HEWLETT-PACKARD CO. — PORTABLE COMPUTER DIVISION

VGER IMAGE PACKAGE

Wed, Jun 5, 1985, 8:40 AM

				MODEL	STK. NO.	
				IMAGE Plug-in ROM Development Utility		
				BY	DATE	
	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 1 OF ??	
REVISIONS				SUPERSEDES	DWG. NO.	

IMAGE PACKAGE INTRODUCTION

The IMAGE plug-in ROM development utility for the HP Portable Plus is composed of three parts: an installable electronic disk (Edisk) driver, a disk image management program, and an EPROM programmer data transfer utility.

EDISK.SYS	Installable RAMdisk driver
IMAGE.COM	Utility program for manipulating data stored in the RAMdisk drive(s) controlled by EDISK.SYS
BURN.COM	Utility program for sending all or part of a to an EPROM programmer

EDISK.SYS is a user-installable multidrive Edisk driver. Once installed, it provides you with one or more "drives" of electronic disk for either general file storage or development of plug-in ROMs. EDISK.SYS is an integral part of the ROM development package; the ROM image utility IMAGE.COM *will not run* if EDISK.SYS is not installed.

IMAGE.COM is an interactive Edisk image maintenance utility. It provides the tools to make the "disks" controlled by EDISK.SYS look (and, with the use of the Romulator card, act) like plug-in ROMs. It provides you with a means of saving an electronic disk drive "image" in an ordinary MS-DOS file in preparation for transfer into an actual ROM or EPROM, and also offers some capabilities to examine and alter the contents of an Edisk drive.

BURN.COM provides a means by which the Edisk images saved by IMAGE.COM can be transferred to an EPROM programmer or other device. It can be used to transfer either an entire file all at once, or extract and transfer a specific subregion of the file. Several different output formats are supported.

IMAGE.COM PLUG-IN ROM DEVELOPMENT UTILITY

When the IMAGE.COM initially starts running, the very first thing it does is try to obtain information from EDISK.SYS about the number and types of Edisks installed. If EDISK.SYS has not been installed, any attempt to run IMAGE will produce the following (or similar) message:

```
A> image

IMAGE [6/4/85]
PLUG-IN ROM DEVELOPMENT UTILITY

EDISK.SYS is not installed

A> _
```

If you see this message, refer to the section on **Installing EDISK.SYS** before attempting to run IMAGE again. Assuming EDISK.SYS has been properly installed and one or more drives of Edisk have been allocated, you should be able to run IMAGE and see a message similar to this:

```
A> image

IMAGE [6/4/85]
Plug-in ROM Development Utility

Drive structure:        Fullbank
Current drive:          "G"
Available drives:       "G".."H"
Edisk RAM segment:      9000
Edisk RAM I/O address:  00E0

IMAGE [G:] _
```

At this point, IMAGE is waiting for you to issue a command. To see what commands are available, type a question mark and then press RETURN.

```
IMAGE [G:] ?
Select commands from the following:

BASE     CHECKSUM DIR      DUMP     ENTER    EXECCODE FENCE    FILL
HELP     HIDE     INITCODE MARK     OEMNAME  QUIT     ROMNAME  SAVE
SHOW     STATUS

IMAGE [G:] _
```

IMAGE is now waiting for you to issue another command. Note that commands can be either upper- or lowercase, and you need type only enough of a command to make it recognizable. For example, you can issue the **STATUS** command by simply entering **ST**.

				MODEL		STK. NO.	
				IMAGE Plug-in ROM Development Utility			
				BY		DATE	
LTR	P.C. NO.	APPROVED	DATE	APPD.		SHEET NO.	3 OF ??
REVISIONS				SUPERSEDES		DWG. NO.	

Providing Commands In A Command File

If you know ahead of time exactly what commands you want to give to IMAGE, you can provide them in a separate file, one command per line, appearing exactly as you would type them. When you then run IMAGE, specify the name of the command file:

```
A> IMAGE filename
```

IMAGE will take its input from the specified file, showing and executing each command exactly as though you had typed it manually. A command file does not need to end with a **QUIT** command; IMAGE will terminate normally when it reaches the end of the file.

Using IMAGE To Create A ROM

The procedure for creating a plug-in ROM generally consists of the following steps:

1. Use EDISK.SYS to install an Edisk drive of the same structure that you wish to use for the final ROM (i.e., halfbank or fullbank).
2. After formatting the drive, use the usual MS-DOS commands to make it look the way you want the final ROM to appear (creating any subdirectories, adding files, etc). Remember that in a plug-in ROM, the entire directory tree *must* come before any files.
3. Run IMAGE.COM, using the following commands:
   MARK *to mark BPB and other information in the boot sector.*
   ROMNAME *to write a name for the ROM in the boot sector.*
   OEMNAME *to write an OEM name into the boot sector.*
   FENCE *to write the directory fence into the boot sector.*
   CHECKSUM *to write the ROM's checksum into the boot sector.*
   SAVE *to save the final Edisk image in a file.*
   QUIT *to exit to MS-DOS.*
4. Run BURN.COM to transfer the Edisk image file into ROM or EPROM.

Depending on your particular needs, more steps may be necessary; the procedure just described outlines the minimum steps required. For a step-by-step walk through the process of creating a fullbank ROM image you should refer to the **IMAGE Example** section later in this document.

BASE command

Usage: **base** [*new base drive identifier*]

The **BASE** command changes the drive identifier that you use within IMAGE to refer to the first of the one or more Edisk drives set up by EDISK.SYS.

At startup, IMAGE assumes that the Edisk drives controlled by EDISK.SYS are the last drives in the system, and it determines the drive identifier for the first of those drives accordingly. If you install an additional disk driver (such as AMIGO.SYS) *after* EDISK.SYS, however, the base Edisk drive identifier assumed by IMAGE will be incorrect. Although there is actually nothing wrong with using an incorrect base identifier (it's purely symbolic anyway), you can, if you want, change it to the correct base (or any other symbol) using the **BASE** command.

CHECKSUM command

Usage: checksum [/*number of clusters*]

This command computes a checksum for the current drive structure and writes it into the boot sector. The checksum is calculated for the total number of clusters on the drive (as reported by the STATUS command), unless you specify fewer clusters by using the */number of clusters* option. Note that a checksum is not required for a plug-in ROM to function; the checksum is only observed by the system diagnostics software.

On a fullbank or halfbank drive, there are always two clusters per sector; since a sector is 512 bytes, one cluster equals 1K-bytes. Any number-of-clusters value that you specify must be in hexadecimal. Further information on the calculation and location of the checksum can be found in the Technical Reference Manual.

DIR command

Usage: dir

The **DIR** command shows you the contents of the root directory of the current drive. *This is not the same as the MS-DOS DIR command.*

When you type the command, you will see something like this:

```
Root directory contents:
      MASM     .EXE a----- at sector 0004 (paragraph 0080) broken
      LINK     .EXE ---shr at sector 00A2 (paragraph 1440) contiguous
<era> iMAGE    .COM a----- at sector 00F6 (paragraph 1EC0) open
_____ ____________ ______ ______________ ________________ __________
  ^         ^          ^          ^               ^             ^
   | ________|______   |  ________|_________      |  ___________|__________
   | |             |   |  |                 |     |  |                     |
   | | Name of file|   |  | Sector in which |     |  | File clusters are   |
   | | or directory|   |  | file begins     |     |  | contiguous, broken  |
   | |_____________|   |  |_________________|     |  | (noncontiguous), or |
  _|______________   __|_____________________     |  | open (available)    |
|                | |                         |    |  |_____________________|
| File has been  | | File attributes:        |  __|______________________
| deleted <era>  | | a-archive   d-directory | |                         |
| or is a direc- | | v-vol lbl   s-system    | | Paragraph at which file |
| tory <dir>     | | h-hidden    r-read only | | begins (segment number) |
|________________| |_________________________| |_________________________|
```

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 7 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

DUMP command

Usage: **dump** [[/][*segment*:]*offset* [*number of bytes*]]

The **DUMP** command shows you, in both hexadecimal and ascii character formats, the contents of memory starting at a specified address. If you simply type **DUMP** without any parameters, you will be shown the first 256 bytes of the current Edisk drive (i.e., the first half of the boot sector in the case of a fullbank). You can specify a different starting point, relative to the beginning of the drive's memory space, by just entering the offset after the word **DUMP**. You can also specify an address in "segment:offset" format -- it will be added to the drive's segment base to determine the absolute memory address at which the dump begins. If you place a slash ("/") before any address you specify, the drive segment base is not added in and the specified value is taken as the absolute memory address at which to begin the dump; by this means you can examine the contents of any memory in the system.

You can optionally include a second parameter specifying the number of bytes to dump (in hex); the default is 100 (256 decimal). If you specify a value, it will be rounded up to the nearest multiple of 16 bytes.

ENTER command

Usage: **enter** *<offset>* *<new data>* [*<new data>* ...]

ENTER allows you to alter the contents of memory *within the confines of the current Edisk drive*; the offset that you specify is relative to the start of the drive (sector 0, or paragraph 0).

The current type of Edisk structure is not taken into account. If you specify more than one byte of new data, the bytes are written sequentially into *contiguous memory locations* with no regard for the every-other-byte format of a halfbank structure.

New data bytes are generally specified as hexadecimal values, but you can additionally specify:

`'<character>`	... an ascii character byte
`"<character>`	... an ascii character byte (same as ')
-	... skip byte (old value unchanged)

EXECCODE command

Usage: execcode <*filename*>

The **EXECCODE** command is used to *pseudodelete* a file in the root directory of a Fullbank drive. **EXECCODE** functions exactly the same as the **HIDE** command, with the addition of changing the first character of the filename to 0E5h. *This is identical to the way MS-DOS deletes a file, except that the clusters occupied by the file remain allocated to that file and cannot be reclaimed by the system. (If you run CHKDSK, the "deleted" file will appear to contain errors, and you will be told that there are lost clusters on the disk.)*

The command is designed to work only with a Fullbank structure since it's intended purpose is to hide files destined to be treated as ROM-executable code; the **DIR** command can be used to determine the relative paragraph within the drive at which the file starts (so you can find out where to "jump" or "call" when you want to execute the file in ROM). intact,

FENCE command

Usage: **fence** [*last directory sector*]

One constraint of a plug-in ROM is that, if it contains subdirectories, the entire directory tree must be located in sectors preceeding any other files. This means that if you are constructing a plug-in ROM image, you should **mkdir** any and all subdirectories immediately after you format the drive and before you begin adding programs and data files. In order for the final directory structure to be properly mapped into a subdirectory of the ROMdisk (drive B:), a *fence* word in the boot sector (sector 0) must be set up specifying the number of the final sector containing directory information.

The FENCE command, with no parameter specified, scans through the current drive's subdirectory tree, looking for the highest-numbered directory sector and checking whether or not all directory information comes ahead of any file sectors. If everything is okay, the number of the last directory sector is displayed and written into the appropriate place in the boot sector; if there is any problem, the resultant value is defaulted to three and a warning is displayed (sector 3 is the last directory sector in an Edisk drive that contains no subdirectories).

You can optionally force a fence value into the boot sector by specifying a new last directory sector number after the command FENCE. Remember that any numeric parameters must be specified in hexadecimal.

FILL command

Usage: fill [*starting sector* [*ending sector* [*value*]]]

The **FILL** command can be used to fill one or more sectors with a single value. If you specify no parameter, all of the sectors on the current Edisk drive are filled with zeros.

If you specify only the starting sector, all sectors from that one onward are filled with zeros. If you specify both a starting and ending sector, all sectors in between and inclusive are filled with zeros. If you want to fill with something other than zero, specify the new fill value as the third parameter.

All parameters must be specified in hexadecimal.

HELP command

Usage: help

This command displays a list of all IMAGE commands. It is effectively the same as typing a question mark ("?") instead of a command.

When you enter a command, you need type only enough letters to make it distinguishable from other commands. For example, you can issue the **FENCE** command by simply typing **FE** (but not just **F** by itself, since the **FILL** command also begins with that letter).

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 13 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

HIDE command

Usage: **hide** <*filename*>

The **HIDE** command is used to *hide* a file in the root directory of a Fullbank drive. This is done by changing the attributes of the file to *System*, *Hidden*, and *Read Only*.

The command is designed to work only with a Fullbank structure since it's intended purpose is to hide files destined to be treated as ROM-executable code; the **DIR** can be used to determine the relative agraph within the drive at which the file starts (so you can find out where to "jump" or "call" when ou want to execute the file in ROM).

				MODEL		STK. NO.	
				IMAGE Plug-in ROM Development Utility			
				BY		DATE	
LTR	P.C. NO.	APPROVED	DATE	APPD.		SHEET NO.	14 OF ??
REVISIONS				SUPERSEDES		DWG. NO.	

INITCODE command

Usage: **initcode** <*filename*>

This command copies the specified file into the boot sector of the current Edisk drive, then sets a boot sector flag byte to indicate that the ROM contains initialization code (to be downloaded into system RAM and executed at when the system is rebooted).

The specified file can be no larger than 464 (decimal) bytes; if it is, an error message is displayed and nothing is copied.

The current drive can be either a Halfbank or Fullbank structure; for the initialization code to be actually downloaded and run, the final ROM must be physically located in ROM slot 7.

MARK command

Usage: **mark**

This command first zeros out the boot sector of the current Edisk drive, and then mark it with the following information:

* ROM type designator bytes (0B1B2h in the case of a Fullbank structure, 0B3B3h in the case of a halfbank).
* BPB (BIOS Parameter Block) information
* Current time and date

Note that this command clears any previous boot sector data (like ROM name, OEM name, checksum, and so forth).

You should issue this command *after* you have created all subdirectories and files on the drive, and *before* you use the **ROMNAME**, **OEMNAME**, **FENCE**, **INITCODE**, and **CHECKSUM** commands. In order for the drive image to appear as a subdirectory of the ROMdisk (drive B:), the very least that must be present in the boot sector is the ROM type designator (added by the **MARK** command) and a ROM name (added by the **ROMNAME** command).

OEMNAME command

Usage: **oemname** [*OEM name string*]

The boot sector of a plug-in ROM can optionally contain an 8-character string, which can say anything. Presumably this string will be the name of the manufacturer of the ROM or some such thing, and is simply written into the boot sector; its presence is totally optional.

If you specify a string of more than eight character, only the first eight are used. If less than eight character, the string is padded with blanks. Any name that you specify will automatically be forced into uppercase. If you do not specify a string, the current OEM name is read from the boot sector and displayed.

QUIT command

Usage: **quit**

Exits the IMAGE utility, returning you to P.A.M. or MS-DOS.

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 18 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

ROMNAME command

Usage: **romname** [*ROM name string*]

The **ROMNAME** command is used to write a unique 8-character ROM name into the boot sector of the current Edisk drive. A ROM name is required if the drive image is to be mapped into a subdirectory of the ROMdisk (drive B:); the ROM name becomes the drive B: subdirectory name.

If you specify a name longer than 8 characters, only the first 8 are used; if less than 8 characters are specified, the name is padded with blanks. Any name you specify will automatically be forced into uppercase. If you do not specify a ROM name, the current name is read from the boot sector and displayed.

SAVE command

Usage: save [*/number of clusters*] <*filename*>

When you have organized the current Edisk drive into the form you want to put into ROMs, you must first save the Edisk drive's *image* (contents) in another file (preferably on some external disk drive).

The **SAVE** command copies the entire drive (or a truncated version, if you specify the number of clusters to save) out to a specified file. The filename should have no extension; if it does, the extension you entered will be removed. A new extension will be appended to the name according to the type of structure you are saving:

`<filename>.HAF`	... Halfbank
`<filename>.EVN`	... Even half of a Fullbank
`<filename>.ODD`	... Odd half of a Fullbank

Saving a Fullbank structure will always generate two files: one for the even address bytes, one for the odd address bytes. Saving a halfbank structure only generates one file (whether it is in the even or odd half bank is irrelevant). Once saved, the **BURN.COM** utility program can be used to extract appropriate sections from each file for transfer into ROMs or EPROMs.

If you specify a number of clusters to save, it must be in hexadecimal. (Remember also that there are two sectors per cluster; since each sector is 512 bytes long, each cluster contains 1K-bytes).

SHOW command

Usage: show [*filename*]

The **SHOW** command is used to *unhide* a file in the root directory of a Fullbank drive. This is done by clearing the *System*, *Hidden*, and *Read Only* attributes of the file, and setting the *Archive* attribute. This effectively undoes the result of the **HIDE** command.

The command is designed to work only with a Fullbank structure since it's intended purpose is to unhide files destined to be treated as ROM-executable code; the **DIR** can be used to determine the relative paragraph within the drive at which the file starts (so you can find out where to "jump" or "call" when you want to execute the file in ROM).

SHOW, without any parameter, performs the same function as the **DIR** command.

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 21 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

STATUS command

Usage: status

The STATUS command displays various pieces of information about the current Edisk drive.

```
ROM (B:subdir) name:     "ROMFILES"  Drive B: subdirectory name (8 chars)
OEM name:                "HP      "  Optional OEM name (8 chars)
Drive structure:         Fullbank    Fullbank, Even or Odd Halfbank
Num sectors on drive:    0200        Maximum number of sectors (in hex)
Num clusters on drive:   0100        Maximum number of clusters (in hex)
Clusters being used:     007B (0080) Actual number of clusters being used
EPROMs req'd per half:   1/27C256    Estimated number of EPROMs needed
Last directory sector:   0003        Last sector containing directory info
Checksum (even half):    EEAA        Checksum word for even address bytes
Checksum (odd half):     D101        Checksum word for odd address bytes
Current drive:           "G"         Drive identifier for current drive
Available drives:        "G".."H"    Identifiers of available Edisk drives
Current RAM/ROM bank:    0           ROMulator bank in which drive resides
Edisk RAM segment:       9000        Segment in which drive resides
Edisk RAM I/O address:   00E0        ROMulator I/O address for this drive
```

Note that all values are in hexadecimal unless otherwise noted. Also note that the number of EPROMs required per half (even and odd addresses) is calculated on a best-fit least-wasted-space basis using commercially available 27C32, 27C64, 27C128, and 27C256 EPROMs; it generally does not reflect the *fewest* number of EPROMs needed to contain the actual number of clusters being used.

: command

Usage: [*Edisk drive identifier*]:

The colon (":") command is used to specify which Edisk drive you want to be the *current* drive. This is very similar to the the way you would normally switch drives in response to an MS-DOS prompt, except that IMAGE allows you to select only from the drives being controlled by EDISK.SYS.

When you select a new drive, you will be informed of which drive you have switch to, its structure (Halfbank or Fullbank), and which drives are available. If you simply type a colon without preceeding it with a drive identifier, you will be shown which drive you are currently working with (this information also appears in the IMAGE command prompt) as well as which drives are available.

? command

Usage: ?

This command displays a list of all IMAGE commands. It is effectively the same as the HELP command.

When you enter a command, you need type only enough letters to make it distinguishable from other commands. For example, you can issue the **FENCE** command by simply typing **FE** (but not just F by itself, since the **FILL** command also begins with that letter).

IMAGE Example

The following is an example of constructing a ROM image up to the point at which it is ready for transfer to EPROM. The example is for a Fullbank ROM, generated using a Romulator card containing two banks of 256K byte; each. The resultant ROM will contain MASM.EXE, LINK.EXE, and EXE2BIN.EXE.

1. *Assume we're starting with:*
   Drive C: is the only external drive in the system.
   Drive A: contains a CONFIG.SYS file that states:
   ```
   DEVICE = EDISK.SYS
   ```
   Drive A: also contains EDISK.SYS and IMAGE.COM.
   Drive C: contains MASM.EXE, LINK.EXE, and EXE2BIN.EXE

2. *Insure the Romulator card switch is set to the RAM position. (When EDISK.SYS is installed, it looks for the card by writing to it, so the card must be "write enabled".)*

3. *Reboot the computer. This will install EDISK.SYS, which should allocate two 256K-byte fullbank drives, D: and E:, in the Romulator card. You should see a message flash by:*
   ```
   MS-DOS version 2.11
   Copyright 1981,82,83 Microsoft Corp.
   EDISK Emulation Driver [6/4/85]
   Found 2 ROM/RAM banks
   F> Unit 0 256K Fullbank
   F> Unit 1 256K Fullbank
   ```

4. *From P.A.M., get into DOS Commands ...*

5. `A> dir d:`
   Don't be alarmed if the directory contains garbage, since the Edisk has not yet been formatted!

   `A> format d:`
   Format the Edisk drive using the standard MS-DOS Format command.

   `A> chkdsk d:`
   CHKDSK can be used to verify the integrity of the Edisk.

   ```
   A> copy c:masm.exe d:
         1 file(s) copied
   A> copy c:link.exe d:
         1 file(s) copied
   A> copy c:exe2bin.exe d:
         1 file(s) copied
   ```

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 25 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

```
A> image

IMAGE [6/4/85]
Plug-in ROM Development Utility

Drive structure:        Fullbank
Current drive:          "D"
Available drives:       "D".."E"
Edisk RAM segment:      9000
Edisk RAM I/O address:  00E0

IMAGE [D:] dir
Root directory contents:
      MASM    .EXE a----- at sector 0004 (paragraph 0080) contiguous
      LINK    .EXE a----- at sector 00A2 (paragraph 1440) contiguous
      EXE2BIN .EXE a----- at sector 00F6 (paragraph 1EC0) contiguous
```

Here we have used the DIR command to verify what files we've added to the root directory, their attributes, and, in the case of ROM-executable files, their starting positions.

```
IMAGE [D:] mark
```

We've just marked the boot sector with BPB information, ROM-type identification, and a time-and-date stamp.

```
IMAGE [D:] fence
Last directory sector: 0003
```

FENCE scans the directory structure for the maximum sector containing directory information. In this case, we only have a root directory, which ends in sector 3.

```
IMAGE [D:] rom tools
ROM (B:subdir) name:   "TOOLS   "
```

The ROM name is used for the name of the subdirectory into which our plug-in ROM will be mapped on drive B:.

```
IMAGE [D:] oem fred
OEM name:              "FRED    "
```

The optional OEM name is written into the boot sector.

```
IMAGE [D:] checksum
Checksum (even half):  B8B2
Checksum (odd half):   EF57
```
Checksums are calculated for 0200 (hex) sectors worth of even and odd addresses. They are then written into the boot sector. Note that is checksum is calculated for the total number of sectors on the drive. If you eventually save fewer than that total, the checksum we have just calculated will be incorrect!

```
IMAGE [D:] status
ROM (B:subdir) name:   "TOOLS   "
OEM name:              "FRED    "
Drive structure:       Fullbank
Num sectors on drive:  0200
Num clusters on drive: 0100
Clusters being used:   007D (0080)
EPROMs req'd per half: 1/27C256 1/27C128 1/27C64 1/27C32 1/27C16
Last directory sector: 0003
Checksum (even half):  B8B2
Checksum (odd half):   EF57
Current drive:         "D"
Available drives:      "D".."E"
Current RAM/ROM bank:  0
Edisk RAM segment:     9000
Edisk RAM I/O address: 00E0
```
A quick status check shows us that we've only used up 007D (hex) clusters out of an available 0100 (hex) total. IMAGE also provides you with a combination of EPROMs that can be used to contain 007D (hex) clusters (with a minimum of empty EPROM space left over). The "(0080)" shows the number of clusters to save in order to minimize the number of EPROMs needed; in this case, two 27C256's per half rather than one 27C256 and four smaller EPROMs.

```
IMAGE [D:] save c:romimage
"c:romimage.EVN" [OK]
"c:romimage.ODD" [OK]
```
The entire Fullbank drive structure is split into even and odd halves and saved on external drive C:.

```
IMAGE [D:] quit
```
The files C:ROMIMAGE.EVN and C:ROMIMAGE.ODD are now ready for transfer to a set of eight (that's right, eight) 27C256 EPROMs!

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 27 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

6. A> dir c:romimage

```
 Volume in drive C is EXAMPLE
 Directory of  C:\

ROMIMAGE EVN    131072   5-31-85   1:55p
ROMIMAGE ODD    131072   5-31-85   1:56p
```

Since the entire 256K-byte disk structure has been saved, we have ended up with two files of 131,072 bytes each. Note, however, that the status report said only 007D (hex) clusters were actually needed; all of the sectors past that point are empty. This is slightly less than half of the 0100 (hex) total clusters. The status report said we minimize the EPROM count by saving 0080 (hex) clusters instead of all 0100, so let's save the image again, this time keeping only half as much!

```
A> image

IMAGE [5/30/85]
Plug-in ROM Development Utility

Drive structure:        Fullbank
Current drive:          "D"
Available drives:       "D".."E"
Edisk RAM segment:      9000
Edisk RAM I/O address:  00E0

IMAGE [D:] st
ROM (B:subdir) name:    "TOOLS    "
OEM name:               "FRED     "
Drive structure:        Fullbank
Num sectors on drive:   0200
Num clusters on drive:  0100
Clusters being used:    007D (0080)
EPROMs req'd per half:  1/27C256 1/27C128 1/27C64 1/27C32 1/27C16
Last directory sector:  0003
Checksum (even half):   B8B2
Checksum (odd half):    EF57
Current drive:          "D"
Available drives:       "D".."E"
Current RAM/ROM bank:   0
Edisk RAM segment:      9000
Edisk RAM I/O address:  00E0
```

Nothing has changed -- we're right where we left off. Note that *we had to enter only enough of the word "STATUS" to make it recognizable.*

```
IMAGE [D:] checksum /80
Checksum (even half):  67AE
Checksum (odd half):   24F4
```

Here we recompute the checksum for just those clusters we actually plan to save. (NOTE: You should always plan to checksum and save a number of clusters equal to the full capacity of the type of ROM you intend to use. Generally, this would be a number of clusters equal to the next power of two that is greater than or equal to the number of clusters actually used, as shown in parentheses in the status report.)

```
IMAGE [D:] save /80 c:romimage
"c:romimage.EVN" [OK]
"c:romimage.ODD" [OK]
```

The save went a little faster this time, since we asked IMAGE to save only the first half of the drive.

```
IMAGE [D:] q

A> dir c:romimage

 Volume in drive C is EXAMPLE
 Directory of  C:\

ROMIMAGE EVN    65536   5-31-85   2:19p
ROMIMAGE ODD    65536   5-31-85   2:19p
```

New versions of C:ROMIMAGE.EVN and C:ROMIMAGE.ODD are now ready for transfer to a set of four 27C256 EPROMs.

EDISK.SYS INSTALLABLE EDISK DRIVER

The most important part of the IMAGE package is EDISK.SYS. This is an installable Edisk driver whose purpose is to set up and control one or more Edisk drives (in addition to the two built-in Edisk drives, A: and B:, and any external drives that may be in the system). When used in conjunction with one or more ROM simulator ("Romulator") plug-in cards, EDISK.SYS converts the available RAM on any such cards into one or more drives of usable disk space. When used without a Romulator card, EDISK.SYS attempts to acquire enough system RAM to compose one or two drives of usable disk space.

Installing EDISK.SYS

To install EDISK.SYS, you must create a file on drive A: called "CONFIG.SYS" that contains the following line:

```
DEVICE = EDISK.SYS
```

When you reboot the computer, MS-DOS automatically searches for any device drivers specified in CONFIG.SYS and attempts to load them into the system. If EDISK.SYS loads successfully, you should briefly see the following message (or something similar) on your screen:

```
EDISK Emulation Driver [6/4/85]
Found 2 ROM/RAM banks
F> Unit 0 256K Fullbank
F> Unit 1 256K Fullbank
```

If you intend to use EDISK.SYS with a Romulator card, you should set the card's ROM/RAM switch to the "RAM" position before rebooting to install the driver. While initializing, EDISK.SYS compiles a list of all accessible Romulator RAM banks; command-line switches are then scanned to determine the manner in which the accumulated banks are to be allocated into Edisk drives.

Specifying Number And Types Of Drives

By specifying one or more *switches* after the "DEVICE = EDISK.SYS", you can tell the driver exactly what kind of drives you want and how many drives to create.

/H	Allocate *two* halfbank drives of 128K-bytes each
/F	Allocate one fullbank drive of 256K-bytes (default)
/D	Allocate one (nonstandard) doublebank drive of 512K-bytes
/M	Allocate one (nonstandard) megabank drive of 1024K-bytes

Additional switches if system RAM is used instead of Romulator cards:

/1	Allocate 64K of system RAM
/2	Allocate 128K of system RAM (default)
/3	Allocate 192K of system RAM
/4	Allocate 256K of system RAM

Two important things should be noted here. First, if you are not using any Romulator cards in the system, you can use the /1 through /4 switches to set aside system RAM for use as two halfbank drives (by also using the /H switch) or one fullbank drive (by using the /F switch or no switch at all); you cannot allocate more than 256K bytes, nor can you set up a double or megabank drive. Also, since the allocated RAM comes out of system memory, any files that you store on the drive(s) *will be lost whenever you reset the computer.* Second, any halfbank and fullbank drives that are set up in Romulator cards are *directly mappable into subdirectory structures of the drive B: ROMdisk.* Doublebank and megabank drives are not supported by the built-in ROMdisk driver, and are provided by EDISK.SYS only as general purpose Edisk drive alternatives.

Edisk Images In Drive B: Subdirectories

The ability of Romulator-resident halfbank and fullbank drives to appear as subdirectories of drive B: is one of the key features of the IMAGE package. By configuring one or more such drives with EDISK.SYS and "stamping" their boot sectors with unique ROM names (using IMAGE.COM), each drive will appear as [1] a general purpose RAM disk with a unique drive identifier (such as G: or H:), and [2] the contents of a subdirectory of the drive B: ROMdisk. The ROM name written into each Edisk's boot sector is automatically mapped by the built-in ROMdisk driver into a corresponding subdirectory of drive B:. Any changes, for example, to files on Edisk drive G:, which might have the ROM name DRIVEG, would simultaneously appear in the directory B:\DRIVEG.

You should keep in mind that this dynamic mapping will only work with *Romulator-resident Fullbank and Halfbank drives,* and that after adding a ROM name (and additional "existance" information -- see the section on the IMAGE MARK command) to the boot sector, it may be necessary to *reset the computer* to get the ROM disk driver to recognize the drives and map them into its own directory structure.

Example EDISK.SYS Installations

The following CONFIG.SYS lines illustrate just a few of the possible drive configurations that could be set up by EDISK.SYS:

Assume one Romulator card containing two RAM banks of 256K each.
Also assume P.A.M. External Drives is set to 2.

`DEVICE = EDISK.SYS /FF`	*Drives E: and F: will both be set up as 256K-byte fullbanks.*
`DEVICE = EDISK.SYS /HF`	*Drives E: and F: will be allocated as 128K-byte halfbanks. Drive G: will be a 256K-byte fullbank.*
`DEVICE = EDISK.SYS /HH`	*Drives E:, F:, G:, and H: will all be set up as 128K-byte halfbanks.*

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
TR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 31 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

`DEVICE = EDISK.SYS /DFH`	*Drive E: will be a 512K-byte drive, using up all the RAM on the card. The F and H requests will consequently be ignored.*

Assume there are no Romulator cards, and system RAM has been partitioned (in P.A.M. System Configuration) to 224K bytes.

`DEVICE = EDISK.SYS`	*Since no parameters are specified, we by default set up drive E: as one 128K-byte fullbank.*
`DEVICE = EDISK.SYS /HD`	*When using system RAM, only the first switch is used. In this case, we set up drives E: and F: as 64K-byte halfbanks.*
`DEVICE = EDISK.SYS /1 /H`	*The /1 says to reserve only 64K bytes of system RAM instead of the usual 128K. This then splits into drives E: and F:, both 32K-byte halfbanks.*
`DEVICE = EDISK.SYS /4`	*A futile attempt to set aside 256K bytes of system RAM for a single fullbank drive. Since there isn't that much RAM available, driver installation will fail.*
`DEVICe = EDISK.SYS /3 /F`	*Drive E:, a 192K-byte fullbank, will be created, but the system may hang since 224K minus 192K leaves only 32K bytes for the operating system and P.A.M. (You should try to leave at least 80K.)*

BURN.COM EPROM DATA TRANSFER UTILITY

Once you have created a ROM image and saved it in a disk file (by using the IMAGE program), you will probably want to transfer the file into one or more ROMs or EPROMs. This can be done with the EPROM programmer data transfer utility, BURN.COM.

BURN is used in a way similar to the TYPE and MORE commands. When you run it, you specify the name of the file you wish to transfer; unless redirected elsewhere, the output goes to the display. To burn EPROMs, you would generally connect an RS-232 EPROM programmer to the computer's AUX (serial) port and then run BURN with output redirected to AUX. You can, of course, redirect the output to any other device or file in the system. You must, however, specify the name of the file to transfer; you cannot pipe data into BURN.

If you run BURN without any parameters, you will be shown the format of the command and all of the available options.

Simple Transfer Using Intel MDS Format

The simplest form of BURN would transfer an entire file to the display, converting each byte into two-digit ascii hex and formatting the result in accordance with the Intel MDS data transfer format. Let's assume we have a 128K-byte file called HALFBANK.HAF (as might be generated by the IMAGE program's SAVE command), and we want to send the entire file to an EPROM programmer (attached to the AUX serial port) that accepts MDS data:

```
A> BURN HALFBANK.HAF >AUX

BURN UTILITY [6/4/85]

File contains 020000 (hex) bytes
Will transfer 020000 (hex) bytes
Starting at offset: 000000 (hex)
Ending at offset:   01FFFF (hex)
Format: Intel MDS

** Press any key to begin transfer **

_
```

At this point, BURN has opened the file and is ready to start sending data. If you haven't already done so, you should now prepare the EPROM programmer to accept the incoming bytes; when it is ready to receive, you simply press any key on the keyboard to begin the transfer. If you are using one of the company-specific transfer formats (described later), BURN will send out an appropriate End-Of-Data record when the transfer is complete.

This straightforward example shows the simplest and probably the most common use of BURN. By specifying additional needs through the use of *switches*, however, you can apply BURN to more particular situations.

Transfering An Extracted Portion Of A File

It's not always desirable to transfer the entire file when you use BURN. If, for example, you want to copy a 128K-byte ROM image file into a set of eight 16K-byte EPROMs, you would like to be able to split the file into eight pieces, and transfer each piece to the programmer one at a time. BURN lets you do just that by extracting and transfering only a specified block from the file. You tell BURN what part of the file you wish to transfer by using a block specification switch. In our previous example, this would look like:

```
        BURN /blocksize:blocknumber HALFBANK.HAF >AUX
```

The blocksize is the size, in K-bytes, of the region you want to extract from the file. The blocknumber specifies which region to transfer. Both the blocksize and the blocknumber can be any integer value from 1 to 256; if you specify a blocksize, but not a blocknumber, transfer starts at the beginning of the file (block 1). *Note that the first blocknumber is 1, not 0.*

Getting back to our example, let's say we want to transfer HALFBANK.HAF, a 128K-byte file, in four 32K chunks. We need only use the block specification switch and run BURN a total of four times:

`BURN /32:1 HALFBANK.HAF >AUX`	Transfer the first 32K bytes
`BURN /32:2 HALFBANK.HAF >AUX`	Transfer the second 32K bytes
`BURN /32:3 HALFBANK.HAF >AUX`	Transfer the third 32K bytes
`BURN /32:4 HALFBANK.HAF >AUX`	Transfer the fourth 32K bytes

Transfering Data In Other Formats

The default transfer format used by BURN is Intel MDS. This is a fairly common ascii-hex format understood by a number of EPROM programmers. You can, however, force the data to be transfered using any one of several different formats by specifying an appropriate format select switch:

/M	Use Intel MDS ascii hex format (this is the default)
/X	Use Motorola EXORCISER ascii hex format
/R	Use RCA COSMAC ascii hex format
/H	Use raw ascii hex format (raw data in two-digit hex form)
/B	Use raw binary format (raw data exactly as read from file)

The various company-specific formats provide a reasonably diverse selection of transfer styles, while raw ascii hex is a straightforward no-extraneous-information format that can be used in quick-and-dirty situations. The raw binary format is essentially an unprocessed transfer, and provides a means of copying an extracted portion of a file to another file, the display, an EPROM programmer, or some other device (you could use it to "type" just the middle 1K bytes of a text file to the display, for example).

Other Switches

Normally when you run BURN, you are shown some transfer statistics before the actual data transfer takes place, then asked to press any key to begin the transfer. Two special switches permit you to alter that behavior:

/Q	Set QUIET mode. No statistics are displayed, and the file transfer begins immediately.
/S	Show statistics only. The transfer does not take place; you are simply shown what region of the file *would* be transfered.

INTEL MDS FORMAT

DATA RECORD

```
-----
  :       START CHARACTER
-----
  B       BC = Byte Count.
  C       This is the hexadecimal number of data bytes in the record.
-----
  A       AAAA = Address of first data byte in the record.
  A       This is a four-digit hexadecimal number.
  A
  A
-----
  T       TT = Data record type (00).
  T
-----
  H       HH = One data byte in two-digit hexadecimal notation.
  H
  .
  .
  .
- - -
- - -
  .
-----
  C       CC = Checksum. Two's complement of binary summation of
  C       preceding byte count, address, and data bytes in hexadecimal.
-----
 <CR>
 <LF>
-----
```

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 36 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

END-OF-FILE RECORD

```
-----
  :      START CHARACTER
-----
  B      BC = Byte count. This is 00 in an End-Of-File record.
  C
-----
  A      AAAA = Address.
  A
  A
  A
-----
  T      TT = End-Of-File Record Type (01).
  T
-----
  C      CC = Checksum. This is FF in an End-Of-File record.
  C
-----
 <CR>
 <LF>
-----
```

MOTOROLA EXORCISER FORMAT

DATA RECORD

```
-----
  S      START CHARACTERS
  1
-----
  B      BC = Byte Count. The number of data bytes plus 3 (1 for
  C      checksum and 2 for address) in hexadecimal notation.
-----
  A      AAAA = Address of first data byte in the record.
  A      This is a four-digit hexadecimal number.
  A
  A
-----
  H      HH = One data byte in two-digit hexadecimal notation.
  H
  .
  .
  .
- - -
- - -
  .
-----
  C      CC = Checksum. One's complement of binary summation of
  C      preceding byte count, address, and data bytes in hexadecimal.
-----
<CR>
<LF>
-----
```

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 38 OF ??
REVISIONS				SUPERSEDES	DWG. NO.

END-OF-FILE RECORD

```
-----
  S     START CHARACTERS
  9
-----
  B     BC = Byte count.  This is 03 in an End-Of-File record.
  C
-----
  A     AAAA = Address.
  A
  A
  A
-----
  C     CC = Checksum.
  C
-----
 <CR>
 <LF>
-----
```

				MODEL	STK. NO.	
				IMAGE Plug-in ROM Development Utility		
				BY	DATE	
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 39 OF ??	
REVISIONS				SUPERSEDES	DWG. NO.	

RCA COSMAC FORMAT

START-OF-FILE RECORD

```
-----
  !     START CHARACTERS
  M
-----
  A     AAAA = Address of first data byte in hexadecimal.
  A
  A
  A
-----
        One space before first data byte.
-----
```

DATA RECORD

```
-----
  A     AAAA = Address of first data byte in the record.
  A     This is a four-digit hexadecimal number.
  A
  A
-----
  H     HH = One data byte in two-digit hexadecimal notation.
  H
  .
  .
  .
- - -
- - -
  .
-----
  ;     End-Of-Record character (semicolon).
-----
<CR>    Each data record ends with a carriage return.
-----
```

END-OF-FILE RECORD

```
-----
<CR>    Carriage return without preceding semicolon.
-----
```

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 40 OF ??
	REVISIONS			SUPERSEDES	DWG. NO.

RAW HEX FORMAT

DATA RECORD

```
-----
  H       HH = One data byte in two-digit hexadecimal notation.
  H
  .
  .
  .
- - -
- - -
  .
-----
```

END-OF-FILE RECORD

```
-----
<CR>      Carriage return/Linefeed.
<LF>
-----
```

				MODEL	STK. NO.
				IMAGE Plug-in ROM Development Utility	
				BY	DATE
LTR	P.C. NO.	APPROVED	DATE	APPD.	SHEET NO. 41 OF ??
REVISIONS				SUPERSEDES	DWG. NO.