System 88
System Programmer's Guide

PolyMorphic
Systems

460 Ward Drive Santa Barbara California 93111 (805) 967-2351

This manual is PolyMorphic Systems part number 810133. It is to be used in conjunction with the following System Programmer Disks.

For Single User:
- 5" SSSD. 820110
- 5" DSDD. 820268
- 8" SSDD. 820269

For TwinSystem:
- 5" DSDD. 820266
- 8" SSDD. 820267

The information in this manual relates to system software released under Version 6, Exec/94 and BASIC C03, with the following system disk part numbers:

For Single User:

820190	Rev F	5" SSSD
820260	Rev B	5" DSDD
820258	Rev B	8" SSDD

For TwinSystem:

820263	Rev B	5" DSDD
820261	Rev B	8" SSDD

This manual was prepared by R.T. Martin, B.F. Smith, F.E. Anderson and D.K. Moe.

Many thanks to C.A. Thompson for his careful proofreading of this manual.

LIMITED WARRANTY AND LIMIT OF LIABILITY

Interactive Products Corporation (dba PolyMorphic Systems) makes No Warranty, express or implied, concerning the applicability of this program to any specific purpose. It is solely the purchaser's responsibility to determine its suitability for a particular purpose. Interactive Products Corporation accepts no liability for loss or damage resulting from the use of this software beyond refunding the original purchase price.

THIS STATEMENT OF LIMITED LIABILITY IS IN LIEU OF ALL OTHER WARRANTIES OR GUARANTEES, EXPRESS OR IMPLIED, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

System 88 System Programmer's Guide

Table of Contents

Section Ø

Introduction to the System 88 System Programmer's Guide

This is the System Programmer's Guide for the System 88. It provides a detailed description of the features and capabilities of the sixth release of the System 88 software (Exec versions 94 and later).

Purpose of This Manual

The purpose of this manual is to help the systems developer build and tailor products based on the System 88. Using system facilities provided at the machine language level requires a detailed functional description. To make best use of this manual, you should be experienced with assembly language program development and in particular with the 8Ø8Ø microprocessor. We assume that you have knowledge of and experience with general systems organization and data structures.

This manual is NOT for the novice!

This manual was written by the designer of the System 88 software. In preparing this manual, a difficult choice was made between the desire to protect the proprietary nature of the system and the wish to provide the systems programmer with the information needed to make best use of system facilities. The writer of this manual hopes that the proprietary nature of this manual and its contents will be respected; the only effective recourse against promiscuous duplication and release of this material is to stop making it available at all.

The Disk Accompanying This Manual

The disk included with this manual is a complete System Disk and includes a number of programs for use by the systems programmer.

Sections of This Manual

Section Ø — is the introduction to the System Programmer's Guide.

Section 1 — contains the summary of the undocumented system commands.

Section 2 — describes the System 88 file system. It describes in detail the structure and allocation of file and directory space.

Section 3 describes the data areas of the system; the second part of Section 3 describes utilities and primitives available to the assembly language programmer. Of particular interest and importance is the section on overlays.

Section 4 describes utility programs provided for the systems programmer: EMEDIT, SZAP, FUTIL, SCOPY, Auth, the Debugger, SPACE, WAIT, TWID, COMPARE, COMP-DISK, CLEAN, ARISE, DIRCOPY, and RECOVER.

Section 5 discusses the System 88 printer driver and its interface with the system.

Section 6 lists System 88 error messages.

Section 7 gives a listing of a sample system overlay.

Section 8 describes in detail the system boot process.

Section 9 describes the interrupt and input/output port structure of the System 88.

Section 1Ø describes the operation of the Volume Manager.

Section 11 describes how CP/M is implemented on the System 88.

Section 12 lists the system symbol tables for the Single and Twin Systems.

Section 13 lists the assembly language routines for doing disk I/O.

Section 14 lists a sample assembly language program that does disk I/O.

NOTE: Many of the discussions in this manual are, of necessity, closely interrelated. These discussions often refer to items defined in Section 3. If an item is unclear, look through the rest of the manual for more information on it or similar topics.

Acknowledgements

The System 88 operating system was designed and written by R.T. Martin. Processors and utilities, as well as able assistance, were provided by Robin Soto, Larry Deran, Frank Anderson, Lennie Araki, Glenn McComb, Brian Smith, and Don Moe. The System 88 hardware was designed by John Stephenson. The packaging and cabinetry were designed by Ron Sanchez.

Credit is also due to many people at Scientific Data Systems/Xerox, especially Ed Bryan, Richard Hustvedt, John Collins, and Mike Macfarlane. Many of the design philosophies behind the System 88 come from the BPM/BTM - UTS - CP/V lineage of systems innovated at SDS/Xerox.

Section 1

Summary of "Undocumented" Commands

The System 88 recognizes a number of commands that are not documented in the System 88 User's Guide. They are not documented because of their complexity or because they have limited application in general use of the system. The manual discusses these commands in more detail later; this section is a single-source reference to these "undocumented" commands. These commands are interpreted by the Exec.

SetSys Command

The SetSys command requests a directory name from the user, then marks every file in that directory a system file (including deleted files). See the section on functions provided by the Dfn1 overlay. The system must be in the ENABLED mode to use this command from Exec.

DUMP Command

The DUMP command is used to display the contents of memory on the screen and printer. The format of the DUMP command is:

DUMP Addr1 Addr2 Comment

where Addr1 and Addr2 are the beginning and ending addresses of the area of memory to dump, and Comment is a comment string that is placed at the top of the dump. See the section on functions provided by the Dfn3 overlay.

Sniff Command

The Sniff command accepts a disk number as an argument in the same format as the LIST or DIR commands (e.g. Sniff 4). If you give no argument to the Sniff command, the System Residence (SYSRES) device is used. The Sniff command reads each sector of the disk from the last sector in use to the beginning of the disk. If an error occurs while Sniff reads a sector, the error code and the number of the problem sector are displayed on the screen. See the section on functions provided by the Dfn2 overlay.

DISPLAY Command

The DISPLAY command displays on the screen the top memory address in the system (MEMTOP), the wild card path, and the last

error code reported in the system (ERROR).

The remaining undocumented commands are for TwinSystem only.

SET Command

The SET command requests allocation of a device on a permanent basis in the mode requested in the set command. See Section 3 on the Devlock for details on the system cell changes.

SET WAIT Command

The WAIT command instructs the devlock facility to wait for a device rather than return reporting an error 'That device is busy!' This is a very dangerous command as it may lead to deadlock where each user has something the other user needs and neither one is going to give up what it has. Both users WAIT forever and very little is accomplished on the system. This command is invoked as follows:

```
SET WAIT ON
    or
SET WAIT OFF
```

SET YAK Command

The YAK command puts the TwinSystem into a very talkative mode. Each time any device allocation is done, a short message appears on the screen describing what happened. Its main use was in debugging the Twin as it was built. This is useful to get a feel for all that is going on in device allocation. This command is invoked as follows:

```
SET YAK ON
    or
SET YAK OFF
```

'porfavor' Command

The porfavor command requests the other user to perform some action. It is only available in ENABLED mode. If the other user is in Exec waiting for a command, it will do the command specified after porfavor. For example,

```
porfavor list 3
```

If the other user is not at the Exec level waiting for a command, the message 'User is busy!' will be displayed.

Section 2

The System 88 File System

The System 88 file system combines versatility with internal simplicity for reliability and ease of use.

A file system consists of file names and extensions. These names and their extensions, together with certain control data, form file directory entries (FDEs). The FDEs and directory management information form the directory on each disk. The directory is first created by the system INIT command, which writes zeros on the disk (thus performing a simple surface test) and then writes out an empty disk directory. The system LIST, DLIST, and DIR commands display the disk directory to the user. The DELETE, UNDELETE, RENAME, SAVE, DNAME, and PACK commands, and the Gfid service modify the directory and FDEs. In addition, the experienced systems programmer may use the system utility program SZAP to manipulate the disk and its contents (see Section 4, Utility Programs).

Disks

The system treats each disk as a sequential collection of 256 byte sectors of data. Data transfers to and from a disk are made in multiples of one sector blocks by the Dio utility. (See Section 3, System Service Vectors, Dio, page 122).

The first four sectors of each disk (Ø, 1, 2, and 3) contain the main file directory for the disk. The system deals with a generalized disk address; Dio breaks this generalized disk address into the proper track and sector required for the device.

Files

A file is a group of contiguous sectors on a disk, accessible through and defined by a file directory entry (FDE) in a disk directory for the device. A file must be totally contained on a single disk, and files may not overlap or share sectors. The internal format of the file is determined by the programs that read and write the file.

File Directory Entries (FDEs)

The File Directory Entry (FDE) defines a file on the disk. The FDE contains all the information required to locate, access, and delimit the file data on the disk. Since the file name in the FDE is of variable length, the FDE itself is also of variable length. The FDE consists of the following information (in this order within the FDE):

/ Flag byte (8 bits)
∩ File name (variable length)
2 File extension (16 bits)
2 FDA - Starting disk address (16 bits)
2 DNS - File length in sectors (16 bits)
2 LA - File load address (16 bits)
2 SA - File start address (16 bits)

The FDE format (in a slightly modified form) is used by the system Gfid utility for looking up and entering file names into the directory.

FDE Flag Byte

The first byte of the FDE contains three one bit flags (D, S, and N) and five bits for the file name length:

```
+-+-+-+-+-+-+-+-+
:D:S:N:...L.....: -> Length of file name
+-+-+-+-+-+-+-+-+
: : +-------------> New file (20H-bit)
: +---------------> System file (40H-bit)
+-----------------> Deleted file (80H-bit)
```

The 80H bit (D above), if set, indicates that the file has been deleted. If this bit is set in an FDE, that FDE will not be examined in the file lookup procedure and will not be displayed by the system LIST command. FDEs marked deleted are returned to normal status by the UNDELETE command or the ARISE program. The space taken up by deleted files, both in the directory area and on the disk, is reclaimed by the system PACK command.

The 40H bit (S above), if set, denotes a "System" file. System commands such as DELETE, RENAME, TYPE, and PRINT check this bit. A file marked by the System bit may not be deleted, renamed, edited, or displayed by PRINT or TYPE.

The 20H bit (N above) denotes a "new" file. When a file is created or changed, its corresponding FDE is marked with the "new" bit to make it eligible for saving by the system file maintenance processor, BACKUP, which then clears the new bit.

Any combination of the above three bits is allowed.

The last five bits of the flag byte give the length of the file name that follows the flag byte. This restricts the file name to 31 characters or less (a file name must be at least one character long). Note that the file name length DOES NOT include the two character extension.

FDE File Name

The file name follows the FDE flag byte and is the only variable length entry in the FDE. The number of bytes used by the file name is contained in the lower five bits of the FDE flag byte. File names usually consist of seven bit ASCII characters, although programs may generate file names consisting of arbitrary eight bit quantities that cannot be entered from the keyboard. When a file name is displayed on the screen, control characters (ASCII ØØ to 1FH) appear as Greek characters.

FDE Extension

The FDE file extension is a sixteen bit (two character) field that follows the file name. The extension identifies the type of file. The bytes appear in the extension in the same order in which they would be typed, rather than the "standard 8Ø8Ø" byte reversed form. For example, the extension "GO" would appear in memory in the FDE as the character "G" followed by "O". Any sixteen bit value may be used as an extension. A number of extensions are predefined and recognized by the system (as the system expands, this list may also expand):

Extension	Use
DX	Sub-Directory
GO	Runnable machine code file
OV	System overlay
BS	BASIC source program
DT	BASIC data file
TX	Text (e.g., assembly language source file)
SY	Symbol table file
RL	Relocatable machine code file
ED	Editor key definition library
FM	Form File
FV	Form Values File
FW	Form Work File
ZO	Onboard code for DSDD Controllers
DD	Device Driver for Hard Disk

FDE FDA - First Disk Address

FDA is a sixteen bit field in the FDE containing the starting disk address for the file.

FDE DNS - File Size in Sectors

DNS is a sixteen bit field in the FDE that contains the number of sectors in the file.

FDE LA - File Load Address

For runnable machine code files (extension .GO or .OV), LA contains the sixteen bit load address for the program. When the file is loaded into memory by the system Runr service, it is read into memory starting at the address contained in LA. For non-runnable files, the LA field in the FDE contains zero. Runr will not load or execute any machine code file with a load address of zero, since there is read only memory at location ØØ in the System 88. For relocatable files LA contains the offset to the relocation bit map.

FDE SA - File Start Address

For runnable machine code files (extension .GO or .OV), SA gives the sixteen bit starting execution address. If the FDA LA field is zero, indicating a non-runnable file, this field may be used for other purposes.

The Disk Directories

The disk directory is the collection of FDEs and control data used to allocate and retrieve files. The main directory appears in sectors Ø, 1, 2, and 3 of each initialized disk. The system INIT command is used to set up the initial directory structure on a disk. Since the directory is a fixed 1Ø24 bytes in length, the number of FDEs it may contain is limited and depends on the length of the file names in the individual FDEs. The directory consists of the following fixed fields (beginning with the first byte of the directory), followed by a list of file directory entries (FDEs):

Displacement (in bytes)	Name	Description
Ø	Dck	8 bit directory checksum
1	Dname	8 byte disk name (main directory only)
9	Nf	Number of files in directory
ØBH	Nfa	Next free directory address
ØDH	Nda	Next free disk address (main dir. only)
ØFH	...	Start of FDE list

In addition to the main directory it is possible to have sub-directories on a disk. Sub-directories look exactly like the main directory except that the disk name and Nda are zeros. A sub-directory is a file on the disk just like any other file except that it is a list of other files on the disk. A sub-directory has an FDE in the directory of which it is a part listing its name with a '.DX' extension. The SA and LA are always 0101 and the file is 4 sectors. Sub-directories are created automatically by Gfid when a new file is created specifying a non-existent sub-directory. For more details see the description of Gfid in section 3.

Since the directory resides in memory in the SBUF1 area, the offsets given above are in hexadecimal from SBUF1.

Filename length versus number of files

The directory on each disk is 1024 bytes. The directory header takes up the first fifteen bytes (decimal) of the directory, leaving 1009 bytes for FDEs. Each FDE has eleven bytes of control data and the varying length name. The table below shows the relation between file name length and the maximum number of files with that name length that fit in the directory.

Table 2.1 Filename length vs. Number of files

Name Length	Maximum number of files
1	84
5	63
10	48
15	38
20	32
25	28
31	24

Dck - Directory checksum

Byte 0 of the directory contains an eight bit checksum computed by the Ckdr service. This checksum is the eight bit sum of the remaining 1023 bytes of the directory, and provides more security in handling the disk directory. When a directory is read into memory, the Ckdr routine is called to calculate the checksum, which is compared to byte 00 of the directory. If the checksums do not match, the directory is considered destroyed, and a 03FFH error results. Whenever the system updates the directory in memory, it also updates the directory checksum.

Dname - Disk Name

The disk name is in an eight character field following the disk directory checksum. The INIT and DNAME commands store the disk name in the directory, and the LIST, DLIST, and DIR commands display the disk name at the top of the directory listing. Dname is empty in sub-directories since the filename of the directory is used as its name.

Nf - Number of files in the directory

Nf is a sixteen bit field containing the total number of files in the directory. This count includes deleted and undeleted files. It is used as a secondary sanity check of the directory structure and is displayed by the system LIST, DLIST, and DIR commands in directory listings.

Nfa - Next FDE Address

Nfa is a sixteen bit pointer to the first free byte after the FDE chain in the directory. Note that this pointer assumes that the directory is residing in SBUF1. When the disk is initialized, Nfa is set to ~~SBUF1+0FH~~. When a file is entered into the directory, it is entered at the address pointed to by Nfa, and then Nfa is updated to point past the newly entered FDE. Nfa is also used to check the space remaining in the directory; it may not exceed SBUF1+1023.

Just SBUF1+0
2800H

Nda - Next Disk Address

Nda contains the sixteen bit disk address of the first free sector on the disk. Nda exists only in the main directory of a disk. Since files are allocated sequentially, Nda is also the number of sectors in use on the disk. When the disk is initialized, Nda is set to 4; this points to the location directly after the directory on the disk. Thus the LIST command on an empty disk displays: "4 sectors in use." When a new file is entered by Gfid in a sub-directory, Nda is updated in the main directory to reflect that additional space in use on the disk.

The Initialized Disk

The user must initialize disks prior to their use. The initialization process fills the disk with zeroes, which performs a simple surface check. Then INIT writes the initial directory to sectors 0, 1, 2, and 3 of the disk. The user specifies the name of the disk in the INIT process. Nf, the number of files on the initialized disk, is set to sixteen bits of 00. Nfa, the next FDE address, is set to SBUF1+0FH. Nda is set to 4, the first free sector on the disk. The remainder of the directory area is set to zero, the checksum computed by calling Ckdr and

stored in Dck, and the directory written to the disk.

Allocating File and Directory Space

The system allocates space on the disk sequentially for files and FDEs. Nda always points to the first sector past the used area of the disk; Nfa always points past the end of the last FDE in the directory. When a file is written to a disk, the data is written starting at the disk address contained in Nda in the main directory. When the FDE is entered into a directory, it is stored at Nfa, and Nfa is updated to point past the new entry. Nda is updated in the main directory by adding the size of the file just entered. (This information is found in the DNS field of the FDE.) This means that space in the directory and the disk is allocated sequentially and contiguously.

Files may not overlap, and the order of FDEs in the directory corresponds to the order of the files on the disk. When files are deleted, the corresponding FDE is marked deleted, but the space in the directory (and the data area of the disk) is not reclaimed until the PACK command is used.

Updating the Disk Directory

The Gfid system service has been provided to update the disk directory. We STRONGLY encourage you to make use of this service and NOT to write programs that update the directory unless absolutely necessary. An improperly updated directory can cause an immediate catastrophe, or the disaster may be postponed until the disk is PACKed or new files are entered on it.

Updating the disk directory in memory (in the SBUF1 area) involves the system cells NFCK, NFDIR, and PATH, described in Section 3, and the system routine Ckdr, described in Section 3. Ckdr computes the checksum of the directory in the SBUF1 area. NFCK is a copy of that checksum. NFDIR is the drive number of the directory currently in SBUF1. The 80H bit of NFDIR if set indicates that the Directory is a subdirectory and the path name of the subdirectory is at PATH. If you MUST update the directory without using the Gfid service, you may use the following procedure:

1) Disable interrupts and compare the drive number desired with the contents of NFDIR and PATH. If the proper directory is in memory, go to step 3.

2) Force the directory into SBUF1 by calling the Gfid Look service to look up a file that does not exist, such as the file with the single byte name 00H. If Look returns any error code other than 0300H, the directory is unreadable.

3) With the interrupts disabled, call Ckdr to compute the directory checksum. This returned checksum must match the contents of NFCK and of byte ØØ of the directory. If it does not match, load Ø3FFH into DE and jump to the system Error routine, since the directory is destroyed.

4) Update the directory with the interrupts disabled, and do it carefully.

5) Call Ckdr to recompute the directory checksum. Store the checksum in NFCK and in byte ØØ of the directory (SBUF1).

6) Call Dio to write four sectors to disk address ØØ (memory address SBUF1) to the device number in NFDIR. In the TwinSystem you may call the Gfid Updir service.

7) If any errors are returned by Dio, store ØFFH into NFDIR and NFCK to prevent the damaged directory from being used, and jump to Error to process the error.

Get File Identifier (Gfid)

A Brief History of Look.

Look was originally (and still is) a system service to look up a file using a file descriptor block or lookup block. Look returns with DE pointing at the FDE in the directory. Gfid used Look to look up files after it had parsed the input text into a file descriptor block. With the advent of the sub-directory, Look was not designed to handle the extended directory file descriptor block. Since Look was very fast and in ROM, it was retained as it existed to do special very speedy lookups of things in base directories using the original file descriptor block. Gfid, however, still needed something with which to look up files that were listed in sub-directories instead of in the main directory. Thus was born 'look'. Now there was a function in Gfid that would do the same thing Look used to do but would do it in sub-directories too! Much too simple, so we called it 'look' so it could forever be confused with Look. In the following discussion references to 'look' mean the one in Gfid unless specifically stated that it is Look in the ROMS. The 'look' in Gfid is called internally by both the Get-file-identifier and the Enter/Replace functions. 'look' is also available as a function of Gfid to be called with an extended directory file descriptor block just as the Look in the ROMS does with the original file descriptor block.

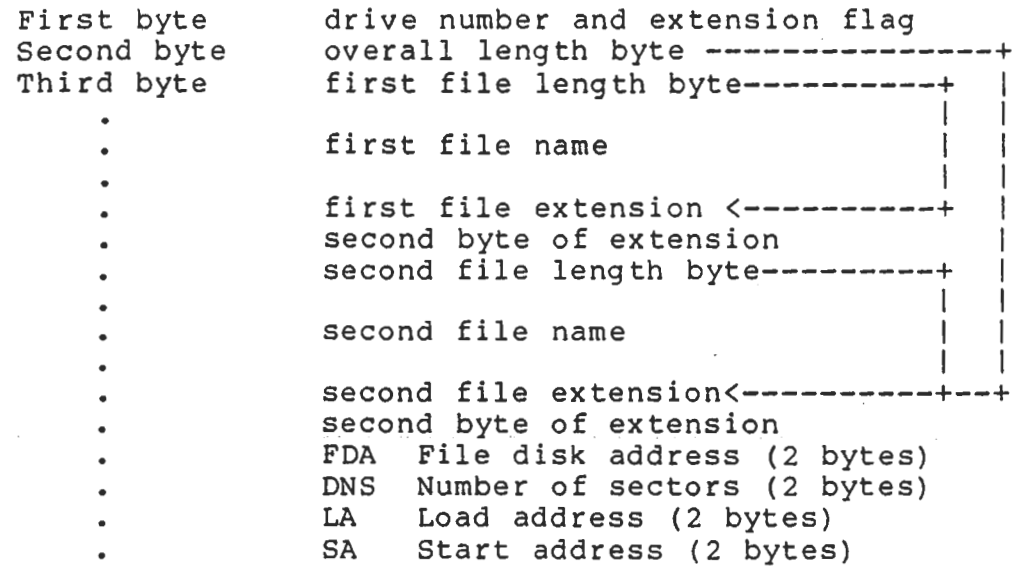
NOTE: There is a file on the disk included with this manual called GFID-DEMO.GO which can be used to observe the effects of various options in using Gfid. GFID-DEMO will be useful in understanding the following discussion.

Original File Descriptor Block

The original file descriptor block built by Gfid consists of one byte containing the specified drive number and an extension presence tag followed by a directory FDE.

Extended Directory File Descriptor Block

The extended directory file descriptor block mentioned above looks like this:



Here are a few examples of this structure. Use GFID-DEMO with the Parse option or use the 'look' option with a non-exisent file to create your own examples.

Text to parse	Resulting lookup block
<2<Gfid.OV	Ø2 Ø5 Ø4 GfidOV
<9<TRIX<UTILS<SCOPY.GO	Ø9 15 Ø4 TRIXDX Ø5 UTILSDX Ø5 SCOPYGO
Dfn1.OV	Ø1 Ø5 Ø4 Dfn1OV (SYSRES is drive 1)
<4<I<U<B<F<DEEP.GO	Ø4 11 Ø1 IDX Ø1 UDX Ø1 FDX Ø4 DEEPGO

As you may have noticed, if the above blocks are viewed as original file descriptor blocks, the length byte once masked to length only (remove NEW, SYS, and DELETED bits) will point at the final extension just as the old type did.

Gfid, Gover, and Ovrto

In the single user system Gfid is an overlay called Gfid.OV. On the TwinSystem, Gfid is a resident system service. For compatibility, the gfid MACRO is provided in SYSTEM.SY (See the section on System Macros). There is a Gfid overlay on the TwinSystem disk that is available so that an overlay call will still work. GFID-DEMO uses the overlay so that even Twin users can see the effect of the overlay mechanism on the variables set by Gfid.

If Gfid is called using the Ovrto function, when Gfid returns the overlay mechanism will cause the previously resident overlay to be loaded back in. When that happens a base directory will be loaded to look up the overlay name and much of what Gfid set in the way of system cells may be changed. DirAddr is guaranteed to be correct but the information in NFDIR, PATH, and SBUF1 will be for the lookup of the overlay, not for the lookup that Gfid did. In the Twin, the information provided by Gfid is preserved if Gfid is called using the macro provided in the SYSTEM.SY file or just using:

```
CALL    Gfid
```

In the single user system the information can be preserved by using the Gover service instead of the Ovrto service. Gover will generally be faster anyway. Of course, if you are in an overlay you must use Ovrto. The effects of the difference in these calls can be seen by using alternately the Gover and Ovrto functions in GFID-DEMO.

Gfid Functions

The Gfid (Get-file-identifier) service provides the assembly language user with the following functions selected by the lower four bits of A:

A and ØFH	Function
Ø	Get file identifier
1	Enter/Replace FDE
2	Look up a file
3	Update directory on disk (Twin only!)

The function performed depends upon the parameter byte passed to Gfid in the A register. All functions return any error codes in DE with the carry bit in the PSW set.

Get-file-identifier Function

Registers on entry:

HL: If the 80H bit is set in A, HL points to a prompt string to be used by Rlwe in prompting the user (see RLWE in Section 3). If the 80H bit is not set in A, HL points to the text buffer to be examined in parsing the file descriptor. Note: this address MUST NOT be in the overlay area (2000H-27FFH) on the single user or if the Twin uses Gfid.OV.

DE: Points to the 44 byte area used to build the file descriptor (described above). Note that this area is first set to zero by Gfid.

BC: If the 20H bit in A is set, BC contains the default extension to use if the user does not specify an extension.

A: Flag bits, as follows:

80H: If set, read from user (via Rlwe) into an internal buffer, using the string pointed to by HL as a prompt string.
If clear, use HL as a pointer to the text to parse into the file identifier.

40H: If set, look up the resulting file. If the file exists, create an original file descriptor block at the address pointed to by DE. If an 0300H error (file does not exist) is returned, return an extended directory file descriptor block with NFA from the main directory in the FDA slot of the FDE.
If clear, just parse the file name into an extended directory file descriptor block.

20H: If set, use the contents of BC as the default extension if the user does not specify one.

1FH: These bits MUST be zero for the get-file-identifier function.

Registers on exit:

HL: Points to the ending delimiter symbol in the text buffer.

DE: If carry bit set in PSW, DE contains an error code/subcode; if carry is not set and 'look' was requested (i.e., 40H bit set in A on entry), then DE points to the FDE address in the directory. IMPORTANT NOTE: Because of the overlay mechanism, the directory in the SBUF1 area on return from Gfid MAY NOT BE FROM THE DISK CONTAINING THE DESIRED FILE (see Gfid, Gover, and Ovrto above).

BC: junk

A: junk

FLG: Carry bit set if errors detected; clear if not. If

'look' (40H in A) was specified and the carry bit is set, the zero flag reset indicates that error was on a directory lookup rather then the file lookup.

Description:

The Get-file-identifier function of Gfid relieves the assembly language programmer of the burden of parsing a generalized file identifier. This function is used extensively within the disk system itself by commands such as SAVE, DELETE, RENAME, COPY, PRINT, and TYPE.

You can either pass Gfid a text buffer to scan (useful where more than one file identifier may appear on a single line, as in the case of DELETE or RENAME) or request that Gfid read a specification from the user. If you direct Gfid to read a file specification directly from the user, you must supply Gfid with the address of a prompt string (as in the case of the system SAVE code). In either case, Gfid scans the appropriate text and attempts to parse it into a valid file identifier. Gfid will then look up the file identified if directed to do so by a set 40H bit in the PSW.

The file descriptor block is assumed to be 44 bytes long in order to contain maximum length file names. The block is initially zeroed by Gfid. If no extension was given by the user in the input to Gfid, the 80H bit of this initial byte (pointed to by DE on entry to Gfid) will be set. If you requested that Gfid look up the file (and the file exists), the buffer specified by DE will contain the drive number of the file plus an extension presence flag followed by the FDE copied from the directory (original file descriptor block). If the file was found, the file descriptor block now contains no information about the pathname for the file. Other system cells are available with the needed information. On return from a call to Gfid with 'look' specified, DirAddr contains the address of the directory that has the FDE of the last file found. If the 80H bit in NFDIR is set, the system cell PATH contains the directory names used to get to the directory of the file. The structure of PATH is similar to the lookup block. A length of name byte is followed by the name plus DX. This is repeated for each directory used in the path to the file. See the PATH printed by GFID-DEMO for examples. PATH is terminated by a zero byte.

If the file did not exist, 'look' will report a 0300H error and the buffer specified by DE will contain an extended directory file description block. Then the FDA slot in the file description block will contain the first free disk address on the specified device. Thus, if the user wants to create a new output file, the returned 0300H error code indicates that the file specified does not currently exist. In that case, the block

contains the first disk address to write to. The file descriptor block built by Gfid is designed to be easily read by the Gfid/Enter function (the function that creates file directory entries (FDEs)) and 'look', the Gfid file lookup function.

Special Note on Using the Get-file-identifier Service:

As mentioned in the register contents descriptions above, on the single user system or if the TwinSystem calls the Gfid overlay, you MUST NOT pass addresses in the HL and DE register pairs that are within the overlay area (2000H-27FFH). Doing so will cause anomalous behavior.

Looking up the file processing <?> as drive selector

Gfid processes the wild card device selector <?> in a file name. If Look was not specified in the call to Gfid, a 0509H error is generated. Note that if the error is returned, the filename has NOT been scanned into the lookup block. When <?> is recognized as the device selector, the disk drives in the system are searched for the file in the following manner:

1) Set drive number to SYSRES
2) If drive is zero go to 4.
3) Convert drive number to ASCII character, and store into string where ? was found. Look up file. If no errors are returned by Look, go to 5.
4) Increment drive number. If drive = SYSRES, put ? back into string and return. If drive number = 10 then set drive number to 0. Go to 2.
5) Store ASCII code for drive number at DEFPATH.

Note carefully that the input string is modified. If the user gives Gfid the string <?>Bessel, and file Bessel.BS is found on drive 2, the string <2>Bessel will be in the user's buffer, and the FDE for file <2>Bessel will be returned in the lookup block. If the file is not found on any drive, the string <?>Bessel will not be changed and the procedure for handling a 0300 error will be followed. Note that ANY error returned by Look causes Gfid to examine the next drive or stop the process. Also, note that SYSRES is examined first, and then the drive number is incremented to maxdrive (9) and then back to drive 1 until SYSRES is again reached.

Processing of <#> as drive selector

Gfid also processes the special symbol # as a drive selector, in coordination with the <?> process. Each time a <?> lookup succeeds, the drive number is bound to the variable #. Thus, if a <?> lookup finds its file on drive 2, a subsequent <#> lookup will search drive 2. The value bound to the # variable is

displayed by the Exec DISPLAY command and may be set by the Exec # command. The combination of <?> and <#> is very useful in command files. For example, the Exec command

```
Asmb <?>Source <#>Object
```

will search all the drives in the system for a file called Source.TX and will produce the object file Object.GO on the same drive the source file was found on. The # symbol is also legal with the commands LIST, PACK, DIR, and UNDELETE. This method of drive searching only searches the main directory of each drive.

The Exec # command may be used to set # to a pathname rather than just a drive number. For example, typing:

```
# <4<sp
DISP
```

will result in the display

```
Top of RAM is DDFF
Wild card path: 4<sp
Last error: 0000
```

The # symbol can now be used to perform operations in the sp sub-directory on drive 4. You can see the usefulness of this feature when the files you want to deal with are nested 5 sub-directories deep.

Interaction of default extension and user extension

When Gfid goes to Look up the file, the following procedure is used:

1) If no extension was passed to Gfid, and no default extension was given in BC, the Look is done with 80H+drive number passed to Look, allowing a match on any file with the same name. The 80H bit is returned in byte 0 of the lookup block, indicating that no extension was given by the user.

2) If no extension was passed to Gfid, but a default extension was passed in BC, the Look is done with the default extension, passing only the drive number to Look in A, requiring an exact match. The 80H bit is returned in byte 0 of the lookup block, indicating that no extension was given by the user.

3) If an extension was passed to Gfid, and no default extension was passed in BC, Look requires an exact match, using the user-supplied extension. The 80H bit

is not set in byte Ø of the returned lookup block, indicating the user supplied an extension.

4) If the user supplies an extension, and a default extension was passed in BC, the user-supplied extension is used in the Look, which follows the procedure given in (3) above.

Note that the 8ØH-bit in byte Ø of the returned lookup block, if set, indicates that the USER did not specify an extension. If the file was looked up by Gfid, an extension is present in the returned lookup block. If the 8ØH bit is returned set, this extension will match the default, if one was passed to Gfid. If no default extension was passed in BC, and the 8ØH bit is returned set, then the extension returned from the Look is from the first matching file on the drive. If no extension is supplied by the user, and no default extension is passed in BC, and the file is not found, the file descriptor block extension is two nulls!!

Termination characters and character scanning:

When Gfid parses the text buffer, it skips leading spaces and tabs. A file specification is delimited by a comma, plus sign, space, tab, or carriage return. The extension is separated from the file name by a dot. If no drive specification is given by the user, the drive number in SYSRES is assumed.

If you are invoking Gfid to scan multiple file specifications on a single line, note that the scan pointer passed in HL must be incremented past a comma or plus sign delimiter, since Gfid will not skip these characters.

Error Codes Returned by Get-file-identifier:

Ø5ØØ	Invalid disk number specified
Ø5Ø1	Name longer than 31 characters
Ø5Ø2	Extension longer than 2 characters
Ø5Ø3	Zero length name given
Ø5Ø9	<?> specified, but Look not requested

If Gfid is invoked requesting Look, then Ø3XX errors may be returned by Look and Dio (see Section 3, System Service Vectors).

Examples of Get-file-identifier Use:

```
        ; Sample coding showing use of Gfid to get a file
        ; identifier.  We want to get an input file
        ; from the user using .TX as a default extension,
        ; and have Gfid look it up for us.  OOPS is our error
        ; bailout point.
        ;
        ;
        BUF     DS      44      ; where to put the body
        Prompt  DB      'Input file is:',Ø
        ;
        Doit    LXI     H,Prompt        ; the prompt to use
                LXI     D,BUF           ; put stuff here.
                LXI     B,'TX'          ; default extension
                MVI     A,ØEØH          ; read, look, ext.
                gfid                    ; call gfid macro.
                JC      OOPS            ; no good. Complain.
        ;
        ; We now have drive # in BUF, FDE starting at BUF+1
        ; We need to pick up FDA and NSCTR, and start reading.
        ;
```

Enter/Replace FDE Function

Registers on entry:

HL: Points to file block built by Gfid (Get-file-identifier function) or file block in same format as a block built by Gfid. First byte of block contains disk drive number; this byte is followed by the FDE that is to be entered in the directory.

DE: unused

BC: "

A: 1H to enter new file into directory; 81H to replace existing FDE (File Directory Entry); ØC1H to replace exiting FDE and clear "new" bit (used by BACKUP).

FLG: unused

Registers on exit:

HL: junk

DE: If carry set in PSW, error code/subcode in register; else junk.

BC: junk

A: junk

FLG: If carry set, DE contains error code/subcode.

Description:

MOST IMPORTANT NOTE:
The Enter/Replace function will accept either the original file descriptor block or the extended directory file descriptor block. However, if the original file descriptor block is used, the file to be entered or replaced must be in the same directory as the last file looked up by Gfid. If the original file descriptor block is used Gfid assumes that the value of DirAddr is the correct address of the directory to be entered into. The safest ways to use the Enter/Replace function are to either just parse the file name and pass the resulting extended directory file descriptor to Gfid or to call Gfid 'look' function without using Ovrto to look up the file and then call it again immediately to do the Enter/Replace.

The Gfid Enter/Replace function allows you to enter or replace FDEs (File Directory Entries) in disk directories. The file block passed to the Gfid/Enter or Gfid/Replace functions is the same block returned by the Gfid/Get-file identifier function, or is in the same format as a file block built by Gfid/Get-file-identifier. The Gfid Enter/Replace functions are selected by a set 1H-bit in A. The Replace function is chosen over the Enter function if the 80H-bit is set in A.

The Enter function creates a new file directory entry (a new FDE) in a specified directory. No undeleted files with the same name and extension as the new file can exist on the disk, or a 0505 error code (file already exists) will be returned. The Replace function replaces the FDE for an existing file with a new FDE for that file. If the file that you specify does not exist, a 0300 error (file does not exist) will be returned. The Replace function CANNOT be used to change file names or extensions, but all other attributes within the FDE may be modified (such as deleted or system status, load and start addresses, etc.). Caution! Do not change the starting disk address (FDA) in the FDE. The PACK command assumes that the sequential ordering of FDEs in the directory corresponds to the sequential ordering of disk sectors in the files on the disk.

Error Codes Returned by Gfid/Enter or Gfid/Replace:

Enter:

Ø505H	File already exists
Ø504H	Directory full (file not entered)

Replace:

Ø300H	File does not exist

Since both Enter and Replace functions work with the directory, Ø3XX or Ø1XX errors may be reported as the result of data transfer errors.

Example of Replace Function Use:

The following routine demonstrates the use of the Gfid/Get-file-identifier and Gfid/Replace FDE functions for setting the "system" bit for specified files. It also demonstrates the use of CMPTR for accessing arguments on the command line. Once the program is assembled, it is invoked to "twiddle" a file by giving the program name, and the name of the file to "twiddle." It also demonstrates the ease of reporting errors by invoking the Emsg overlay.

```
;
; Example showing Gfid use to tweak system bit
; in FDE's....
;
        REFS    SYSTEM
        REF
;
CR      EQU     ØDH
TAB     EQU     9
FF      EQU     ØCH
;
        ORG     USER
        IDNT    $,$             ; set up LA and SA
;
        JMP     Start
        JMP     Start
;
ISON    db      'System Bit Now On!',CR,Ø
ISOFF   db      'System Bit Now Off!',CR,Ø
;
;
; The code.
;
Start   LHLD    CMPTR           ; point to arg
        LXI     D,BUF           ; buffer area
```

```
                MVI     A,40H           ; look up the file.
                gfid                    ; call appropriate gfid
                JC      OOPS            ; something wrong!
                LXI     H,BUF+1
                MOV     A,M             ; get tags byte
                XRI     40H             ; toggle sys bit
                MOV     M,A
                DCX     H               ; point at block start
                MVI     A,81H           ; tell 'em to replace
                gfid
                JC      OOPS            ; nope....
                LXI     H,ISON          ; point at on message
                LDA     BUF+1           ; get flag byte
                ANI     40H             ; check system bit
                JNZ     Msg             ; return printing on
                LXI     H,ISOFF         ; it's off
                JMP     Msg             ; return printing off
        ;
        ; Error reporting- kick Emsg to squeal on this thing!
        ;
        OOPS    CALL    Gover
                DB      'Emsg'          ; process the error.
                MVI     A,CR            ; doesn't do a CR!
                JMP     WH1             ; return doing CR.
        ;
        ;       The buffer for the file block
        ;
        BUF     EQU     $       ; file buffer
        ;
        ; The End
        ;
                END
```

Look Function

Registers on entry:

HL: Points to file block built by Gfid (Get-file-identifier function) or file block in same format as a block built by Gfid. First byte of block contains disk drive number; this byte is followed by the FDE that is to be entered in the directory.

DE: unused

BC: "

A: 2

FLG: unused

Registers on exit:

HL: junk

DE: If carry set in PSW, error code/subcode in register;

else DE points to FDE in SBUF1 (Remember about Ovrto).
BC: junk
A: junk
FLG: If carry set, DE contains error code/subcode, and ZERO indicates whether error was on a directory or the file.

Description:

MOST IMPORTANT NOTE:
The 'look' function will accept either the original file descriptor block or the extended directory file descriptor block. However, if the original file descriptor block is used, the file to be looked up must be in the same directory as the last file looked up by Gfid or it will not be found. If the original file descriptor block is used Gfid assumes that the value of DirAddr is the correct address of the directory to be examined. The safest ways to use the lookup function are to either just parse the file name and pass the resulting extended directory file descriptor to Gfid or to call Gfid get-file-identifer function with 'look' specified (40H bit in A).

'look' determines which type of file descriptor has been passed to it and loads the appropriate directory. ('look' uses DirAddr if passed an original file descriptor block and the path information in the file descriptor block if passed an extended directory file descriptor.) The function then scans the directory for the filename specified and if it finds it, sets DE to the address of the FDE in the directory. If the file is not found, 'look' returns CARRY and error code in DE. If passed an extended directory file descriptor, and CARRY is returned, the zero flag will be reset if the error occurred on a directory look up instead of the file look up. In other words, if all the directories specified in the file descriptor block were found but the file wasn't, 'look' will return CARRY and ZERO. This is useful in verifying that the directory specified was in fact found even though 'look' was given a file that did not exist in that directory.

Updir Function

Registers on entry:
HL: unused
DE: unused
BC: unused
A: 3
FLG: unused

Registers on exit:
HL: junk
DE: If carry set in PSW, error code/subcode in register; else junk.
BC: junk
A: junk
FLG: If carry set, DE contains error code/subcode.

Description:

MOST IMPORTANT NOTE:
The Updir function depends on the accuracy of DirAddr and NFDIR to provide its service. The directory at SBUF1 will be written to DirAddr on NFDIR so the directory that is in the directory area should have been loaded by looking up a file in that directory without using Ovrto.

Updir checksums the area at SBUF1, stores the checksum at NFCHK, gets the drive number from NFDIR, and writes 4 sectors at disk address DirAddr from memory address SBUF1. After updating the directory, it checks the other user's NFDIR and if it is on the same disk, invalidates it so the other user will not use an incorrect directory.

Section 3

System 88 Architecture

Memory Map of the System 88

The 8080A central processor can address 64K (K=1024) bytes of memory. This address space is segmented into the following regions on the System 88 disk system:

Locations 0000H-0BFFH	System ROM
Locations 0C00H-0DFFH	System Stack and Wormholes
Locations 0E00H-0FFFH	System Stack and Wormholes
Locations 1000H-17FFH	8" Controller Data area.
Locations 1800H-1BFFH	Video board RAM
Locations 1C00H-1EFFH	5" DD Controller Ram
Locations 1F00H-1FDFH	Reserved for expansion
Locations 1FE0H-1FEFH	8" Controller Control Area
Locations 1FF0H-1FFFH	NorthStar floating point board
Locations 2000H-FFFFH	Disk system RAM

The RAM area from 2000 onward is used differently in the Single and Twin systems. In the Twin system all shared code is in the area from E000H-FFFFH. Below is a functional listing of the major components and their locations in both single and twin:

Overlay Area	Single and Twin: 2000H-27FFH	
Directory Area	Single and Twin: 2800H-2BFFH	
Reserved System Area	Single and Twin: 2C00H-2EFFH	
Printer Driver	Single: 2F00H-31FFH	Twin: FC00-FFFFH
User Area	Single: 3200H-FFFFH	Twin: 2F00-DDFFH
Stack Area	Single: C80H-FFFH	Twin: DE00H-DFFFH
Shared Area	Single: n/a	Twin: E000H-FFFFH

In using system routines or data areas in assembly language programs, it is a good idea to use REF statements to define symbol values from the symbol file SYSTEM.SY rather than using an EQU with the value given in this manual. If system symbol values change from version to version, those programs using REFs require only reassembly, where those using EQUs require a great deal of editing. The use of REF also forces commonality in names of system routines and data areas.

SYSTEM EQUATES

This section of the System Programmer's Guide details the equates found in the SYSTEM file for both single and Twin systems. Unless otherwise noted, a symbol appears in both single and Twin system files with the same value.

UNLESS OTHERWISE NOTED, ALL VALUES ARE IN HEXADECIMAL

SYMBOL	PAGE	Single	Twin
BRGEN	31		0004
cmdf	32		0001
DBARF	35	0020	0020
DEVMASK	33	000F	000F
EERR	35	0040	0040
EIC	36	0080	0080
excl	32		0007
fupd	32		0004
KBD	34	0018	
mung	32		0005
PHANTOM	33		0060
rd	32		0002
USERS	31	0001	0002
Version	34	0081	
wlock	32		0006
wrt	32		0003

Symbol name: **USERS**

Single value: 0001
Twin value: 0002

Description:

USERS is defined as 1 or 2 primarily for use in conditional assembly in building the system. For example, in the IMAGE code, on a TwinSystem we must have exclusive use of the drive, and must call Devlock to get this. So, USERS is used as follows:

```
IF      USERS=2
MOV     C,A             ; get exclusive
CALL    Devlock         ; on this unit
JC      Oops            ; we didn't get it!
ENDIF
```

Symbol name: **BRGEN**

Twin value: 0004

See Also: PHANTOM, BRG

Description:

BRGEN defines the output port address of the baud rate generator in the system. It is used in the Twin core and in the printer driver to access the baud rate generator.

VERY IMPORTANT NOTE: Do not send any data out BRGEN or modify BRG, as a "crash" of the Twin System will be the result.

Symbol name: **cmdf**
Twin value: 0001

Symbol name: **rd**
Twin value: 0002

Symbol name: **wrt**
Twin value: 0003

Symbol name: **fupd**
Twin value: 0004

Symbol name: **mung**
Twin value: 0005

Symbol name: **wlock**
Twin value: 0006

Symbol name: **excl**
Twin value: 0007

See Also: Devlock, Dio, sett, clrt, setp, clrp, devlock

Description:

These equates define the access classes for the TwinSystem device manager, Devlock. They stand for, in order, command file read, read, write, file update, directory mung, write lock, and exclusive. Directory mung is known as "dirmod" in the SET command. The access class is passed in B to the Devlock system service; see the Devlock service for details, as well as the sett, clrt, setp, clrt, and devlock macros.

In use, read and write access is requested automatically by Dio; when Dio detects a write to sector 0000 of a volume, it requests mung access rather than write. BASIC is the sole user of file update access to prevent both users from opening INOUT files on the same drive. Exclusive is used by services such as INIT, PACK, and IMAGE for the destination drive. Directory mung is used by all services that may alter a directory, such as DELETE, RENAME, and creating files through the editor, assembler, or BASIC.

Symbol name: **PHANTOM**

Twin value: 0060

See Also: BRG, BRGEN, PMASK, Giveup

Description:

PHANTOM defines the bits in the baud rate generator that accomplish switching between users. The bits defined by PHANTOM cause switching of memory from one user to the other. As the baud rate generator on the CPU cannot be read, a copy of its current contents is kept in BRG.

VERY IMPORTANT NOTE: Do not send any data out BRGEN or modify BRG, as a "crash" of the Twin System will be the probable result.

Symbol name: **DEVMASK**

Value: 000F

Description:

DEVMASK is the device mask used for restricting unit numbers passed to Dio. Note that in previous versions of the system, this mask was effectively 7, restricting device numbers to Dio to the range 1 through 7. With this change, the range is 1 through F. Note that only 1 through 7 are defined in the Single User, and 1 through 9 in the TwinSystem.

Symbol name: **KBD**

Single value: 0018

See Also: SCRHM

Description:

KBD defines the address of the keyboard port in the system. It is also shifted over 8 bits to give the video board address in the single user system. It is not defined in the Twin, as it is not a good idea to go accessing the keyboard directly, because it may latch up the keyboard interrupt handlers. Rather than using KBD to get the address of the video display, use the contents of SCRHM to get the upper eight bits of this address.

Symbol name: **Version**

Single value: 0081

Description:

Version is the version number of the CPU board ROMS this system was assembled with.

Symbol name: **EIC**

Value: 0080

See Also: EFLG1, SBRK, PVEC, Ovrto, Gover

Description:

EIC is set in EFLG1 to indicate that the Exec is in control. This is used to mask out ^Y interrupts. It is also set by other system services that would rather not be interrupted by ^Y. It is cleared automatically by the overlay services Ovrto and Gover when they enter an overlay.

Symbol name: **DBARF**

Value: 0020

See Also: EFLG1

Description:

DBARF is a bit set by Exec in EFLG1 to tell Emsg and other parts of the system not to quit on errors. This bit gets set as the result of recognizing a question mark (?) at the start of a command to the Exec. Its action is to suppress the calling of Killi by Emsg and other error reporting services.

The Init code checks DBARF before calling Killi. This allows the INIT command to be run from a command file if it is preceded by a question mark:

```
LDA     EFLG1           ; see if we quit on
ANI     DBARF           ; command files
CZ      Killi           ; yup, don't want 'em
```

Symbol name: **EERR**

Value: 0040

See Also: Err, ERROR, EFLG1

Description:

EERR is set in EFLG1 by the root Err service to tell the Exec that it has an error code stored in ERROR to process. If, when Err is called, EERR is already set in EFLG1, the system takes an error halt, as it was unable to process the previous error. See the description of Err for more information.

SYSTEM MACROS

This section of the System Programmer's Guide details the assembler macros found in the SYSTEM file for both single and Twin systems. Unless otherwise noted, a macro appears in both single and Twin system files with the same definition.

Those unfamiliar with macros should look through the Macro 88 User's Guide, or other literature on macro processing. The sample overlay shown later in this volume uses a number of the macros shown here.

MACROS	PAGE	Single	Twin
ALIGN	51		*
clrp	40		*
clrt	39		*
db	55	*	*
dequ	48		*
devlock	41		*
dw	55	*	*
enter	43		*
gfid	53	*	*
giveup	50		*
gover	44		*
ioret	50		*
leave	43		*
lock	44		*
overlay	54	*	*
overto	46		*
print	45		*
ralign	52		*
rddef	51		*
rds	52		*
rorg	52		*
setp	40		*
sett	39		*
show	45		*
unlock	44		*
userpgm	47		*
vcb	42		*
vect	49		*
verdate	42		*

Symbol name: **clrt**

Twin Macro:

```
clrt    MACRO
#L      PUSH    B
        MVI     B,ØEØH+#1
        CALL    ØEØ6FH
        POP     B
        ENDM
```

See Also: Devlock, cmdf, rd, wrt, fupd, mung, wlock, excl, sett, clrp, setp

Description:

The clrt macro is used to clear a temporary device allocation through Devlock on the TwinSystem. It takes as its one parameter the device access code, one of (cmdf, rd, wrt, fupd, mung, wlock, excl). BC is preserved over the call to Devlock, and the device number for Devlock is expected to be in C. An absolute hex address is used in the macro expansion because the Devlock service vector is not expected to move. Although it does eliminate the need for a separate REF statement, it is a questionable practice; using the symbolic name and requiring the REF would be more easily understood.

Symbol name: **sett**

Twin Macro:

```
sett    MACRO
#L      PUSH    B
        MVI     B,ØAØH+#1
        CALL    ØEØ6FH
        POP     B
        ENDM
```

See Also: Devlock, cmdf, rd, wrt, fupd, mung, wlock, excl, clrt, clrp, setp

Description:

The sett macro is used to set (get) a temporary device allocation through Devlock on the TwinSystem. It takes as its one parameter the device access code, one of (cmdf, rd, wrt, fupd, mung, wlock, excl). BC is preserved over the call to Devlock, and the device number for Devlock is expected to be in C.

Symbol name: **clrp**

Twin Macro:

```
clrp    MACRO
#L      PUSH    B
        MVI     B,0C0H+#1
        CALL    0E06FH
        POP     B
        ENDM
```

See Also: Devlock, cmdf, rd, wrt, fupd, mung, wlock, excl, sett, clrt, setp

Description:

The clrp macro is used to clear a permanent device allocation through Devlock on the TwinSystem. It takes as its one parameter the device access code, one of (cmdf, rd, wrt, fupd, mung, wlock, excl). BC is preserved over the call to Devlock, and the device number for Devlock is expected to be in C.

Symbol name: **setp**

Twin Macro:

```
setp    MACRO
#L      PUSH    B
        MVI     B,80H+#1
        CALL    0E06FH
        POP     B
        ENDM
```

See Also: Devlock, cmdf, rd, wrt, fupd, mung, wlock, excl, clrt, sett, clrp

Description:

The setp macro is used to set a permanent device allocation through Devlock on the TwinSystem. It takes as its one parameter the device access code, one of (cmdf, rd, wrt, fupd, mung, wlock, excl). BC is preserved over the call to Devlock, and the device number for Devlock is expected to be in C.

Symbol name: **devlock**

Twin Macro:

```
devlock MACRO
#L      PUSH    B
        MVI     B,+#1
        CALL    ØE06FH
        POP     B
        ENDM
```

See Also: Devlock

Description:

The devlock macro is used to call the Devlock service in the TwinSystem. Its one argument is the access code, one of (cmdf, rd, wrt, fupd, mung, wlock, excl) which is loaded into B. C is expected to contain the device number, and BC is preserved over the call. This macro is used in code that has been changed from the single user system to the Twin by placing it before a call to Dio to get some special access. An absolute hex address is used in the macro expansion because the Devlock service vector is not expected to move. Although it does eliminate the need for a separate REF statement, it is a questionable practice; using the symbolic name and requiring the REF would be more easily understood.

Symbol name: **vcb**

Twin Macro:

```
vcb     MACRO
#L      DB      #1+0,#2+0,#3+0
        DW      #4+0,#5+0,#6+0,#7+0,#8+0
        ENDM
```

Description:

The vcb macro is used internally in the system to set up the control tables for disk devices. As there are no hooks in that part of the system allowing user access, it will not be described further.

Symbol name: **verdate**

Twin Macro:

```
verdate MACRO
#L      DB      '7/22/80 RTM'
        ENDM
```

Description:

The verdate macro expands into the creation date for the TwinSystem resident.

Symbol name: **leave**

Twin Macro:

```
leave   MACRO
#L      PUSH    H
        LXI     H,#1
        CALL    Leave
        POP     H
        ENDM
```

See Also: Leave, Enter, enter

Description:

The leave macro is used to call the Leave service, which leaves the critical region defined by the semaphore address defined by the argument to the macro. The semaphore address must be in memory accessible by both users.

Symbol name: **enter**

Twin Macro:

```
enter   MACRO
#L      PUSH    H
        LXI     H,#1
        CALL    Enter
        POP     H
        ENDM
```

See Also: Leave, Enter, leave

Description:

The enter macro is used to invoke the Enter service, requesting entry into a critical section defined and protected by the semaphore address given as the argument. The semaphore address must be in memory accessable by both users. If the semaphore is currently held, the user is blocked until the semaphore is released.

Symbol name: **unlock**

Twin Macro:

```
unlock  MACRO
#L      CALL    Unlock
        ENDM
```

See Also: Unlock, Lock, lock

Description:

The unlock macro invokes the Unlock service, which allows switching between users to take place again. This service and the Lock service should be used for extremely short periods of time, and with caution, as careless use will cause performance degradation or system failure.

Symbol name: **lock**

Twin Macro:

```
lock    MACRO
#L      CALL    Lock
        ENDM
```

See Also: Lock, Unlock, unlock

Description:

The lock macro calls the Lock service, which locks the current user in the Twin against switching. This service and the Unlock service should be used for extremely short periods of time, and with caution, as careless use will cause performance degradation or system failure.

Symbol name: **print**

Twin Macro:

```
print   MACRO
#L      CALL    Print
        DB      #A,Ø
        ENDM
```

See Also: Print, Show, show

Description:

This service prints the text supplied as the argument on the system printer through WH7. Note that the macro supplies a zero terminating byte, so that multiline text cannot be printed using this macro.

Symbol name: **show**

Twin Macro:

```
show    MACRO
#L      CALL    Show
        DB      #A,Ø
        ENDM
```

See Also: Print, Show, print

Description:

This macro displays the text passed as the argument on the display screen. Note that a terminating zero byte is supplied by the macro.

Symbol name: **gover**

Twin Macro:

```
gover   MACRO
#L      EQU     $
        IF      NOT NULL[#2]
        MVI     A,#2
        ENDIF
        CALL    Gover
        DB      '#1'
        IF      NOT NULL[#3]
        JC      #3
        ENDIF
        ENDM
```

Symbol name: **overto**

Twin Macro:

```
overto  MACRO
#L      EQU     $
        IF      NOT NULL[#2]
        MVI     A,#2
        ENDIF
        CALL    Ovrto
        DB      '#1'
        IF      NOT NULL[#3]
        JC      #3
        ENDIF
        ENDM
```

See Also: Gover, Ovrto, gover

Description:

These macros are used to invoke overlays in a fairly general manner. The first argument is mandatory, and is the 4 character overlay name, without quotes. If the second argument is present, it is loaded into A as a function code. If the third argument is present, it is used as the address to jump to if the overlay returns with the Carry bit set in PSW. The differences between the Gover and Ovrto are described in the section on system services; briefly, Ovrto "remembers" the overlay currently in the overlay area and restores it on returning, and Gover does not.

Symbol name: **userpgm**

Twin Macro:

```
     userpgm MACRO
             ORG     USER
             IDNT    $,$
             JMP     $+6
             JMP     #2
             LXI     H,Ø
             DAD     SP
             SHLD    ESP
             IF      NOT NULL[#3]
             CALL    Show
             DB      #3,ØDH,Ø
             ENDIF
             JMP     #1
     ESP     DW      Ø
             ENDM
```

See Also: Show, USER

Description:

This macro is used to generate a program header for user programs. The first argument is mandatory, and is the starting address of the program. The second argument is also mandatory, and is the reentry address in the program. The third argument is optional, and if present is expected to be a text string enclosed in quotes. This string is displayed on the screen when the program begins execution. The entry stack pointer is also stored in ESP for use in error recovery and stack limit checking in the program.

Symbol name: **dequ**

Twin Macro:

```
dequ    MACRO
#L      EQU     #1
        DEF     #L
        ENDM
```

Description:

The dequ macro is used as a shorthand to equate a symbol to a value and define that symbol. It is used in building the system symbol files, and in BASIC.

Symbol name: **vect**

Twin Macro:

```
vect    MACRO
#L      EQU     rpc
        DEF     #L
rpc     SET     rpc+3
        ENDM
```

Description:

The vect macro is used in the TwinSystem to define the jump vectors starting at E000H. The symbol given is equated to the value of rpc, and the symbol defined. Then rpc is incremented by 3 to account for the JMP instruction that will be generated in the eventual code.

This macro is used in the following manner in generating the symbol file for the TwinSystem:

```
rpc     SET     0E000H
Cold    vect                    ; 00 cold start
Warm    vect                    ; 03 warm start
Msg     vect                    ; 06 display msg
```

This defines the symbol Cold with value E000H, Warm with value E003H, and Msg with value E006H, while not generating any code.

Symbol name: **ioret**

Twin Macro:

```
ioret   MACRO
#L      JMP     40H
        ENDM
```

See Also: Ioret

Description:

This macro expands into a call to Ioret, to return from an interrupt or to return to an environment placed on the stack. An absolute hex address is used as the code for Ioret is in ROM.

Symbol name: **giveup**

Twin Macro:

```
giveup  MACRO
#L      CALL    Giveup
        ENDM
```

See Also: Giveup

Description:

The giveup macro expands into a call to Giveup, which gives up the processor in the TwinSystem.

Symbol name: **ALIGN**

Twin Macro:

```
ALIGN   MACRO
        ORG     ($+#1) AND NOT (#1-1)
#L      EQU     $
        ENDM
```

Description:

The ALIGN macro is used to force the assembler's program counter ($) to the specified boundary. For example, ALIGN 100H forces the program counter to the next page boundary (lower 8 address bits all zero). Note that it is a bad programming practice to assume that the area from the current location counter to the ALIGNed area contains zeros.

Symbol name: **rddef**

Twin Macro:

```
rddef   MACRO
#L      EQU     rpc
        DEF     #L
rpc     SET     #1+rpc
        ENDM
```

See Also: rds, rorg, ralign

Description:

The rddef macro is used to define a symbol and calculate space for it while not generating any code. It is used extensively in generating the symbol file for the TwinSystem.

The following example shows use of the rddef and dequ macros in generating the TwinSystem symbol file:

```
rpc     rorg    2000H           ; Stuff starts here

OVRLY   rddef   2048            ; 2K for overlay
OVENT   dequ    OVRLY+4         ; overlay entry point

SBUF1   rddef   1024            ; directory area
CBUF    rddef   256             ; command file buffer
```

Symbol name: **ralign**

Twin Macro:

```
ralign  MACRO
rpc     SET     (rpc+#1) AND NOT (#1-1)
#L      EQU     rpc
        ENDM
```

See Also: rddef, rds, rorg

Description:

The ralign macro is used with the rddef and rds macros to force the pseudo-location counter rpc to the specified boundary.

Symbol name: **rds**

Twin Macro:

```
rds     MACRO
#L      EQU     rpc
rpc     SET     #1+rpc
        ENDM
```

See Also: rorg, rddef, ralign

Description:

The rds macro is used to advance the pseudo-location counter rpc. It is used in building the TwinSystem symbol file.

Symbol name: **rorg**

Twin Macro:

```
rorg    MACRO
#L      EQU     #1
rpc     SET     #1
        ENDM
```

See Also: rddef, rds, ralign

Description:

The rorg macro is used with the rddef, rds, and ralign macros to force the pseudo-location counter rpc to a specified starting value. See the example given for the rddef macro.

Symbol name: **gfid**

Single Macro:

```
gfid    MACRO
        IF      USERS=2
#L      CALL    Gfid
        ELSEIF  $>2800H
#L      CALL    Ovrto
        DB      'Gfid'
        ELSE
#L      CALL    OVGFID
        ENDIF
        ENDM
```

Twin Macro:

```
gfid    MACRO
#L      CALL    Gfid
        ENDM
```

See Also: USERS, Gfid, OVRLY, EIC, Ovrto, Gover

Description:

The gfid macro expands into a call to the Gfid service on single and Twin systems. On the Twin system, Gfid is a resident service, so it is called directly. On the single user system, Gfid is an overlay, and so must be invoked differently if the macro is expanded from within the overlay area (program counter between 2000H and 2800H). The OVGFID routine called in single user overlays disconnects ^Y by pushing the contents of PVEC onto the stack and pointing PVEC at Ioret. When Gfid returns, the original contents of PVEC are restored. This is done to correct a timing window in the processing of ^Y in the single user system when overlays are fetched (see description of Ovrto for details).

Symbol name: **overlay**

Macro:

```
overlay MACRO
        ORG     OVRLY
        IDNT    $,$
        DB      #1
        IF      NOT NULL[#2]
        JMP     #2
        ENDIF
        ENDM
```

Description:

The overlay macro expands into the header for an overlay. The first mandatory argument is the overlay name in quotes. The second argument, which is optional, expands into a jump to the address given.

Symbol name: **db**

Macro:

```
db      MACRO
#L      DB      #A
        ENDM
```

Symbol name: **dw**

Twin Macro:

```
dw      MACRO
#L      DW      #A
        ENDM
```

Description:

The dw and db macros are used mainly to suppress the display of generated code from long data lists. If MACLIST Ø is included in the assembly source file, macro expansions are not listed. So, when the dw or db macro is used instead of the DW or DB statements, no generated code is displayed in the listing.

SYSTEM DATA AREAS

This section details data areas used by the single and Twin systems. Many symbols appear in both single and Twin systems, but have differing addresses in each. As in the descriptions for system equates and service vectors, references are made to other interacting items in the system.

UNLESS SPECIFICALLY STATED, ALL VALUES ARE IN HEXADECIMAL

SYMBOL	PAGE	Single	Twin
BHA	74		E05F
BOOTVOL	88		2D92
BRG	82		0C60
BUGS	103	2DFC	2EB4
BUSIES	85	0C6E	0C6E
CBUF	63	2C00	2C00
CMDA	65	2D8C	2E82
CMDD	64	2D89	2E7F
CMDF	63	2D88	2E7E
CMDN	65	2D8E	2E84
CMDP	64	2D8A	2E80
CMND	96	2D40	2E3C
CMPTR	96	2DC7	2E7C
Command	81	0C4C	
DEFPATH	99	2E27	2D80
DioA	84	0C66	0C66
DioBsy	85	0C6C	0C6C
DioDn	84	0C6B	0C6B
DioDrv	84	0C69	0C69
DioHL	84	0C67	0C67
DirAddr	93	2E02	2EA7
DONT	79	2D90	2E8E
DRVADTAB	83	0C7E	0C7E
EFLG1	102	2DC9	2EB0
EFLG2	102	2DCA	2EB1
ERROR	104	2D9A	2EB7
EXECSP	92	2DAF	2EBD
FILE	92	2DCB	2EBF
GFLOCK	75		E07E
IOIP	83		0C62
JOBST	103	2D9E	2EB2
KBEX	70	2D86	2E38
KBIG	67	2D84	2E36
KBIP	67	2D82	2E34
KBMODE1	71		2E3A
KBMODE2	71		2E3B
KBUF	66	2D00	2DC0
LERR	104	2D9C	2EB9
LOCK	100		2E8F
LUSER	103	2DC6	2EB3

SYMBOL	PAGE	Single	Twin
MemAdd	81	0C49	
MEMTOP	92	2D80	2EBB
MTO	89	2DA2	
MUNG1	95	2DA7	2E9F
MUNG2	95	2DA9	2EA1
MUNG3	95	2DAB	2EA3
MUNG4	95	2DAD	2EA5
MUNGP	88		2D00
NDRIVES	89	2D9F	2EA9
NFA	101	2E00	2EAD
NFCK	94	2DA1	2EAC
NFDIR	94	2DA0	2EAB
ONCE	101	2DC5	2EAF
OVBC	72	2DC1	2E9B
OVDE	72	2DBF	2E99
OVENT	86	2004	2004
OVHL	72	2DBD	2E97
OVMEM	100	2E53	
OVNM	72	2DB6	2E90
OVPSW	72	2DC3	2E9D
OVRLY	86	2000	2000
Pages1	81	0C4B	
PATH	99	2E04	2D40
PMASK	74		E05D
POS	99	0C0E	2E0A
PVEC	76	2D93	2E87
SBRK	76	2D91	2E86
SBUF1	87	2800	2800
SBUF2	87	2900	
SBUF3	87	2A00	
SBUF4	87	2B00	
SCEND	98	0C1E	2E08
SCHR	78	2D98	2E8C
SCREEN	86	1800	
SCRHM	98	0C1F	2E09
SINT	89	2DB3	
SRA1	60	0C10	0C10
SRA2	60	0C12	0C12
SRA3	60	0C14	0C14
SRA4	60	0C16	0C16

SYMBOL	PAGE	Single	Twin
SRA5	61	ØC18	ØC18
SRA7	1Ø5	ØC1C	2EEC
SRA7I	61		ØC1C
STACK	85	1ØØØ	
SUWH8	62		ØC4Ø
SYSRES	93	2D92	2EAA
TIMER	91	ØCØØ	2EØ2
UBRK	77	2D97	2E8B
UCHR	78	2D99	2E8D
USER	73	32ØØ	2FØØ
USP	9Ø		2EØØ
USRNAME	9Ø		2EEE
USTATS	75	2DB1	EØ5E
UTIME	91		2EØ2
UVEC	77	2D95	2E89
VCBTAB	83	ØC63	ØC63
VERLOC	8Ø	Ø439	
WAKEUP	97	ØC1A	2EØ6
WHICH	83		ØC61
WH8	62	ØC4Ø	2E2C
XTIMER	8Ø		ØCØØ

Symbol name: **SRA1**

Value: ØC1Ø

Description:

SRA1 contains the address of the interrupt handler for memory parity interrupts generated by new memory boards such as the 48K memory used in the Twin. This location should not be modified by the user.

Symbol name: **SRA2**

Value: ØC12

Description:

SRA2 is the service vector for sector pulse interrupts from the 5" SSSD disk controller. It should not be modified by the user.

Symbol name: **SRA3**

Value: ØC14

Description:

SRA3 is reserved for future use by the PolyNet interface.

Symbol name: **SRA4**

Value: ØC16

Description:

SRA4 contains the address of the interrupt service routine for the USART. Caution should be used in handling of this interrupt by user programs. In the TwinSystem, this interrupt is best left to the system, as the USART baud rate generator latch controls user switching.

Symbol name: **SRA5**

Value: ØC18

Description:

SRA5 contains the address of the keyboard interrupt service routine. This location should not be modified by the user in either single or Twin systems.

Symbol name: **SRA7I**

Twin value: ØC1C

Description:

SRA7I is the TwinSystem symbol for the single step interrupt vector. In the TwinSystem, each user has a separate copy of SRA7, and the system vectors the common SRA7I fielded interrupt through the proper user's SRA7 vector. This allows each user to use the front panel code at the same time.

Symbol name: **WH8**

Single value: 0C40
Twin value: 2E2C

Symbol name: **SUWH8**

Twin value: 0C40

See Also: Ticker, WAKEUP

Description:

WH8 is used as an interrupt service vector. It contains the address of the routine to be entered when the real time clock interrupts. In the single user system, it is connected to the normal clock processing logic. In the TwinSystem, it is connected to Ticker, the TwinSystem clock handler. TwinSystem WH8 is not used. WH8 should not be modified by the user, as the real time clock is fundamental to the operation of the system. The WAKEUP vector is provided for use of the real time clock by user programs. The coding for the clock interrupt handler in ROM is essentially:

```
Clock   PUSH    PSW
        PUSH    B
        PUSH    D
        PUSH    H       ; std save sequence
        LHLD    WH8     ; get vector
        PCHL            ; go do it.
```

Symbol name: **CBUF**

Value: 2CØØ

See Also: CMDD, CMDF, CMDP, CMDN, BASIC

Description:

CBUF is the 256 byte buffer used to hold the current sector of a command file.

BASIC also makes use of CBUF. When running programs, if BASIC detects that command files are not in use, by CMDF containing zero, it uses CBUF to hold its line number cache. This cache is a list of 32 bit entries (64 in number, to give 256 bytes), consisting of 16 bits of BASIC line number followed by 16 bits of memory address for the start of the program line. An entry is placed in the cache by Findln only when it is searched for as the result of a GOTO, GOSUB, or similar statement. This makes a 64 entry cache quite effective. When the cache fills up, new entries are placed in the cache treating it as a circular list. This use of a line number cache greatly speeds up line number searches in BASIC, since BASIC normally searches for target line numbers by starting at the beginning of the program text and scanning forward line by line.

Symbol name: **CMDF**

Single value: 2D88
Twin value: 2E7E

See Also: Killi, DBARF, CMDD, CMDN, CMDP, CBUF

Description:

CMDF is a single byte flag used to indicate that characters are being read from a command file. If CMDF is zero, command file mode is not active, and requests for characters through WHØ are satisfied from the keyboard buffer. If CMDF is nonzero, character requests through WHØ are satisfied from the command file buffer CBUF. CMDF is set nonzero by the Exec in setting up a command file, and set to zero by either Killi in killing command files, or by the command file code itself when end of file is detected on the command file.

Symbol name: **CMDD**

Single value: 2D89
Twin value: 2E7F

See Also: CMDN, Devlock, cmdf, Killi

Description:

CMDD is used to hold the drive number for the active command file. It is set by the Exec in starting up a command file.

In the TwinSystem, each time a sector is read from the drive noted by CMDD into CBUF, permanent command file read access (cmdf) is requested through Devlock. This permanent command file read access is cleared when end of file is reached on the command file, or through calling Killi and aborting command files.

Symbol name: **CMDP**

Single value: 2D8A
Twin value: 2E8Ø

See Also: CBUF, CMDA

Description:

CMDP is the 16 bit pointer into the command file buffer CBUF, and points to the next character to remove from the buffer. When the pointer points to CBUF+1ØØH, another sector must be read from the disk; the drive number is in CMDD and the disk address is in CMDA. When a command file is set up by the Exec, CMDP is initialized to CBUF+1ØØH, so that the first character requested through WHØ causes the first sector of the command file to be read into CBUF.

Symbol name: **CMDA**

Single value: 2D8C
Twin value: 2E82

See Also: CBUF, CMDD, CMDP, CMDF

Description:

CMDA is the 16 bit disk address of the next sector to read from the disk. It is initialized by the Exec to the starting disk address of the command file, and incremented after each new sector is read from the disk.

For Exec/90 and later systems, CMDA will be properly adjusted by PACK if the disk containing the currently active command file is PACKed. On earlier systems, this is not done, with the result that the next sector read, after the disk is shuffled by PACK, may no longer be the next sector of the original command file!

Symbol name: **CMDN**

Single value: 2D8E
Twin value: 2E84

See Also: CMDF, cmdf, Devlock

Description:

CMDN is the 16 bit number of sectors remaining in the command file. It is set by the Exec to the number of sectors in the command file, and decremented after each sector is read. When a new sector is needed, and CMDN is zero, CMDF is set to zero to disable command files, as end of file has been hit. On the TwinSystem, permanent command file read (cmdf) is also cleared for the drive.

Symbol name: **KBUF**

Single value: 2DØØ
Twin value: 2DCØ

See Also: KBIP, KBIG, Killi, Flush

Description:

KBUF is the address of the 64 byte keyboard ring buffer. As keyboard characters are picked up, they are placed in the ring buffer until the buffer fills. Once the buffer is full, new characters are dropped on the floor.

Symbol name: **KBIP**

Single value: 2D82
Twin value: 2E34

Symbol name: **KBIG**

Single value: 2D84
Twin value: 2E36

See Also: KBUF, Killi, Flush, KBEX

Description:

KBIP and KBIG are the keyboard ring buffer "put" and "get" pointers. They are altered at the interrupt level. They should not be modified by the user at any time. They may be examined to see if there are characters in the ring buffer by code such as the following, which is extracted from BASIC's INP(Ø) function:

```
DI              ; don't bug me!
LHLD    KBIP    ; get put pointer
LDA     KBIG    ; and lsb of get
EI              ; allow ints again
CMP     L       ; see if equal
LXI     H,Ø     ; assume so, return Ø
JZ      AINP1   ; jmp/empty, return Ø
INR     L       ; else return 1, we've
JMP     AINP1   ; got something there.
```

The above code returns a ØØ in HL if there are no characters in the ring buffer, and a 1 in HL if there are. This is determined by comparing the put and get pointers. Since these pointers are altered at the interrupt level, interrupts must be disabled while they are fetched to insure that both are correct. If both pointers are the same, the buffer is empty.

Here is the code used to put a character into the ring buffer. It is copied from the keyboard interrupt service code for the TwinSystem, the character to be placed in the buffer is in C, and the interrupts are disabled:

```
LHLD    KBIP    ; Put pointer
MOV     M,C     ; poke into buffer
DCR     L       ; dink pointer
MVI     A,(KBUF-1) AND ØFFH
CMP     L       ; did we wrap around?
JNZ     Kbi1    ; jmp/nope
MVI     L,(KBUF+63) AND ØFFH ; reset if so
```

```
Kbil    LDA     KBIG    ; see if we're full
        CMP     L
        JZ      Kbxx    ; jmp/yup, drop on floor!
        SHLD    KBIP    ; else update put ptr
        JMP     Kbxx    ; and split
```

The ring buffer put pointer is loaded, and the character stored into the ring buffer. The put pointer is decremented and checked for wraparound. The ring buffer get pointer is loaded and compared to the put pointer. If they are equal, we have 63 characters in the ring buffer; it is full. We exit without storing the updated put pointer. Once the buffer fills, incoming characters are stored on top of the last character in the buffer. If the buffer is not full, we store the updated put pointer and exit.

Here is the code that removes characters from the ring buffer, again taken from the TwinSystem. Cin is hooked up to WHØ. Note the checks for command files (CMDF) and the use of giveup:

```
Cwt     POP     H       ; restore saved thing
        giveup          ; wait for a while

; Try to get a chr!

Cin     LDA     CMDF    ; see if command files active
        ORA     A
        JNZ     Cfin    ; jmp/yup, go check over there

        DI              ; don't bug me!
        PUSH    H       ; save HL on the stack
        LHLD    KBIG    ; get pointer
        LDA     KBIP    ; put pointer
        CMP     L       ; anything there?
        JZ      Cwt     ; jmp/nope, go wait & try again

        MOV     A,M     ; pick up chr
        PUSH    PSW     ; stash on stack
        DCR     L       ; dink ptr
        MVI     A,(KBUF-1) AND ØFFH
        CMP     L
        JNZ     Cin1    ; jmp/didn't wrap around
        MVI     L,(KBUF+63) AND ØFFH
Cin1    SHLD    KBIG    ; save new pointer
        POP     PSW     ; restore chr
        EI              ; allow interrupts
        LHLD    KBEX    ; get tail end handler address
        XTHL            ; swap onto stack, get HL back
        RET             ; return thru post processor
```

This routine is entered at Cin from WHØ. First, the check for command files being active is made, and control is transferred to the command file get routine (Cfin) if so. Then interrupts are disabled and HL is pushed onto the stack. Interrupts are disabled because KBIP and KBIG may be altered at the interrupt level. HL is pushed so we do not alter any registers but A and PSW. If the get and put pointers are equal, the buffer is empty; we go to Cwt where HL is restored, and we give up the processor. When control is returned to our task, we try again falling into Cin. If the buffer had something in it, we load the character, decrement the pointer mod ring size, and store the updated pointer. Control is returned to the caller of Cin/WHØ by going through the routine pointed to by KBEX. This is usually Fold, Flip, or just a RET instruction. KBEX can be pointed to custom routines for special purposes.

These routines show the interaction of KBIP, KBIG, and KBUF. KBIG and KBIP are initially set up by calling Killi, which resets them and marks the ring buffer clear. Note also the use of KBEX, which is used to include post processing routines such as Flip and Fold.

Symbol name: **KBEX**

Single value: 2D86
Twin value: 2E38

See Also: KBUF, KBIG, KBIP, Flip, Fold

Description:

KBEX holds the address of the character input postprocessing routine. The code pointed to by KBEX is executed for each character read through WHØ. KBEX is initialized to point to a RET instruction. The Exec Flip and Fold commands point KBEX at routines Flip and Fold. The routine connected to KBEX must not modify registers BC, DE, or HL. See KBIP and KBIG for the code involving KBEX.

Symbol name: **KBMODE1**
Twin value: 2E3A

Symbol name: **KBMODE2**
Twin value: 2E3B

See Also: PHANTOM, SET SKM command

Description:

KBMODE1 and KBMODE2 are used in the Twin to control single keyboard mode, which is enabled by the Exec command SET SKM ON. When active, this mode allows both users to be controlled from a single keyboard. As KBMODE1 and KBMODE2 are in per-user memory, SKM is enabled on a per-user basis. That is, if user 1 has enabled SKM, user 1 can type into user 1 or user 2, but for user 2 to type into user 1, user 2 must also enable SKM.

Single keyboard mode (SKM) when enabled is controlled by the I and II function keys on Keyboard III, (on Keyboard II, ^ = I = 1C hex and ^] = II = 1D hex). When SKM is active, these character codes will not be seen by programs; they are trapped within the keyboard handler.

If KBMODE1 is zero, SKM is disabled. KBMODE1 nonzero means that SKM is active, and KBMODE2 has ØØ to indicate the characters go to user 1, and PHANTOM if the characters go to user 2.

Note that since WordMaster II depends on the use of the I and II function keys, it disables SKM by storing a zero in KBMODE1.

Symbol name: **OVNM**

Single value: 2DB6
Twin value: 2E90

See Also: Ovrto, Gover

Description:

OVNM is a 6 byte area used internally by the system in overlay fetch processing. It contains the name of the target overlay, followed by the extension OV for Look to find. It should not be used or modified by the user.

Symbol name: **OVHL**

Single value: 2DBD
Twin value: 2E97

Symbol name: **OVDE**

Single value: 2DBF
Twin value: 2E99

Symbol name: **OVBC**

Single value: 2DC1
Twin value: 2E9B

Symbol name: **OVPSW**

Single value: 2DC3
Twin value: 2E9D

Description:

OVHL, OVDE, OVBC, and OVPSW are temporaries used in overlay processing to hold the contents of the registers. They should not be used or modified by the user.

Symbol name: **USER**

Single value: 3200
Twin value: 2F00

Description:

USER is the start of the user memory area. All user programs are assumed to start here. Program start addresses are assumed to be equal to USER if the Exec START command is to work, and program reentry addresses are assumed to be USER+3 for the Exec REENTER command to work.

Symbol name: **PMASK**

Twin value: E05D

See Also: BHA, PHANTOM, SET SOLO command

Description:

PMASK is the processor switch mask in the TwinSystem. If PMASK contains 00, then the system is either running as a single user, on single user hardware, or is running in SOLO mode as the result of the SET SOLO command. If PMASK is zero as the result of the SET SOLO command, the 20H bit in BHA will be set, allowing PMASK to be reset to PHANTOM when the SET TWIN command is given.

Altering PMASK is an excellent way of blowing up the TwinSystem and probably the quickest.

Symbol name: **BHA**

Twin value: E05F

See Also: Devlock, SET YAK, SET SOLO commands

Description:

BHA contains a number of flag bits used in the TwinSystem. The currently allocated bits are:

80H YAK enabled. If the 80H bit in BHA is set, device allocations through Devlock will be reported on the video screen for both users. This is enabled by the Exec command SET YAK ON and disabled by the Exec command SET YAK OFF as well as by modifying the bit directly.

40H Two users. The 40H bit is used in the processing of the SET SOLO command. It is set during SET SOLO to indicate that before solo mode was started, two users were running. This bit should not be altered by the user.

20H WAIT mode. Setting the 20H bit in BHA enables device WAIT mode. This is normally set by the Exec command SET WAIT ON and disabled by SET WAIT OFF Exec command. See the description of Devlock for more details. This bit can be turned on and off by user programs, as long as other bits are not affected.

1FH These bits are reserved for future use in the TwinSystem.

Symbol name: **USTATS**

Single value: 2DB1
Twin value: E05E

Description:

USTATS contains the current status of the USART maintained by the system printer driver. It is not normally modified by user programs. On the TwinSystem especially, any modifications should be done with extreme care and an understanding of the printer driver and user switching.

Symbol name: **GFLOCK**

Twin value: E07E

See Also: Gfid

Description:

GFLOCK is the semaphore for Gfid. It should not be modified by the user under any circumstances, as a complete failure of the file system may result.

Symbol name: **SBRK**

Single value: 2D91
Twin value: 2E86

See Also: PVEC, DONT, EIC, EFLG1, GFLOCK

Description:

SBRK is the single byte flag set nonzero when a ^Y is recognized at the keyboard interrupt level. It is the user's responsibility to clear SBRK. Note that SBRK is set nonzero by ^Y if the ^Y action is masked by DONT (or GFLOCK in the Twin), but is not set if EIC is set in EFLG1; see the coding for escape characters and the other descriptions for details.

Symbol name: **PVEC**

Single value: 2D93
Twin value: 2E87

See Also: SBRK, DONT, EIC, EFLG1, GFLOCK, Ioret, Iexec

Description:

PVEC contains the address of the routine to enter as the result of a ^Y being received from the keyboard. The action of ^Y is masked by DONT, EIC set in EFLG1, and GFLOCK on the TwinSystem. The routine is expected to return by executing a JMP Ioret. All registers except SP may be altered, as the environment of the interrupted program is on the stack. PVEC is initialized in the Exec to point to Iexec, which runs the Exec overlay as the result of a ^Y interrupt. If the action of ^Y is masked by EIC in EFLG1, DONT, or GFLOCK, SBRK will still be set nonzero. For example, BASIC points PVEC at Ioret, and checks the state of SBRK at the top of its interpretation loop to see if a ^Y has been hit.

When using PVEC or UVEC in a program, it is a good practice to save the old values of these vectors and restore them upon exiting the program. The user should not assume that PVEC or UVEC are initialized, or that SBRK or UBRK contain zero.

Symbol name: **UVEC**

Single value: 2D95
Twin value: 2E89

See Also: UCHR, UBRK, DONT, GFLOCK

Description:

UVEC contains the address of the routine to enter as the result of receiving the character contained in UCHR at the keyboard interrupt level. This action is masked by DONT and GFLOCK on the TwinSystem. The routine is expected to return by executing a JMP Ioret. All registers except SP may be altered, as the environment of the interrupted program is on the stack. UVEC is initialized in the Exec to point to Ioret. If the action is masked by DONT, or GFLOCK, UBRK will still be set nonzero.

When using PVEC or UVEC in a program, it is a good practice to save the old values of these vectors and restore them upon exiting the program. The user should not assume that PVEC or UVEC are initialized, or that SBRK or UBRK contain zero.

Symbol name: **UBRK**

Single value: 2D97
Twin value: 2E8B

See Also: UVEC, UCHR, DONT, GFLOCK

Description:

UBRK is a single byte flag set nonzero at the keyboard interrupt level when the character in UCHR is recognized. It is the user's responsibility to clear UBRK. Note that UBRK is set nonzero even if the action of UVEC is masked by DONT.

Symbol name: **SCHR**

Single value: 2D98
Twin value: 2E8C

See Also: DONT, GFLOCK, Fpanel

Description:

SCHR contains the character that will cause activation of the front panel code when the character is recognized at the keyboard interrupt level. It is set to zero by the Exec DISABLE command, and in the boot process. It is set to ^Z by the Exec ENABLE command. The contents of SCHR is used throughout the system to determine if the system is in ENABLEd or DISABLEd mode.

Symbol name: **UCHR**

Single value: 2D99
Twin value: 2E8D

See Also: UVEC, UBRK, DONT, GFLOCK, KBMODE1, KBMODE1

Description:

UCHR contains the character that will cause the program to interrupt to the routine pointed at by UVEC when recognized at the keyboard interrupt level. Additionally, UBRK will be set nonzero when this character is recognized. Good programming practice and common sense dictate that UVEC be set up before UCHR is initialized.

At the keyboard interrupt level when checking for an interrupt character, the system first checks for ^Y. Then the character is compared to SCHR, the system enable character. After this check, the character is compared to UCHR. Additionally in the TwinSystem, if single keyboard mode is enabled (SKM, see KBMODE1 and KBMODE2), checks for function keys I and II are made prior to checking for ^Y. This means that UCHR should not be set to ^Y, the contents of SCHR (usually ^Z), or in the TwinSystem, function codes I or II. If UCHR is set to one of these values, that character will most likely never be seen, as it is intercepted before the test for UCHR is made.

Symbol name: **DONT**

Single value: 2D9Ø
Twin value: 2E8E

See Also: PVEC, SBRK, UVEC, UCHR, UBRK, SCHR, Dio

Description:

DONT is the single byte interlock set and reset by Dio that disables action by interrupt characters ^Y and the contents of UCHR and SCHR. It is set nonzero at the start of an I/O operation and cleared when the operation completes. Program interruptions may be blocked by non-I/O programs by setting DONT nonzero. If a program doing I/O wishes to block out interruptions, it should point PVEC and UVEC at Ioret, and set SCHR to zero. Programs should not disable interrupts by using a DI (disable interrupts) instruction for other than very short (less than .Ø5 seconds) critical sections, as this severely affects overall performance, especially typeahead and printer buffering, and is very detrimental on the TwinSystem.

Symbol name: **VERLOC**

Single value: 0439

Description:

VERLOC is the location in ROM of the ROM version number. It is not included in the TwinSystem, as the TwinSystem will not boot on other than version 81 ROMS.

Symbol name: **XTIMER**

Twin value: 0C00

Description:

XTIMER is the internal name in the Twin for the clock area in CPU board RAM. Separate copies of TIMER are kept for each user in the Twin.

Note: The following three symbols are used in the operation of the 5" DSDD disk controller. See the hardware reference manual for the 5" DSDD disk controller for more detail. The user is cautioned against modifying these cells while the system is in operation as it may cause system failure.

Symbol name: **MemAdd**

Single value: ØC49

Description:

MemAdd is a temporary used by the DDSD 5" controller code to hold the memory address involved in the data transfer. It should not be modified by the user.

Symbol name: **Pages1**

Single value: ØC4B

Description:

Pages1 is a temporary used by the DSDD 5" controller code to hold the number of pages remaining in this I/O transfer. It should not be modified by the user.

Symbol name: **Command**

Single value: ØC4C

Description:

Command is a temporary used by the DSDD 5" controller code to hold the current I/O command. It should not be modified by the user.

Symbol name: **BRG**

Twin value: ØC6Ø

See Also: BRGEN, Giveup, PHANTOM

Description:

BRG is used to hold the contents of the CPU board baud rate generator latch, since this latch on the CPU is not readable. The bits defined by PHANTOM determine if this user is named user 1 or user 2. The code in the Exec that does this for enabled mode prompting is:

```
        IF      USERS=2         ; only for twin
        LDA     BRG
        ANI     PHANTOM         ; who are we?
        MVI     A,'1'           ; assume user 1
        JZ      Erda
        INR     A               ; must be user 2!
Erda    CALL    WH1
```

Symbol name: **WHICH**

Twin value: ØC61

Description:

WHICH was used in the Twin to point to the current user. This cell is still used by some internal functions, but its contents cannot and should not be depended on.

Symbol name: **IOIP**

Twin value: ØC62

Description:

IOIP was used by the Twin to mark I/O in progress for each user. While it is no longer used for that purpose, it is still used by some system functions, and as such should not be used.

Symbol name: **VCBTAB**

Value: ØC63

Description:

VCBTAB contains the address of the active volume control block tables that define disks to the I/O system. This location should not be modified by the user.

Symbol name: **DRVADTAB**

Value: ØC7E

Description:

DRVADTAB contains the address of the driver address table, used by the Volume Manager to locate the physical device drivers. This location should not be modified by the user.

Symbol name: **DioA**

Value: ØC66

Description:

DioA is a temporary used in Dio processing to hold the contents of the A register. It should not be modified by the user.

Symbol name: **DioHL**

Value: ØC67

Description:

DioHL is a temporary used in Dio processing to hold the contents of HL. It should not be modified by the user.

Symbol name: **DioDrv**

Value: ØC69

Description:

DioDrv is a temporary used in Dio processing to hold the address of the disk driver to call when access to the attached controller has been granted. It should not be modified by the user.

Symbol name: **DioDn**

Value: ØC6B

Description:

DioDn is a temporary used in Dio processing to hold the translated drive number for passing to the driver routine. It should not be modified by the user.

Symbol name: **DioBsy**

Value: ØC6C

Description:

DioBsy is a temporary used in Dio processing to hold the address of the controller semaphore for this drive. It should not be modified by the user.

Symbol name: **BUSIES**

Value: ØC6E

Description:

BUSIES is the start of a 16 byte area used for controller semaphores by the Dio code. This area should not be modified by the user.

Symbol name: **STACK**

Single value: 1ØØØ

Description:

STACK is the initial stack pointer value loaded during system boot. It should not be used to reset the stack in the Twin! If the stack must be reset in the TwinSystem, it should be reset to the value present at the start of program execution!

Symbol name: **SCREEN**

Single value: 1800

See Also: SCRHM

Description:

SCREEN is the address of the video display in memory. While this address is accurate, a better technique is to use the contents of SCRHM as the high order byte of the video display.

Symbol name: **OVRLY**

Value: 2000

See Also: Ovrto, Gover, OVENT, overlay

Description:

OVRLY is the start of the 2K byte system overlay area. Users wishing to make use of this area should read and understand the section of the System Programmer's Guide relating to overlays.

Symbol name: **OVENT**

Value: 2004

See Also: Ovrto, Gover, OVRLY, overlay

Description:

OVENT is the start address for code in overlays. All overlays must either have code or a JMP instruction at this location, as that is where they are entered by the overlay manager.

Symbol name: **SBUF1**

Value: 2800

Description:

SBUF1 is the address of the 1K byte file directory. The contents of this area is detailed in the separate section on the file system.

Symbol name: **SBUF2**
Single value: 2900

Symbol name: **SBUF3**
Single value: 2A00

Symbol name: **SBUF4**
Single value: 2B00

Description:

SBUF2, SBUF3, and SBUF4 are single user symbols for the second, third, and fourth pages of the file directory area. These symbols are not used, but the entire 1K directory area is. These symbols are excess baggage, and maybe we'll delete them from the file someday.

Symbol name: **MUNGP**

Twin value: 2DØØ

Description:

MUNGP is the address of a 64 byte system scratch area. It should not be modified by user programs.

Symbol name: **BOOTVOL**

Twin value: 2D92

Description:

BOOTVOL in the TwinSystem holds the drive number of the system disk. It is used throughout the Twin in checking the system drive, such as in the INIT code to detect attempts at initializing the system disk. Note that BOOTVOL is in per-user memory; each user has a separate copy of BOOTVOL as each user may have a different disk drive as system residence.

Symbol name: **USP**

Twin value: 2EØØ

See Also: Giveup

Description:

USP holds the user's SP while the other user is running. This location ESPECIALLY should not be modified by the user!

Symbol name: **USRNAME**

Twin value: 2EEE

Description:

USRNAME is a 16 byte area that was originally intended to hold the user name in the TwinSystem. Parts of this area are used in the Twin, so this area should not be modified by the user program.

Symbol name: **MTO**

Single value: 2DA2

Description:

MTO is the motor time-out counter for the 5" SSSD disk controller code. As 5" SSSD disks are not supported on the Twin, this byte is unused.

Symbol name: **SINT**

Single value: 2DB3

Description:

SINT is the on-sector bailout vector used by the 5" SSSD disk controller code. As 5" SSSD disks are not supported by the Twin, this word is unused.

Symbol name: **NDRIVES**

Single value: 2D9F
Twin value: 2EA9

See Also: DEFPATH

Description:

NDRIVES was used in earlier versions (pre Exec/78) to hold the drive number resulting from a <?> search. It is unused, as its function has been expanded and replaced by DEFPATH.

Symbol name: **TIMER**

Single value: ØCØØ
Twin value: 2EØ2

See Also: WAKEUP

Description:

TIMER is the address of a four byte area that is ~~decremented~~ incremented on each real time clock tick (60Hz line clock). Note that BOTH user's clocks are decremented in the Twin even if one user has the processor locked through use of the Lock service, or the user is blocked waiting for a system semaphore.

Symbol name: **UTIME**

Twin value: 2EØ2

Description:

UTIME is the same thing as TIMER. Originally there were two separate timer cells per user in the Twin.

Symbol name: **MEMTOP**

Single value: 2D8Ø
Twin value: 2EBB

See Also: Runr

Description:

MEMTOP contains the 16 bit address of the last good location of memory. It is set as part of the boot process. Programs should not assume that the low order byte of the last good address is FF. Also, programs that alter MEMTOP should return it to its previous contents when the program exits. The Exec DISPLAY command displays the contents of MEMTOP. Note that the Runr service does not check MEMTOP when loading a program into memory. Thus, Runr may overwrite existing information, or load a program into nonexistent memory.

Symbol name: **EXECSP**

Single value: 2DAF
Twin value: 2EBD

Description:

EXECSP is used to hold the 16 bit value of the stack pointer. This is used by the Exec in preventing stack overflow.

Symbol name: **FILE**

Single value: 2DCB
Twin value: 2EBF

Description:

FILE is a 44 byte area used for holding file control blocks. It is used by system functions, but may be used by the user program.

Symbol name: **DirAddr**

Single value: 2EØ2
Twin value: 2EA7

Description:

DirAddr contains the 16 bit sector address of the directory currently in SBUF1. It is set by Gfid, and should not be modified by the user program.

Symbol name: **SYSRES**

Single value: 2D92
Twin value: 2EAA

Description:

SYSRES is the byte containing the drive number of the system drive. Since it is in per-user memory in the TwinSystem, each user may be running off a different system drive, as long as both users are running exactly the same version of the TwinSystem. This location should not be modified by the user program.

Symbol name: **NFDIR**

Single value: 2DAØ
Twin value: 2EAB

See Also: NFCK, DirAddr, SBUF1, Gfid, Look

Description:

NFDIR is the byte containing the drive number of the directory currently in SBUF1. Both ØØ and FF in NFDIR indicate the contents of SBUF1, DirAddr, and NFCK are invalid. If the 8ØH bit of NFDIR is set, then the directory in SBUF1 is a subdirectory, and its disk address is contained in DirAddr. This byte is set and updated by Look and Gfid.

Symbol name: **NFCK**

Single value: 2DA1
Twin value: 2EAC

See Also: SBUF1, Ckdr

Description:

NFCK contains the single byte checksum of the directory in SBUF1 as computed by Ckdr. This byte is expected to match the first byte of the directory area. It is updated by Look and Gfid.

Symbol name: **MUNG1**
Single value: 2DA7
Twin value: 2E9F

Symbol name: **MUNG2**
Single value: 2DA9
Twin value: 2EA1

Symbol name: **MUNG3**
Single value: 2DAB
Twin value: 2EA3

Symbol name: **MUNG4**
Single value: 2DAD
Twin value: 2EA5

Description:

MUNG1 through MUNG4 are scratch locations used internally in the system. They should not be altered by user programs.

Symbol name: **CMND**

Single value: 2D4Ø
Twin value: 2E3C

See Also: CMPTR, ONCE

Description:

CMND is the 64 byte Exec command buffer. Before reading commands, the Exec fills the buffer with carriage returns (ØD). CMPTR, the Exec's command pointer, usually points into CMND for overlays and other system components to pick up arguments and filenames.

Note that when the Exec is invoked, if the 4ØH bit in ONCE is not set, the Exec assumes that a command is already in CMND for it to process, and so does not prompt or read from the user. This is the mechanism used to start the INITIAL (or INITIAL1 or INITIAL2) file, and is also used by CØ2 BASIC to pass an exit command string to the Exec. So, for a program to force the Exec to execute a command, the command text delimited by a carriage return should be placed in CMND, and the 4ØH bit (and ONLY that bit) turned off in ONCE. When the Exec gains control, it will interpret and perform that command.

Symbol name: **CMPTR**

Single value: 2DC7
Twin value: 2E7C

See Also: CMND

Description:

CMPTR is the 16 bit pointer set by the Exec to point after the command name in the command line. It usually points into CMND. CMPTR is used by system functions and programs such as BASIC to scan for filenames and flags on the command line.

Symbol name: **WAKEUP**

Single value: ØC1A
Twin value: 2EØ6

See Also: TIMER

Description:

WAKEUP contains the address of the routine to call when the four byte TIMER area is ~~decremented~~ incremented to zero. On the single user system, this routine is JMP'd to at the interrupt level; it is expected (hoped) to return by JMPing to Ioret. On the TwinSystem, an interrupt environment is built onto the user's stack, and the routine pointed to by WAKEUP will be executed the next time that user is run. As with the single user system, when the WAKEUP routine completes, it should JMP to Ioret.

Symbol name: **SCEND**

Single value: ØC1E
Twin value: 2EØ8

Description:

SCEND contains the high byte of the video screen ending address. It is modified by some programs to "protect" four line chunks of the screen. It is also ignored by the Editor and the graphics functions in BASIC, so care must be exercised in its manipulation.

Symbol name: **SCRHM**

Single value: ØC1F
Twin value: 2EØ9

Description:

SCRHM contains the high byte of the video display starting address. It may be modified by user programs, but with caution. While the Editor and the graphics functions in BASIC use the contents of this cell to determine the starting address of the video display, both assume that the video display always has 1K bytes available starting at this address. Catastrophic system failure can result from incrementing this address to protect the top four lines of the display screen and then using the PLOT and DRAW functions in BASIC; they will alter memory used as parameter passing and control for the DSDD 5" and MS disk controllers!

Symbol name: **POS**

Single value: ØCØE
Twin value: 2EØA

See Also: Vti, WH1

Description:

POS points to the cursor on the video screen. It is updated by the video display driver, Vti, which is usually called via WH1.

Symbol name: **PATH**

Single value: 2EØ4
Twin value: 2DØ4

See Also: Gfid, NFDIR, SBUF1

Description:

PATH is the 64 (decimal) byte pathname holder used by Gfid to determine the path to the current subdirectory in SBUF1 if the 8ØH bit is set in NFDIR. It should not be modified by user programs.

Symbol name: **DEFPATH**

Single value: 2E27
Twin value: 2DØØ

See Also: Gfid

Description:

DEFPATH is the 43 (decimal) byte area used to hold the default path associated with "#". It is set by the Exec "#" command, as well as by <?< searches in the Exec and Gfid. The default path specification in DEFPATH is terminated by an 8ØH byte. An opening bracket, either "<" or ">" is deleted in the processing of the default path specifier.

Symbol name: **OVMEM**

Single value: 2E53

Description:

OVMEM marks the start of the overlay scratch area in the single user system. From this point to 2F00H may be used by overlays, but is not protected by other overlays. User written overlays may make use of this area but cannot expect it to go unmodified if system overlays are invoked; this includes the Exec.

Symbol name: **LOCK**

Twin value: 2E8F

See Also: Vti, Lock, lock, Unlock, unlock

Description:

LOCK is a single byte flag used to inhibit time slicing in the TwinSystem. Normally, the TwinSystem switches between users 60 times a second as the result of real time clock interrupts. If the contents of LOCK are nonzero, this switching is inhibited. While switching between users is inhibited, keyboard characters are still accepted from the other user's keyboard and placed in the typeahead buffer. LOCK is set by Vti, the screen driver, and by the Editor to prevent switching while screen updating and screen scrolling is in progress. If used, it should be set and reset using the lock and unlock macros, or by calling the Lock and Unlock system services. It should be used only for short periods of time, as its effect on the system can be quite noticeable.

Symbol name: **NFA**

Single value: 2E00
Twin value: 2EAD

Description:

NFA in the single user system contains a disk address used by Gfid, usually in the creation of subdirectories. It should not be modified by the user. In the TwinSystem, this is in Gfid local storage and is not globally defined.

Symbol name: **ONCE**

Single value: 2DC5
Twin value: 2EAF

See Also: CMND, Boot process description

Description:

ONCE is the flag byte used to sequence the system boot process. The 40H bit in ONCE tells the Exec if it is to read a command into CMND or parse the string already there. See CMND for information on this interaction. The remaining bits of ONCE should not be modified by the user.

Symbol name: **EFLG1**

Single value: 2DC9
Twin value: 2EB0

See Also: EIC, EERR, DBARF, SBRK

Description:

EFLG1 is a flag byte kept by the Exec. The 80H bit is EIC, Exec in control. This is set while the Exec is running, and disables ^Y. See descriptions of EIC and SBRK for more information. The 20H bit is DBARF, which inhibits the call to Killi in Emsg when errors are reported; see DBARF for more information. The 40H bit is EERR which tells the Exec it has an error code in ERROR to process; see EERR, ERROR, and Err for more information. The remaining bits (1FH) are used for default device number and other tags used in command processing.

Symbol name: **EFLG2**

Single value: 2DCA
Twin value: 2EB1

Description:

EFLG2 is a byte of flags used by the Exec in scanning and processing the user command line. The 10H bit, if set, notes that there is something loaded into memory. This is the bit that the Exec uses in disabled mode to determine what to do with the REENTER command. If the 10H bit is set in EFLG2, REENTER is allowed in disabled mode. If the 10H bit is not set, the Exec doesn't think there is anything in user memory to run, and it doesn't allow the REENTER command. If a user program does not wish to honor the REENTER command in disabled mode, it may clear the 10H bit in EFLG2, although placing a RET instruction at USER+3 would be more effective.

Symbol name: **JOBST**

Single value: 2D9E
Twin value: 2EB2

Symbol name: **LUSER**

Single value: 2DC6
Twin value: 2EB3

Description:

JOBSTS and LUSER are each single byte flags used internally in the Exec in command processing. While the Exec is not running, they may be used by the program. Programs should not expect them to be undisturbed when the Exec runs, however.

Symbol name: **BUGS**

Single value: 2DFC
Twin value: 2EB4

Description:

BUGS is a three byte area used to count up soft 102, 103, and 104 errors from the 5" SSSD disk controller. These counts each have a maximum value of FF; that is, they will not be incremented from FF to 00. In systems prior to Exec/90, these counters were displayed by the DISPLAY and SQUEAL commands. Due to lack of room in the Exec, the SQUEAL command and the display of these cells is no longer supported. Additionally, these cells are not clocked by errors on 8" drives or 5" DSDD drives.

Symbol name: **ERROR**

Single value: 2D9A
Twin value: 2EB7

See Also: EERR, LERR, Err

Description:

ERROR is the 16 bit error code last reported to the system. ERROR is updated either by the Emsg overlay, called with the error code in DE, or by a JMP Err with the error code in DE.

Symbol name: **LERR**

Single value: 2D9C
Twin value: 2EB9

See Also: EERR, ERROR, Err

Description:

LERR is a 16 bit error code. Depending on how the last error was reported, it may contain the same code as ERROR, or it may contain the code of the error reported previous to the one in ERROR. It is updated by Err in the root and by Emsg.

Symbol name: **SRA7**

Single value: 0C1C
Twin value: 2EEC

Description:

SRA7 contains the address of the service routine for RST 7 and single step interrupts. In the TwinSystem, it is in per user memory so that each user may have separate control over this interrupt for such uses as debugging. This interrupt is generated either by triggering the single step logic, or by executing a RST 7 instruction (FFH). This FF pattern will occur when an instruction is fetched from nonexistent memory.

SYSTEM SERVICE VECTORS AND ROUTINES

The following section details the resident services provided by the single and Twin systems. Some services are only provided in one version of the system; some are provided in both. Some services are supplied in both systems but with different spellings (DEOUT and Deout).

REGISTERS - For each service, entry and exit registers are described where applicable, as is the state of the interrupt system on return. If registers are not specified in the description, it is safe to assume that they are neither used nor modified. If a register on exit is described as containing "junk", its contents should not be depended on- even if they seem useful; as there is NO promise made that the next version of the system will return the same junk!

INTERRUPTS - For each service, the state of the interrupt system when the service completes is given. If nothing is stated about interrupts in the description, interrupts are not modified during execution of the service. Depending on the path taken in the service, interrupts may or may not be altered. For example in WHØ, if the character returned is from the keyboard, interrupts will always be enabled on return. If the character was taken from the command file buffer and no read was required, the interrupts are unchanged from their state on entry to WHØ. As part of good design and programming practice, interrupts should be disabled only when absolutely necessary, and for the minimal number of instructions required to complete a critical section. The usual reason for disabling interrupts is to protect data areas that may be altered by code executing at the interrupt level, or to block out interrupts themselves.

As with any system functions, care should be exercised in debugging and experimenting with assembly language programs, especially in the TwinSystem. Causing a system failure while the other user is in the process of packing a disk may be hazardous to your health!

SYMBOL	PAGE	Single	Twin
Byte	151		E05A
Ckdr	119	0433	E02D
Cold	115		E000
DEOUT	151	03D1	
Deout	151	03D1	E045
Devlock	157		E06F
Dhalt	121	0409	
Dio	122	0406	2E30
Enter	155		E04E
Err	130	040F	E00C
Fdfp	163		E075
Flip	118	042D	E02A
Flipem	143		E039
Flush	117	041E	E01B
Fold	118	042A	E027
Gdfp	163		E072
Gfid	148		E042
Giveup	143		E03C
gLook	121		E07B
Gover	131	0415	E012
Iexec	140	0436	E030
Ioret	160	0064	E066
Killi	117	041B	E018
Leave	156		E051
Lock	153		E048
Look	134	0421	E01E
Mfos	120		E063
Move	161		E06C
Moven	161		E069
Msg	116	040C	E006
Mtos	120		E060
Ovrto	131	0412	E00F
Pmsg	116		E009
Print	142		E036
Rlgc	121	0430	

SYMBOL	PAGE	Single	Twin
Rlwe	139	0427	E024
Rtn	133	0418	E015
Runr	137	0424	E021
Show	141		E033
SUWH0	128		0C20
SUWH1	128		0C24
SUWH2	129		0C28
SUWH3	129		0C2C
SUWH4	129		0C30
SUWH5	129		0C34
SUWH6	129		0C38
SUWH7	129		0C3C
SUWH9	129		0C44
Ticker	145		E03F
Unlock	154		E04B
Vmgr	162		E054
Vti	111		E057
Warm	115	0403	E003
WH0	109	0C20	2E0C
WH1	111	0C24	2E10
WH2	113	0C28	2E14
WH3	113	0C2C	2E18
WH4	113	0C30	2E1C
WH5	114	0C34	2E20
WH6	114	0C38	2E24
WH7	114	0C3C	2E28
WH9	122	0C44	2E30

Symbol name: **WHØ**

Single value: ØC2Ø
Twin value: 2EØC

Entry: WHØ takes no inputs

Exit:
A: Character from keyboard or command file
INT: not modified/enabled

See Also: KBEX, Flip, Fold, KBUF, KBIP, KBIG, CBUF, CMDF, CMDD, CMDN, CMDP, CMDA, Killi, Flush, DBARF

Description:

WHØ is called to return a character from the user. The character is returned either from the keyboard ring buffer (see KBUF, KBIP, KBIG) or from the command file buffer (see CBUF, CMDF). A request to read a character past the end of a currently active command file causes the system to switch automatically back to the keyboard for input. The character returned in A is not echoed to the screen; this must be done by the program. WHØ is initialized by the boot process to point to Cin (label not defined in the single user system).

If command files are not active, and a character is not present in the keyboard buffer, the system waits until a character is available. In the single user system, the processor enables interrupts, halts to wait for an interrupt, and then checks again for a character. In the TwinSystem, Giveup is called to give up the processor (see description of KBIP for this code).

If the character returned is from the keyboard buffer, or a disk read was required to bring in command file data, the interrupts are returned enabled. If the character is returned from the command file buffer (CBUF) and a read is not required, the interrupt system is unaltered. Any errors in attempting to read from the command file cause control to be transferred to Err, essentially aborting the program; restart may be difficult.

The character removed from either the command file buffer or the keyboard buffer is passed through the routine connected to KBEX; this is commonly a null routine, but may also be either Flip or Fold, or a user specified routine.

As the result of invoking Killi or Flush, the contents of

the keyboard typeahead buffer may be flushed. Killi also aborts command files if in progress.

Symbol name: **WH1**

Single value: ØC24
Twin value: 2E1Ø

Symbol name: **Vti**
Twin value: EØ57

Entry:
A: character to be displayed on screen

Exit: All registers and interrupts unchanged.

See Also: SCRHM, SCEND, POS, Vti, Lock, Unlock

Description:

WH1 is called to display a character on the video screen. It is initialized by the system boot process to point to the video display driver (Vti in the TwinSystem, no separate label for the single user system). Certain character values have special effects when displayed. They are:

Code	Name - Action
Ø9H	HT - Tab cursor
ØBH	VT - Move cursor to top of screen
ØCH	FF - Clear screen and move cursor to top
ØDH	CR - Move cursor to start of next line
18H	CAN - Erase remainder of line
7FH	DEL - Move cursor left one position

Values less than 2ØH not appearing in the above list are ignored. Note that displaying a character may cause the screen to scroll up a line, destroying the information in the top line of the screen.

The display driver finds the starting and ending page address of the screen by examining SCRHM and SCEND. On exit from the display driver, POS contains the address of the cursor within the video display.

In the TwinSystem, and in ROMs version 81 and later, the screen driver does not alter the state of the interrupt system. Earlier versions of the ROMs always enabled interrupts.

In the TwinSystem, erasing lines, clearing the screen, or scrolling the screen is done with the task protected against

slicing by first calling the Lock service. This prevents the screen from being left partially updated. The Unlock service is called as part of exiting the screen driver (Vti) and WH1, even if Lock was not called; this means that a program wishing to remain locked against slicing may not use any service that may call WH1.

Symbol name: **WH2**

Single value: ØC28
Twin value: 2E14

Description:

WH2 has no specific meaning in the system. In the TwinSystem, it is initialized to STC/RET. It is not initialized on single user systems.

Symbol name: **WH3**

Single value: ØC2C
Twin value: 2E18

Description:

WH3 has no specific meaning in the system. In the TwinSystem, it is initialized to STC/RET. It is not initialized on single user systems.

Symbol name: **WH4**

Single value: ØC3Ø
Twin value: 2E1C

Description:

WH4 has no specific meaning in the system. In the TwinSystem, it is initialized to STC/RET. It is not initialized on single user systems.

Symbol name: **WH5**
Single value: ØC34
Twin value: 2E2Ø

Symbol name: **WH6**
Single value: ØC38
Twin value: 2E24

Symbol name: **WH7**
Single value: ØC3C
Twin value: 2E28

Description:

WH5, WH6, and WH7 are used with the printer and printer driver. See the section on the printer driver for details. In both the single and Twin systems, if the Prnt overlay is not present, these wormholes are initialized with a STC/RET pair.

Symbol name: **Cold**

Twin value: E000

Description:

Cold is the cold start location in the TwinSystem. Once the Twin has completed its boot process, it is turned into a JMP 0000, which if executed, will cold start the system. It should not be called, since it is the equivalent of pushing the Load button!

Symbol name: **Warm**

Single value: 0403
Twin value: E003

Entry: No inputs

Exit: Does not return

Description:

Warm is called to warm start a user. The stack pointer is reinitialized and the Exec reloaded by calling Gover. Here is the coding for Warm from the TwinSystem resident:

```
Warm    LXI     SP,0E000H ; reset stack
        CALL    Gover
        DB      'Exec'    ; Go run the Exec
        JMP     Warm      ; do that again.
```

Symbol name: **Msg**

Single value: Ø40C
Twin value: EØØ6

Entry:
HL: Address of text string delimited by ØØ byte

Exit:
HL: Points to ØØ byte
A: ØØ

Description:

Msg displays the message pointed to by HL on the screen by calling WH1 with each character and incrementing HL until a ØØ byte is encountered. Here is the coding for Msg:

```
Msg     MOV     A,M     ; get chr
        ORA     A       ; done yet?
        RZ              ; return if so
        CALL    WH1     ; display
        INX     H
        JMP     Msg     ; do another one.
```

Symbol name: **Pmsg**
Twin value: EØØ9

Entry:
HL: Address of text string delimited by ØØ byte

Exit:
HL: Points to ØØ byte
A: ØØ

Description:

Pmsg prints the message pointed to by HL on the printer by calling WH7 with each character and incrementing HL until a ØØ byte is encountered. Here is the coding for Pmsg:

```
Pmsg    MOV     A,M     ; get chr
        ORA     A       ; done yet?
        RZ              ; return if so
        CALL    WH7     ; display
        INX     H
        JMP     Pmsg    ; do another one.
```

Symbol name: **Killi**

Single value: Ø41B
Twin value: EØ18

Entry: No inputs

Exit:
A: Junk
PSW: Junk
INT: Disabled

See Also: CMDF, DBARF, Err, SBRK, UBRK
Devlock, CMDD, cmdf (TwinSystem only)

Symbol name: **Flush**

Single value: Ø41E
Twin value: EØ1B

Entry: No inputs

Exit:
INT: Disabled

See Also: Interrupt Character Processing
KBIG, KBIP, KBUF

Description:

Killi is called to kill typeahead and abort command files if in progress. Flush is called to just kill typeahead; it is used by Killi after command files have been taken care of. When Killi is called, it checks CMDF to see if command files are in progress.If the contents of CMDF are nonzero, the text "(Cmdf Abort)" is displayed on the screen by calling Msg, and CMDF is set to ØØ. On the TwinSystem, permanent command file read access to the command file device specified by CMDD is released.

If CMDF contained zero, or if Flush was called, typeahead is cancelled by initializing KBIP and KBIG to KBUF+63 (decimal). This processing is done with the interrupts DISABLED, and both Killi and Flush return with interrupts DISABLED.

Flush is also called as part of the system initialization sequence in both single and Twin systems to initialize KBIP and KBIG pointers.

Symbol name: **Fold**

Single value: Ø42A
Twin value: EØ27

Entry:
A: Ascii character

Exit:
A: Upper case Ascii character
PSW: See code for Flip and Fold

Symbol name: **Flip**

Single value: Ø42D
Twin value: EØ2A

Entry:
A: Ascii Character

Exit:
A: Flipped alphabetic character

See Also: KBEX

Description:

Fold and Flip are post processing routines hooked up to KBEX, usually as the result of the Exec Fold and Flip commands. Fold causes all lower case alphabetic characters (a-z) to be folded to upper case (A-Z). Flip causes the case of alphabetic characters only to be inverted, for example a lower case "a" becomes "A", and "A" becomes "a". Here is the coding for Flip and Fold:

```
Flip    CPI     'A'
        RC              ; split if not interesting
        CPI     'Z'+1
        JM      Flil    ; jmp/flop upper to lower
Fold    CPI     'a'
        RC              ; split if not a-z, A-Z
        CPI     'z'+1
        RNC             ; split- not interesting
; Flip upper and lower case
Flil    XRI     20H     ; do the dirty work.
        RET             ; and split.
```

Symbol name: **Ckdr**

Single value: Ø433
Twin value: E Ø2D

Entry: No inputs

Exit:
A: Single byte checksum of directory area

See Also: SBUF1, NFCK, Look, Gfid

Description:

Ckdr is called to compute the checksum of the directory currently in SBUF1. This checksum is used to validate the disk directory. Here is the coding for Ckdr:

```
Ckdr    PUSH    H
        PUSH    D               ; save DE and HL
        LXI     H,SBUF1+1       ; start here
        XRA     A               ; with ØØ
Ckdr1   ADD     M
        INX     H
        MOV     D,A
        MOV     A,H
        CPI     CBUF/256        ; <-- watch this one!
        MOV     A,D
        JNZ     Ckdr1
        POP     D
        POP     H
        RET
```

Symbol name: **Mtos**

Twin value: E060

Entry:

HL:	Destination addr in other user
DE:	Source addr in this user
BC:	# bytes to move (positive)

Exit:

HL:	Dest addr + count + 1
DE:	Source addr + count + 1
BC:	0000
A:	00
INT:	Disabled

Symbol name: **Mfos**

Twin value: E063

Entry:

HL:	Destination address in this user
DE:	Source address in other user
BC:	# bytes to move (positive)

Exit:

HL:	Dest addr + count + 1
DE:	Source addr + count + 1
BC:	0000
A:	00
INT:	Disabled

Description:

Mtos and Mfos are move to other space and move from other space. In the TwinSystem, they are used in moving blocks of memory from one user space to the other. Interrupts are disabled for the duration of the transfer between users, and interrupts are returned disabled. Extreme care should be used with these services.

Symbol name: **Dhalt**

Single value: Ø4Ø9

Description:

Dhalt was provided in old 5" SSSD systems for shutting down the 5" disk drives. It is no longer supported and should not be called.

Symbol name: **Rlgc**

Single value: Ø43Ø

Description:

Rlgc is an internal entry into Rlwe defined in single user systems. It should not be used by user programs, and is not used by the system. The definition is not present in the TwinSystem.

Symbol name: **gLook**

Twin value: EØ7B

Description:

gLook is an internal entry to the Look process used by TwinSystem initialization. It should not be called by the user, as it bypasses file system interlocks.

Symbol name: **WH9**

Single value: ØC44
Twin value: 2E3Ø

Symbol name: **Dio**

Single value: Ø4Ø6
Twin value: 2E3Ø

Entry:

HL:	Disk address
DE:	Memory address
BC:	B: Command: Ø=write, 1=read, 5=disk size C: Unit number: 1-F
A:	Number of sectors, 1 <= # <= 255 (decimal)
PSW:	unused

Exit:

HL:	Disk size if B=5 on entry, else junk
DE:	If carry bit set in PSW, error code If carry bit not set, junk
BC:	junk
A:	junk
PSW:	Carry bit set if error, clear otherwise All other flags unknown
INT:	Enabled/unchanged

See Also: DONT, Devlock, MEMTOP, Giveup

Error conditions:

The error codes reported by Dio are reported in DE, with Carry set in PSW:

Ø1Ø1 Bad parameters passed to Dio: A=Ø; invalid command in B; invalid drive in C; disk address in HL, or disk address in HL plus number of sectors to be transferred is greater than the number of sectors on the disk.

Ø1Ø2 The sector preamble is bad. This indicates a non-initialized disk or a serious error with the hardware or with the contents of the disk.

Ø1Ø3 Incorrect sector checksum on data read from the disk.

Ø1Ø4 A verify operation finds that the contents of memory and the contents of the specified sector (or sectors) do not match.

Ø1Ø5 An attempt was made to write on a write-protected disk. No data will be transferred to the disk.

Ø1Ø6 This error occurs when the system does not receive sector interrupts from the selected drive. Several conditions may cause this: no drive on the system with the specified drive number (e.g., you tried to access drive 9 on a one drive system); there is no disk in the drive specified; the door on the drive specified is open; the disk is inserted wrong.

Ø1Ø7 No controller present for that drive. An attempt was made to access a DSDD 5" drive with no DSDD controller in the system, or an MS drive with no MS controller in the system. Note that it takes approximately 3 seconds to detect this error, during which the system cannot be interrupted.

Ø1Ø8 Transfer error. An error occurred in the transfer of data from the DSDD 5" controller board and main memory. This may be caused by bad main memory or a problem with the 5" DSDD controller.

Ø1Ø9 No such drive. The drive requested does not exist on either the 5" DSDD or MS controllers.

Ø1ØB Seek error. This error is reported by the DSDD 5" controller in attempting to position to the requested track on the disk. It indicates an improperly initialized disk or a problem with the DSDD controller.

Ø18X (Ø18Ø - Ø18F) These error codes indicate that the controller in question has failed. Detailed information on these error codes is contained in the MS and DSDD controller Theory of Operations manuals.

Ø1C2 That device is busy. (TwinSystem only) The type of access requested was denied by the device interlock mechanism (see Devlock) because of the operation being done by the other user in the system.

Description:

Dio is the central system service routine for transferring data to and from disk, and verifying those transfers. Call Dio with the register contents outlined above. Think of the disk as a set of sectors; Dio worries about tracks and track position. Each sector contains 256 (decimal) bytes. Each write operation is automatically verified by comparing the data written on the disk with the contents of memory.

Dio does not check MEMTOP to see that the transfer requested is to valid memory; these checks must be made by the user program. This is especially important in the TwinSystem; user programs must be careful not to modify memory above DFFFH.

> NOTE: The Hard Disk volume manager, if present filters all calls to Dio (See section 1Ø, Volume Manager).

Single User Dio

In the single user system, Dio is called through the normal ROM vector (406H). CPU board ROMs version 81 and later vector this through WH9 to support user written disk device drivers. While I/O is in progress, DONT is set nonzero to inhibit the action of interrupt characters (see DONT for details). DONT is cleared on I/O completion. Note that earlier version ROMs may not correctly support the disk size function (B=5); earlier ROMs may return with an error code of 101 (invalid code), or return an incorrect result. The following code is used by the system to return disk size when the ROM version is not known:

```
        LXI     H,1FEEH ; bad disk address
        MVI     B,5     ; disk size function
        MVI     A,1     ; read 1 from drive in C
        LXI     D,0     ; memory address 00
        PUSH    B
        CALL    Dio     ; see what it says
        POP     B       ; preserve BC over call
        JNC     Gotsize ; got size in HL!
        MOV     A,C
        CPI     4       ; small or big drives?
        LXI     H,350   ; small ones are 350 sectors
        JC      Gotsize ; if < 4, small ones.

; For MS drives, we must force the head to load to set the
; single/double side flag correctly.

        LXI     H,1FEEH ; illegal address
        MVI     B,1
        MOV     A,B     ; read 1 sector (or try to!)
        CALL    Dio
        LXI     H,2464  ; single size flag
        LDA     1FEEH   ; single/double flag
        ORA     A
        JNZ     Gotsize ; jmp/single size
        DAD     H       ; double is twice as much.
Gotsize
```

Of course, the above code does not properly handle double side single density 5" drives (not supported by PolyMorphic Systems). The easiest way around having to use code like this is to upgrade the machine to CPU ROMs version 81 or later!

TwinSystem Dio

Disk I/O in the TwinSystem is more complex than in the single user system. Disk I/O follows these basic steps:

1. If the contents of A (# of sectors to transfer) is less than or equal to CHUNK, go to step 4.

2. Save registers on stack; set A to CHUNK size for device to do CHUNK number of sectors. Call step 4; if it returns with C set, unwind stack and return the error code in DE.

3. Restore registers from stack. Subtract CHUNK from A, add CHUNK * 256 to DE, go to step 1.

4. Check for device number in C between 1 and F inclusive, and command in B between 1 and F inclusive. Return error code 101 if either check fails.

5. Call Devlock to get permission to access the device specified in C. If the operation requested is a read, get temporary read access. If the operation is a write, get temporary write access. If the operation is a write to sector 0000, get mung access. If Devlock returns with an error, return that error code. Note that if WAIT mode is enabled (see BHA and Devlock for details), the program may wait at this step until the device is available.

6. Wait for the device semaphore to indicate the device is available.

7. With interrupts disabled, set DONT nonzero and set the device semaphore to indicate the device is busy.

8. Enable interrupts and call the device driver (note that DONT has been set nonzero).

9. With interrupts disabled, clear DONT and the device semaphore; return enabling interrupts through Giveup, giving up the processor, and returning any error code returned by the device driver.

This process shows a number of key features of disk I/O on the TwinSystem. Steps 1 through 3 break up large (> optimum for device) I/O requests into CHUNK sector pieces. Large requests are done CHUNK sectors at a time; when each CHUNK is completed, the device controller is released and the other user allowed to run. Breaking up large transfers in this manner allows one user

to do large operations (such as IMAGE and COPY of large files) while allowing the other user to access that device, or other devices on the same controller. If large operations were not broken up, all devices on the controller would be locked out until the operation completed, which could take a while (such as a 48K byte write to an MS). Eight sectors is optimum on the 8" controller, 20 (a full track) on the 5", and 127 for the hard disk.

The call to Devlock in step 5 provides a built in level of security for device accesses. Note that a write request to a write locked device returns a 01C2, device busy, if WAIT mode is not enabled, and if WAIT mode is enabled, it waits until the write can be done.

The semaphore referred to in step 6 is specified on a per controller basis. This means that on TwinSystems with both 8" and 5" intelligent controllers, I/O operations can be in progress for both users on different drive types at the same time. This provides a substantial performance improvement when the system is on a 5" DSDD drive and main file storage is on 8" MS disks.

When the transfer completes, Dio returns through Giveup to allow the other user to execute. This balances out performance, and since the device semaphore was released, insures the other can use the controller or device if it is waiting for it.

Symbol name: **SUWHØ**
Twin value: ØC2Ø

Symbol name: **SUWH1**
Twin value: ØC24

See Also: WHØ, WH1

Description:

On the TwinSystem, single user wormholes Ø and 1 are connected to their Twin counterparts; this is done primarily so that the Msg service may be used from the CPU ROMs. Programmers should always use the symbols WHØ and WH1.

Symbol name: **SUWH2**
Twin value: ØC28

Symbol name: **SUWH3**
Twin value: ØC2C

Symbol name: **SUWH4**
Twin value: ØC30

Symbol name: **SUWH5**
Twin value: ØC34

Symbol name: **SUWH6**
Twin value: ØC38

Symbol name: **SUWH7**
Twin value: ØC3C

Symbol name: **SUWH9**
Twin value: ØC44

Description:

On the TwinSystem, single user wormholes 2, 3, 4, 5, 6, 7, and 9 are pointed to the single user abort routine. This is done to trap out attempts to run programs assembled for single user systems on the Twin. Since many system addresses have changed between the single and Twin systems, these programs will not execute properly, and may destroy the system. If one of these single user wormholes are called, the message

Single user trap- Reboot!

is displayed on the offending user's screen, and that user is stopped. The other user in the Twin may be unaffected, but the system should be rebooted as soon as it is possible, or convenient, as system memory and pointers may have been damaged before the errant program was trapped.

Symbol name: **Err**

Single value: 040F
Twin value: E00C

Entry:
DE: Error code to process

Exit:
Through Warm

See Also: EERR, EFLG1, Killi, Warm, ERROR, LERR

Description:

Err is called with an error code/subcode pair in DE. The typeahead buffer is flushed, and command file mode is aborted if active. (This is done by calling Killi.)

If the flag bit EERR in EFLG1 is set, we have an error condition arising from an attempt to report a previously reported error: we are in serious trouble, because that flag should have been cleared (by the Exec). The code/subcode in ERROR is displayed on the screen in the form:

(Error xxyy)

where xx is the code in D and yy is the subcode in E. After displaying the error code, the system HALTS. The Emsg overlay is responsible for clearing the EERR bit in EFLG1. User written Emsg handlers must remember to clear this bit, or a system panic halt will result.

If EERR in EFLG1 is not set, Err sets it, so that when Exec begins execution it knows that it has an error to process. Err then stores the code/subcode in the system cell ERROR. We then jump to Warm to warmstart the system. The Exec, after doing its cleanup, will see the EERR flag set in EFLG1 and invoke the system error message handler, Emsg, to process the error code. If a message is present in the error writer, the message will be displayed; if no message is present, the text

?No message found for error xxyy

will be displayed, where xx and yy are the error code and the error subcode contained in DE.

Symbol name: **Ovrto**

Single value: Ø412
Twin value: E ØF

Entry: All registers passed to target overlay

Exit: All registers returned from overlay

Symbol name: **Gover**

Single value: Ø415
Twin value: EØ12

Entry: All registers passed to target overlay

Exit: All registers returned from target overlay

See Also: Runr, Dio, Look, EFLG1, SYSRES, OVRLY, OVENT

Description:

Ovrto and Gover provide the mechanisms for invoking system functions by name and for extending the available system services in a powerful manner. These facilities are the cornerstones on which the System 88 disk operating system is built.

Use Ovrto or Gover to invoke a function that is in an overlay. (See below for the differences between Ovrto and Gover.) The overlay desired may or may not be in memory before you invoke it. Both the entering and exiting register contents are defined by the overlay invoked. Common system conventions for overlays that process more than one function suggest that the function code be passed in A. The invocation of an overlay takes the form of the example below (assuming that the registers have already been set up to hold the proper contents):

```
CALL    Ovrto
DB      'Dfn1'
```

Overlay names are defined to be four bytes long, and the overlay name must follow the call to Gover or Ovrto. If the overlay named is not currently in memory, it is read into memory from the SYSRES device by calling Runr. We enter at the overlay start address, OVENT. We will return from the function to the byte following the text of the overlay name in the Ovrto or Gover call.

As part of the overlay load procedure, EIC in EFLG1 is cleared. There is an important difference in this between single and Twin systems that results in a timing window with respect to ^Y. In the single user system, EIC in EFLG1 is cleared before calling Look to find the destination overlay. This leaves a "window" where ^Y is not disabled by EIC from the time Look is called until interrupts are disabled prior to entering the overlay. In the TwinSystem, this overlay does not exist; EIC, if set in EFLG1 is not cleared until after the target overlay is loaded and interrupts are disabled.

In looking for an overlay, the system calls Runr to find a file on the SYSRES disk with the name specified after the call to Ovrto or Gover and the extension OV. If the file is not found or is not runnable, Err is called to process the error. If the file is found, Runr reads it into memory at the load address specified in the file; we do NOT check to see that this is OVRLY! The overlay, when loaded, is entered at OVENT. The overlay name in locations OVRLY through OVNENT+3 is used by the system to "remember" what overlay is in memory for the Ovrto service.

Differences between Ovrto and Gover:

Both Ovrto and Gover invoke a function in an overlay, which may not be in memory at the time, and both return control to the program just after the overlay name following the call to Ovrto or Gover. The only difference between Ovrto and Gover is that Ovrto "remembers" the overlay currently in the overlay area and restores that overlay before returning to the caller, while Gover does not. Both Ovrto and Gover are "super subroutine" calls; they can call subroutines that do not have to be in memory at the time. Ovrto can be used from WITHIN one overlay to call a function in another overlay, since the original overlay is restored after the called overlay completes its processing. Gover does not "remember" or restore the overlay currently in the overlay area, and so it can only be used from programs outside the overlay area.

Error conditions:

If an error occurs in invoking an overlay, the error code/subcode is passed to Err, which reports the error and warmstarts the system. Any errors reported within the overlay are handled by that overlay.

Symbol name: **Rtn**

Single value: Ø418
Twin value: EØ15

Entry: From overlay

Exit: Same as entry

Description:

Rtn is the entry point used by the system for returning from an Ovrto call. On entry to Rtn, the stack has on it the name of the old overlay (to be restored), which was pushed from OVRLY and OVRLY+2.

Under some circumstances in overlays, it may be necessary to use this entry. For example, BASIC provides a number of commonly used service routines in its resident (above USER) portion that are called by overlays. In some cases these resident routines may call functions in other overlays. So, some overlay routines must call services in the BASIC resident "remembering" their overlay name. Here's what the code looks like; assume we're in an overlay and calling service PFIXE in the resident:

```
Pfixe   SHLD    HLtemp  ; stash HL
        LHLD    OVRLY
        PUSH    H       ; first 2 chrs of our name
        LHLD    OVRLY+2
        PUSH    H       ; last 2 chrs of name
        LXI     H,Rtn
        PUSH    H       ; make sure we're here!
        LHLD    HLtemp  ; HL back again
        JMP     PFIXE
```

In the overlay, to use resident service PFIXE, the intermediate routine Pfixe is called. Since Pfixe was called, there is a return address on the stack. The overlay name is pushed onto the stack, then the address of Rtn, and PFIXE JMPed to. When PFIXE returns, it returns to Rtn, which unwinds the overlay name from the stack, insures that overlay is in memory, and then returns to the address on the top of the stack. All registers are preserved.

Symbol name: **Look**

Single value: Ø421
Twin value: EØ1E

Entry:

HL:	Address of lookup block. HL points to a byte containing the length of the file name (from 1 to 31 bytes) followed by the text for the name and the two byte extension (if present).
DE:	unused
BC:	"
A:	Drive number of disk to search for the file (1-9). If the 8ØH bit is set, then the extension is not checked and a match will occur on equal names.
PSW:	unused

Exit:

HL:	unchanged
DE:	If carry is set in PSW, DE contains error code resulting from Look; If carry not set, register contains FDE directory address.
BC:	junk
A:	junk
PSW:	Carry is set on error; clear otherwise.

See Also: Dio, Ckdr, SBUF1, NFCK, NFDIR, GFLOCK, Gfid

Error Conditions:

Error codes are returned in DE with C set in PSW. If an error is returned from Dio in reading in the directory, the code in D is changed from a 1 to a 3; thus error 1Ø9 becomes 3Ø9. Look specific error codes are:

ØØØ The file requested was not found.

ØFF The disk directory is destroyed. The directory checksum computed by Ckdr does not match the checksum stored in the first byte of the directory. All information on the disk is probably lost. If an error other than ØØØ is reported, NFDIR is set to ØØ to invalidate the data currently held in SBUF1, the directory area.

Description:

Look looks up files in the main directory on a disk. It is called with HL pointing to a "lookup block," which consists of the length of the file name (1 <= length <= 31), the text of the name, and the extension (if present). A contains the number of the drive to search, and the 80H bit of A is used to indicate whether or not the extension has to match. Note that Look only searches the root directory for the file; it does not search subdirectories, Gfid must be used for that purpose. If the file is found in the directory, Look returns the address of the FDE (File Directory Entry) within the directory in DE. If for some reason the file is not found or an error occurs when reading the directory, the error code is passed back to the caller with the carry bit in the PSW set. An example of a lookup block and coding to look up file GRONK.BC on disk 2 would be:

```
;
Txt     DB      5,'GRONKBC'
;
        LXI     H,Txt
        MVI     A,2
        CALL    Look
        JC      Oops
```

Description of the Look process:

Look first checks to see if the directory to be searched is resident in the SBUF1 area; system cell NFDIR contains the drive number of the disk whose directory is in the directory area of memory.

If the proper directory is not in memory, Dio is called to read the directory (sectors 0-3) from the specified disk into SBUF1; errors reported from Dio are passed back to the caller with the code in D changed to 03 (error 0103 becomes 0303, etc.).

When the proper directory is in SBUF1, its checksum is computed by calling Ckdr, stored in cell NFCK, and compared to the first byte of the directory. If this checksum does not match that first byte, we consider the directory destroyed and return to the user reporting an 03FF error. If the directory checksum is good, we mark NFDIR with the directory number and scan the directory for the specified file, skipping those files marked deleted. If we come to the end of the directory before finding a match, we report an 0300 error. If the 80H bit was passed in A, noting not to check the extension, Look will report a match on the first file in the directory with the specified name.

In the TwinSystem, Look must aquire the file system semaphore GFLOCK before examining the directory (either in memory

or read from disk). This insures that only one user at a time is accessing the file system.

Symbol name: **Runr**

Single value: Ø424
Twin value: EØ21

Entry:

HL:	Address of lookup block (see Look for description).
DE:	unused
BC:	"
A:	Drive number to search; 8ØH bit set if extension does not have to match.
PSW:	unused

Exit:

HL:	If carry is clear, register holds start address from FDE.
DE:	If carry is set, register holds error code; else it holds junk.
BC:	junk
A:	junk
PSW:	Carry is set if error; clear otherwise.

See Also: Dio, Look, MEMTOP

Error Conditions:

Error codes are returned in DE with C set in PSW if an error has occurred. Runr calls Look and Dio, and so can return any error code generated by these services. Look specific error codes are:

201 No load or start address. The load address field of the file descriptor block is zero. This usually indicates a text file. Note that a start address of ØØØØ is valid.

Description:

Runr is called pointing to a "lookup block" (see Look for description) and a drive number. Runr attempts to find the program identified in the main directory of the drive specified and load it into memory. If it is successful, it returns the start address of the file in HL. If unsuccessful, Runr returns an error code/subcode in DE.

Runr first calls Look with register contents the same as on entry to Runr. Runr returns if Look returns with the carry set, thus passing any Look errors to the caller of Runr. If the file

asked for exists, the FDE (File Directory Entry) is examined for a load address (LA). If the load address is 0000, we report a 201H error, since the file is not runable. If the load address is nonzero, we call Dio to read the file into the memory address given as LA in the FDE. Any Dio errors are passed to the caller. If no errors occur during the read, Runr returns with the start address from the FDE in HL, and the carry flag in the PSW is clear. Note that although calling Runr does not automatically execute the desired program, the program is loaded into memory, possibly overwriting the routine calling Runr. If no extension was given on the file passed to Runr, Look will match on the first file on the specified disk with the given name-- which may not be a "runnable" file. For example, if disk 2 has files "Flange.TX" and "Flange.GO" appearing in that order, telling Runr to run file "Flange" without specifying an extension will return a 201H error, as Look will find file Flange.TX, rather than Flange.GO.

Symbol name: **Rlwe**

Single value: Ø427
Twin value: EØ24

Entry:

HL:	Address of user buffer to read into.
DE:	Prompt string terminated by ØØ byte.
BC:	C: Maximum number of characters to read. B: If Ø, echo termination character. If 1, do not echo termination character.

Exit:

HL:	Points to last character in buffer.
DE:	junk
BC:	B: Length of line read. C: junk
A:	termination character
PSW:	junk

See Also: WHØ, WH1, CBUF, CMDF, Msg, Killi

Description:

Rlwe is used to read an input line. It provides an input prompt by using Msg (see Msg in this section) to output to the screen the string pointed to by DE. Rlwe then reads in characters (allowing editing of those characters) into the user buffer pointed to by HL. C contains the maximum buffer size, and B contains a flag that controls echoing of the termination character. Characters are read into the user buffer until one of the following conditions is met: 1) the buffer is full; 2) the user enters a carriage return (CR). Rlwe returns with HL pointing at the termination character in the buffer, the termination character in A, and the line length in B.

Editing functions supported by Rlwe:

Single characters are deleted in Rlwe by use of the DEL key (DELETE). Delete words by using Control-W. A word is defined as a sequence of contiguous characters a-z, A-Z,Ø-9. Delete an entire line by using Control-X.

Symbol name: **Iexec**

Single value: 0436
Twin value: E030

Entry: No inputs

Exit: May return to environment on stack

See Also: PVEC, EIC, DONT, SBRK

Description:

Iexec is usually connected to PVEC, and causes the Exec to be run when ^Y is hit. The coding for Iexec is as follows:

```
Iexec   CALL    Ovrto
        DB      'Exec'
        JMP     Ioret
```

When Iexec is entered, the Exec overlay is loaded into the overlay area and run. If the "CONTINUE" command is given to the Exec, it returns, which will restore the overlay present before ^Y was hit, and return through Ioret, which will restore registers and continue execution of the program.

Note that this technique is fairly powerful; a program can be running, possibly using its own custom overlays. It is interrupted and the Exec run. From this, disks can be listed, files deleted or typed; any command issued that does not alter user memory or cause the system to warmstart, and the program can be continued.

Symbol name: **Show**

Twin value: E033

Entry: No register inputs

Exit:
A: ØØ
PSW: Z set

See Also: WH1, show, Print, print

Description:

Show is called to display the text following the Show call on the screen, until a ØØ byte is hit. Program execution resumes with the ØØ byte. Text is displayed through WH1. Here is the code for Show:

```
Show    XTHL            ; save HL, get RA
Show1   MOV     A,M
        ORA     A       ; done?
        JZ      Show2   ; jmp/yup
        CALL    WH1
        INX     H
        JMP     Show1

Show2   XTHL            ; get HL back
        RET
```

Symbol name: **Print**

Twin value: EØ36

Entry: No register inputs

Exit:
A: ØØ
PSW: Z set

See Also: WH7, print

Description:

Print is called to print the text following the Print call, until a ØØ byte is hit. Program execution resumes with the ØØ byte. Text is displayed through WH7. Here is the code for Print:

```
Print   XTHL            ; save HL, get RA
Print1  MOV     A,M
        ORA     A       ; done?
        JZ      Show2   ; jmp/yup
        CALL    WH7
        INX     H
        JMP     Print1
```

Symbol name: **Flipem**

Twin value: EØ39

Entry: Internal vector, do not call

See Also: Ticker, PMASK

Description:

Flipem is an internal service that flips from one user to the other in the TwinSystem. It should not be called or used directly. See the following description of Giveup for details.

Symbol name: **Giveup**

Twin value: E03C

Entry: No inputs

Exit:
INT: Enabled, no registers modified

See Also: PMASK, PHANTOM, BRG, BRGEN, LOCK, giveup
Ticker, USP, Ioret

Description:

Giveup is called to give up the processor, either by calling Giveup, or by using the giveup macro. It saves the user's registers on the stack, saves the stack pointer, flips to the other user, and loads that user's environment, and starts executing that user. Let's look at the code in detail:

```
Giveup  DI
        PUSH    PSW     ; std save sequence
        PUSH    B
        PUSH    D
        PUSH    H
        XRA     A
        STA     LOCK    ; insure not locked!

; Flipem flips to the other process. We first stash the
; stack pointer at USP, and then flip the PHANTOM bit of
; the baud rate generator, load the new SP from USP, and
; return thru Ioret to load the new environment.

Flipem  DI
```

```
        LXI     H,Ø     ; assume regs stashed!
        DAD     SP
        SHLD    USP     ; save SP, user now parked!
Flipl   LDA     PMASK   ; get flip mask
        MOV     B,A
        LDA     BRG     ; current BRG contents
        XRA     B       ; flip the magic bit
        OUT     BRGEN   ; we are the other guy now!
        STA     BRG     ; mark it done
        LHLD    USP
        SPHL            ; load new SP
        JMP     Ioret   ; go to it!
```

This code is very important; it's the core of the TwinSystem. When we enter at Giveup, we already have a PC on the stack from whoever called Giveup. We then push the registers on the stack; this forms the "environment" that Ioret will restore when we return to this user. We clear LOCK, as the user is giving up the processor, and fall into Flipem.

The first thing Flipem does is to save the user's stack pointer. This is done with the interrupts disabled so that nothing can be modified. With the user's stack pointer stored at USP, the user is now "parked". We have saved all the information necessary to restart the user later on. Note carefully that USP, as well as the stack, is kept in the 48K board that is switched with the user. We pick up the contents of PMASK, which contains PHANTOM if we are running two users, and ØØ if we are running in solo mode or on a single user system. We pick up the current baud rate generator contents, and flip the bits set in PMASK. Doing the OUT BRGEN causes the PHANTOM line on the backplane to change, assuming we're running two users. We have now selected the other user. We store the new baud rate generator contents (BRG is in system common memory, in CPU board RAM). The new stack pointer is loaded from USP (the new user's). When we jump to Ioret, it unwinds the registers from the new user's stack, and continues executing that user's code.

Symbol name: **Ticker**

Twin value: EØ3F

Entry: From RTC interrupt via SUWH8

Exit: Through Ioret

See Also: USP, UTIME, WAKEUP, PMASK, BRG
BRGEN, LOCK, Flipem, Lock, Unlock

Description:

Ticker is the TwinSystem clock server. It is connected through SUWH8 and is entered on every RTC interrupt, 6Ø per second. Its basic purpose is to cause the switching between users to share the CPU. It has the subsidiary tasks of incrementing user clocks, posting the WAKEUP interrupt, and enforcing the processor lock via LOCK. Let's look at the code and then discuss it; we are entered via the PCHL in the interrupt code with interrupts disabled and all registers pushed onto the stack:

```
Ticker  OUT     8       ; reenable interrupt
        LHLD    XTIMER
        INX     H       ; bump ØCØØ clock
        SHLD    XTIMER  ; for everybody to see

TickØ   LXI     H,Ø
        MOV     B,L     ; mark diddling current user
        DAD     SP
        SHLD    USP     ; user now parked!

Tick1   LXI     H,UTIME ; user timer
        MVI     C,4     ; 4 bytes long
Tick2   INR     M
        JNZ     Tick3   ; jmp/not time yet.
        INX     H       ; bump next cell
        DCR     C       ; did we hit all ØØ?
        JNZ     Tick2   ; jmp/not yet.

; User clock went to ØØØØ. Build environment to invoke
; WAKEUP routine next time we run 'em.

        LHLD    USP     ; pick up SP
        SPHL            ; and insure loaded
        LHLD    WAKEUP  ; vector
        PUSH    H       ; new PC
        PUSH    PSW
```

```
        PUSH    B
        PUSH    D       ; reg contents don't matter
        PUSH    H
        LXI     H,Ø
        DAD     SP      ; save updated SP,
        SHLD    USP     ; all done.

; End of clock processing. See if we've diddled both
; clocks, and split thru Ioret when we have. B will be
; nonzero if we've done both users.

Tick3   MOV     A,B
        ORA     A       ; done both yet?
        JNZ     Tick4   ; jmp/yup
        LDA     PMASK
        ORA     A       ; anybody else to do?
        JZ      Tick4   ; jmp/nope!
        MOV     B,A     ; remember doing other guy
        LDA     BRG
        XRA     B
        OUT     BRGEN   ; ***** SWAP *****
        JMP     Tick1   ; go do other guy.

Tick4   LDA     BRG
        OUT     BRGEN   ; insure mapped to first one
        LXI     H,LOCK
        MOV     A,M     ; software lock set?
        JZ      Flip1   ; jmp/not locked, flip 'em
        MVI     M,2     ; mark quantum overrun
        LHLD    USP     ; get stack back
        SPHL
        JMP     Ioret   ; and let 'em run some more!
```

If you understand this code, Giveup, and Flipem, you understand how the TwinSystem runs two users. When we are entered from the RTC vector, we reset the clock interrupt and bump the clock kept at ØC ØØH. The user's PC and registers are already on the stack; we save the SP in USP to "park" him. From TickØ through Tick3 we use the B register to note if we are working on the current user (B contains ØØ) or the other user (B contains PHANTOM).

The general idea in the remainder of the code is to bump both user's timers. If a user's timer increments to all zero, we want to schedule the specified WAKEUP routine to be called the next time the user is run. Note that in the single user system, we just jump to the WAKEUP routine directly from the interrupt level. This would adversely affect performance in the Twin.

The first 7 lines of code at Tick1 take care of incrementing

the user's clock. Note the clock is kept in per-user memory (in the 48K board) so each user has his own copy, at the same address. If the clock doesn't go to all zero, we go to Tick3. If the clock does go to all zero, we fall into the second chunk of code. The first thing we do is load the user's stack pointer; we may not have the correct one from switching users in Tick3. We then build an environment on the stack to take the user to the WAKEUP specified routine the next time the user is run, and update USP.

At Tick3 we check to see if we have "diddled" both user's clocks, if there are two users. If we have done both users, we go off to Tick4 to check the software lock. If we only have one user, we go to Flip1 to continue running that user. Note that we do this through Flip1 to insure the hardware and the BRG are set up properly, rather than going directly to Ioret. If we haven't done both users, we flip to the other user, setting B nonzero for the next time we come through Tick3. Note that we don't update BRG; this swap is only temporary while we update the other user's clock.

At Tick4, we have updated clocks for both users. If the software lock is not set, we go to Flip1 to flip to the other user. If the software lock was set, we change it to indicate that the locked user ran over a clock tick. When the user calls the Unlock system service, Unlock will Giveup as a result. Since the processor is software locked, we restore the user's stack, and go to Ioret to continue execution.

Symbol name: **Gfid**

Twin value: E042

Entry:

A and 0FH	Function
0	Get file identifier
1	Enter/replace FDE
2	Look up a file
3	Update directory (TwinSystem only)

The registers on entry contain:

HL: Points to either the prompt string to use in reading from the user (Via RLWE) or the address of the text buffer to examine.

DE: Points to the buffer that is used to return the file specification in form suitable for feeding to LOOK.

BC: May contain the default extension to use if the user does not give one.

A: 80H if set, read (RLWE) from the user, prompting with the MSG -> by HL.

40H if set, LOOK the file up. If it exists, copy the FDE to the buffer -> (DE)+1, thus returning FDA, NSCTR, LA, SA. If an 0300 error is returned from LOOK, return NFA from directory in the FDE slot in the buffer -> (DE)+1.

20H If set, use the extension in BC if the user does not specify one.

Return:

HL: Points to the ending delimiter in the text buffer.
DE: If C set in PSW, error code. If not, FDE address if LOOK was requested, else junk.
BC: Junk
A(PSW): C set if error, clear if not.

Errors:

0500	Invalid disk #
0501	Name longer than 31 characters
0502	Extension longer than 2 characters
0503	Name zero length

Enter FDE:

HL: Points to a file block as built by get file

function. First byte has disk #, next byte has flags and name length, etc.

A: 80H If set, replace an existing FDE with the one pointed to by HL, else enter a new one at the end of the disk.

40H If set and 80H set, replace existing FDE and clear "new" bit.

Returns:

A(PSW): If C set, then DE has error code/subcode. If C not set, then registers scrambled.

Errors:

0504 Directory is full

Update directory: uses NFDIR, DirAddr, SBUF1 information

See Also: GFLOCK, NFDIR, NFCK, SBUF1, Dio

Description:

Single User Gfid

In the single user system, Gfid is an overlay (Gfid.OV). Gfid provides three main services: Get and parse file identifier, Enter or replace directory entry, and Look up a file. The Look function provided by Gfid differs from the resident Look service in that it accepts full pathnames involving subdirectories and the resident service does not.

TwinSystem Gfid

Gfid in the TwinSystem is a resident service. For compatibility, a Gfid overlay is provided; it just jumps to the resident service vector. On the Twin, the file system and Gfid is considered a "critical section", only one user may be running that code at a time. For this reason, entry into Gfid or into Look on the Twin is controlled by the GFLOCK semaphore.

If the user requests that Gfid read a line from the user in the Get function (A AND 0FH =0), the line is read before the semaphore is acquired.

Some system functions, such as RENAME, DELETE, and UNDELETE, modify the contents of directories. In the Twin, Gfid has an additional command for updating a directory. Gfid looks at the contents of NFDIR and DirAddr when this command is given. If NFDIR does not have the 80H bit set, the directory in SBUF1 is a root level directory; its disk address is 0000. If the 80H bit in NFDIR is set, this is a subdirectory and DirAddr has its disk

address. Gfid computes and updates the checksum of the directory in SBUF1 and writes it to the disk.

TwinSystem Gfid - Invalidating Directories

In the TwinSystem, each user has a directory area. So, when Gfid updates any directory on disk, it must see if the other user has a copy of that directory. If the user does, this copy is now invalid; Gfid sets NFDIR for that user to ØØ. This is why Look and Gfid are protected by the GFLOCK semaphore; so that only one user at a time may modify directories. This is also why looking into the directory area is not a good idea; programs should use Gfid to get file information.

NOTE: For more information on Gfid, see Section 2 on the file system.

Symbol name: **DEOUT**
Single value: Ø3D1

Symbol name: **Deout**
Single value: Ø3D1
Twin value: EØ45

Entry:
DE: 16 bit value to display on screen

Exit:
A: Last ASCII character output

See Also: Byte, WH1

Description:

Deout (or DEOUT) is called to display the contents of the DE registers on the screen in hex. This is done by calling Byte with the D and E registers.

Symbol name: **Byte**

Twin value: EØ5A

Entry:
A: Hex byte to display

Exit:
A: Last ASCII character displayed

See Also: Deout, WH1

Description:

Byte displays the byte in A in hex on the screen. The characters are displayed by calling WH1. Here is the TwinSystem code for Deout and Byte:

```
Deout   MOV     A,D
        CALL    Byte    ; display D
        MOV     A,E     ; then E
Byte    PUSH    PSW     ; save it
        RRC
        RRC
        RRC
        RRC
        CALL    Byte1   ; top 4 bits
```

```
            POP     PSW     ; then bottom 4 bits
     Byte1  ANI     0FH     ; just 4 bits at a time
            ADI     90H
            DAA
            ACI     40H
            DAA
            JMP     WH1
```

Symbol name: **Lock**

Twin value: E048

Entry: No inputs

Exit: All registers and interrupts unchanged

See Also: LOCK, Ticker, Lock, Unlock, Vti, WAKEUP

Description:

Lock is called to lock the processor against the normal time slicing done by Ticker as a result of real time clock interrupts. It is used, for example, by the screen driver (Vti) and the editor to prevent slicing while updating and scrolling the screen. Using Lock prevents switching to the other user for normal processing, interrupt character processing, WAKEUP processing, or I/O completion processing. Only character typeahead is performed, although environments may be stacked for interrupt characters and WAKEUP events. That Lock blocks I/O completion processing is important to understand. Let's say user 1 requests a read from drive 4. The read is started, and user 1 gives up the processor. Then user 2 calls Lock to lock the processor. User 2 cannot access drive 4 (or 5, 6, or 7 on a normal MS configuration), as user 1 still has an I/O operation uncompleted. If user 2 tries to access drive 4, the system will deadlock. So, Lock should be used sparingly!

Symbol name: **Unlock**

Twin value: E04B

Entry: No inputs

Exit:

A:	junk
PSW:	junk
INT:	Enabled/unchanged

See Also: LOCK, Giveup, Ticker, Lock, lock, unlock

Description:

Unlock is called to remove the processor lock set by the Lock system service or the lock macro. It may be called directly or through the unlock macro. It may also be called if the processor was not locked. When called, the software lock is cleared. If the real time clock did not tick while the processor was locked, control is returned directly to the user. If the processor was locked over a clock tick, Giveup is called to give up the processor before returning to the user. See the coding for Ticker and Giveup for details.

Symbol name: **Enter**

Twin value: E04E

Entry:

HL: Address of semaphore

Exit:

A: 00
PSW: Z set
INT: Disabled

See Also: Gfid, gLook, enter, Giveup, Leave
GFLOCK, Interrupt character processing

Description:

Enter is used to enter a critical code section protected by the semaphore byte pointed to by HL. The semaphore must be in shared memory (i.e. not between 2000H and DFFFH), although this is not validated by the Enter code. Here is the code for Enter:

```
Enterw  CALL    Giveup  ; wait a while

Enter   DI
        MOV     A,M     ; get value
        ORA     A
        JNZ     Enterw  ; wait till it goes 00
        LDA     BRG
        ORI     80H     ; insure nonzero
        MOV     M,A     ; mark we have it
        XRA     A
        RET             ; return to user
```

Note very carefully what Enter does and does not do. It does not check to insure that the semaphore is in shared memory. It does not insure that we do not already have the semaphore. It does not insure that the semaphore starts out zero. It just waits till the semaphore byte goes to zero, then marks it taken and returns with interrupts disabled. The semaphore is set with 80H ORed with the user tag primarily for use with GFLOCK inhibiting the action of interrupt characters. See the coding example for Interrupt Character Processing in the TwinSystem for details.

Symbol name: **Leave**

Twin value: E051

Entry:

HL: Address of semaphore

Exit:

Registers unchanged, interrupts enabled

See Also: Enter, enter, leave

Description:

Leave is called to leave the critical section marked by the semaphore byte pointed to by HL, clearing the semaphore. It returns with interrupts enabled. The semaphore is not checked to see if the user calling Leave currently owns the semaphore.

Symbol name: **Devlock**

Twin value: EØ6F

Entry:

C: Device number, Ø-F
B: Command, as high and low nybbles:

B AND ØFØH:

8Ø	Set permanent allocation
9Ø	Initialize
AØ	Set temp allocation
BØ	Invalid command
CØ	Clear permanent allocation
DØ	Invalid command
EØ	Clear temp allocation
FØ	Reset to permanent allocations

B AND ØFH:

1	cmdf	- Command file read
2	rd	- Read
3	wrt	- Write
4	fupd	- File update
5	mung	- Dir mung/file create/ rename **
6	wlock	- Write lock
7	excl	- Exclusive use

Exit:

DE: Error code if C set in PSW
PSW: C set on error, C clear otherwise
INT: Disabled/unchanged

See Also: BHA, Giveup, Dio, DONT, SBRK, cmdf, rd, wrt, fupd, mung, lock, excl, sett, setp, clrt, clrp, devlock

The interlock mechanism arbitrates device access between users in the TwinSystem. Its main purpose is to keep users from stomping each other, or otherwise getting in each other's way. Device ØØ is the printer; devices 1 through 7 correspond to disk drives 1 through 7. Devices 8 through F are for expansion; device codes C-F may be used by user programs wishing to use Devlock.

Devlock is called directly by Dio before performing any disk operation; it is also called by printer functions to allocate the printer. Devlock is called by system programs to get other classes of device access; for example, PACK attempts to get

exclusive use of the disk to pack. Devlock is also called by the Exec SET command.

The access classes are shown above, and are defined as symbols cmdf, rd, wrt, fupd, mung, wlock, and excl. The access class is combined with a command in the B register, and the device number in the C register, for Devlock to use.

Devlock uses a set of lists indexed by device number. Each user has two lists: one for temporary allocations, and one for permanent allocations. Any allocation set in the permanent list is also set in the temporary list.

Clear Allocation

The clear allocation commands, clear permanent (CØH) and clear temporary (EØH), simply clear the allocation class specified. If the allocation had not been granted, the service returns. Note that clear permanent also clears the temporary allocation of that class for the device.

Grant Allocation

In granting an allocation, through set temporary (AØH) or set permanent (8ØH) commands, a number of steps are taken. First, if the system is running on single user hardware (PMASK contains ØØ and the 4ØH bit in BHA is clear), the allocation is granted. Next, a check is made to see if the allocation has already been granted, and we return if so. If not already granted and this is a permanent request, we see if the allocation is set in the temporary list. If it is, we set it in permanent and return.

If the permission was not already present in one of the lists, we look to see if it is implied by an already granted permission, either permanent or temporary. For example, write permission implies command file and read permission. Exclusive use implies all permissions. So, if the access is implied by an already granted permission, we set that in the proper list and return.

If the permission was neither already set nor implied, we must check the other user's tables to see if that user has been granted an access that excludes our request. Based on the numeric values of the access classes specified above and by the equates, the permission matrix looks like this:

= granted X = denied		<-- CURRENT STATE --> Ø	1	2	3	4	5	6	7
R	1								X
E	2								X
Q	3							X	X
U	4					X		X	X
E	5						X	X	X
S	6				X	X	X		X
T	7		X	X	X	X	X	X	X

If there is no conflict, the permission bit is set in the tables and we return. If there is a conflict, our action is determined by the contents of PMASK and BHA. If PMASK contains ØØ, indicating we are in SOLO mode (we checked for strict single user earlier), this means that the other user holds the device, and we have shut that user off; we abort with a Ø1C2 error (device busy). If PMASK is nonzero, denoting Twin operation, we examine the 2ØH bit of BHA. This bit is set in response to the Exec command SET WAIT ON. If this bit is clear, WAIT mode is not enabled; we return the Ø1C2 error to the requesting user; the device is busy.

WAIT Mode

If WAIT mode is enabled, we set DONT nonzero to inhibit interrupt character action. SBRK is tested to see if its contents are nonzero. If SBRKs contents are nonzero, we clear DONT and return reporting the Ø1C2 error; ^Y has been hit by the user. If SBRK contains zero, we unwind the saved registers from the stack, give up the processor, and reenter Devlock to try again.

Symbol name: **Ioret**

Single value: 0064

Twin value: E066

Entry: No inputs

Exit:
INT: Enabled, registers returned from stack.

Description:

Ioret is jumped to to return to an environment on the stack, usually pushed as the result of an interrupt, or time slicing on the TwinSystem. Here is the coding for Ioret:

```
Ioret   POP     H
        POP     D
        POP     B
        POP     PSW
        EI              ; let in interrupts
        RET
```

Symbol name: **Moven**

Twin value: E069

Entry:

HL:	Source address
DE:	Destination address
BC:	- # bytes to move

Exit:

HL:	Source + count + 1
DE:	Dest + count + 1
B:	Checksum of data transferred
C:	ØØ
A:	junk

Symbol name: **Move**

Twin value: E06C

Entry:

HL:	Source address
DE:	Destination address
BC:	# bytes to move (positive)

Exit:

HL:	Source + count + 1
DE:	Dest + count + 1
C:	ØØ
B:	Checksum of data transferred

Description:

Moven and Move are block move routines available in the TwinSystem. They differ only in that Moven takes the number of bytes to move in BC as a negative number, and Move uses a positive number. The move routine uses what is known as an "unrolled loop", and is very fast, especially for large blocks of data. It is used internally for scrolling the screen and transferring data to and from the disk controller buffers.

Symbol name: **Vmgr**

Twin value: E054

Description:

Vmgr is a service routine used internally in the system as part of the disk I/O mechanism. It should not be called by the user.

Symbol name: **Gdfp**

Twin value: EØ72

Entry: No inputs

Exit: Interrupts enabled

See Also: Devlock, BHA, TwinSystem printer driver

Symbol name: **Fdfp**

Twin value: EØ75

Entry: No inputs

Exit: Interrupts enabled

See Also: Devlock, BHA, printer driver

Description:

Gdfp and Fdfp are called to get and free the printer in the TwinSystem. When the printer driver is initialized, wormholes WH5, WH6, and WH7 are initialized by Fdfp to contain calls to Gdfp. When a program calls WH5, 6, or 7, Gdfp is invoked. It attempts to get temporary exclusive use of device ØØ through Devlock; device ØØ is the printer. If Devlock returns with C set in PSW, we jump to Err to abort and report the error. Note that if wait mode was enabled (see Devlock and BHA), the user will wait until the printer is available. Once the printer is available, the wormholes are connected to the printer driver, and the proper wormhole invoked. Gdfp can also be called from the user program to insure the printer is connected.

SYSTEM OVERLAYS

The internal structure and flexibility of the System 88 disk system is based on the overlay mechanism. This section describes the internal structure of overlays in the System 88 and the facilities provided to the assembly language programmer by the various system overlays. These discussions assume that you have perused the descriptions of Gover and Ovrto (the overlay linkage facilities).

The overlay area in System 88 memory is from 2000H (OVRLY) to 27FFH; overlays are therefore assembled for this area of memory and do not exceed 2K bytes in size. Overlay names are four characters long and may not contain blanks, tabs, or other control characters. The first four bytes of the overlay (OVRLY to OVRLY+3) must contain the overlay name, which must match the file name. The file name and the internal name must match so that Ovrto can "remember" the current overlay. When an overlay is brought in by Ovrto or Gover, it is entered at OVENT. The contents of the registers are unchanged from the call to Gover or Ovrto.

When writing overlays, the 2K byte space reserved for overlays may be used in any manner you choose. Overlays are assumed to be "pure" code, code that does not modify itself. Portions of the overlay area may be used by the overlay itself for buffers or data. Remember that such data is lost if another overlay is invoked. Arguments may not be passed to other overlays through the overlay area itself.

When entered at OVENT, interrupts are disabled and the EIC (Exec in Control) bit in EFLG1 is not set. If the overlay wishes to process Control-Y interrupts from the user, PVEC should be set accordingly. Note that if 1) the user types a Control-Y, 2) the EIC bit is not set, and 3) your program did not set PVEC, then the overlay area will be overwritten by the Exec overlay (brought in as a response to the Control-Y). If the user then gives the Exec a CONTINUE command, the previous overlay will be restored from disk and reentered at the point where it was interrupted with the original register contents but without any data stored in the overlay area.

The overlays provided as part of the standard disk system are protected from abuse (deletion, renaming, etc.) by having the system bit set in the FDE for each. You may set the system bit in the FDE (or clear it) using the Szap utility described in Section 4 of this manual, or the "Tweak" program listed in section 2. You must use either Szap or the Gfid replace function described later in this section. This degree of difficulty encourages caution and planning, and discourages thoughtless experimenting.

For an example of an overlay, refer to the listing of the system error message writer overlay, Emsg, given in Section 7.

System Overlays

As shipped, the single and Twin systems have a number of overlays that perform system functions. Some of the functions provided by the overlays may be invoked by the user. Each overlay will be discussed, with the functions it provides.

Overlay	Function
Exec.OV	System executive Command processing
Emsg.OV	Error message display
Dfn1.OV	Disk functions
Dfn2.OV	Disk functions
Dfn3.OV	Disk functions
Gfid.OV	Get/enter file services
Prnt.OV	Printer functions
Pack.OV	Pack a disk
Efun.OV	Editor functions
Mfun.OV	Misc functions (Twin only)
Berr.OV	BASIC error reporting
Bfun.OV	BASIC functions
Bslv.OV	BASIC program save/load
Bdir.OV	BASIC direct commands
Xref.OV	BASIC cross reference
Amsg.OV	Assembler error reporting
Vmgr.OV	Volume Manager

Note that the overlays used by BASIC, the assembler, Editor, and printer are for a specific version. Overlays from one version of BASIC may not be used by another version. If such a mix is attempted, disaster will be the result. Since the overlays on the system disk are marked as system files, the user can't meddle with them. This problem can only be caused by a systems programmer!

The overlay functions described are for the Exec/94 release of the system, and may not correspond to previous versions of the system. This is especially true when dealing with Dfn1, Dfn2, and Dfn3. In producing the TwinSystem, functions performed in these overlays were shuffled around to meet the 2K byte size constraint. Also, some functions were reduced or eliminated entirely.

Some overlays are driven from the function code passed in A. Most overlays do **not check** to see that this value is within the expected range. A bogus function code will cause bogus results.

Exec.OV is central to the operation of the system. It parses user commands and initiates their execution. It also processes errors reported through Err, and is the default handler for control Y (^Y) as set through Iexec. The Exec also plays an important role in the system boot sequence.

Emsg.OV is the system error message handler. When invoked with an error code in DE, it displays the appropriate text on the screen. Note that the text does not terminate with a carriage return; this must be supplied by the user program, if needed. Emsg is set up for use with the system error message editor, Emedit.

Dfn1.OV provides the following functions, based on the function code passed in A:

Code	Command/Function
Ø	Unused
1	IMAGE
2	INIT
3	Unused
4	RENAME
5	SetSys
6	Unused
7	Unused

The IMAGE, INIT, and SetSys functions all request information from the user (via reads through Rlwe and WHØ), so they are not particularly useful as callable functions. The command string for RENAME is pointed to by CMPTR, so this function may be invoked by the user program. The overlay will always return to the calling program, but does not give any indication of error or success to the caller. The overlay takes control of PVEC and may not restore it before returning.

Dfn2.OV provides the following functions, based on the function code passed in A:

Code	Command/Function
Ø	LIST
1	DELETE
2	UNDELETE
3	Unused
4	Unused
5	Sniff
6	Unused
7	DIR

8	boot
9	DLIST

The LIST, DIR, and DLIST commands pick up their argument using CMPTR. Note that LIST and DLIST stop after each 15 lines of display, requesting a character from WHØ before continuing. DELETE, UNDELETE, Sniff, and boot also expect CMPTR to be pointing to their arguments. The boot command should be used with extreme caution on the TwinSystem; both system disks must be the same revision (of the TwinSystem). The overlay will always return to the calling program, but does not give any indication of error or success to the caller. The overlay takes control of PVEC and may not restore it before returning.

Dfn3.OV provides the following functions, based on the function code passed in A:

Code	Command/Function
Ø	Unused
1	Memory Dump Function
2	DUMP command in Exec.
3	COPY
4	DNAME
5	PRINT
6	TYPE
7	SAVE

The Memory Dump function of Dfn3 is called with the registers as follows:

HL: Starting address of the memory area to dump to the printer.
DE: Ending address of area to dump.
BC: The address of a string to be printed at the top of the memory dump, terminated by a CR and a ØØ byte.
A: 1 (To select the memory dump function in Dfn3.)

On exit from the overlay all registers contain junk.

Dump dumps the selected memory area specified by the contents of HL and DE to the printer. The string pointed to by BC is printed along with the memory limits as the first line of the memory dump. This string should be terminated by a carriage return and a zero (ØØ) byte. Dump first outpust the memory limits to the printer, then begins dumping memory in hexadecimal form, sixteen bytes per line. After a line is printed, the next sixteen bytes are examined to see if they are identical to the previous 16 bytes. If they are identical, this 16 byte area is not printed, as it is a duplication of the preceding area.

The DUMP Exec command does the same thing as the Dump Memory function directly from Exec. The form of the command is:

DUMP ADR1 ADR2 COMMENT

This command is valid only in the ENABLED mode. ADR1 and ADR2 are the beginning and ending memory limits, and COMMENT is any string of characters up to a carriage return that is displayed on the header line of the dump.

The SAVE command requests input from the user, and is probably not useful as a callable function. PRINT, TYPE, and COPY all use CMPTR to retrieve arguments. DNAME requests input directly from the user. The overlay will always return to the calling program, but does not give any indication of error or success to the caller. The overlay takes control of PVEC and may not restore it before returning.

Gfid.OV is responsible for parsing file names, entering files into directories, and replacing file entries in directories. For more information on Gfid, see the descriptions in the sections on the file system and system service vectors. In the TwinSystem, the Gfid overlay just jumps to the resident service vector.

Prnt.OV handles printer setup and modification. See the section on the printer for more information.

Pack.OV is called to pack a disk. It assumes that user memory has been set to zero by the Exec. It uses CMPTR to locate the text of the drive to pack. This function should not be invoked by the user.

Efun.OV provides services to the editor. It has no functions that can be used by other programs.

Mfun.OV appears only on the TwinSystem. It contains miscellaneous routines and functions used in the Twin. It provides the following functions, based on the function code passed in A:

Code	Command/Function
Ø	SET
1	Porfavor
2	ziffle

The ziffle command uses CMPTR to retrieve its argument. The argument is expected to be a file name followed by a space and the seed string. The seed string may fill the remainder of the line, just as long as you can remember it. The file is encrypted

in place, one sector at a time, so the process should not be interrupted. Porfavor and SET also use CMPTR to retrieve arguments. The overlay will always return to the calling program, but does not give any indication of error or success to the caller. The overlay takes control of PVEC and may not restore it before returning.

Amsg.OV is the Macro-88 Assembler error message handler. It has no functions that can be used by other programs.

Vmgr.OV handles the hard disk volume allocation. Refer to section 10 for more information.

BASIC Overlays

The descriptions that follow are for BASIC version CØ3. With the exception of Berr, the overlays for BASIC do not provide user callable functions. These overlays are used to reduce the amount of memory BASIC needs to run. They assume that the BASIC resident is in memory, and they call functions provided in this resident, as well as modify data areas defined by the resident. This is why switching components between versions won't work, and why it's hard to use these overlays other than with BASIC.

Bdir.OV contains "direct" commands, as well as cold start and reenter code. The direct commands are: LIST, DELETE, RENUMBER, DIGITS, and program line entry.

Xref.OV contains the cross reference generator for BASIC.

Berr.OV contains the error processing and recovery code for BASIC. Note that the first part of error processing, including ON ERROR statements, is handled by the BASIC resident. If Berr is called with an error code in DE and a 5 in A, it will display the corresponding error message on the screen. All other function codes in A rely on internal functions or data in BASIC and should not be invoked by the user. Berr is set up to work with the system error message editor, Emedit.

Bslv.OV provides SAVE, LOAD, SAVEF, SAVEP, CHAIN, and LINK functions.

Bfun.OV provides many of BASIC's internal functions; this overlay is resident most of the time when a BASIC program is running. It performs graphics functions (PLOT, DRAW), scientific functions (MOD, EXP, SIN, COS, SINH, COSH, SQRT, ^, LOG, TAN, TANH, ATAN, ASIN, LOGlØ), matrix functions (SUM, PROD, MAX, MIN, MEAN, STD), as well as INP and OUT.

BASIC can be run with from 6 to 26 digits of precision. This means that the scientific functions must be implemented in a manner that will give full accuracy over a wide range. For this reason, power series expansions are used for most functions, with terms being generated until the point is reached where subsequent terms will not add to the accuracy of the result. SQRT uses a modified Newtonian iterative approximation. The number of iterations is determined by the number of digits precision. Expressions of the form X^Y are evaluated by repeated multiplication for Y an integer less than 1ØØ (decimal), otherwise a log expansion is used.

Section 4

Utilities for the System Programmer

Section 4 describes the utilities for the System 88. Each of these programs is on the disk included with this manual. The programs included are the following:

EMEDIT an editor for error message overlays. This program allows the systems builder to tailor system error messages to the end user and to add new messages for use by applications systems.

SZAP a program used for examining and manipulating the contents of disks and memory. SZAP is a powerful tool meant for use by experienced programmers.

SCOPY a program used for copying a large number of files at one time or copying a new version of a program over the old version.

FUTIL a file utility program used to create a command file to copy entire directories or parts of directories.

RDB a debugger for machine language programs.

Auth.OV may be installed on a user's system disk to inhibit access by unauthorized personnel.

SPACE displays the number of bytes remaining in the directory named on the command line.

WAIT allows a pause during command file execution.

TWID displays the contents of a symbol table file.

COMPARE compares two files.

COMP-DISK compares two disks.

CLEAN reinitializes the root directory on a disk.

ARISE undeletes a selected deleted file.

DIRCOPY copies all files within a directory, including the subdirectories within it.

RECOVER recovers a file from a disk with a bad directory.

EMEDIT - An Editor for Error Message Overlays

EMEDIT allows the system programmer to examine and modify error message overlays in the System 88. Using EMEDIT, you can view messages in an error message overlay, delete or add messages, list the messages to the system printer, or replace existing messages.

Restrictions

EMEDIT will edit only system error message overlays. This means the name of the file to be edited must be exactly four characters long. The load and start addresses in the file must be 2000H. Location 2007H in the overlay is expected to contain a pointer to the body of messages within the overlay (see Section 7). In addition, EMEDIT must be invoked in the enabled mode. If invoked in disabled mode, EMEDIT returns to the Exec. At this time, the error message overlays on the disk are Emsg.OV, Berr.OV, and Amsg.OV.

Using EMEDIT

The user invokes EMEDIT when the system is in the enabled mode. EMEDIT then displays its version number and command list on the screen. Give commands to EMEDIT by typing a command character, in either upper or lower case (EMEDIT folds lower case to upper case). The command characters are:

Character	Command
A	Add message
D	Delete message
E	Edit error file
R	Replace message
V	View messages
X	Exit
S	Sort
L	List messages

If EMEDIT does not recognize the character typed as a legal command character, EMEDIT again displays the command list on the screen.

Adding messages to the text: The A Command

In response to the A command, EMEDIT prompts the user for the error code to add by displaying the text

Add error code:

and accepts a hexadecimal number followed by a carriage return. Lower case letters are folded to upper case, and the conversion stops when a character not in the set a-f, A-F, Ø-9 is encountered. This number is the number under which the new message will be stored. Any existing messages in the file with that code will be deleted. EMEDIT then displays:

Terminate new message with ESC-CR

on the screen, and prompts the user for lines of input with the prompt character <. Input is accepted until a line ending with an escape (ESC) character followed by a carriage return (CR) is detected. This terminates processing in the Add command. If no file has been opened for editing by using the E command, the error text

No file open for editing- use E first

is displayed on the screen. If the added text would force the overlay over the maximum size of 2K bytes, EMEDIT displays the message:

Message truncated-Overlay is full!

EMEDIT truncates the added text and terminates the Add command.

Deleting a Message: The D Command

The D command is used to delete messages from the file. The user is prompted with

Delete error code:

and a hexadecimal number is input by the user for the message code to delete. If the message is not found, the text

I can't find that message

is displayed. If no file has been opened for editing, the text

No file open for editing- use E first

is displayed and the command terminated.

Opening a File for Editing: The E Command

After typing E to invoke the Edit command, EMEDIT prompts with

Edit file name:

and waits for the user to enter the name of the error message overlay to edit (followed by a carriage return). The file is validated, as described in the section on Restrictions. Any errors in looking up the file are reported to the user and terminate the command. If the file does not look like an error message overlay, the text

That's not an error message overlay!

is displayed and the command terminated. If another file was open for editing at the time the E command was given, that file is closed, and it is rewritten to disk if modifications have been made.

Replacing a Message: The R Command

The R command is similar to the A command for adding a message, but it assumes that there is a message with that code already in the file. The user is first prompted with the text

Replace error code:

and the user inputs the error code. If no message with that code is found within the overlay text, the message

I can't find that message.

is displayed on the screen and the command terminates. If the message exists, it is deleted, and the A (Add) command invoked to add the message.

Viewing the Contents of the File: The V Command

The V command displays the messages in the file on the screen. The display stops at the end of each page, and a dot is displayed. The user may type either the single character x or X to abort the display at that point, or any other character to continue the display. The error codes and texts are displayed in their order of appearance in the file. Messages added with the A (Add) or R (Replace) commands will appear at the end of the file.

If no file is open for input, the message

No file open for editing- use E first

is displayed and the response to the command aborted.

Listing messages to the system printer: The L Command

The L command lists all error messages in the currently open file to the system printer. An error message results if no file is currently open for editing. The error messages are listed in the same format as produced by the V (View) command.

Exiting the program: The X Command

If no file was open for input when the X command was given, or no modifications had been made to the file currently open, EMEDIT returns to the Exec. If the currently open file was modified by use of the D (Delete), A (Add), or R (Replace) commands, the new file will be written to disk. If the file has not increased in size, the new contents will be written over the old file on the disk. If the file has increased in size, the old copy of the file is deleted (even if it is a system file), and a new copy of the file is created on the disk.

Editing BASIC Error Messages

The error messages for BASIC are in the error message overlay Berr.OV. To save space in the overlay, if the last character in a message is the letter e, this will be expanded to the word "error." This expansion turns the string "Syntax e" in the Berr file into the string "Syntax errror" on the screen when the error is reported from BASIC. The user be aware of inserting messages into the Berr file that end in the single character e.

Suggestions for using EMEDIT

When adding new error codes to the system, write them down and add them to the User's Guide, as well as to the System Programmer's Guide and any applications documents. Make error messages clear, giving as much information about what caused the error as possible, and use good grammar. Be cautious in inserting obscene error messages- if a disk containing such error messages accidentally gets released or sent to customers, it can cause a lot of trouble!

Utility - SZAP

SZAP (SuperZap) is a utility program that allows the experienced system programmer to examine and modify the contents of both system RAM and disk storage. SZAP is a powerful tool when used correctly and is capable of destroying the contents of disks and main memory when used incorrectly.

SZAP allows the system programmer to display a selected 256 byte page of main memory or a selected disk sector. SZAP displays the page in hexadecimal form, with an optional character display. The user may move an editing cursor through the selected page by using the cursor controls and so display the previous or next page of memory or disk. To modify data present in the display, the programmer enters either hexadecimal bytes or character strings. Additionally, SZAP can zero the contents of the page from the cursor to the end of the page in response to a single keystroke. SZAP makes modifications to main memory pages as the user enters the new data. A modified disk page (sector) is written to the disk when a request is made to display another page or to exit the program. The programmer may disable error checking and reporting when modifying the disk; this allows the systems programmer to attempt to reconstruct damaged disk directories and the like.

Running SZAP

The user must be in the enabled mode to execute SZAP. If the user tries to invoke SZAP when in the disabled mode, SZAP immediately returns control to Exec. When SZAP begins execution, the Control-Y vector is set to force exiting of the program (that is, a control-Y will cause you to exit from SZAP). SZAP displays a command summary and the version number. SZAP then waits for the device selection, ":" and a drive number or Ø for memory.

A SZAP command is either a single character or a hexadecimal number. (The single character commands appear below.) A number alone is an implicit command to SZAP to place that byte at the cursor position in the page that is displayed on the screen. A hexadecimal number is terminated by a space, whether it appears as the default data entry command or as a command argument.

The command characters recognized by version 2.3 of SZAP (and their associated functions) are:

Character	Function
Control-E	Exit after updating disk
Control-Y	Exit without updating disk
'	Begin text entry (single quote)
ESC	Toggle text display mode
ESC	Terminate text entry (escape)
:n	Select device n for display
/n	Select page n for display
I	Display indirect
!	Toggle error check on disk data transfers
Control C	Checksums 4 sectors.
Z	Zero data from cursor to end of page
RET	(Carriage return) Display next page
LF	(Line feed) Display previous page

(n = a hexadecimal number)

SZAP folds lower case letters to upper case when accepting commands or hexadecimal numbers as input.

The four cursor control arrows have the following functions:

Cursor Arrow	Function
UP	Move to beginning of previous line
DOWN	Move to beginning of next line
LEFT	Move cursor left one byte
RIGHT	Move cursor right one byte

SZAP displays the preceding page if the user moves UP or LEFT from the top of the screen display; it displays the next page if the user moves DOWN or RIGHT from the bottom of the display. The cursor appears in the upper left hand corner (byte ØØ) of a new page display.

Exiting SZAP: Control-E

To exit from SZAP the user types a Control-E. Any modified disk pages not yet written out will be written to the selected disk drive. Once the user exits SZAP, SZAP may not be restarted or reentered by way of the system commands START and REENTER. The user must re-invoke SZAP to use it again.

Hexadecimal Data Entry

Entering a hexadecimal number is an implicit command to SZAP to store the least significant eight bits of that number in the location pointed to by the cursor. The number is delimited by a space. Once the space has been entered, the selected byte is updated and the cursor moved to the right (the next location in the page). Typing errors are corrected by simply typing enough characters so that the least significant eight bits (the last two digits) of the number are correct. The strings 2, 9A02, and 3E002 all store the eight bit hexadecimal quantity 02.

Text Entry: The ' Command

A single quote symbol places SZAP in the text entry mode. All characters typed from that point on, with the exception of Control-Y and ESC (escape), will be entered into successive locations in the displayed page. This includes control characters such as carriage return and cursor control keys! ESC (escape) is used to terminate the text entry mode. To terminate text entry and exit from SZAP, the user types an ESC/ Control-E.

Toggling the Text Display: The ESC Command

The user may display the page in text form on the right portion of the screen. The ESC command enables or disables this display. When the text display is enabled, the frame address is not displayed, and those characters in the range 00 to 7F hexadecimal are displayed in their normal ASCII form; the values 80H through FFH display as blanks. NOTE: SZAP will not display the contents of the screen properly if you try to display the video board itself!

Selecting the Device: The : Command

The command character : (colon) followed by a hexadecimal number selects the device to be displayed and edited. Device zero denotes main machine memory, device 1 is disk drive 1, and so on. If the page currently on display represents a disk page that has been modified, that page will be written out to the proper device before the : command is processed. When a disk is edited, the frame number displayed in the upper right corner of the screen consists of the device number and a four digit hexadecimal number representing the sector address. (Note: the frame address does not appear when the text display is enabled.) When a disk is selected, sector 00 is automatically displayed on the screen as the current page.

Selecting the Displayed Page: The / Command

To display a particular page of the device being edited, the user types /nnnn, where nnnn is a hexadecimal number. This number selects the desired page. When device ØØ (main memory) is being edited, only the upper eight bits (the two most significant digits) of the last four digits of the number are used to select the page to be displayed. When the user is editing a disk (by using the : command), SZAP uses the entire number as a sector address. In either case, the number is terminated by a space. Typing errors are corrected by entering more digits, since only the last four hexadecimal digits of the number are used. For example, suppose the user enters /12345678. If a disk is being edited, sector number 5678H will be displayed (or at least SZAP will try to do so!). If main memory is being edited, as selected by "device" Ø, memory starting at location 56ØØH will be displayed.

Display Indirect: The I Command

The I command uses the sixteen bit address pointed to by the cursor in the frame currently being displayed as the new frame to display. This number is treated in standard 8Ø8Ø fashion, less significant byte first. If the user executes the I command while the cursor is pointing to the two bytes containing the number 3D Ø1, sector 13DH is displayed (if a disk is being edited). If main memory is being edited, page 1ØØH is displayed. If the page currently on display is a modified disk sector, that sector will be written out to the disk before the next sector is displayed.

Disabling Disk Error Reporting: The ! Command

WARNING: THIS IS DANGEROUS!

The ! command toggles a flag that enables or disables SZAP disk error detection and reporting. When the user inputs the ! command, this flag is displayed on the screen following the frame address. A value of ØØØØ indicates that errors will be reported; a value of FFFF indicates that errors will be ignored. It is sometimes useful to disable error detection and reporting when attempting to recover destroyed or unreadable disk directories. Although useful, this feature is dangerous-- use with extreme caution!

SZAP Display of Error Conditions

When SZAP encounters an error such as a disk transfer error, it clears the text display flag and displays the error code on the screen to the right of the frame address. The error code displayed is the one reported by the systemstores the checksum in the first byte of the current sector. Since SZAP has no way of

knowing when a sub-directory is being modified, it is the users responsibility to use this command after modifying a sub-directory so that it will be properly checksummed when checked by Gfid. If a sub-directory is modified and not checksummed, the first access of that sub-directory will report one of your favorite error messages -- Disk Directory Destroyed! When this happens use SZAP again and checksum the sub-directory using the ^C command. Main directories are automatically checksummed.

Zeroing the Page: The Z Command

The Z command zeroes the contents of the page on display from the cursor position to the end of the page. The previous contents of the page are lost. If the user accidentally gives this command while viewing a memory page, that page of memory is zeroed and the previous contents of that page are lost beyond recovery. If the user gives this command by accident while displaying a disk page, the two ways to prevent the (partially) zeroed sector from being written to the disk are to RESET THE SYSTEM by pushing the Load button (and be more careful from then on) or to type Control-Y to abort to Exec without updating the disk.

Displaying the Next Page: The RETURN Command

To display the next page, the user types a carriage return (CR or RETURN). If a disk is being edited, the next sector on the disk is displayed. If main memory is being edited, the next 256 byte page is displayed.

Displaying the Previous Page: The LINE FEED Command

The previous page will be displayed when a LINE FEED (LF) is typed.

Cursor Movement Using the Cursor Keys

The four arrow keys at the right of the keyboard are used to move the cursor up, down, left, and right within the page on display. Their use may also cause the previous or next page to be displayed if they are used to move off the top or bottom of the frame being displayed. The left and right arrows move the cursor left or right one byte. The up arrow moves the cursor either to the beginning of the current line or to the beginning of the previous line. The down arrow moves the cursor to the beginning of the next display line. The cursor keys in combination with LINE FEED and RETURN allow the user to move the cursor forward or backward one byte, one line (sixteen bytes), or one page (256 bytes).

Attempting to Reconstruct Directories

NOTE: A complete understanding of Section 2 of this manual is necessary, but may not be sufficient, in attempting to reconstruct a damaged disk directory. Making backup copies of important disks on a regular basis is much easier than trying to reconstruct a damaged directory.

When SZAP is instructed to read disk sectors Ø, 1, 2, or 3 of a disk device (the directory sectors), one sector is read into the internal editing buffer. When another sector is selected or any other event takes place that would cause that updated sector to be written out to the disk as part of the directory, SZAP follows the following procedure:

1) Each of the directory sectors Ø, 1, 2, and 3 of the selected device are read into the system directory one sector at a time. This means that four individual calls to Dio are made to read the directory, each requesting one sector, rather than one call to Dio requesting four sectors.

2) The sector updated by SZAP is copied to its correct place in the directory area.

3) The directory checksum is recomputed and stored in both the directory header and NFCK (see Section 3).

4) The four directory sectors are written out by one call to Dio.

If a disk directory is unreadable because of a checksum error on one of its sectors or some similar error, the following procedure is suggested, BUT NOT GUARANTEED:

1) Try reading the disk directory on other drives in the system.

2) Image the disk onto a scratch disk, and try to read that disk on other drives.

3) If (1) and (2) have not succeeded, use SZAP to examine the first four sectors of the disk to determine what type of problem exists and which sector or sectors are affected. You can also use Sniff to check for hard errors.

4) If the system can read sectors Ø through 3, chances are some program has gone wild and stomped the directory. In this case, the directory may be carefully reconstructed by hand, one sector at a time.

5) If a checksum or preamble error has occurred, making one or more sectors of the directory unreadable, the ! command may be used to disable error checking. You can then read the offending sector into memory, correct it by hand, and then write it back to disk. After this is done, use the ! command again to enable error checking. Re-examine the directory sectors to determine if there is a hard media error or if the error has been covered up.

6) After the disk has been "fixed" by performing (4), (5), or other procedures, the important files on it should be INDIVIDUALLY copied to other disks, and then the offending disk should be re-initialized by using the INIT command. This is very important, especially if a directory was rebuilt by hand! Such a reconstructed directory may have subtle errors in it that are not immediately apparent but that will cause a catastrophe the first time a file is deleted, the disk is packed, or a new file is created on the disk.

Morals on Reconstructing Directories

The following suggestions are made in the hope you will never need this section and the trauma that accompanies it:

1) Perform preventative maintenance on your system on a regularly scheduled basis. This should consist of running the memory test, cleaning the heads on disk drives, etc.

2) Log hard disk errors, such as checksum errors and preamble errors, recording both the name of the disk and the offending drive. This information may help in tracking down a bad drive, compatibility problems between drives, or bad media.

3) If possible, write-protect system disks.

4) Keep write-protected backup disks. The more important the contents of a disk is, the more often it should be backed up. When making backup copies, use a SET of disks for backup, and rotate the usage of the backup disks so that you write over the oldest backup copy each time. After making a backup copy, "Sniff" the disk, or use some other procedure to verify that the backup is good. Backup disks should be write-protected and stored away from other disks.

The general moral of this section is to treat your system like a "real computer." Regularly scheduled and performed preventive maintenance can detect problems before they cause

trouble. Regular backup of the file system leaves you less vulnerable if disaster does strike. Preventative measures take time and use up disks, but can minimize losses.

Utility Program SCOPY

SCOPY allows a user to copy files over old files (if they fit) and to copy files without using Dfn3 which avoids overlay swapping in the single user system and therfore speeds up the copying process. System files may be copied without resetting the system bit. SCOPY accepts commands in two formats. In order to copy one file, you can type the command in the format:

SCOPY pathname1 pathname2 {anychar}

where pathname1 is the source file, pathname2 is the destination, and the optional anychar is any printing character. If anychar is present, SCOPY will copy pathname1 over the already existing pathname2 if the file pathname2 is the same size or smaller than pathname1. If pathname2 is larger, so that it will not fit in the former space, SCOPY will give the error 'What?'. In order to copy more than one file, type

SCOPY

The program will display its version number and the prompt

SCOPY

You can then enter sets of pathnames followed by an optional anychar as above. SCOPY will copy the files specified and again prompt for another set of filenames. In order to end the process type RETURN. Errors shown by SCOPY are prefixed by T:, F:, or x:, where x is the drive number that caused the error. T: specifies the To or destination disk and F: specifies the From or source disk.

SCOPY is used in the command file created by FUTIL to do the actual copying of the files.

Utility Program FUTIL

FUTIL is a BASIC program that allows the user to copy, delete, or move some or all of the files in a directory. FUTIL works by generating a command file called 'foo'. FUTIL first asks the user for the volume number of the source disk. This is the drive number of the disk from which you wish to copy, delete or move files and must be in the range 1-9. The disk specified is then read, and each file is displayed on the screen as it is found. Next all of the directories found are displayed on the screen and the prompt:

M(ove), C(opy), D(elete), N(ew source) or E(xit)?

is displayed. Respond with the one character corresponding to the task you wish to perform and a RETURN.

Move will copy files from one directory to another and then delete the original files. After move is selected, FUTIL will display the source volume number and wait for the user to specify a subdirectory if desired. To move files from the main directory simply press RETURN. FUTIL will then ask for the Destination Directory. Type a legal pathname including all necessary angle brackets such as:

Source: **<1<** Destination: **<2<TRIX<**

The final angle bracket is not necessary but is allowed. FUTIL will then ask:

Copy all or part? (a or p):

If all files are to be copied, enter a, otherwise enter p. If p is entered FUTIL will ask:

move <1<Exec.OV to <2<TRIX<Exec.OV? (y, n, x):

'y' will cause the file to be entered into the command file. 'n' will cause the file to be skipped. 'x' will cause FUTIL to stop asking about any more files in the directory. FUTIL will ask for each file in the source directory until all have been done or the user types 'x'. FUTIL will then return to the options question to allow moving of another directory or any of the other options.

Copy will copy files from one directory to another in much the same manner as move.

Delete will delete files from the source directory.

New source will read another volume for copying. Keep in mind that when the command file is executed the volumes specified

must be in place as they were when the command file was generated. There is no time for disk shuffling.

Exit causes the command file to be closed and asks if the user wants to run the command file now. If so the file will be run immediately. Otherwise, the file can be run by typing 'foo'. 'foo' can also be edited to change the action taken when it is run. For instance, Delete can be specified to FUTIL and instead of running the command file, the user may edit foo and change DELETE to PRINT and insert PAGE between each of the commands and have a command file to print all the files in a directory.

FUTIL is provided in an unprotected and untokenized format so that the user can play with it and add features if desired.

Debugger

'RDB' is the debugger for PolyMorphic Systems System 88 computers. It represents a significant improvement in features and human engineering over the "front panel" for debugging assembly language programs. This short note describes the facilities available in this debugger, but is not a course in assembly language debugging or programming. It is strictly a note on debugging aids.

Using the debugger

The debugger desired is invoked by typing its name, RDB. It then prints its herald and exits. The user may then start any other program or set of programs. The debugger is initially entered by typing Control-U on the keyboard.

The debugger is a relocatable program that must be loaded prior to loading the program to be debugged. It locates itself just below MemTop and resets MemTop accordingly.

Warnings on the use of the debugger

The programmer should not use RST 1 or RST 7 instructions, and should not use interrupt level 1. The debugger assumes contol over these interrupt levels as well as interrupt level 7. The user should not single-step through the monitor root, especially those portions dealing with disk I/O and overlays. Breakpoints must be very carefully set in overlays. They won't work if the overlay is swapped out and reloaded.

Debugger Display

The debugger displays in the upper left corner the current values of the SP, HL, DE, and BC registers with the four bytes pointed to by them. The A and PSW registers are shown with the status of the C, Z, M, PE, and AC flags.

The left side shows the next 9 instructions to be executed with their addresses. The first corresponds to the program counter.

Debugger Commands

The commands to the debugger are one character, optionally preceded by a 16 bit hexadecimal number.

G Go

The G command is used to continue program execution from the current value in PC.

X Single step

The X command causes a single instruction to be executed. Note that if this is a DI instruction, a sequence of instructions may be executed before control is returned to the debugger. The argument, if present, is ignored.

I Indirect Display

The I command indirectly sets the window address. If in the numeric display mode, the window will display the the address presently pointed to in byte reversed form. If in the instruction display mode, the window is set to the address contained one byte past the current window position. This is handy for JMP or CALL instructions.

J Temporary breakpoint

The J command puts in a temporary breakpoint at the instruction following the current one to be executed. When single stepping a program and encountering a CALL ####, the J will execute the subroutine and return to the debugger when the subroutine returns. If a normal breakpoint is encountered before the temporary, the temporary one is lost.

M Move data

The M command moves a block of data. When prompted, enter the starting address of the source, the ending address of the source and the starting address of the destination. If no starting address is given, the move is aborted. Note that data cannot be moved on top of itself.

O Output

The O command sends a byte of data to an output port.

Q Quit

The Q command restores MEMTOP, sets UVEC to point to Ioret, and UCHR to Ø, and warm starts the Exec. If a BASIC program is expected to continue functioning, don't Quit the debugger, just remove the breakpoints and Go.

R Restore screen

The R command restores the screen display until any other key is pressed.

S Search

The S command searches for a string of up to 1Ø numbers. Enter ## ## ## where each number is terminated by a space. The last byte is terminated by RETURN. Enter the starting address and press RETURN. The window will point to the match. "C" will continue to the next match. The string will always be found at least once since it is saved in a buffer in the debugger.

T Set MEMTOP

The T command allows the changing of MEMTOP. Enter the value of the desired MEMTOP and press space. If no number or zero is given, MEMTOP is not changed.

V View

The V command sets the window address if preceded by a number. Otherwise it advances the window address either 8 or 64 bytes for instruction or number mode, respectively.

W Warm start Exec

The W command sets PC to Ø4Ø3H and does a GO.

Z Fill memory

The Z command will fill memory with a byte from a starting address to an ending address. If no byte is given the fill is aborted. Note the distinction with the ;Z command.

Arrows Move window pointer

The arrow keys move the window pointer:

Up Up 8 lines in instruction mode. Up 1 line (8 bytes) in the number mode.

Down Down 1 line (8 bytes) in the number mode. If in the instruction mode, it may show trash as it can't disassemble backwards.

Left Left one byte in the number mode. Up 1 line in the instruction mode.

Right Right one byte in number mode. In the instruction

mode, it may show trash as it can't disassemble backwards.

xxxx! Set breakpoint

The ! command sets a breakpoint at the location specified. Up to 8 breakpoints may be active at any time. The instruction breakpoints, and the instuction on which the stop will occur are displayed in the upper right portion of the screen while the debugger is active. If the address given is already breakpointed, no action is taken. Care should be exercised in setting breakpoints in the overlay or other system areas.

% Clear breakpoints

The % command clears all instruction breakpoints, and the instruction breakpoint display on the screen disappears. No facilities are provided for clearing only one breakpoint. All breakpoints are cleared at once.

xxxx: Modify register pair

The : command allows the user to modify the contents of general register pairs (no facilities are provided for changing a single register). The various commands are:

```
xxxx:P   Set PC
xxxx:H   Set HL
xxxx:D   Set DE
xxxx:B   Set BC
xxxx:A   Set A, PSW
```

NOTE: No facility is provided for changing SP.

xxxx;I Display in instruction format

The ;I command sets the display block format to the instruction mode. The contents of the memory area set by the V command will be displayed as instruction (see also ;N, ;Z, V). If an argument is given, it will be used as the display start location. If no argument is given, the current diplay location will be used.

xxxx;N Display in numeric format

The ;N command sets the display block format to numeric. The contents of the memory area set by the V command will be displayed in hexidecimal, with the ASCII character equivalents of these memory locations displayed to the right of the numeric display. Arguments are the same as the ;I command.

xxxx;P Display input port mode

The ;P command sets the window to display the input ports selected. Note this mode will not display ports 18H through 1BH, as these are the single step ports.

;Z Clear display block

The ;Z command clears the memory block display. This speeds up the display.

xxxxV View memory

The V command, in conjunction with the ;I and ;N commands allows the user to diplay a selected block of memory in either numeric/character, or instruction format. If an argument is given with the V command, that location will become the start of the display, using the current format. If no argument is given, the locations following the currently displayed block will be displayed.

. Entering hexadecimal data

The . command initiates the entry of hexadecimal, 8 bit data into the memory block shown by the V, ;I and ;N commands. After the . has been recognized, hexadecimal data is input from the keyboard and the least significant 8 bits are stored in successive locations. Data input is terminated by ESC. The data byte (if any) preceeding the ESC is not stored into memory. Any non-hexadecimal character terminates an 8 bit quantity, displays the stored data, and increments the pointer. The entry pointer is reset by the V, ;N or ;I commands. The arrow keys may be used to move to a specific memory byte.

' Enter text

The ' command allows the user to enter text into the memory block displayed by the V, ;I or ;N commands. After the ' is recognized, text is accepted from the keyboard and stored into successive memory locations until an ESC is entered. The text is echoed to the screen with control characters displayed as special symbols. The arrow keys may be used to move to a specific memory byte.

Authorization Overlay - Auth.OV

The Auth overlay is an optional component of the System 88 that requires users to give an authorized name and password before using the system. Systems containing Exec versions number 52 or later perform this authorization process if the Auth overlay is present on the system disk. This authorization process is not meant to be "totally secure," or to totally prevent unauthorized use of the system; it IS meant to make unauthorized use of the system difficult.

Signing Onto the System

The system checks for the Auth overlay during every system boot. If the overlay is on the system disk, the system invokes it with a function code of ØØ, which it passes to Auth in the Accumulator. The function code of ØØ tells Auth to ask for a user name and password. Auth prompts the user to enter his or her name; then Auth checks the name against an internal list of authorized names. If the name is present, Auth then prompts the user to enter a password. The password does not echo to the screen as the user enters it; instead, question marks appear. If Auth does not find the name on its authorization list, or if the password is wrong, it displays an error message to the user; the system then goes into a loop after disabling interrupts and zeroing part of memory. At that point the user must re-boot the system to try again.

The user name may be up to sixty characters long, and must be terminated by a carriage return. When processing the password, Auth reads up to sixty characters terminated by a carriage return; however, it uses only the first sixteen in the validation process. If the password contains less than sixteen characters, Auth automatically appends nulls to fill it out to that length. The initial greeting message, the password request message, and the failure message are in the system error message writer, Emsg. The systems programmer may use the EMEDIT utility described in Section 4.1 to tailor these messages.

The Exec Auth Command

With the authorization processor, a new command, Auth, is added. This command, which must be given in the enabled mode, allows the system user to add, delete, and list authorized users, as well as change passwords for users.

The commands to Auth are single characters, as follows:

Command Character	Auth Function
X	Exit Auth, warm-starting the system
A	Add user to authorization list
D	Delete user from list
C	Change password for user
L	List names of authorized users

Commands that modify the authorization list (Add, Delete, and Change) cause the system to re-write the overlay to the system disk at the time the command is processed; therefore, the disk must not be write-protected when these commands are given.

Exiting Auth: The X Command

The X command causes Auth to exit, warm-starting the system.

Adding Users: The A Command

The A command is used to add users to the authorization list. Auth first asks for the user name. If this name already appears in the user list, Auth gives an error message and aborts the A command. If the name does not appear on the list, Auth requests the password. The password echoes to the screen as a sequence of question marks (?). The name and password are entered onto the user list, and the overlay is written back to the disk.

Deleting Users: The D Command

The D command is used to remove a user name from the list. The user is first asked for the name, which must be on the list, or an error message results. The user is then asked for the password. This must match the password in the file, or an error message is given, and the name is not removed from the list. If the password matches, the name is removed and the overlay re-written to the disk.

Changing Passwords: The C Command

The C command is used to change a user's password. The user is first prompted for a name, which must appear on the authorization list or an error message is generated. The old password must then be entered, and must match that currently in the file. A new password is then asked for; it replaces the old password in the file, which is then re-written to disk.

Listing User Names: The L Command

The L command displays the list of authorized users on the screen. Passwords are not displayed.

Installing Auth on the System 88

To install the authorization checker, copy the file Auth.GO from the disk included with the System Programmer's Guide to the desired system disk as file Auth.OV. Note that for Auth to be used, the Exec on the system must be version 52 or later. Using the Exec Auth command, authorize one or more users. No users are authorized in the file as it is shipped. The SZAP utility may be used to set the system bit (see Section 2) on the Auth.OV file to insure it is not deleted, or the SetSys command may be used (see Section 3) to make all files on the system disk system files. If the Exec is version 52 or later and the Auth.OV file resides on the system disk, whenever the system is booted, the user must enter a name and password before the system may be used.

How Auth Connects to the Exec

In the initialization process, before the Exec looks for the INITIAL file, it checks to see if the file Auth.OV exists on the system disk. If this file exists, it is called by an Ovrto (see Section 3) with a function code of ØØ in A. The Auth overlay "disconnects" PVEC and UVEC and sets the system in disabled mode by clearing SCHR to prevent it from being interrupted by the user. If the user is authorized, the Auth overlay returns. If the user is not authorized, the remainder of the overlay area is zeroed, and the system hangs.

User-Written Auth Overlays

For more security, or for other reasons, the systems user may want to provide a custom Auth overlay. This overlay should be written to conform to the conventions described for overlays in this manual. As noted before, since Auth is called very early in the boot process, MEMTOP has not been set, so no system services that depend on this cell should be used. The user written Auth overlay should recognize two function codes passed in the A register:

Code in A	Auth Function
ØØ	Verify user authorization
Ø1	Exec Auth command given

Storage of Names and Passwords in Auth

The list of authorized user names and passwords is stored as part of the Auth overlay. The password associated with each name is stored in an encrypted form. The encryption used is simple-minded and is present as a hindrance in obtaining the passwords of others rather than as absolute security. In the validation process or in validating a password for the C (change) or D (delete) commands, the password entered by the user is encrypted and compared to the encrypted entry within Auth. This insures that the "clear text" of the password is not left in memory for very long.

"I forgot my password," or, How to Break Auth

All that is required to "break" Auth is a system disk that does not have Auth connected and a system with more than one disk drive. The "unprotected" system may be booted and used to delete the copy of Auth from the protected system disk. If the copy of Auth is marked a system file, protecting it from deletion and renaming, Szap or a similar program may be used to clear the system bit and then delete Auth. A different method is to copy everything from the protected system disk except Auth.

Once users are authorized, those authorizations may not be changed or removed without knowing the associated passwords. A new, "clean" copy of Auth may be installed, without any authorizations, and then user names added. It should be possible for the persistent user to break the encryption used on the passwords, but no details on the algorithm used will be given here.

Suggestions for Using Auth

User names may be as long as desired, up to sixty characters. A password should be easy for the user to remember. A password that is quick and easy to type is desirable if others are going to be watching you type your password.

Remember: if you forget your password, it is very difficult to recover. To be effective at a computer installation, all system disks should have Auth on them, including backup disks. The Auth processor is NOT meant to provide "absolute" security from unauthorized use of the system; it is meant to hinder unauthorized use.

Utility Program SPACE

This utility displays the number of bytes remaining in a directory. It is invoked as follows:

SPACE 4

for the main directory on unit #4

SPACE 5<TRIX

for the TRIX subdirectory in unit #5. If no disk number is given, the space remaining in the SysRes disk directory will be displayed.

Utility Program WAIT

This utility may be called in a command file to cause the execution of the command file to pause. It displays the word "Waiting...", until the operator presses a key.

Utility Program TWID

This utility is used to view, change, and print the values in an Assembler Symbol Table File. Since it operates in a manner similar to Emedit, the error message editor, refer to its instructions.

Utility Program COMPARE

This utility compare two files, byte by byte, that are specified on the command line. If there are any mismatches, it displays the byte count from the start of the file and the two bytes that differ. It is invoked as follows:

COMPARE Filename1 Filename2

It asks whether the comparison is to be sent to the printer in addition to the screen.

Utility Program COMP-DISK

This utility compares two disks and can be called after a disk image to verify the copy. It is invoked with only its name and it prompts for the source and destination disks. If the disks are not the identical, the program displays "Verify error!"

Utility Program CLEAN

This utility reinitializes the main directory of a disk. It can only be called from the ENABLEd mode of Exec. The number of the disk unit must be on the command line. The SysRes disk cannot be "Cleaned".

Utility Program ARISE

This utility resets the delete bit on a file entry in a directory, restoring the file to active status. The program is used in conjunction with the Exec command DLIST to determine which copy of the deleted file is to be undeleted. It is invoked as follows:

```
ARISE Filename N
```

where N is a decimal number corresponding to the file entry desired. If the same filename already exists in the directory, an error message results.

Utility Program DIRCOPY

This utility copies all files within a directory to another directory, and all of its subdirectories. It is invoked as follows:

For main directories on unit 4 to unit 5:

```
DIRCOPY 4 5 *
```

is essentially an IMAGE and PACK in one operation.

For SubDirectories Sub1 to Sub2:

```
DIRCOPY <d<Sub1 <d<Sub2 *
```

where the optional "*" means replace the file if presently in the directory, "d" is the option disk unit number. Many combinations are possible, ie. copying from a main to subdirectory, from two subdirectories, etc.

If the "*" is not on the command line and a file with the same name and extension exists in the destination directory, the program will pause saying:

```
Output file already exists!
Should I delete it? (Y or N)
```

If the answer is "N", then it asks:

```
OK, give me a new name for the file I'm backing up.
Filename:
```

If the answer is "Y", then it replaces the file with the one from the source directory.

The replace function is performed as follows: If the files have the same sector length, the source file is written over the destination file on the disk. If the files don't have the same sector length, the destination file is marked deleted and the source file is copied onto the destination disk at the next available disk address.

If the destination disk becomes full, the message:

```
Destination disk is full. Insert new disk,
then press RETURN to continue.
```

The copying process continues with the first file in the current directory being processed.

If the primary function being performed is a backup of a complete disk rather than a general subdirectory copy, the program BACKUP supplied on the system disk should be used. It checks the "New" bit of each file and resets it after a successful copy, thus not repeating the copying process with the first file of the directory endlessly. BACKUP otherwise operates identically to DIRCOPY.

Utility Program RECOVER

This utility is used to recover a file that is on a disk, but not accessible due to a "crashed" directory, or past the "known" area of the disk. Use SZAP to locate the file starting and ending disk addresses. Then invoke the program by name. It asks the following questions:

```
Enter Disk Drive Number: 5
Enter Starting Sector Number: 30C
Enter Ending Sector Number: 31A
Enter Program Load Address: 3200
Enter Program Start Address: 3200
Enter New Filename with Extension: <4<SUB<FILENAME.GO
```

A typical response by the user is in bold face.

Section 5

The System 88 Printer Driver

The printer driver is an integral component of the System 88. It provides an interface between the printer and Exec commands, BASIC programs, the formatter, and user programs. The goal of this section is to describe the System 88 Printer Overlay and show how the system interfaces to it. With this information, users may write routines to interrogate or setup the printer dynamically from user programs.

Prnt.OV Overlay

These are the function codes that can be passed in A to the Prnt overlay and conditions for other registers if applicable.

~~1 = Hookup default printer.~~
2 = Printer command. HL points at string which is printer name or command.
3 = Turn off logging.
4 = Turn on logging.
5 = Show page parameters on screen.
6 = Set page parameters from keyboard.
7 = Set page parameters from registers.
B = lines per page.
C = characters per line.
D = top margin.
E = bottom margin. (line number from bottom of page)
H = edge offset.
8 = Get page parameters into registers. Registers returned same as 7, except for E, which is the number of the last printing line.

Wormhole 5

Wormhole 5 filters TAB, LF, VTAB, FF, and CR.
LF checks for top and bottom margin.
CR does only CR, no line feed.
VTAB does form feed if NOT at top of form.
FF does form feed.

All of the above actions are taken according to the current printer specifications as to margins and understanding of TAB's and FF's.

If the following characters are sent to WH5 they will return the following information in A:

80H = current line number.
81H = current character position.
82H = lines per page.
83H = characters per line.

The address for Wormhole 5 is 0C34H for the Single user and 2E20H for the Twin. In BASIC the address is obtained as follows:

```
100 X=PEEK(2)+20 \ REM X=Address of WH5
```

Serial I/O Driver - Sio.PS

The standard system loads the code in the Sio.PS file into the printer driver block starting at address 3000H in the Single User or FD00H in the Twin. This code actually handles the interrupt level processing of characters to and from the serial printer.

Direct entry to Sio.PS

There are occasions when an applications program needs to send escape code sequences directly to the printer without having them trapped and/or modified by the printer driver. This may be accomplished by calling directly the serial I/O driver code.

For assembly programs, load the character to be sent into the A register, set B to 1 and call either 3000H for the single user or FD00H for the twin.

For BASIC applications, either of two methods may be used. The first copies the assembly method, as follows:

```
100 IF PEEK(5)=0 THEN X=12288 ELSE X=64768\ REM Select User
110 Z=CALL(X,A,256)\ REM Send ASCII code in A
120 RETURN
```

The other method defines a special device driver attached to file channel 3, as follows:

```
10 REM Buffers for assembly code
20 DIM A$(1:10)\A=MEM(A$)
30 DIM A1$(1:1)\A1=MEM(A1$)
40 DIM U$(1:2)
50 REM Determine user
60 U$=CHR$(0)
```

```
70 IF PEEK(5)=0 THEN U$=U$+CHR$(48) ELSE U$=U$+CHR$(253)
80 REM Load assembly vector code
90 A$=CHR$(120)+CHR$(6)+CHR$(1)+CHR$(195)+U$
100 A1$=CHR$(201) \ REM Return code
110 REM Define Special File Channel, Output only
120 FILE:3,DEF,A1,A,A1
```

To send characters through this device driver, merely do the following:

```
200 PRINT:3,CHR$(27),CHR$(13),CHR$(5)
```

This sends an Escape, Carriage Return, Ctrl-E to the printer.

Additional information on the System 88 Printer Driver is contained in the System 88 User's Manual, Appendix H.

Section 6

Error Messages

This section contains a list of the System 88 error messages in numerical order. The first section is the Emsg.OV error messages. The second section is the Berr.OV error messages.

Emsg Error Messages

The following error messages are generated by the Emsg error message writer:

> Messages with error code Ø1 are generated by Dio as a result of either bad parameters passed for disk transfer or an error in attempting the disk transfer.

Error code Ø1Ø1
DIO says: Bad parameters!

Error code Ø1Ø2
Hard error! Preamble bad!

Error code Ø1Ø3
Checksum error!

Error code Ø1Ø4
Verify error!

Error code Ø1Ø5
Write protected!

Error code Ø1Ø6
No disk in drive, or door open!

Error code Ø1Ø7
No controller for that device.

Error code Ø1Ø8
DIO says: Data transfer error!

Error code Ø1Ø9
No such drive.

Error code Ø1ØB
Seek error!

Error code Ø11Ø (Single user only!)
System PROMS must be version 74 or later!

Error code Ø111
I can't do that to the System drive!

Error code Ø112
I can't, too much data for destination disk.

Error code Ø1CØ (Twin only!)
Nothing assigned to that channel!

Error code Ø1C2 (Twin only!)
That device is busy.

Error codes Ø1DØ to Ø1D6 are issued by the Volume Manager.

Error code Ø1DØ
That unit is already connected.

Error code Ø1D1
That volume is already connected.

Error code Ø1D2
I can't find that volume.

Error code Ø1D3
No volumes available

Error code Ø1D4
Only 1 volume on that device.

Error code Ø1D5
No device driver.

Error code Ø1D6
Device definition block bad.

Error codes Ø2 are issued by the Exec.

Error code Ø2Ø1
I can't run that file

Error code Ø2Ø2
Nothing to run!

Error code Ø2Ø3
DONT what?

Error code Ø2Ø4
What?

Error code Ø2Ø5
I don't know what to do with that file

Error code Ø2Ø6
I don't have enough memory to do that!

Error code Ø2Ø7
I can only pack entire disks.

Error code Ø2Ø8 (Twin only!)
You need a video display to do that!

Error code Ø2Ø9 (Twin only!)
That disk is not a Two User System Disk!

~~Error codes Ø3 are issued by Look.~~

Error code Ø3Ø0
I can't find that file

Error code Ø3Ø1
I can't access that device!

Error code Ø3Ø2
Preamble error - directory unreadable!

Error code Ø3Ø3
Checksum error - directory unreadable!

Error code Ø3Ø6
No disk in drive, or door open!

Error code Ø3Ø7
No controller

Error code Ø3Ø9
No such drive!

Error code Ø3ØB
Seek Error!

Error code Ø3C2 (Twin only!)
That device is busy.

Error code Ø3FF
Disk directory destroyed!

Error codes Ø5, Ø6 and Ø7 are issued by Gfid, the text editor, and the assembler.

Error code Ø5ØØ
Bad disk identifier

Error code Ø5Ø1
Name too long

Error code Ø5Ø2
Illegal extension

Error code Ø5Ø3
Name null or weird!

Error code Ø5Ø4
The directory is full

Error code Ø5Ø5
I can't write: the disk is full

Error code Ø5Ø6
I can't rename across directories: use copy

Error code Ø5Ø7
No new extension given

Error code Ø5Ø8
I can't do that to a system file!

Error code Ø5Ø9
"<?<" is not allowed here

Error code Ø5ØC
I can't copy directories

Error code Ø6ØØ
That file already exists

Error code Ø6Ø1
That file does not exist

Error code Ø7Ø1
Output file not specified

Error code Ø7Ø2
Output file already exists

Error code Ø7Ø3
Input file not specified

Error code Ø7Ø4
I can't edit that file!

Error code Ø7Ø5
Input file does not exist

Error code Ø7Ø6
I can't have two OUTPUT files open on the same drive!

Error code Ø7Ø7
That drive already has an output file opened to it!

Error codes Ø9 are issued by the Prnt.OV.

Error code Ø9Ø1
Printer has not been defined

Error code Ø9Ø2
That printer has already been defined

Error code Ø9Ø3
Please specify a printer name!

Error code Ø9Ø4
I can't change that!

The following codes are the result of catastrophic system failure.

Error code ØDØØ
I can't find that overlay!

Error code DEAD
SYSTEM FAILURE: CHECKSUM CHANGED!

Berr.OV Error messages

The following messages are generated by Berr, the BASIC error message writer. Remember that if a Berr message ends in e, the e will be expanded to "error" when displayed.

Error code Ø4ØØ
Syntax e

Error code Ø4Ø1
Syntax e

Error code Ø4Ø2
Subscript e

Error code Ø4Ø3
Bad argument e

Error code Ø4Ø4
Dimension e

Error code Ø4Ø5
Function definition e

Error code Ø4Ø6
Out of bounds e

Error code Ø4Ø7
Type e

Error code Ø4Ø8
Format e

Error code Ø4Ø9
I can't find that line

Error code Ø4ØA
FOR-NEXT e

Error code Ø4ØB
RETURN without GOSUB

Error code Ø4ØC
Division by zero

Error code Ø4ØD
Function definition e

Error code Ø4ØE
Missing matching NEXT

Error code Ø4ØF
Read e

Error code Ø41Ø
Oops...BASIC goofed!

Error code Ø411
Oops...BASIC goofed!

Error code Ø412
Input e

Error code Ø413
Out of memory

Error code Ø414
I can't do that directly

Error code Ø415
Argument mismatch e

Error code Ø416
Length e

Error code Ø417
Overflow e

Error code Ø41A
Can't continue!

Error code Ø41B
That's not a BASIC file!

Error code Ø41C
Nothing to save!

Error code Ø41D
That channel not open!

Error code Ø41E
That channel not open for input

Error code Ø41F
That channel not open for output

Error code Ø42Ø
End of file on that channel

Error code Ø421
That program is for a different version of BASIC!

Error code Ø422
That program must be saved in tokenized format

Error code Ø423
That record is past the end of the file

Error code Ø424
I can only do that to a disk file

Error code Ø425
End of file on that channel

Error code Ø426
Type error on READ

Error code Ø427
That's not a BASIC data file

Error code Ø428
MAT subscript e

Error code Ø429
I can't do that to a protected file!

Error code Ø430
Too many digits for hardware!

Error code Ø431
Renumbering e

Error code Ø432
The minimum allowable precision is 6.

Error code Ø433
The maximum allowable precision is 26.

Error code Ø44Ø
....LOAD interrupted

Error code Ø4FF
I can't do that to an OUT file

Section 7

Sample System Overlay

The following assembly listing gives a sample of the form that a system overlay takes. The assembly listing for Emsg.OV, the system error message overlay shows the use of macros in the assembler, the REFS and REF statement, and conditional assembly for single and Twin systems. Using the REF statement makes the program easier to update if a system symbol changes; re-assembly is all that is required. For error message overlays, note the pointer at OVRLY+7, which points to the start of the text. This pointer is used by the error message editor, Emedit, to access the text.

```
                ; The error message handler.
                ;
                ; Last updated:
                ;       7/18/79 RTM     Added support for DBARF
                ;                       bit in EFLG1
                ;       10/17/79 RTM    Two user system
                ;       12/28/79 RTM    Disaster recovery!
                ;       2/26/80  RTM    Interlock msg
                ;       4/15/80  RTM    Preserves all regs.
                ;       7/20/80  BFS    Integrate 1 and 2 users
                ;                       sources.
                ;       01/22/81 BFS    Rip out Auth put in HD
                ;                       errors.
                ;       02/03/81 BFS    Changed drive to unit
                ;                       in 1D0H.
                ;
                ; We are invoked with the error code expected in DE.
                ; We put it into ERROR, moving the previous contents
                ; to LERR first, and then look for a message associated
                ; with that error number, and spits it out. If we don't
                ; find the text, we display an "I don't know" and split.

                        MACLIST 0
                        REFS    SYSTEM
                        REF

              +         overlay 'Emsg',GO

2007  8720               DW      ETXT    ; pointer for error message
                                          ; editor

                ; And away we go....

2009  F5        GO      PUSH    PSW
200A  C5                PUSH    B
200B  D5                PUSH    D
```

```
200C  E5              PUSH    H
200D  3AC92D          LDA     EFLG1
2010  E620            ANI     DBARF   ; do we flush input?
2012  CC1B04          CZ      Killi   ; Yes, and abort
                                      ;command files
2015  2A9A2D    Go1   LHLD    ERROR
2018  229C2D          SHLD    LERR    ; move over, please
201B  EB              XCHG            ; new one
201C  229A2D          SHLD    ERROR   ; plotz.
201F  3AC92D          LDA     EFLG1
2022  F640            ORI     EERR
2024  EE40            XRI     EERR    ; clear it.
2026  32C92D          STA     EFLG1   ; for recovery.

                ;     Convert 306-30B to 106-10B to save text space.

2029  7C              MOV     A,H     ; Check high byte for 3.
202A  FE03            CPI     3
202C  C23C20          JNZ     Cont    ; Don't convert.
202F  7D              MOV     A,L     ; it's a candidate check more.
2030  FE06            CPI     6
2032  DA3C20          JC      Cont    ; less than 6
2035  FE0C            CPI     0CH
2037  D23C20          JNC     Cont    ; greater than 0BH
203A  2601            MVI     H,1     ; change to 1

203C  118720    Cont  LXI     D,ETXT  ; start of the text
203F  EB              XCHG

                ; We now search the text. It is in the form code,sub
                ; followed by the message, followed by zero. The end
                ; of the list is an FF byte.

2040  7E        EFND  MOV     A,M
2041  FEFF            CPI     0FFH    ; end hit?
2043  CA7B20          JZ      Nope    ; jmp/yup, no such msg.
2046  BA              CMP     D       ; is this the one, then?
2047  C25120          JNZ     EFN1    ; jmp/nope.
204A  23              INX     H
204B  7E              MOV     A,M
204C  BB              CMP     E       ; this the one?
204D  CA5C20          JZ      Yup     ; jmp/yes, go print it
2050  2B              DCX     H
2051  23        EFN1  INX     H
2052  23        EFN2  INX     H
2053  7E              MOV     A,M
2054  B7              ORA     A
2055  C25220          JNZ     EFN2
2058  23              INX     H       ; point past the stinker.
2059  C34020          JMP     EFND    ; find this one's end.
```

```
                ; Found it.

205C  23        Yup     INX     H       ; point past subcode, dummy...
205D  CD0C04            CALL    Msg     ; display it
2060  C36400            JMP     Ioret   ; split

                ; Didn't find it.

2063  3F204E6F  NT      db      '? No message for error ',0
2067  206D6573
206B  73616765
206F  20666F72
2073  20657272
2077  6F722000
207B  216320    Nope    LXI     H,NT
207E  CD0C04            CALL    Msg
2081  CDD103            CALL    Deout   ; print the code on the way out.
2084  C36400            JMP     Ioret

                ; Now comes the text. Macros make life simple...

                m       MACRO
                        DB      (#1H SHR 8 AND 0FFH),(#1H AND 0FFH)
                        DB      #2,0
                        ENDM

3087            ETXT    DS      0

              + m 101,'DIO says: Bad parameters!'
              + m 102,'Hard error! Preamble bad!'
              + m 103,'Checksum error!'
              + m 104,'Verify error!'
              + m 105,'Write protected!'
              + m 106,'No disk in drive, or door open!'
              + m 107,'No controller for that device.'
              + m 108,'Data transfer error!'
              + m 109,'No such drive.'
              + m 10B,'Seek error!'
FFFF                  IF USERS=1
              + m 110,'System PROMS must be version 74 or later!'
                      ENDIF
              + m 111,'I can''t do that to the System drive!'
              + m 112,'I can''t, too much data for destination disk.'
0000                  IF USERS=2
                       m 1C0,'Nothing assigned to that channel!'
                       m 1C2,'That device is busy.'
                      ENDIF

                ;
                ;       01Dx are Volume Manager errors.
                ;
```

```
          + m 1D0,'That unit is already connected.'
          + m 1D1,'That volume is already connected.'
          + m 1D2,'I can''t find that volume.'
          + m 1D3,'No volumes available.'
          + m 1D4,'Only 1 volume on that device.'
          + m 1D5,'No device driver.'
          + m 1D6,'Device definition block bad.'


          + m 201,'I can''t run that file'
          + m 202,'Nothing to run!'
          + m 204,'What?'
          + m 205,'I don''t know what to do with that file'
          + m 206,'I don''t have enough memory to do that!'
          + m 207,'I can only pack entire disks.'
0000                IF USERS=2
            m 208,'You need a video display to do that!'
            m 209,'That disk is not a Two User System Disk!'
                ENDIF

          + m 300,'I can''t find that file'
          + m 301,'I can''t access that device!'
          + m 302,'Preamble error - directory unreadable!'
          + m 303,'Checksum error - directory unreadable!'
0000                IF USERS=2
            m 3C2,'That device is busy.'
                ENDIF
          + m 3FF,'Disk directory destroyed!'

          + m 500,'Bad disk identifier'
          + m 501,'Name too long'
          + m 502,'Illegal extension'
          + m 503,'Name null or weird!'
          + m 504,'The directory is full'
          + m 505,'I can''t write: the disk is full'
          + m 506,'I can''t rename across directories: use copy'
          + m 507,'No new extension given'
          + m 508,'I can''t do that to a system file!'
          + m 509,'"<?>" is not allowed here'
          + m 50C,'I can''t copy directories'

          + m 600,'That file already exists'
          + m 601,'That file does not exist'

          + m 701,'Output file not specified'
          + m 702,'Output file already exists'
          + m 703,'Input file not specified'
          + m 704,'I can''t edit that file!'
          + m 705,'Input file does not exist'
          + m 706,'I can''t have two OUTPUT files open on '
```

```
                    'the same drive!'
                + m 707,'That drive already has an output file '
                    'opened to it!'

                + m 901,'Printer has not been defined'
                + m 902,'That printer has already been defined'
                + m 903,'Please specify a printer name!'
                + m 904,'I can''t change that!'

                + m ØDØØ,'I can''t find that overlay!'
                + m ØDEAD,'SYSTEM FAILURE: CHECKSUM CHANGED!'
                 ; End of stuff for now.

271F  FFFF              DW      -1              ; insurance....
2800                    ORG     2800H

                        END

Error total =     Ø
```

Macros defined in this assembly:

db gfid m overlay

Labels defined in this assembly:

UGS	2DFC	BUSIES	ØC6E	CBUF	2CØØ	CMDA	2D8C
CMDD	2D89	CMDF	2D88	CMDN	2D8E	CMDP	2D8A
CMND	2D4Ø	CMPTR	2DC7	Ckdr	Ø433	Command	ØC4C
Cont	203C	DBARF	ØØ2Ø	DEFPATH	2E27	DEOUT	Ø3D1
DEVMASK	ØØØF	DONT	2D9Ø	DRVADTAB	ØC7E	Deout	Ø3D1
Dhalt	Ø4Ø9	Dio	Ø4Ø6	DioA	ØC66	DioBsy	ØC6C
DioDn	ØC6B	DioDrv	ØC69	DioHL	ØC67	DirAddr	2EØ2
EERR	ØØ4Ø	EFLG1	2DC9	EFLG2	2DCA	EFN1	2Ø51
EFN2	2Ø52	EFND	2Ø4Ø	EIC	ØØ8Ø	ERROR	2D9A
ETXT	2Ø87	EXECSP	2DAF	Err	Ø4ØF	FILE	2DCB
Flip	Ø42D	Flush	Ø41E	Fold	Ø42A	GO	2ØØ9
Gol	2Ø15	Gover	Ø415	Iexec	Ø436	Ioret	ØØ64
JOBST	2D9E	KBD	ØØ18	KBEX	2D86	KBIG	2D84
KBIP	2D82	KBUF	2DØØ	Killi	Ø41B	LERR	2D9C
LUSER	2DC6	Look	Ø421	MEMTOP	2D8Ø	MTO	2DA2
MUNG1	2DA7	MUNG2	2DA9	MUNG3	2DAB	MUNG4	2DAD
MemAdd	ØC49	Msg	Ø4ØC	NDRIVES	2D9F	NFA	2EØØ
NFCK	2DA1	NFDIR	2DAØ	NT	2Ø63	Nope	2Ø7B
ONCE	2DC5	OVBC	2DC1	OVDE	2DBF	OVENT	2Ø04
OVHL	2DBD	OVMEM	2E53	OVNM	2DB6	OVPSW	2DC3
OVRLY	2ØØØ	Ovrto	Ø412	PATH	2E04	POS	ØC0E
PVEC	2D93	Pages1	ØC4B	Rlgc	Ø43Ø	Rlwe	Ø427
Rtn	Ø418	Runr	Ø424	SBRK	2D91	SBUF1	28ØØ
SBUF2	29ØØ	SBUF3	2AØØ	SBUF4	2BØØ	SCEND	ØC1E

SCHR	2D98	SCREEN	1800	SCRHM	0C1F	SINT	2DB3				
SRA1	0C10	SRA2	0C12	SRA3	0C14	SRA4	0C16				
SRA5	0C18	SRA7	0C1C	STACK	1000	SYSRES	2D92				
TIMER	0C00	UBRK	2D97	UCHR	2D99	USER	3200				
USERS	0001	USTATS	2DB1	UVEC	2D95	VCBTAB	0C63				
VERLOC	0439	Version	0081	WAKEUP	0C1A	WH0	0C20				
WH1	0C24	WH2	0C28	WH3	0C2C	WH4	0C30				
WH5	0C34	WH6	0C38	WH7	0C3C	WH8	0C40				
WH9	0C44	Warm	0403	Yup	205C						

Section 8

The System 88 Boot Sequence

This section describes in detail the boot sequence for single and Twin systems. The boot sequence followed by the System 88 is different from that usually found in disk based computer systems. Traditionally, the boot sequence usually involves reading a track from a predefined disk and disk address into memory and jumping to it. This function is provided by a small ROM.

When System 88 boots, rather than loading a predetermined number of sectors from a disk, it selects the boot volume, and looks at the file directory on that volume to run an overlay named Exec. This provides flexibility; the Exec overlay does not have to be the first thing on the disk, and systems can be customized by providing an overlay on the disk called Exec. But, this approach also requires quite a bit of resident code; the system has to be able to do not only disk I/O, but also interpret file directories, and perform overlay linkages.

The TwinSystem gets around the ROMs by loading the system code into high memory (E000H to FFFFH) at boot time. Essentially, this is done by having the Exec run a file called Boot.2U on the TwinSystem disk.

Single User Boot Sequence

The boot process in the System 88 is fairly complex; by the time the System 88 first "talks" to the user, it has exercised the file lookup mechanism, the overlay mechanism, CPU card and main machine memory, the disk controller and disk. Part of this process is handled by code in the system ROM, and the remainder is done by the Exec. ROM based initialization is discussed first.

Differences In Rom Versions

The discussions that follow are based on version 81 root roms. Earlier versions may do things in a different manner, such as searching for the system drive, and what wormholes and interrupts are initialized.

Initialization Done by the Disk System Roms

The disk system ROMS are entered at system reset. The interrupt system is disabled. The stack pointer is reset to 1000H, and the screen pointers used by the video driver in ROM are set to reflect the video board at location 1800H. The disk controller is then initialized. Memory from 2000H to 31FF is then set to zero. This initializes various system cells, "cleans out" the overlay and directory areas, and rewrites the parity bit for memory boards with that feature. The interrupt handlers UVEC and PVEC are initialized to point to Ioret. The keyboard interrupt is conected to the keyboard handler in the disk system ROMS, and the input wormhole, WH0, is set. Wormholes 8 and 9 are set to provide the real time clock vector and the disk I/O vector; the parity interrupt handler is connected to SRA1. A form feed is output to clear the video screen.

Selecting Sysres

To select the system drive (SYSRES), the ROM code first tests location 1FEFH, to detect the MS controller. If the controller is present, Look is called to find Exec on drive 4. If it is found, SYSRES is set to 4 and we boot off the 8" disk. If the controller did not exist, reported errors, or the disk did not have Exec on it, we set SYSRES to 1 and attempt to boot from that drive. Note that the call to Look to find Exec on drive 4 just looks for Exec with no extension specified. This may cause trouble if the disk in drive 4 at boot time has Exec.TX on it. The system will try to boot from this disk and fail.

Loading the Exec Overlay

After selecting SYSRES and clearing the screen, we fall into Warm. Warm first resets the stack pointer, then does a Gover call to overlay Exec. Since we set 2000H to 31FFH to zero, the overlay is not found in memory; Gover calls Runr to load file Exec.OV from the SYSRES device. Runr calls Look to find this file, and since NFDIR and the directory area have been cleared, Look reads the directory from the SYSRES device. If all goes well, the file Exec.OV is read from the SYSRES device into memory, and the overlay is entered at OVENT with interrupts DISABLED. It is in the first I/O to the disk that interrupts are enabled for the first time in the boot sequence.

Initialization Done by the Exec

The Exec is entered DISABLED by the Gover call at Warm. It sets the EIC bit in Eflg1, to disable Control-Y action when the interrupts are enabled later. A check is made for the EERR bit in EFLG1, which notes an error present in ERROR for the Exec to process. The ROM part of the boot process cleared EFLG1 and also cleared ONCE. Since the ONCE flag is zero, the Exec calls its initialization routine.

The first thing done is to Look up the file Auth.OV on the SYSRES device. If the overlay exists, it is invoked via Ovrto. Note that the first thing Exec did when it was entered was to set EIC in EFLG1, disabling Control-Y. SCHR was cleared by the ROM part of the boot, disabling front panel entry. Auth is entered with interrupts disabled, and the first thing it does is to "lock all the doors" so it cannot be interrupted! If Auth returns from the Ovrto to the Exec, the user is authorized.

If Auth.OV did not exist, or returned, Exec then stores a 0C9H (a RET instruction) into WH5, WH6, and WH7, the printer wormholes. Exec then Looks up file Prnt.OV on the SYSRES device. If it exists, it is invoked via Ovrto with a function code of 01 (initialize default).

After setting up the printer, Exec looks for any file named INITIAL on the SYSRES device. If an INITIAL file is found, the string "INITIAL" followed by a carriage return is moved into the Exec command buffer and an internal flag set to inhibit reading a command from the user.

Nothing is displayed on the screen if "INITIAL" was set in the command buffer, so that the first thing the user sees will be controlled by the INITIAL program or command file; otherwise the

Exec version number message is displayed.

Since the ONCE flag has not been set to note the completion of the boot process, Exec scans for the end of user memory. Exec starts the scan at USER (3200H). The code used to scan for the end of memory is:

```
Mscan   LXI     H,USER
Msc1    MOV     A,M
        CMA             ; flip
        MOV     M,A
        CMP     M       ; see if it flipped over
        JNZ     Msc2    ; jmp/nope, found the end.
        CMA             ; flop
        MOV     M,A     ; put it back like we found it
        CMP     M       ; see if that worked.
        INX     H
        JZ      Msc1    ; jmp/look at next one.
        DCX     H
Msc2    DCX     H       ; last good spot
        SHLD    MEMTOP  ; remember that.
```

Note that this look also rewrites all of memory, setting up the parity bit on memory boards so equipped. Parity was rewritten for memory from 2000H to 31FFH by the CPU memory scan. If this rewrite was not done, a fetch from memory could generate a parity error, as the parity was not properly set up.

If the INITIAL text was not loaded into the command buffer, the address stored in MEMTOP is displayed on the screen. If the text was put into the command buffer, the user is not prompted for input. If the text was not loaded into the buffer, the user is prompted and a command line read. After this, ONCE is set to 0FFH to note the completion (at last!) of the boot process. Note that the INITIAL file is handled in such a way that it seems to the Exec that the user typed INITIAL as a command. Calling the INITIAL program is special-cased; the program is entered with EIC set in EFLG1 to disable Control-Y. This allows the program to set up exit and interrupt control without being harassed by the user. Other than in the special case of invoking the program INITIAL at boot time, the Exec always enters a program with EIC cleared in EFLG1 and PVEC set to Iexec in the disk system ROM.

TwinSystem Boot Sequence

Since the Twin uses the same CPU ROMs as the single user system, its boot sequence is the same, up to the point where the Exec is entered. The ROM code goes through the same steps in finding and loading Exec.OV from the boot disk, and enter the Exec at OVENT. What the ROMs don't know is that they've loaded the TwinSystem Exec; there's still a good bit of initialization to be done!

The Twin Exec distinguishes between system boot and a normal Exec entry by looking at single user wormhole Ø (SUWHØ). If SUWHØ contains ØCDH (a CALL opcode), the TwinSystem has not been initialized; the Exec jumps to a special routine. If the system does not have RAM at EØØØH, an error message is displayed on the screen and the system halts; we don't have enough memory to run the Twin. If there was RAM at ØEØØØH, SYSRES is copied into SUWH1. Then, the single user Runr service is called to load Boot.2U from the system disk. Any errors are reported by calling single user Err. If Boot.2U loads, it is entered at its start address.

Initialization by Boot.2U

The initialization code in Boot.2U is at the end of the module; currently it starts at EFØØH. This area is used after the system is initialized to hold Gfid. This is important for two reasons.

First, the initialization code can be as long and as complex as needed (up to 3K or so), as it is thrown away after it is used. Second, because Gfid is loaded over the init code, the init services cannot use Gfid! When the resident (Boot.2U) is loaded, the Gfid vector is connected to a small routine that uses Runr to load Gfid into memory.

Note that Boot.2U is entered with interrupts disabled. They stay that way for a while.

The first thing Boot.2U checks in initialization is that the CPU ROMs are version 81 or later. The Twin cannot run with earlier version ROMs, as they do not have the WH8 hook needed to steal the real time clock.

Next, the upper 8K (EØØØH to FFFFH) is rewritten to correct the parity bit. Then, the resident portion of Boot.2U (from EØØØH to EEFFH) is checksummed. This insures that the code has not been modified, and that it was loaded into good memory. If

the checksum test fails, an error message is displayed and the system halts (with interrupts disabled).

After the checksum test, the interrupt vectors and single user wormholes are set up. This changes SUWHØ from a CDH to C3H; the next time the Twin Exec gets control, it will think the system has been set up. Then, the PHANTOM line is switched on and off to see if we have one or two users in the system. This is done by putting a pattern in low memory (2ØØØH), flipping PHANTOM, and looking to see if the pattern is there. If it is, we must be running on a single user system (56K, no PHANTOM line). If the pattern isn't there, we must be on a Twin with two 48K boards connected to PHANTOM. PMASK is set accordingly; ØØ for single user systems, and PHANTOM for a Twin.

At this point, we are ready to perform per-user setup for either one or two users in the system. Before this is done, we modify the JMP instruction at Cold (EØØØH) to be a JMP ØØØØ. If this instruction is executed, the system will reboot.

Per-User Setup

The per-user setup code is called to initialize the user's memory space, wormholes, and stack. It is called with the user's address space selected (using PHANTOM and the BRG), so that the one routine is used for setting up both users in a Twin. First, memory from 2ØØØH to EFFFH is rewritten to correct the parity bit. Then, the area from USP through KBEX is loaded. After data is copied into this area, it is checked. If it did not copy correctly, we assume that user memory is bad and we halt the system with an error message displayed on the proper user's screen. The area following KBEX, to USER is then zeroed, and a memory error declared if this memory does not go to zero. The user's SYSRES, PVEC, UVEC, and SRA7 are then set up, and WH1 called to clear the screen. Note that we are still disabled!

At this point, the initial environment is built onto the user's stack. Later on when the system is "turned loose" for normal operation, the user will be placed in operation by Flipem jumping to Ioret to load the environment from the stack. So, we build an environment onto the stack with the PC value EØØ3H, Warm. So, when the user is placed in execution, the Exec will be loaded. USP is updated after this environment is built.

Common Setup

After initializing one or both users by calling PerUser, we still have a few things to do. We still can't run as a full system yet, and can't allow user timeslicing, so we save the computed value for PMASK, and store a ØØ in PMASK to prevent slicing. Note that interrupts are still disabled at this point.

We first try to initialize the printer. In the TwinSystem, the printer driver itself lives in system common memory (FCØØH), as it is directly connected to the USART interrupt. Look is called to find Prnt.OV on the SYSRES disk. Note that during this call to Look, interrupts are enabled for the first time since the Exec was entered. If the file exists, we call it with a Gover and a function code of 1 in A (initialize default).

After the printer, we try to load the cache code (Cache.ZO) for the 8" disk controller. If this file exists, we try to feed it to the 8" controller, which may or may not be there.

Twin Startup

At this point in time, both users have been set up; memory has been rewritten to set up the parity bit. Both screens have been cleared. The printer driver and cache functions have been initialized. The wormholes (single and per user) and interrupt vectors have been initialized. Each user's stack has been set up to contain a dummy environment to take the user to Warm. Assuming we've got both users, we're currently mapped and running as user 1. Since we've been doing setup, user 1 has a valid directory in SBUF1. We've not run as user 2 yet, so user 2 does not have a valid directory in its SBUF1.

We're ready to turn things loose. So, PMASK is restored to its earlier computed value, and we jump to Ticker to start system operation.

Return of the Twin Exec!

At Ticker, we park this user (user 1). Note that Ticker assumes that an environment is already on the stack; it's the environment we put there in the peruser setup code. We get to Flipem, where we flip to the other user (user 2), load that user's environment through Ioret, and go to the stacked PC. This puts us back at Warm, but this time it's the Twin's Warm (EØØ3H) rather than the single user Warm (4Ø3H). Now user 2 needs to get the Exec. It goes through the same sequence as usual; the overlay isn't in memory, so it must be loaded. Look has to call Dio to load in the directory from SYSRES. About the time we get to Dio as user 2, we either get a clock tick that causes us to switch users, or we start the read to the SYSRES disk, and Giveup waiting for the read to complete. From this point on, we'll just look at what one of the Execs does. The user switching code works, so we can pay attention to other things.

Here we are back in the Twin Exec again. SUWHØ doesn't have a CDH in it, so we have done system initialization. ONCE contains zero, so we have to do our part of initialization. MEMTOP is set to DDFFH; we know how much memory we have. The

stack runs from DFFF down to DE00; this is more space for the stack than in the single user system. We then force in Gfid by calling it to find the Exec on the system disk. We are in big trouble if this fails!

After forcing in Gfid, we look for the Auth overlay, and call that if it exists to perform user authorization.

We then look at BRG to see if we are user 1 or user 2, and try o find the appropriate INITIAL file, either INITIAL1 or INITIAL2. If that file exists, we move its name into CMND so we will do it. If not, we look for INITIAL and if found do that.

Note that Boot.2U already took care of printer initialization for us.

Section 9

System 88 Interrupts, Input/Output Ports, and Switching

The System 88 is interrupt-driven. Tasks such as disk input and output are initiated by interrupts. The keyboard generates interrupts, as does the real-time clock. This section of the System Programmer's Guide gives an overview of the interrupt system and the input/output port structure of the System 88 as it affects the system programmer.

Interrupts: The Interrupt Environment and Ioret

In the 8080 processor as implemented on the PolyMorphic Systems CPU card, response to interrupts is indistinguishable from the execution of RST instructions. Both cases will be referred to as interrupts except where the distinction is important. When an interrupt occurs with interrupts enabled, the processor pushes the address of the next instruction onto the system stack and jumps to one of the interrupt vector locations defined in the monitor root. (See appendix, Listing of the 4.0 Monitor.) The interrupt receiver code in the root pushes registers PSW, B, D, and H onto the system stack, in that order. This is called the interrupt environment. The monitor code then loads the new program counter value from the corresponding interrupt vector and transfers control to it. Note that the receiver is entered with the environment on the stack and interrupts disabled. After completing its processing, the service routine jumps to Ioret in the monitor root to restore the interrupt environment, enable interrupts, and resume the interrupted task.

A Moral for Interrupt Level Code

There is a very simple moral for the design and implementation of code running at the interrupt level: keep it simple, keep it fast, and keep it disabled. The first part of the moral comes from common sense and the difficulty of debugging code entered as the result of an interrupt. As for the second part, the faster the code executing at the interrupt level, the more time is left for other processing, and also the more interrupts may be handled within a given period. Interrupt handling code should be kept disabled (especially in a system with more than one kind of interrupt) to prevent the following scenario. An interrupt occurs, and during its processing, the interrupt handler enables the interrupts. An interrupt of a different type occurs and is processed. The original interrupt handler is reentered, and another interrupt of that type occurs. This scenario, if it continues, causes system failures in one of two ways (usually!).

In the first failure mode, the system stack overflows as

interrupt environments are pushed onto the stack faster than they are removed. The stack grows, and grows, and then either marches over system data, causing a failure in some other part of the system, or grows into read-only-memory. When the environment "pushed" into ROM is restored, it will probably not be correct. On the System 88, the "blown stack" failure usually makes itself known by marching repeating patterns over the video screen.

The usual cause of failure other than "blown stack" is the "non-reentrant" handler. Let's say that a particular interrupt handler was coded (ignoring the moral above!) in such a way that it enabled interrupts and used some fixed temporary storage locations. During processing of an interrupt, it enables and gets re-entered by an interrupt of the same type. This interrupt is processed, CHANGING the temporary locations used by the interrupt handler. Then the handler is reentered at its point of interruption with registers restored, but its temporary locations modified. From here, things usually get quite confused and quite hard to predict!

In summary, if the interrupt level code is kept simple, fast, and disabled, it will work. Interrupts and interrupt systems allow computers to process randomly timed asynchronous events with a minimum of software overhead. It is the opinion of the designer of the System 88 software that some people are opposed to interrupt-driven systems because they don't understand interrupts and are not capable of the careful design and implementation required for interrupt processing.

Table 1: INTERRUPT LEVELS, RST's, VECTORS, AND FUNCTIONS

INT LVL	RST	VECTOR ADDR		FUNCTION IN SYSTEM 88
7	Ø	.....		System Reset- no vector
6	1	ØC1ØH		Not used
5	2	ØC12H		Single density disk controller
4	3	ØC14H		Not used
3	4	ØC16H	(UISR)	USART interrupt (See Note 1!)
2	5	ØC18H		Keyboard interrupt
1	6	ØC1AH		Real time clock interrupt
Ø	7	ØC1CH		Single step interrupt

Note 1: Before diddling with the USART, read section 5.

Table 2: SYSTEM 88 INPUT AND OUTPUT PORTS

PORT (HEX)	I/O	FUNCTION
Ø	I/O	USART data (See Note 1!)
1	I/O	USART control (See Note 1!)
4-7	O	Baud rate, device and user selection
8-B	O	Real time clock reset
8-B	I	ROM on for CP/M (See note 3)
C-F	O	Single step trigger
C-F	I	ROM off for CP/M (See note 3)
18	I	Keyboard data
2Ø-2F	I/O	Single density disk controller

Note 1: Before diddling with the USART, read section 5!

Discussion of Table 1

Table 1 lists interrupt levels, RST instruction codes, vector addresses, and interrupt functions defined in the System 88. The INT LVL column corresponds to the vectored interrupt (VI) numbers on the CPU board and the bus. In the next column is the equivalent RST instruction code. The address given under VECTOR ADDR is the address of the 16-bit vector location used by the monitor root (see appendix). The function of the interrupt is also given. Proper operation of the System 88 requires that those interrupt levels used by the system are not modified. For example, the disk transfer logic makes use of the system real-time clock. Interrupt levels 6 (RST 1) and 4 (RST 3) are not used PRESENTLY in the System 88 (use with caution). Remember that any code operating at the interrupt level is effectively part of the operating system. For more information on the exact processing of interrupts, refer to the monitor listing in the appendix.

Discussion of Table 2

Table 2 gives a sketchy description of dedicated input and output ports used by the System 88. Further information on these ports may be obtained from the hardware manual associated with the specific board, and may (possibly) be inferred from the listing of the monitor root in the appendix. As with system interrupts, the proper operation of the System 88 depends on these control ports and on the fact that the operating system is the only code manipulating them. Other programs fiddling with system ports will most likely cause the system to become confused and fail.

Twin System User Switching: The Baud Rate Generator Latch

The baud rate port is used for three purposes in the System 88. The baud rate latch is a six bit latch. The least significant four bits are used to select the baud rate. Bit 4 (10H) is used to select a device on the mini cards (printer interface and cassette interface). The device selected by bit 4 corresponds to the 0 or 1 jumper on the mini card. Bit 5 is used to select user 1 or 2 on the Twin. The output of bit 5 of the BRG latch is connected to pin 67 of the bus which controls the switching of memory boards and video boards. In early versions of the CPU board bit 5 of the BRG latch was controlled by bit 6 of the 8080. Later versions use bit 5. The system software, in order to cover all the bases, uses both bits (60H) when attempting to set bit 5 of the BRG. This is for information only! Almost any attempt at changing the BRG latch is guaranteed to put you into binary oblivion.

Interrupt Character Processing

Following is the single user interrupt character processing code extracted from the CPU ROMs. It is entered at the interrupt level, with interrupts disabled. We enter at KBSB if the character was a ^Y, at KBSP for ^Z, and at KBUB for the character contained in SCHR. Note that characters from command files do not follow this path, and cannot cause these interrupts.

```
KBSB    LDA     EFLG1   ; is the Exec running?
        ANI     EIC
        JNZ     Ioret   ; jmp/don't do it!
        INR     A
        STA     SBRK    ; set ^Y flag non-zero
        LHLD    PVEC    ; here's where we go
KBEV    LDA     DONT    ; so do we or don't we?
        ORA     A
        JNZ     Ioret   ; jmp/we dont, something happening
        CALL    Killi   ; abort command files and flush
        PCHL            ; follow the yellow brick road...

KBUB    LHLD    UVEC    ; vector for UCHR int
        STA     UBRK    ; set flag nonzero
        JMP     KBEV    ; go to it.

KBSP    LXI     H,PANEL ; front panel code
        JMP     KBEV    ; go to it.
```

This code shows the interactions and uses of the various character interrupt cells. DONT completely masks these interrupt characters if its contents is nonzero. EIC set in EFLG1 masks ^Y. For both ^Y and the user character SCHR, flag bytes are set nonzero even if the resulting interrupt is masked by DONT.

Things become a little more complex in the TwinSystem. Where control is transferred to the interrupt routine directly from the interrupt service routine in the single user system, we must stack the desired interrupt environment to be run the next time this user is run. Additionally, character interrupts are masked out if the user is currently in Gfid, noted by possession of the GFLOCK semaphore. As in the single user code, Kbsb is entered for ^Y, Kbub for SCHR, and Kbsp for ^Z.

```
        ; Handle processing of interrupt characters. When we have
        ; one, we build a phony environment on the user's stack
        ; after picking up USP, and update USP so that we'll enter
        ; the code the next time we run the user.

Kbsb    LDA     EFLG1
        ANI     EIC     ; is the Exec running?
        JNZ     Kbxx    ; jmp/don't do it!
        INR     A
        STA     SBRK    ; mark we got a ^Y
        LHLD    PVEC    ; where is the wizard???
Kbev    LDA     DONT    ; do we or don't we do it?
        ORA     A
        JNZ     Kbxx    ; jmp/better not, something doing

        ; Let's see if Gfid is locked. If it is, and we've got an
        ; interrupt from the user that has it locked, ignore 'em!

        LDA     GFLOCK
        ORA     A       ; locked it is?
        JZ      Kbxy    ; jmp/no
        ANI     PHANTOM ; by which user locked?
        JZ      Kbuø    ; by user ø
Kbu1    LDA     BRG     ; user 1 has lock
        XRA     B       ; does user 1 want goose?
        JZ      Kbxy    ; jmp/no, allow goose!
        JMP     Kbxx    ; locked, ignore goose!

Kbuø    LDA     BRG
        XRA     B       ; does user ø want goose?
        JNZ     Kbxy    ; jmp/nope, allow it
        JMP     Kbxx    ; ignore goose for locked user

        ; Load user's stack pointer so we can build interrupt
        ; environment on right stack, kill typeahead.

Kbxy    XCHG            ; addr to DE
        LHLD    USP     ; get user stack pointe
        SPHL            ; and load it in

        PUSH    D       ; the PC
        LXI     H,Killi ; but go through Killi first!
        PUSH    H       ; first PC
```

```
            LXI     H,Ø
            PUSH    PSW       ; PSW
            PUSH    B         ; BC
            PUSH    D         ; DE
            PUSH    H         ; HL
            DAD     SP        ; and update saved SP in USP
            SHLD    USP
            JMP     Kbxx      ; then split.

Kbub        LHLD    UVEC      ; where do we go from here?
            STA     UBRK      ; set user break has occurred
            JMP     Kbev      ; go do it.

Kbsp        LXI     H,Fpanel  ; where we would like to go
            JMP     Kbev      ; Zoom off...
```

The above code is straightforward, just littered with details. The main difference from the single user is that we don't want to compromise system performance by entering the user's interrupt routine from the interrupt level. Especially if Killi must abort a command file; this displays text on the screen, and can cause the entire display to scroll, which takes a good bit of processor time. This problem is solved by adding another interrupt environment to the stack, containing not only the address specified in the proper vector, but Killi as well. When we pick up the resulting environment to run the user again, we'll first call Killi, and Killi will return to the routine specified in the interrupt vector.

Section 1Ø

Volume Manager

Introduction

This is a description of what the volume manager is, how it works, and how to talk to it. The volume manager allows the System 88 to better allocate its non-volatile storage resources (e.g. floppy disks, hard disks, magnetic tape, bubble memory).

History

In the beginning the number and type of non-volatile storage resources that could be connected to the System 88 and addressed through the system's DIO utility was limited to three 5.25 inch single sided single density floppy disks. There was no way, aside from rewriting the system, to connect any other devices to the system. The drives were also permanently addressed as devices 1, 2, and 3. The code which drove these devices resided in ROM, and there were no vectors in RAM which could be used to filter calls to DIO.

As the system became more widely accepted for use in business applications a need was realized for more storage. This need was filled with the System 88/MS which could handle an additional four 8 inch double or single sided floppy disks. Once again the system was rewritten and the device drivers were burned into ROM with no RAM vectors, effectively casting it in brass and precluding any possible changes or additions to the device drivers. These drives were permanently allocated as devices 4, 5, 6, and 7.

Shortly after the introduction of the MS it was discovered that with a few minor alterations the double density controller which was previously used to run the 8 inch drives could be used to run the new 5 inch double sided drives. These new drives, which were just becoming available, would allow us to quadruple the storage on five inch diskettes.

At this point it was becoming apparent to the designers of the System 88 that as new and different physical devices became available, we would want to be able to connect them to our computer. For this reason, when the system was rewritten again to handle the new drives, the DIO calls were vectored through a location in RAM so they could easily be changed. These new drives replaced the single density drives (devices 1, 2, and 3) in the device addressing. Another function was added to DIO which would return the number of sectors on any physical device when called.

Now we come to the present. Technology has struck again with Winchester fixed disks. It is desirable to enhance our product line with these new devices. However, they present two rather tough problems to the system software. The first of these is that the system only understands a single digit device number (1-9) and most of these (1-7) are already used up. The second problem is that the system is unable to address the entire fixed disk. Re-writing the operating system to fix these limitations would be a major undertaking and would obsolete much of the already developed applications software. In addition, handling devices of this size would be very unwieldy for the current system. The solution to these problems is a volume manager.

Definition

What do we mean by the phrase "volume manager"? It is easier to understand if you can think of the file specifier (e.g. <4<HELLO) as a logical "volume" number followed by a path name rather than a physical drive number followed by a path name. The physical drive "4" may correspond to the logical volume <4<, but does not necessarily have to.

Physical devices may contain more than one volume. For example, we may decide to set up a hard disk so that different areas on it appear to the system as volumes <6<, <7<, and <8<. The volume manager is responsible for mapping logical volume numbers onto physical devices. It figures out which physical device the volume is on, and where on that device it resides.

In the System 88 the volume manager resides as a filter on the input of the DIO function. Instead of having the system talk directly to the physical device drivers, it talks to the volume manager. The volume manager then translates the system calls and feeds them to the actual device drivers. This is the only part of the volume manager which the end user sees, but is by no means the only function of the volume manager code.

The volume manager is actually five programs. The first program is the filter on the input of DIO which we just discussed. The second program is the "Vmgr" overlay which allows you to connect and disconnect both volumes and physical devices to the system. The third program is called "Device-Configure". It allows a physical device to be broken up into logical volumes of any size, and then encodes that information onto the device. The fourth is VLIST, that lists how the volumes are currently assigned. The last program is called "Volume-Default" and is used to configure the default setup of your system.

How It Works

Resident Code

The volume manager is essentially a table driven device. When the system (with the volume manager connected) is passed a request for DIO the volume control block for the volume specified is indexed. Information from the block is used to translate the parameters passed to DIO to the parameters which the device drivers need. There is a volume control block for each of the nine volumes which the system can access. Each volume control block contains the following information:

Offset	Size	Use
0	2	"Busy" semaphore address
2	2	Device driver address
4	2	Parameter area address
6	2	Volume Size
8	3	Offset from physical 00 on device
B	1	Physical drive number
C	1	Volume status

The "Busy" semaphore address is the location in memory where the flag is kept which indicates whether the controller for that device is in the process of doing I/O. This is necessary so that, in the TwinSystem, one user does not try to initiate an I/O while the other user is in the process of doing one.

The device driver address is the location in memory which is called in order to do the actual Disk I/O after the parameters have been translated by the volume manager.

The parameter area address is the location in memory where the physical disk address is passed to the device driver. If this field in the volume control block is 0000 then the physical address is passed in HL. This is needed is in order to handle devices which have physical address capabilities larger than 0FFFFH.

The volume size contains the number of sectors that the volume contains for volumes with fixed sizes. An example of a volume with a fixed size would be one of the volumes on a hard disk. If the number contained is 0000 then the volume size is variable, and the device driver must be called for the size. An example of this would be an MS which can contain either a single or a double sided diskette.

The offset from physical 0 is the amount which must be added to the disk address to access the correct area on the physical device.

The physical drive number is the physical address which must be passed to the device driver in order to access the device.

The volume status tells us if there is a device plugged onto the other end of this volume. If this byte is 0 then the volume is not in use. Non-zero status tells us there is a device mapped onto this volume. Other codes are reserved for future use such as read-only or other protective status.

> **CAUTION: These volume control tables are sacred! They are not designed to be written into. If you want to re-configure the system use the Vmgr overlay. That's what it's there for. Any tampering with these tables is likely to cause a disastrous loss of data on your disk.**

Vmgr Overlay

The function of the "Vmgr" overlay is to set up the tables in the resident portion of the code in a way that nothing gets stomped on. There are five function codes which can be passed to the overlay. They are:

Code	Function
0H	Initialize I/O drivers and VCB tables
10H	Return a pointer to the specified VCB in HL
11H	Connect a volume
12H	Get device definition block
13H	Disconnect a volume

Commands are passed to the overlay in the A register.

The initialize function (command 0) is called by the System 88 executive when the system is reset. There are two sets of things that the overlay must initialize. The first set is the device driver(s) and the second is the default hookups for the volume control table. The initialization of the device drivers is done differently for single and twin systems.

Single User Driver Initialization

The first thing the overlay does is look for a file on the disk called "Driver.DD". This file contains the hard disk driver, any other device drivers the user may have, and the volume manager filter. If the file does not exist, "Vmgr" will return with an error code of 01D5H (No device driver). No other initialization will be done in this case.

If the file exists, it will be relocatably-loaded to just

below the top of RAM, and executed. This execution will perform the initialization of the device driver itself. After the device initialization is performed, the program jumps to the volume control block initialization code.

TwinSystem Driver Initialization

In the TwinSystem initialization the overlay also looks for a file called "Driver.DD". THIS IS NOT THE SAME AS A SINGLE USER Driver.DD FILE. This file only contains any optional device drivers. This file is also not relocatable, but has a fixed load and start address of FA00H. If the file exists and looks OK, it will be loaded and executed. This execution does driver initialization. When the driver code returns the routine will jump to the volume control block initialization code.

If no "Driver.DD" file exists, device initialization will be skipped, and it will jump directly to the volume control block initialization code. No error code will be returned in this case.

Volume Control Block Initialization

The last thing the initialization function does is to set up the volume control blocks with their initial values. It does this by repetitively calling the connect function with the default values that the user has set up for his nine volumes. Any errors encountered during these hookups will be reported back to the Exec.

The return a pointer function (command 10H) takes the volume number which is passed in register C and points HL to that volume control block. The volume control block contents are defined earlier in this discussion.

The connect function (command 11H) can be invoked in two ways. It can either be called to hook up a named volume, or it can be called to hook up the next available volume on a physical device. To invoke the command, the overlay is called with the volume number in C, and the physical device number in B. Optionally, a string of 8 characters may be pointed to by HL when hooking up a named device. If HL = 0 (no name pointed to) then it will be assumed that the user wants to connect the next available volume on that physical device.

There are five possible errors that the "Vmgr" can return when trying to hook up a device. They are:

```
01D0H:  That drive is already connected.
01D1H:  That volume is already connected.
01D2H:  I can't find that volume. (for named volumes)
01D3H:  No volumes available. (for unnamed volumes)
01D4H:  That device has no volumes defined.
01D5H:  No device driver.
```

The volume manager will not connect a new volume while an old volume is connected to the requested number. In addition it will not duplicate any volumes. For example, volumes <4< and <5< would not be allowed to point at the same spot on one physical device. If the volume manager allowed this the system interlocks would not work, and files would be scrambled.

In order to allow different volume sizes on a physical device and not foul up the pointers to different volumes on that device, each volume-separable device has information regarding its configuration encoded on the device. This information is stored on the first physical sector of the device. This sector is not accessible via system calls to DIO. The volume manager uses this information when connecting a volume to assure that the pointers to different areas of the physical device remain secure.

The read device definition block command (12H) makes this information available to the user. The register setup for this command is: B contains the physical device number of the device being interogated; HL points to the memory area where the device definition block is to be placed. The format of the device definition block is as follows:

Size	Value	Definition
1	1-19	Number of logical volumes contained on this device.
* The following is a sample entry. This entry is repeated for each logical volume contained here.		
2	100-65535	Volume size
3	0-(dsize-100)	Offset from physical 00
8	ASCII chars.	Volume name

The disconnect volume (13H) function is passed the volume number in C. Before the volume is disconnected, it checks to see if the volume is busy. This will assure that one user does not disconnect a volume while the other user is using it. The only

error that this function can return is 01C2H: That device is busy.

Device-Configure Program

The purpose of the device configuration program is to set up the device definition block on a physical device. The device definition block was discussed in the previous section. When the program is run it first comes up and warns the invoker that it is about to zero the data on the entire device. It then asks him for a device number. The selected device is then formatted with error mapping enabled.

After the initialization is complete the program begins asking the user for volume names and sizes. This process is terminated in one of three ways. It will terminate when there is no more room for a volume, when the maximum number of volumes (19) has been reached, or when the user answers the size question with a null (just hits return). At that point the program will construct the device definition block and write it to the first physical sector of the device.

Volume-Default Program

The volume default program sets up the default definitions for volume hookup which the "Vmgr" overlay uses on initialization.

> NOTE: This program modifies the "Vmgr" overlay to accomplish its task. This means that each system disk which is used on the system can have a different set of default values for volume hookup. This should not be confused with the Drive Configure program which permanently sets up a physical device. Care should be taken that you make all system disks with volumes defaulted compatible with the physical device setups which the program will be connected to.

When the program is invoked it will ask you, sequentially for each volume number 1-9, what physical device you want connected to that volume. If that device has more than one logical volume on it the program will then display a list of the volume names for your selection. If you don't select a name the program will connect the next available logical volume on that device.

After you have made your selections for all the volume mapping the program will ask you which volume you wish your system to "boot" from. When the program has all this information it will rewrite the last sector of the "Vmgr" overlay back out to

the disk with the new information in it. This works in much the same way as the printer driver's "Setup" works.

Connecting New Device Drivers

If the system programmer wishes to connect new devices and their associated software drivers (e.g. cartridge tape drives, other hard disk types, etc.) to the system, a mechanism has been provided in the volume manager code to allow this. The mechanism is different for the single user system and for the TwinSystem.

In either system, when the "Vmgr" overlay is asked to connect a new volume to the system it gets information from the resident portion of the volume manager as to where the memory locations are that it needs to hook up the driver. The table contains twenty entries which correspond to the twenty physical devices which are possible to connect to the system. The entries in the table are the existence flag, the type flag, and locations in memory of the device driver, parameter area, and the busy flag. This table is called the driver definition table. Entries in the table are organized as folows:

Size	Offset	Definition
1	0	Existence Flag
2	1	Busy Flag Address
2	3	Device Driver Address
2	5	Parameter Area Address

The existence flag tells the "Vmgr" overlay whether a device driver exists for the physical device. If none exists, then the overlay will refuse to connect a volume to it.

The busy flag address is the one byte flag in memory where the TwinSystem can look to see if the device driver is being used. It is necessary to have these flags on a per-controller basis to avoid I/O collisions and their consequent destruction of data.

The device driver address is the address in memory where the code exists, which will do transfers of data from physical sector numbers. The format of the parameters passed to the driver code will be the same as for DIO with the possible exception of the disk address. For devices which are incapable of being divided into logical volumes, the disk address will be passed in HL just as it is in DIO. On devices which are capable of being divided, the disk address will be placed in the three byte buffer in memory which is pointed to by the next table entry. This memory address will also be passed in HL to the device driver.

The parameter area address is the three byte buffer in

memory where the physical disk address is passed to device drivers which are capable of having disk addresses larger than 65535 (0FFFFH). These are also the devices which can be broken up into logical volumes. If the device is not of this type the entry in the table for this field will be 0.

The base of the table just described is pointed to by the system variable DRVADTAB which can be referenced in the SYSTEM.SY symbol table file.

Single User Device Driver Addition

In order to add a device driver to the single user system you must modify the device driver software in the "Driver.DD" program. This modification entails adding your driver code to the code which is already there, adding the initialization code for your driver to the initialization code already there, and making entries into the driver definition table which specify the variables for your driver. The driver definition table for the single user system is part of the "Driver.DD" file which is being modified.

A source disk and listing for the "Driver.DD" file for a single user system is available from PolyMorphic Systems.

TwinSystem Device Driver Addition

In order to add device drivers to the TwinSystem you must write the "Driver.DD" program yourself. This program should be assembled to run at FA00H. The initialization code for your device should be jumped to by a vector at the beginning of the code. One of the tasks which must be performed by your initialization routine is to insert the appropriate addresses and flags into the driver definition table. The size of the driver code which you can add to the TwinSystem is limited to 1/2 K (two sectors).

Your driver will be automatically loaded into the appropriate slot in memory when the initialize code is fed to the "Vmgr" overlay by the system executive. You can even have the volume tables defaulted to it.

DRVADTAB → driver definition table (physical definitions)

20 entries of:
- (0) 1 existance flag (driver code existance)
- (1) 2 busy flag address (I/O collision prevention)
- (3) 2 device driver address (Dio style)
- (5) 2 parameter area address (0 = not divisible)

Volume-Divisible device: Dio calls will set HL to param. address
Indivisible: HL holds physical sector number, like Dio.

VCBTAB → volume control block table (logical assignments)

9 entries of:
- (0) 2 busy flag address } copied from above during "connect"
- (2) 2 device driver address } copied from above during "connect"
- (4) 2 parameter area address } copied from above during "connect"
- (6) 2 volume size (0 means ask dev. driver) } copied from DDB during "connect"
- (8) 3 offset from phys 00 on device } copied from DDB during "connect"
- (B) 1 phys device # } copied from DDB during "connect"
- (C) 1 volume status (FF = in use, 00 = not) } copied from DDB during "connect"

Device definition block: 1 byte: # of volumes on device (nvol)

nvol entries of:
- (0) 2 volume size (100...65535)
- (2) 3 offset from phys 00 (0... [dsize-100])
- (5) 8 volume name (ASCII)

[stored on last sector of device.]

Section 11

CP/M Implementation on the System 88

CP/M Switching: The CP/M Line.

CP/M requires RAM at address Ø-2ØØØH where all the System 88 ROM and controllers and video board live. In order to have the RAM available, a "soft-switch" was created which turns off the CPU ROM and RAM, the controllers, and the video board, and changes the address of the 8K RAM board at EØØØH to Ø. When CP/M needs to access a controller or the video board the CBIOS uses port 8 to allow access and then uses port ØCH to switch back. Bus pin 16 is used for this switching and is called CPM- (read as "CPM Not").

All I/O in CP/M is done through code called the BIOS (Basic I/O System). This code along with the modified System 88 printer driver is the heart of Poly's implementation of CP/M. The code is written to be both easily understood and modified although some words of caution must be mentioned here.

When CP/M runs, it takes over the entire machine. It grabs all the Poly SRA vectors (to handle interrupt processing when doing I/O). The coding for the ISRs which are installed at ØØØØH in the bottom 8K was written to reduce stack nesting during interrrupt processing to the absolute minimum (2 bytes, namely the return addr). The interrupt service routines share an internal temporary stack called IntStack so therefore do not enable interrupts during the middle of an ISR.

To accomplish automatic booting of CP/M a System 88 directory is left in sectors Ø-3 of all CP/M disks. This directory has the strangeness of having an Exec.OV in sector 1 of the directory. This was done to save space for the 5" single-density disks which only have 35Ø sectors. The directory is created by the CP/M program INIT.COM. Note that the directory is completely full (NFDA is SBUF1+3FFH) and that the disk is also full (NFA is Dsize). The directory is also inconsistent (NFDA does not coincide with the first zero byte) which will guarantee that the disk cannot be touched by the System 88 (Exec/93 or later). The Exec.OV program is very simple, it simply calls the ROMs to "Runr" Boot.GO on the SYSRES drive and if the first instruction is C3H (a JMP), it blindly leaps into it. The code it jumps to is the program Boot.TX. This code is contained on the CP/M "boot track" and is a minimal BIOS which calls the ROMs to do its disk I/O. The code looks on the CP/M file structure using standard CP/M BDOS functions and initializes the MS controller by loading CACHE.S88 and gets the real BIOS into memory by loading BIOS.S88. It moves this real BIOS on top of itself and jumps to it.

The code which it jumps to is finally the real BIOS. Up to this point we have used the Root Roms to do all of the disk I/O (which possibly writes all over RAM between 2800H and 2C00H). The final BIOS contains a driver for the single density disks (both the North Star read/write and Poly 5" SSSD), but still calls the ROMs for 8" MS and 5" DD disk I/O.

Disk Structure

The disks' I/O routines were written to make them appear as IBM 3740 disks with no sector skewing. Thus when a file is looked at with SZAP, the bytes are contiguous.

The disks are mapped in CP/M because the BDOS written by Digital Research assumes you will alway boot off of logical drive 0 while this system boots off of either 1 or 4. The mapping is described in the addendum to the CP/M manuals published by Digital Research.

The console I/O routines came partially from the Root ROM code, with several enhancements. The keyboard routines have basically remained unchanged, and unlike most CP/Ms this one has typeahead. The console out (display driver), however, has been almost entirely rewritten and enhanced to include cursor movement and direct cursor positioning character codes as well as a flash facility. Normally the cursor doesn't flash but when it appears on top of a non-blank character - it alternates between the character and the ever-famous brick. Note: systems with the old ROMs (not one-size fits all) may exhibit irregular cursor flash rates.

CP/M is not supported on the hard disk.

Special Copy Program - PCOPY.COM

A special program, PCOPY.COM, was developed to allow copying of files to/from the Poly directory structure from/to the CP/M directory structure. It runs under CP/M and is operated as follows:

```
A>PCOPY B:FILENAME.TXT

CP/M <--> Poly Xfer v1.0 (01/19/81)

From Poly or CP/M (P or C)? P
Enter Poly Filename: <2<SUB<FILE.TX
Is this correct? (Y/N) Y
Working...please wait.
Success! Finished.
```

where the bold portions are the user responses.

The 'FILENAME.TXT' on the command line is the CP/M filename. The direction of the copy is determined by the 'P' or 'C' answer.

NOTE: Care should be exercised in the case of copying to the CP/M disk, as any file with the same name will replaced.

Note: CP/M is a trademark of Digital Research.

Section 12

Exec/94 System Symbol Tables

The following are the values contained in the SYSTEM.SY files for both Single and Twin User systems. They are listed in both alphabetical order and value order.

Single User Macro Definitions:

db gfid overlay

Single User Labels in Alphabetical Order:

BUGS	2DFC	BUSIES	0C6E	CBUF	2C00	CMDA	2D8C
CMDD	2D89	CMDF	2D88	CMDN	2D8E	CMDP	2D8A
CMND	2D40	CMPTR	2DC7	Ckdr	0433	Command	0C4C
DBARF	0020	DEFPATH	2E27	DEOUT	03D1	DEVMASK	000F
DONT	2D90	DRVADTAB	0C7E	Deout	03D1	Dhalt	0409
Dio	0406	DioA	0C66	DioBsy	0C6C	DioDn	0C6B
DioDrv	0C69	DioHL	0C67	DirAddr	2E02	EERR	0040
EFLG1	2DC9	EFLG2	2DCA	EIC	0080	ERROR	2D9A
EXECSP	2DAF	Err	040F	FILE	2DCB	Flip	042D
Flush	041E	Fold	042A	Gover	0415	Iexec	0436
Ioret	0064	JOBST	2D9E	KBD	0018	KBEX	2D86
KBIG	2D84	KBIP	2D82	KBUF	2D00	Killi	041B
LERR	2D9C	LUSER	2DC6	Look	0421	MEMTOP	2D80
MTO	2DA2	MUNG1	2DA7	MUNG2	2DA9	MUNG3	2DAB
MUNG4	2DAD	MemAdd	0C49	Msg	040C	NDRIVES	2D9F
NFA	2E00	NFCK	2DA1	NFDIR	2DA0	ONCE	2DC5
OVBC	2DC1	OVDE	2DBF	OVENT	2004	OVHL	2DBD
OVMEM	2E53	OVNM	2DB6	OVPSW	2DC3	OVRLY	2000
Ovrto	0412	PATH	2E04	POS	0C0E	PVEC	2D93
Pages1	0C4B	Rlgc	0430	Rlwe	0427	Rtn	0418
Runr	0424	SBRK	2D91	SBUF1	2800	SBUF2	2900
SBUF3	2A00	SBUF4	2B00	SCEND	0C1E	SCHR	2D98
SCREEN	1800	SCRHM	0C1F	SINT	2DB3	SRA1	0C10
SRA2	0C12	SRA3	0C14	SRA4	0C16	SRA5	0C18
SRA7	0C1C	STACK	1000	SYSRES	2D92	TIMER	0C00
UBRK	2D97	UCHR	2D99	USER	3200	USERS	0001
USTATS	2DB1	UVEC	2D95	VCBTAB	0C63	VERLOC	0439
Version	0081	WAKEUP	0C1A	WH0	0C20	WH1	0C24
WH2	0C28	WH3	0C2C	WH4	0C30	WH5	0C34
WH6	0C38	WH7	0C3C	WH8	0C40	WH9	0C44
Warm	0403						

Single User Labels sorted by Value:

USERS	0001	DEVMASK	000F	KBD	0018	DBARF	0020
EERR	0040	Ioret	0064	EIC	0080	Version	0081
Deout	03D1	DEOUT	03D1	Warm	0403	Dio	0406
Dhalt	0409	Msg	040C	Err	040F	Ovrto	0412
Gover	0415	Rtn	0418	Killi	041B	Flush	041E
Look	0421	Runr	0424	Rlwe	0427	Fold	042A
Flip	042D	Rlgc	0430	Ckdr	0433	Iexec	0436
VERLOC	0439	TIMER	0C00	POS	0C0E	SRA1	0C10
SRA2	0C12	SRA3	0C14	SRA4	0C16	SRA5	0C18
WAKEUP	0C1A	SRA7	0C1C	SCEND	0C1E	SCRHM	0C1F
WH0	0C20	WH1	0C24	WH2	0C28	WH3	0C2C
WH4	0C30	WH5	0C34	WH6	0C38	WH7	0C3C
WH8	0C40	WH9	0C44	MemAdd	0C49	Pages1	0C4B
Command	0C4C	VCBTAB	0C63	DioA	0C66	DioHL	0C67
DioDrv	0C69	DioDn	0C6B	DioBsy	0C6C	BUSIES	0C6E
DRVADTAB	0C7E	STACK	1000	SCREEN	1800	OVRLY	2000
OVENT	2004	SBUF1	2800	SBUF2	2900	SBUF3	2A00
SBUF4	2B00	CBUF	2C00	KBUF	2D00	CMND	2D40
MEMTOP	2D80	KBIP	2D82	KBIG	2D84	KBEX	2D86
CMDF	2D88	CMDD	2D89	CMDP	2D8A	CMDA	2D8C
CMDN	2D8E	DONT	2D90	SBRK	2D91	SYSRES	2D92
PVEC	2D93	UVEC	2D95	UBRK	2D97	SCHR	2D98
UCHR	2D99	ERROR	2D9A	LERR	2D9C	JOBST	2D9E
NDRIVES	2D9F	NFDIR	2DA0	NFCK	2DA1	MTO	2DA2
MUNG1	2DA7	MUNG2	2DA9	MUNG3	2DAB	MUNG4	2DAD
EXECSP	2DAF	USTATS	2DB1	SINT	2DB3	OVNM	2DB6
OVHL	2DBD	OVDE	2DBF	OVBC	2DC1	OVPSW	2DC3
ONCE	2DC5	LUSER	2DC6	CMPTR	2DC7	EFLG1	2DC9
EFLG2	2DCA	FILE	2DCB	BUGS	2DFC	NFA	2E00
DirAddr	2E02	PATH	2E04	DEFPATH	2E27	OVMEM	2E53
USER	3200						

Twin User Macro Definitions:

ALIGN	clrp	clrt	db
dequ	devlock	dw	enter
gfid	giveup	gover	ioret
leave	lock	overlay	overto
print	ralign	rddef	rds
rorg	setp	sett	show
unlock	userpgm	vcb	vect
verdate			

Twin User Labels in Alphabetical Order:

BHA	E05F	BOOTVOL	2D92	BRG	0C60	BRGEN	0004
BUGS	2EB4	BUSIES	0C6E	Byte	E05A	CBUF	2C00
CHUNK	2EFE	CMDA	2E82	CMDD	2E7F	CMDF	2E7E
CMDN	2E84	CMDP	2E80	CMND	2E3C	CMPTR	2E7C
Ckdr	E02D	Cold	E000	DBARF	0020	DEFPATH	2D80
DEVMASK	000F	DONT	2E8E	DRVADTAB	0C7E	Deout	E045
Devlock	E06F	Dio	2E30	DioA	0C66	DioBsy	0C6C
DioDn	0C6B	DioDrv	0C69	DioHL	0C67	DirAddr	2EA7
EERR	0040	EFLG1	2EB0	EFLG2	2EB1	EIC	0080
ERROR	2EB7	EXECSP	2EBD	Enter	E04E	Err	E00C
FILE	2EBF	Fdfp	E075	Flip	E02A	Flipem	E039
Flush	E01B	Fold	E027	Fpanel	E078	GFLOCK	E07E
Gdfp	E072	Gfid	E042	Giveup	E03C	Gover	E012
IOIP	0C62	Iexec	E030	Ioret	E066	JOBST	2EB2
KBEX	2E38	KBIG	2E36	KBIP	2E34	KBMODE1	2E3A
KBMODE2	2E3B	KBUF	2DC0	Killi	E018	LERR	2EB9
~~LOCK~~	~~2E8F~~	~~LUSER~~	~~2EB3~~	~~Leave~~	~~E051~~	~~Lock~~	~~E048~~
Look	E01E	MEMTOP	2EBB	MUNG1	2E9F	MUNG2	2EA1
MUNG3	2EA3	MUNG4	2EA5	MUNGP	2D00	Mfos	E063
Move	E06C	Moven	E069	Msg	E006	Mtos	E060
NDRIVES	2EA9	NFA	2EAD	NFCK	2EAC	NFDIR	2EAB
ONCE	2EAF	OVBC	2E9B	OVDE	2E99	OVENT	2004
OVHL	2E97	OVNM	2E90	OVPSW	2E9D	OVRLY	2000
Ovrto	E00F	PATH	2D40	PHANTOM	0060	PMASK	E05D
POS	2E0A	PVEC	2E87	Pmsg	E009	Print	E036
Rlwe	E024	Rtn	E015	Runr	E021	SBRK	2E86
SBUF1	2800	SCEND	2E08	SCHR	2E8C	SCRHM	2E09
SRA1	0C10	SRA2	0C12	SRA3	0C14	SRA4	0C16
SRA5	0C18	SRA7	2EEC	SRA7I	0C1C	SUWH0	0C20
SUWH1	0C24	SUWH2	0C28	SUWH3	0C2C	SUWH4	0C30
SUWH5	0C34	SUWH6	0C38	SUWH7	0C3C	SUWH8	0C40
SUWH9	0C44	SYSRES	2EAA	Show	E033	TIMER	2E02
Ticker	E03F	UBRK	2E8B	UCHR	2E8D	USER	2F00
USERS	0002	USP	2E00	USRNAME	2EEE	USTATS	E05E
UTIME	2E02	UVEC	2E89	Unlock	E04B	VCBTAB	0C63
Vmgr	E054	Vti	E057	WAKEUP	2E06	WH0	2E0C
WH1	2E10	WH2	2E14	WH3	2E18	WH4	2E1C
WH5	2E20	WH6	2E24	WH7	2E28	WH8	2E2C
WH9	2E30	WHICH	0C61	Warm	E003	XTIMER	0C00
cmdf	0001	excl	0007	fupd	0004	gLook	E07B
mung	0005	rd	0002	wlock	0006	wrt	0003

Twin User Labels sorted by Value:

cmdf	0001	rd	0002	USERS	0002	wrt	0003
fupd	0004	BRGEN	0004	mung	0005	wlock	0006
excl	0007	DEVMASK	000F	DBARF	0020	EERR	0040
PHANTOM	0060	EIC	0080	XTIMER	0C00	SRA1	0C10
SRA2	0C12	SRA3	0C14	SRA4	0C16	SRA5	0C18
SRA7I	0C1C	SUWH0	0C20	SUWH1	0C24	SUWH2	0C28
SUWH3	0C2C	SUWH4	0C30	SUWH5	0C34	SUWH6	0C38
SUWH7	0C3C	SUWH8	0C40	SUWH9	0C44	BRG	0C60
WHICH	0C61	IOIP	0C62	VCBTAB	0C63	DioA	0C66
DioHL	0C67	DioDrv	0C69	DioDn	0C6B	DioBsy	0C6C
BUSIES	0C6E	DRVADTAB	0C7E	OVRLY	2000	OVENT	2004
SBUF1	2800	CBUF	2C00	MUNGP	2D00	PATH	2D40
DEFPATH	2D80	BOOTVOL	2D92	KBUF	2DC0	USP	2E00
UTIME	2E02	TIMER	2E02	WAKEUP	2E06	SCEND	2E08
SCRHM	2E09	POS	2E0A	WH0	2E0C	WH1	2E10
WH2	2E14	WH3	2E18	WH4	2E1C	WH5	2E20
WH6	2E24	WH7	2E28	WH8	2E2C	WH9	2E30
Dio	2E30	KBIP	2E34	KBIG	2E36	KBEX	2E38
KBMODE1	2E3A	KBMODE2	2E3B	CMND	2E3C	CMPTR	2E7C
CMDF	2E7E	CMDD	2E7F	CMDP	2E80	CMDA	2E82
CMDN	2E84	SBRK	2E86	PVEC	2E87	UVEC	2E89
UBRK	2E8B	SCHR	2E8C	UCHR	2E8D	DONT	2E8E
LOCK	2E8F	OVNM	2E90	OVHL	2E97	OVDE	2E99
OVBC	2E9B	OVPSW	2E9D	MUNG1	2E9F	MUNG2	2EA1
MUNG3	2EA3	MUNG4	2EA5	DirAddr	2EA7	NDRIVES	2EA9
SYSRES	2EAA	NFDIR	2EAB	NFCK	2EAC	NFA	2EAD
ONCE	2EAF	EFLG1	2EB0	EFLG2	2EB1	JOBST	2EB2
LUSER	2EB3	BUGS	2EB4	ERROR	2EB7	LERR	2EB9
MEMTOP	2EBB	EXECSP	2EBD	FILE	2EBF	SRA7	2EEC
USRNAME	2EEE	CHUNK	2EFE	USER	2F00	Cold	E000
Warm	E003	Msg	E006	Pmsg	E009	Err	E00C
Ovrto	E00F	Gover	E012	Rtn	E015	Killi	E018
Flush	E01B	Look	E01E	Runr	E021	Rlwe	E024
Fold	E027	Flip	E02A	Ckdr	E02D	Iexec	E030
Show	E033	Print	E036	Flipem	E039	Giveup	E03C
Ticker	E03F	Gfid	E042	Deout	E045	Lock	E048
Unlock	E04B	Enter	E04E	Leave	E051	Vmgr	E054
Vti	E057	Byte	E05A	PMASK	E05D	USTATS	E05E
BHA	E05F	Mtos	E060	Mfos	E063	Ioret	E066
Moven	E069	Move	E06C	Devlock	E06F	Gdfp	E072
Fdfp	E075	Fpanel	E078	gLook	E07B	GFLOCK	E07E

Section 13

Disk I/O in Assembly Programs

The code in this section is the disk I/O code used to open input and output files, and to read and write on the disk.

```
;
; ***** DISK I/O *****
;
;       Read from HL buffer to get file names to open for
;  both input and output (respectively).
;
OPEN    CALL    NXTC            ;read command line
        JNZ     OPEN            ;Scan to delimiter
        XRA     A               ;clear input/output file flags
        STA     INFLG
        STA     OTFLG
        CALL    OPIN            ;Open input file
        JC      Err             ;Ret/ error - mail out.
        MOV     A,M             ;is next character a CR
        CPI     CR
        RZ                      ;Output file is optional.
        LXI     B,'GO'          ;Default extension.
        CALL    OPOUT           ;Open output file.
        JC      Err
        RET
;
;       Open an input file.  Read from buffer at HL to get
;  filename.
;
OPIN    SHLD    FSTPTR          ;Save ptr
        MVI     A,CR
        DCX     H
        CMP     M               ;We looking at CR yet?
        JZ      ER0703          ;Yes - error.
        INX     H
        LXI     D,IFDE          ;Input File Dir Entry buffer
        MVI     A,40H
        gfid                    ;call Gfid with Macro
        JC      ER0705          ;can't find input file
        MVI     A,1
        STA     INFLG           ;set input flag
        PUSH    H               ;Save ptr to input
        LXI     H,IFDE+1        ;--> input FDE
        LXI     D,3             ;set DE to 3 to add length
        MOV     A,M             ;get path name length byte
        ANI     1FH             ;mask off Sys, Del, New bits
        ADD     E               ;add it in
        MOV     E,A
```

```
        DAD     D               ;--> FDA slot in FDE
        SHLD    IFDP            ;save Input File Disk Pointer.
        LXI     D,IFDA          ;In File Disk Address
        MVI     B,4
OPLP    MOV     A,M
        STAX    D               ;copy disk address & length
        INX     H
        INX     D
        DCR     B
        JNZ     OPLP            ;Copy FDE good stuff.
        LHLD    IFDA            ;Initialize REWIND variables.
        SHLD    REW1            ;disk address
        LHLD    INBS            ;file length
        SHLD    REW2            ;sector counter
        LXI     H,INBUF+0FFH    ;disk buffer
        SHLD    IPTR            ;input pointer
        POP     H               ;Restore H to ptr
        RET
;
;       Open an output file.  BC=Default Extension.
;  Skip the first char, assuming it is a delimiter.
;
OPOUT   MVI     A,CR
        CMP     M
        INX     H               ;HL-> 2nd FD if there
        JNZ     TWOFDS          ;jmp/ two FD'S given
        LHLD    FSTPTR          ;--> First FD in buffer.
TWOFDS  LXI     D,OFDE          ;Setup for Gfid
        MVI     A,60H           ;Lookup function.
        gfid                    ;call Gfid with Macro.
                                ;Default EXT passed in BC as
                                ;a parameter by caller.
        JNC     ER0702          ;If already there, report.
        PUSH    H
        LXI     H,0300H         ;Must get a 300H error
        CALL    DCMP            ;normally.
        POP     H
        JNZ     FAIL            ;If not 300H, ret CARRY.

; For two user systems only, get MUNG access to device.

        IF      USERS=2
        LDA     OFDE
        ANI     15              ; device #
        MOV     C,A
        sett    mung            ; see if we can MUNG it
        JC      FAIL            ; crap out if we can't
        ENDIF

        MVI     A,1
```

```
        STA     OTFLG
        LXI     H,OFDE+1        ;--> output FDE
        LXI     D,3
        MOV     A,M
        ANI     1FH             ;mask name length
        ADD     E
        MOV     E,A
        DAD     D               ;--> FDA in putput FDE
        SHLD    OFDP            ;Save pointer
        LXI     D,OFDA
        MOV     A,M
        STAX    D
        INX     H
        INX     D
        MOV     A,M
        STAX    D               ;copy FDA
        LXI     H,Ø
        SHLD    ONBS            ;Clear output number sectors.
        LXI     H,OTBUF
        SHLD    OPTR            ;Clear output buffer pointer.
        RET
;
; Get a character from disk input file.
;
GETC    CALL    GETBYTE         ;Get a binary byte.
        RC                      ;If empty file, ret CARRY.
        ORA     A               ;Skip zeroes.
        JZ      GETC
        STC
        CMC                     ;Ret NO CARRY.
        RET
;
GETBYTE LDA     INFLG
        ORA     A
        JZ      ERØ7Ø3          ;No input file.
        PUSH    B
        PUSH    D
        PUSH    H
        LHLD    IPTR
        INR     L               ;Bump bottom byte (!!!!RLD).
        JZ      GTBUF           ;jmp/ fill tank
        SHLD    IPTR
GTGNG   MOV     A,M
GOAWY   POP     H
        POP     D
        POP     B
        RET
;
GTBUF   LHLD    INBS            ;Sector counter.
        MOV     A,H
```

```
        ORA     L
        JNZ     GTBF1           ;jmp/ more out there.
        STC
        JMP     GOAWY           ;Ret CARRY on EOF.
;
GTBF1   DCX     H
        SHLD    INBS            ;bump sector ctr.
        LHLD    IFDA
        INX     H
        SHLD    IFDA            ;Inc disk address.
        DCX     H
        LDA     IFDE
        ANI     15              ;Unit #
        MOV     C,A
        MVI     A,1             ;1 sector
        LXI     D,INBUF
        MOV     B,A             ;Read
        CALL    Dio
        JC      Err             ;Oops.
        LXI     H,INBUF
        SHLD    IPTR            ;Reset ptr
        JMP     GTGNG
;
;  Put a character to the output file.
;
PUTC    PUSH    B
        PUSH    D
        PUSH    H
        MOV     B,A
        PUSH    B
        LDA     OTFLG
        ORA     A               ;Is there an output file?
        JZ      PTAWY           ;Jmp/ nope.
        LHLD    OPTR
        MOV     M,B
        INR     L
        SHLD    OPTR
        JNZ     PTAWY           ;jmp/ not full
        LHLD    OFDA            ;output disk address
        LDA     OFDE
        ANI     15              ;Unit number
        MOV     C,A             ;in C.
        LXI     D,OTBUF         ;Flush out buff.
        MVI     A,1             ;Tell Dio to move 1 sector.
        MVI     B,Ø             ;Write command.
        CALL    Dio
        JC      Err             ;Oops!
        LHLD    OFDA
        INX     H
        SHLD    OFDA            ;Inc output disk address.
```

```
        LHLD    ONBS
        INX     H
        SHLD    ONBS            ;and output file sector ctr.
        XRA     A               ;Set zero flag.
        ;
PTAWY   POP     B
        MOV     A,B             ;Save A but not flags.
        POP     H
        POP     D
        POP     B
        RET
;
; Rewind the input file.
;
REWIND  PUSH    H
        LHLD    REW1
        SHLD    IFDA
        LHLD    REW2
        SHLD    INBS
        LXI     H,INBUF+0FFH
        SHLD    IPTR
        POP     H
        RET
;
; Close output file.
;
SHUT    LDA     OTFLG
        ORA     A
        RZ                      ;Ignore if no output file.
        LHLD    OPTR            ;Clear rest of buffer.
        DCR     L
        INR     L               ;Set Z according to L.
        JZ      SHUT3           ;If already full, quit.
SHUT2   XRA     A
        CALL    PUTC            ;Put zeroes until full.
        JNZ     SHUT2
SHUT3   LDA     OFDE+1          ;Get name length.
        ANI     31              ;Leave only name length.
        ORI     20H             ;Set the 'new' bit.
        STA     OFDE+1
        LHLD    OFDP            ;-->FDA slot
        INX     H
        INX     H               ;--> NBS slot
        LXI     D,ONBS          ;--> source block
        MVI     B,6
SHTLP   LDAX    D               ;Move NBS, LA & SA to FDE.
        MOV     M,A
        INX     H
        INX     D
        DCR     B
```

```
        JNZ     SHTLP
;
        LXI     H,OFDE
        MVI     A,1
                                ;ENTER function
        gfid                    ;call Gfid with Macro
        JC      Err
;
        IF      USERS=2         ;for two user
        LDA     OFDE            ;clear mung access
        ANI     15
        MOV     C,A
        clrt    mung
        JC      Err
        ENDIF
;
        RET
;
; Disk I/O utilities.
;
NXTC    MOV     A,M
        INX     H
        CPI     CR
        RZ
        CPI     TAB
        RZ
        CPI     ' '
        RZ
        CPI     ','
        RET
;
; Disk I/O errors.
;
ER0702  MVI     E,2     ;** Output file already exists **
        DB      21H
ER0703  MVI     E,3     ;** No input file specified **
        DB      21H
ER0705  MVI     E,5     ;** Input file does not exist **
        MVI     D,07H
        STC             ;Return waving the flag.
        RET
;
; Error processing
;
FAIL    JMP     Err
;
; Data Area
;
INFLG   DS      1       ;input file flag
OTFLG   DS      1       ;output file flag
```

```
FSTPTR  DS      2       ;pointer save location
;
; For Input File
IFDE    DS      44      ;dir entry buffer
IFDP    DS      2       ;disk pointer
IFDA    DS      2       ;disk address
INBS    DS      2       ;sector length
IPTR    DS      2       ;read pointer
REW1    DS      2       ;rewind pointer disk address
REW2    DS      2       ;rewind pointer sector counter
INBUF   DS      256     ;disk sector buffer
;
; For Output File
OFDE    DS      44      ;dir entry buffer
OFDP    DS      2       ;disk pointer
OFDA    DS      2       ;disk address
ONBS    DS      2       ;sector length
OFLA    DS      2       ;load address
OFSA    DS      2       ;start address
OPTR    DS      2       ;read pointer
OTBUF   DS      256     ;disk sector buffer
;
        END
```

Section 14

Sample Assembly Program

The following program is a sample of an assembly language program that does disk I/O. This particular program is included on the system programmer's disk as RECOVER.GO. Refer to Section 4, Utilities for the System Programmer, for instructions on this program.

```
                        ;*********************************************
                        ;*                                           *
                        ;*                  RECOVER                  *
                        ;*                                           *
                        ;*      Version 1.Ø             Donald Moe   *
                        ;*      Ø3/Ø9/81                             *
                        ;*                                           *
                        ;* Copyright (c) 1981, Interactive Products  *
                        ;* Corporation dba PolyMorphic Systems       *
                        ;* 46Ø Ward Dr., Santa Barbara, CA.  93111   *
                        ;*                                           *
                        ;*********************************************
                        ;
                                REFS    SYSTEM  ;get system symbol file
                                REF
                        ;
ØØØD                    CR      EQU     13
                        ;
                                MACLIST Ø
                        ;
                        emsg    MACRO
                        #L      CALL    Gover
                                db      'Emsg'
                                ENDM
                        ;
                        msg     MACRO
                        #L      LXI     H,#1
                                CALL    Msg
                                ENDM
                        ;
                                MACLIST Ø       ;suppress listing of
                                                ;macro expansions
                        ;
3200                            ORG     USER
3200                            IDNT    $,$
                        ;
3200  C3F833                   JMP     START
3203  C3F833                   JMP     START
```

```
                        ;
                       +Hello   db      'File Recovery Program. '
                                db      'Version 3/5/81',0
                       +Msg0    db      CR,'Enter Disk Drive Number: ',0
                       +Hexmsg  db      CR,'Enter the following numbers'
                                db      ' in Hexadecimal.',0
                       +Msg1    db      CR,'Enter Starting Sector'
                                db      ' Number: ',0
                       +Msg2    db      CR,'Enter Ending Sector'
                                db      ' Number: ',0
                       +Msg3    db      CR,'Enter Program Load'
                                db      ' Address: ',0
                       +Msg4    db      CR,'Enter Program Start'
                                db      ' Address: ',0
                       +Msg5    db      CR,'Enter New Filename with'
                                db      ' Extension: ',0
                       +Msg6    db      'Working...please stand by.'
                                db      CR,0
                       +Msg7    db      CR,'Recovery Completed.'
                                db      ' Another (Y/N)? ',0
                       +Msg8    db      'At Sector: ',0
                       +Errmsg  db      CR,'* * Illegal answer.'
                                db      ' Reenter. * *',0
                       +Outerr  db      '* * Output file already'
                                db      ' exists. * *',0
                       +Sectmsg db      CR,'* * Starting sector greater'
                                db      ' than Ending sector. * *',0
                       +Busy    db      '* * Output device busy. * *',0
                        ;
                       +START   msg     Hello   ;sign on
33FE  AF                        XRA     A       ;clear output flag
33FF  32BE35                    STA     OTFLG
                        ;
                        ; ask for drive number
3402  112C32           MSG0     LXI     D,Msg0  ;drive number message
3405  0E05                      MVI     C,5     ;five chars
3407  CD0A35                    CALL    NUMB    ;get drive number
340A  7D                        MOV     A,L     ;put into A
340B  B7                        ORA     A       ;is it zero
340C  CA1434                    JZ      MSG01   ;yep/ not legal
340F  FE0A                      CPI     10      ;is it less than 10
3411  DA1734                    JC      MSG02   ;nope/ not legal
3414  C38B34           MSG01    JMP     MSGERR  ;display error message
3417  32C735           MSG02    STA     DRIVE   ;save it
                        ;
                       +        msg     Hexmsg  ;display hex message
                        ;
                        ; ask for starting sector number
3420  117432           MSG1     LXI     D,Msg1  ;start sect # msg
3423  0E06                      MVI     C,6     ;six chars
3425  CD0A35                    CALL    NUMB    ;get sector number
```

```
3428  22C835            SHLD    STSECT  ;save it
                        ;
                        ; ask for ending sector number
342B  119432    MSG2    LXI     D,Msg2  ;end sect # msg
342E  0E06              MVI     C,6
3430  CD0A35            CALL    NUMB    ;get sector number
3433  22CA35            SHLD    ENSECT  ;save it
                        ;
                        ;check for start <= end sector number
3436  EB                XCHG            ;put endsect into DE
3437  2AC835            LHLD    STSECT  ;get start sector
343A  EB                XCHG            ;swapem back
343B  CD4135            CALL    DCMP    ;compare them
343E  D24A34            JNC     MSG3    ;no carry => Ok
                        ;
                      + msg     Sectmsg ;display error message
3447  C32034            JMP     MSG1    ;go try again
                        ;
                        ; ask for file Load Address
344A  11B232    MSG3    LXI     D,Msg3  ;load addr message
344D  0E06              MVI     C,6
344F  CD0A35            CALL    NUMB
3452  22FE35            SHLD    LOADADR ;save it
                        ;
                        ; ask for file Start Address
3455  11D032    MSG4    LXI     D,Msg4  ;start addr message
3458  0E06              MVI     C,6
345A  CD0A35            CALL    NUMB
345D  220036            SHLD    STADR   ;save it
                        ;
                        ; ask for new filename through Gfid
3460  21EF32    MSG5    LXI     H,Msg5  ;filename prompt
3463  11CC35            LXI     D,OFDE  ;buffer
3466  015854            LXI     B,'TX'  ;default extension TX
3469  3EE0              MVI     A,0E0H  ;read from user,
                                        ;look it up,
                                        ; default extension
                      + gfid            ;call Gfid
3472  D29434            JNC     MSGERR1 ;file already there
3475  210003            LXI     H,300H  ;check for 300 error
3478  CD4135            CALL    DCMP    ;compare them
347B  CAA634            JZ      READIT  ;yep/ Ok
                      + emsg            ;call Emsg
3485  DA0F04            JC      Err     ;if error now, abort
3488  C36034            JMP     MSG5    ;loop back
                        ;
                        ; Illegal answer message
                     +MSGERR  msg     Errmsg
3491  C30234            JMP     MSG0
                        ;
                        ; Output file exists message
```

```
                    +MSGERR1 msg     Outerr
349A  C36034                JMP     MSG5
                     ;
                     ; Device busy message
                    +FAIL    msg     Busy
34A3  C36034                JMP     MSG5
                     ;
                     ; Now we have the file looked up, and the
                     ; sector numbers and addresses determined.
                     ; Let's now start reading it.
                     ;
                    +READIT  msg     Msg6    ;put out working message
                    +        msg     Msg8    ;display "at sector"
34B2  CD4735                CALL    OUTSET  ;setup output indexes
34B5  DA9D34                JC      FAIL    ;if carry, device busy
34B8  2ACA35                LHLD    ENSECT  ;get ending sector
34BB  EB                    XCHG            ;swap into DE
34BC  2AC835                LHLD    STSECT  ;get start sector
34BF  E5             READLP PUSH    H
34C0  D5                    PUSH    D
34C1  CDF034                CALL    SECTDISP ;show sector
34C4  CD6635                CALL    GETIN   ;get sector
34C7  CD7735                CALL    PUTOUT  ;put it out
34CA  D1                    POP     D
34CB  E1                    POP     H
34CC  CD4135                CALL    DCMP    ;compare them
34CF  23                    INX     H       ;increment sector number
34D0  C2BF34                JNZ     READLP  ;loop until done
34D3  CD9C35                CALL    SHUT    ;close output file
                     ;
34D6  113033        AGAIN   LXI     D,Msg7  ;display end message
34D9  21BF35                LXI     H,RLWEBUF ;buffer
34DC  010200                LXI     B,02    ;2 chars, echo CR
34DF  CD2704                CALL    Rlwe    ;read from user
34E2  11BF35                LXI     D,RLWEBUF ;point to buffer
34E5  CD3935                CALL    LCFLD   ;convert to upper case
34E8  FE59                  CPI     'Y'     ;yes
34EA  CAF833                JZ      START
34ED  C30304                JMP     Warm    ;else no
                     ;
                     ; Show current sector being processed
34F0                 SECTDISP
34F0  E5                    PUSH    H       ;save regs
34F1  D5                    PUSH    D
34F2  3E20                  MVI     A,' '   ;put out space
34F4  CD240C                CALL    WH1
34F7  EB                    XCHG            ;put sector number into
DE
34F8  CDD103                CALL    Deout   ;display it
34FB  2A0E0C                LHLD    POS     ;get cursor location
34FE  367F                  MVI     M,127   ;blank it
```

```
3500  11FBFF          LXI     D,-5    ;back up 5
3503  19              DAD     D
3504  220E0C          SHLD    POS     ;reset for next pass
3507  D1              POP     D
3508  E1              POP     H
3509  C9              RET
                ;
                ;
                ; Hex input routines
                ;
                ; NUMB reads a hex number using Rlwe
                ; On Entry:     DE points to prompt string
                ;               C number of chars to read, max
                ; On Exit:      HL contains hex number
                ;
350A  21BF35    NUMB    LXI     H,RLWEBUF ;where to put chars
350D  0601              MVI     B,1     ;don't echo term char
350F  CD2704            CALL    Rlwe    ;read from user
3512  11BF35            LXI     D,RLWEBUF ;point at buffer
                ;
                ;fall into conversion
                ;
                ; HEXC converts a variable length hex number in
                ; RLWEBUF
                ;
3515  210000    HEXC    LXI     H,0     ;zero conversion buffer
3518  4D                MOV     C,L
3519  CD3935    NXNYB   CALL    LCFLD   ;get case-folded char
351C  47                MOV     B,A     ;save it
351D  FE30              CPI     '0'     ;return if less than '0'
351F  D8                RC
3520  D630              SUI     '0'     ;convert to binary
3522  FE0A              CPI     10
3524  DA2F35            JC      NXNB1
3527  D607              SUI     7
3529  FE0A              CPI     10
352B  D8                RC              ;return if not hex
352C  FE10              CPI     16
352E  D0                RNC             ;return if not hex
352F  0C        NXNB1   INR     C       ;incr count of chars
3530  29                DAD     H       ;shift over result
3531  29                DAD     H
3532  29                DAD     H
3533  29                DAD     H
3534  B5                ORA     L       ;or in next digit
3535  6F                MOV     L,A
3536  C31935            JMP     NXNYB
                ;
3539  1A        LCFLD   LDAX    D       ;get next char
353A  13                INX     D       ;bump pointer
353B  FE60              CPI     060H    ;if upper case, skip it
```

```
353D  D8              RC
353E  D620            SUI     20H     ;fold it to upper
3540  C9              RET
                ;
                ; Compare DE to HL for equality
3541  7C        DCMP  MOV     A,H
3542  BA              CMP     D
3543  C0              RNZ
3544  7D              MOV     A,L
3545  BB              CMP     E
3546  C9              RET
                ;
                ;
                ; ***** DISK I/O *****
                ;
                ; For two user systems only, get MUNG access to
                ; device.
3547            OUTSET
0000                    IF      USERS=2
                        LDA     OFDE
                        ANI     15      ; device #
                        MOV     C,A
                        sett    mung    ; see if we can MUNG it
                        RC              ; if carry, we can't
                        ENDIF
                ;
3547  21CD35            LXI     H,OFDE+1 ;--> output FDE
354A  110300            LXI     D,3
354D  7E                MOV     A,M
354E  E61F              ANI     1FH     ;mask name length
3550  83                ADD     E
3551  5F                MOV     E,A
3552  19                DAD     D       ;--> FDA in output FDE
3553  22F835            SHLD    OFDP    ;Save FDE pointer
3556  11FA35            LXI     D,OFDA
3559  7E                MOV     A,M
355A  12                STAX    D
355B  23                INX     H
355C  13                INX     D
355D  7E                MOV     A,M
355E  12                STAX    D       ;copy FDA
355F  210000            LXI     H,0
3562  22FC35            SHLD    ONBS    ;Clear number sectors.
3565  C9                RET
                ;
                ; get next sector from input disk
                ; HL has sector number to read
3566  3AC735    GETIN   LDA     DRIVE   ;Unit #
3569  4F               MOV     C,A
356A  3E01              MVI     A,1     ;1 sector
356C  110436            LXI     D,OTBUF ;buffer to read into
```

```
356F  47              MOV     B,A       ;Read
3570  CD0604          CALL    Dio       ;do disk I/O
3573  DA0F04          JC      Err       ;Oops.
3576  C9              RET
                ;
                ;  Put a sector to the output file.
                ;
3577  2AFA35    PUTOUT  LHLD    OFDA      ;output disk address
357A  3ACC35          LDA     OFDE
357D  E60F            ANI     15        ;Unit number
357F  4F              MOV     C,A       ;in C.
3580  110436          LXI     D,OTBUF   ;Flush out buff.
3583  3E01            MVI     A,1       ;do 1 sector
3585  0600            MVI     B,0       ;Write command.
3587  CD0604          CALL    Dio
358A  DA0F04          JC      Err       ;Oops!
358D  2AFA35          LHLD    OFDA
3590  23              INX     H
3591  22FA35          SHLD    OFDA      ;Inc disk address.
3594  2AFC35          LHLD    ONBS
3597  23              INX     H
3598  22FC35          SHLD    ONBS      ;put file sector ctr.
359B  C9              RET
                ;
                ; Close output file.
                ;
359C  2AF835    SHUT    LHLD    OFDP      ;-->FDA slot
359F  23                INX     H
35A0  23                INX     H         ;--> NBS slot
35A1  11FC35            LXI     D,ONBS    ;--> source block
35A4  0606              MVI     B,6
35A6  1A        SHTLP   LDAX    D         ;Move NBS, LA & SA
                                          ;to FDE.
35A7  77                MOV     M,A
35A8  23                INX     H
35A9  13                INX     D
35AA  05                DCR     B
35AB  C2A635            JNZ     SHTLP
                ;
35AE  21CC35            LXI     H,OFDE
35B1  3E01              MVI     A,1
                +       gfid              ;ENTER it
35BA  DA0F04            JC      Err
                ;
0000                    IF      USERS=2   ;for two user system
                        LDA     OFDE      ;clear mung access
                        ANI     15
                        MOV     C,A
                        clrt    mung
                        JC      Err       ;if carry, we can't
                        ENDIF
```

```
35BD  C9                RET
                        ;
                        ;
                        ;
                        ; Data Area
                        ;
35BE            OTFLG   DS      1       ;output file flag
35BF            RLWEBUF DS      8       ;input buffer for Rlwe
                        ;
                        ; For Input File
35C7            DRIVE   DS      1       ;drive number
35C8            STSECT  DS      2       ;starting sector number
35CA            ENSECT  DS      2       ;ending sector number
                        ;
                        ; For Output File
35CC            OFDE    DS      44      ;dir entry buffer
35F8            OFDP    DS      2       ;disk pointer
35FA            OFDA    DS      2       ;disk address
35FC            ONBS    DS      2       ;sector length
35FE            LOADADR DS      2       ;file load address
3600            STADR   DS      2       ;file start address
3602            OPTR    DS      2       ;read pointer
3604            OTBUF   EQU     $       ;disk sector buffer
                        ;
3704            end     EQU     OTBUF+256
                        ;
                        END

Error total =     0

        Macros defined in this assembly:

db          emsg        gfid        msg
overlay

        Labels defined in this assembly:

AGAIN       34D6 BUGS       2DFC BUSIES     0C6E Busy       33DC
CBUF        2C00 CMDA       2D8C CMDD       2D89 CMDF       2D88
CMDN        2D8E CMDP       2D8A CMND       2D40 CMPTR      2DC7
CR          000D Ckdr       0433 Command    0C4C DBARF      0020
DCMP        3541 DEFPATH    2E27 DEOUT      03D1 DEVMASK    000F
DONT        2D90 DRIVE      35C7 DRVADTAB   0C7E Deout      03D1
Dhalt       0409 Dio        0406 DioA       0C66 DioBsy     0C6C
DioDn       0C6B DioDrv     0C69 DioHL      0C67 DirAddr    2E02
EERR        0040 EFLG1      2DC9 EFLG2      2DCA EIC        0080
ENSECT      35CA ERROR      2D9A EXECSP     2DAF Err        040F
Errmsg      3361 FAIL       349D FILE       2DCB Flip       042D
Flush       041E Fold       042A GETIN      3566 Gover      0415
```

HEXC	3515	Hello	3206	Hexmsg	3247	Iexec	0436
Ioret	0064	JOBST	2D9E	KBD	0018	KBEX	2D86
KBIG	2D84	KBIP	2D82	KBUF	2D00	Killi	041B
LCFLD	3539	LERR	2D9C	LOADADR	35FE	LUSER	2DC6
Look	0421	MEMTOP	2D80	MSG0	3402	MSG01	3414
MSG02	3417	MSG1	3420	MSG2	342B	MSG3	344A
MSG4	3455	MSG5	3460	MSGERR	348B	MSGERR1	3494
MTO	2DA2	MUNG1	2DA7	MUNG2	2DA9	MUNG3	2DAB
MUNG4	2DAD	MemAdd	0C49	Msg	040C	Msg0	322C
Msg1	3274	Msg2	3294	Msg3	32B2	Msg4	32D0
Msg5	32EF	Msg6	3314	Msg7	3330	Msg8	3355
NDRIVES	2D9F	NFA	2E00	NFCK	2DA1	NFDIR	2DA0
NUMB	350A	NXNB1	352F	NXNYB	3519	OFDA	35FA
OFDE	35CC	OFDP	35F8	ONBS	35FC	ONCE	2DC5
OPTR	3602	OTBUF	3604	OTFLG	35BE	OUTSET	3547
OVBC	2DC1	OVDE	2DBF	OVENT	2004	OVHL	2DBD
OVMEM	2E53	OVNM	2DB6	OVPSW	2DC3	OVRLY	2000
Outerr	3383	Ovrto	0412	PATH	2E04	POS	0C0E
PUTOUT	3577	PVEC	2D93	Pages1	0C4B	READIT	34A6
READLP	34BF	RLWEBUF	35BF	R1gc	0430	R1we	0427
Rtn	0418	Runr	0424	SBRK	2D91	SBUF1	2800
SBUF2	2900	SBUF3	2A00	SBUF4	2B00	SCEND	0C1E
SCHR	2D98	SCREEN	1800	SCRHM	0C1F	SECTDISP	34F0
SHTLP	35A6	SHUT	359C	SINT	2DB3	SRA1	0C10
SRA2	0C12	SRA3	0C14	SRA4	0C16	SRA5	0C18
SRA7	0C1C	STACK	1000	STADR	3600	START	33F8
STSECT	35C8	SYSRES	2D92	Sectmsg	33A7	TIMER	0C00
UBRK	2D97	UCHR	2D99	USER	3200	USERS	0001
USTATS	2DB1	UVEC	2D95	VCBTAB	0C63	VERLOC	0439
Version	0081	WAKEUP	0C1A	WH0	0C20	WH1	0C24
WH2	0C28	WH3	0C2C	WH4	0C30	WH5	0C34
WH6	0C38	WH7	0C3C	WH8	0C40	WH9	0C44
Warm	0403	end	3704				

INDEX