For Model 4/4P users . . . .

When you change disks in DRIVE Ø, you must press RESET if the disks contain different versions of the operating system. For example:

A change from 6.1.2 to 6.2.Ø requires RESET

A change from 6.2.Ø to 6.2.Ø does not require RESET

If you do not press RESET under these conditions, data on your diskette may be damaged.

Thank You!

Radio Shack®
A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102
8759314 TAP-08/84
8759554 TSA-08/84

To our customers . . .

This package contains:

- TRSDOS 6.2 diskette
- TRSDOS 6.2 Hard Disk Initialization diskette
- *Model 4/4P Disk System Owner's Manual* updated for TRSDOS 6.2
- Replacement pages for the *Model 4 Technical Reference Manual* that update the software section to TRSDOS 6.2
- *Hard Disk System Startup* for hard disk users, also updated for TRSDOS 6.2

The *Model 4/4P Disk System Owner's Manual* has been improved and new commands have been added. If you have been using an earlier version of this manual, look for these major changes and additions in your new manual:

CAT displays the directory for one or more drives. It works like DIR command with ALL parameter turned OFF.

CLS clears the screen.

HELP displays information about TRSDOS keywords. Using the HELP command's parameters, you can display additional information on the screen or printer.

LOG allows you to change from one system diskette to another. It is very useful when you want to change system diskettes without resetting the system.

TOF advances the printer to the top of the next page before printing. (TOF can only be used with certain printers.)

FLOPPY/DCT lets you define a logical drive as a floppy drive. Use it with the SYSTEM command.

Parameters have been added to the DIR, FORMAT, SYSTEM, and SETCOM commands. CLICK/FLT has an additional parameter, "CHAR."

With TRSDOS 6.2 you can create an Immediate Execution Program (IEP). Once you create an IEP, you can load and run it at the TRSDOS Ready prompt by pressing (*) (ENTER).

The cursor has been changed from a double underline to a single underline.

The default step rate has been changed to 6ms.

Pressing (CTRL) (R) at the TRSDOS Ready prompt repeats the previous TRSDOS command.

Note to non-U.S.A. users: Changing between 50 and 60 HERTZ is now done with the SYSTEM command instead of the HERTZ/JCL file.

Thank you!
Radio Shack
A Division of Tandy Corporation
TSA 875-9527
TAP 875-9306

Replacement Pages
for
Model 4 Technical Reference Manual

Attached are replacement pages for the *Model 4 Technical Reference Manual* to update the Software Section to TRSDOS 6.2.

If your *Model 4 Technical Reference Manual* has already been updated, it will have the following note on the copyright page:

Software Section updated to TRSDOS 6.2

If your manual has this note, you do not need the replacement pages. If it does not, insert the pages into your manual.

If you have not purchased a *Model 4 Technical Reference Manual*, but may in the future, keep these pages.

You may purchase the *Model 4 Technical Reference Manual* through your Radio Shack dealer.

Thank You

Radio Shack®
A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

TSA 8759526
TAP 8749529

In TRSDOS Version 6.2, this overlay contains the message "No ECI is present at SYS13" if you have not implemented an Extended Command Interpreter (ECI) or an Immediate Execution Program (IEP). You may purge this overlay if you do not intend to use an ECI or an IEP. See Appendix F, Using SYS13, for more information.

Utility Programs

BACKUP — Used to duplicate data from one disk to another.

COMM — A communications package for use with the RS-232C hardware.

CONV — Used to copy files from Model III TRSDOS to TRSDOS Version 6.

DOS/HLP — (Version 6.2 only) The data file used with the HELP utility.

FORMAT — Used to put track, sector, and directory information on a disk.

HELP/CMD — (Version 6.2 only) Used to provide on-line information about the TRSDOS commands.

LOG — Used to log in a double-sided diskette in Drive 0. Also updates the Drive Code Table information as with the DEVICE library command.

PATCH — Used to make changes to existing files.

REPAIR — Used to correct certain information on non-TRSDOS formatted diskettes.

TAPE100 — A disk/tape, tape/disk utility for cassette tape operations with the TRS-80 Model 100.

Device Driver Programs

COM/DVR — The RS-232C communications driver.

FLOPPY/DCT — Configures floppy drives in the system. Not needed with a floppy-only system.

JL/DVR — The Joblog driver program.

MEMDISK/DCT — Used to establish a pseudo floppy drive in memory.

Filter Programs

CLICK/FLT — Produces a short tone as each key is pressed.

FORMS/FLT — Used to select printer parameters and perform character translation.

KSM/FLT — The Keystroke Multiply feature, which allows the assigning of user-determined phrases to alphabetic keys.

Creating a Minimum Configuration Disk

All files except certain /SYS files may be purged from your Drive 0 disk. Additionally, if you place the needed /SYS files in high memory with the SYSTEM (SYSRES) command, it will be possible to run with a minimum configuration disk in Drive 0 after booting the system. Keep the following points in mind when purging system files:

- For operation, SYS files 1, 2, 3, 4, 10, and 12 should remain on the Drive 0 disk or be resident in memory.

- SYS2 must be on the system disk if a configuration file is to be loaded.
- SYS11 must be present only if any JCL files will be used.
- All three libraries (SYS files 6, 7, and 8) may be purged if no library command will be used.
- SYS5 and SYS9 may be purged if the system DEBUG package is not needed.
- SYSØ may be removed from any disk not used for booting.
- SYS11 (the JCL processor) and SYS6 (containing the DO library command) must both be on the disk if the DO command is to be used. Also, if you remove SYS6, you may as well remove SYS11.
- SYS13 may be removed if you have not implemented an ECI, an IEP file, or if you do not intend to use them.

The presence of any utility, driver, or filter program is dependent upon your individual needs. You can save most of the TRSDOS features in a configuration file using the SYSTEM (SYSGEN) command, so the driver and filter programs will not be needed in run time applications. If you intend to use the HELP utility, your disk must contain the DOS/HLP file.

The owner (update) passwords for TRSDOS files are as follows:

File Type	Extension	Owner Password
System files	(/SYS)	LSIDOS
Filter files	(/FLT)	FILTER
Driver files	(/DVR)	DRIVER
Utility files	(/CMD)	UTILITY
BASIC		BASIC
BASIC overlays	(/OV$)	BASIC
CONFIG/SYS		CCC
Drive Code Table Initializer	(/DCT)	UTILITY

5/Drive Access

Drive Code Table (DCT)

TRSDOS uses a Drive Code Table (DCT) to interface the operating system with specific disk driver routines. Note especially the fields that specify the allocation scheme for a given drive. This data is essential in the allocation and accessibility of file records.

The DCT contains eight 10-byte positions — one for each logical drive designated 0-7. TRSDOS supports a standard configuration of two-floppy drives. You may have up to four floppy drives. This is the default initialization when TRSDOS is loaded.

Here is the Drive Code Table layout:

DCT+0

This is the first byte of a 3-byte vector to the disk I/O driver routines. This byte is normally X'C3'. If the drive is disabled or has not been configured (see the SYSTEM command in the *Disk System Owner's Manual*), this byte is a RET instruction (X'C9').

DCT+1 and DCT+2

Contain the entry address of the routines that drive the physical hardware.

DCT+3

Contains a series of flags for drive specifications.

Bit 7 — Set to "1" if the drive is software write protected, "0" if it is not. (See the SYSTEM command in the *Disk System Owner's Manual.*)

Bit 6 — Set to "1" for DDEN (double density), or "0" for SDEN (single density).

Bit 5 — Set to "1" if the drive is an 8" drive. Set to "0" if it is a 5¼" drive.

Bit 4 — A "1" causes the selection of the disk's second side. The first side is selected if this bit is "0." This bit value matches the side indicator bit in the sector header written by the Floppy Disk Controller (FDC).

Bit 3 — A "1" indicates a hard drive (Winchester). A "0" denotes a floppy drive (5¼" or 8").

Bit 2 — Indicates the time delay between selection of a 5¼" drive and the first poll of the status register. A "1" value indicates 0.5 second and a "0" indicates 1.0 second. See the SYSTEM command in the *Disk System Owner's Manual* for more details.

If the drive is a hard drive, this bit indicates either a fixed or removable disk: "1" = fixed, "0" = removable.

Bits 1 and 0 — Contain the step rate specification for the Floppy Disk Controller. (See the SYSTEM command in the *Disk System Owner's Manual.*) In the case of a hard drive, this field may indicate the drive address (0-3).

DCT+4

Contains additional drive specifications.

Bit 7 — (Version 6.2 only) If "1", no @CKDRV is done when accessing the drive. If an application opens several files on a drive, this bit can be set to speed I/O on that drive after the first successful open is performed.

In versions prior to TRSDOS 6.2, this bit is reserved for future use. In order to maintain compatibility with future releases of TRSDOS, do not use this bit.

Bit 6 — If "1", the controller is capable of double-density mode.

Bit 5 — "1" indicates that this is a 2-sided floppy diskette; "0" indicates a 1-sided floppy disk. Do not confuse this bit with Bit 4 of DCT+3. This bit shows if the disk is double-sided; Bit 4 of DCT+3 tells the controller what side the current I/O is to be on.

If the hard drive bit (DCT+3, Bit 3) is set, a "1" denotes double the cylinder count stored in DCT+6. (This implies that a logical cylinder is made up of two physical cylinders.)

Bit 4 — If "1," indicates an alien (non-standard) disk controller.

Bits 0-3 — Contain the physical drive address by bit selection (0001, 0010, 0100, and 1000 equal logical Drives 0, 1, 2, and 3, respectively, in a default system). The system supports a translation only where no more than one bit can be set.

If the alien bit (Bit 4) is set, these bits may indicate the starting head number.

DCT+5

Contains the current cylinder position of the drive. It normally stores a copy of the Floppy Disk Controller's track register contents whenever the FDC is selected for access to this drive. It can then be used to reload the track register whenever the FDC is reselected.

If the alien bit (DCT+4, Bit 4) is set, DCT+5 may contain the drive select code for the alien controller.

DCT+6

Contains the highest numbered cylinder on the drive. Since cylinders are numbered from zero, a 35-track drive is recorded as X'22,' a 40-track drive as X'27,' and an 80-track drive as X'4F.' If the hard drive bit (DCT+3, Bit 3) is set, the true cylinder count depends on DCT+4, Bit 5. If that bit is a "1," DCT+6 contains only half of the true cylinder count.

DCT+7

Contains allocation information.

Bits 5-7 — Contain the number of heads for a hard drive.

Bits 0-4 — Contain the highest numbered sector relative to zero. A 10-sector-per-track drive would show X'09.' If DCT+4, Bit 5 indicates 2-sided operation, the sectors per cylinder equals twice this number.

DCT+8

Contains additional allocation information.

Bits 5-7 — Contain the number of granules per track allocated in the formatting process. If DCT+4, Bit 5 indicates 2-sided operation, the granules per cylinder equals twice this number. For a hard drive, this number is the total granules per cylinder.

Bits 0-4 — Contain the number of sectors per granule that was used in the formatting operation.

DCT+9

Contains the number of the cylinder where the directory is located. For any directory access, the system first attempts to use this value to read the directory. If this operation is unsuccessful, the system examines the BOOT granule (cylinder 0) directory address byte.

Bytes DCT + 6, DCT + 7, and DCT + 8 must relate without conflicts. That is, the highest numbered sector (+ 1) divided by the number of sectors per granule (+ 1) must equal the number of granules per track (+ 1).

Disk I/O Table

TRSDOS interfaces with hardware peripherals by means of software drivers. The drivers are, in general, coupled to the operating system through data parameters stored in the system's many tables. In this way, hardware not currently supported by TRSDOS can easily be supported by generating driver software and updating the system tables.

Disk drive sub-systems (such as controllers for 5¼" drives, 8" drives, and hard disk drives) have many parameters addressed in the Drive Code Table (DCT). Besides those operating parameters, controllers also require various commands (SELECT, SECTOR READ, SECTOR WRITE, and so on) to control the physical devices. TRSDOS has defined command conventions to deal with most commands available on standard Disk Controllers.

The function value (hexadecimal or decimal) you wish to pass to the driver should go in register B. The available functions are:

Hex	Dec	Function	Operation Performed
X'00'	0	DCSTAT	Test to see if drive is assigned in DCT
X'01'	1	SELECT	Select a new drive and return status
X'02'	2	DCINIT	Set to cylinder 0, restore, set side 0
X'03'	3	DCRES	Reset the Floppy Disk Controller
X'04'	4	RSTOR	Issue FDC RESTORE command
X'05'	5	STEPI	Issue FDC STEP IN command
X'06'	6	SEEK	Seek a cylinder
X'07'	7	TSTBSY	Test to see if requested drive is busy
X'08'	8	RDHDR	Read sector header information
X'09'	9	RDSEC	Read sector
X'0A'	10	VRSEC	Verify if the sector is readable
X'0B'	11	RDTRK	Issue an FDC track read command
X'0C'	12	HDFMT	Format the device
X'0D'	13	WRSEC	Write a sector
X'0E'	14	WRSYS	Write a system sector (for example, directory)
X'0F'	15	WRTRK	Issue an FDC track write command

Function codes X'10' to X'FF' are reserved for future use.

Directory Records (DIREC)

The directory contains information needed to access all files on the disk. The directory records section is limited to a maximum of 32 sectors because of physical limitations in the Hash Index Table. Two additional sectors in the directory cylinder are used by the system for the Granule Allocation Table and the Hash Index Table. The directory is contained on one cylinder. Thus, a 10-sector-per-cylinder formatted disk has, at most, eight directory sectors. See the sec-

tion on the Hash Index Table for the formula to calculate the number of directory sectors.

A directory record is 32 bytes in length. Each directory sector contains eight directory records (256/32 = 8). On system disks, the first two directory records of the first eight directory sectors are reserved for system overlays. The total number of files possible on a disk equals the number of directory sectors times eight (since 256/32 = 8). The number available for use is reduced by 16 on system disks to account for those record slots reserved for the operating system. The following table shows the directory record capacity (file capacity) of each format type. The dash suffix (-1 or -2) on the items in the density column represents the number of sides formatted (for example, SDEN-1 means single density, 1-sided).

	Sectors per Cylinder	Directory Sectors	User Files on Data Disk**	User Files on SYS Disk
5″ SDEN-1	10	8	62	48
5″ SDEN-2	20	18	142	128
5″ DDEN-1	18	16	126	112
5″ DDEN-2	36	32	254	240
8″ SDEN-1	16	14	110	96
8″ SDEN-2	32	30	238	224
8″ DDEN-1	30	28	222	208
8″ DDEN-2	60	32	254	240
Hard Disk*				

*Hard drive format depends on the drive size and type, as well as the user's division of the physical drive into logical drives. After setting up and formatting the drive, you can use the FREE library command to see the available files.

**__Note:__ Two directory records are reserved for BOOT/SYS and DIR/SYS, and are not included in the figures for this column.

TRSDOS Version 6 is upward compatible with other TRSDOS 2.3 compatible operating systems in its directory format. The data contained in the directory has been extended. An SVC is included to either display an abbreviated directory or place its data in a user-defined buffer area. For detailed information, see the @DODIR and @RAMDIR SVCs.

The following information describes the contents of each directory field:

DIR + 0

Contains all attributes of the designated file.

Bit 7 — If "0," this flag indicates that the directory record is the file's primary directory entry (FPDE). If "1," the directory record is one of the file's extended directory entries (FXDE). Since a directory entry can contain information on up to four extents (see notes on the extent fields, beginning with DIR + 22), a file that is fractured into more than four extents requires additional directory records.

Bit 6 — Specifies a SYStem file if "1," a nonsystem file if "0."

Bit 5 — If set to "1," indicates a Partition Data Set (PDS) file.

Bit 4 — Indicates whether the directory record is in use or not. If set to "1," the record is in use. If "0," the directory record is not active, although it may appear to contain directory information. In contrast to some operating systems that zero out the directory record when you remove a file, TRSDOS only resets this bit to zero.

Bit 3 — Specifies the visibility. If "1," the file is INVisible to a directory display or other library function where visibility is a parameter. If a "0," then the file is VISible. (The file can be referenced if specified by name by an @INIT or @OPEN SVC.)

Bits 0-2 — Contain the USER protection level of the file. The 3-bit binary value is one of the following:

0 = FULL	2 = RENAME	4 = UPDATE	6 = EXECUTE
1 = REMOVE	3 = WRITE	5 = READ	7 = NO ACCESS

DIR + 1

Contains various file flags and the month field of the packed date of last modification.

Bit 7 — Set to "1" if the file was "CREATEd" (see CREATE library command in the *Disk System Owner's Manual*). Since the CREATE command can reference a file that is currently existing but non-CREATEd, it can turn a non-CREATEd file into a CREATEd one. You can achieve the same effect by changing this bit to a "1."

Bit 6 — If set to "1," the file has not been backed up since its last modification. The BACKUP utility is the only TRSDOS facility that resets this flag. It is set during the close operation if the File Control Block (FCB + 0, Bit 2) shows a modification of file data.

Bit 5 — If set to "1," indicates a file in an open condition with UPDATE access or greater.

Bit 4 — If the file was modified during a session where the system date was not maintained, this bit is set to "1." This specifies that the packed date of modification (if any) stored in the next three fields is not the actual date the modification occurred. If this bit is "1," the directory command displays plus signs (+) between the date fields if you request the (A) option.

Bits 0-3 — Contain the binary month of the last modification date. If this field is a zero, DATE was not set when the file was established or since if it was updated.

DIR + 2

Contains the remaining date of modification fields.

Bits 3-7 — Contain the binary day of last modification.

Bits 0-2 — Contain the binary year minus 80. For example, 1980 is coded as 000, 1981 as 001, 1982 as 010, and so on.

DIR + 3

Contains the end-of-file offset byte. This byte and the ending record number (ERN) form a pointer to the byte position that follows the last byte written. This assumes that programmers, interfacing in machine language, properly maintain the next record number (NRN) offset pointer when the file is closed.

DIR + 4

Contains the logical record length (LRL) specified when the file was generated or when it was later changed with a CLONE parameter.

DIR + 5 through DIR + 12

Contain the name field of the filespec. The filename is left justified and padded with trailing blanks.

DIR + 13 through DIR + 15

Contain the extension field of the filespec. It is left justified and padded with trailing blanks.

DIR + 16 and DIR + 17

Contain the OWNER password hash code.

DIR + 18 and DIR + 19

Contain the USER password hash code. The protection level in DIR + 0 is associated with this password.

DIR + 20 and DIR + 21

Contain the ending record number (ERN), which is based on full sectors. If the ERN is zero, it indicates that no writing has taken place (or that the file was not closed properly). If the LRL is not 256, the ERN represents the sector where the EOF occurs. You should use ERN minus 1 to account for a value relative to sector 0 of the file.

DIR + 22 and DIR + 23

This is the first extent field. Its contents indicate which cylinder stores the first granule of the extent, which relative granule it is, and how many contiguous grans are in use in the extent.

DIR + 22 — Contains the cylinder value for the starting gran of that extent.

DIR + 23, Bits 5-7 — Contain the number of the granule in the cylinder indicated by DIR + 22 which is the first granule of the file for that extent. This value is relative to zero ("0" denotes the first gran, "1" denotes the second, and so on).

DIR + 23, Bits 0-4 — Contain the number of contiguous granules, relative to 0 ("0" denotes one gran, "1" denotes two, and so on). Since the field is five bits, it contains a maximum of X'1F' or 31, which represents 32 contiguous grans.

DIR + 24 and DIR + 25

Contain the fields for the second extent. The format is identical to that for Extent 1.

DIR + 26 and DIR + 27

Contain the fields for the third extent. The format is identical to that for Extent 1.

DIR + 28 and DIR + 29

Contain the fields for the fourth extent. The format is identical to that for Extent 1.

DIR + 30

This is a flag noting whether or not a link exists to an extended directory record. If no further directory records are linked, the byte contains X'FF.' A value of X'FE' in this byte establishes a link to an extended directory entry. (See "Extended Directory Records" below.)

DIR + 31

This is the link to the extended directory entry noted by the previous byte. The link code is the Directory Entry Code (DEC) of the extended directory record. The DEC is actually the position of the Hash Index Table byte mapped to the directory record. For more information, see the section "Hash Index Table."

Extended Directory Records

Extended directory records (FXDE) have the same format as primary directory records, except that only Bytes 0, 1, and 21-31 are utilized. Within Byte 0, only Bits 4 and 7 are significant. Byte 1 contains the DEC of the directory record of which this is an extension. An extended directory record may point to yet another directory record, so a file may contain an "unlimited" number of extents (limited only by the total number of directory records available).

Granule Allocation Table (GAT)

The Granule Allocation Table (GAT) contains information on the free and assigned space on the disk. The GAT also contains data about the formatting used on the disk.

A disk is divided into cylinders (tracks) and sectors. Each cylinder has a specified number of sectors. A group of sectors is allocated whenever additional space is needed. This group is called a granule. The number of sectors per granule depends on the total number of sectors available on a logical drive. The GAT provides for a maximum of eight granules per cylinder.

In the GAT bytes, each bit set to "1" indicates a corresponding granule in use (or locked out). Each bit reset to "Ø" indicates a granule free to be used. In a GAT byte, bit Ø corresponds to the first relative granule, bit 1 to the second relative granule, bit 2 the third, and so on. A 5¼" single density diskette is formatted at 1Ø sectors per cylinder, 5 sectors per granule, 2 granules per cylinder. Thus, that configuration uses only bits Ø and 1 of the GAT byte. The remainder of the GAT byte contains all 1's, denoting unavailable granules. Other formatting conventions are as follows:

	Sectors per Cylinder	Sectors per Granule	Granules per Cylinder	Maximum No. of Cylinders
5" SDEN	1Ø	5	2	8Ø
5" DDEN	18	6	3	8Ø
8" SDEN	16	8	2	77
8" DDEN	3Ø	1Ø	3	77
5-MEG HARD*	32	16	8	153

*Hard drive format depends on the drive size and type, as well as the user's division of the drive into logical drives. These values assume that one physical hard disk is treated as one logical drive.

The above table is valid for single-sided disks. TRSDOS supports double-sided operation if the hardware interfacing the physical drives to the CPU allows it. A two-headed drive functions as a single logical drive, with the second side as a cylinder-for-cylinder extension of the first side. A bit in the Drive Code Table (DCT + 4, Bit 5) indicates one-sided or two-sided drive configuration.

A Winchester-type hard disk can be divided by heads into multiple logical drives. Details are supplied with Radio Shack drives.

The Granule Allocation Table is the first relative sector of the directory cylinder. The following information describes the layout and contents of the GAT.

GAT + X'ØØ' through GAT + X'5F'

Contains the free/assigned table information. GAT + Ø corresponds to cylinder Ø, GAT + 1 corresponds to cylinder 1, GAT + 2 corresponds to cylinder 2, and so on. As noted above, bit Ø of each byte corresponds to the first granule on the cylinder, bit 1 to the second granule, and so on. A value of "1" indicates the granule is not available for use.

GAT + X'6Ø' through GAT + X'BF'

Contains the available/locked out table information. It corresponds cylinder for cylinder in the same way as the free/assigned table. It is used during mirror-image backups to determine if the destination diskette has the proper capacity to effect a backup of the source diskette. This table does not exist for hard disks; for this reason, mirror-image backups cannot be performed on hard disk.

GAT + X'CØ' through GAT + X'CA'

Used in hard drive configurations; extends the free/assigned table from X'ØØ' through X'CA'. Hard drive capacity up to 2Ø3 (Ø-2Ø2) logical or 4Ø6 physical cylinders is supported.

GAT + X'CB'

Contains the operating system version that was used in formatting the disk. For example, disks formatted under TRSDOS 6.1 have a value of X'61' contained in this byte. It is used to determine whether or not the disk contains all of the parameters needed for TRSDOS operation.

GAT + X'CC'

Contains the number of cylinders in excess of 35. It is used to minimize the time required to compute the highest numbered cylinder formatted on the disk. It is excess 35 to provide compatibility with alien systems not maintaining this byte. If you have a disk that was formatted on an alien system for other than 35 cylinders, this byte can be automatically configured by using the REPAIR utility. (See the section on the REPAIR utility in the *Disk System Owner's Manual*.)

GAT + X'CD'

Contains data about the formatting of the disk.

Bit 7 — If set to "1," the disk is a data disk. If "Ø," the disk is a system disk.

Bit 6 — If set to "1," indicates double-density formatting. If "Ø," indicates single-density formatting.

Bit 5 — If set to "1," indicates 2-sided disk. If "Ø," indicates 1-sided disk.

Bits 3-4 — Reserved.

Bits Ø-2 — Contain the number of granules per cylinder minus 1.

GAT + X'CE' and GAT + X'CF'

Contain the 16-bit hash code of the disk master password. The code is stored in standard low-order, high-order format.

GAT + X'DØ' through GAT + X'D7'

Contain the disk name. This is the name displayed during a FREE or DIR operation. The disk name is assigned during formatting or during an ATTRIB disk renaming operation. The name is left justified and padded with blanks.

GAT + X'D8' through GAT + X'DF'

Contain the date that the diskette was formatted or the date that it was used as the destination in a mirror image backup operation in the format mm/dd/yy.

GAT + X'EØ' through GAT + X'FF'

Reserved for system use.

In Version 6.2:

GAT + X'EØ' through GAT + X'F4'

Reserved for system use.

GAT + X'F5' through GAT + X'FF'

Contain the Media Data Block (MDB).

GAT + X'F5' through GAT + X'F8' — the identifying header. These four bytes contain a 3 (X'Ø3'), followed by the letters LSI (X'4C',X'53',X'49').

GAT + X'F8' through GAT + X'FF' — the last seven bytes of the DCT in use when the media was formatted. FORMAT, MemDISK, and TRSFORM4 install this information. See Drive Control Table (DCT) for more information on these bytes.

Hash Index Table (HIT)

The Hash Index Table is the key to addressing any file in the directory. It pinpoints the location of a file's directory with a minimum of disk accesses, keeping overhead low and providing rapid file access.

The system's procedure is to construct an 11-byte filename/extension field. The filename is left-justified and padded with blanks. The file extension is then inserted and padded with blanks; it occupies the three least significant bytes of

the 11-byte field. This field is processed through a hashing algorithm which produces a single byte value in the range X'01' through X'FF'. (A hash value of X'00' indicates a spare HIT position.)

The system then stores the hash code in the Hash Index Table (HIT) at a position corresponding to the directory record that contains the file's directory. Since more than one 11-byte string can hash to identical codes, the opportunity for "collisions" exists. For this reason, the search algorithm scans the HIT for a matching code entry, reads the directory record corresponding to the matching HIT position, and compares the filename/extension stored in the directory with that provided in the file specification. If both match, the directory has been found. If the two fields do not match, the HIT entry was a collision and the algorithm continues its search from the next HIT entry.

The position of the HIT entry in the hash table is called the Directory Entry Code (DEC) of the file. All files have at least one DEC. Files that are extended beyond four extents have a DEC for each extended directory entry and use more than one filename slot. To maximize the number of file slots available, you should keep your files below five extents where possible.

Each HIT entry is mapped to the directory sectors by the DEC's position in the HIT. Think of the HIT as eight rows of 32-byte fields. Each row is mapped to one of the directory records in a directory sector: The first HIT row is mapped to the first directory record, the second HIT row to the second directory record, and so on. Each column of the HIT field (0-31) is mapped to a directory sector. The first column is mapped to the first directory sector in the directory cylinder (not including the GAT and HIT). Therefore, the first column corresponds to sector 2, the second column to sector 3, and so on. The maximum number of HIT columns used depends on the disk formatting according to the formula: N = number of sectors per cylinder minus two, up to 32.

The following chart shows the correlation of the Hash Index Table to the directory records. Each byte value shown represents the position in the HIT. This position value is the DEC. The actual contents of each byte is either a X(00) indicating a spare slot, or the 1-byte hash code of the file that occupies the corresponding directory record.

	Columns															
Row 1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Row 2	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
Row 3	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
Row 4	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
Row 5	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
Row 6	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
Row 7	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
Row 8	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

A 5¼" single density disk has 10 sectors per cylinder, two of which are reserved for the GAT and HIT. Since only eight directory sectors are possible, only the first eight positions of each HIT row are used. Other formats use more columns of the HIT, depending on the number of sectors per cylinder in the formatting scheme.

The eight directory records for sector 2 of the directory cylinder correspond to assignments in HIT positions 00, 20, 40, 60, 80, A0, C0, and E0. On system

disks, the following positions are reserved for system overlays. On data disks, these positions (except for Ø0 and Ø1) are available to the user.

Ø0 — BOOT/SYS	20 — SYS6/SYS
Ø1 — DIR/SYS	21 — SYS7/SYS
Ø2 — SYSØ/SYS	22 — SYS8/SYS
Ø3 — SYS1/SYS	23 — SYS9/SYS
Ø4 — SYS2/SYS	24 — SYS1Ø/SYS
Ø5 — SYS3/SYS	25 — SYS11/SYS
Ø6 — SYS4/SYS	26 — SYS12/SYS
Ø7 — SYS5/SYS	27 — SYS13/SYS

These entry positions correspond to the first two rows of each directory sector for the first eight directory sectors. Since the operating system accesses these overlays by position in the HIT rather than by filename, these positions are reserved on system disks.

The design of the Hash Index Table limits the number of files on any one drive to a maximum of 256.

Locating a Directory Record

Because of the coding scheme used on the entries in the HIT table, you can locate a directory record with only a few instructions. The instructions are:

```
AND   1FH
ADD   A,2
```

(calculates the sector)

and

```
AND   ØEØH
```

(calculates the offset in that sector)

For example, if you have a Directory Entry Code (DEC) of X'84', the following occurs when these instructions are performed:

	Value of accumulator A = X'84'
AND 1FH	
	A = X'Ø4'
ADD A,2	
	A = X'Ø6' The record is in the seventh sector of the directory cylinder (Ø-6)

Using the Directory Entry Code (DEC) again, you can find the offset into the sector that was found using the above instructions by executing one instruction:

	Value of accumulator A = X'84'
AND ØEØH	
	A = X'8Ø' The directory record is X'8Ø' (128) bytes from the beginning of the sector

If the record containing the sector is loaded on a 256-byte boundary (LSB of the address is X'ØØ') and HL points to the starting address of the sector, then you can use the above value to calculate the actual address of the directory record by executing the instruction:

```
LD    L,A
```

When executed after the calculation of the offset, this causes HL to point to the record. For example:

```
A=X'80'
        LD    HL,4200H   ;Where sector is loaded
        LD    L,A        ;Replace LSB with offset
```

HL now contains 4280H, which is the address of the directory record you wanted.

If you cannot place the sector on a 256-byte boundary, then you can use the following instructions:

```
A=X'80'
        LD    HL,4256H   ;Where sector is loaded
        LD    E,A        ;Put offset in E (LSB)

        LD    D,0        ;Put a zero in D (MSB)
        ADD   HL,DE      ;Add two values together
```

HL now contains 42D6H, which is the address of the directory record.

Note that the first DEC found with a matching hash code may be the file's extended directory entry (FXDE). Therefore, if you are going to write system code to deal with this directory scheme, you must properly deal with the FPDE/FXDE entries. See Directory Records for more information.

Programming With Restart Vectors

The Restart instruction (RST) provides the assembly language programmer with the ability to call a subroutine with a one-byte call. If a routine is called many times by a program, the amount of space that is saved by using the RST instruction (instead of a three-byte CALL) can be significant.

In TRSDOS a RST instruction is also used to interface to the operating system. The system uses RST 28H for supervisor calls. RSTS 00H, 30H, and 38H are for the system's internal use.

RSTs 08H, 10H, 18H, and 20H are available for your use. Caution: Some programs, such as BASIC, may use some of these RSTs.

Each RST instruction calls the address given in the operand field of the instruction. For example, RST 18H causes the system to push the current program counter address onto the stack and then set the program counter to address 0018H. RST 20H causes a jump to location 0020H, and so on.

Each RST has three bytes reserved for the subroutine to use. If the subroutine will not fit in three bytes, then you should code a jump instruction (JP) to where the subroutine is located. At the end of the subroutine, code a return instruction (RET). Control is then transferred to the instruction that follows the RST.

For example, suppose you want to use RST 18H to call a subroutine named "ROUTINE." The following routine loads the restart vector with a jump instruction and saves the old contents of the restart vector for later use.

```
SETRST:  LD     IX,0018H   ;Restart area address
         LD     IY,RDATA   ;Data area address
         LD     B,3        ;Number of bytes to move
LOOP:    LD     A,(IX)     ;Read a byte from
                           ;restart area
         LD     C,(IY)     ;Read a byte from data
                           ;area
         LD     (IX),C     ;Store this byte in
                           ;restart area
         LD     (IY),A     ;Store this byte in data
                           ;area
         INC    IX         ;Increment restart area
                           ;pointer
         INC    IY         ;Increment data area
                           ;pointer
         DJNZ   LOOP       ;Loop till 3 bytes moved
         RET               ;Return when done
RDATA:   DEFB   0C3H       ;Jump instruction (JP)
         DEFW   ROUTINE     ;Operand (name of
                           ;subroutine)
```

Before exiting the program, calling the above routine again puts the original contents of the restart vector back in place.

KFLAG$ (BREAK), (PAUSE), and (ENTER) Interfacing

KFLAG$ contains three bits associated with the keyboard functions of BREAK, PAUSE ((SHIFT) (@)), and ENTER. A task processor interrupt routine (called the KFLAG$ scanner) examines the physical keyboard and sets the appropriate KFLAG$ bit if any of the conditions are observed. Similarly, the RS-232C driver routine also sets the KFLAG$ bits if it detects the matching conditions being received.

Many applications need to detect a PAUSE or BREAK while they are running. BASIC checks for these conditions after each logical statement is executed (that is, at the end of a line or at a ":"). That is how, in BASIC, you can stop a program with the (BREAK) key or pause a listing.

One method of detecting the condition in previous TRSDOS operating systems was to issue the @KBD supervisor call to check for BREAK or PAUSE ((SHIFT)(@)), ignoring all other keys. Unfortunately, this caused keyboard type-ahead to be ineffective; the @KBD SVC flushed out the type-ahead buffer if any other keystrokes were stacked up.

Another method was to scan the keyboard, physically examining the keyboard matrix. An undesirable side effect of this method was that type-ahead stored up the keyboard depression for some future unexpected input request. Examining the keyboard directly also inhibits remote terminals from passing the BREAK or PAUSE condition.

In TRSDOS Version 6, the KFLAG$ scanner examines the keyboard for the BREAK, PAUSE, and ENTER functions. If any of these conditions are detected, appropriate bits in the KFLAG$ are set (bits 0, 1, and 2 respectively).

Note that the KFLAG$ scanner only sets the bits. It does not reset them because the "events" would occur too fast for your program to detect. Think of the KFLAG$ bits as a latch. Once a condition is detected (latched), it remains latched until something examines the latch and resets it — a function to be performed by your KFLAG$ detection routine.

Under Version 6.2, you can use the @CKBRKC SVC, SVC 106, to see if the BREAK key has been pressed. If a BREAK condition exists, @CKBRKC resets the break bit of KFLAG$.

For illustration, the following example routine uses the BREAK and PAUSE conditions:

```
KFLAG$   EQU   10
@FLAGS   EQU   101
@KBD     EQU   8
@KEY     EQU   1
@PAUSE   EQU   16
CKPAWS   LD    A,@FLAGS          ;Get Flags pointer
         RST   28H               ;into register IY
         LD    A,(IY+KFLAG$)     ;Get the KFLAG$
         RRCA                    ;Bit 0 to carry
         JP    C,GOTBRK          ;Go on BREAK
         RRCA                    ;Bit 1 to carry
         RET   NC                ;Return if no pause
         CALL  RESKFL            ;Reset the flag
         PUSH  DE
FLUSH    LD    A,@KBD            ;Flush type-ahead
         RST   28H               ;buffer while
         JR    Z,FLUSH           ;ignoring errors
         POP   DE
PROMPT   PUSH  DE
         LD    A,@KEY            ;Wait on key entry
         RST   28H
         POP   DE
         CP    80H               ;Abort on (BREAK)
         JP    Z,GOTBRK
         CP    60H               ;Ignore PAUSE;
         JR    Z,PROMPT          ;else . . .
RESKFL   PUSH  HL                ;reset KFLAG$
         PUSH  AF
         LD    A,@FLAGS          ;Get flags pointer
         RST   28H               ;into register IY
RESKFL1  LD    A,(IY+KFLAG$)     ;Get the flag
         AND   0F8H              ;Strip ENTER,
```

```
        LD      (IY+KFLAG$),A ;PAUSE, BREAK
        PUSH    BC
        LD      B,16
        LD      A,@PAUSE      ;Pause a while
        RST     28H
        POP     BC
        LD      A,(IY+KFLAG$) ;Check if finger is
        AND     3             ;still on key
        JR      NZ,RESKFL1    ;Reset it again
        POP     AF            ;Restore registers
        POP     HL            ;and exit
        RET
```

The best way to explain this KFLAG$ detection routine is to take it apart and discuss each subroutine. The first piece reads the KFLAG$ contents:

```
KFLAG$  EQU     10
CKPAWS  LD      A,@FLAGS      ;Get Flags pointer
        RST     28H           ;into register IY
        LD      A,(IY+KFLAG$) ;Get the KFLAG$
        RRCA                  ;Bit 0 to carry
        JP      C,GOTBRK      ;Go on BREAK
        RRCA                  ;Bit 1 to carry
        RET     NC            ;Return if no Pause
```

The @FLAGS SVC obtains the flags pointer from TRSDOS. Note that if your application uses the IY index register, you should save and restore it within the CKPAWS routine. (Alternatively, you could use @FLAGS to calculate the location of KFLAG$, use register HL instead of IY, and place the address into the LD instructions of CKPAWS at the beginning of your application.)

The first rotate instruction places the BREAK bit into the carry flag. Thus, if a BREAK condition is in effect, the subroutine branches to "GOTBRK," which is your BREAK handling routine.

If there is no BREAK condition, the second rotate places what was originally in the PAUSE bit into the carry flag. If no PAUSE condition is in effect, the routine returns to the caller.

This sequence of code gives a higher priority to BREAK (that is, if both BREAK and PAUSE conditions are pending, the BREAK condition has precedence). Note that the GOTBRK routine needs to clear the KFLAG$ bits after it services the BREAK condition. This is easily done via a call to RESKFL.

The next part of the routine is executed on a PAUSE condition:

```
        CALL    RESKFL        ;Reset the flag
        PUSH    DE
FLUSH   LD      A,@KBD        ;Flush type-ahead
        RST     28H           ;buffer while
        JR      Z,FLUSH       ;ignoring errors
        POP     DE
```

First the KFLAG$ bits are reset via the call to RESKFL. Next, the routine takes care of the possibility that type-ahead is active. If it is, the PAUSE key was probably detected by the type-ahead routine and so is stacked in the type-ahead buffer also. To flush out (remove all stored characters from) the type-ahead buffer, @KBD is called until no characters remain (an NZ is returned).

Now that a PAUSEd state exists and the type-ahead buffer is cleared, the routine waits for a key input:

```
PROMPT  PUSH    DE
        LD      A,@KEY        ;Wait on key entry
        RST     28H
        POP     DE
        CP      80H           ;Abort on (BREAK)
        JP      Z,GOTBRK
```

```
        CP    60H             ;Ignore PAUSE;
        JR    Z,PROMPT        ;else . . .
```

The PROMPT routine accepts a BREAK and branches to your BREAK handling routine. It ignores repeated PAUSE (the 60H). Any other character causes it to fall through to the following routine which clears the KFLAG$:

```
RESKFL  PUSH  HL              ;reset KFLAG$
        PUSH  AF
        LD    A,@FLAGS        ;Get flags pointer
        RST   28H             ;into register IY
RESKFL1 LD    A,(IY+KFLAG$)   ;Get the flag
        AND   0F8H            ;Strip ENTER,
        LD    (IY+KFLAG$),A   ;PAUSE, BREAK
        PUSH  BC
        LD    B,16
        LD    A,@PAUSE        ;Pause a while
        RST   28H
        POP   BC
        LD    A,(IY+KFLAG$)   ;Check if finger is
        AND   3               ;still on key
        JR    NZ,RESKFL1      ;Reset it again
        POP   AF              ;Restore registers
        POP   HL              ;and exit
        RET
```

The RESKFL subroutine should be called when you first enter your application. This is necessary to clear the flag bits that were probably in a "set" condition. This "primes" the detection. The routine should also be called once a BREAK, PAUSE, or ENTER condition is detected and handled. (You need to deal with the flag bits for only the conditions you are using.)

Interfacing to @ICNFG

With the TRSDOS library command SYSGEN, many users may wish to SYSGEN the RS-232C driver. Before doing that, the RS-232C hardware (UART, Baud Rate Generator, etc.) must be initialized. Simply using the SYSGEN command with the RS-232C driver resident is not enough; some initialization routine is necessary. The @ICNFG (Initialization CoNFiGuration) vector is included in TRSDOS to provide a way to invoke a routine to initialize the RS-232C driver when the system is booted. It also provides a way to initialize the hard disk controller at power-up (required by the Radio Shack hard disk system).

The final stages of the booting process loads the configuration file CONFIG/SYS if it exists. After the configuration file is loaded, an initialization subroutine CALLs the @ICNFG vector. Thus, any initialization routine that is part of a memory configuration can be invoked by chaining into @ICNFG.

If you need to configure your own routine that requires initialization at power-up, you can chain into @ICNFG. The following procedure illustrates this link. The first thing to do is to move the contents of the @ICNFG vector into your initialization routine:

```
        LD    A,@FLAGS        ;Get flags pointer
        RST   28H             ;into register IY
        LD    A,(IY+28)       ;Get opcode
        LD    (LINK),A
        LD    L,(IY+29)       ;Get address LOW
        LD    H,(IY+30)       ;Get address HIGH
        LD    (LINK+1),HL
```

This subroutine does this by transferring the 3-byte vector to your routine. You then need to relocate your routine to its execution memory address. Once this

is done, transfer the relocated initialization entry point to the @ICNFG vector as a jump instruction:

```
        LD      HL,INIT         ;Get (relocated)
        LD      (IY+29),L       ;init address
        LD      (IY+30),H
        LD      A,0C3H          ;Set JP instruction
        LD      (IY+28),A
```

If you need to invoke the initialization routine at this point, then you can use:

```
        CALL    ROUTINE         ;Invoke your routine
```

Your initialization routine would be unique to the function it was to perform, but an overall design would look like this:

```
INIT    CALL    ROUTINE         ;Start of init
LINK    DEFS    3               ;Continue on
ROUTINE
        your initialization routine

        RET
```

After linking in your routine, perform the SYSGEN. If you have followed these procedures, your routine will be invoked every time you start up TRSDOS.

Interfacing to @KITSK

Background tasks can be invoked in one of two ways. For tasks that do not require disk I/O, you can use the RTC (Real Time Clock) interrupt and one of the 12 task slots (or other external interrupt). For tasks that require disk I/O, you can use the keyboard task process.

At the beginning of the TRSDOS keyboard driver is a call to @KITSK. This means that any time that @KBD is called, the @KITSK vector is also called. (The type-ahead task, however, bypasses this entry so that @KITSK is not called from the type-ahead routine.) Therefore, if you want to interface a background routine that does disk I/O, you must chain into @KITSK.

The interfacing procedure to @KITSK is identical to that shown in the section "Interfacing to @ICNFG," except that IY+31 through IY+33 is used to reference the @KITSK vector. You may want to start your background routine with:

```
START   CALL    ROUTINE         ;Invoke task
LINK    DEFS    3               ;For @KITSK hook
ROUTINE EQU     $               ;Start of the task
```

Be aware of one major pitfall. The @KBD routine is invoked from @CMNDI and @CMNDR (which is in SYS1/SYS). This invocation is from the @KEYIN call, which fetches the next command line after issuing the "TRSDOS Ready" message. If your background task executes and opens or closes a file (or does anything to cause the execution of a system overlay other than SYS1), then SYS1 is overwritten by SYS2 or SYS3. When your routine finishes, the @KEYIN handler tries to return to what called it—SYS1, which is no longer resident. Therefore, any task chained to @KITSK which causes a resident SYS1 to be overwritten must reload SYS1 before returning.

You can use the following code to reload SYS1 if SYS1 was resident prior to your task's execution:

```
ROUTINE LD      A,@FLAGS        ;Get flags pointer
        RST     28H             ;into register IY
        LD      A,(IY-1)        ;Get resident over-
        AND     8FH             ;lay and remove
        LD      (OLDSYS+1),A    ;the entry code
```

```
                rest of your task

EXIT      EQU   $
OLDSYS    LD    A,0             ;Get old overlay #
          CP    83H             ;Was it SYS1?
          RET   NZ              ;Return if not; else
          RST   28H             ;Get SYS1 per reg. A
                                ;(no RET needed)
```

Interfacing to the Task Processor

This section explains how to integrate interrupt tasks into your applications.

One of the hardware interrupts in the TRS-80 is the real time clock (RTC). The RTC is synchronized to the AC line frequency and pulses at 60 pulses per second, or once every 16.67 milliseconds. (Computers operating with 50 Hz AC use a 50 pulses per second RTC interrupt. In this case, all time relationships discussed in this section should be adjusted to the 50 Hz base.)

A software task processor manages the RTC interrupt in performing background tasks necessary to specific functions of TRSDOS (such as the time clock, blinking cursor, and so on). The task processor allows up to 12 individual tasks to be performed on a "time-sharing" basis.

These tasks are assigned to "task slots" numbered from 0 to 11. Slots 0-7 are considered "low priority" tasks (executing every 266.67 milliseconds). Slots 8-10 are medium priority tasks (executing every 33.33 milliseconds). Slot 11 is a high priority task (executing every 16.66 milliseconds SYSTEM (FAST) or 33.33 milliseconds SYSTEM (SLOW)). Task slots 3, 7, 9, and 10 are reserved by the system for the ALIVE, TRACE, SPOOL, and TYPE-AHEAD functions, respectively.

TRSDOS maintains a Task Control Block Vector Table (TCBVT) which contains 12 vectors, one for each of the 12 task slots. TRSDOS contains five supervisor calls that manage the task vectors. The five SVCs and their functions are:

@CKTSK	Checks to see whether a task slot is unused or active
@ADTSK	Adds a task to the TCBVT
@RMTSK	Removes a task from the TCBVT
@KLTSK	Removes the currently executing task
@RPTSK	Replaces the TCB address for the current task

The TRSDOS Task Control Block Vector Table contains vector pointers. Each TCBVT vector points to an address in memory, which in turn contains the address of the task. Thus, the tasks themselves are indirectly addressed.

When you are programming a task to be called by the task processor, the entry point of the routine needs to be stored in memory. If you make this storage location the beginning of a Task Control Block (TCB), the reason for indirect vectoring of interrupt tasks will become more clear. Consider an example TCB:

```
MYTCB     DEFW  MYTASK
COUNTER   DEFB  15
TEMPY     DEFS  1
MYTASK    RET
```

This is a useless task, since the only thing it does is return from the interrupt. However, note that a TCB location has been defined as "MYTCB" and that this location contains the address of the task. A few more data bytes immediately following the task address storage have also been defined.

Upon entry to a service routine, index register IX contains the address of the TCB. You can therefore address any TCB data using index instructions. For example, you could use the instruction "DEC (IX+2)" to decrement the value contained in COUNTER in the above routine.

8/Using the Supervisor Calls

Supervisor Calls (SVCs) are operating system routines that are available to assembly language programs. These routines alter certain system functions and conditions, provide file access, and perform various computations. They also perform I/O to the keyboard, video display, and printer.

Each SVC has a number which you specify to invoke it. These numbers range from 0 to 104.

In addition, under Version 6.2, you can write your own operating system routines using the numbers 124 through 127 to install your own SVC's. See Appendix E, "Programmable SVCs" for more information.

Calling Procedure

To call a TRSDOS SVC:

1. Load the SVC number for the desired SVC into register A. Also load any other registers which are needed by the SVC, as detailed under Supervisor Calls.
2. Execute a RST 28H instruction.

Note: If the SVC number supplied in register A is invalid, the system prints the message "System Error *xx*", where *xx* is usually 2B. It then returns you to TRSDOS Ready (*not* to the program that made the invalid SVC call).

The alternate register set (AF', BC', DE', HL') is not used by the operating system.

Program Entry and Return Conditions

When a program executed from the @CMNDI SVC is entered, the system return address is placed on the top of the stack. Register HL will point to the first non-blank character following the command name. Register BC will point to the first byte of the command line buffer.

Three methods of return from a program back to the system are available: the @ABORT SVC, the @EXIT SVC, and the RET instruction. For application programs and utilities, the normal return method is the @EXIT SVC. If no error condition is to be passed back, the HL register pair must contain a zero value. Any non-zero value in HL causes an active JCL to abort.

The @ABORT SVC can be used as an error return back to the system; it automatically aborts any active JCL processing. This is done by loading the value X'FFFF' into the HL register pair and internally executing an @EXIT SVC.

If stack integrity is maintained, a RET instruction can be used since the system return address is put on the stack by @CMNDI. This allows a return if the program was called with @CMNDR.

Most of the SVCs in TRSDOS Version 6 set the Z flag when the operation specified was successful. When an operation fails or encounters an error, the Z flag is reset (also known as NZ flag set) and a TRSDOS error code is placed in the A register. The remaining SVCs use the Z/NZ flag in differing ways, so you should refer to the description of the SVCs you are using to determine the exit conditions.

Supervisor Calls

The TRSDOS Supervisor Calls are:

Keyboard SVCs

@KBD
@KEY
@KEYIN

Byte I/O SVCs

@CTL
@GET
@PUT

Printer and Video SVCs

@DSP
@DSPLY
@LOGER
@LOGOT
@MSG
@PRT
@PRINT
@VDCTL

File Control SVCs

@CLOSE
@FEXT
@FNAME
@FSPEC
@INIT
@REMOV
@OPEN
@RENAM

Disk SVCs

@DCINIT
@DCRES
@DCSTAT
@RDSEC
@RDSSC
@RSLCT
@RSTOR
@SEEK
@SLCT
@STEPI
@VRSEC
@WRSEC
@WRSSC
@WRTRK

Disk File Handler SVCs

@BKSP
@CKEOF
@LOC
@LOF
@PEOF
@POSN
@READ
@REW
@RREAD
@RWRIT
@SEEKSC
@SKIP
@VER
@WEOF
@WRITE

System Control SVCs

@ABORT
@BREAK
@CMNDI
@CMNDR
@EXIT
@FLAGS
@HIGH$
@IPL
@LOAD
@RUN

TRSDOS Task Control SVCs

@ADTSK
@CKTSK
@KLTSK
@RMTSK
@RPTSK

Special Purpose Disk SVCs

@DIRRD
@DIRWR
@GTDCT
@HDFMT
@RDHDR
@RDTRK

Special Overlay SVCs

@CKDRV
@DEBUG
@DODIR
@ERROR
@PARAM
@RAMDIR

@CKBRKC SVC Number 106

Check BREAK bit and clear it Version 6.2 only

Checks to see if the BREAK key has been pressed. If a BREAK condition exists, @CKBRKC resets the break bit, Bit 0 of KFLAG$.

Entry Conditions:

A = 106(X'6A')

Exit Conditions:

Success always.
If Z flag is set, the break bit was not detected. If NZ flag is set, the break bit was detected and is cleared. If the BREAK key is being depressed, the SVC will not return until the key is released.

General:

Only AF is altered by this SVC.

Check Drive

Checks a drive reference to ensure that the drive is in the system and a TRSDOS Version 6 or LDOS 5.1.3 (Model III Hard Disk Operating System) formatted disk is in place.

Entry Conditions:

A=33 (X'21')
C=*logical drive number* (Ø-7)

Exit Conditions:

Success always.
If Z flag is set, the drive is ready.
If CF is set, the disk is write protected.
If NZ flag is set, the drive is not ready. The user may examine DCT+Ø to see if the drive is disabled.

Example:

See Sample Program D, lines 35-55.

Check for End-Of-File

Checks for the end of file at the current logical record number.

Entry Conditions:

A = 62 (X'3E')
DE = *pointer to the FCB of the file to check*

Exit Conditions:

Success always.
If Z flag is set, LOC does not point at the end of file (LOC < LOF).
If NZ flag is set, test A for error number:
If A = X'1C', LOC points at the end of the file (LOC = LOF).
If A = X'1D', LOC points beyond the end of the file (LOC > LOF).
If A ≠ X'1C' or X'1D', then A = *error number.*

General:

Only AF is altered by this SVC.

Example:

See Sample Program C, lines 352-353.

Check if Task Slot in Use

Checks to see if the specified task slot is in use.

Entry Conditions:

A = 28 (X'1C')
C = *task slot to check* (0-11)

Exit Conditions:

Success always.
If Z flag is set, the task slot is available for use.
If NZ flag is set, the task slot is already in use.

General:

AF and HL are altered by this SVC.

Example:

See Sample Program F, lines 70-73.

Close a File or Device

Terminates output to a file or device. Any unsaved data in the buffer area is saved to disk and the directory is updated. All files that have been written to must be closed, as well as all files opened with UPDATE or higher access.

If you remove a diskette containing an open file, any attempt to close the file results in the message:

** CLOSE FAULT ** *error message,* <ENTER> to retry, <BREAK> to abort

where *error message* is usually "Drive not ready". You may put the diskette back in the drive and:

1. Press (ENTER) to close the file.
2. Press (BREAK) to abort the close.

If you press (BREAK), the NZ flag is set and Register A contains X'20', the error code for an Illegal drive number error.

Entry Conditions:

A = 60 (X'3C')
DE = *pointer to FCB or DCB to close*

Exit Conditions:

Success, Z flag set. The file or device was closed. The filespec (excluding the password) or the devspec is returned to the FCB or DCB.
Failure, NZ flag set.
A = *error number*

General:

Only AF is altered by this SVC.

Example:

See Sample Program C, lines 360-368.

@CLS — SVC Number 105

Clear Video Screen — Version 6.2 only

Clears the video screen by sending a Home Cursor (X'1C') and Clear to End of Frame (X'1F') sequence to the video driver.

Entry Conditions:

A = 105(X'69')

Exit Conditions:

Success, Z flag is set.
Failure, NZ is set.
A = *error number*

General:

Only AF is altered by this SVC.

Output a Control Byte

Outputs a control byte to a logical device. The DCB TYPE byte (DCB + Ø, Bit 2) must permit CTL operation. See the section "@CTL Interfacing to Device Drivers" for information on which of the functions listed below are supported by the system device drivers.

Entry Conditions:

A = 5 (X'Ø5')
DE = *pointer to DCB to control output*
C selects one of the following functions:
If C = Ø, the status of the specified device will be returned.
If C = 1, the driver is requested to send a BREAK or force an interrupt.
If C = 2, the initialization code of the driver is to be executed.
If C = 3, all buffers in the driver are to be reset. This causes all pending I/O to be cleared.
If C = 4, the wakeup vector for an interrupt-driven driver is specified by the caller.
IY = address to vector when leaving driver. If IY = Ø, then the wakeup vector function is disabled. The RS-232C driver COM/DVR ($CL), is the only system driver that provides wakeup vectoring.
If C = 8, the next character to be read will be returned. This allows data to be "previewed" before the actual @GET returns the character.

Exit Conditions:

If C = Ø,
Z flag set, device is ready
NZ flag set, device is busy
A = status image, if applicable
Note: This is a hardware dependent image.
If C = 1,
Success, Z flag set. BREAK or interrupt generated.
Failure, NZ flag set
A = *error number*
If C = 2,
Success, Z flag set. Driver initialized.
Failure, NZ flag set
A = *error number*
If C = 3,
Success, Z flag set. Buffers cleared.
Failure, NZ flag set.
A = *error number*
If C = 4,
Success always.
IY = *previous vector address*
This function is ignored if the driver does not support wakeup vectoring.
If C = 8,
Success, Z flag set. Next character returned.
A = *next character in buffer*
Failure, NZ flag set. Test register A:
If A = Ø, no pending character is in buffer
If A ≠ Ø, A contains *error number*. (TRSDOS driver returns Error 43.)

General:

BC, DE, HL, and IX are saved.

Function codes 5 to 7, 9 to 31, and 255 are reserved for the system. Function codes 32 to 254 are available for user definition.

Entry and exit conditions for user-defined functions are up to the design of the user-supplied driver.

Example:

See the section "Device Driver and Filter Templates."

Entry to Post an Error Message

Provides an entry to post an error message. If bit 7 of register C is set, the error message is displayed and return is made to the calling program. If bit 6 is not set, the extended error message is displayed. Under versions prior to 6.2 the error display is in the following format:

```
*** Errcod=xx, Error message string ***
      <filespec or devspec>
Referenced at X'dddd'
```

Under Version 6.2 the error display is in the following format:

```
** Error code = xx, Returns to X"dddd'
** Error message string
<filespec, devspec, or open FCB/DCB status>
Last SVC = nnn, Returned to X"rrrr'
```

dddd is the return address of the @ERROR SVC in the application program.
nnn is the last SVC executed before the @ERROR SVC request.
rrrr is the address the previous SVC returned to in the application program.

If bit 6 is set, then only the "Error message string" is displayed. This bit is ignored if bit 6 of SFLAG$ (the extended error message bit) is set. If bit 6 of CFLAG$ is set, then no error message is displayed. If bit 7 of CFLAG$ is set, then the "Error message string" is placed in a user buffer pointed to by register pair DE. See @FLAGS (SVC 101) for more information on SFLAG$ and CFLAG$.

Entry Conditions:

A = 26 (X'1A')
C = *error number with bits 6 and 7 optionally set*

Exit Conditions:

Success always.

General:

To avoid a looping condition that could result from the display device generating an error, do not check for errors after returning from @ERROR.
If you do not set bit 6 of register C, then you should execute this SVC only after an error has actually occurred.

Example:

See Sample Program C, lines 379-389.

@EXIT — SVC Number 22

Exit to TRSDOS

This is the normal program exit and return to TRSDOS. An error exit can be done by placing a non-zero value in HL. Values 1 to 62 indicate a primary error as described in TRSDOS Error Codes (Appendix A). (A non-zero value in HL causes an active JCL to abort.)

Entry Conditions:

A = 22 (X'16')
HL = *Return Code*
If HL = 0, then no error on exit.
If HL ≠ 0, then the @ABORT SVC returns X'FFFF' in HL automatically.

General:

This SVC does not return.

Example:

See Sample Program B, lines 206-207.

Set Up Default File Extension

Inserts a default file extension into the File Control Block if the file specification entered contains no extension. @FEXT must be done before the file is opened.

Entry Conditions:

A =79 (X'4F')
DE=*pointer to FCB*
HL=*pointer to default extension* (3 characters; alphabetic characters must be upper case and first character must be a letter)

Exit Conditions:

Success always.
AF and BC are altered by this SVC.
If the default extension is used, HL is also altered.

Example:

See Sample Program C, lines 111-132.

Point IY to System Flag Table

Points the IY register to the base of the system flag table. The status flags listed below can be referenced off IY. You can alter those bits marked with an asterisk (*). Bits without an asterisk are indicators of current conditions, or are unused or reserved.

Note: You may wish to save KFLAG$ and SFLAG$ if you intend to modify them in your program, and restore them on exit.

Entry Conditions:

A = 101 (X'65')

Exit Conditions:

Success always.

IY = *pointer to the following system information:*

IY − 1 Contains the overlay request number of the last system module resident in the system overlay region.

IY + 0 = AFLAG$ (allocation flag under Version 6.2 only)
Contains the starting cylinder number to be used when searching for free space on a diskette. It is normally 1. If the starting cylinder number is larger than the number of cylinders for a particular drive, 1 is used for that drive.

IY + 2 = CFLAG$

* bit 7 — If set, then @ERROR will transfer the "Error message string" to your buffer instead of displaying it. The message is terminated with X'0D.'
* bit 6 — If set, do not display system error messages 0-62. See @ERROR (SVC 26) for more information.
* bit 5 — If set, sysgen is not allowed.
* bit 4 — If set, then @CMNDR will execute only system library commands.
bit 3 — If set, @RUN is requested from either the SET or SYSTEM (DRIVER=) commands.
bit 2 — If set, @KEYIN is executing due to a request from SYS1.
bit 1 — If set, @CMNDR is executing. This bit is reset by @EXIT and @CMNDI.
* bit 0 — If set, HIGH$ cannot be changed using @HIGH$ (SVC 100). This bit is reset by @EXIT and @CMNDI.

IY + 3 = DFLAG$ (device flag)

* bit 7 — "1" if GRAPHIC printer capability desired on screen print (CONTROL :) causes screen print. See the SYSTEM (GRAPHIC) command under "Technical Information on TRSDOS Commands and Utilities.")
bit 6 — "1" if KSM module is resident
bit 5 — Currently unused
bit 4 — "1" if MemDisk active
bit 3 — Reserved
bit 2 — "1" if Disk Verify is enabled
* bit 1 — "1" if TYPE-AHEAD is active
bit 0 — "1" if SPOOL is active

IY + 4 = EFLAG$ (ECI flag under Version 6.2 only)
Indicates the presence of an ECI program. If any of the bits are set, an ECI is used, rather than the SYS1 interpreter. The ECI program may use these bits as neccesary. However, at least one bit must be set or the ECI is not executed.

IY+5 = FEMSK$ (mask for port 0FEH)
IY+8 = IFLAG$ (international flag)
* bit 7 — If "1," 7-bit printer filter is active
If "0," normal 8-bit filters are present
* bit 6 — If "1," international character translation will be performed by printer driver
If "0," characters received by printer driver will be sent to the printer unchanged
bit 5 — Reserved for future languages
bit 4 — Reserved for future languages
bit 3 — Reserved for future languages
bit 2 — Reserved for future languages
bit 1 — If "1," German version of TRSDOS is present
bit 0 — If "1," French version of TRSDOS is present
If bits 5-0 are all zero, then USA version of TRSDOS is present.
IY+10 = KFLAG$ (keyboard flag)
bit 7 — "1" if a character is present in the type-ahead buffer
bit 6 — Currently unused
* bit 5 — "1" if CAPS lock is set
bit 4 — Currently unused
bit 3 — Currently unused
* bit 2 — "1" if (ENTER) has been pressed
* bit 1 — "1" if (SHIFT) @ has been pressed (PAUSE)
* bit 0 — "1" if (BREAK) has been pressed
Note: To use bits 0-2, you must first reset them and then test to see if they become set.
IY+12 = MODOUT (image of port 0ECH)
IY+13 = NFLAG$ (network flag under Version 6.2)
bit 7 — Reserved for system use.
bit 6 — If set, the application program is in the task processor. Programmers must **not** modify this bit.
bit 5 — Reserved for system use.
bit 4 — Reserved for system use.
bit 3 — Reserved for system use.
bit 2 — Reserved for system use.
bit 1 — Reserved for system use.
* bit 0 — If set, the "file open bit" is written to the directory.
IY+14 = OPREG$ (memory management & video control image)
IY+17 = RFLAG$ (retry flag under Version 6.2 only)
Indicates the number of retrys for the floppy disk driver. This should be an even number larger than two.
IY+18 = SFLAG$ (system flag)
bit 7 — "1" if DEBUG is to be turned on
* bit 6 — "1" if extended error messages desired (see @ERROR for message format); overrides the setting of bit 6 of register C on @ERROR (SVC 26) and should be used only when testing
bit 5 — "1" if DO commands are being executed
* bit 4 — "1" if BREAK disabled
bit 3 — "1" if the hardware is running at 4 mhz (SYSTEM (FAST)). If "0," the hardware is running at 2 mhz (SYSTEM (SLOW)).
* bit 2 — "1" if LOAD called from RUN
* bit 1 — "1" if running an EXECute only file
* bit 0 — "1" specifies no check for matching LRL on file open and do not set file open bit in directory. This bit should be set just before executing an @OPEN (SVC 59) if you want to force the opened file to be READ only during current I/O operations. As soon as either call is executed, SFLAG$ bit 0 is reset. If you want to disable LRL checking on another file, you must set SFLAG$ bit 0 again.

IY + 19 = TFLAG$ (type flag under Version 6.2 only)
Identifies the Radio Shack hardware model. TFLAG$ allows programs to be aware of the hardware environment and the character sets available for the display. Current assignments are:

2 indicates Model II
4 indicates Model 4
5 indicates Model 4P
12 indicates Model 12

IY + 20 = UFLAG$ (user flag under Version 6.2 only)
May be set by application programs and is sysgened properly.

IY + 21 = VFLAG$
bit 7 — Reserved for system use
* bit 6 — "1" selects solid cursor, "0" selects blinking cursor
bit 5 — Reserved for system use
* bit 4 — "1" if real time clock is displayed on the screen
bits 0-3 — Reserved for system use

IY + 22 = WRINTMASK$ (mask for WRINTMASK port)
IY + 26 = SVCTABPTR$ (pointer to the high order byte of the SVC table address; low order byte = 00)
IY + 27 = Version ID byte (60H = TRSDOS version 6.0.x.x, 61H = TRSDOS version 6.1.x.x, etc.)
IY − 47 = Operating system release number. Provides a third and fourth character (12H = TRSDOS version x.x.1.2)
IY + 28
to
IY + 30 = @ICNFG vector
IY + 31
to
IY + 33 = @KITSK vector

@FNAME

SVC Number 80

Get Filename

Gets the filename and extension from the directory using the specified Directory Entry Code (DEC) for the file.

Entry Conditions:

A = 80 (X'50')
DE = *pointer to 15-byte buffer to receive filename/extension:drive, followed by a X'0D' as a terminator*
B = *DEC of desired file*
C = *logical drive number of drive containing file* (0-7)

Exit Conditions:

Success, Z flag set.
HL = *pointer to directory entry specified by register B*
Failure, NZ flag set.
A = *error number*
HL is altered.

General:

AF and BC are always altered.
If the drive does not contain a disk, this SVC may hang indefinitely waiting for formatted media to be placed in the drive. The programmer should perform a @CKDRV SVC before executing this call.
If the Directory Entry Code is invalid, the SVC may not return or it may return with the Z flag set and HL pointing to a random address. Care should be taken to avoid using the wrong value for the DEC in this call.

Example:

See Sample Program C, lines 274-286.

Video Functions

Performs various functions related to the video display. The B register is used to pass the function number.

Entry Conditions:

A = 15 (X'0F')

B selects one of the following functions:

If B = 1, return the character at the screen position specified by HL.
H = *row on the screen* (0-23), where 0 is the top row
L = *column on the screen* (0-79), where 0 is the leftmost column

If B = 2, display the specified character at the position specified by HL.
C = *character to be displayed*
H = *row on the screen* (0-23), where 0 is the top row
L = *column on the screen* (0-79), where 0 is the leftmost column

If B = 3, move the cursor to the position specified by HL. This is done even if the cursor is not currently displayed.
H = *row on the screen* (0-23), where 0 is the top row
L = *column on the screen* (0-79), where 0 is the leftmost column

If B = 4, return the current position of the cursor.

If B = 5, move a 1920-byte block of data to video memory.
HL = *pointer to 1920-byte buffer to move to video memory*

If B = 6, move a 1920-byte block of data from video memory to a buffer you supply. In 40 line by 24 character mode, there must be a character in each alternating byte for proper display.
HL = *pointer to 1920-byte buffer to store copy of video memory* HL must be in the range X'23FF' < HL < X'EC01.'

If B = 7, scroll protect the specified number of lines from the top of the screen.
C = *number of lines to scroll protect* (0-7). Once set, scroll protect can be removed only by executing @VDCTL with B = 7 and C = 0, or by resetting the system. Clearing the screen with (SHIFT)(CLEAR) erases the data in the scroll protect area, but the scroll protect still exists.

If B = 8, change cursor character to specified character. If the cursor is currently not displayed, the character is accepted anyway and is used as the cursor character when it is turned back on. The default cursor character is an underscore (X'5F') under Version 6.2 and a X'B0' under previous versions.
C = *character to use as the cursor character*

If B = 9, (under Version 6.2 only) transfer 80 characters to or from the screen.
If C = 0, move characters from the buffer to the screen
If C = 1, move characters from the screen to the buffer
H = *row on the screen*
DE = *pointer to 80 byte buffer*

Note: The video RAM area in the Models 4 and 4P is 2048 bytes (2K). The first 1920 bytes can be displayed. The remaining bytes contain the type-ahead buffer and other system buffers.

Exit Conditions:

If B = 1:
Success, Z flag set.
A = *character found at the location specified by HL*
DE is altered.
Failure, NZ flag set.
A = *error number*

If B = 2:
Success, Z flag set.
DE is altered.
Failure, NZ flag set.
A = *error number*

If B = 3:
Success, Z flag set.
DE and HL are altered.
Failure, NZ flag set.
A = *error number*

If B = 4:
Success always.
HL = *row and column position of the cursor.* H = row on the screen (0-23), where 0 is the top row; L = column on the screen (0-79), where 0 is the leftmost column.

If B = 5:
Success always.
HL = *pointer to the last byte moved to the video + 1*
BC and DE are altered.

If B = 6:
Success always.
BC, DE, and HL are altered.

If B = 7:
Success always.
BC and DE are altered.

If B = 8:
Success always.
A = *previous cursor character*
DE is altered.

If B = 9 (under Version 6.2 only):
Success, Z flag set.
BC, HL, DE are altered.
Failure, NZ flag set because H is out of range.
A = error code 43 (X'2B').

General:

Functions 5, 6, and 7 do not do range checking on the entry parameters.
If HL is not in the valid range in functions 5 and 6, the results may be unpredictable.
Only function 3 (B = 3) moves the cursor.
If C is greater than 7 in function 7, it is treated as modulo 8.
AF and B are altered by this SVC.

Example:

See Sample Program F, lines 304-327.

Sample Program B, continued

```
ØØ272   ;These are the storage declarations.
ØØ273
ØØ274   BUF6:    DEFS    6
ØØ275   BUF5     DEFS    5
ØØ276   BUF4:    DEFS    4
ØØ277   BUF3:    DEFS    3
ØØ278   BUF2:    DEFS    2
ØØ279   DIVR1:   DEFB    Ø
ØØ28Ø   DIVD1:   DEFB    Ø
ØØ281   ANS1:    DEFB    Ø
ØØ282   REM1:    DEFB    Ø
ØØ283   MCAND1:  DEFB    Ø
ØØ284   MIER1:   DEFB    Ø
ØØ285   MCAND2:  DEFW    Ø
ØØ286   DIVD2:   DEFW    Ø
ØØ287   ANS2:    DEFW    Ø
ØØ288
ØØ289   ;Below are messages and prompting text used in the program.
ØØ29Ø
ØØ291           DEFB    13              ;Number of blanks to print after message 1
ØØ292   MESS1:   DEFM    'Enter a number (1-255).'
ØØ293           DEFB    3               ;Message-terminating character
ØØ294           DEFB    21              ;Number of blanks to print after message 3
ØØ295   MESS3:   DEFM    'The answer is'
ØØ296           DEFB    3               ;Terminating character
ØØ297           DEFB    18              ;Blanks after message
ØØ298   MESS4:   DEFM    'The remainder is'
ØØ299           DEFB    3               ;Terminating character
ØØ3ØØ           DEFB    6               ;Blanks after message
ØØ3Ø1   MESS6:   DEFM    'Enter a number (4369-65535).'
ØØ3Ø2           DEFB    3               ;Terminating character
ØØ3Ø3           DEFB    15              ;Blanks after message
ØØ3Ø4   MESS8:   DEFM    'Enter a number (1-28).'
ØØ3Ø5           DEFB    3               ;Terminating character
ØØ3Ø6           DEFB    16              ;Blanks after message
ØØ3Ø7   MESS9:   DEFM    'In hex ASCII, that is'
ØØ3Ø8           DEFB    3               ;Terminating character
ØØ3Ø9           DEFB    17              ;Blanks after message
ØØ31Ø   MESS1Ø:  DEFM    'Enter a number (1-9).'
ØØ311           DEFB    3               ;Terminating character
ØØ312           DEFB    11              ;Blanks after message
ØØ313   MESS11:  DEFM    'Enter a number (1-41ØØ).'
ØØ314           DEFB    3               ;Terminating character
ØØ315           DEFB    15              ;Blanks after message
ØØ316   MESS12:  DEFM    'Enter a number (1-15).'
ØØ317           DEFB    3               ;Terminating character
ØØ318   MESS13:  DEFM    'The product of those 2 numbers is '
ØØ319           DEFB    3               ;Terminating character
ØØ32Ø   MESS14:  DEFM    'Press <BREAK> to end or any other key to continue.'
ØØ321           DEFB    ØDH             ;Terminating character
ØØ322
ØØ323           END     START
```

Sample Program C

```
Ln #            Source Line

ØØØØ1   ;       This program prompts for two filenames, opens the first
ØØØØ2   ;       file, and creates the second.  Then the data in the first
ØØØØ3   ;       file is copied to the second file.  While the Copy progresses,
ØØØØ4   ;       the current record number is displayed in parentheses.
ØØØØ5
ØØØØ6           PSECT   3ØØØH           ;This program starts at x'3ØØØ'
ØØØØ8
ØØØØ9   ;       First, declare the equates for the SVCs we intend to use.
ØØØ1Ø   ;       This is not mandatory, but it makes the program easier to follow.
ØØØ11
ØØØ12   @CLOSE: EQU     6Ø             ;Close a file or device
ØØØ13   @DIRRD: EQU     87             ;Read a directory record
ØØØ14   @DSP:   EQU     2              ;Display character at cursor
ØØØ15   @DSPLY: EQU     1Ø             ;Display a message
ØØØ16   @ERROR: EQU     26             ;Display an error message
ØØØ17   @EXIT:  EQU     22             ;Exit and return to TRSDOS or the caller
ØØØ18   @FEXT:  EQU     79             ;Add a default file extension
ØØØ19   @FNAME: EQU     8Ø             ;Fetch a filespec from the directory
ØØØ2Ø   @FSPEC: EQU     78             ;Verify and load a filespec into the FCB
ØØØ21   @HEXDEC:EQU     97             ;Convert a binary value to decimal ASCII
ØØØ22   @INIT:  EQU     58             ;Open an existing file or create a new file
ØØØ23   @KBD:   EQU     8              ;Scan the keyboard for a character
ØØØ24   @KEYIN: EQU     9              ;Accept a line of text from the *KI device
ØØØ25   @LOC:   EQU     63             ;Return the current logical record number
ØØØ26   @OPEN:  EQU     59             ;Open an existing file
ØØØ27   @READ:  EQU     67             ;Read a record from an open file
ØØØ28   @REMOV: EQU     57             ;Delete a file from disk
ØØØ29   @VER:   EQU     73             ;Write a record to disk.  Does the same thing
ØØØ3Ø                                  ;as @WRITE (Svc 75), but it also makes sure
ØØØ31                                  ;the written data is readable.
ØØØ32
ØØØ33   ;       First, prompt for the source filespec using the @DSPLY svc.
ØØØ34
ØØØ35   BEGIN:  LD      HL,MESG1       ;Get the first message
ØØØ36           LD      A,@DSPLY       ;Display a line on the screen
ØØØ37           RST     28H            ;Call the @DSPLY svc
ØØØ38
ØØØ39   ;       Now, read the filename from the keyboard using the @KEYIN svc.
ØØØ4Ø
ØØØ41           LD      HL,FILE1       ;Put the name of the 1st file here
ØØØ42           LD      B,24           ;Allow up to 24 characters
ØØØ43           LD      C,Ø            ;A zero is required by the svc
ØØØ44           LD      A,@KEYIN       ;Get a filename from the user
ØØØ45           RST     28H            ;Call the @KEYIN svc
ØØØ46           JP      C,QUIT         ;The user pressed <Break>
ØØØ47           JP      NZ,ERR         ;An Error occurred
ØØØ48
ØØØ49           LD      A,B            ;Get the number of characters
ØØØ5Ø           OR      A              ;See if that value was zero
ØØØ51           JR      Z,BEGIN        ;Nothing was entered, ask again
ØØØ52
ØØØ53   ;       The user has typed something, so it must be checked for validity
ØØØ54   ;       using the @FSPEC svc.
ØØØ55
ØØØ56           LD      HL,FILE1       ;Point at the text the user entered
ØØØ57           LD      DE,FCB1        ;Point at the File Control Block
ØØØ58                                  ;that is to be used for the source file.
ØØØ59           LD      A,@FSPEC       ;The @FSPEC svc will make sure the filename
ØØØ6Ø                                  ;that is in buffer named "file1" is valid.
ØØØ61                                  ;If it is, it is copied into the File
ØØØ62                                  ;Control Block (FCB) to be used by the @OPEN
ØØØ63                                  ;or @INIT svc later on.
ØØØ64           RST     28H            ;Call the @FSPEC svc
ØØØ65           JR      Z,ASK2         ;The name for file 1 is ok, so skip this
ØØØ66
ØØØ67   ;       At this point the filename specified for file 1 has been found
```

Sample Program C, continued

```
00339           LD      HL,BUFFER       ;Point at the data read from file 1
00340           LD      A,@VER          ;Write a record to the target file
00341                                   ;The @VER does the same thing as the
00342                                   ;@WRITE svc, only it also checks the
00343                                   ;data to make sure it is readable.
00344           RST     28H             ;Call the @VER svc
00345           JR      NZ,ERR          ;An error occurred on write; possibly
00346                                   ;the disk is full.
00347           JR      LOOP            ;Loop until an error occurs.
00348
00349   ;       This code checks the error to make sure it was an end of file
00350   ;       condition and, if so, closes the source & target files.
00351
00352   EOF:    CP      28              ;Was it an end of file encountered?
00353           JR      Z,EOFYES        ;Yes, close the file
00354           CP      29              ;Was it "Record number out of range"?
00355           JR      NZ,ERR          ;No, must be some other error
00356
00357   ;       It is possible to get Error 29 if the file being copied has
00358   ;       an EOF that is not a multiple of the file's LRL
00359
00360   EOFYES: LD      DE,FCB1         ;Point at file 1 (source file)
00361           LD      A,@CLOSE        ;Close the file
00362           RST     28H             ;Call the @CLOSE svc
00363           JR      NZ,ERR          ;An error occurred, abort
00364
00365           LD      DE,FCB2         ;Point at file 2 (target file)
00366           LD      A,@CLOSE        ;Close it also
00367           RST     28H             ;Call the @CLOSE svc
00368           JR      NZ,ERR          ;An error occurred, abort
00369
00370           LD      HL,OK           ;Print a message saying the copy is done
00371           LD      A,@DSPLY
00372           RST     28H             ;Call the @DSPLY svc
00373
00374   QUIT:   LD      A,@EXIT         ;Exit to TRSDOS or the calling program
00375           RST     28H             ;Call the @EXIT svc
00376
00377   ;       The @EXIT svc does not return.
00378
00379   ERR:    OR      040H            ;Turn on bit 6, which
00380                                   ;will cause the @ERROR svc to print
00381                                   ;the short error message. Bit 7
00382                                   ;is not set, which instructs the @ERROR
00383                                   ;to abort this program and return to
00384                                   ;TRSDOS Ready.
00385           LD      C,A             ;Put error code & flags in register C
00386           LD      A,@ERROR        ;Call the system error displayer
00387           RST     28H             ;Call the @ERROR svc
00388
00389   ;       Because bit 7 is not set, the @ERROR svc will not return.
00390
00391   ;       Storage Declaration
00392
00393   SPACES: DEFM    '  '            ;ASCII Space char.for display formatting
00394           DEFB    3
00395   ARROW:  DEFM    '=>  '          ;Arrow for display shows data direction
00396           DEFB    3
00397   OK:     DEFB    10%25           ;Advance cursor 10 spaces without erasing
00398           DEFM    '[Ok]'          ;Used to indicate the Copy is complete
00399           DEFB    0DH             ;Terminated with an <Enter>
00400   MESG1:  DEFM    'Copy Filespec >'
00401           DEFB    3
00402   MESG2:  DEFM    'To Filespec >'
00403           DEFB    3
00404   FEXST:  DEFM    'Destination File Already Exists - Ok to Delete it (Y/N) ?'
00405           DEFB    3
```

Sample Program C, continued

```
00406   BADFIL: DEFM    'Invalid Filename - Try Again'
00407           DEFB    0DH
00408   LOCMSG: DEFM    ' 12345)'       ;This will be used in building the LOC
00409                                   ;Display will appear as (d) to (ddddd).
00410           DEFB    7%24            ;Backspace without erasing
00411           DEFB    3               ;Etx, used to get the @DSPLY svc to stop
00412
00413   FILE1:  DEFS    32              ;User Text Originally placed here
00414   FILE2:  DEFS    32              ;Target Filename goes here
00415   FCB1:   DEFS    32              ;32 bytes for the File Control Block
00416   FCB2:   DEFS    32              ;32 bytes for the File Control Block
00417   COPY:   DEFS    32              ;An extra copy of the target FCB goes here
00418   LRL:    DEFB    0               ;The Logical Record Length of the source
00419                                   ;file will be stored here
00420   BUF1:   DEFS    256             ;System buffer for File 1
00421   BUF2:   DEFS    256             ;System buffer for File 2
00422   BUFFER: DEFS    256             ;Data buffer for both files
00423
00424           END     BEGIN           ;"begin" is the starting address
```

Sample Program F, continued

```
ØØ313            OR      A              ;Is it time to stop putting this on
ØØ314                                   ;the display?
ØØ315            RET     Z              ;Yes, return to the caller
ØØ316            PUSH    HL             ;Save the registers, as the SVC will
ØØ317            PUSH    DE             ;alter the contents
ØØ318            PUSH    BC
ØØ319            LD      C,A            ;Put the character here
ØØ32Ø            LD      A,@VDCTL       ;Put character on screen at specified position
ØØ321            RST     28H            ;Call the @VDCTL svc
ØØ322            POP     BC             ;Restore registers
ØØ323            POP     DE
ØØ324            POP     HL
ØØ325            INC     L              ;Advance display position
ØØ326            INC     DE             ;Point to next character to display
ØØ327            JR      TSKLP          ;Loop till date is completely displayed
ØØ328
ØØ329    MODEND: END     BEGIN          ;End of task and main program
```

Sample Program G

```
00001  ;       This program is a sample Extended Command Interpreter.  You
00002  ;       may make the ECI as large or small as you require.  You may
00003  ;       use allof main memory, or you can restrict yourself to the
00004  ;       system overlay area (x'2600' to x'2FFF').
00005  ;       To pass a command to the normal system interpreter for
00006  ;       processing, use the @CMNDI svc.  TRSDOS executes the command
00007  ;       and reloads the ECI.  If you want to have multiple entry
00008  ;       points, Bits 2 - 0 in EFLAG$ are in Register A on entry
00009  ;       (in Bits 6 - 4),or you may read EFLAG$ yourself.
00010  ;       EFLAG$ is totally dedicated to the ECI, and may contain any
00011  ;       non-zero value.  If EFLAG$ contains a zero, TRSDOS uses its
00012  ;       own interpreter.  Other programs that want to activate an ECI,
00013  ;       should set the EFLAG$ to a non-zero value and execute a @EXIT
00014  ;       svc.
00015
00016  ;       To install an ECI, use the command:
00017  ;               COPY filename SYS13/SYS.LSIDOS:d (C=N)
00018  ;       If you omit the C=N option, the SYS13 file loses it's "SYS"
00019  ;       status and you will receive 'Error 07' messages when you try
00020  ;       to use it as a ECI.
00021
00022  ;       When SYS1 (the normal command interpreter) has completed it's
00023  ;       normal housekeeping and is about to display the "TRSDOS Ready"
00024  ;       prompt, it checks EFLAG$.  If EFLAG$ contains a non-zero
00025  ;       value, TRSDOS loads and executes the Extended Command
00026  ;       Interpreter.
00027  ;       To execute this program, type <*><Enter>.
00028
00029  ;       This program checks EFLAG$ to see if it is zero.  If so, it
00030  ;       sets it to a non-zero value.  This causes this program to be
00031  ;       used instead of the normal interpreter when you execute an
00032  ;       @EXIT or @ABORT SVC.  (@CMNDI and @CMNDR invoke the TRSDOS
00033  ;       interpreter.)  If EFLAG$ is non-zero, the ECI displays a few
00034  ;       prompts and the names of all visible /CMD files on logical
00035  ;       Drive 0.
00036  ;       The operator may then type the name of a program to execute.
00037
00038  ;       If you press <Break>,  this program sets EFLAG$ to 0, executes
00039  ;       an @EXIT SVC and returns to TRSDOS Ready.
00040
00041  ;       By pressing a number, 0 through 7, you can specify the drive
00042  ;       that TRSDOS searches.  This program stores this value in
00043  ;       EFLAG$.  Each time this program is invoked, it reads the value
00044  ;       from EFLAG$ and uses that drive.
00045
00046  ;       Note that if a drive is not enabled, not formatted, doesn't
00047  ;       exist, or contains no visible /CMD files, this program
00048  ;       redisplays the prompt.
00049
00050           PRINT   SHORT,NOMAC
00051
00052           PSECT   3000H           ;This program starts at x'3000'
00053
00054  ;       Declare the equates for the SVCs used.
00055  ;       This is not mandatory, but it makes the program easier to
00056  ;       follow.
00057  @EXIT:   EQU     22             ;Exit and return to TRSDOS
00058  @DSPLY:  EQU     10             ;Display a string
00059  @FLAGS:  EQU     101            ;Locate the system flag area
00060  @DODIR:  EQU     34             ;Get the names of filenames
00061  @KEYIN:  EQU     9              ;Accept a command and allow editing
00062  @CMNDI:  EQU     24             ;Execute a command (using SYS1)
00063
00064  ;       On entry, determine if EFLAG$ is set to zero or not.  If it
00065  ;       is set to zero, this program is being started by typing
00066  ;       PROGRAM<Enter> or <*><Enter>.  In that case, set EFLAG$ to a
00067  ;       non-zero value so that in future, TRSDOS uses this interpreter
00068  ;       instead of it's own.
```

Sample Program G, continued

```
00069  ;       If EFLAG$ is non-zero, this initialization has already been
00070  ;       done and can be skipped.
00071
00072  BEGIN:  LD      A,@FLAGS        ;Get the starting address of the flag
area
00073          RST     28H             ;Call the @FLAGS svc
00074
00075          LD      A,(IY+4)        ;Read the EFLAG$ (ECI flag)
00076          OR      A               ;Is it set to zero?
00077          JR      NZ,ECIRUN       ;Run the ECI
00078
00079          LD      A,8             ;Get a non-zero value.  The value
00080                                  ;needs to be a non-zero value that
00081                                  ;does not set Bits 0, 1 or 2.  The
00082                                  ;default drive # is kept in these bits.
00083          LD      (IY+4),A        ;Set the EFLAG$ to a non-zero value
00084          LD      HL,PROMPT       ;Explain how this works
00085          JR      ECIGO           ;Display message
00086
00087  ;       When the system is about to display
00088  ;       TRSDOS Ready, it executes this code instead.
00089
00090  ECIRUN: LD      HL,SPROMPT      ;Point at the prompt to use
00091  ECIGO:  LD      A,@DSPLY        ;Display the prompt
00092          RST     28H             ;Call the @DSPLY svc
00093
00094  ;       Display the names of all /CMD files
00095
00096          LD      A,(IY+4)        ;Get the EFLAG$
00097          AND     7               ;Delete all but the drive number field
00098          LD      C,A             ;Store the drive number for the svc
00099          LD      A,@DODIR        ;Do a directory display
00100          LD      B,2             ;Display visible, non-system files
00101          LD      HL,CMDTXT       ;that match "CMD" (stored at CMDTXT)
00102          RST     28H             ;Call the @DODIR svc
00103
00104  ;       Prompt for a filename or a function key.
00105
00106  ASK:    LD      HL,BUFFER       ;Point at text buffer
00107          LD      B,9             ;Allow up to 8 characters and <Enter>
00108          LD      C,0             ;Required by the svc
00109          LD      A,@KEYIN        ;Input text with edit capability
00110          RST     28H             ;Call the @KEYIN svc
00111
00112          JR      C,QUIT          ;The carry flag is set when the
00113                                  ;operator presses <BREAK>. Zero the
00114                                  ;EFLAG$ and exit to TRSDOS
00115
00116          LD      HL,BUFFER       ;Point at the start of the buffer
00117          LD      A,(HL)          ;Get the character
00118
00119          CP      0DH             ;Did they type anything?
00120          JR      Z,ASK           ;No, just repeat the prompt.
00121                                  ;If you want to redisplay the
00122                                  ;directory, change "ASK" to "ECIRUN".
00123
00124          SUB     '0'             ;Convert value to binary
00125          CP      7+1             ;Is the character a 0 - 7?
00126          JR      NC,NAME         ;Must be a filename
00127
00128  ;       The operator has typed 1 or more characters that start with
00129  ;       a number.  This program assumes that the operator is defining
00130  ;       a new drive number and stores this value in EFLAG$ for
00131  ;       future use.  TRSDOS does not alter this value.
00132  ;       The next time this program is run, EFLAG$ contains the
00133  ;       same value and this program knows what drive to scan.
00134
00135          LD      B,A             ;Save the drive number
00136          LD      A,(IY+4)        ;Get the EFLAG$
```

Sample Program G, continued

```
ØØ137           AND     8               ;Delete the old drive number
ØØ138           OR      B               ;Insert the new drive number
ØØ139           LD      (IY+4),A        ;Save that value for future use
ØØ14Ø           JR      ECIRUN          ;Scan the new drive
ØØ141
ØØ142   ;       The operator pressed <Break>.  Turn off the ECI and return to
ØØ143   ;       TRSDOS.
ØØ144   QUIT:   XOR     A               ;Get a zero
ØØ145           LD      (IY+4),A        ;Set EFLAG$ to zero
ØØ146           LD      HL,EPROMPT      ;Point at the shutdown message
ØØ147           LD      A,@DSPLY        ;And acknowledge the <Break>
ØØ148           RST     28H             ;Call the @DSPLY svc
ØØ149           LD      A,@EXIT         ;Return to TRSDOS Ready
ØØ15Ø           RST     28H             ;Call the @EXIT svc
ØØ151
ØØ152   ;       The operator entered what might be a filename or a library
ØØ153   ;       command.  Pass it to TRSDOS for processing.  If there is an
ØØ154   ;       error, TRSDOS is responsible for determining what the error is
ØØ155   ;       and printing a message.
ØØ156   ;       (HL already points at the start of the buffer.)
ØØ157
ØØ158   NAME:   LD      A,ØDH           ;Look for this character
ØØ159   FDIV:   CP      (HL)            ;In the command
ØØ16Ø           JR      Z,FOUND         ;Found the end of the filename
ØØ161           INC     HL              ;Move character to next byte
ØØ162           JR      FDIV            ;Find the divider (in this case, a ØDH)
ØØ163
ØØ164  ;       Found the end of a filename, and add the drive number from
EFLAG$.
ØØ165   ;       Note that this program may not work properly if the operator
ØØ166   ;       supplies a drive number as part of the filename.
ØØ167
ØØ168   FOUND:  LD      (HL),':'        ;Add a drive number to the filename
ØØ169           INC     HL              ;Advance the pointer to the next byte
ØØ17Ø           LD      A,(IY+4)        ;Get the EFLAG$ value
ØØ171           AND     7               ;Delete all but the drive number
ØØ172           ADD     A,'Ø'           ;Convert the binary value to ASCII
ØØ173           LD      (HL),A          ;Add that to the filename
ØØ174           INC     HL              ;Advance the pointer to the next byte
ØØ175           LD      (HL),ØDH        ;Write a terminator on the end
ØØ176           LD      HL,BUFFER       ;Point at the text entered
ØØ177           LD      A,@CMNDI        ;Execute the command, but do not
ØØ178                                   ;return. Since this program is the
ØØ179                                   command processor at this time,TRSDOS
ØØ179                                   ;returns control to the beginning of
ØØ18Ø                                   ;this module after executing the
ØØ181                                   ;command.
ØØ182           RST     28H             ;Call the @CMNDI svc
ØØ183
ØØ184   ;       Messages and text storage
ØØ185
ØØ186   PROMPT: DEFM    '[Extended Command Interpreter Is Now Operational]'
ØØ187           DEFB    ØAH
ØØ188           DEFB    ØAH
ØØ189           DEFM    'Press <BREAK> to use the normal interpreter,'
ØØ19Ø           DEFB    ØAH
ØØ191           DEFM    'type <Number><ENTER> to change the default drive
                        number,'
ØØ192           DEFB    ØAH
ØØ193           DEFM    'or type the name of the program to run and press
                        <ENTER>'
ØØ194           DEFB    ØDH             ;Terminate the display
ØØ195
ØØ196   SPROMPT:DEFB    ØAH
ØØ197           DEFM    '[ECI On] <BREAK> to abort, n<ENTER> for new drive or
                        type:'
ØØ198           DEFM    ' program<ENTER>'
ØØ199           DEFB    ØDH             ;Terminate the message
ØØ2ØØ
```

Sample Program G, continued

```
ØØ2Ø1  EPROMPT:DEFM    '[Extended Command Interpreter Is Now Disabled]'
ØØ2Ø2          DEFB    ØDH
ØØ2Ø3
ØØ2Ø4  CMDTXT: DEFM    'CMD'
ØØ2Ø5  BUFFER: DEFS    11              ;Allow for filename, drivespec and ØDH
ØØ2Ø6
ØØ2Ø7          END     BEGIN           ;"BEGIN" is the starting address
```

HIT read error (Error 22, X'16')

A disk error occurred during the reading of the Hash Index Table. The problem may be media, hardware, or program failure. Move the diskette to another drive and try the operation again.

HIT write error (Error 23, X'17')

A disk error occurred during the writing of the Hash Index Table. The HIT may no longer be reliable. If the problem recurs, use a different drive or different diskette.

Illegal access attempted to protected file (Error 37, X'25')

The USER password was given for access to a file, but the requested access required the OWNER password. (See the ATTRIB library command in your *Disk System Owner's Manual*.)

Illegal drive number (Error 32, X'20')

The specified disk drive is not included in your system or is not ready for access (no diskette, non-TRSDOS diskette, drive door open, and so on). See the DEVICE command in your *Disk System Owner's Manual*.)

Illegal file name (Error 19, X'13')

The specified filespec does not meet TRSDOS filespec requirements. See your *Disk System Owner's Manual* for proper filespec syntax.

Illegal logical file number (Error 16, X'10')

A bad Directory Entry Code (DEC) was found in the File Control Block (FCB). This usually indicates that your program has altered the FCB improperly. Check for an error in your application program.

Load file format error (Error 34, X'22')

An attempt was made to load a file that cannot be loaded by the system loader. The file was probably a data file or a BASIC program file.

Lost data during read (Error 3, X'03')

During a sector read, the CPU did not accept a byte from the Floppy Disk Controller (FDC) data register in the time allotted. The byte was lost. This may indicate a hardware problem with the drive. Move the diskette to another drive and try again. If the error recurs, try another diskette.

Lost data during write (Error 11, X'0B')

During a sector write, the CPU did not transfer a byte to the Floppy Disk Controller (FDC) in the time allotted. The byte was lost; it was not transferred to the disk. This may indicate a hardware problem with the drive. Move the diskette to another drive and try again. If the error recurs, try another diskette.

LRL open fault (Error 42, X'2A')

The logical record length specified when the file was opened is different than the LRL used when the file was created. COPY the file to another file that has the specified LRL.

No device space available (Error 33, X'21')

You tried to SET a driver or filter and all of the Device Control Blocks were in use. Use the DEVICE command to see if any non-system devices can be removed to provide more space. This error also occurs on a "global" request to initialize a new file (that is, no drive was specified), if no file can be created.

No directory space available (Error 26, X'1A')

You tried to open a new file and no space was left in the directory. Use a different disk or REMOVE some files that you no longer need.

No error (Error 0)

The @ERROR supervisor call was called without any error condition being detected. A return code of zero indicates no error. Check for an error in your application program.

Parameter error (Error 44,X'2C')

(Under Version 6.2 only) An error occurred while executing a command line or utility because a parameter that does not exist was specified. Check the spelling of the parameter name, value, or abbreviation.

Parity error during header read (Error 1, X'01')

During a sector I/O request, the system could not read the sector header successfully. If this error occurs repeatedly, the problem is probably media or hardware failure. Try the operation again, using a different drive or diskette.

Parity error during header write (Error 9, X'09')

During a sector write, the system could not write the sector header satisfactorily. If this error occurs repeatedly, the problem is probably media or hardware failure. Try the operation again, using a different drive or diskette.

Parity error during read (Error 4, X'04')

An error occurred during a sector read. Its probable cause is media failure or a dirty or faulty disk drive. Try the operation again, using a different drive or diskette.

Parity error during write (Error 12, X'0C')

An error occurred during a sector write operation. Its probable cause is media failure or a dirty or faulty disk drive. Try the operation again, using a different drive or diskette.

Program not found (Error 31, X'1F')

The file cannot be loaded because it is not in the directory. Either the filespec was misspelled or the disk that contains the file was not loaded.

Protected system device (Error 40, X'28')

You cannot REMOVE any of the following devices: *KI, *DO, *PR, *JL, *SI, *SO. If you try, you get this error message.

Record number out of range (Error 29, X'1D')

A request to read a record within a random access file (see the @POSN supervisor call) provided a record number that was beyond the end of the file. Correct the record number or try again using another copy of the file.

Seek error during read (Error 2, X'02')

During a read sector disk I/O request, the cylinder that should contain the sector was not found within the time allotted. (The time is set by the step rate specified in the Drive Code Table.) Either the cylinder is not formatted or it is no longer readable, or the step rate is too low for the hardware to respond. You can set an appropriate step rate using the SYSTEM library command. The problem may also be caused by media or hardware failure. In this case, try the operation again, using a different drive or diskette.

Seek error during write (Error 10, X'0A')

During a sector write, the cylinder that should contain the sector was not found within the time allotted. (The time is set by the step rate specified in the Drive Code Table.) Either the cylinder is not formatted or it is no longer readable, or the step rate is too low for the hardware to respond. You can set an appropriate step rate using the SYSTEM library command. The problem may also be caused by media or hardware failure. In this case, try the operation again, using a different drive or diskette.

— Unknown error code

The @ERROR supervisor call was called with an error number that is not defined. Check for an error in your application program.

Write fault on disk drive (Error 14, X'0E')

An error occurred during a write operation. This probably indicates a hardware problem. Try a different diskette or drive. If the problem continues, contact a Radio Shack Service Center.

Write protected disk (Error 15, X'0F')

You tried to write to a drive that has a write-protected diskette or is software write-protected. Remove the write-protect tab, if the diskette has one. If it does not, use the DEVICE command to see if the drive is set as write protected. If it is, you can use the SYSTEM library command with the (WP = OFF) parameter to write enable the drive. If the problem recurs, use a different drive or different diskette.

Numerical List of Error Messages

Decimal	Hex	Message
0	X'00'	No Error
1	X'01'	Parity error during header read
2	X'02'	Seek error during read
3	X'03'	Lost data during read
4	X'04'	Parity error during read
5	X'05'	Data record not found during read
6	X'06'	Attempted to read system data record
7	X'07'	Attempted to read locked/deleted data record
8	X'08'	Device not available
9	X'09'	Parity error during header write
10	X'0A'	Seek error during write
11	X'0B'	Lost data during write
12	X'0C'	Parity error during write
13	X'0D'	Data record not found during write
14	X'0E'	Write fault on disk drive
15	X'0F'	Write protected disk
16	X'10'	Illegal logical file number
17	X'11'	Directory read error
18	X'12'	Directory write error
19	X'13'	Illegal file name
20	X'14'	GAT read error
21	X'15'	GAT write error
22	X'16'	HIT read error
23	X'17'	HIT write error
24	X'18'	File not in directory
25	X'19'	File access denied
26	X'1A'	Full or write protected disk
27	X'1B'	Disk space full
28	X'1C'	End of file encountered
29	X'1D'	Record number out of range
30	X'1E'	Directory Full — can't extend file
31	X'1F'	Program not found
32	X'20'	Illegal drive number
33	X'21'	No device space available
34	X'22'	Load file format error
37	X'25'	Illegal access attempted to protected file
38	X'26'	File not open
39	X'27'	Device in use
40	X'28'	Protected system device

41	X'29'	File already open
42	X'2A'	LRL open fault
43	X'2B'	SVC parameter error
63	X'3F'	Extended error
—		Unknown error code

Appendix D/Keyboard Code Map

The keyboard code map shows the code that TRSDOS returns for each key, in each of the modes: control, shift, unshift, clear and control, clear and shift, clear and unshift.

For example, pressing (CLEAR), (SHIFT), and (1) at the same time returns the code X'A1'.

A program executing under TRSDOS — for example, BASIC — may translate some of these codes into other values. Consult the program's documentation for details.

(BREAK) Key Handling

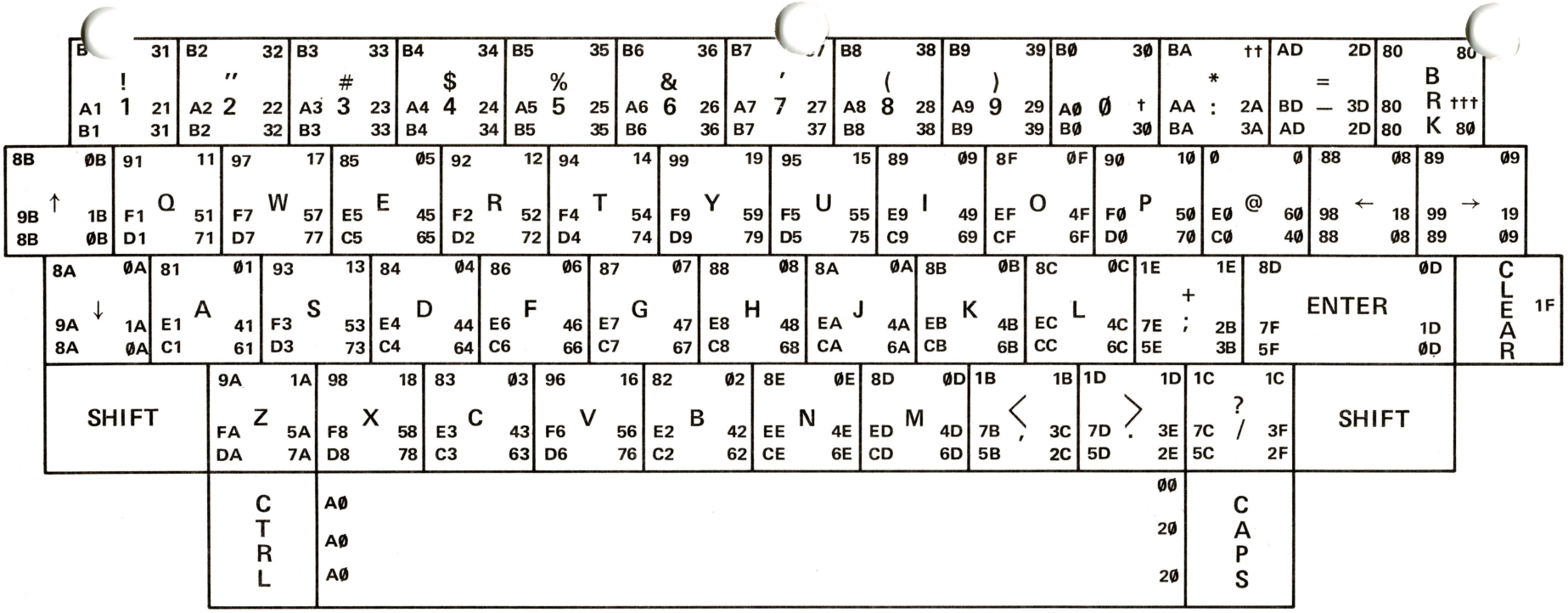
The (BREAK) key (X'80') is handled in different ways, depending on the settings of three system functions. The table below shows what happens for each combination of settings.

Break Enabled	Break Vector Set	Type-Ahead Enabled	
Y	N	Y	If characters are in the type-ahead buffer, then the buffer is emptied.* If the type-ahead buffer is empty, then a BREAK character (X'80') is placed in the buffer.*
Y	N	N	A BREAK character (X'80') is placed in the buffer.
Y	Y	Y	The type-ahead buffer is emptied of its contents (if any), and control is transferred to the address in the BREAK vector (see @BREAK SVC).*
Y	Y	N	Control is transferred to the address in the BREAK vector (see @BREAK SVC).
N	X	X	No action is taken and characters in the type-ahead buffer are not affected.

*Because the (BREAK) key is checked for more frequently than other keys on the keyboard, it is possible for (BREAK) to be pressed after another key on the keyboard and yet be detected first.

Y means that the function is on or enabled
N means that the function is off or disabled
X means that the state of the function has no effect

Break is enabled with the SYSTEM (BREAK = ON) command (this is the default condition).
The break vector is set using the @BREAK SVC (normally off).
Type-ahead is enabled using the SYSTEM (TYPE = ON) command (this is the default condition).



The keys may be positioned differently on your keyboard. However, they produce the same codes.

LEGEND:

Clear and Control — Control

Clear and Left Shift — Shift
Clear and Unshift — Unshift

Note: Pressing CONTROL, SHIFT, and @ at the same time generates an EOF (end of file) — — X'1C' with NZ return flag.

Whenever pressing CLEAR, SHIFT, and another key at the same time, be sure to use the left SHIFT key — not the right SHIFT key.

† Pressing SHIFT and Ø at the same time (or CAPS alone) turns the CAPS mode on or off.

†† Pressing CONTROL and : at the same time causes a screen print.

††† Pressing SHIFT and BREAK at the same time reselects the last drive.

Codes for these keys are the same as for the main keyboard.

81 F1 81	82 F2 82	83 F3 83
7	8	9
4	5	6
1	2	3
Ø	•	ENT

Appendix E/Programmable SVCs

(Under Version 6.2 only)

SVC numbers 124 through 127 are reserved for programmer installable SVCs. To install an SVC the programmer must write the routine to execute when the SVC is called.

The routine should be written as high memory module if it is to be available at all times. If you execute a SYSGEN command when a programmable SVC is defined, the address of the routine is saved in the SYSGEN file and restored each time the system is configured. If the routine is a high memory module, the routine is saved and restored as well. This makes the SVC always available. For more information on high memory modules, see Memory Header and Sample Program F.

To install an SVC, the program must access the SVC table. The SVC table contains 128 two-byte positions, a two-byte position for each usable SVC. Each position in the table contains the address of the routine to execute when the SVC is called.

To access the SVC table, execute the @FLAGS SVC (SVC 101). IY+26 contains the MSB of the SVC table start address. The LSB of the SVC table address is always 0 because the SVC table always begins on a page boundary.

Store the address of the routine to be executed at the *SVC number times 2* byte in the table. For example, if you are installing SVC 126, store the address of the routine at byte 252 in the table. Addresses are stored in LSB-MSB format.

When the SVC is executed, control is transferred to the address in the table. On entry to your SVC, Register A contains the same value as Register C. All other registers retain the values they had when the RST 28 SVC instruction was executed.

To exit the SVC, execute a RET instruction. The program should save and restore any registers used by the SVC.

Initially, SVCs 124 through 127 display an error message when they are executed. When installing an SVC you should save the original address at that location in the table and restore it when you remove the SVC.

These program lines insert a new SVC into the system SVC table, save the previous value of the table, and reinsert that value before execution ends. You could check the existing value to see if the address is above X'2600'. If it is, the SVC is already assigned and should not be used at this time.

This code inserts SVC 126, called MYSVC:

```
LD      A,@FLAGS        ;Locate start of SVC table
RST     28H             ;Execute @FLAGS SVC
LD      H,(IY+26)       ;Get MSB of address
LD      L,126*2         ;Want to use SVC 126
LD      (OSVC126A),HL   ;Save address of SVC entry
LD      E,(HL)          ;Get current SVC address
INC     HL
LD      D,(HL)
LD      (OSVC126V),DE   ;Save the old value
DEC     HL
LD      DE,MYSVC        ;Get address of routine for
                        ;SVC 126
LD      (HL),E          ;Insert new SVC address into
                        ;table
INC     HL
```

```
        LD      (HL),D
        .
        . Code that uses MYSVC (SVC 126)
        .
        .
```

This code removes SVC 126:

```
        LD      HL,(OSVC126A)   ;Get address of SVC entry
        LD      DE,(OSVC126V)   ;Get original value
        LD      (HL),E          ;Insert original SVC address
        INC     HL
        LD      (HL),D
```

Appendix F/Using SYS13/SYS

(Under Version 6.2 only)

With TRSDOS Version 6.2, you can create an Extended Command Interpreter (ECI) or an Immediate Execution Program (IEP). TRSDOS can store either an ECI or IEP in the SYS13 file. Both programs cannot be present at the same time.

At the TRSDOS Ready prompt when you type (*) (ENTER), TRSDOS executes the program stored in SYS13/SYS. Because TRSDOS recognizes the program as a system file, TRSDOS includes the file when creating backups and loads the program faster.

If you want to write additional commands for TRSDOS, you can write an interpreter to execute these commands. Your ECI can also execute TRSDOS commands by using the @CMNDI SVC to pass a command to the TRSDOS interpreter.

If EFLAG$ contains a non-zero value, TRSDOS executes the program in SYS13/SYS. If EFLAG$ contains a zero, TRSDOS uses its own command interpreter.

Sample Program G is an example of an ECI. It is important to note that your ECI must be executable by pressing (*) (ENTER) at the TRSDOS Ready prompt.

An ECI can use all of memory or you can restrict it to use the system overlay area (X'2600' to X'2FFF').

To implement an IEP or ECI, use the following syntax:

COPY *filespec* SYS13/SYS.LSIDOS:*drive* (C=N) (ENTER)

filespec can be any executable (/CMD) program file. *drive* specifies the destination drive. The destination drive must contain an original SYS13/SYS file.

Example

COPY SCRIPSIT/CMD:1 SYS13/SYS.LDI:0 (C=N)

TRSDOS copies SCRIPSIT/CMD from Drive 1 to SYS13/SYS in Drive 0. At the TRSDOS Ready prompt, when you press (*) (ENTER), TRSDOS executes SCRIPSIT.

Index

Index

Index

Index

Index