum useable cylinder number (8 low bits)      ;
;               CL      2 high bits of cylinder plus maximum sector number
;               DH      Maximum head number                               ;
```

```
;               DL      Maximum drive number                                    ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
GetParameters   proc    near
                cmp     AL,PhysicalDrives
                jb      gp1
                xor     CX,CX
                xor     DX,DX
                xor     AL,AL
                mov     AH,BIOS_INITFAILED
                mov     StatusByte,BIOS_INITFAILED
                stc                     ;Set carry [error]
                jmp     short gp3

        gp1:    mov     CX,DX           ;Return size based on output from
                jcxz    gp2             ;  partitioning exit
                dec     CX

        gp2:    xchg    CH,CL           ;Get 2 high order bits of cyl to CL
                mov     CL,MAXSECS      ;Insert number of sectors
                mov     DH,MAXHEADS-1   ;Insert number of heads
                mov     DL,LogicalDisks
                cmp     MaxBase,MAXBOOT ;Did we boot from MAXTOR?
                je      gp2a            ;Yes, skip next line
                add     DL,2            ;Include drives on IBM adapter
        gp2a    label   near

                xor     AX,AX
                clc                     ;No errors

        gp3:    mov     savecx,CX       ;Return info in CX
                mov     savedx,DX       ;  and DX
                ret                     ;Return with carry as appropriate
GetParameters   endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       GetStatus       Get status byte from previous operation                  ;
;                                                                               ;
;       Input   DS      BIOS Data Segment                                       ;
;                                                                               ;
;       Output  AL      status byte                                             ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
GetStatus       proc    near
                mov     AL,StatusByte   ;Get last status byte
                mov     StatusByte,0    ;Reset status info
                clc                     ;Clear carry flag
                ret
GetStatus       endp
```

```
;--------------------------------------------------------------------------;
;                                                                          ;
;       MaxInstalled    Indicate to Caller that this BIOS is Installed     ;
;                                                                          ;
;       Input   DS      BIOS Data Segment                                  ;
;                                                                          ;
;       Output  Carry   OFF = Installed                                    ;
;               AH      Zero                                               ;
;               AL      Number of physical drives attached                 ;
;                                                                          ;
;--------------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
MaxInstalled    proc    near
                xor     AH,AH           ;Zero AH
                mov     AL,PhysicalDrives
                clc                     ;Clear carry flag
                ret
MaxInstalled    endp

;--------------------------------------------------------------------------;
;                                                                          ;
;       ResetDisk       DASD Reset                                         ;
;                                                                          ;
;       Input   DS      BIOS Data Segment                                  ;
;                                                                          ;
;       Output  Carry   ON=error OFF=no error                              ;
;               AX-DX   Destroyed                                          ;
;                                                                          ;
;--------------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
ResetDisk       proc    near
                mov     AL,EXT_SELENABLE        ;Select, enable drive 0
                mov     DX,ZFC_EXTREG
                out     DX,AL
                mov     AL,SDH_SECSIZE+SDH_ECCENABLE    ;Set sec size
                mov     DX,ZFC_SDHREG           ;Select drive 0, head 0, ECC
                out     DX,AL
                mov     AL,WPRECOMPVAL
                mov     DX,ZFC_WPRECOMPREG
                out     DX,AL                   ;Init write precomp
                mov     AL,CMD_RESTORE+SLOWRATE ;Use longer rate than usual case
                mov     DX,ZFC_CMDSTATREG
                out     DX,AL           ;Issue slow home cmd to get to cyl 0
                call    WaitOnSeek      ;Wait for seek complete
                jc      rd9
                mov     AL,CMD_RESTORE+STEPRATE
                mov     DX,ZFC_CMDSTATREG
                out     DX,AL           ;Do another home to set proper rate
                call    WaitOnSeek      ;Wait for seek complete
                xor     AH,AH
                clc
```

```
                jmp     short rd9a

        rd9     label   near
                mov     AH,BIOS_INITFAILED

        rd9a:   ret
ResetDisk       endp

;--------------------------------------------------------------------------
;
;       ReadSectors     Read one or more sectors
;
;       Input   AH      Number of sectors to read
;               AL      Drive number            [0 thru PhysicalDrives-1]
;               BH      Head number             [0 thru MAXHEADS-1]
;               BL      Sector number           [0 thru MAXSECS-1]
;               CX      Cylinder number         [0 thru MAXCYLS-1]
;               DI      Buffer offset address
;               ES      Buffer segment address
;               DS      BIOS Data Segment
;
;       Output  AX-CX   Destroyed
;               DI      One beyond last byte transferred
;            if Carry = ON, error occurred & regs as follows:
;               AH      BIOS Error code (also in StatusByte)
;               DX      Zobex Error and Status registers
;
;--------------------------------------------------------------------------
                assume  DS:BiosData,ES:nothing
ReadSectors     proc    near
                mov     Count,AH
                mov     Drive,AL
                mov     Head,BH
                mov     Sector,BL
                mov     Cyl,CX

        rs0:    call    Read1Sector     ;Read a single sector
                mov     CX,CS           ;Pass retry entry point to exit
                mov     BX,offset ReadSectorsF
                call    TerminateExit   ;Take user I/O exit
                jc      rs9             ;Error occurred
                dec     Count           ;Decrement sector count
                jz      rs9
                call    NextSector      ;Bump address for next time
                jmp     rs0

        rs9     label   near
                mov     StatusByte,AH
                ret
ReadSectors     endp
```

```
;-----------------------------------------------------------------------;
;                                                                       ;
;       WriteSectors    Write one or more sectors                       ;
;                                                                       ;
;       Input   AH      Number of sectors to write                      ;
;               AL      Drive number            [0 thru PhysicalDrives-1]
;               BH      Head number             [0 thru MAXHEADS-1]     ;
;               BL      Sector number           [0 thru MAXSECS-1]      ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]      ;
;               DI      Buffer offset  address                          ;
;               ES      Buffer segment address                          ;
;               DS      BIOS Data Segment                               ;
;                                                                       ;
;       Output  AX-CX   Destroyed                                       ;
;               DI      One beyond last byte transferred                ;
;            if Carry = ON, error occurred & regs as follows:           ;
;               AH      BIOS Error code (also in StatusByte)            ;
;               DX      Zobex Error and Status registers                ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
WriteSectors    proc    near
                mov     Count,AH
                mov     Drive,AL
                mov     Head,BH
                mov     Sector,BL
                mov     Cyl,CX

        ws0:    call    Write1Sector    ;Write a single sector
                mov     CX,CS           ;Pass retry entry point to exit
                mov     BX,offset WriteSectorsF
                call    TerminateExit   ;Take user I/O exit
                jc      ws9             ;Error occurred
                dec     Count           ;Decrement sector count
                jz      ws9
                call    NextSector      ;Bump address for next time
                jmp     ws0

        ws9     label   near
                mov     StatusByte,AH
                ret
WriteSectors    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       VerifySectors   Verify one or more sectors                      ;
;                                                                       ;
;       Input   AH      Number of sectors to verify                     ;
;               AL      Drive number            [0 thru PhysicalDrives-1]
;               BH      Head number             [0 thru MAXHEADS-1]     ;
;               BL      Sector number           [0 thru MAXSECS-1]      ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]      ;
```

```
;               DS      BIOS Data Segment                                       ;
;                                                                               ;
;       Output  AX-CX   Destroyed                                               ;
;            if Carry = ON, error occurred & regs as follows:                   ;
;               AH      BIOS Error code (also in StatusByte)                    ;
;               DX      Zobex Error and Status registers                        ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
VerifySectors   proc    near
                mov     Count,AH
                mov     Drive,AL
                mov     Head,BH
                mov     Sector,BL
                mov     Cyl,CX

        vs0:    call    Verify1Sector   ;Verify a single sector
                mov     CX,CS           ;Pass retry entry point to exit
                mov     BX,offset VerifySectorsF
                call    TerminateExit   ;Take user I/O exit
                jc      vs9             ;Error occurred
                dec     Count           ;Decrement sector count
                jz      vs9
                call    NextSector      ;Bump address for next time
                jmp     vs0

        vs9     label   near
                mov     StatusByte,AH
                ret
VerifySectors   endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       NextSector      Increment the geometric address in                      ;
;                       Drive, Cyl, Head, Sector, and reload argument regs      ;
;                                                                               ;
;       Input   DS      BIOS Data Segment, containing:                          ;
;               Drive   Drive number            [0 thru PhysicalDrives-1]       ;
;               Cyl     Cylinder number         [0 thru MAXCYLS-1]              ;
;               Head    Head number             [0 thru MAXHEADS-1]             ;
;               Sector  Sector number           [0 thru MAXSECS-1]              ;
;                                                                               ;
;       Output  Cyl     Updated                                                 ;
;               Head    Updated                                                 ;
;               Sector  Updated                                                 ;
;               AL      Drive number            [0 thru PhysicalDrives-1]       ;
;               BH      Head number             [0 thru MAXHEADS-1]             ;
;               BL      Sector number           [0 thru MAXSECS-1]              ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]              ;
;               AH      Destroyed                                               ;
```

```
;               Carry   Always set off                                          ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
NextSector      proc    near
        ns1:    mov     AH,Head
                mov     AL,Sector
                inc     AL
                cmp     AL,MAXSECS
                jl      ns2             ;If Sector >= MAXSECS then
                xor     AL,AL           ;   Sector = 0
                inc     AH              ;   Head = Head + 1
                cmp     AH,MAXHEADS
                jl      ns2             ;   if Head >= MAXHEADS then
                xor     AH,AH           ;      Head = 0
                inc     Cyl             ;      Cyl = Cyl + 1

        ns2:    mov     Head,AH         ;Update Head and Sector
                mov     Sector,AL
                mov     BX,AX           ;Now reload appropriate regs
                mov     AL,Drive
                mov     CX,Cyl
                clc
                ret
NextSector      endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       NativeFormat    Format a Track using Supplied Bad Block Array           ;
;                                                                               ;
;       Input   AL      Drive number            [0 thru PhysicalDrives-1]       ;
;               BH      Head number             [0 thru MAXHEADS-1]             ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]              ;
;               DI      offset  address of Bad Block Array                      ;
;               ES      segment address of Bad Block Array                      ;
;                                                                               ;
;       Bad Block Array consists of MAXSECS bytes, where any non-zero           ;
;       byte means that the corresponding sector should have the bad block      ;
;       mark set on.                                                            ;
;                                                                               ;
;       Output  AX-DX   Destroyed                                               ;
;               SI      Destroyed                                               ;
;            if Carry = ON, error occurred & regs as follows:                   ;
;               AH      Zobex Error register                                    ;
;               AL      Zobex Status register                                   ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
InterleaveTable db      0,5,10,15,3,8,13,1,6,11,16,4,9,14,2,7,12
ITLEN           equ     $-InterleaveTable
NativeFormat    proc    near
                mov     BL,MAXGAPSIZE-3 ;Use gapsize-3 in place of sector number
```

```
        call    SetDiskParms    ;Set drive, cyl, head and sector values
        mov     AL,MAXSECS
        mov     DX,ZFC_SECCNTREG
        out     DX,AL           ;Indicate number of sectors/head
        mov     AL,CMD_FORMAT
        mov     DX,ZFC_CMDSTATREG
        out     DX,AL           ;Issue write sector cmd
        call    WaitOnDRQ       ;Wait for DRQ
        jc      nf7             ;Exit if timeout error
        mov     CX,ITLEN
        mov     DX,ZFC_DATAREG

        mov     SI,offset InterleaveTable
        xor     BH,BH

nf2:    mov     BL,CS:[SI]      ;Get sector number from Interleave Table
        mov     AL,ES:[DI+BX]   ;Get corresponding bad block array entry
        cmp     AL,0
        je      nf3
        mov     AL,BADBLOCKFLAG ;Supply Bad Block flag for Zobex ctlr

nf3:    out     DX,AL           ;Bad block flag
        mov     AL,BL
        out     DX,AL           ;Sector number

        inc     SI              ;Next entry from Interleave Table
        loop    nf2             ;Count down 8 bytes at a time

        mov     AL,0
        mov     CX,SECSIZE-(2*ITLEN)

nf4:    out     DX,AL
        loop    nf4

        call    WaitOnNotBusy   ;Wait for command to complete
        jc      nf7             ;Exit if timeout error
        test    AL,STA_ERROR    ;Did an error occur?
        jz      nf8             ;No, everything ok
        mov     AH,AL           ;Keep status in AH for a while
        mov     DX,ZFC_ERRORREG
        in      AL,DX           ;Get error reg into AL
        and     AL,0FFh-ERR_UNUSED      ;Zero unused error bits
        xchg    AL,AH           ;Now: Error in AH, Status in AL

nf6:    stc                     ;Set Carry on
        jmp     short nf9

nf7:    mov     AH,ERR_TIMEOUT  ;Means timeout occurred
        jmp     short nf6

nf8:    xor     AH,AH
        clc                     ;Indicate no error
```

```
        nf9     label   near
                ret
NativeFormat    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       WritelSector    Write a single sector                           ;
;                                                                       ;
;       Input   AL      Drive number            [0 thru PhysicalDrives-1]
;               BH      Head number             [0 thru MAXHEADS-1]     ;
;               BL      Sector number           [0 thru MAXSECS-1]      ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]      ;
;               DI      buffer offset  address                          ;
;               ES      buffer segment address                          ;
;                                                                       ;
;       Output  AX-CX   Destroyed                                       ;
;               DI      One beyond last byte transferred                ;
;            if Carry = ON, error occurred & regs as follows:           ;
;               AH      BIOS Error code (also in StatusByte)            ;
;               DX      Zobex Error and Status registers                ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
WritelSector    proc    near
                call    SetDiskParms    ;Set drive, cyl, head and sector values
                mov     AL,CMD_WRITESEC
                mov     DX,ZFC_CMDSTATREG
                out     DX,AL           ;Issue write sector cmd
                call    WaitOnDRQ       ;Wait for DRQ
                jc      wls9            ;Exit if timeout error
                mov     CX,SECSIZE
                mov     DX,ZFC_DATAREG

                xchg    SI,DI           ;Write data from DS:[SI]

                push    DS              ;Exchange DS and ES
                mov     AX,ES
                pop     ES
                mov     DS,AX

                cld                     ;Set direction flag

        wls2:   lodsb                   ;This loop fits within the 8088's
                out     DX,AL           ; 4 byte instruction cache.
                loop    wls2            ;Transfer all bytes

                push    DS              ;Exchange DS and ES
                mov     AX,ES
                pop     ES
                mov     DS,AX
```

```
                xchg    SI,DI           ;Restore DI register

                call    WaitOnNotBusy   ;Wait for command to complete
                jnc     wls3            ;Timeout error?
                sub     DI,SECSIZE      ;Yes, correct last byte transferred
                stc                     ;Make sure carry still set
                jmp     short wls9      ;Exit

        wls3:   test    AL,STA_ERROR    ;Did an error occur?
                jz      wls4            ;No, everything ok
                sub     DI,SECSIZE      ;Yes, correct last byte transferred
                jmp     IOError         ;Go handle error

        wls4    label   near
                xor     AH,AH
                clc                     ;Indicate no error

        wls9    label   near
                ret
WritelSector    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       ReadlSector     Write a single sector                           ;
;                                                                       ;
;       Input   AL      Drive number            [0 thru PhysicalDrives-1]
;               BH      Head number             [0 thru MAXHEADS-1]     ;
;               BL      Sector number           [0 thru MAXSECS-1]      ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]      ;
;               DI      buffer offset  address                          ;
;               ES      buffer segment address                          ;
;                                                                       ;
;       Output  AX-CX   Destroyed                                       ;
;               DI      One beyond last byte transferred                ;
;            if Carry = ON, error occurred & regs as follows:           ;
;               AH      BIOS Error code (also in StatusByte)            ;
;               DX      Zobex Error and Status registers                ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ReadlSector     proc    near
                call    SetDiskParms    ;Set drive, cyl, head and sector values
                mov     AL,CMD_READSEC
                mov     DX,ZFC_CMDSTATREG
                out     DX,AL           ;Issue read sector cmd
                call    WaitOnNotBusy   ;Wait for read to complete
                jc      rls9            ;Exit if timeout error
                test    AL,STA_ERROR    ;Did any error occur
                jz      rls1
                jmp     IOError         ;Yes, generate error code

        rls1    label   near
```

```
                mov     CX,SECSIZE
                mov     DX,ZFC_DATAREG
                cld                             ;Set direction flag

        rls2:   in      AL,DX                   ;This loop fits within the 8088's
                stosb                           ; 4 byte instruction cache.
                loop    rls2                    ;Transfer all bytes

                xor     AH,AH
                clc                             ;Clear error flag

        rls9    label   near
                ret
Read1Sector     endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       Verify1Sector   Verify a single sector                                ;
;                                                                             ;
;       Input   AL      Drive number            [0 thru PhysicalDrives-1]     ;
;               BH      Head number             [0 thru MAXHEADS-1]           ;
;               BL      Sector number           [0 thru MAXSECS-1]            ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]            ;
;                                                                             ;
;       Output  AX-CX   Destroyed                                             ;
;            if Carry = ON, error occurred & regs as follows:                 ;
;               AH      BIOS Error code (also in StatusByte)                  ;
;               DX      Zobex Error and Status registers                      ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
Verify1Sector   proc    near
                call    SetDiskParms    ;Set drive, cyl, head and sector values
                mov     AL,CMD_READSEC
                mov     DX,ZFC_CMDSTATREG
                out     DX,AL           ;Issue read sector cmd
                call    WaitOnNotBusy   ;Wait for read to complete
                jc      vls9            ;Exit if timeout error
                test    AL,STA_ERROR    ;Did any error occur
                jnz     IOError         ;Yes, generate error code
                xor     AH,AH           ;No, everything ok
                clc
                ret

        vls9    label   near
                ret
Verify1Sector   endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       IOError         Handle I/O errors                                     ;
;                                                                             ;
```

```
;       Input   AL      Status from ZFC_CMDSTATREG                      ;
;                                                                       ;
;       Output  Carry   Always on [error occurred]                      ;
;               AH      BIOS Error code                                 ;
;               AL      Destroyed                                       ;
;               DX      Zobex Error and Status registers                ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
IOError         proc    near
                mov     AH,AL           ;Keep status in AH for a while
                mov     DX,ZFC_ERRORREG
                in      AL,DX           ;Get error reg into AL
                and     AL,0FFh-ERR_UNUSED      ;Zero unused error bits
                push    AX              ;Return this in DX on exit

                test    AH,STA_DRIVEREADY
                jnz     ioe2            ;Check status reg for drive not
                mov     AH,BIOS_TIMEOUT ; ready condition
                jmp     short ioe99

        ioe2:   test    AL,ERR_IDNOTFOUND
                jz      ioea
                mov     AH,BIOS_RECNOTFOUND
                jmp     short ioe99
        ioea    label   near
                test    AL,ERR_CMDABORTED
                jz      ioeb
                mov     AH,BIOS_CTLRFAILED
                jmp     short ioe99
        ioeb    label   near
                test    AL,ERR_TRACKZERO
                jz      ioec
                mov     AH,BIOS_NOADDRMARK+BIOS_RECNOTFOUND
                jmp     short ioe99
        ioec    label   near
                test    AL,ERR_BADBLOCK
                jz      ioed
                mov     AH,BIOS_BADTRACK
                jmp     short ioe99
        ioed    label   near
                test    AL,ERR_DATACRC
                jz      ioee
                mov     AH,BIOS_BADECC
                jmp     short ioe99
        ioee    label   near
                test    AL,ERR_NOADDRMARK
                jz      ioef
                mov     AH,BIOS_NOADDRMARK
                jmp     short ioe99
        ioef    label   near
```

```
        ioe99   label   near
                pop     DX              ;Return Zobex registers also
                xchg    DH,DL           ;DH = Error reg, DL = Status reg
                stc                     ;Indicate error
                ret                     ;Exit
IOError         endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       SpecialRead     Setup to Read n Sector (user handles I/O xfer)  ;
;                                                                       ;
;       Input   AH      Number of sectors to read                       ;
;               AL      Drive number            [0 thru PhysicalDrives-1]
;               BH      Head number             [0 thru MAXHEADS-1]     ;
;               BL      Sector number           [0 thru MAXSECS-1]      ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]      ;
;               DS      BIOS Data Segment                               ;
;                                                                       ;
;       Output  AX-CX - Destroyed                                       ;
;            if Carry = ON, error occurred & regs as follows:           ;
;               AH      BIOS Error code (also in StatusByte)            ;
;               DX      Zobex Error and Status registers                ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
srret           dw      sr2             ;Return point after error handling
SpecialRead     proc    near
                mov     Count,AH
                mov     Drive,AL
                mov     Head,BH
                mov     Sector,BL
                mov     Cyl,CX

        sr0:    call    SetDiskParms    ;Set drive, cyl, head and sector values
                mov     AL,CMD_READSEC
                mov     DX,ZFC_CMDSTATREG
                out     DX,AL           ;Issue read sector cmd
                call    WaitOnNotBusy   ;Wait for read to complete
                jc      sr2             ;Take exit if timeout error
                test    AL,STA_ERROR    ;Did any error occur
                jz      sr1
                push    srret
                jmp     IOError         ;Yes, generate error code

        sr1:    mov     CX,SECSIZE
                mov     DX,ZFC_DATAREG
                xor     AH,AH
                clc                     ;Clear error flag

        sr2:    mov     CX,CS           ;Pass retry entry point to exit
                mov     BX,offset SpecialReadF
                call    TerminateExit   ;Take user I/O exit
```

```
                jc      sr9             ;Error occurred
                dec     Count           ;Decrement sector count
                jz      sr9
                call    NextSector      ;Bump address for next time
                jmp     sr0

        sr9     label   near
                mov     StatusByte,AH
                ret
SpecialRead     endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       SelectDrive     Select drive [SDH will have drive value]        ;
;                                                                       ;
;       Input   none                                                    ;
;                                                                       ;
;       Output  none                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
SelectDrive     proc    near
                push    AX
                push    DX
                mov     AL,EXT_SELENABLE ;Enables drive select lines in SDH reg
                mov     DX,ZFC_EXTREG
                out     DX,AL
                pop     DX
                pop     AX
                ret
SelectDrive     endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       DeselectDrive    Deselect drive                                 ;
;                                                                       ;
;       Input   none                                                    ;
;                                                                       ;
;       Output  none                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
DeselectDrive   proc    near
                push    AX
                push    DX
                mov     AL,EXT_SELDISABLE       ;Disable drive select lines
                mov     DX,ZFC_EXTREG
                out     DX,AL
                pop     DX
                pop     AX
                ret
DeselectDrive   endp
```

```
;--------------------------------------------------------------------------;
;                                                                          ;
;       WaitOnXXXX      Routines to wait for various states to occur        ;
;                                                                          ;
;       Input   None                                                       ;
;                                                                          ;
;       Output  Carry   ON=error OFF=no error                              ;
;               AH      Error code if carry on [only timeout detected here];
;               AL      Status from ZFC_CMDSTATREG                         ;
;               BX-DX   Destroyed                                          ;
;                                                                          ;
;--------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
WaitOnSeek      proc    near
                call    WaitOnNotBusy   ;First wait for ZFC not busy
                mov     AH,STA_SEEKDONE ;Now wait for seek to complete
                jmp     GeneralWait
WaitOnSeek      endp

WaitOnDRQ       proc    near
                mov     AH,STA_DATAREQ  ;Wait for DRQ from ZFC
                jmp     short GeneralWait
WaitOnDRQ       endp

WaitOnNotBusy   proc    near
                mov     AH,STA_BUSY     ;Wait for ZFC to complete a command.
WaitOnNotBusy   endp

GeneralWait     proc    near
                mov     BX,10           ;Load timeout counter
                sub     CX,CX
                mov     DX,ZFC_CMDSTATREG

        gw2:    in      AL,DX
                xor     AL,STA_BUSY     ;Convert busy bit to done bit
                test    AL,AH           ;[Sets carry off]
                jnz     gw4             ;Exit if non-zero
                loop    gw2
                dec     BX
                jnz     gw2
                mov     AH,BIOS_TIMEOUT ;Timeout
                mov     DH,AL
                xor     DL,DL
                stc                     ;Indicate error

        gw4:    ret
GeneralWait     endp

;--------------------------------------------------------------------------;
;                                                                          ;
;       SetDiskParms    Set Disk I/O Parameters                            ;
```

```
;                                                                              ;
;       Input   AL      Drive number            [0 thru PhysicalDrives-1]      ;
;               BH      Head number             [0 thru MAXHEADS-1]            ;
;               BL      Sector number           [0 thru MAXSECS-1]             ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]             ;
;                                                                              ;
;       Output  None                                                           ;
;               AX-DX   Destroyed                                              ;
;                                                                              ;
;------------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
SetDiskParms    proc    near
                push    AX              ;Save original drive value

;       Set External Cmd/Status register
;
                mov     AL,EXT_SELENABLE;Set drive select enable
                cmp     BH,8
                jl      sdp1            ;If 7 or less no high order bit
                or      AL,EXT_HIGHHEAD ;Select heads 8-14
        sdp1    label   near

                mov     DX,ZFC_EXTREG   ;External cmd/status register
                out     DX,AL

;       Set SDH register
;
                pop     AX              ;Restore original AX
                xor     AH,AH           ;Supply high order zeros to drive number
                and     AL,03h          ;Only values 0-3 are acceptable
                shl     AL,1            ;Shift drive number into position
                shl     AL,1            ; for SDH reg
                shl     AL,1
                or      AL,SDH_SECSIZE+SDH_ECCENABLE
                and     BH,7            ;Only use 3 low order head bits
                or      AL,BH           ;Set low order head bits
                mov     DX,ZFC_SDHREG   ;Select SDH register
                out     DX,AL           ;Indicate sector size, drive, head, ECC

;       Set cylinder number
;
                mov     DX,ZFC_HIGHCYLREG
                mov     AL,CH
                out     DX,AL           ;Output MSB of cylinder number
                mov     DX,ZFC_LOWCYLREG
                mov     AL,CL
                out     DX,AL           ;Output LSB of cylinder number

;       Set sector number
;
                mov     DX,ZFC_SECTORREG
                mov     AL,BL           ;Get sector number
```

```
                out     DX,AL
                ret
SetDiskParms    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       Entry Points for Far Alternate Sector Retry Handlers            ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ReadSectorsF    proc    far
                call    ReadSectors
                ret
ReadSectorsF    endp

WriteSectorsF   proc    far
                call    WriteSectors
                ret
WriteSectorsF   endp

VerifySectorsF  proc    far
                call    VerifySectors
                ret
VerifySectorsF  endp

SpecialReadF    proc    far
                call    SpecialRead
                ret
SpecialReadF    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       TerminateExit   User I/O Termination Exit                       ;
;                                                                       ;
;       Input   Carry   ON = Error occurred                             ;
;                       OFF = No errors detected                        ;
;               BX      Offset  address of retry routine                ;
;               CX      Segment address of retry routine                ;
;               DS      BIOS Data Segment                               ;
;            if Carry = ON then:                                        ;
;               AH      BIOS error code                                 ;
;               DX      Zobex error and status registers                ;
;                                                                       ;
;       Output  Carry   ON = Terminate I/O with error indication        ;
;                       OFF = Continue with next sector of I/O operation;
;               AH      Must be preserved                               ;
;               DX      Must be preserved                               ;
;               DI      Must be preserved                               ;
;               DS      Must be preserved                               ;
;               ES      Must be preserved                               ;
;                                                                       ;
;-----------------------------------------------------------------------;
```

```
;               assume  DS:BiosData,ES:nothing
;TerminateExit  proc    far
;
;       This routine may be used to:
;
;               1. Recover using alternate sector retry
;               2. Terminate multisector operations prematurely
;               3. Track progress of I/O operations
;               4. Transfer data from the controller (for SpecialRead only)
;
;       Termination exits must be installed by placing a far entry point
;       in the field TerminateExit in the BIOS Data Segment.  This field
;       is assumed to be initialized with a pointer to a far return.
;
;TerminateExit  endp

EndOfMAXTORBIOS equ     $

Cseg            ends
                end
```

Formatted by SCRIPT from file 'MAXBOOT PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
                page    56,132
                title   'Power On Boot Routine for MAXTOR BIOS'

;-------------------------------------------------------------------------;
;                                                                         ;
;       Power On Boot Routine for MAXTOR 1140 Hard Disk BIOS Package      ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1         ;
;       Version:        1.9 05/10/84                                      ;
;       DOS Required:   None                                              ;
;       Classification: IBM Internal Use Only                             ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------;

                include maxdata.dcl     ;MAXTOR BIOS error equates

BootROMData     segment at 40h          ;RAM needed for MAXTOR BOOT
                org     75h
HardDiskCount   db      ?               ;IBM HD BIOS places drive count here
                org     0B8h
PartOrigin      dw      ?               ;DOS Boot Partition origin cylinder
PartCylCount    dw      ?               ;DOS Boot Partition size in cyls
                org     7800h
IOBuffer        dw      512 dup(?)      ;Temporary buffer used during boot
BootROMData     ends

LowMemory       segment at 0            ;Segment 0 map
                org     4Ch
DiskRequestIV   dd      ?               ;Int 13h Disk(ette) I/O Request
                org     64h
BootStrapIV     dd      ?               ;Int 19h BootStrap
                org     78h
DisketteParms   dd      ?               ;Pointer to diskette parameters
                org     104h
FixedDiskParms  dd      ?               ;Pointer to fixed disk parameters
                org     7C00h
BootLocation    label   far             ;Bootstrap code loaded here
LowMemory       ends

Cseg            segment public 'code'
                assume  CS:Cseg,SS:nothing

IBMCopyright    segment byte common 'IBMCopyright'
                db      ' (C) Copyright IBM Corporation, 1984 '
                db      'Written by L K Raper, 05/10/84 '
IBMCopyright    ends
```

```
                extrn   DetPhysDrives:near

                extrn   ReadSectors:near
                extrn   RequestRouter:near
                extrn   ResetDisk:near

;       Generic BIOS Header for Adapter ROMs
;
                db      055h,0AAh       ;BIOS ROM extension signature
                db      4               ;Maximum number of 512 sections
                jmp     short InitMAX


;       Local constants
;
DDPARTITION     equ     8000h           ;DOS Partition

;       Computed constants
;
ChecksumAdjust  db      0               ;Adjust for proper checksum before
                                        ; burning into ROM
BiosDataSeg     dw      seg BiosData    ;Segment address of BIOS data
PartHeader      db      'Partition Record'      ;Header in Partition Record

;-----------------------------------------------------------------------------;
;                                                                             ;
;       Initialize MAXTOR BIOS Support During Boot                            ;
;                                                                             ;
;       Input   none                                                          ;
;                                                                             ;
;       Output  None                                                          ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:BiosData,ES:LowMemory
InitMAX         proc    far
                cli
                xor     DI,DI
                mov     ES,DI
                mov     DS,BiosDataSeg

                mov     AX,word ptr DiskRequestIV
                mov     OrigDiskReqOff,AX
                mov     AX,word ptr DiskRequestIV+2
                mov     OrigDiskReqSeg,AX
                mov     word ptr DiskRequestIV,offset RequestRouter
                mov     word ptr DiskRequestIV+2,CS
                mov     word ptr BootStrapIV,offset BootStrap
                mov     word ptr BootStrapIV+2,CS

                mov     word ptr TerminateExit,offset NullDDExit
                mov     word ptr TerminateExit+2,CS
                mov     word ptr Partitioner,offset BootPartitioner
```

```
                mov     word ptr Partitioner+2,CS
                mov     MaxBase,MAXBOOT         ;Indicate boot from MAXTOR
                mov     DS:HardDiskCount,1      ;DOS requires that this be set
                sti

                call    ResetDisk               ;Reset ZOBEX disk controller
                call    DetPhysDrives           ;Find out how many drives
                mov     PhysicalDrives,AL

                call    InitBootPart            ;Initialize Boot Partition info
                ret
InitMAX         endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       NullDDExit      Null Device Driver Multi-block Exit                   ;
;                                                                             ;
;       Input   none                                                          ;
;                                                                             ;
;       Output  none                                                          ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
NullDDExit      proc    far
                ret
NullDDexit      endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       BootPartitioner Adjust Drive and Cylinder Values for Boot Partition   ;
;       Note: Devices A0 and A1 are considered unpartitioned MAXTORs.         ;
;                                                                             ;
;       Input   AL      Logical drive           [Only 0 or unpartitioned]     ;
;               CX      Cylinder logical drive                                ;
;                                                                             ;
;       Output  AL      Physical drive address  [Always 0]                    ;
;               CX      Absolute cylinder       [0 thru partition limit]      ;
;               DX      Number of cylinders in partition                      ;
;               Carry   ON=range error, OFF=everything OK                     ;
;               AH      Error code if carry is ON, else undisturbed           ;
;                                                                             ;
;-----------------------------------------------------------------------------;

                assume  DS:BiosData,ES:nothing
BootPartitioner proc    near
                push    BX              ;Save working regs
                push    DS
                mov     DS,BiosDataSeg

                add     AL,MaxBase
                cmp     AL,0A0h         ;Absolute drive with no partitioning?
                jb      bp0
```

```
                sub     AL,0A0h
                mov     DX,MAXCYLS
                jmp     short bp3

        bp0:    sub     AL,MaxBase
                cmp     AL,LogicalDisks ;Does partition exist?
                jae     bp1             ;No, error
                mov     BL,AL
                xor     BH,BH
                shl     BL,1
                shl     BL,1

                assume  DS:BootROMData
                mov     DX,PartCylCount[BX]
                cmp     CX,DX           ;Is request within range of partition?
                jae     bpe             ;No, error
                add     CX,PartOrigin[BX]       ;Yes, compute absolute cyl
                jmp     short bp3

        bp1:    mov     AH,BIOS_TIMEOUT ;Drive does not exist
                stc
                jmp     short bp9

        bp3:    xor     AL,AL           ;Boot partitions only on drive 0
                clc                     ;Clear carry flag

        bp9:    pop     DS              ;Restore working regs
                pop     BX

        farret  proc    far
                ret     2               ;Return to caller
        farret  endp

        bpe:    stc                     ;Indicate error resolving partition
                mov     AH,BIOS_RECNOTFOUND
                jmp     bp9
BootPartitioner endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       InitBootPart    Initialize Boot Partition Info                        ;
;                                                                             ;
;       Input   none                                                          ;
;                                                                             ;
;       Output  none    [PartOrigin and PartCylCount set]                     ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:BootROMData,ES:nothing
InitBootPart    proc    near
                mov     DS,BiosDataSeg
                mov     ES,BiosDataSeg
                mov     PartOrigin,0
```

```
                mov     PartCylCount,0
                mov     PartOrigin+4,0
                mov     PartCylCount+4,0
                mov     DS:LogicalDisks,0

                mov     AX,0100h        ;Read 1 sector Partition Record, drive 0
                mov     BH,PARTRECHD
                mov     BL,PARTRECSEC
                mov     CX,PARTRECCYL
                mov     DI,offset IOBuffer
                call    ReadSectors
                mov     AL,0            ;Initialize partition count
                jc      ibp9

                mov     SI,offset IOBuffer
                mov     CX,16
                push    CS
                pop     ES
                mov     DI,offset PartHeader
                cld
                repe    cmpsb
                jne     ibp9

                mov     CX,8            ;At most 8 entries in partition table
                xor     DI,DI
                mov     DL,2            ;Boot supports at most 2 partitions

        ibp2:   test    word ptr DS:[SI],DDPARTITION
                jz      ibp3
                mov     BX,word ptr DS:[SI+2]
                cmp     BX,0            ;Does partition really exist?
                je      ibp3            ;No, skip over it
                mov     PartCylCount[DI],BX
                mov     BX,word ptr DS:[SI]
                and     BX,0FFFFh-DDPARTITION
                mov     PartOrigin[DI],BX
                inc     DS:LogicalDisks
                add     DI,4
                dec     DL
                jz      ibp4

        ibp3:   add     SI,4
                loop    ibp2
        ibp4    label   near

        ibp9:   call    DetPhysDrives   ;Seek back to Cyl 0
                ret                     ;Return to caller
InitBootPart    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       BootStrap       Boot from diskette or MAXTOR                    ;
```

```
;
;       Input   none                                                    ;
;                                                                       ;
;       Output  none                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
DisketteTable   db      0CFh,2,25h,2,8,2Ah,0FFh,50h,0F6h,19h,4
MaxtorTable     db      83h,0,15,0,0,0,0,0,0,0,0B4h,28h,0,0,0,0
                assume  DS:LowMemory,ES:nothing
BootStrap       proc    near
                cli
                xor     AX,AX
                mov     DS,AX
                mov     SS,AX           ;Place stack where it will not
                mov     SP,offset BootLocation  ; interfere with boot code

                mov     word ptr FixedDiskParms,offset MaxtorTable
                mov     word ptr FixedDiskParms+2,CS
                mov     word ptr DisketteParms,offset DisketteTable
                mov     word ptr DisketteParms+2,CS
                sti

;       First attempt to boot from diskette
;
                mov     CX,3            ;Set retry count

        bs1:    push    CX              ;Save retry count
                sub     DX,DX           ;Drive zero
                sub     AX,AX           ;Reset the diskette
                int     13h
                jc      bs2             ;If error, try again
                mov     AX,0201h        ;Read in a single sector

                sub     DX,DX
                mov     ES,DX           ;Establish buffer location
                mov     BX,offset BootLocation

                mov     CX,1            ;Sector 1, track 0
                int     13h

        bs2:    pop     CX              ;Recover retry count
                jnc     bs4             ;Continue if successful read
                cmp     AH,80h          ;If time out, no retry
                jz      bs5             ;Try fixed disk
                loop    bs1             ;Do it for retry times
                jmp     bs5             ;Unable to boot the diskette

        bs4:    jmp     BootLocation    ;Boot was successful

;       Next attempt bootstrap from MAXTOR
;
        bs5:    sub     AX,AX           ;Reset diskette
```

```
                sub     DX,DX
                int     13h
                mov     CX,3            ;Set retry count

        bs6:    push    CX              ;Save retry count
                mov     DX,0080h        ;Fixed disk zero
                sub     AX,AX           ;Reset the MAXTOR
                int     13h
                jc      bs7             ;If error, try again
                mov     AX,0201h        ;Read in a single sector
                sub     BX,BX
                mov     ES,BX
                mov     BX,offset BootLocation  ;To the boot location
                mov     DX,80h          ;Drive number
                mov     CX,1            ;Sector 1, track 0
                int     13h

        bs7:    pop     CX              ;Recover retry count
                jc      bs8
                mov     AX,word ptr BootLocation+510
                cmp     AX,0AA55h       ;Test for generic boot block
                jz      bs4

        bs8:    loop    bs6             ;Do it for retry times

;       Unable to boot from the diskette or MAXTOR
;
                int     18h             ;Start resident BASIC
BootStrap       endp

Cseg            ends
                end
```

Formatted by SCRIPT from file 'MAXDD PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
                page    56,132
                title   'PC DOS2 Device Driver for MAXTOR 1140 Hard Disk'

;-----------------------------------------------------------------------;
;                                                                       ;
;       DOS2 Device Driver for the MAXTOR 1140 Hard Disk                ;
;       Program Property of IBM                                         ;
;                                                                       ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1       ;
;       Version:        2.E 06/20/84                                    ;
;       DOS Required:   2.0 or later                                    ;
;       Classification: IBM Internal Use Only                           ;
;                                                                       ;
;       (C) Copyright IBM Corporation, 1984                             ;
;       Developed for the Information Technology Center at              ;
;       Carnegie-Mellon University                                      ;
;                                                                       ;
;-----------------------------------------------------------------------;

                include maxdata.dcl     ;MAXTOR BIOS error equates
                include maxdos2.dcl     ;PC DOS Device Driver constants


Cseg            segment public 'code'
                assume  CS:Cseg,SS:nothing

IBMCopyright    segment byte common 'IBMCopyright'
                db      ' (C) Copyright IBM Corporation, 1984 '
IBMCopyright    ends

                extrn   CvtToGeometric:near
                extrn   DetPhysDrives:near
                extrn   PrintHexByte:near
                extrn   PrintHexWord:near
                extrn   PrintCRLF:near
                extrn   PrintString:near

                extrn   ObtainAST:near
                extrn   RecoveryViaAST:near

                extrn   DeselectDrive:near
                extrn   EndOfMAXTORBIOS:byte
                extrn   ReadSectors:near
                extrn   RequestRouter:near
                extrn   ResetDisk:near
                extrn   SelectDrive:near
                extrn   VerifySectors:near
                extrn   WriteSectors:near

;-----------------------------------------------------------------------;
;                                                                       ;
;       Device header table                                             ;
;                                                                       ;
```

```
;       Must be located at beginning of module                          ;
;                                                                       ;
;-----------------------------------------------------------------------;

DeviceHeader    dd      -1              ;Pointer to next device
                dw      4000h           ;Block device, IOCTL, IBM-format
                dw      StrategyRoutine ;Offset to strategy routine
                dw      ServiceRoutine  ;Offset to service routine
                db      8 dup(0)        ;Name [for character devices]

FunctionTable   dw      Initialize      ;Initialization
                dw      MediaCheck      ;Media check    [block only]
                dw      BuildBPB        ;Build BPB      [block only]
                dw      DeviceInfo      ;IOCTL input
                dw      ReadBlocks      ;Input (read)
                dw      ReqError        ;Non-dest. input no wait [char only]
                dw      ReqError        ;Input status   [char only]
                dw      ReqError        ;Input flush    [char only]
                dw      WriteBlocks     ;Output (write)
                dw      WriteVerBlocks  ;Output (write) with verify
                dw      ReqError        ;Output status  [char only]
                dw      ReqError        ;Output flush   [char only]
                dw      ToggleAST       ;IOCTL output

;       Local constants
;
CR              equ     13              ;Carriage return
LF              equ     10              ;Line feed
DOSIND          equ     1               ;Indicates DOS partition to FDISK
FDSIGORG        equ     1FEh            ;Offset of FDISK signature field
FDSIG           equ     0AA55h          ;Actual value of FDISK signature
FDISKTABORG     equ     1BEh            ;Origin of FDISK part table
READCOMMAND     equ     4               ;Operation code for DD ReadBlocks
MEDIADESC       equ     0F8h            ;DOS Media Descriptor for Fixed Disks
DDPARTITION     equ     8000h           ;Partition available for Dos Driver
BiosDataSeg     dw      seg BiosData    ;Segment address of BIOS data
BlocksPerCyl    dw      MAXSECS*MAXHEADS;Number of blocks per cylinder
PartHeader      db      'Partition Record'      ;Header in Partition Record

;-----------------------------------------------------------------------;
;                                                                       ;
;       Private static variables                                        ;
;                                                                       ;
;-----------------------------------------------------------------------;

DebugFlag       db      0               ;Non-zero means display debugging msgs
RHoff           dw      ?               ;Offset address of DOS Request Header
RHseg           dw      ?               ;Segment address of DOS Request Header
spoff           dw      ?               ;Offset  address of caller's stack
spseg           dw      ?               ;Segment address of caller's stack
mcunit          db      ?               ;Unit for last media check
dddrive         db      ?               ;Current physical drive
```

```
ddhead          db      ?               ;Current head number
ddsector        db      ?               ;Current sector number
ddcyl           dw      ?               ;Current cylinder number
blockcount      dw      ?               ;Total blocks transferred
startblock      dw      ?               ;Starting block number from req hdr
EndOfDosDriver  dw      EndOfMAXTORBIOS ;High water mark for DOS Driver
ZStatus         dw      ?               ;Status from Controller
IgnoreError     db      0               ;Non-zero means disregard error status
SuppressAST     db      0               ;Non-zero means AST disabled
UnitCount       db      ?               ;Number of units supported
UnitOffset      db      0               ;1st MAXTOR partition supported by
                                        ; this driver (skips a booted partition)
Drv1PartLimit   db      0               ;Last Partition on Drive 1 [1-N]
Drv2PartLimit   db      0               ;Last Partition on Drive 2 [N+1 - ...]
DosPartTable    dw      32 dup(0)       ;Describes partitions 0 thru 15
DosFPTable      dw      32 dup(0)       ;Has FDISK boundaries for DOS access
DosCylOrgAndLen dw      32 dup(0)       ; [Same, but by cyls rather than blocks]
                dw      256 dup(0)      ;Local stack area
LocalStack      equ     $

;       Bios parameter block
;
BPBLEN          equ     13
BPBSecsPerAlloc equ     2               ;Offset to sectors per allocation unit
BPBDiskSizeOff  equ     8               ;Offset to logical disk size in BPB
BPBSecsPerFAT   equ     11              ;Offset to sectors per FAT in BPB
BPB             label   byte
                rept    16
                dw      512             ;Sector size
                db      8               ;Sectors/allocation unit (computed)
                dw      1               ;Reserve 1 boot sector
                db      2               ;Maintain 2 FATs
                dw      512             ;Number of directory entries
                dw      0               ;Logical Disk size (computed)
                db      MEDIADESC       ;Media descriptor
                dw      12              ;Number of sectors per FAT (computed)
                endm

BPBptrarray     label   word
                dw      BPB+(BPBLEN*00) ;Bios parm block ptr array
                dw      BPB+(BPBLEN*01)
                dw      BPB+(BPBLEN*02)
                dw      BPB+(BPBLEN*03)
                dw      BPB+(BPBLEN*04)
                dw      BPB+(BPBLEN*05)
                dw      BPB+(BPBLEN*06)
                dw      BPB+(BPBLEN*07)
                dw      BPB+(BPBLEN*08)
                dw      BPB+(BPBLEN*09)
                dw      BPB+(BPBLEN*10)
                dw      BPB+(BPBLEN*11)
                dw      BPB+(BPBLEN*12)
```

```
                dw      BPB+(BPBLEN*13)
                dw      BPB+(BPBLEN*14)
                dw      BPB+(BPBLEN*15)

;-----------------------------------------------------------------------;
;                                                                       ;
;       StrategyRoutine Captures Request Information from DOS           ;
;                                                                       ;
;       Input   ES:[BX] Request Header                                  ;
;       Output  None                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
StrategyRoutine proc    far
                mov     RHoff,BX        ;Capture address of request header
                mov     RHseg,ES
                ret
StrategyRoutine endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       ServiceRoutine  Perform Requested Operation                     ;
;                                                                       ;
;       Input   RHseg   Segment address of request header               ;
;               RHoff   Offset  address of request header               ;
;                                                                       ;
;       Output  None    [Request Header has status]                     ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ServiceRoutine  proc    far
                cli
                mov     spseg,SS                ;Save SS, SP
                mov     spoff,SP
                mov     SP,CS                   ;Set up local stack
                mov     SS,SP
                mov     SP,offset LocalStack
                sti
                push    ES                      ;Save registers
                push    DS
                push    SI
                push    DI
                push    DX
                push    CX
                push    BX
                push    AX
                mov     DS,RHseg                ;Setup DS:[SI] to point to
                mov     SI,RHoff                ; DOS Request Header
                call    PrintReqHdr

                mov     RH_status,0             ;Reset Status prior to service
                mov     ddcyl,0                 ;Reset working variables
```

```
                mov     ddhead,0
                mov     ddsector,0
                mov     startblock,0
                mov     blockcount,0            ;No blocks transferred yet

                call    SelectDrive             ;Enable disk select
                mov     AL,RH_command           ;Get command code
                xor     AH,AH                   ;AX = function code
                add     AX,AX                   ;Form index into word table
                mov     DI,AX
                call    FunctionTable[DI]       ;Call processing routine
                call    DeselectDrive

                or      RH_status,RH_DONE       ;Indicate request finished

                test    RH_status,RH_IOERROR
                jz      srd1
                call    PrintError

        srd1:   pop     AX                      ;Restore registers
                pop     BX
                pop     CX
                pop     DX
                pop     DI
                pop     SI
                pop     DS
                pop     ES
                cli
                mov     SS,spseg
                mov     SP,spoff
                sti
                ret
ServiceRoutine  endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       Initialize      Initialize device and install driver                  ;
;                                                                             ;
;       Input   DS:[SI] Request Header                                        ;
;                                                                             ;
;       Output  None    [All registers destroyed except DS and SI]            ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ExecutionFlag   db      0                       ;Modified in executing copy only
Initialize      proc    near
                call    PrintString
                db      'MAXTOR 1140 Device Driver '
                db      '- Version 2.E 06/20/84',CR,LF
                db      '(C) Copyright IBM Corporation, 1984 '
                db      CR,LF,0
```

```
        push    DS                      ;Save RH addressability
        push    SI
        mov     ExecutionFlag,0FFh      ;This makes us different from
                                        ; code still in DOS buffer area
        push    CS
        pop     ES
        mov     AX,0400h
        cld

i1:     inc     AX                      ;Locate 1st copy of us
        mov     DS,AX
        mov     DI,offset ExecutionFlag
        mov     SI,DI
        mov     CX,100                  ;Check 100 words to be sure
        repe    cmpsw
        jne     i1
        mov     CX,DS                   ;CX now has first copy address
        mov     BX,CS

        xor     SI,SI                   ;Point to low memory
        mov     DS,SI
        mov     ES,BiosDataSeg
        assume  DS:nothing,ES:BiosData

        cli
        cmp     CX,BX                   ;Is current instance the first?
        jne     i2                      ;No, don't recapture IV
        cmp     MaxBase,MAXBOOT         ;Did we boot from MAXTOR?
        jne     i1a                     ;No, skip following portion
        mov     AL,LogicalDisks         ;Yes, remember for later
        mov     UnitOffset,AL           ; and skip IBM hard disk section
        jmp     short i1b
i1a     label   near

        mov     MaxBase,MAXBOOT+2       ;Allow for IBM hard disks
        mov     AX,word ptr DS:[SI+4Ch]
        mov     OrigDiskReqOff,AX
        mov     AX,word ptr DS:[SI+4Eh]
        mov     OrigDiskReqSeg,AX
        mov     AX,offset RequestRouter
        mov     DS:[SI+4Ch],AX
        mov     DS:[SI+4Eh],CS
i1b     label   near

        mov     AX,offset DivideByZero  ;SPY utility program seems to
        mov     DS:[SI],AX              ; need this
        mov     DS:[SI+2],CS            ;

i2:     mov     word ptr TerminateExit,offset DevDriverExit
        mov     word ptr TerminateExit+2,CS
        mov     word ptr Partitioner,offset DosPartitioner
        mov     word ptr Partitioner+2,CS
```

```
                sti

                mov     DS,BiosDataSeg
                assume  DS:BiosData,ES:nothing

                call    ResetDisk               ;Reset ZOBEX disk controller
                call    DetPhysDrives           ;Find out how many drives
                mov     PhysicalDrives,AL

                push    CS
                pop     ES
                mov     DI,offset EndOfMAXTORBIOS+1
                mov     DX,offset DosPartTable
                mov     LogicalDisks,0
                cmp     PhysicalDrives,1
                jb      i4
                mov     AL,0
                call    ObtainAST               ;Get Alternate Sector Table
                add     DI,ASTSECCNT*SECSIZE
                mov     EndOfDosDriver,DI
                call    InitDosPart             ;Initialize DOS Partition Table
                mov     Drv1PartLimit,AL
                mov     LogicalDisks,AL

                cmp     PhysicalDrives,2
                jb      i4a
                mov     AL,1                    ;If 2 drives. do another one
                call    ObtainAST               ;Get Alternate Sector Table
                add     DI,ASTSECCNT*SECSIZE
                mov     EndOfDosDriver,DI
                call    InitDosPart             ;Get more partition info
                add     AL,Drv1PartLimit
                mov     Drv2PartLimit,AL
                mov     LogicalDisks,AL
                jmp     short i4a

i4:             call    PrintString
                db      'No MAXTOR devices attached'
                db      CR,LF,0

i4a:            mov     CL,LogicalDisks
                mov     UnitCount,CL

                xor     CH,CH
                jcxz    i4b
                call    ReadFDISKParts          ;Determine FDISK partitions
                jmp     short i4c

i4b:            jmp     i6

i4c:            mov     BX,offset BPBptrarray
                xor     DI,DI
```

```
        mov     AL,UnitOffset           ;Subtract 1 if booted from
        sub     UnitCount,AL            ; MAXTOR
        cmp     AL,0                    ;Boot from MAXTOR?
        je      i5                      ;No, continue
        xor     AH,AH                   ;Yes, bypass DOS partitions
        shl     AX,1                    ; handled by Boot
        shl     AX,1
        add     DI,AX

i5:     mov     SI,CS:[BX]              ;Fill in BPBs appropriately
        mov     AX,BlocksPerCyl
        mul     CS:DosCylOrgAndLen[DI+2]
        mov     word ptr CS:[SI+BPBDiskSizeOff],AX

        push    CX
        xor     CL,CL
        mov     DX,1                    ;Determine number of sectors
        cmp     AX,200h                 ; per allocation unit just
        jb      i5a                     ; as DOS does. Basically this
        inc     CL                      ; is as follows:
        mov     DL,2                    ;
        cmp     AX,800h                 ; Block count   SecsPerAlloc
        jb      i5a                     ;   40h-01FFh        1
        inc     CL                      ;  200h-07FFh        2
        mov     DL,4                    ;  800h-1FFFh        4
        cmp     AX,2000h                ; 2000h-7FA7h        8
        jb      i5a                     ; 7FA8h or more     16
        inc     CL
        mov     DL,8
        cmp     AX,7FA8h
        jb      i5a
        and     AL,0F0h
        inc     CL
        mov     DL,16

i5a:    mov     byte ptr CS:[SI+BPBSecsPerAlloc],DL
        dec     DX                      ;Determine sectors per FAT
        add     DX,AX                   ;This is a mysterious
        shr     DX,CL                   ; algorithm, but it matches
        inc     DX                      ; DOS!
        and     DL,0FEh
        mov     AX,DX
        shr     DX,1
        add     DX,AX
        add     DX,1FFh
        shr     DH,1
        mov     DL,DH
        xor     DH,DH
        mov     word ptr CS:[SI+BPBSecsPerFAT],DX
        pop     CX

        add     BX,2
```

```
                add     DI,4
                loop    i5

                assume  DS:nothing,ES:nothing
        i6:     pop     SI                      ;Restore RH addressability
                pop     DS

                mov     AL,UnitCount            ;Tell DOS how many disks
                cmp     AL,0                    ;[But never tell DOS '0'
                jne     i7                      ; because it causes problems]
                inc     AL

        i7:     mov     RH_unitcount,AL
                mov     RH_bpbarrayoff,offset BPBpcrarray
                mov     RH_bpbarrayseg,CS
                push    CS:EndOfDosDriver
                pop     RH_endoffptr
                mov     RH_endsegptr,CS
                ret

DivideByZero:   iret                            ;Ignore divide by zero for SPY

Initialize      endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       InitDosPart     Initialize Dos Partition Table                        ;
;                                                                             ;
;       Input   AL      Physical drive                                        ;
;               DX      Offset from CS for Table                              ;
;               ES:[DI] Buffer area for 1 Sector                              ;
;                                                                             ;
;       Output  DX      Points to next available table slot                   ;
;               AH      Destroyed                                             ;
;               AL      Number of partitions found                            ;
;                                                                             ;
;-----------------------------------------------------------------------------;

;       Local storage for InitDosPart
;
IDPLOCALLEN     equ     6
idptaboff       equ     word ptr [BP-6]
idpbufoff       equ     word ptr [BP-4]
idpbufseg       equ     word ptr [BP-2]

                assume  DS:nothing,ES:nothing
InitDosPart     proc    near
                push    BP
                mov     BP,SP           ;Establish frame pointer
                sub     SP,IDPLOCALLEN  ;Reserve storage for local variables

                push    BX
```

```
        push    CX
        push    SI
        push    DI
        push    DS
        push    ES
        mov     idpbufoff,DI    ;Save parameters
        mov     idpbufseg,ES
        mov     idptaboff,DX

        mov     AH,1            ;Read 1 sector Partition Record
        mov     BH,PARTRECHD    ; [drive is already in AL]
        mov     BL,PARTRECSEC
        mov     CX,PARTRECCYL
        mov     DS,BiosDataSeg
        call    ReadSectors
        mov     AL,0            ;Initialize partition count
        jc      idp9

        mov     DS,idpbufseg
        mov     SI,idpbufoff
        mov     CX,16
        push    CS
        pop     ES
        mov     DI,offset PartHeader
        cld
        repe    cmpsb
        jne     idp9

        mov     DI,idptaboff
        mov     CX,8            ;At most 8 entries in partition table

idp2:   test    word ptr DS:[SI],DDPARTITION
        jz      idp3
        mov     BX,word ptr DS:[SI]
        and     BX,0FFFFh-DDPARTITION
        mov     CS:[DI],BX
        mov     BX,word ptr DS:[SI+2]
        mov     CS:[DI+2],BX
        inc     AL
        add     DI,4

idp3:   add     SI,4
        loop    idp2
        mov     idptaboff,DI

idp9:   push    AX
        call    DetPhysDrives   ;Seek back to Cyl 0
        pop     AX
        mov     DX,idptaboff    ;Restore regs
        pop     ES
        pop     DS
        pop     DI
```

```
                pop     SI
                pop     CX
                pop     BX

                add     SP,IDPLOCALLEN  ;Deallocate local storage
                pop     BP              ;Restore BP
                ret                     ;Return to caller
InitDosPart     endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       ReadFDISKParts  Read all FDISK partition tables                 ;
;                                                                       ;
;       Input   none                                                    ;
;                                                                       ;
;       Output  none    [DosFPTable and DosCylOrgAndLen tables constructed]
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ReadFDISKParts  proc    near
                push    AX              ;Save some registers
                push    BX
                push    CX
                push    DS

                mov     DS,BiosDataSeg
                assume  DS:BiosData
                mov     AL,MaxBase      ;Start with 1st possible device address
                assume  DS:nothing

                xor     BX,BX
                mov     CX,16           ;At most 16 partitions supported
                                        ; on two MAXTOR drives
        rfp1:   cmp     DosPartTable[BX+2],0
                je      rfp2            ;If no MAXTOR partition, skip
                call    ReadFDISKTable  ; else read FDISK info

        rfp2:   inc     AL
                add     BX,4
                loop    rfp1

        rfp3:   pop     DS              ;Restore regs
                pop     CX
                pop     BX
                pop     AX
                ret                     ;Return to caller
ReadFDISKParts  endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       ReadFDISKTable  Read the FDISK Partition Table for a Single     ;
;                       MAXTOR Partition                                ;
```

```
;
;       Input   AL      BIOS device address [80h-8Fh or 82h-91h]
;               BX      Offset into DosFPTable and DosCylOrgAndLen tables
;
;       Output  none    [DosFPTable and DosCylOrgAndLen entries constructed]
;
;-------------------------------------------------------------------------

;       Local storage for ReadFDISKTable
;
RFTLOCALLEN     equ     2
rftPToffset     equ     word ptr [BP-2]

                assume  DS:nothing,ES:nothing
ReadFDISKTable  proc    near
                push    BP
                mov     BP,SP           ;Establish frame pointer
                sub     SP,RFTLOCALLEN  ;Reserve storage for local variables

                push    AX              ;Save registers
                push    BX
                push    CX
                push    DX
                push    SI
                push    DI
                push    DS
                push    ES

                mov     rftPToffset,BX  ;Use later as PT index

        rft1:   push    CS              ;Establish sector buffer
                pop     ES
                mov     BX,EndOfDosDriver
                inc     BX              ;Buffer starts 1 beyond current end

                mov     AH,2            ;Read
                mov     CX,1            ; from cylinder 0 sector 1
                xor     DH,DH           ; head 0
                mov     DL,AL           ; device y
                mov     AL,1            ; 1 sector
                int     13h             ;Read using AST mechanism if necessary
                jc      rft9            ;Error reading FDISK table
                mov     DI,BX
                cmp     word ptr ES:[DI+FDSIGORG],FDSIG
                jne     rft9            ;Not initialized by FDISK

                add     DI,FDISKTABORG  ;Start of FDISK partition table
                mov     CX,4

        rft2:   cmp     byte ptr ES:[DI+4],DOSIND
                je      rft3            ;Found FDISK DOS partition info
                add     DI,16
```

```
                loop    rft2
                jmp     short rft9      ;Error, no FDISK DOS partition

        rft3:   mov     BX,rftPToffset
                mov     AX,ES:[DI+8]    ;Get relative block origin of partition
                mov     DosFPTable[BX],AX
                mov     AX,ES:[DI+12]   ;Get length of partition in blocks
                mov     DosFPTable[BX+2],AX
                xor     AH,AH
                mov     AL,ES:[DI+2]
                shl     AX,1
                shl     AX,1
                mov     AL,ES:[DI+3]    ;Get cylinder origin
                mov     DosCylOrgAndLen[BX],AX
                xor     CH,CH
                mov     CL,ES:[DI+6]
                shl     CX,1
                shl     CX,1
                mov     CL,ES:[DI+7]
                sub     CX,AX           ;Compute length in cylinders
                inc     CX
                mov     DosCylOrgAndLen[BX+2],CX

        rft9:   pop     ES              ;Restore registers
                pop     DS
                pop     DI
                pop     SI
                pop     DX
                pop     CX
                pop     BX
                pop     AX

                add     SP,RFTLOCALLEN  ;Deallocate local storage
                pop     BP              ;Restore BP
                ret                     ;Return to caller
ReadFDISKTable  endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       MediaCheck      Device Driver Media Check Function              ;
;                                                                       ;
;       Input   DS:[SI] Request Header                                  ;
;                                                                       ;
;       Output  RH updated to indicate media unchanged                  ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
MediaCheck      proc    near
                mov     AL,RH_drive             ;Keep a copy of this
                mov     mcunit,AL               ; due to a DOS 2 bug.
                mov     RH_medchkstatus,1       ;Flag media not changed
```

```
                ret
MediaCheck      endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       BuildBPB        Device Driver Build BPB Function                ;
;                                                                       ;
;       Input   DS:[SI] Request Header                                  ;
;                                                                       ;
;       Output  RH updated to point to BPB array                        ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
bpbsize         db      BPBLEN
BuildBPB        proc    near
                mov     AL,RH_drive
                add     AL,UnitOffset
                mul     bpbsize
                add     AX,offset BPB
                mov     RH_bpboffptr,AX
                mov     RH_bpbsegptr,CS
                ret
BuildBPB        endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       DeviceInfo      IOCTL Function to Return Device Information     ;
;                                                                       ;
;       Input   DS:[SI] Request Header                                  ;
;                                                                       ;
;       Output  Device Info placed in user's buffer as follows:         ;
;                                                                       ;
;               byte    Device address for use with INT 13h             ;
;               word    Number of cylinders in FDISK partition          ;
;               word    Starting cylinder within MAXTOR partition       ;
;               word    Number of blocks in FDISK partition             ;
;                                                                       ;
;-----------------------------------------------------------------------;
di_structure    struc                   ;Device Info returned via IOCTL
di_drivaddr     db      ?               ;Absolute address for BIOS interface
di_drivcyls     dw      ?               ;Number of accessible cylinders
di_drivorg      dw      ?               ;Starting cylinder number
di_drivblocks   dw      ?               ;Number of accessible blocks
di_structure    ends

                assume  DS:nothing,ES:nothing
DeviceInfo      proc    near
                mov     ES,RH_dtaseg
                mov     BX,RH_dtaoff
                mov     AL,mcunit
                add     AL,UnitOffset
                xor     CX,CX
```

```
                pushf
                push    CS
                call    DosPartitioner
                jc      di9

                mov     AL,mcunit
                add     AL,UnitOffset
                xor     AH,AH
                mov     DI,AX                   ;Save unit index for later

                push    DS
                mov     DS,BiosDataSeg
                assume  DS:BiosData
                add     AL,MaxBase
                assume  DS:nothing
                pop     DS

                shl     DI,1
                shl     DI,1                    ;Compute offset for tables

                mov     ES:di_drivaddr[BX],AL   ;Device address
                mov     DX,DosCylOrgAndLen[DI+2];Get length in cylinders
                mov     ES:di_drivcyls[BX],DX
                mov     DX,DosCylOrgAndLen[DI]  ;Get starting cylinder
                mov     ES:di_drivorg[BX],DX
                mov     DX,DosFPTable[DI+2]     ;Get length in blocks
                mov     ES:di_drivblocks[BX],DX

                mov     RH_blockcount,type di_structure
                ret

        di9:    mov     RH_blockcount,0
                ret
DeviceInfo      endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       ToggleAST       IOCTL Function to Enable/Disable AST Retry            ;
;                                                                             ;
;       Input   DS:[SI] Request Header, with data in transfer area            ;
;                       as follows:                                           ;
;                                                                             ;
;                       byte    0 = Allow AST, non-zero = Suppress AST retry  ;
;                                                                             ;
;       Output  SuppressAST flag set to match passed argument.                ;
;                                                                             ;
;-----------------------------------------------------------------------------;
ta_structure    struc                   ;Data passed from DOS IOCTL call
ta_suppressast  db      ?               ;0 = Allow AST, non-zero = suppress AST
ta_structure    ends
```

```
                assume  DS:nothing,ES:nothing
ToggleAST       proc    near
                mov     ES,RH_dtaseg
                mov     BX,RH_dtaoff
                mov     AL,ES:[BX]
                mov     SuppressAST,AL
                mov     RH_blockcount,type ta_structure
                ret
ToggleAST       endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       ReqError        Unimplement Device Driver Operation                   ;
;                                                                             ;
;       Input   DS:[SI] Request Header                                        ;
;                                                                             ;
;       Output  RH_status updated to indicate unknown command                 ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ReqError        proc    near
                mov     RH_status,RH_IOERROR+RH_BADCOMMAND
                ret
ReqError        endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       WriteVerBlocks  Write Verify One or More Blocks                       ;
;                                                                             ;
;       Input   DS:[SI] Request Header                                        ;
;                                                                             ;
;       Output  carry   ON=error OFF=no error                                 ;
;           [if carry on, RH_status has error code]                           ;
;               AX-DX   Destroyed                                             ;
;               DI      Buffer offset  address                                ;
;               ES      Buffer segment address                                ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
WriteVerBlocks  proc    near
                push    DS
                call    WriteBlocks     ;First write the data
                jc      wvx
                call    SetupDDtoBIOS   ;Setup for BIOS
                jc      wvx
                call    VerifySectors

        wvx:    call    ErrorCheck
                pop     DS
                mov     AX,blockcount   ;Update RH block count
                mov     RH_blockcount,AX
```

```
                ret
WriteVerBlocks  endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       WriteBlocks     Write One Or More Blocks                        ;
;                                                                       ;
;       Input   DS:[SI] Request Header                                  ;
;                                                                       ;
;       Output  carry   ON=error OFF=no error                           ;
;           [if carry on, RH_status has error code]                     ;
;               AX-DX   Destroyed                                       ;
;               DI      Buffer offset  address                          ;
;               ES      Buffer segment address                          ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
WriteBlocks     proc    near
                push    DS
                call    SetupDDtoBIOS   ;Setup for BIOS
                jc      wsx             ;Parameter error
                call    WriteSectors    ;Do the write operation

        wsx:    call    ErrorCheck
                pop     DS
                mov     AX,blockcount   ;Update RH block count
                mov     RH_blockcount,AX
                ret
WriteBlocks     endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       ReadBlocks      Read One Or More Blocks                         ;
;                                                                       ;
;       Input   DS:[SI] Request Header                                  ;
;                                                                       ;
;       Output  carry   ON=error OFF=no error                           ;
;           [if carry on, RH_status has error code]                     ;
;               AX-DX   Destroyed                                       ;
;               DI      Buffer offset  address                          ;
;               ES      Buffer segment address                          ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ReadBlocks      proc    near
                push    DS
                call    SetupDDtoBIOS   ;Setup for BIOS
                jc      rsx             ;Parameter error
                call    ReadSectors     ;Do the read operation

        rsx:    call    ErrorCheck
                pop     DS
```

```
                mov     AX,blockcount   ;Update RH block count
                mov     RH_blockcount,AX
                ret
ReadBlocks      endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       SetupDDtoBIOS   Transform Device Driver Data to BIOS Interface  ;
;                                                                       ;
;       Input   DS:[SI] Request Header                                  ;
;                                                                       ;
;       Output  carry   ON=error OFF=no error                           ;
;           [if carry on, RH_status has error code]                     ;
;               AH      Number of sectors to read                       ;
;               AL      Drive number            [0 thru MAXDRIVES-1]    ;
;               BH      Head number             [0 thru MAXHEADS-1]     ;
;               BL      Sector number           [0 thru MAXSECS-1]      ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]      ;
;               DI      Buffer offset address                           ;
;               ES      Buffer segment address                          ;
;           [if carry off]                                              ;
;               DS      BIOS data segment address                       ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
SetupDDtoBIOS   proc    near
                push    DX              ;Save DX
                mov     BL,RH_drive     ;Get unit number
                add     BL,UnitOffset   ; (skip over units handled by DOS)
                xor     BH,BH
                shl     BX,1            ;Compute offset in DosFPTable
                shl     BX,1            ; and DosCylOrgAndLen

                mov     AX,RH_startblock
                mov     startblock,AX   ;For error message use
                mov     blockcount,0    ;Number of blocks transferred so far
                mov     ES,RH_dtaseg
                mov     DI,RH_dtaoff

                cmp     DosCylOrgAndLen[BX+2],0
                je      sdbf            ;No FDISK partition defined
                cmp     AX,DosFPTable[BX+2]
                jae     sdb8
                add     AX,DosFPTable[BX]       ;Adjust origin to within
                                        ;         FDISK defined partition
        sdb1:   mov     BX,AX
                xor     CX,CX
                mov     AX,RH_blockcount
                call    CvtToGeometric

                mov     AL,RH_drive
                add     AL,UnitOffset   ;Skip over booted partition if any
```

```
                pushf
                push    CS
                call    DosPartitioner
                mov     dddrive,AL
                mov     ddcyl,CX
                mov     ddhead,BH
                mov     ddsector,BL
                jc      sdb8

                mov     DS,BiosDataSeg
                pop     DX              ;Restore DX
                clc                     ;Clear carry flag
                ret

        sdb8:   mov     RH_status,RH_IOERROR+RH_SEEKERROR

        sdb9:   stc                     ;Sector number out of range
                pop     DX              ;Restore DX
                ret

;       The following exception handling is provided to permit
;       IOCTL calls to reach the driver, even when no DOS partition
;       exists (DOS insists on reading the FAT first).
;
        sdbf:   cmp     AX,1            ;Attempt by DOS to read FAT?
                jne     sdb8            ;No, just fail request
                cmp     RH_command,READCOMMAND
                jne     sdb8
                cmp     RH_blockcount,1
                jne     sdb8
                mov     blockcount,1    ;Supply a fake FAT temporarily
                xor     AL,AL
                mov     CX,SECSIZE
                cld
                push    DI
                rep     stosb
                pop     DI
                mov     byte ptr ES:[DI],MEDIADESC
                mov     word ptr ES:[DI+1],0FFFFh
                mov     IgnoreError,1
                jmp     sdb9
SetupDDtoBIOS   endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       ErrorCheck      Check for errors from BIOS                      ;
;                                                                       ;
;       Input   carry   ON=error, OFF=no error                          ;
;           [if carry on                                                ;
;               AH      Error code from BIOS]                           ;
;                                                                       ;
```

```
;       Output  DS:[SI] Set to point to request header                          ;
;               RH_status updated with DOS style error code                    ;
;                                                                              ;
;-------------------------------------------------------------------------------
                assume  DS:nothing,ES:nothing
ErrorCheck      proc    near
                mov     DS,RHseg        ;Restore RH addressability
                mov     SI,RHoff
                mov     ZStatus,DX      ;Set Maxtor status field
                jc      ec1             ;Did an error occur?

        ec0:    mov     ZStatus,0       ;No, zero Maxtor status field
                mov     IgnoreError,0   ;Reset IgnoreError flag
                ret                     ;Return inline

        ec1:    cmp     IgnoreError,0
                jne     ec0
                cmp     AH,BIOS_INITFAILED
                je      ecnr
                test    AH,BIOS_NOADDRMARK
                jnz     ecse
                test    AH,BIOS_RECNOTFOUND
                jnz     ecse
                test    AH,BIOS_BADCOMMAND
                jz      ecnr
                mov     RH_status,RH_BADCOMMAND+RH_IOERROR
                ret

        ecnr:   mov     RH_status,RH_NOTREADY+RH_IOERROR
                ret

        ecse:   mov     RH_status,RH_SEEKERROR+RH_IOERROR
                ret
ErrorCheck      endp

;-------------------------------------------------------------------------------
;                                                                              ;
;       DevDriverExit   User exit from BIOS                                    ;
;                                                                              ;
;       Input   DS      Segment address of BIOS data segment                   ;
;                                                                              ;
;       Output  carry   ON=abort operation, OFF=continue with operation        ;
;                                                                              ;
;-------------------------------------------------------------------------------
                assume  DS:nothing,ES:nothing
DevDriverExit   proc    far
                jnc     dde1            ;Error so far?
                cmp     SuppressAST,0   ;Has AST function been disabled?
                je      dde0            ;No, give it a try
                stc                     ;Yes, indicate error
                jmp     short dde1      ; and return via BIOS
```

```
        dde0:   call    RecoveryViaAST  ;Yes, try to correct
                jnc     dde1            ;Ok, all corrected now
                ret                     ;Could not correct, return failure

        dde1:   pushf                   ;Don't change flags unless abort needed

                jc      dde2            ;Don't bump count if I/O error
                inc     blockcount
        dde2    label   near

                popf
                ret
DevDriverExit   endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       DosPartitioner  Map Drive Number to Appropriate DOS Partition   ;
;       Note: Devices A0 and A1 are considered unpartitioned MAXTORs.   ;
;                                                                       ;
;       Input   AL      Logical drive           [0 thru LogicalDisks-1] ;
;               CX      Cylinder logical drive                          ;
;                                                                       ;
;       Output  AL      Physical drive address  [0 thru PhysicalDrives-1] ;
;               CX      Absolute cylinder       [0 thru MAXCYLS-1]      ;
;               DX      Number of cylinders in partition                ;
;               Carry   ON=range error, OFF=everything OK               ;
;               AH      Error code if carry is ON, else undisturbed     ;
;                                                                       ;
;-----------------------------------------------------------------------;

                assume  DS:BiosData,ES:nothing
DosPartitioner  proc    near
                push    BX              ;Save working regs
                push    SI
                push    DS
                mov     DS,BiosDataSeg

                add     AL,MaxBase
                cmp     AL,0A0h         ;Absolute drive with no partitioning?
                jb      dp0
                sub     AL,0A0h
                mov     DX,MAXCYLS
                jmp     short dp3

        dp0:    sub     AL,MaxBase
                mov     BL,AL           ;Convert logical drive to table offset
                xor     BH,BH
                shl     BX,1
                shl     BX,1

                lea     SI,DosPartTable[BX]
                mov     DX,word ptr CS:[SI+2]
```

```
                cmp     DX,0            ;Does partition exist?
                je      dpe             ;No, error
                cmp     CX,DX           ;Is request within range of partition?
                jae     dpe             ;No, error
                add     CX,word ptr CS:[SI]     ;Yes, compute absolute cyl

                cmp     AL,Drv1PartLimit
                jae     dp1
                mov     AL,0
                jmp     short dp3

        dp1:    cmp     AL,Drv2PartLimit
                jae     dp2
                mov     AL,1
                jmp     short dp3

        dp2:    mov     AH,BIOS_TIMEOUT ;Drive does not exist
                stc
                jmp     short dp9

        dp3:    clc                     ;Clear carry flag

        dp9:    pop     DS              ;Restore working regs
                pop     SI
                pop     BX

        farret  proc    far
                ret     2               ;Return to caller
        farret  endp

        dpe:    stc                     ;Indicate error resolving partition
                mov     AH,BIOS_RECNOTFOUND
                jmp     dp9
DosPartitioner  endp

;--------------------------------------------------------------------------
;                                                                         ;
;       PrintReqHdr     Display Request Header Info                       ;
;                                                                         ;
;       Input   DS:[SI] Request Header                                    ;
;                                                                         ;
;       Output  None                                                      ;
;                                                                         ;
;--------------------------------------------------------------------------
                assume  DS:nothing,ES:nothing
PrintReqHdr     proc    near
                cmp     DebugFlag,0
                jne     prh1
                ret

        prh1:   push    AX
                push    BX
```

```
                push    CX
                call    PrintString
                db      CR,'Maxdd RH: op=',0
                mov     AL,RH_command
                call    PrintHexByte
                call    PrintString
                db      ' unit=',0
                mov     AL,RH_drive
                call    PrintHexByte
                mov     CL,RH_length
                cmp     CL,13
                jbe     prh9
                call    PrintString
                db      ' data=',0
                lea     BX,RH_data
                xor     CH,CH
                sub     CL,13

        prh2:   mov     AL,DS:[BX]
                call    PrintHexByte
                inc     BX
                loop    prh2

        prh9:   call    PrintCRLF
                pop     CX
                pop     BX
                pop     AX
                ret
PrintReqHdr     endp

;-------------------------------------------------------------------------------
;                                                                              ;
;       PrintError      Display a Message with Error Info                      ;
;                                                                              ;
;       Input   DS:[SI] Request Header                                         ;
;                                                                              ;
;       Output  None                                                           ;
;                                                                              ;
;-------------------------------------------------------------------------------
                assume  DS:nothing,ES:nothing
PrintError      proc    near
                cmp     DebugFlag,0
                jne     pe1
                ret

        pe1:    push    AX
                call    PrintString
                db      CR,'Error: cmd=',0
                mov     AL,RH_command
                call    PrintHexByte
                call    PrintString
                db      ' unit=',0
```

```
                mov     AL,RH_drive
                call    PrintHexByte
                call    PrintString
                db      ' status=',0
                mov     AX,RH_status
                call    PrintHexWord
                call    PrintString
                db      ' block=',0
                mov     AX,startblock
                call    PrintHexWord
                call    PrintString
                db      ' cyl=',0
                mov     AX,ddcyl
                call    PrintHexWord
                call    PrintString
                db      ' hd=',0
                mov     AL,ddhead
                call    PrintHexByte
                call    PrintString
                db      ' sec=',0
                mov     AL,ddsector
                call    PrintHexByte
                call    PrintString
                db      ' zfc=',0
                mov     AX,ZStatus
                call    PrintHexWord
                call    PrintCRLF
                pop     AX
                ret
PrintError      endp

Cseg            ends
                end
```

Formatted by SCRIPT from file 'MAXUTILS PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
                page    5o,132
                title   'Utility package for MAXTOR 1140 Hard Disk Support'

;-----------------------------------------------------------------------;
;                                                                       ;
;       Utility package for MAXTOR 1140 Hard Disk Support               ;
;       Program Property of IBM                                         ;
;                                                                       ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1       ;
;       Version:        1.6 0o/26/84                                    ;
;       DOS Required:   None                                            ;
;       Classification: IBM Internal Use Only                           ;
;                                                                       ;
;       (C) Copyright IBM Corporation, 1984                             ;
;       Developed for the Information Technology Center at              ;
;       Carnegie-Mellon University                                      ;
;                                                                       ;
;-----------------------------------------------------------------------;

                include maxdata.dcl     ;Data seg for MAXTOR BIOS support

                public  CvtToGeometric
                public  DetPhysDrives
                public  GetOrigDrives
                public  PrintCRLF
                public  PrintHexByte
                public  PrintHexWord
                public  PrintString

Cseg            segment public 'code'
                assume  CS:Cseg,SS:nothing

IBMCopyright    segment byte common 'IBMCopyright'
                db      ' (C) Copyright IBM Corporation, 1984 '
IBMCopyright    ends

                extrn   VerifySectors:near
                extrn   ReadSectors:near

CR              equ     13              ;Carriage return
LF              equ     10              ;Line feed
BiosDataSeg     dw      seg BiosData

;-----------------------------------------------------------------------;
;                                                                       ;
;       CvtToGeometric  Convert Block Number to Geometric Address       ;
;                                                                       ;
;       Input   AX      Block count                                     ;
;               BX      Low  order word of block number                 ;
;               CX      High order word of block number                 ;
;                                                                       ;
;       Output  AH      Number of sectors to read                       ;
```

```
;               AL      Destroyed                                               ;
;               BH      Head number             [0 thru MAXHEADS-1]             ;
;               BL      Sector number           [0 thru MAXSECS-1]              ;
;               CX      Cylinder number         [0 thru MAXCYLS-1]              ;
;                                                                               ;
;-------------------------------------------------------------------------------;

;       Constants for CvtToGeometric
;
SectorsPerHead  db      MAXSECS         ;Sectors per track
SectorsPerCyl   dw      (MAXSECS*MAXHEADS)      ;Sectors per cylinder

;       Local storage for CvtToGeometric
;
CTGLOCALLEN     equ     6
ctgblocklo      equ     word ptr [BP-6]
ctgblockhi      equ     word ptr [BP-4]
ctgblockcnt     equ     word ptr [BP-2]
ctgblockcnlo    equ     byte ptr [BP-2]

                assume  DS:nothing,ES:nothing
CvtToGeometric  proc    near
                push    BP
                mov     BP,SP           ;Establish frame pointer
                sub     SP,CTGLOCALLEN  ;Reserve storage for local variables
                push    DX              ;Preserve DX also
                mov     ctgblockcnt,AX
                mov     ctgblocklo,BX
                mov     ctgblockhi,CX

                mov     AX,ctgblocklo   ;Get block number
                mov     DX,ctgblockhi
                div     SectorsPerCyl
                mov     CX,AX           ;Set starting cylinder number

                mov     AX,DX           ;DX = remaining sectors
                div     SectorsPerHead  ;AL = head, AH = sector
                mov     BH,AL           ;Set starting head, sector number
                mov     BL,AH

                mov     AH,ctgblockcnlo ;Set sector count

                pop     DX              ;Restore DX
                add     SP,CTGLOCALLEN  ;Deallocate local storage
                pop     BP              ;Restore BP
                ret                     ;Return to caller
CvtToGeometric  endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       DetPhysDrives   Determine Number of Physical Drives Attached            ;
;                                                                               ;
```

```
;       Input   none                                                    ;
;                                                                       ;
;       Output  AL      Number of drives attached to ZOBEX controller   ;
;                                                                       ;
;-----------------------------------------------------------------------;

;       Local storage for DetPhysDrives
;
DPDLOCALLEN     equ     1
dpddrvcnt       equ     byte ptr [BP-1]

                assume  DS:nothing,ES:nothing
DetPhysDrives   proc    near
                push    BP
                mov     BP,SP           ;Establish frame pointer
                sub     SP,DPDLOCALLEN  ;Reserve storage for local variables
                mov     dpddrvcnt,0     ;Initialize drive count
                push    AX              ;Save working regs
                push    BX
                push    CX
                push    DX
                push    DS

                mov     AX,0100h        ;Try Drive 0
                xor     BX,BX
                xor     CX,CX
                mov     DS,BiosDataSeg
                call    VerifySectors
                jnc     dpd1
                cmp     AH,BIOS_TIMEOUT
                je      dpd9

        dpd1:   inc     dpddrvcnt

                mov     AX,0101h        ;Try Drive 1
                xor     BX,BX
                xor     CX,CX
                mov     DS,BiosDataSeg
                call    VerifySectors
                jnc     dpd2
                cmp     AH,BIOS_TIMEOUT
                je      dpd9

        dpd2:   inc     dpddrvcnt

        dpd9:   pop     DS              ;Restore working regs
                pop     DX
                pop     CX
                pop     BX
                pop     AX
                mov     AL,dpddrvcnt    ;Return drive count in AL
                add     SP,DPDLOCALLEN  ;Deallocate local storage
```

```
                pop     BP              ;Restore BP
                ret                     ;Return to caller
DetPhysDrives   endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       GetOrigDrives   Set OrigDriveCnt field to number of drives        ;
;                       attached to the IBM adapter                       ;
;                                                                         ;
;       Input   none                                                      ;
;                                                                         ;
;       Output  AL      Number of drives attached to IBM adapter          ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:BiosData,ES:nothing
GetOrigDrives   proc    near
                push    DS              ;Save registers
                push    DX
                push    CX
                push    AX
                pushf

                mov     DS,BiosDataSeg  ;Provide addressability to data seg
                mov     AH,8            ;Op = Get Parameters
                mov     DL,80h          ;See if drive 80 attached
                pushf
                call    dword ptr OrigDiskReqOff
                jc      god2            ;Drive 80 not attached

        god1:   popf                    ;Restore registers
                pop     AX
                mov     AL,DL           ;Pass back number of IBM drives
                pop     CX
                pop     DX
                pop     DS
                ret

        god2:   mov     AH,8
                mov     DL,81h          ;Well, how about drive 81?
                pushf
                call    dword ptr OrigDiskReqOff
                jmp     god1
GetOrigDrives   endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       PrintString     Display a string                                  ;
;                                                                         ;
;       Input   SP      Offset address in code segment of where string is.;
;                       String is assumed to be terminated by hex 00.     ;
;                                                                         ;
```

```
;       Output  none    Returns to offset following 'end-of-string' byte        ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
PrintString     proc    near            ;Output string to console.
                push    AX
                push    BP
                push    BX
                mov     BP,SP
                add     BP,6
                mov     BX,[BP]

        pstr2:  mov     AL,CS:[BX]      ;Get next char
                inc     BX              ;Advance string ptr
                or      AL,AL
                jz      pstr4           ;Done if EOS
                call    PrintChar       ;Print next char
                jmp     pstr2           ;Repeat till EOS found

        pstr4:  mov     [BP],BX         ;Update return adr
                pop     BX
                pop     BP
                pop     AX
                ret
PrintString     endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       PrintChar       Display One Character                                   ;
;                                                                               ;
;       Input   AL      Character to display                                    ;
;                                                                               ;
;       Output  None                                                            ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
PrintChar       proc    near
                push    BP
                push    AX
                push    BX
                push    SI
                push    DI
                mov     AH,0Eh
                mov     BX,7
                int     10h             ;Output char to display in TTY mode
                pop     DI
                pop     SI
                pop     BX
                pop     AX
                pop     BP
                ret
PrintChar       endp
```

```
;-----------------------------------------------------------------------;
;                                                                       ;
;       PrintCRLF       Output a carriage return and line feed          ;
;                                                                       ;
;       Input   none                                                    ;
;       Output  None                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
PrintCRLF       proc    near            ;Print CR, LF at console.
                call    PrintString
                db      CR,LF,0
                ret
PrintCRLF       endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       PrintHexWord    Display a Word in Printable Hex                 ;
;                                                                       ;
;       Input   AX      Word to display                                 ;
;                                                                       ;
;       Output  None                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
PrintHexWord    proc    near
                push    AX
                mov     AL,AH
                call    PrintHexByte    ;Print high byte
                pop     AX
                push    AX
                call    PrintHexByte    ;Print low byte
                pop     AX
                ret
PrintHexWord    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       PrintHexByte    Display a Byte in Printable Hex                 ;
;                                                                       ;
;       Input   AL      Word to display                                 ;
;                                                                       ;
;       Output  None                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
PrintHexByte    proc    near
                push    AX
                shr     AL,1
                shr     AL,1
                shr     AL,1
```

```
                shr     AL,1
                call    PrintNibble     ;Print high nibble
                pop     AX
                push    AX
                call    PrintNibble     ;Print low nibble
                pop     AX
                ret
PrintHexByte    endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       PrintNibble     Display a Half-Byte in Printable Hex            ;
;                                                                       ;
;       Input   AL      Low order 4 bits have nibble to display         ;
;                                                                       ;
;       Output  None                                                    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
PrintNibble     proc    near
                and     AL,0fh
                add     AL,90h
                daa
                adc     AL,40h
                daa
                jmp     PrintChar       ;Output char, exit
PrintNibble     endp

Cseg            ends
                end
```

Formatted by SCRIPT from file 'ORDER PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
; These are macros that convert the order of multiprecision integers from
; the top's format to the 8088's format and back again
        extrn   ?wbyteorder:abs
        extrn   ?lwordorder:abs
        extrn   ?lhbyteorder:abs
        extrn   ?llbyteorder:abs
;
; This macro will take a TOP type 2 byte integer in AX, and convert it to
; an 8088 type 2 byte integer in AX
xlatword macro
        dw      ?wbyteorder
        endm
;
; This macro will take an 8088 type 2 byte integer in AX, and convert it to
; a TOP order 2 byte integer in AX
revxlatword macro
        dw      ?wbyteorder
        endm
;
; This macro will take a TOP type 4 byte integer in DXAX, and convert it to
; an 8088 type 4 byte integer in DXAX
xlatdword macro
        db      ?lwordorder
        dw      ?lhbyteorder,?llbyteorder
        endm
;
; This macro will take an 8088 type 4 byte integer in DXAX, and convert it to
; a TOP type 4 byte integer in DXAX
revxlatdword macro
        dw      ?lhbyteorder,?llbyteorder
        db      ?lwordorder
        endm
```

Formatted by SCRIPT from file 'MAXVEC PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
                page    56,132
                title   'Vector Device Class Driver for MAXTOR 1140 Hard Disk'

;-----------------------------------------------------------------------------;
;                                                                             ;
;       Vector Device Class Driver for MAXTOR 1140 Hard Disk                  ;
;       Program Property of IBM                                               ;
;                                                                             ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1             ;
;       Version:        1.9 05/24/84                                          ;
;       DOS Required:   None                                                  ;
;       Classification: IBM Internal Use Only                                 ;
;                                                                             ;
;       (C) Copyright IBM Corporation, 1984                                   ;
;       Developed for the Information Technology Center at                    ;
;       Carnegie-Mellon University                                            ;
;                                                                             ;
;-----------------------------------------------------------------------------;

                include maxdata.dcl     ;BIOS error constants

                include order.inc       ;SRITEK macros for byte/word/dword order

IBMCopyright    segment byte common 'IBMCopyright'
                db      ' (C) Copyright IBM Corporation, 1984 '
IBMCopyright    ends

                public  MaxtorWrite
                public  MaxtorRead
                public  MaxtorOpen
                public  MaxtorClose
                public  MaxtorTerm
                public  MaxtorIoctl
                public  MaxtorExist
                public  MaxtorSelect

MSGblock        segment at 0
                db      3 dup(?)        ;Reserved by SRITEK                  0-2
MSGsigbytes     db      ?               ;Length of current request            3
MSGrqtype       db      ?               ;Type of request                      4
MSGdevnum       db      ?               ;Device number                        5
MSGdone         db      ?               ;1 = Polled, 0 = Interrupt when done  6
MSGcharct       dw      ?               ;Total bytes to transfer             7-8
MSGstatus       dw      ?               ;Return status info                  9-10
                db      2 dup(?)        ;Not used by MAXTOR                 11-12
MSGaddress      db      4 dup(?)        ;DASD address                       13-16
MSGvectorlist   db                      ;Vector list starts here              17
  MSGbytect     equ     word  ptr MSGvectorlist+0  ;Each entry has a byte count
  MSGphysaddr   equ     dword ptr MSGvectorlist+2  ; & starting coprocessor addr
MSGblock        ends

Cseg            segment byte public 'code'
```

```
                assume  CS:cseg,SS:nothing

;       SRITEK routines
;
                extrn   addbatch:near
                extrn   gather:near
                extrn   getheap:near
                extrn   returnheap:near
                extrn   scatter:near
                extrn   vdone:near
                extrn   vwhois:near

;       MAXTOR BIOS routines
;
                extrn   CvtToGeometric:near
                extrn   DetPhysDrives:near
                extrn   PrintCRLF:near
                extrn   PrintHexByte:near
                extrn   PrintHexWord:near
                extrn   PrintString:near

                extrn   ObtainAST:near
                extrn   RecoveryViaAST:near

                extrn   DeselectDrive:near
                extrn   ReadSectors:near
                extrn   ResetDisk:near
                extrn   WriteSectors:near

;       Computed Constants
;
SectorSize      dw      SECSIZE         ;Bytes per sector
BiosDataSeg     dw      seg BiosData    ;Data area for MAXTOR BIOS support
PartHeader      db      'Partition Record'      ;Header in Partition Record

;       Local constants
;
VEC_SOFTWERR    equ     2800h           ;Indicates software detected error
VEC_NOTERM      equ     VEC_SOFTWERR+1  ;Nothing to terminate
VEC_FORCEDTERM  equ     VEC_SOFTWERR+2  ;Request forcibly terminated
VEC_OUTOFRANGE  equ     VEC_SOFTWERR+3  ;Sector address out of range
VEC_TIMEOUT     equ     VEC_SOFTWERR+4  ;Software timeout occurred

BUFSIZE         equ     32*1024         ;Maximum buffer size
CR              equ     13              ;Carriage return
LF              equ     10              ;Line feed
FIRSTMAXTOR     equ     8               ;1st  position in SRITEK Vector Table
DDPARTITION     equ     8000h           ;Non-vector partition

;       Private variables
;
DebugFlag       db      0               ;Non-zero means produce debugging output
```

```
buflen          dw      0               ;Length of buffer
bufoff          dw      0               ;Offset of PC disk transfer area
bufseg          dw      0               ;Seg    of PC disk transfer area
dtaoff          dw      ?               ;Offset pointer used by read/write
dtaseg          dw      ?               ;Seg    pointer used by read/write
blockcount      dw      ?               ;Number of blocks actually transferred
reqstatus       dw      ?               ;Internal status field
AST1seg         dw      0               ;Storage segment for 1st AST
AST2seg         dw      0               ;Storage segment for 2nd AST
TermRequested   db      0               ;1 = Forcible Termination requested
Device          db      0FFh            ;Current device
DriverBusy      db      0               ;1 = I/O request in progress
DriveCount      db      0               ;Number of MAXTOR drives attached
UnixPartTable1  dw      16 dup(0)       ;Describes drive 0 UNIX partitions a-h
UnixPartTable2  dw      16 dup(0)       ;Describes drive 1 UNIX partitions a-h
IOBuf           db      512 dup(?)      ;I/O Buffer for UNIX partition init

;-----------------------------------------------------------------------------;
;                                                                             ;
;       MaxtorRead      Read From MAXTOR via Vector Interface                 ;
;                                                                             ;
;       Input   ES:[3]  Pointer to MSG Block                                  ;
;                                                                             ;
;       Output  none    [Batch read scheduled]                                ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
MaxtorRead      proc    near
                mov     BX,offset BatchRead
                jmp     short MaxtorRW

;-----------------------------------------------------------------------------;
;                                                                             ;
;       MaxtorWrite     Write to MAXTOR via Vector Interface                  ;
;                                                                             ;
;       Input   ES:[3]  Pointer to MSG Block                                  ;
;                                                                             ;
;       Output  none    [Batch write scheduled]                               ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
MaxtorWrite     proc    near
                mov     BX,offset BatchWrite

MaxtorRW        proc    near
                call    addbatch
                ret
MaxtorRW        endp
MaxtorWrite     endp
MaxtorRead      endp
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       MaxtorOpen      Open MAXTOR for Vector Interface                  ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG Block                              ;
;                                                                         ;
;       Output  none    [Batch open scheduled]                            ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
MaxtorOpen      proc    near
                mov     BX,offset BatchOpen
                call    addbatch
                ret
MaxtorOpen      endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       MaxtorClose     Close MAXTOR Vector Interface                     ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG Block                              ;
;                                                                         ;
;       Output  none    [vdone called to return status in MSG block]      ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
MaxtorClose     proc    near
                mov     AX,AST1seg      ;Release any AST buffers
                cmp     AX,0
                je      mc1
                call    returnheap
                mov     AST1seg,0

        mc1:    mov     AX,AST2seg
                cmp     AX,0
                je      mc2
                call    returnheap
                mov     AST2seg,0

        mc2:    mov     AX,bufseg       ;Release data buffers
                cmp     AX,0
                je      mc3
                call    returnheap
                mov     bufseg,0

        mc3:    xor     AX,AX           ;Status = 0
                xor     CX,CX           ;Chars transferred
                call    vdone
                ret
MaxtorClose     endp
```

```
;-----------------------------------------------------------------------;
;                                                                       ;
;       MaxtorTerm      Forcibly Terminate Current Vector Operation     ;
;                                                                       ;
;       Input   ES:[3]  Pointer to MSG Block                            ;
;                                                                       ;
;       Output  none    [vdone called to return status in MSG block]    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
MaxtorTerm      proc    near
                cli
                cmp     DriverBusy,0    ;Is anything going on?
                je      mt1             ;No, nothing to terminate
                call    GetDevnum       ;Yes, is it for the indicated
                cmp     AL,Device       ;  device?
                jne     mt1             ;No, fail term request
                mov     TermRequested,1 ;Yes, cause termination to occur
                mov     AX,blockcount
                mov     BX,SECSIZE
                mul     BX
                mov     CX,AX
                xor     AX,AX
                jmp     short mt2

        mt1:    mov     AX,VEC_NOTERM   ;Nothing to terminate
                xor     CX,CX           ;No chars transferred for this req.

        mt2:    sti
                call    vdone
                ret
MaxtorTerm      endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       MaxtorIoctl     IOCTL for MAXTOR Vector Interface               ;
;                                                                       ;
;       Input   ES:[3]  Pointer to MSG Block                            ;
;                                                                       ;
;       Output  none    [vdone called to return status in MSG block]    ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
MaxtorIoctl     proc    near
                xor     AX,AX           ;Status = 0
                xor     CX,CX           ;Chars transferred
                call    vdone
                ret
MaxtorIoctl     endp

;-----------------------------------------------------------------------;
;                                                                       ;
```

```
;       MaxtorExist     Determine Whether Devnum Unit Exists                    ;
;                                                                               ;
;       Input   ES:[3]  Pointer to MSG Block                                    ;
;                                                                               ;
;       Output  none    [# cylinders, heads/cyl, sectors/head, & sector         ;
;                       size passed back in as data via the MSG Block vectors.  ;
;                       # cylinders = 0 if Devnum does not exist]               ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
Ecylcnt         dw      0
Ehdspercyl      dw      MAXHEADS
Esecsperhd      dw      MAXSECS
Esecsize        dw      SECSIZE
MaxtorExist     proc    near
                xor     CX,CX
                call    GetDevnum       ;Get device number
                mov     BL,AL           ;Convert to partition table index
                xor     BH,BH
                shl     BX,1
                shl     BX,1
                mov     BX,word ptr CS:UnixPartTable1[BX+2]
                mov     Ecylcnt,BX      ;Zero means partition does not exist
                mov     SI,offset Ecylcnt
                mov     CX,8
                call    vwhois
                ret
MaxtorExist     endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       MaxtorSelect    Indicates Whether Next Operation Will Block             ;
;                                                                               ;
;       Input   ES:[3]  Pointer to MSG Block                                    ;
;                                                                               ;
;       Output  none    [vdone called to return status in MSG block]            ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
IWillBlock      db      1,1
IWillNotBlock   db      0,0
MaxtorSelect    proc    near
                cmp     DriverBusy,0
                je      ms2             ;No block will occur
                mov     SI,offset IWillBlock

        ms1:    mov     CX,2
                call    vwhois
                ret

        ms2:    mov     SI,offset IWillNotBlock
```

```
                jmp     ms1
MaxtorSelect    endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       BatchRead       Read Data from MAXTOR                             ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG Block                              ;
;                                                                         ;
;       Output  none    [vdone called to return status in MSG block]      ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
BatchRead       proc    near
                mov     DriverBusy,1    ;Indicate Driver now busy
                mov     blockcount,0    ;Clear blocks transferred
                call    DumpMSGBlock    ;Display MSG block for debugging
                call    GetBuffer       ;Acquire a buffer if necessary
                call    ReadBlocks

                mov     AX,MSGcharct    ;Get original byte count
                xlatword
                mov     CX,AX           ;Assume this much transferred
                cmp     reqstatus,0     ;Unless errors occurred
                je      mr1
                mov     AX,SECSIZE      ;Pass back correct count
                mul     blockcount
                mov     CX,AX

        mr1:    mov     AX,MSGcharct    ;Get original byte count
                xlatword
                mov     CX,AX           ;This is the amount to scatter
                mov     DI,offset MSGvectorlist
                mov     SI,bufoff       ;This is where to scatter from
                mov     DS,bufseg
                xor     BP,BP
                call    scatter         ;Copy into coprocessor memory

                mov     AX,reqstatus
                mov     DriverBusy,0    ;Driver no longer busy
                call    vdone
                ret
BatchRead       endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       BatchWrite      Write Data to MAXTOR                              ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG Block                              ;
;                                                                         ;
```

```
;       Output  none    [vdone called to return status in MSG block]          ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
BatchWrite      proc    near
                mov     DriverBusy,1    ;Indicate Driver now busy
                mov     blockcount,0    ;Clear blocks transferred
                call    DumpMSGBlock    ;Display MSG block for debugging
                call    GetBuffer       ;Acquire buffer if necessary
                mov     AX,MSGcharct    ;Get original byte count
                xlatword
                mov     CX,AX           ;This is the amount to gather
                mov     DI,offset MSGvectorlist
                mov     SI,bufoff       ;This is where to gather into
                mov     dtaoff,SI
                mov     DS,bufseg
                mov     dtaseg,DS
                xor     BP,BP
                call    gather          ;Copy from coprocessor memory

                call    WriteBlocks     ;Now write it to the device

                mov     AX,MSGcharct    ;Get original byte count
                xlatword
                mov     CX,AX           ;Assume this much transferred
                cmp     reqstatus,0     ;Unless errors occurred
                je      mw1
                mov     AX,SECSIZE      ;Pass back correct count
                mul     blockcount
                mov     CX,AX

        mw1:    mov     AX,reqstatus
                mov     DriverBusy,0    ;Indicate Driver no longer busy
                call    vdone
                ret
BatchWrite      endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       BatchOpen       Determines how many drives attached [maximum of 2]    ;
;                       In-storage partition table inititialized              ;
;                                                                             ;
;       Input   ES:[3]  Pointer to MSG block                                  ;
;                                                                             ;
;       Output  none    [vdone called to return status in MSG block]          ;
;                                                                             ;
;-----------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
BatchOpen       proc    near
                call    DumpMSGBlock    ;Display MSG block for debugging
                call    Initialize
                xor     AX,AX           ;Status = 0
```

```
                xor     CX,CX           ;Chars transferred
                call    vdone
                ret
BatchOpen       endp

;-------------------------------------------------------------------------------
;                                                                              ;
;       Initialize      Initialize MAXTOR Device(s)                            ;
;                                                                              ;
;       Input   none                                                           ;
;                                                                              ;
;       Output  none                                                           ;
;                                                                              ;
;-------------------------------------------------------------------------------
                assume  DS:nothing,ES:MSGblock
Initialize      proc    near
                push    ES
                push    DS
                push    AX
                push    BX
                push    CX
                push    DX
                mov     DS,BiosDataSeg
                call    ResetDisk               ;Reset ZOBEX controller
                call    DetPhysDrives           ;Find out how many drives
                mov     DS:PhysicalDrives,AL
                mov     DriveCount,AL
                cmp     AL,0
                je      i2

                push    CS
                pop     ES
                mov     DI,offset IOBuf
                dec     AL                      ;Cvt drive number to drive addr

                cmp     AL,1
                jne     i1
                mov     DX,offset UnixPartTable2
                call    InitUnixPart            ;Initialize 2nd UNIX Part Table
                call    GetAST
                dec     AL

        i1:     cmp     AL,0
                jne     i2
                mov     DX,offset UnixPartTable1
                call    InitUnixPart            ;Initialize 1st UNIX Part Table
                push    AST1seg
                pop     AST2seg
                call    GetAST
        i2      label   near

                call    DeselectDrive
```

```
                pop     DX
                pop     CX
                pop     BX
                pop     AX
                pop     DS
                pop     ES
                ret
Initialize      endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       BiosUserExit    Checks if Current Operation Should be Aborted     ;
;                                                                         ;
;       Input   none                                                      ;
;                                                                         ;
;       Output  carry   ON=abort, OFF=continue                            ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
BiosUserExit    proc    far
                jnc     buel
                call    RecoveryViaAST  ;Try alternate sector
                jnc     buel            ;Recovered with alternate sector
                ret                     ;Could not recover, fail I/O request

        buel:   cmp     TermRequested,0 ;Forced termination?
                je      bue2            ;No, continue
                mov     TermRequested,0
                or      reqstatus,VEC_FORCEDTERM
                stc                     ;Indicate error occurred
                ret

        bue2:   add     word ptr dtaoff,SECSIZE ;Advance DTA
                inc     blockcount
                clc                     ;Clear carry flag
                ret
BiosUserExit    endp


;-------------------------------------------------------------------------;
;                                                                         ;
;       ReadBlocks      Read Blocks using BIOS Read Sector Interface      ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG block                              ;
;                                                                         ;
;       Output  reqstatus                                                 ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
ReadBlocks      proc    near
                call    GetDASDAddr
                call    CharctToBlockct
```

```
                call    CvtToGeometric
                call    GetDevnum
                call    AbsDriveAndCyl  ;Resolve partition info
                jc      rb9             ;Partitioning Error

                push    ES
                mov     DI,bufoff       ;Setup contiguous buffer pointers
                mov     dtaoff,DI
                mov     ES,bufseg
                mov     dtaseg,ES
                mov     DS,BiosDataSeg
                assume  DS:BiosData,ES:nothing

                push    word ptr TerminateExit
                push    word ptr TerminateExit+2
                cli
                mov     word ptr TerminateExit,offset BiosUserExit
                mov     word ptr TerminateExit+2,CS
                sti
                call    ReadSectors     ;Read data from device
                pop     word ptr TerminateExit+2
                pop     word ptr TerminateExit

                pushf
                call    DeselectDrive
                popf

                pop     ES
                assume  DS:nothing,ES:MSGblock
                call    ErrorCheck
        rb9:    ret
ReadBlocks      endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       WriteBlocks      Write Blocks using BIOS Read Sector Interface    ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG block                              ;
;                                                                         ;
;       Output  reqstatus                                                 ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
WriteBlocks     proc    near
                call    GetDASDAddr
                call    CharctToBlockct
                call    CvtToGeometric
                call    GetDevnum
                call    AbsDriveAndCyl  ;Resolve partition info
                jc      wb9             ;Partitioning error

                push    ES
```

```
                mov     DI,bufoff       ;Setup contiguous buffer pointers
                mov     dtaoff,DI
                mov     ES,bufseg
                mov     dtaseg,ES
                mov     DS,BiosDataSeg
                assume  DS:BiosData,ES:nothing

                push    word ptr DS:TerminateExit
                push    word ptr DS:TerminateExit+2
                cli
                mov     word ptr DS:TerminateExit,offset BiosUserExit
                mov     word ptr DS:TerminateExit+2,CS
                sti
                call    WriteSectors    ;Write data to device
                pop     word ptr DS:TerminateExit+2
                pop     word ptr DS:TerminateExit

                pushf
                call    DeselectDrive
                popf

                pop     ES
                assume  DS:nothing,ES:MSGblock
                call    ErrorCheck
        wb9:    ret
WriteBlocks     endp

;-----------------------------------------------------------------------;
;                                                                       ;
;       CharctToBlockct Convert Character Count to Block Count          ;
;                                                                       ;
;       Input   ES:[3]  Pointer to MSG block                            ;
;                                                                       ;
;       Output  AX      Block count (rounded up if necessary)           ;
;                                                                       ;
;-----------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
CharctToBlockct proc    near
                push    DX
                mov     AX,MSGcharct
                xlatword
                xor     DX,DX
                div     SectorSize
                cmp     DX,0
                je      ctb1
                inc     AX

        ctb1:   pop     DX
                ret
CharctToBlockct endp
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       GetDASDAddr     Get DASD Block Address into BX,CX                 ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG block                              ;
;                                                                         ;
;       Output  BX      Low  order word of block number                   ;
;               CX      High order word of block number                   ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
GetDASDAddr     proc    near
                push    AX
                push    DX
                mov     AX,word ptr MSGaddress
                mov     DX,word ptr MSGaddress+2
                xlatdword
                mov     CX,DX
                mov     BX,AX
                pop     DX
                pop     AX
                ret
GetDASDAddr     endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       AbsDriveAndCyl  Determine Physical Drive and Absolute Cyl Address ;
;                                                                         ;
;       Input   AL      Logical Drive           [0-15]                    ;
;               CX      Cylinder logical drive  [0 thru UnixPartTable entry] ;
;                                                                         ;
;       Output  AL      Physical drive address  [0 thru PhysicalDrives-1] ;
;               CX      Absolute cylinder       [0 thru MAXCYLS-1]        ;
;               Carry   ON=range error, OFF=everything OK                 ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
AbsDriveAndCyl  proc    near
                push    DX
                mov     DX,offset UnixPartTable1
                call    UnixPartitioner ;Determine partition
                jnc     adac9           ;Check for error
                mov     reqstatus,VEC_OUTOFRANGE

        adac9:  pop     DX
                ret
AbsDriveAndCyl  endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       GetDevnum       Get Adjusted Device number into AL                ;
;                                                                         ;
```

```
;       Input   ES:[3]  Pointer to MSG block                                    ;
;                                                                               ;
;       Output  AL      Logical device number   [0-15]                          ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
GetDevnum       proc    near
                mov     AL,MSGdevnum
                sub     AL,FIRSTMAXTOR  ;1st position in SRITEK vhandler table
                ret
GetDevnum       endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       ErrorCheck      Check for errors from BIOS                              ;
;                                                                               ;
;       Input   carry   ON=error, OFF=no error                                  ;
;           [if carry on                                                        ;
;               AH      Error code from BIOS]                                   ;
;               DX      Zobex error and status registers                        ;
;                                                                               ;
;       Output  reqstatus updated with vector style error code                  ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
ErrorCheck      proc    near
                jc      ec1             ;Did an error occur?
                mov     reqstatus,0     ;No, set good status
                ret                     ;Return

        ec1:    cmp     AH,BIOS_TIMEOUT ;Did software timeout occur?
                jne     ec2
                mov     DX,VEC_TIMEOUT

        ec2:    mov     reqstatus,DX    ;Set return status
                ret
ErrorCheck      endp

;-------------------------------------------------------------------------------;
;                                                                               ;
;       GetBuffer       Acquire a Buffer from the SRITEK IOP                    ;
;                                                                               ;
;       Input   none                                                            ;
;                                                                               ;
;       Output  none                                                            ;
;                                                                               ;
;-------------------------------------------------------------------------------;
                assume  DS:nothing,ES:nothing
GetBuffer       proc    near
                cmp     bufseg,0        ;Have we acquired buffer yet?
                jne     gb1             ;Yes, exit
                mov     AX,BUFSIZE
```

```
                call    getheap
                mov     bufseg,AX
                mov     bufoff,0
                mov     buflen,CX
        gbl:    ret
GetBuffer       endp

;-------------------------------------------------------------------------------
;                                                                               ;
;       InitUnixPart    Initialize Unix Partition Table                         ;
;                                                                               ;
;       Input   AL      Physical drive                                          ;
;               DX      Offset from CS for Table                                ;
;               ES:[DI] Buffer area for 1 Sector                                ;
;                                                                               ;
;       Output  none    [Table placed at offset in DX]                          ;
;                                                                               ;
;-------------------------------------------------------------------------------

;       Local storage for InitUnixPart
;
IUPLOCALLEN     equ     6
iuptaboff       equ     word ptr [BP-6]
iupbufoff       equ     word ptr [BP-4]
iupbufseg       equ     word ptr [BP-2]

                assume  DS:nothing,ES:nothing
InitUnixPart    proc    near
                push    BP
                mov     BP,SP           ;Establish frame pointer
                sub     SP,IUPLOCALLEN  ;Reserve storage for local variables

                push    AX              ;Save regs
                push    BX
                push    CX
                push    DX
                push    SI
                push    DI
                push    DS
                push    ES
                mov     iupbufoff,DI    ;Save parameters
                mov     iupbufseg,ES
                mov     iuptaboff,DX

                mov     AH,1            ;Read 1 sector Partition Record
                mov     BH,PARTRECHD    ; [drive is already in AL]
                mov     BL,PARTRECSEC
                mov     CX,PARTRECCYL
                mov     DS,BiosDataSeg
                call    ReadSectors
                jc      iup9
```

```
                push    CS              ;Copy partition table
                pop     ES
                mov     DS,iupbufseg
                mov     SI,iupbufoff
                mov     CX,16
                mov     DI,offset PartHeader
                cld
                repe    cmpsb
                jne     iup9

                mov     DI,iuptaboff
                mov     CX,16
                rep     movsw

                mov     CX,8            ;Eliminate any non-vector partitions
                mov     BX,iuptaboff

        iup1:   test    word ptr CS:[BX],DDPARTITION
                jz      iup2
                mov     word ptr CS:[BX],0
                mov     word ptr CS:[BX+2],0

        iup2:   add     BX,4
                loop    iup1

        iup9:   call    DetPhysDrives   ;Seek back to Cyl 0
                pop     ES              ;Restore regs
                pop     DS
                pop     DI
                pop     SI
                pop     DX
                pop     CX
                pop     BX
                pop     AX

                add     SP,IUPLOCALLEN  ;Deallocate local storage
                pop     BP              ;Restore BP
                ret                     ;Return to caller
InitUnixPart    endp

;-------------------------------------------------------------------------------
;                                                                              ;
;       UnixPartitioner Map Drive Number to Appropriate UNIX Partition         ;
;                                                                              ;
;       Input   AL      Logical drive           [0 thru 15]                    ;
;               CX      Cylinder logical drive  [0 thru UnixPartTable entry]   ;
;               DX      Offset (from CS) of Unix partition tables              ;
;                                                                              ;
;       Output  AL      Physical drive address  [0 thru PhysicalDrives-1]      ;
;               CX      Absolute cylinder       [0 thru MAXCYLS-1]             ;
```

```
;               Carry   ON=range error, OFF=everything OK                    ;
;                                                                             ;
;-----------------------------------------------------------------------------;

                assume  DS:nothing,ES:nothing
UnixPartitioner proc    near
                push    BX              ;Save working regs
                push    DX
                push    SI

                mov     BL,AL           ;Convert logical drive to table offset
                xor     BH,BH
                shl     BX,1
                shl     BX,1

                add     DX,BX
                mov     SI,DX
                mov     DX,word ptr CS:[SI+2]
                cmp     DX,0            ;Does partition exist?
                je      upe             ;No, error
                cmp     CX,DX           ;Is request within range of partition?
                jae     upe             ;No, error
                add     CX,word ptr CS:[SI]     ;Yes, compute absolute cyl

                and     AL,08h          ;Convert logical to physical drive
                shr     AL,1
                shr     AL,1
                shr     AL,1

                clc                     ;Clear carry flag

        up9:    pop     SI              ;Restore working regs
                pop     DX
                pop     BX
                ret                     ;Return to caller

        upe:    stc                     ;Indicate error resolving partition
                jmp     up9
UnixPartitioner endp

;-----------------------------------------------------------------------------;
;                                                                             ;
;       GetAST          Get the Alternate Sector Table                        ;
;                                                                             ;
;       Input   AL      Physical drive                                        ;
;                                                                             ;
;       Output  none    [AST obtained]                                        ;
;                                                                             ;
;-----------------------------------------------------------------------------;

                assume  DS:nothing,ES:nothing
GetAST          proc    near
                push    AX              ;Save regs
```

```
                push    CX
                push    DX
                push    DI
                push    ES
                push    DS

                mov     DL,AL           ;Save drive number for a moment
                mov     AX,SECSIZE*ASTSECCNT
                call    getheap         ;Get storage for AST
                cmp     CX,SECSIZE*ASTSECCNT
                jne     ga9             ;Error, not enough storage available
                mov     AST1seg,AX      ;Save for MaxtorClose later
                mov     ES,AX           ;Point ES:[DI] to location
                xor     DI,DI           ;  for AST
                mov     AL,DL           ;Refresh physical drive number
                mov     DS,BiosDataSeg
                call    ObtainAST

        ga9:    pop     DS              ;Restore regs
                pop     ES
                pop     DI
                pop     DX
                pop     CX
                pop     AX
                ret                     ;Return to caller
GetAST          endp

;-------------------------------------------------------------------------;
;                                                                         ;
;       DumpMSGBlock    Format a MSG Block for Debugging Output           ;
;                                                                         ;
;       Input   ES:[3]  Pointer to MSG block                              ;
;                                                                         ;
;       Output  none                                                      ;
;                                                                         ;
;-------------------------------------------------------------------------;
                assume  DS:nothing,ES:MSGblock
DumpMSGBlock    proc    near
                cmp     DebugFlag,0
                jne     dmb1
                ret

        dmb1:   push    AX
                call    PrintString
                db      CR,0
                mov     AX,ES
                call    PrintHexWord
                call    PrintString
                db      ' MSG: sb=',0
                mov     AL,MSGsigbytes
                call    PrintHexByte
                call    PrintString
```

```
                db      ' rq=',0
                mov     AL,MSGrqtype
                call    PrintHexByte
                call    PrintString
                db      ' dn=',0
                mov     AL,MSGdevnum
                call    PrintHexByte
                call    PrintString
                db      ' ct=',0
                mov     AL,byte ptr MSGcharct
                call    PrintHexByte
                mov     AL,byte ptr MSGcharct+1
                call    PrintHexByte
                call    PrintString
                db      ' st=',0
                mov     AL,byte ptr MSGstatus
                call    PrintHexByte
                mov     AL,byte ptr MSGstatus+1
                call    PrintHexByte
                call    PrintString
                db      ' ad=',0
                mov     AL,byte ptr MSGaddress
                call    PrintHexByte
                mov     AL,byte ptr MSGaddress+1
                call    PrintHexByte
                mov     AL,byte ptr MSGaddress+2
                call    PrintHexByte
                mov     AL,byte ptr MSGaddress+3
                call    PrintHexByte
                pop     AX
                ret
DumpMSGBlock    endp

Cseg            ends
                end
```

Formatted by SCRIPT from file 'CHECKSUM PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
/*-------------------------------------------------------------------------;
;                                                                         ;
;       Checksum Computation for Adapter ROMs                             ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1         ;
;       Version:        1.0 05/10/84                                      ;
;       DOS Required:   DOS 2.0 or later                                  ;
;       Classification: IBM Internal Use Only                             ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------*/

char    IBMCopyright[] = " (C) Copyright IBM Corporation, 1984 ";

#include "stdio.h"

main (argc,argv)
int    argc;
char  *argv[];
  {
  char ifname[65];
  int sum,temp;
  FILE *fi,*fo;

  if (argc == 2) {
    makefnam (argv[1],".sys",ifname);
    fi = fopen (ifname,"rb");
    }
  else
    fi = 0;
  if (fi == 0) {
    printf("Usage: checksum [drive:[\\path\\]filename]\n");
    return (1);
    }
  sum = 0;
  while ((temp = getc(fi)) != EOF) {
    sum += (temp & 0xFF);
    }
  fclose (fi);
  sum &= 0xFF;
  sum = (256-sum) & 0xFF;
  printf ("Checksum is %02x\n",sum);
  if (sum == 0) return (0);
  temp = 0;
  printf ("Append to file? [Y or N]: ");
  temp = toupper(getchar());
  if (temp != 'Y' && temp != 'N') {
    printf ("[N assumed]\n");
```

```
    return (0);
    }
  if (temp == 'Y') {
    fo = fopen (ifname,"ab");
    putc (sum,fo);
    if (fclose (fo) != 0) {
      printf ("Error detected while trying to append checksum.\n");
      return (-1);
      }
    else {
      printf ("Done\n");
      return (0);
      }
    }
  }
```

Formatted by SCRIPT from file 'MAXFMT PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
/*-----------------------------------------------------------------------------
;                                                                             ;
;       Formatting Utility for PC-DOS MAXTOR Partitions                       ;
;       Program Property of IBM                                               ;
;                                                                             ;
;       Author:         Mike West, Jr                                         ;
;       Version:        2.0 07/05/84                                          ;
;       DOS Required:   DOS 2.0 or later                                      ;
;       Classification: IBM Internal Use Only                                 ;
;                                                                             ;
;       (C) Copyright IBM Corporation, 1984                                   ;
;       Developed for the Information Technology Center at                    ;
;       Carnegie-Mellon University                                            ;
;                                                                             ;
;-----------------------------------------------------------------------------*/

char    IBMCopyright[] = " (C) Copyright IBM Corporation, 1984 ";

#include "stdio.h"
typedef unsigned char UCHAR;
#define TRUE 1
#define FALSE 0
typedef char BOOLEAN;
#define VOLUMETYPE 11
#define NAMETYPE 8
#define EXTTYPE 3
#define BUFTYPE 0x200
#define BUFFERTYPE 0x100
#define FILETYPE 0x40
#define CHART 0x8000

struct regs { unsigned ax,bx,cx,dx,si,di,ds,es; };
struct segregs { unsigned CSseg,SSseg,DSseg,ESseg; };
struct devinfo {
                UCHAR address;
                unsigned cylinder;
                unsigned startcyl;
                unsigned blockcount;
                };
struct FCBtype {
                UCHAR prefix[NAMETYPE];
                UCHAR filename[NAMETYPE];
                UCHAR fileext[EXTTYPE];
                unsigned dummy1;
                unsigned recordsize;
                UCHAR dummy2[0x11];
                unsigned recno;
                unsigned dummy3;
                };
struct BPBtype {
                UCHAR start1[3];
                UCHAR start2[8];
```

```
                unsigned bytespersec;
                UCHAR spau;
                unsigned reservesec;
                UCHAR noFAT;
                unsigned rootdir;
                unsigned logicalimage;
                UCHAR descriptor;
                unsigned FATsec;
                unsigned secpertrack;
                unsigned headno;
                unsigned hiddensec;
                UCHAR dummy[0x1E2];
                };

unsigned GetSlash()
{
     unsigned flags;
     struct regs registers;

     registers.ax = 0x3700;
     flags = sysint21(&registers,&registers);
     return(registers.dx & 0x00FF);
}

BOOLEAN CheckOption(argv,i,s,v,inc)
UCHAR *argv[];
unsigned i, inc;
BOOLEAN *s,*v;
{
     UCHAR check;
     UCHAR a, b;
     BOOLEAN error = FALSE;

     check = GetSlash();
     if (*(argv[i] + inc) == check) {
          ++inc;
          a = *(argv[i] + inc);
          if (a=='s' || a=='S' || a=='v' || a=='V') {
               while (a=='s' || a=='S' || a=='v' || a=='V' || a==check) {
                    if (a=='s' || a=='S')
                         *s = TRUE;
                    else if (a=='v' || a=='V')
                         *v = TRUE;
                    else if (a != check)
                         error = TRUE;
                    ++inc;
                    a = *(argv[i] + inc);
               }
               if (a!='\0') {
                    error = TRUE;
               }
          }
```

```
            else
                error = TRUE;
        }
        if (error) {
            printf("Unknown option:  %c\n",a);
        }
        return(error);
}

BOOLEAN GetOptions(drive,s,v,argc,argv)
UCHAR *drive;
BOOLEAN *s,*v;
unsigned argc;
UCHAR *argv[];
{
        unsigned i, j;
        BOOLEAN flag,error;

        *s = FALSE;
        *v = FALSE;
        flag = FALSE;
        error = FALSE;
        *drive = *argv[1];
        *drive = toupper(*drive);
        if (isupper(*drive)) {
            error = *(argv[1] + 1) != ':';
            j = 2;
            if (!error) {
                for (i = 1;(i < argc) && (!flag);++i) {
                    flag = CheckOption(argv,i,s,v,j);
                    j = 0;
                }
            }
        }
        else
            error = TRUE;
        if (error)
            printf("Invalid drive specification.\n");
        if (flag)
            error = flag;
        return(error);
}

BOOLEAN CheckDrive(drive,address)
UCHAR drive;
UCHAR *address;
{
        unsigned check,driveaddress;
        UCHAR limitdrive;
        struct regs registers;
        unsigned flags;
        BOOLEAN error;
```

```
     registers.ax = 0x1900;
     flags = sysint21(&registers,&registers);
     driveaddress = registers.ax & 0x00FF;
     registers.dx = driveaddress;
     registers.ax = 0x0E00;
     flags = sysint21(&registers,&registers);
     check = registers.ax & 0x00FF;
     limitdrive = check + 'A' - 1;
     error = drive > limitdrive;
     if (error)
          printf("There is no such drive on this system.\n");
     *address = (driveaddress + 'A');
     return(error);
}

BOOLEAN TranslateDrive(drive,device)
UCHAR drive;
struct devinfo *device;
{
     struct regs registers;
     struct segregs segregisters;
     unsigned flags;
     BOOLEAN error;

     segread(&segregisters);
     registers.bx = drive - 'A' + 1;
     registers.ax = 0x4404;
     registers.cx = 7;
     registers.dx = device;
     registers.ds = segregisters.SSseg;
     flags = sysint21(&registers,&registers);
     error = (flags % 2);
     if (error)
          printf("Use DOS format command for drive %c:.\n",drive);
     return(error);
}

BOOLEAN CheckError(device)
struct devinfo *device;
{
     BOOLEAN error;

     error = device->cylinder == 0;
     if (error) {
          printf("Use the FDISK utility to create a DOS partition ");
          printf("and reboot.\n");
     }
     return(error);
}
```

```
intro(drive)
UCHAR drive;
{
     struct regs registers;
     unsigned check;

     printf("Press any key to begin formatting %c:\n",drive);
     registers.ax = 0x0000;
     check = sysint(0x1e,&registers,&registers);
     printf("\n");
     printf("Formatting ... ");
}

makeFAT(FAT)
UCHAR FAT[];
{
     unsigned x;

     FAT[0] = 0xF8;
     FAT[1] = 0xFF;
     FAT[2] = 0xFF;
     for (x=3; x<=0x7FFF; FAT[x++] = 0);
     return;
}

GetValues(address,h,s)
UCHAR address;
UCHAR *h,*s;
{
     struct regs registers;
     unsigned check;

     registers.ax = 0x0800;
     registers.dx = address;
     check = sysint(0x13,&registers,&registers);
     *h = registers.dx / 0x100;
     *s = (registers.cx % 0x100) & 0x3F;
     return;
}

GetMax(device,h,s,secperall)
struct devinfo *device;
UCHAR *h,*s,*secperall;
{
     unsigned sectors,address;

     address = device->address;
     GetValues(address,h,s);
     sectors = (*s * (*h + 1) * device->cylinder);
     if ((sectors >= 0x40) && (sectors <=0x1FF)) {
          *secperall = 1;
     }
```

```
     if ((sectors >= 0x200) && (sectors <= 0x7FF)) {
          *secperall = 2;
     }
     if ((sectors >= 0x800) && (sectors <= 0x1FFF)) {
          *secperall = 4;
     }
     if ((sectors >= 0x2000) && (sectors <= 0x7FA7)) {
          *secperall = 8;
     }
     if (sectors >=0x7FA8) {
          *secperall = 16;
     }
     return;
}

unsigned FATsize(secperall,device)
UCHAR *secperall;
struct devinfo *device;
{
     unsigned y,secperFAT,power;
     double pow();
     UCHAR h,s;

     GetMax(device,&h,&s,secperall);
     power = *secperall;
     y = (((((*secperall - 1) + device->blockcount) / power) + 1) & 0xFFFE);
     secperFAT = (((y / 2) + y + 0x1FF) / 0x200);
     return(secperFAT);
}

unsigned CylAssign(cyl,sector)
unsigned cyl,sector;
{
     UCHAR ch, cl;

     ch = cyl & 0x00FF;
     cl = ((cyl & 0x0300) / 4) + sector;
     return((ch * 0x0100) + cl);
}

unsigned GetCluster(cyl,head,secperall,device,secperFAT,error)
unsigned cyl;
UCHAR head,secperall;
struct devinfo *device;
unsigned secperFAT;
BOOLEAN *error;
{
     unsigned x, y, z, cluster;
     UCHAR h, s, d;

     GetValues(device->address,&h,&s);
     x = ((cyl - device->startcyl) * (h + 1) * s);
```

```
     y = x + (head * s);
     z = y - 1 - 0x20 - (2 * secperFAT);
     *error = z == 0;
     cluster = z / secperall;
     return(cluster + 2);
}

EvenFAT(cluster,value,FAT)
unsigned cluster,value;
UCHAR FAT[];
{
     unsigned offset;

     offset = (cluster * 3) / 2;
     FAT[offset] = value % 0x100;
     FAT[offset + 1] = (FAT[offset + 1] & 0xF0) | ((value / 0x100) & 0x0F);
     return;
}

OddFAT(cluster,value,FAT)
unsigned cluster,value;
UCHAR FAT[];
{
     unsigned offset;

     offset = (cluster * 3) / 2;
     FAT[offset + 1] =  value / 0x10;
     FAT[offset] = (FAT[offset] & 0x0F) | ((value & 0x0F) * 0x10);
     return;
}

BOOLEAN RecordError(device,cylinder,head,FAT,secperall,secperFAT)
struct devinfo *device;
unsigned cylinder;
UCHAR head;
UCHAR FAT[];
UCHAR secperall;
unsigned secperFAT;
{
     unsigned cluster,offset;
     UCHAR x, y;
     BOOLEAN error = FALSE;

     cluster = GetCluster(cylinder,head,secperall,device,
               secperFAT,&error);
     for (x=0; x<=(0x10/secperall); x++) {
          if (cluster % 2) {
               OddFAT(cluster,0x0FF7,FAT);
               offset = (cluster * 3) / 2;
          }
          else {
               EvenFAT(cluster,0x0FF7,FAT);
```

```
                offset = (cluster * 3) / 2;
            }
            cluster++;
        }
        return(error);
}

BOOLEAN VerifyDisk(device,FAT,secperall,secperFAT)
struct devinfo *device;
UCHAR FAT[];
UCHAR secperall;
unsigned secperFAT;
{
        unsigned c;
        UCHAR ah, h, maxsec, maxhead;
        struct regs registers;
        BOOLEAN error = FALSE;
        unsigned check;

        GetValues(device->address,&maxhead,&maxsec);
        for (c=device->startcyl; c<(device->startcyl+device->cylinder); c++) {
            for (h=0; h<=maxhead; h++) {
                registers.ax = 0x0B00;
                sysint21(&registers,&registers);
                registers.ax = 0x0400 + maxsec;
                registers.dx = (h * 0x0100) + device->address;
                registers.cx = CylAssign(c,1);
                check = sysint(0x13,&registers,&registers);
                ah = registers.ax / 0x0100;
                if (ah) {
                    error = RecordError(device,c,h,FAT.secperall,
                            secperFAT);
                }
            }
        }
        return(error);
}

Flush()
{
        struct regs registers;
        unsigned check;

        registers.ax = 0x0D00;
        check = sysint21(&registers,&registers);
        return;
}


/*
 *  Routine to perform DOS Absolute Disk Read Function (Sysint 0x25)
 *  Maintains stack at proper level and returns flags from int 25.
```

```
 */

unsigned        Sysint25 (block, drive, buffer)
unsigned        block;                  /* Number of block to read */
UCHAR   drive;                          /* Drive ID ('A' thru 'Z') */
UCHAR   *buffer;                        /* Ptr to 512 byte buffer  */
  {
  struct {unsigned AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {unsigned CSseg, SSseg, DSseg, ESseg;} segregs;
  unsigned   flags;
  unsigned   oldIP,oldCS;
  static UCHAR altint25[] = {0xCD, 0x25,/* int 25 */
                             0x07,      /* pop ES */
                             0xCF};     /* iret   */

  segread (&segregs);                   /* --reg H---  --reg L--- */
  regs.AX = 0x3580;                     /* Get IV      IV = 0x80  */
  sysint(0x21,&regs,&regs);
  oldIP = regs.BX;
  oldCS = regs.ES;
                                        /* --reg H---  --reg L--- */
  regs.AX = 0x2580;                     /* Set IV      IV = 0x80  */
  regs.DX = (unsigned) altint25;
  regs.DS = segregs.SSseg;
  sysint(0x21,&regs,&regs);
                                        /* --reg H---  --reg L--- */
  regs.AX = drive - 'A';                /* ----------  drive id   */
  regs.BX = buffer;                     /* offset addr of buffer  */
  regs.CX = 1;                          /* # of blocks to read    */
  regs.DX = block;                      /* starting block number  */
  regs.DS = segregs.SSseg;              /* segment addr of buffer */
  flags = sysint(0x80,&regs,&regs);
                                        /* --reg H---  --reg L--- */
  regs.AX = 0x2580;                     /* Set IV      IV = 0x80  */
  regs.DX = oldIP;
  regs.DS = oldCS;
  sysint(0x21,&regs,&regs);             /* Restore original IV 80 */
  return (flags);                       /* Return flags from DOS  */
  }


/*
 *  Routine to perform DOS Absolute Disk Write Function (Sysint 0x26)
 *  Maintains stack at proper level and returns flags from int 26.
 */

unsigned        Sysint26 (block, drive, buffer)
unsigned        block;                  /* Number of block to write*/
UCHAR   drive;                          /* Drive ID ('A' thru 'Z') */
UCHAR   *buffer;                        /* Ptr to 512 byte buffer  */
  {
  struct {unsigned AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
```

```
  struct {unsigned CSseg, SSseg, DSseg, ESseg;} segregs;
  unsigned   flags;
  unsigned unsigned oldIP,oldCS;
  static UCHAR altint26[] = {0xCD, 0x26,/* int 26 */
                             0x07,      /* pop ES */
                             0xCF};     /* iret   */

  segread (&segregs);                   /* --reg H---  --reg L--- */
  regs.AX = 0x3580;                     /* Get IV      IV = 0x80  */
  sysint(0x21,&regs,&regs);
  oldIP = regs.BX;
  oldCS = regs.ES;
                                        /* --reg H---  --reg L--- */
  regs.AX = 0x2580;                     /* Set IV      IV = 0x80  */
  regs.DX = (unsigned) altint26;
  regs.DS = segregs.SSseg;
  sysint(0x21,&regs,&regs);
                                        /* --reg H---  --reg L--- */
  regs.AX = drive - 'A';                /* ----------  drive id   */
  regs.BX = buffer;                     /* offset addr of buffer  */
  regs.CX = 1;                          /* # of blocks to read    */
  regs.DX = block;                      /* starting block number  */
  regs.DS = segregs.SSseg;              /* segment addr of buffer */
  flags = sysint(0x80,&regs,&regs);
                                        /* --reg H---  --reg L--- */
  regs.AX = 0x2580;                     /* Set IV      IV = 0x80  */
  regs.DX = oldIP;
  regs.DS = oldCS;
  sysint(0x21,&regs,&regs);             /* Restore original IV 80 */
  return (flags);                       /* Return flags from DOS  */
  }

AbsWriteDisk(block,drive,buffer)
unsigned block;
UCHAR drive;
UCHAR *buffer;
{
     Sysint26(block,drive,buffer);
     return;
}

AbsReadDisk(block,drive,buffer)
unsigned block;
UCHAR *drive;
UCHAR *buffer;
{
     unsigned  rc;
     struct regs registers;

     rc = Sysint25(block,*drive,buffer);
     if (rc % 2) {
          printf("\n");
```

```
            printf("Insert DOS diskette in drive A:\n");
            printf("Press any key to continue ...\n");
            registers.ax = 0x0000;
            rc = sysint(0x16,&registers,&registers);
            *drive = 'A';
            AbsReadDisk(block,drive,buffer);
      }
      return;
}

WriteFAT(FAT,secperFAT,drive)
UCHAR FAT[];
unsigned secperFAT;
UCHAR drive;
{
      unsigned w, v, u;
      UCHAR buffer[BUFTYPE];
      struct regs registers;
      UCHAR *oldFAT;

      oldFAT = FAT;
      for (u=0; u<=1; u++) {
            FAT = oldFAT;
            for (v=0; v<secperFAT; v++) {
                  AbsWriteDisk(((v + 1) + (u * secperFAT)),drive,FAT);
                  FAT += 512;
            }
      }
      for (v=0; v<0x200; buffer[v++] = 0);
      for (u=0; u<0x20; u++) {
            AbsWriteDisk((u + 1) + (2 * secperFAT)),drive,buffer);
      }
      Flush();
      return;
}

MakeBPB(address,drive,secperall,secperFAT,device)
UCHAR *address,drive,secperall;
unsigned secperFAT;
struct devinfo *device;
{
      struct BPBtype BPB;

      AbsReadDisk(0,address,&BPB);
      BPB.bytespersec = 0x200;
      BPB.spau = secperall;
      BPB.reservesec = 1;
      BPB.noFAT = 2;
      BPB.rootdir = 0x200;
      BPB.logicalimage = device->blockcount;
      BPB.descriptor = 0xF8;
      BPB.FATsec = secperFAT;
```

```
     BPB.secpertrack = 0x11;
     BPB.headno = 0xF;
     BPB.hiddensec = 1;
     AbsWriteDisk(0,drive,&BPB);
     return;
}

CreateFCB(FCB,attribute,drive,filename,fileext)
struct FCBtype *FCB;
unsigned attribute;
UCHAR drive,filename[],fileext[];
{
     UCHAR *strncpy();
     struct regs registers;
     struct segregs segregisters;
     unsigned x, check;

     segread(&segregisters);
     FCB->prefix[0] = 0xFF;
     for (x=1; x<=5; FCB->prefix[x++] = 0x00);
     FCB->prefix[6] = attribute;
     FCB->prefix[7] = drive - 'A' + 1;
     strncpy(FCB->filename,filename,8);
     strncpy(FCB->fileext,fileext,3);
     FCB->dummy1 = 0;
     FCB->recordsize = 0x100;
     for (x=1; x<0x11; FCB->dummy2[x++] = 0);
     registers.ax = 0x1600;
     registers.ds = segregisters.SSseg;
     registers.dx = FCB;
     check = sysint21(&registers,&registers);
     FCB->dummy3 = 0;
     return;
}

unsigned OpenFile(filename,al,flag)
UCHAR filename[];
unsigned al;
BOOLEAN *flag;
{
     struct regs registers;
     struct segregs segregisters;
     unsigned check,handle,i;

     if (*flag) {
          filename[0] = 'A';
     }
     segread(&segregisters);
     registers.ax = 0x3D00 + al;
     registers.ds = segregisters.SSseg;
     registers.dx = filename;
     check = sysint21(&registers,&registers);
```

```
        handle = registers.ax;
        if (check % 2) {
                printf("\n");
                printf("Insert DOS diskette in drive A:\n");
                printf("Press any key to continue ...\n");
                registers.ax = 0x0000;
                check = sysint(0x1b,&registers,&registers);
                *flag = TRUE;
                handle = OpenFile(filename,al,flag);
        }
        return(handle);
}

CopyFile(handle,FCB)
unsigned handle;
struct FCBtype *FCB;
{
        UCHAR buffer[BUFFERTYPE];
        unsigned bytesread,x,check;
        struct regs registers;
        struct segregs segregisters;

        x = 0;
        segread(&segregisters);
        registers.ax = 0x1A00;
        registers.ds = segregisters.SSseg;
        registers.dx = buffer;
        check = sysint21(&registers,&registers);
        do {
                registers.bx = handle;
                registers.cx = 0x100;
                registers.ax = 0x3F00;
                registers.ds = segregisters.SSseg;
                registers.dx = buffer;
                check = sysint21(&registers,&registers);
                bytesread = registers.ax;
                if (bytesread) {
                        FCB->recno = x++;
                        registers.ax = 0x2800;
                        FCB->recordsize = bytesread;
                        registers.ds = segregisters.SSseg;
                        registers.dx = FCB;
                        registers.cx = 1;
                        check = sysint21(&registers,&registers);
                }
        } while (bytesread);
        return;
}

CloseFile(handle)
unsigned handle;
{
```

```
     struct regs registers;
     unsigned check;

     registers.ax = 0x3E00;
     registers.bx = handle;
     check = sysint21(&registers,&registers);
     return;
}

CloseFCB(FCB)
struct FCBtype *FCB;
{
     struct regs registers;
     struct segregs segregisters;
     unsigned check;

     segread(&segregisters);
     registers.ds = segregisters.SSseg;
     registers.dx = FCB;
     registers.ax = 0x1000;
     check = sysint21(&registers,&registers);
     return;
}

TransferFile(inputfile,outputfile,attribute,drive,address,flag)
UCHAR inputfile[],outputfile[];
unsigned attribute;
UCHAR drive,address;
BOOLEAN *flag;
{
     unsigned handle;
     struct FCBtype FCB;
     UCHAR fileext[EXTTYPE];
     unsigned  i;

     inputfile[0] = address;
     handle = OpenFile(inputfile,0x00,flag);
     fileext[0] = 'C';
     fileext[1] = 'O';
     fileext[2] = 'M';
     CreateFCB(&FCB,attribute,drive,outputfile,fileext);
     CopyFile(handle,&FCB);
     CloseFile(handle);
     CloseFCB(&FCB);
     return;
}

SystemTransfer(drive,address)
UCHAR drive,address;
{
     BOOLEAN flag = FALSE;
```

```
     TransferFile(" :\\IBMBIO.COM","IBMBIO  ",0x07,drive,address,&flag);
     TransferFile(" :\\IBMDOS.COM","IBMDOS  ",0x07,drive,address,&flag);
     TransferFile(" :\\COMMAND.COM","COMMAND ",0x00,drive,address,&flag);
     return;
}

GetLine(s,lim)
UCHAR s[];
unsigned lim;
{
     unsigned c, i;

     for (i=0; i<lim && (c=getchar())!=EOF && c!='\n'; s[i++] = c);
     return(i);
}

AddVLabel(drive)
UCHAR drive;
{
     UCHAR VolumeLabel[VOLUMETYPE];
     unsigned x,length;
     UCHAR name[NAMETYPE];
     UCHAR ext[EXTTYPE];
     struct FCBtype FCB;

     for (x=0; x<VOLUMETYPE; VolumeLabel[x++] = ' ');
     printf("\n");
     printf("Volume label (11 characters, ENTER for none? ");
     length = GetLine(VolumeLabel,VOLUMETYPE);
     printf("\n");
     for (x=0; x<8; name[x++] = ' ');
     for (x=0; x<3; ext[x++] = ' ');
     for (x=0; x<length; x++) {
          if (x < 8) {
               name[x] = VolumeLabel[x];
               name[x] = toupper(name[x]);
          }
          else {
               ext[x - 8] = VolumeLabel[x];
               ext[x - 8] = toupper(ext[x - 8]);
          }
     }
     CreateFCB(&FCB,0x08,drive,name,ext);
     CloseFCB(&FCB);
}

main(argc,argv)
unsigned argc;
UCHAR *argv[];
{
     UCHAR secperall;
     struct devinfo device;
```

```
    UCHAR drive,address;
    BOOLEAN error,s_opt,v_opt;
    UCHAR FAT[CHART];
    unsigned secperFAT;

    error = FALSE;
    error = GetOptions(&drive,&s_opt,&v_opt,argc,argv);
    if (!error) {
        error = CheckDrive(drive,&address);
        if (!error) {
            error = TranslateDrive(drive,&device);
            if(!error) {
                error = CheckError(&device);
                if(!error) {
                    intro(drive);
                    makeFAT(FAT);
                    secperFAT = FATsize(&secperall,&device);
                    error = VerifyDisk(&device,FAT,secperall,
                            secperFAT);
                    if (!error) {
                        WriteFAT(FAT,secperFAT,drive);
                        MakeBPB(&address,drive,secperall,secperFAT,
                                &device);
                        printf("Format complete.\n");
                        if (s_opt) {
                            SystemTransfer(drive,address);
                            printf("System transferred.\n");
                        }
                        if (v_opt) {
                            AddVLabel(drive);
                        }
                        Flush();
                    }
                    else {
                        printf("\n");
                        printf("Unrecoverable error in system area.\n");
                        printf("Cannot format drive %c:.\n",drive);
                    }
                }
            }
        }
    }
}
```

Formatted by SCRIPT from file 'MAXHDR PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
/*-------------------------------------------------------------------------;
;                                                                         ;
;       Constants and Global Declarations for MAXTOR C Utilities          ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1         ;
;       Version:        1.9 05/15/84                                      ;
;       DOS Required:   DOS 2.0 or later                                  ;
;       Classification: IBM Internal Use Only                             ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------*/

#ifdef  MAINPROG
char    IBMCopyright[] = " (C) Copyright IBM Corporation, 1984 ";
#endif

                                /* Universal constants */
#define NULL    0
#define PRIVATE static
#define CARRY   0x001
#define TRUE    1
#define FALSE   0
#define SUCCEED 0
#define FAIL    (-1)
#define VALID   1
#define INVALID 0

                                /* Constants for PC keyboard support */
#define BS      8
#define TAB     9
#define CR      13
#define ESC     27
#define BACKTAB 271
#define F3      317
#define HOME    327
#define UP      328
#define LEFT    331
#define RIGHT   333
#define DOWN    336

                                /* Constants for PC display support */
#define NORMAL          7
#define INTENSE         15
#define REVERSE         0x70
#define BLINK           0x80
#define BLUEBACK        0x10
#define GREENBACK       0x20
```

```
#define REDBACK         0x40
#define REDFORE         0x04
#define BLACKFORE       0x00

                                /* MAXTOR related constants */
#define SECSIZE         512
#define SECSPERHD       17
#define HDSPERCYL       15
#define SIGLEN          16
#define MAXSIG          "Partition Record"
#define SYSTEMCYL       917
#define PARTCNT         8
#define PARTRECCYL      0x95C0
#define PARTRECHD       0
#define PARTRECSEC      1

struct  partentry {
  unsigned int  startcyl, cylcount;
  };
struct  partrec {
  char          signature[16];
  struct        partentry pe[PARTCNT];
  };
```

Formatted by SCRIPT from file 'MAXINIT PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
/*-------------------------------------------------------------------------;
;                                                                         ;
;       MAXTOR Initialization Utility                                     ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:         L K Raper   8-363-6775   CMULKR at PGHVM1         ;
;       Version:        2.3 06/26/84                                      ;
;       DOS Required:   DOS 2.0 or later                                  ;
;       Classification: IBM Internal Use Only                             ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------*/

#define MAINPROG
#include "maxhdr.h"
#include "stdio.h"

#define PROMPT   1
#define NOPROMPT 0

#define MAXPATS 8
PRIVATE char    initpart[PARTCNT];
PRIVATE char    *patbuf[MAXPATS];
PRIVATE int     pattern[MAXPATS] =      {0x8888, 0x9249, 0xE656, 0xAAAA,
                                         0x5555, 0x0F0F, 0x2222, 0x0000};
PRIVATE char    goodbba[SECSPERHD];
PRIVATE char    saflag = FALSE;         /* TRUE when writing System Area  */
PRIVATE char    surfacescan = TRUE;     /* FALSE to skip surface scan     */
PRIVATE char    astupdated = FALSE;     /* TRUE if ast updated            */
PRIVATE int     curdrive;               /* Drive to initialize            */
PRIVATE int     rc = 0;                 /* Return code for DOS            */

#define ASTCYL          917
#define ASTHD           0
#define ASTSEC          2
#define ASTSECCNT       2
#define ASTSIZE         252
#define ASTSIG          "AST"
#define ASTSIGLEN       4
#define ASTOFF          1
#define ASTON           0

struct  astentry {                      /* Alternate Sector Table entry   */
  int           acyl;
  unsigned char ahd,asec;
  };

struct  ast     {                       /* Alternate sector table         */
```

```
  char  astsignature[ASTSIGLEN];        /* Identifies this as AST        */
  int   astcylinder;                    /* Cyl Location of alternate secs */
  int   astbbcount;                     /* Number of alt sectors assigned */
  struct astentry astbb[ASTSIZE];
  }     *astbuf;                        /* Pointer to working copy of AST */


/*
 *   Routine to read the MAXTOR Partition record if it exists
 *
 */

PRIVATE struct  partrec *ReadMAXPartRecord (drive)
int     drive;
  {
  char  *buf;
  int   i;

  buf = (char *) GetIOBuffer (SECSIZE);
  for (i=0; i<SECSIZE; i++)
    buf[i] = 0;
  if ((ReadMAXSectors (buf,drive,SYSTEMCYL,PARTRECHD,PARTRECSEC,1) == FAIL) ||
      (strncmp (buf, MAXSIG, SIGLEN) != 0)) {
    free (buf);
    return (NULL);
    }
  return (buf);
  }


/*
 *   Routine to obtain the working copy of the AST
 *
 */

PRIVATE ObtainAST (drive)
int     drive;
  {
  int   i;
  char  *p;

  astbuf = (struct ast *) GetIOBuffer (SECSIZE*ASTSECCNT);
  p = (char *) astbuf;
  for (i=0; i<(SECSIZE*ASTSECCNT); i++)
    p[i] = 0;
  if ((ReadMAXSectors (astbuf,drive,ASTCYL,ASTHD,ASTSEC,ASTSECCNT) == FAIL) ||
      (strncmp (astbuf->ast.astsignature, ASTSIG, ASTSIGLEN) != 0)) {
    strncpy (astbuf -> ast.astsignature, ASTSIG, ASTSIGLEN);
    astbuf -> ast.astcylinder = ASTCYL;
    astbuf -> ast.astbbcount = 0;
    }
  }
```

```
/*
 *   Routine to store the working copy of the AST on the MAXTOR
 *
 */

PRIVATE StoreAST (drive)
int     drive;
  {
  if (astupdated)
    if (WriteMAXSectors (astbuf,drive,ASTCYL,ASTHD,ASTSEC,ASTSECCNT) == FAIL) {
      printf ("Error storing AST.  Alternate sector assignments lost.\n");
      rc = 2;
      return (FAIL);
      }
  return (SUCCEED);
  }


ClearExistingAST(drive)
int     drive;
  {
  int i;
  if (astbuf->ast.astbbcount != 0) {
    printf ("Alternate sector data will be lost.  ");
    if (PromptForXorY("Continue?",'Y','N') == 'N')
      return;
    astupdated = TRUE;
    }
  astbuf->ast.astbbcount = 0;
  for (i=0;i<ASTSIZE;i++) {
    if (astbuf->ast.astbb[i].asec != 0xFF)
      astbuf->ast.astbb[i].asec = 0;
    astbuf->ast.astbb[i].acyl = 0;
    astbuf->ast.astbb[i].ahd  = 0;
    }
  if (StoreAST(drive) == SUCCEED)
    printf ("AST cleared.\n");
  astupdated = FALSE;
  }


PRIVATE DisplayPartitions (drive,buf)
int     drive;
struct  partrec *buf;
  {
  int   i;
  printf ("\nPartition   Origin   Size\n");
  for (i=0;i<PARTCNT;i++) {
    if (buf -> pe[i].cylcount != 0) {
      printf("     %c        %3d     %3d",i+'a',
```

```
            buf->pe[i].startcyl&0x7FFF,buf->pe[i].cylcount);
      if (initpart[i])
        printf ("       Initialized\n");
      else
        printf ("\n");
      }
    }
  printf("\n");
  }


PRIVATE ControlBreak()
  {
  printf("\n");
  EndInit (99);
  }


PRIVATE EndInit(retcode)
int     retcode;
  {
  ASTRetry(ASTON);
  if (astupdated && (StoreAST(curdrive) == SUCCEED)) {
    printf ("\nAST updated.  Bad block count now at %d.\n",
            astbuf->ast.astbbcount);
    printf("Reboot to read new AST.\n");
    }
  HomeMax (curdrive);
  CursorOn();
  exit(retcode);
  }


PRIVATE int     GetUpperUntilNL()
  {
  int   temp;
  CursorOn();
  do
    temp = toupper(getchar());
    while (temp == EOF);
  CursorOff();
  return (temp);
  }


PRIVATE int     GetUpperCase()
  {
  int   temp;
  CursorOn();
  do
    temp = toupper(getchar());
    while ((temp == EOF) || (temp == '\n'));
```

```
  CursorOff();
  return (temp);
  }


PRIVATE int     PromptForXorY(s,x,y)
char    *s;
char    x,y;
  {
  int   temp = 0;
  while (temp != toupper(x) && temp != toupper(y)) {
    printf("%s [%c or %c]: ",s,x,y);
    temp = GetUpperCase();
    }
  return (temp);
  }


PRIVATE GeneratePattern (buf,pat,len)
int     pat,len;
int     (*buf)[];
  {
  int   i;
  i = len/sizeof(int);
  for (i = 0; i <= len/2; i++)
    (*buf)[i]=pat;
  }


PRIVATE int     ByteToSectorNumber(byte)
int     byte;
  {
  static char   interleave[] = {1,6,11,16,4,9,14,2,7,12,17,5,10,15,3,8,13};
  static int    sectoffset[] = {586,1157,1728,2299,2870,3441,4012,4583,5154,
                                5725,6296,6867,7438,8009,8580,9151,9722};

  int   i;
  for (i=0;i<SECSPERHD;i++) {
    if (sectoffset[i] >= byte)
      return (interleave[i]);
    }
  return (0);
  }


PRIVATE CreatePatternBuffers ()
  {
  int   i;
  for (i=0; i<MAXPATS; i++) {
    patbuf[i] = (char *) GetIOBuffer (SECSIZE);
    GeneratePattern (patbuf[i],pattern[i],SECSIZE);
    }
```

```
  }


PRIVATE AskAboutWholeDrive (drive)
int     drive;
  {
  int   i;
  if (PromptForXorY("Initialize entire drive?",'Y','N') == 'Y') {
    if (PromptForXorY("Perform surface analysis?",'Y','N') == 'Y')
      surfacescan = TRUE;
    else
      surfacescan = FALSE;
    InitializeEntireDrive (drive,NOPROMPT);
    }
  else {
    printf ("1. Use MAXPART utility next to create MAXTOR partition record.\n");
    printf ("2. Run MAXINIT again to initialize individual partitions.\n");
    }
  }


PRIVATE CheckTrackPatterns (drive,cyl,hd)
int     drive,cyl,hd;
  {
  char    *buf;
  int   i,j;
  for (j=0;j<MAXPATS;j++) {
    buf = patbuf[j];
    for (i=1;i<=SECSPERHD;i++)
      if (WriteMAXSectors (buf,drive,cyl,hd,i,1) != SUCCEED)
        return (FAIL);
    if (VerifyMAXSectors (drive,cyl,hd,1,17) != SUCCEED)
      return (FAIL);
    }
  }


PRIVATE int     AddBBtoAST (cyl,hd,sec)
int     cyl,hd,sec;
  {
  int i;
  if (cyl == ASTCYL) {
    if ((hd == ASTHD) && (sec >= ASTSEC) && (sec < ASTSEC+ASTSECCNT)) {
      printf("AST location defective; no alternate sectors can be assigned.\n");
      return (FAIL);
      }
    i = (hd * SECSPERHD) + sec - 1 - ASTSECCNT;
    astbuf -> ast.astbb[i].asec = 0xFF;
    if (astbuf -> ast.astbbcount < i)
      astbuf -> ast.astbbcount = i;
    }
  else {
```

```
    for (i=0;i<astbuf->ast.astbbcount;i++) {
      if ((astbuf->ast.astbb[i].acyl == cyl) &&
          (astbuf->ast.astbb[i].ahd  == hd)  &&
          (astbuf->ast.astbb[i].asec == sec)) {
        printf ("Cylinder %d track %d sector %d already in AST.\n",cyl,hd,sec);
        return (FAIL);
        }
      }
    for (;;) {
      i = astbuf -> ast.astbbcount;
      if (i >= ASTSIZE) {
        printf ("AST Overflow occurred.  No alternate sector assigned.\n");
        return (FAIL);
        }
      if (astbuf ->ast.astbb[i].asec == 0xFF)
        astbuf -> ast.astbbcount++;
      else
        break;
      }
    astbuf -> ast.astbb[i].asec = sec;
    astbuf -> ast.astbb[i].acyl = cyl;
    astbuf -> ast.astbb[i].ahd  = hd;
    astbuf -> ast.astbbcount++;
    }
  astupdated = TRUE;
  return (SUCCEED);
  }


PRIVATE ASTRetry (x)
char    x;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;
  int    i,flags;
  char   onoroff;

  onoroff = x;
  segread (&segregs);
  for (i=3;TRUE;i++) {                  /* --reg H---  --reg L--- */
    regs.AX = 0x4405;                   /* IOCTL(write) by drive  */
    regs.CX = 1;                        /* Number of control bytes*/
    regs.BX = i;                        /* ----------  drive id   */
    regs.DX = &onoroff;
    regs.DS = segregs.SSseg;            /* Auto is in stack segment */
    flags = sysint (0x21,&regs,&regs);
    if ((flags & CARRY) && (regs.AX == 5))
      break;
    }
  }
```

```
PRIVATE int     ForceBadBlock (drive,cyl,hd,sec)
int     drive,cyl,hd,sec;
  {
  int   i,j;
  char  *buf[SECSPERHD];
  char  bba[SECSPERHD];
  if ((AddBBtoAST (cyl,hd,sec)==SUCCEED) &&
     (StoreAST (drive) == SUCCEED)) {
    ASTRetry(ASTOFF);
    for (i=0;i<SECSPERHD;i++) {
      buf[i] = GetIOBuffer (SECSIZE);
      bba[i] = ReadMAXSectors (buf[i],drive,cyl,hd,i+1,1);
      }
    bba[sec-1] = 0xFF;
    FormatMAXTrack (drive,cyl,hd,bba);
    ASTRetry(ASTON);
    for (i=0;i<SECSPERHD;i++) {
      if (bba[i] == 0)
        WriteMAXSectors (buf[i],drive,cyl,hd,i+1,1);
      if (i != sec-1)
        free (buf[i]);
      }
    i = sec-1;
    j = (astbuf -> ast.astbbcount) + ASTSECCNT;
    hd = j / SECSPERHD;
    sec = (j % SECSPERHD) + 1;
    WriteMAXSectors (buf[i],drive,ASTCYL,hd,sec,1);
    free (buf[i]);
    return (SUCCEED);
    }
  return (FAIL);
  }


PRIVATE InitializeSector (drive,cyl,hd,sec)
int     drive,cyl,hd,sec;
  {
  int   i;
  char  *buf;
  for (i=0; i<MAXPATS; i++) {
    buf = patbuf[i];
    if (WriteMAXSectors (buf,drive,cyl,hd,sec,1) != SUCCEED)
      return (FAIL);
    if (VerifyMAXSectors (drive,cyl,hd,sec,1) != SUCCEED)
      return (FAIL);
    }
  return (SUCCEED);
  }


PRIVATE InitializeTrack (drive,cyl,hd)
int     drive,cyl,hd;
```

```
  {
  int   i,r,bbflag;
  char  bba[SECSPERHD];

  bbflag = FALSE;
  if (saflag == FALSE) {
    r = CursorRow();
    PositionCursor (r,0);
    printf("Formatting cylinder %d track %d   ",cyl,hd);
    }
  FormatMAXTrack (drive,cyl,hd,goodbba);
  if (surfacescan) {
    if (CheckTrackPatterns (drive,cyl,hd) != SUCCEED) {
      for (i=1; i<=SECSPERHD; i++)
        if (InitializeSector (drive,cyl,hd,i) != SUCCEED) {
          bba[i-1] = FAIL;
          printf ("   Bad block at sector %d\n",i);
          AddBBtoAST (cyl,hd,i);
          bbflag = TRUE;
          }
        else
          bba[i-1] = SUCCEED;
      if (bbflag)
        FormatMAXTrack (drive,cyl,hd,bba);
      }
    }
  }


PRIVATE InitializeCylinder (drive,cyl)
int     drive,cyl;
  {
  int   i;
  for (i=0; i<HDSPERCYL; i++) {
    InitializeTrack (drive,cyl,i);
    }
  }


PRIVATE InitializeSystemArea (drive)
int     drive;
  {
  saflag = TRUE;
  InitializeCylinder (drive,SYSTEMCYL);
  saflag = FALSE;
  }


PRIVATE InitializePartition (buf,part,drive)
int     part,drive;
struct  partrec *buf;
  {
```

```
  int   i,j;
  printf ("Initializing partition %c on drive %d.\n",part+'a',drive);
  j = buf->pe[part].cylcount;
  for (i=buf->pe[part].startcyl&0x7FFF;j>0;i++,j--) {
    InitializeCylinder (drive,i);
    }
  initpart[part] = TRUE;
  printf("\n");
  }


PRIVATE InitializeEntireDrive (drive,pflag)
int     drive;
char    pflag;
  {
  int   i;
  if (pflag) {
    printf ("All existing data will be lost.  ");
    if (PromptForXorY("Continue?",'Y','N') == 'N')
      return;
    }
  astbuf->ast.astbbcount = 0;
  ClearExistingAST(drive);
  for (i=0; i<SYSTEMCYL; i++)
    InitializeCylinder (drive,i);
  printf ("\n");
  printf ("Initialization for Drive %d complete.\n",drive);
  }


PRIVATE InitializeDrive (drive)
int     drive;
  {
  int   i,temp;
  struct partrec *buf,*buf2;
  if ((buf=ReadMAXPartRecord(drive))==NULL) {
    if (FormatMAXTrack (drive,SYSTEMCYL,0,goodbba) == FAIL)
      abort ("MAXTOR Drive %d does not respond.",drive);
    printf ("Initializing System Area on drive %d ...\n",drive);
    InitializeSystemArea (drive);
    buf = (struct partrec *) GetMAXPartRecord (drive);
    PutMAXPartRecord (drive,buf);
    free (buf);
    printf("System Area initialized.\n");
    ObtainAST (drive);
    AskAboutWholeDrive (drive);
    }
  else {
    printf ("Drive %d: System Area present.\n",drive);
    ObtainAST (drive);
    temp = 0;
    for (i=0;i<PARTCNT;i++)
```

```
    if (buf -> pe[i].cylcount != 0)
      temp++;
  if (temp == 0) {
    printf ("No partitions currently defined.\n");
    AskAboutWholeDrive(drive);
    }
  else {
    while (TRUE) {
      DisplayPartitions (drive,buf);
      printf("     s        [initialize System Area]\n");
      if (astbuf -> ast.astbbcount != 0) {
        printf("     t        [display Alternate Sector Table]\n");
        printf("     u        [clear Alternate Sector Table]\n");
        }
      if (surfacescan == FALSE)
        printf("     x        [activate surface analysis]\n");
      else
        printf("     x        [deactivate surface analysis]\n");
      printf("     y        [declare bad block]\n");
      printf("     z        [initialize entire usable area]\n");
      printf("     q        [quit]\n");
      printf("\n");
      printf ("Enter partition to initialize: ");
      temp = 0;
      while (temp != '\n') {
        temp = GetUpperUntilNL();
        switch (temp) {
          case 'Q': return;

          case 'S': {
                    char c;
                    c = astupdated;
                    buf2 = (struct partrec *) GetMAXPartRecord (drive);
                    if (astbuf->ast.astbbcount != 0) {
                      printf ("Data in alternate sectors will be lost.  ");
                      if (PromptForXorY("Continue?",'Y','N') == 'N')
                        break;
                      astupdated = TRUE;
                      }
                    printf("Initializing System Area on drive %d.\n",drive);
                    InitializeSystemArea (drive);
                    printf ("System Area initialized.\n");
                    PutMAXPartRecord (drive,buf2);
                    printf ("Partition table contents preserved.  ");
                    if (StoreAST(drive) == SUCCEED)
                      printf ("AST preserved.");
                    printf ("\n");
                    astupdated = c;
                    break;
                    }

          case 'T': {
```

```
                int i,j;
                j = astbuf -> ast.astbbcount;
                printf ("Bad blocks:\n");
                for (i=0;i<j;i++) {
                  printf ("     cylinder %3d track %2d sector %2d",
                    astbuf->ast.astbb[i].acyl,
                    astbuf->ast.astbb[i].ahd,
                    astbuf->ast.astbb[i].asec);
                  if ((i+1)%2 == 0)
                    printf ("\n");
                  }
                if ((i%2) != 0)
                  printf ("\n");
                break;
                }

        case 'U': {
                ClearExistingAST(drive);
                break;
                }

        case 'X': surfacescan = !surfacescan;
                if (surfacescan)
                  printf ("Surface analysis activated.\n");
                else
                  printf ("Surface analysis deactivated.\n");
                break;

        case 'Y': {
                int  c,h,s;
                char inp[80];
                CursorOn();
                printf ("\n");
                do {
                  do {
                    DosInput("Enter cylinder number",inp);
                    } while (ValidNumber(inp) == FALSE);
                  sscanf (inp,"%d",&c);
                  }  while (c >= SYSTEMCYL);
                do {
                  do {
                    DosInput("Enter track number",inp);
                    } while (ValidNumber(inp) == FALSE);
                  sscanf (inp,"%d",&h);
                  }  while (h >= HDSPERCYL);
                if (PromptForXorY("By byte offset or sector number?",
                    'B','S') == 'B') {
                  do {
                    do {
                      DosInput("Enter byte offset",inp);
                      } while (ValidNumber(inp) == FALSE);
                    sscanf (inp,"%d",&s);
```

```
                        } while ((s=ByteToSectorNumber(s)) == 0);
                      }
                    else
                      do {
                        do {
                          DosInput("Enter sector number",inp);
                          } while (ValidNumber(inp) == FALSE);
                        sscanf (inp,"%d",&s);
                        }  while (s==0 || s > SECSPERHD);
                    if (ForceBadBlock (drive,c,h,s) == SUCCEED)
                      printf ("Cylinder %d track %d sector %d added to AST\n",
                        c,h,s);
                    else
                      printf ("Bad block assignment failed.\n");
                    CursorOff();
                    break;
                    }

            case 'Z': InitializeEntireDrive (drive,PROMPT);
                    return;

            default:  if (temp != '\n') {
                        i = temp - 'A';
                        if ((i>-1) && (i<PARTCNT) && (buf->pe[i].cylcount != 0))
                          InitializePartition (buf,i,drive);
                        else
                          printf ("Incorrect response: %c - no action taken.\n",
                            temp);
                        }
                    }           /* end switch (temp)              */
                  }             /* end while (temp != '\n')       */
                }               /* end while (TRUE)               */
              }                 /* end else [temp != 0]           */
            }                   /* end else [system area present] */
          }

main (argc,argv)
int     argc;
char    *argv[];
  {
  int drive;
  char deflag = FALSE;
  char temp[80];

  if (argc == 2)
    sscanf (argv[1],"%d",&drive);
  else {
    deflag = TRUE;
    drive = 1;
    }
  if (argc > 2) {
    printf ("Usage: maxinit [drive number]\n");
```

```
    exit (100);
    }
  if ((drive>2) || (drive<1))
    abort ("Argument should be MAXTOR drive number [1 or 2], not %s.", argv[1]);
  if ((ConfirmMAXInstalled() == 2) && (deflag))
    do {
      do {
        DosInput("Which drive? [1 or 2]",temp);
        } while (ValidNumber(temp) == FALSE);
      sscanf (temp,"%d",&drive);
      } while (drive>2 || drive<1);
  CreatePatternBuffers();
  curdrive = drive;
  intrinit (ControlBreak,1000,0x23);
  CursorOff();
  InitializeDrive (drive);
  EndInit (rc);
  }
```

Formatted by SCRIPT from file 'MAXPART PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
/*-------------------------------------------------------------------------;
;                                                                         ;
;       MAXTOR Partition Record Manipulation Utility                      ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:          L K Raper   8-363-6775   CMULKR at PGHVM1        ;
;       Version:         2.1 06/21/84                                     ;
;       DOS Required:    DOS 2.0 or later                                 ;
;       Classification:  IBM Internal Use Only                            ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------*/


#define MAINPROG
#include "maxhdr.h"


/*
 *  Program to manipulate MAXTOR Partition Records
 *
 */

main (argc,argv)
int    argc;
char  *argv[];
  {
  int drive;
  char deflag = FALSE;
  char temp[80];

  if (argc == 2)
    sscanf (argv[1],"%d",&drive);
  else {
    deflag = TRUE;
    drive = 1;
    }
  if (argc > 2) {
    printf ("Usage: maxpart [drive number]\n");
    exit (100);
    }
  if ((drive>2) || (drive<1))
    abort ("Argument should be MAXTOR drive number [1 or 2], not %s.", argv[1]);
  if ((ConfirmMAXInstalled() == 2) && (deflag))
    do {
      do {
        DosInput("Which drive? [1 or 2]",temp);
        } while (ValidNumber(temp) == FALSE);
      sscanf (temp,"%d",&drive);
```

```
      } while (drive>2 || drive<1);
  return (UpdateMAXPart(drive));
  }


PRIVATE int     Valnum(key)
int     key;
  {
  if ((key<'0') || (key>'9')) {
    ShowError("Numeric value required");
    return (INVALID);
    }
  return (VALID);
  }


PRIVATE int     YesNo(key)
int     key;
  {
  if ((key=='Y') || (key=='N'))
    return (VALID);
  ShowError("Only Y for YES or N for NO allowed");
  return (INVALID);
  }


PRIVATE int     StConsis(buf,part)
struct  partrec *buf;
int     part;
  {
  int   scyl;
  scyl = buf->pe[part].startcyl & 0x7FFF;
  if ((scyl>916) ||
      (scyl+(buf->pe[part].cylcount)>917)) {
    ShowError ("Partition extends beyond 917 cylinders");
    return (INVALID);
    }
  if (DosPartition(buf,part))
    return (DosOverlap(buf,part,scyl,buf->pe[part].cylcount));
  else
    return (VALID);
  }


PRIVATE int     CtConsis(buf,part)
struct  partrec *buf;
int     part;
  {
  int   scyl;
  int   dp;
  scyl = buf->pe[part].startcyl & 0x7FFF;
  if ((buf->pe[part].cylcount>917) ||
```

```
      ( (scyl+(buf->pe[part].cylcount)) >917)) {
    ShowError ("Partition extends beyond 917 cylinders");
    return (INVALID);
    }
  dp = DosPartition(buf,part);
  if ((buf->pe[part].cylcount>128) && dp) {
    ShowError ("DOS partitions cannot exceed 128 cylinders");
    return (INVALID);
    }
  if (dp)
    return (DosOverlap(buf,part,scyl,buf->pe[part].cylcount));
  else
    return (VALID);
  }


PRIVATE int     DosConsis(buf,part)
struct  partrec *buf;
int     part;
  {
  int   dp;
  dp = DosPartition(buf,part);
  if ((buf->pe[part].cylcount>128) && dp) {
    ShowError ("DOS partitions cannot exceed 128 cylinders");
    return (INVALID);
    }
  if (dp)
    return (DosOverlap(buf,part,buf->pe[part].startcyl&0x7FFF,
      buf->pe[part].cylcount));
  else
    return (VALID);
  }


PRIVATE int     Updtint(p,valstr)
int     *p;
char    *valstr;
  {
  int   temp;
  sscanf(valstr,"%d",&temp);
  *p = (temp & 0x7FFF) | (*p & 0x8000);
  return 0;
  }


PRIVATE int     Updtflg(p,valstr)
int     *p;
char    *valstr;
  {
  int   temp;
  *p &= 0x7FFF;
  if (strcmp(valstr,"Y")==0)
```

```
    *p |= 0x8000;
  return 0;
  }


PRIVATE int     DosOverlap(buf,part,start,cyllen)
struct  partrec *buf;
int     part,start,cyllen;
  {
  int   i,j,istart,ilen,inv;
  if (cyllen==0)
    return (VALID);
  inv = FALSE;
  for (i=0;i<PARTCNT;i++) {
    if (i != part) {
      if (DosPartition(buf,i)) {
        ilen   = buf->pe[i].cylcount;
        if (ilen != 0) {
          istart = buf->pe[i].startcyl & 0x7FFF;
          if ((start >= istart) && (start <= (istart+ilen-1))) {
            inv = TRUE;
            break;
            }
          j =  start+cyllen-1;
          if ((j >= istart)  &&
              (j <= istart+ilen-1)) {
            inv = TRUE;
            break;
            }
          }
        }
      }
    }
  if (inv) {
    ShowError ("Overlapping DOS partitions not allowed");
    return (INVALID);
    }
  else
    return (VALID);
  }


PRIVATE struct  ftabentry {
  int   fc,flen;
  int   (*ivalfn)(),(*iconfn)(),(*updatef)();
  int   *realfield;
  }     ftab[3] = {33,3,Valnum,StConsis,Updtint,NULL,
                   46,3,Valnum,CtConsis,Updtint,NULL,
                   59,1,YesNo,DosConsis,Updtflg,NULL};

/*
```

```
 *   Routine to update the MAXTOR Partition Record
 *
 */

UpdateMAXPart (drive)
int     drive;
  {
  struct partrec *buf;
  unsigned char tempf[4];
  unsigned char defstrg[4];
  int   i,j,k,l;
  unsigned char GetDispChar();

  if ((buf = GetMAXPartRecord(drive)) == NULL)
    abort ("MAXTOR drive %d does not respond.", drive);
  HomeMAX(drive);

  CursorOff();
  ShowMAXPartRecord(drive,buf);
  ftab[0].realfield = &(buf->pe[0].startcyl);
  ftab[1].realfield = &(buf->pe[0].cylcount);
  ftab[2].realfield = &(buf->pe[0].startcyl);
  j=0; k=0;
  for (;;) {
    for(l=0;l<ftab[k].flen;l++) {
      defstrg[l]=GetDispChar(j+9,ftab[k].fc+l);
      }
    defstrg[ftab[k].flen]=0;
    do {
      i = GetInputField(j+9,ftab[k].fc,ftab[k].flen,tempf,
                        defstrg,ftab[k].ivalfn);
      if (i==ESC) break;
      (*(ftab[k].updatef))(ftab[k].realfield+(j*2),tempf);
      } while ((*(ftab[k].iconfn))(buf,j)==INVALID);
    if ((i==ESC) || (i==F3)) break;
    if ((i==RIGHT) || (i==CR) || (i==TAB)) {
      k++;
      if (k>2) {
        k=0;
        j++;
        }
      }
    else if ((i==LEFT) || (i==BACKTAB)) {
      k--;
      if (k<0) {
        k=2;
        j--;
        }
      }
    else if (i==UP)
      j--;
    else if (i==DOWN)
```

```
      j++;
    else if (i==HOME) {
      j=0; k=0;
      }
    if (j<0)
      j=7;
    else if (j>7)
      j=0;
    }
  if (i==F3) {
    PutMAXPartRecord(drive,buf);
    HomeMAX(drive);
    ShowString("Reboot for updates to take effect",20,23,INTENSE);
    }
  PositionCursor(23,0);
  CursorOn();
  }


/*
 *   Routine to display the MAXTOR Partition Record
 *
 */

ShowMAXPartRecord (drive,buf)
int     drive;
struct  partrec *buf;
  {
  int    i;
  ShowPartHeader (drive,buf);
  for (i=0;i<PARTCNT;i++)
    ShowPartition (buf,i);
  }


/*
 *   Routine to display the MAXTOR Partition Record Header
 *
 */

PRIVATE ShowPartHeader (drive,buf)
int     drive;
struct  partrec *buf;
  {
  char  temp[2];
  sprintf(temp,"%d",drive);
  ClearDisplay();
  ShowDoubleBox(1,4,22,75,NORMAL);
  ShowString("Partition Record for MAXTOR Drive",3,22,NORMAL);
  ShowString(temp,3,56,INTENSE);
  ShowString("Total capacity is",4,24,NORMAL);
  ShowString("917",4,42,INTENSE);
```

```
  ShowString("cylinders",4,46,NORMAL);
  ShowString("Partition     Starting     Cylinder     PC-DOS",6,17,NORMAL);
  ShowString("   Id         Cylinder      Count       Access",7,17,NORMAL);
  ShowString("Press",19,17,NORMAL);
  ShowString("Esc",19,23,INTENSE);
  ShowString("to exit without update,",19,27,NORMAL);
  ShowString("F3",19,51,INTENSE);
  ShowString("to update",19,54,NORMAL);
  }
```

```
/*  *-------------------------------------------------------------------*
    |                                                                   |
    |                  Partition Record for MAXTOR Drive 1              |
    |                    Total capacity is 917 cylinders                |
    |                                                                   |
    |             Partition     Starting     Cylinder     PC-DOS        |
    |                Id         Cylinder      Count       Access        |
    |                                                                   |
    |                 a            0            0          N           |
    |                 b            0            0          N           |
    |                 c            0            0          N           |
    |                 d            0            0          N           |
    |                 e            0            0          N           |
    |                 f            0            0          N           |
    |                 g            0            0          N           |
    |                 h            0            0          N           |
    |                                                                   |
    |                                                                   |
    |             Press Esc to exit without update, F3 to update        |
    |             <-------- Error messages appear here -------->        |
    |                                                                   |
    *-------------------------------------------------------------------* */
```

```
/*
 *   Routine to display the information for a single partition
 *
 */

PRIVATE ShowPartition (buf,part)
struct  partrec *buf;
int     part;
  {
  char  temp[7],partid;
  int   row;
  row=part+9;
  partid = 'a'+part;
  PositionCursor(row,21);
  ShowChar(partid,NORMAL);
  sprintf(temp,"%6d",(buf->pe[part].startcyl & 0x7FFF));
  ShowString(&temp[3],row,33,NORMAL);
```

```
  sprintf(temp,"%6d",buf->pe[part].cylcount);
  ShowString(&temp[3],row,46,NORMAL);
  PositionCursor(row,59);
  if (buf->pe[part].startcyl & 0x8000)
    ShowChar('Y',NORMAL);
  else
    ShowChar('N',NORMAL);
  }
```

Formatted by SCRIPT from file 'MAXSUBS PRTSHELL A1'.
Genuine even without omega

Printed by CMULKR

```
/*-------------------------------------------------------------------------;
;                                                                         ;
;       Subroutine Package for MAXTOR C Utilities                         ;
;       Program Property of IBM                                           ;
;                                                                         ;
;       Author:          L K Raper   8-363-6775   CMULKR at PGHVM1        ;
;       Version:         2.0 05/24/84                                     ;
;       DOS Required:    DOS 2.0 or later                                 ;
;       Classification:  IBM Internal Use Only                            ;
;                                                                         ;
;       (C) Copyright IBM Corporation, 1984                               ;
;       Developed for the Information Technology Center at                ;
;       Carnegie-Mellon University                                        ;
;                                                                         ;
;-------------------------------------------------------------------------*/

#include "maxhdr.h"


/*
 *   Routine to allocate an I/O buffer of requested length [<32K]
 *   guaranteed safe for DMA [ie, does not cross a 64K boundary].
 *
 */

char    *GetIOBuffer (length)
int     length;
  {
  extern char *malloc();
  extern int  free();
  char   *buf,*buf2;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;

  segread (&segregs);
  buf = malloc (length);

  if ( ((((((int)buf)+length-1)>>4) & 0x0FFF) + (segregs.SSseg & 0x0FFF))
       & 0xF000) {
    buf2 = malloc (length);
    free (buf);
    buf = buf2;
    }

  if (buf == NULL)
    abort ("Insufficient memory to allocate %d byte I/O buffer", length);
  return (buf);
  }


/*
 *   Routine to Confirm that the MAXTOR Device Support is Installed
```

```
 *
 */

int     ConfirmMAXInstalled()
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  int    flags;

  regs.AX = 0x1700;                 /* Identify driver ------ */
  flags = sysint (0x13,&regs,&regs);
  if (flags & CARRY)
    abort ("MAXTOR device support not installed.");
  return (regs.AX & 0x00FF);        /* return number of drives */
  }


/*
 *   Routine to Read n MAXTOR Sectors
 *
 */

int     ReadMAXSectors (buf,drive,cyl,hd,sec,n)
char    *buf;
int     drive,cyl,hd,sec,n;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;
  int    i,flags;

  segread (&segregs);
  drive += 0xA0-1;

  i = (cyl & 0x0300)>>2;
  i += (cyl & 0x00FF)<<8;
                                    /* --reg H---  --reg L--- */
  regs.AX = 0x0200+n;               /* Read        n Sectors  */
  regs.CX = i+sec;                  /* on Cyl x    Sector y   */
  regs.DX = 256*hd+drive;           /* at Hd  z    Drive [var]*/
  regs.BX = buf;
  regs.ES = segregs.SSseg;          /* Heap is in stack segment */
  flags = sysint (0x13,&regs,&regs);

  if (flags & CARRY)
    return FAIL;
  else
    return SUCCEED;
  }


/*
 *   Routine to Write n MAXTOR Sectors
 *
```

```
 */

int     WriteMAXSectors (buf,drive,cyl,hd,sec,n)
char    *buf;
int     drive,cyl,hd,sec,n;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;
  int    i,flags;

  segread (&segregs);
  drive += 0xA0-1;

  i = (cyl & 0x0300)>>2;
  i += (cyl & 0x00FF)<<8;
                                  /* --reg H---  --reg L--- */
  regs.AX = 0x0300+n;             /* Write       n Sectors  */
  regs.CX = i+sec;                /* on Cyl x    Sector y   */
  regs.DX = 256*hd+drive;         /* at Hd  z    Drive [var]*/
  regs.BX = buf;
  regs.ES = segregs.SSseg;        /* Heap is in stack segment */
  flags = sysint (0x13,&regs,&regs);

  if (flags & CARRY)
    return FAIL;
  else
    return SUCCEED;
  }


/*
 *   Routine to Verify n MAXTOR Sectors
 *
 */

int     VerifyMAXSectors (drive,cyl,hd,sec,n)
int     drive,cyl,hd,sec,n;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;
  int    i,flags;

  segread (&segregs);
  drive += 0xA0-1;

  i = (cyl & 0x0300)>>2;
  i += (cyl & 0x00FF)<<8;
                                  /* --reg H---  --reg L--- */
  regs.AX = 0x0400+n;             /* Verify      n Sectors  */
  regs.CX = i+sec;                /* on Cyl x    Sector y   */
  regs.DX = 256*hd+drive;         /* at Hd  z    Drive [var]*/
  flags = sysint (0x13,&regs,&regs);
```

```
  if (flags & CARRY)
    return FAIL;
  else
    return SUCCEED;
  }


/*
 *   Routine to Format a MAXTOR track
 *
 */

int     FormatMAXTrack (drive,cyl,hd,bba)
int     drive,cyl,hd;
char    bba[];
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;
  int    i,flags;

  segread (&segregs);
  drive += 0xA0-1;

  i = (cyl & 0x0300)>>2;
  i += (cyl & 0x00FF)<<8;
                                  /* --reg H---  --reg L--- */
  regs.AX = 0x1600;               /* Max Format  ---------- */
  regs.CX = i;                    /* on Cyl x    ---------- */
  regs.DX = 256*hd+drive;         /* at Hd  z    Drive [var]*/
  regs.ES = segregs.SSseg;        /* Heap is in stack segment */
  regs.BX = bba;
  flags = sysint (0x13,&regs,&regs);

  if (flags & CARRY)
    return FAIL;
  else
    return SUCCEED;
  }


/*
 *   Routine to read a MAXTOR Partition Record
 *
 */

struct  partrec *GetMAXPartRecord (drive)
int     drive;
  {
  struct partec *buf;
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;
```

```
  int    i,flags;

  segread (&segregs);
  buf = (struct partrec *) GetIOBuffer (SECSIZE);
  drive += 0xA0-1;
                                    /* --reg H---  --reg L--- */
  regs.AX = 0x0800;                 /* Get parms   ---------- */
  regs.DX = drive;                  /* ----------  drive id   */
  flags = sysint (0x13,&regs,&regs);
  if (flags & CARRY) return NULL;
                                    /* --reg H---  --reg L--- */
  regs.AX = 0x0201;                 /* Read        1 Sector   */
  regs.CX = PARTRECCYL+PARTRECSEC;  /* on Cyl x    Sector y   */
  regs.DX = 256*PARTRECHD+drive;    /* at Hd  z    Drive [var]*/
  regs.BX = buf;
  regs.ES = segregs.SSseg;          /* Heap is in stack segment */
  flags = sysint (0x13,&regs,&regs);

  if (flags & CARRY)
    abort ("Unable to read MAXTOR Partition Record. Status = %02x",
      regs.AX>>8);
  if (strncmp(buf -> signature, MAXSIG, SIGLEN) != 0) {
    strncpy (buf-> signature, MAXSIG, SIGLEN);
    for (i=0;i<PARTCNT;i++) {
      buf->pe[i].startcyl = 0;
      buf->pe[i].cylcount = 0;
      }
    }
  return (buf);
  }


/*
 *   Routine to write the MAXTOR Partition Record
 *
 */

PutMAXPartRecord (drive,buf)
int     drive;
struct  partrec *buf;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  struct {int CSseg, SSseg, DSseg, ESseg;} segregs;
  int    i,ok,flags;

  if (strncmp(buf -> signature, MAXSIG, SIGLEN) != 0)
    abort ("Invalid signature in MAXTOR Partition Record");

  drive += 0xA0-1;

  segread (&segregs);
```

```
  ok = FALSE;
  for (i=0;i<5;i++) {
                                        /* --reg H---  --reg L--- */
    regs.AX = 0x0301;                   /* Write       1 Sector   */
    regs.CX = PARTRECCYL+PARTRECSEC;    /* on Cyl x    Sector y   */
    regs.DX = 256*PARTRECHD+drive;      /* at Hd  z    Drive [var]*/
    regs.BX = buf;
    regs.ES = segregs.SSseg;            /* Heap is in stack segment */
    flags = sysint (0x13,&regs,&regs);

    if (flags & CARRY)
      abort ("Unable to write MAXTOR Partition Record. Status = %02x",
        regs.AX>>8);
                                        /* --reg H---  --reg L--- */
    regs.AX = 0x0401;                   /* Verify      1 Sector   */
    regs.CX = PARTRECCYL+PARTRECSEC;    /* on Cyl x    Sector y   */
    regs.DX = 256*PARTRECHD+drive;      /* at Hd  z    Drive [var]*/
    flags = sysint (0x13,&regs,&regs);

    if (flags & CARRY)
      ;
    else {
      ok = TRUE;
      break;                            /* Quit on 1st successful verify */
      }
    }
  if (ok == FALSE)
    abort ("Verify failed for MAXTOR Partition Record. Status = %02x",
      regs.AX>>8);
  }


/*
 *   Function to determine if a MAXTOR partition is accessible to PC DOS
 *
 */

int     DosPartition(buf,part)
struct  partrec *buf;
int     part;
  {
  if (buf->pe[part].startcyl & 0x8000)
    return (TRUE);
  else
    return (FALSE);
  }


/*
 *   Routine to leave the MAXTOR positioned at cylinder 0
 *
 */
```

```
HomeMAX(drive)
int     drive;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  drive += 0xA0-1;
                                /* --reg H---  --reg L--- */
  regs.AX = 0x0401;             /* Verify      1 Sector   */
  regs.CX = 0x0001;             /* on Cyl 0    Sector 1   */
  regs.DX = drive;              /* at Track 0  Drive [var]*/
  sysint (0x13,&regs,&regs);
  }


int     currow,curcol;          /* Current cursor location */


ClearDisplay()
  {
  if (QueryVideoMode()==7)
    ClearWindow(0,0,24,79,NORMAL);
  else {
    SetVideoMode(3);
    ClearWindow(0,0,24,79,BLUEBACK+REDFORE);
    }
  }


PositionCursor(r,c)
int     r,c;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
                                /* --reg H---  --reg L--- */
  regs.AX = 0x0200;             /* Cursor Pos  ---------- */
  regs.BX = 0x0000;             /* page 0      ---------- */
  regs.DX = (r<<8)+c;           /* row         column     */
  sysint (0x10,&regs,&regs);
  currow = r; curcol = c;
  }


QueryCursorPosition()
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
                                /* --reg H---  --reg L--- */
  regs.AX = 0x0300;             /* Read Cursor ---------- */
  regs.BX = 0x0000;             /* page 0      ---------- */
  sysint (0x10,&regs,&regs);
  currow = regs.DX>>8; curcol = regs.DX & 0x00FF;
  }
```

```
int     CursorRow()
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
                                /* --reg H---  --reg L--- */
  regs.AX = 0x0300;             /* Read Cursor ---------- */
  regs.BX = 0x0000;             /* page 0      ---------- */
  sysint (0x10,&regs,&regs);
  return (regs.DX>>8);
  }


CursorOff()
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
                                /* --reg H---  --reg L--- */
  regs.AX = 0x0100;             /* Cursor Type ---------- */
  regs.CX = 0x2506;             /* means invisible cursor */
  sysint (0x10,&regs,&regs);
  }


CursorOn()
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  if (QueryVideoMode()==7)      /* --reg H---  --reg L--- */
    regs.CX = 0x0C0D;           /*  top line   bottom line*/
  else
    regs.CX = 0x0607;

  regs.AX = 0x0100;             /* Cursor Type ---------- */
  sysint (0x10,&regs,&regs);
  }


int     QueryVideoMode ()
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
                                /* --reg H---  --reg L--- */
  regs.AX = 0x0F00;             /* Video State ---------- */
  sysint (0x10,&regs,&regs);
  return (regs.AX & 0x00FF);
  }


SetVideoMode(mode)
int     mode;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
                                /* --reg H---  --reg L--- */
  regs.AX = mode & 0x00FF;      /* Set Mode      [mode]   */
  sysint (0x10,&regs,&regs);
  }
```

```
ClearWindow (r1,c1,r2,c2,attr)
int     r1,c1,r2,c2,attr;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
                                /* --reg H---  --reg L--- */
  regs.AX = 0x0600;             /* Clear entire window    */
  regs.BX = attr<<8;            /* attr        ---------- */
  regs.CX = (r1<<8)+c1;         /* from  r1    c1         */
  regs.DX = (r2<<8)+c2;         /* to    r2    c2         */
  sysint (0x10,&regs,&regs);
  }


ShowChar (c,attr)
unsigned char   c;
int     attr;
  {
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;

                                /* --reg H---  --reg L--- */
  regs.AX = 0x0900+c;           /* Write Char  Char       */
  regs.BX = 0x0000+attr;        /* page 0      Attr       */
  regs.CX = 0x0001;             /* repeat count for Char  */
  sysint (0x10,&regs,&regs);
  }


ShowString(s,r,c,attr)
char    *s;
int     r,c,attr;
  {
  unsigned char x;

  while (x = *s++) {
    PositionCursor (r,c++);
    ShowChar(x,attr);
    }
  }


/*
 *   Routine to display a box outlined with double lines
 *
 */

ShowDoubleBox (ulr,ulc,lrr,lrc,attr)
int     ulr,ulc,lrr,lrc,attr;
  {
  int   i;
  PositionCursor(ulr,ulc);
```

```
  ShowChar(0xC9,attr);
  PositionCursor(lrr,ulc);
  ShowChar(0xC8,attr);
  for (i=ulc+1;i<lrc;i++) {
    PositionCursor(ulr,i);
    ShowChar(0xCD,attr);
    PositionCursor(lrr,i);
    ShowChar(0xCD,attr);
    }
  PositionCursor(ulr,lrc);
  ShowChar(0xBB,attr);
  PositionCursor(lrr,lrc);
  ShowChar(0xBC,attr);
  for (i=ulr+1;i<lrr;i++) {
    PositionCursor(i,ulc);
    ShowChar(0xBA,attr);
    PositionCursor(i,lrc);
    ShowChar(0xBA,attr);
    }
  }


int     GetKey()
  {
  int temp;
  struct {unsigned int AX,BX,CX,DX,SI,DI,DS,ES;} regs;
                                /* --reg H---  --reg L--- */
  regs.AX = 0;                  /* Get key     ---------- */
  sysint (0x16,&regs,&regs);
  temp = regs.AX & 0x00FF;
  return temp ? temp : (regs.AX>>8)+256;
  }


#define ASCIICHAR(i) ((i>0x1F) && (i<256))


int     GetValidKey(ivalfn)
int     (*ivalfn)();
  {
  int temp;
  for (;;) {
    temp = GetKey();
    if (ASCIICHAR(temp)) {
      temp = toupper(temp);
      if ((*ivalfn)(temp)==VALID)
        break;
      }
    else
      break;
    }
  ClearError();
```

```
  return (temp);
  }


int     GetKeyWithEcho(attr)
int     attr;
  {
  int temp;
  temp = GetKey();
  if (ASCIICHAR(temp)) {
    QueryCursorPosition();
    ShowChar(temp,attr);
    BumpCursor();
    }
  return (temp);
  }


int     DosInput (promptmsg,buf)
char    *promptmsg,*buf;
  {
  char  inpbuf[80];
  int   i;
  inpbuf[0] = 78;
  inpbuf[1] = 0;
  while (inpbuf[1] == 0) {
    printf ("%s: ",promptmsg);
    bdos (10,inpbuf);
    }
  printf("\n");
  for (i=0;i<inpbuf[1];i++)
    buf[i]=inpbuf[i+2];
  buf[inpbuf[1]] = 0;
  return (inpbuf[1]);
  }


BumpCursor()
  {
  if (curcol<79)
    curcol++;
  else {
    curcol=0;
    currow++;
    if (currow>24)
      currow = 0;
    }
  }


unsigned char   GetDispChar(r,c)
int     r,c;
```

```
  {
  int   tempr,tempc;
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  tempr = currow;
  tempc = curcol;
  PositionCursor(r,c);
                              /* --reg H---  --reg L--- */
  regs.AX = 0x0800;           /* Read Char   ---------- */
  regs.BX = 0x0000;           /* page 0      ---------- */
  sysint (0x10,&regs,&regs);
  PositionCursor(tempr,tempc);
  return (regs.AX & 0x00FF);
  }


unsigned char   GetDispAttr(r,c)
int     r,c;
  {
  int   tempr,tempc;
  struct {int AX,BX,CX,DX,SI,DI,DS,ES;}    regs;
  tempr = currow;
  tempc = curcol;
  PositionCursor(r,c);
                              /* --reg H---  --reg L--- */
  regs.AX = 0x0800;           /* Read Char   ---------- */
  regs.BX = 0x0000;           /* page 0      ---------- */
  sysint (0x10,&regs,&regs);
  PositionCursor(tempr,tempc);
  return ((regs.AX & 0xFF00)>>8);
  }


PRIVATE int ErrorFlag = FALSE;


PRIVATE ClearError()
  {
  int   tempr,tempc,i;
  if (!ErrorFlag) return;
  tempr = currow;
  tempc = curcol;
  for (i=6;i<73;i++) {
    PositionCursor(20,i);
    ShowChar(' ',NORMAL);
    }
  PositionCursor(tempr,tempc);
  ErrorFlag = FALSE;
  }


ShowError(s)
char *s[];
```

```
  {
  int   sc;
  sc = 36-(strlen(s)/2);
  ShowString(">>>",20,sc,BLINK+INTENSE);
  ShowString(s,20,sc+4,INTENSE);
  ShowString("<<<",20,sc+5+strlen(s),BLINK+INTENSE);
  ErrorFlag = TRUE;
  }


int     GetInputField(r,c,len,field,defstrg,ivalfn)
int     r,c,len;
unsigned char   field[],defstrg[];
int     (*ivalfn)();
  {
  int   i,j,attr;
  attr = GetDispAttr(r,c);
  for(j=0;j<len;j++) {
    field[j]=GetDispChar(r,c+j);
    }
  field[len]=0;
  ShowString(field,r,c,REVERSE);
  i = GetValidKey(ivalfn);
  if (!ASCIICHAR(i)&&(i!=BS))
    {}
  else {
    int resetflag;
    attr = INTENSE;
    resetflag = TRUE;
    while (ASCIICHAR(i)||(i==BS)) {
      if (resetflag) {
        for (j=0;j<len;j++)
          field[j]=' ';
        resetflag = FALSE;
        }
      if (i==BS) {
        for (j=len-1;j>0;j--)
          field[j]=field[j-1];
        field[0]=' ';
        }
      else {
        for (j=0;j<len-1;j++)
          field[j]=field[j+1];
        field[len-1] = i;
        }
      if (strncmp(field,"    ",len)==0) {
        strncpy(field,defstrg,len);
        resetflag = TRUE;
        }
      ShowString(field,r,c,REVERSE);
      i = GetValidKey(ivalfn);
      }
```

```
    }
  for(j=0;j<len;j++) {
    field[j]=GetDispChar(r,c+j);
    }
  ShowString(field,r,c,attr);
  return (i);
  }


int     ValidNumber (s)
char    *s;
  {
  int   i,j;
  j = strlen(s);
  for (i=0;i<j;i++)
    if (isspace(s[i]) == FALSE)
      break;
  if (i==j)
    return (FALSE);
  for (;i<j;i++)
    if (isdigit(s[i]) == FALSE)
      break;
  if (i==j)
    return (TRUE);
  for (;i<j;i++)
    if (isspace(s[i]) == FALSE)
      return (FALSE);
  return (TRUE);
  }
```