CAT. NO. 26-1143

# TRS-80® Model I

## Double-Density Disk System Owners Manual

Radio Shack®

The biggest name in little computers®

CUSTOM MANUFACTURED IN THE USA BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

# *A Note On Double-Density BACKUP . . .*

The TRS-80® Model I Double-Density System diskette supplied with this package will *not BACKUP* in Drive 0 when the diskette has a write-protect tab.

We suggest you keep a write-protect tab on your original Double-Density TRSDOS diskette *until* you are ready to duplicate it. Then *remove the write-protect tab* and follow the instructions on Page 3 of the Owner's Manual to make a safe copy.

When you have finished backing it up, *replace* the write-protect tab to label it as the "master" diskette.

Thank You

**Radio Shack®**

8759175

# SERVICE POLICY

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

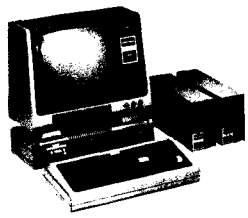
1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.
2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.
3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.

# TRS-80®
# Model I

# Double-Density Disk System Owner's Manual

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

10 9 8 7 6 5 4 3 2 1



# To Our Customers

Congratulations on your purchase of a Model I Double-Density Disk System. Your Single-Density Disk System has been enhanced with additional commands and the capability of storing more data on a 5¼" diskette.

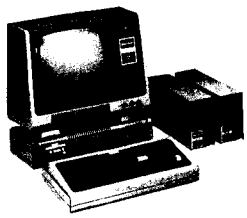
Application programs and associated data in both Assembly Language and BASIC can be copied by the user from a Single-Density Model I diskette to Double-Density. Most of these programs can then be run taking advantage of the Double-Density increased storage capability. However, Radio Shack applications software is not available or supported for double-density operation.

The Model I Disk Operating System has been updated to be comparable to the TRSDOS in Model III including most of the Library (and BASIC) commands. There are several additional Library commands such as TRACE, UNKILL and VERIFY. DIRectory will list a file from either Double-Density or Single-Density in the Model III format. COPY allows you to copy from Single-Density to Double-Density or vice versa. BACKUP and FORMAT only function in Double-Density.

Shugart Disk Drives will not support 40 track operation. Disk drives with a serial number followed by a (-1) are manufactured by Tandon or Texas Peripherals and are certified for double-density and can be CONFIGured to 40 tracks.

In addition to the new operating system, there is a hardware change that must be made to your computer. This Double-Density Adapter must be installed by a qualified Radio Shack technician. With this hardware modification, you can operate your Model I computer with your present Single-Density operating system or with the new Double-Density operating system which is supplied with this package.

Your system can continue to grow in power and convenience. When Radio Shack issues improvements and enhancements to the system programs, you can "install" them simply by obtaining a new release of the TRSDOS diskette.



# Introduction

The Double-Density Disk System allows increased storage on a 5¼" diskette because the data is stored closer together (more dense). An 80% increase in disk storage capability is available using 35 tracks and more than double the capacity if 40 tracks are used.

Because the disk structure is different, Double-Density TRSDOS will only support the following library commands on single-density diskettes; DIR (directory) and COPY.

There are many new commands in Double-Density TRSDOS/BASIC that allow your TRS-80 Model I to perform almost like a Model III and utilize enhancements that were not previously available. Information stored as single-density may easily be converted to double-density using FORMAT and COPY. However, Double-Density TRSDOS will not read a diskette which is in Model III format.

On power-up, single-density TRSDOS will be used. If a double-density operating system diskette is in Drive 0, a dual disk controller is activated and subsequent operations are in double-density. If a single-density operating system is in Drive 0, the system will remain in single-density and you can use all the features of the resident operating system.

A hardware modification is necessary to allow use of double-density TRSDOS. With the installation of a Double-Density Adapter, you have the option of using a single- or double-density operating system. In other words, after modification, your disk system can still be used as a single-density system.

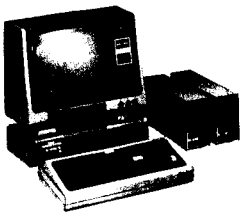
## Important Notice

The Double-Density adapter must be installed by a qualified Radio Shack Service Technician. Unauthorized installation will void the warranty.

# Model I Manuals

Publications related to the use of the Model I Double-Density Disk System:

1. *Model I Double-Density Disk System Owner's Manual* (this manual). We'll call it the "Double-Density Manual" for short.
2. *Model I Double-Density Disk System Quick Reference Card.*
3. *Model I Disk System Owner's Manual,* the "Disk Manual" for short.
4. *Level II BASIC Reference Manual,* the "Model I Manual" for short.





**For Disk Operation:**

This Double-Density Disk Manual supplements the Level II BASIC Reference Manual and the Model I Disk System Owner's Manual. Use the prior manuals as the primary source of information.

**For Programming Information:**

The Level II Manual contains most of the programming information, except that which pertains to disk input/output. In this manual, we will assume that you are familiar with the BASIC programming definitions and details given in the Model I manuals.

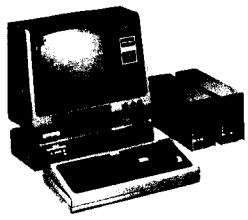
# About This Manual

The Model I Double-Density Disk System is intended for use by novices as well as experienced computer operators and programmers. In designing and writing this Double-Density Disk Manual, we've tried to define and satisfy the needs of both groups:

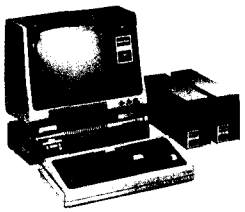
- Novices who might prefer a sequential presentation which emphasizes procedures and explains the purpose of various features.
- Experienced users who might prefer a more analytical presentation which makes it easy to find specific information.

In this manual, you'll find information that should satisfy your needs, whichever group you might belong to.

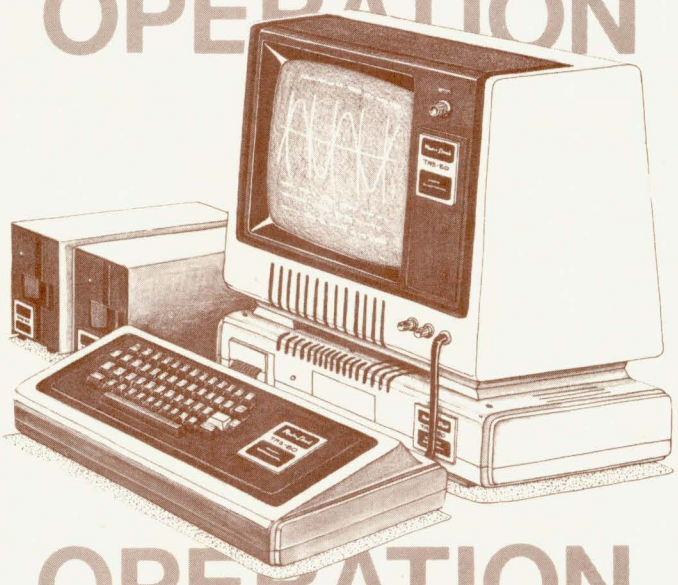
The TRSDOS and BASIC commands used with Double-Density are similar to Model I commands. The sections that include these commands have been included in their entirety in this manual to avoid confusion.



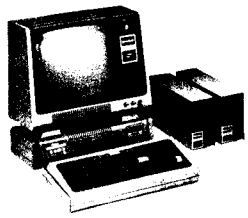
# Contents



OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION



OPERATION
OPERATION
OPERATION
OPERATION
OPERATION



# 1/Getting Started with the Double-Density System

## Start-Up Sequence

When turning power to the Computer on or off, all drives should be empty. Leaving the diskettes in the drives may cause information previously stored to be changed or even destroyed.

Do not turn a peripheral on or off during a disk read/write operation (when the drive LED is illuminated). Work done on a currently open file may be lost. Also note that turning the peripherals on and off while the Computer is on may confuse the system and cause abnormal operation.

The power switch for each Mini Disk is on the rear of the unit. Power is "on" when the toggle switch is in the up position, and "off" when the switch is down.

The Start-Up sequence is as follows:

1. Turn all peripherals (printer, disk drives, etc.) ON.
2. Turn the Expansion Interface ON.
3. Turn the CPU/keyboard ON Wait until all disk drive motors have stopped, then carefully insert your double-density TRSDOS diskette into Drive 0. You

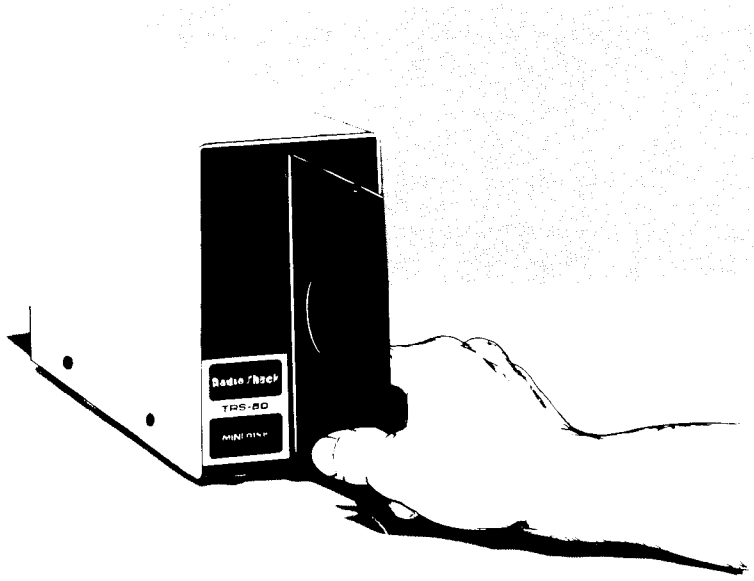
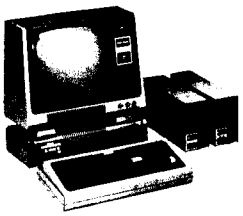



**Figure 1.** Inserting a Diskette.





may also want to insert formatted diskettes in the other drives. **Note:** It is normal for random "garbage" to be displayed on the video at this time.

4. Press the RESET button (located at the left rear side of the CPU/keyboard).
5. LOADING TRSDOS should now be displayed on the Video, if not repeat Steps #3 & 4.

TRSDOS should now load its start-up dialog. If it does not load, check the following items:

- Is the diskette you are using a TRSDOS "system" diskette?
- Is the diskette properly inserted into Drive 0?
- If more than one drive is present, are they properly connected and turned on?

# TRSDOS Start-Up Dialog

Whenever you start up or reset the Model I Double-Density Disk System, the message `LOADING TRSDOS` will be displayed. After you insert a System disk into Drive 0 and close the door, TRSDOS will load and the start-up dialog will appear.

1. The TRSDOS version number and date of creation will be displayed.
2. Displayed next is the amount of RAM and the number of drives in the system.
3. TRSDOS will prompt you to enter the date in the form of `MM/DD/YY` (i.e., 07/14/81 for July 14, 1981). Type in the correct date and press (ENTER). TRSDOS will not continue until the date is entered correctly.
4. Next, TRSDOS will prompt you to enter the time in the form of `HH:MM:SS` (i.e., 14:45:00 for 2:45 p.m.). Type in the correct time and press (ENTER). If you don't wish to set the time, simply press (ENTER) at the beginning of the line. TRSDOS will set the time to `00:00:00`.
5. TRSDOS will now display the message:

```
TRSDOS READY
```

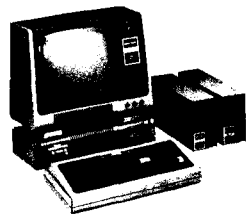
Whenever this is displayed, you are in the TRSDOS READY mode and may type in a TRSDOS command.

**Note:** Once the time and date has been entered, it will not have to be re-entered again as long as:

- Power remains ON.
- If reset, the TRSDOS version does not change.

# Making a BACKUP (Duplicate) of TRSDOS

Your first operation should be to duplicate the TRSDOS diskette you received from Radio Shack. The TRSDOS diskette contains a utility program called



BACKUP to accomplish this. (Just for safety, place a write protect tab on the TRSDOS diskette until you have duplicated it).

1. Locate the TRSDOS diskette and a new, blank diskette. The TRSDOS diskette will be referred to as the "source," while the blank one will be called the "destination," during BACKUP.
2. Start TRSDOS as explained in the previous section. `TRSDOS READY` should be displayed.
3. Type: `BACKUP` (ENTER).
4. TRSDOS will now load and start BACKUP. It will ask you:

   `SOURCE DRIVE NUMBER?`

   Specify the drive which contains the original TRSDOS diskette by typing: `0` (ENTER).
5. Next TRSDOS will ask:

   `DESTINATION DRIVE NUMBER?`

   Now specify the drive which will be used for making the duplicate TRSDOS. If you have two or more drives in your system, type: `1` (ENTER). If you only have one disk drive, type: `0` (ENTER). You will be prompted when to change diskettes.
6. TRSDOS will ask:

   `SOURCE DISK MASTER PASSWORD?`

   Press (ENTER) (PASSWORD will be used).
7. TRSDOS will ask:

   `QUICK or FULL VERIFY?`

   Press (ENTER) (QUICK VERIFY will be used).
8. TRSDOS will analyze the diskette to determine if it contains data or has been previously FORMATted.
9. If there is data on the diskette, TRSDOS will ask:

   `DISKETTE CONTAINS DATA, USE DISK OR NOT?`

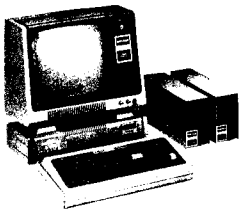
   Type `Y`, and press (ENTER).
10. FORMAT or the continuation of the duplication process will begin. No more questions will be asked.
11. On completion, the following message will appear:

```
* * BACKUP COMPLETE * *
TRSDOS READY
```

# Making a Data Diskette (FORMAT)

*This section applies to multi-drive systems only.*

Drive 0 must always contain a TRSDOS diskette, so the Computer can have access to the system programs stored there. Much of the storage capacity of this





diskette is taken up by the system programs.

However, the other drives in the system may contain "data" diskettes which do not have system programs. All of the storage capacity of such diskettes is available for your programs and data.

The FORMAT utility program takes a diskette and initializes or "formats" it. If the diskette was previously formatted and contained data, all prior information can be lost. The resultant diskette does not contain system files and may only be used in Drive 1, 2 or 3.

1. After TRSDOS READY, type: FORMAT (ENTER)
2. TRSDOS will start the formatter program and ask you:

   FORMAT WHICH DRIVE?

   Type: 1 (ENTER)

   DISKETTE NAME?

   Press (ENTER), (TRSDOS will be used).

   MASTER PASSWORD?

   Press (ENTER), (PASSWORD will be used).

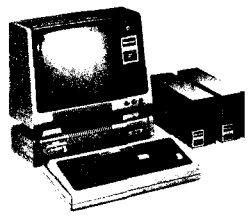
   ANALYZING DISKETTE is displayed. The destination diskette may be new or unformatted. If the diskette is not formatted, the FORMAT process will automatically begin.

   The destination diskette may contain data. TRSDOS will warn:

   DISKETTE CONTAINS DATA, USE DISK OR NOT?

   Type: Y (ENTER), and the FORMAT process will automatically begin, and all prior data will be erased. After formatting is complete, the following message will be displayed:

   WRITING SYSTEM INFORMATION
   TRSDOS READY



# Disk BASIC

## Quick Instructions for Using Disk BASIC

In this section we'll "walk you" through the following procedures:

- Starting Disk BASIC
- Running a simple program
- Saving a program in a disk file
- Loading a program from a disk file

For programming information, see the **Disk BASIC** section of this manual. Here we are showing procedures only.

## Starting Disk BASIC

Under TRSDOS READY, type: `BASIC` (ENTER)

The Computer will load and start BASIC. First, it will ask two questions. Press (ENTER) in response to each of them.

```
HOW MANY FILES? (ENTER)
MEMORY SIZE? (ENTER)
```

A heading will be displayed, followed by:

```
READY
>
```

You may now begin using Disk BASIC.

## Saving a Program

You should have a program in memory and be in BASIC's READY mode. Type:

`SAVE "PROGRAM"` (ENTER)

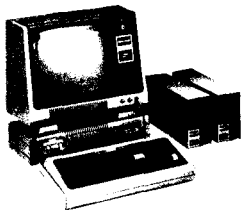
BASIC will now save the program in a disk file we arbitrarily named "PROGRAM." Any other suitable **file name** would do.

## Loading a Program

For this sample session, we will load the program just saved.

First type: `NEW` (ENTER) to erase it from memory. (This is to show that it *can* be retrieved from the disk file.)

Now type: `LOAD "PROGRAM"` (ENTER) and BASIC will load the specified program.



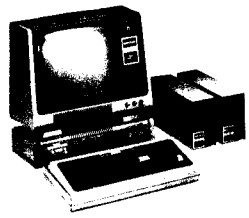


You may now list it and run it.

For further information on using Disk BASIC, see Section 3 of this manual.

## Cassette Baud Rate under Disk BASIC

TRSDOS sets the cassette Baud Rate at 500 Baud. This Baud Rate is fixed by the hardware configuration of TRS-80 Model I.



# Troubleshooting and Maintenance

If you have problems operating your Model I Disk System, please check the following symptoms and cures.

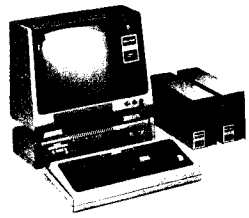
If you can't solve the problem, take the unit to your local Radio Shack. We'll have it fixed and returned to you as soon as possible.

| Symptom | Cure |
|---|---|
| Disk drive motors run continuously when the Computer is turned on. | Check external drive connection sequence. Drive 26-1160 must always be the last external drive. |
| Computer will not load TRSDOS. | 1. Make sure you have inserted the TRSDOS diskette properly in Drive 0.<br>2. Make sure all peripherals are properly connected. |
| Error Messages | Look up the message in the **TRSDOS** or **BASIC Error Message** Section. The "cure" should be listed. |
| Frequent disk I/O errors | 1. Diskette is partially erased. Backup the diskette, then re-format it.<br>2. Diskette is worn out. Use backup copy, if available, to make a new working copy.<br>3. Disk drives need cleaning or alignment by Radio Shack service technicians.<br>4. Drive is not configured to the proper stepping rate. Reconfigure or reboot. (See CONFIG.) |

## Maintenance

For reliable operation, the disk drives must be kept clean and properly aligned. These procedures should be done by Radio Shack service technicians, according to the following schedule:

| Degree of Use | Maintenance Interval |
|---|---|
| Commercial data processing environment | Every 6 months for medium use. |
| Occasional home use | Every 8-10 months; more often if needed. |



# Notation and Abbreviations

For the sake of clarity and brevity, we've used some special notation and type styles in this book.

CAPITALS and punctuation

indicate material which must be entered exactly as it appears. (The only punctuation symbols not entered are ellipses, explained below.) For example, in the line:

**DUMP LISTER (START=7000,END=7100,TRA=7004)**

every letter and character should be typed as indicated.

*lowercase italics*

represent words, letters, characters or values you supply from a set of acceptable values for a particular command. For example, the line:

**LIST *filespec***

indicates that you can supply any valid **file specification** after LIST.

. . .

Ellipses indicates that the preceding items can be repeated. For example:

**ATTRIB *filespec (option, . . .)***

indicates that several options may be repeated inside the parentheses.

b

This special symbol is used occasionally to indicate a blank-space character (ASCII code 32 decimal, 20 hexadecimal).

**PRINT "bHbIb!"**

X'*nnnn*'

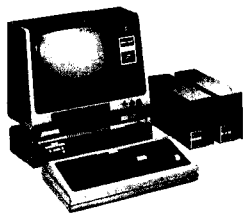
Indicates that *nnnn* is a hexadecimal number. All other numbers in the text of this book are in decimal form, unless otherwise noted.

**X'7000'**

indicates the hexadecimal value 7000 (decimal 28672).

COMPUTER TYPE

Any words, letters, or numbers that are displayed on the Screen will be in computer type (dot-matrix). Uppercase letters are used; however, your Screen at times may display lowercase letters instead.

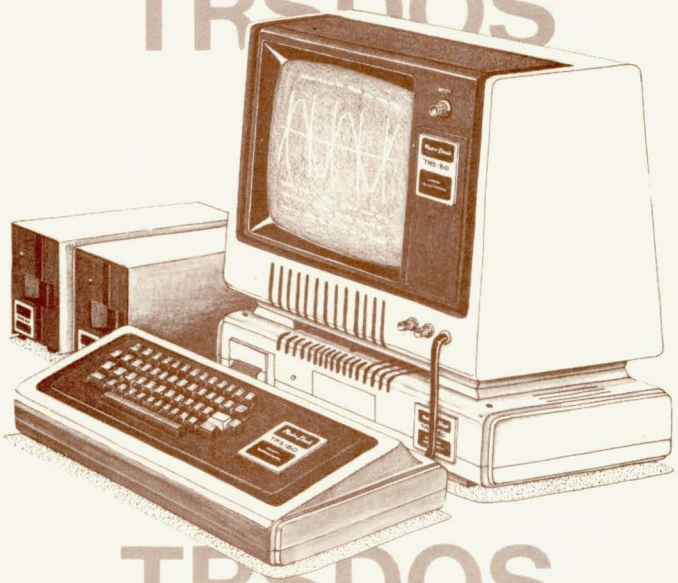


# Specifications

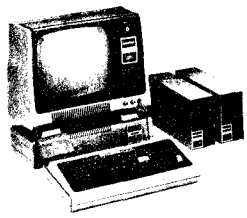
| | |
|---|---|
| Diskettes | 5¼" mini-diskettes<br>Radio Shack Catalog<br>Number 26-305,<br>26-405 (package of three),<br>or 26-406 (package of 10) |
| Diskette Organization | |
| Formatted Diskette | |
| Double-Density | Single-Sided<br>Double-Density<br>40 or 35 Tracks<br>18 Sectors/Track<br>256 Bytes/Sector<br>6 granules/track |
| Single-Density | Single-Sided<br>Single-Density<br>35 Tracks<br>10 Sectors/Track<br>256 Bytes/Sector<br>2 granules/track |
| Storage Capacity | |
| Double-Density (35 tracks) | |
| Formatted Diskette (DATA) | 198 Granules |
| TRSDOS Diskette without BASIC | 128 Granules |
| TRSDOS Diskette with BASIC | 120 Granules |
| Operating Temperature | 55 to 80 Degrees Fahrenheit<br>13 to 27 Degrees Celsius |
| Power Requirements | 120 VAC, 60 Hz, 28 VA<br>240 VAC, 50 Hz, Australian;<br>220 VAC, 50 Hz, European. |

**Double-Density diskettes are required for proper operation of the Double-Density Adapter. Radio Shack Double-Density diskettes can be identified by a "hub ring" in the center of the diskette.**

TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS



TRSDOS
TRSDOS
TRSDOS
TRSDOS



# 2 / Description of TRSDOS

## What Is TRSDOS?

TRSDOS (pronounced "TRISS-DOSS") stands for "TRS-80 Disk Operating System." It is a comprehensive set of system routines and file management utilities. TRSDOS fulfills three roles:

1. Master Program
2. Command Interpreter
3. Program Manager

As the master program, TRSDOS enables the microprocessor and its various components to interact efficiently. The components include:

- Random Access Memory (RAM). TRSDOS reserves space for its own needs and allocates space for user programs.
- Disk Drives. TRSDOS interfaces with the disk hardware and provides a file system for storing system and user data on diskettes.
- Input/output devices. These include the keyboard, video display, printer, and RS-232-C equipment (if installed).

TRSDOS is also a command interpreter. Whenever it displays `TRSDOS READY`, you may enter commands that control how the system works. These are known as "library" commands.

In its role as program manager, TRSDOS will load and run system or user programs. During this time, the system or user program is in control of the Computer.

**Figure 2** illustrates the relationships between these three roles.

## Where Does BASIC Fit In?

Referring to **Figure 2**, you'll see that Disk BASIC falls under the "language package" category.

Disk BASIC consists of some general enhancements to Level II BASIC, plus the disk input/output capability. It uses Level II BASIC (stored in ROM) whenever possible. For instance, the Level II BASIC ROM includes all of the mathematical functions.

If you're used to the non-disk system, there's one difference you should understand from the beginning: In the non-disk system, BASIC is in control when you start-up. In the disk system, however, TRSDOS is in control when you start-up. You have to tell TRSDOS to load and run BASIC. Only *then* can you begin running a program written in BASIC.

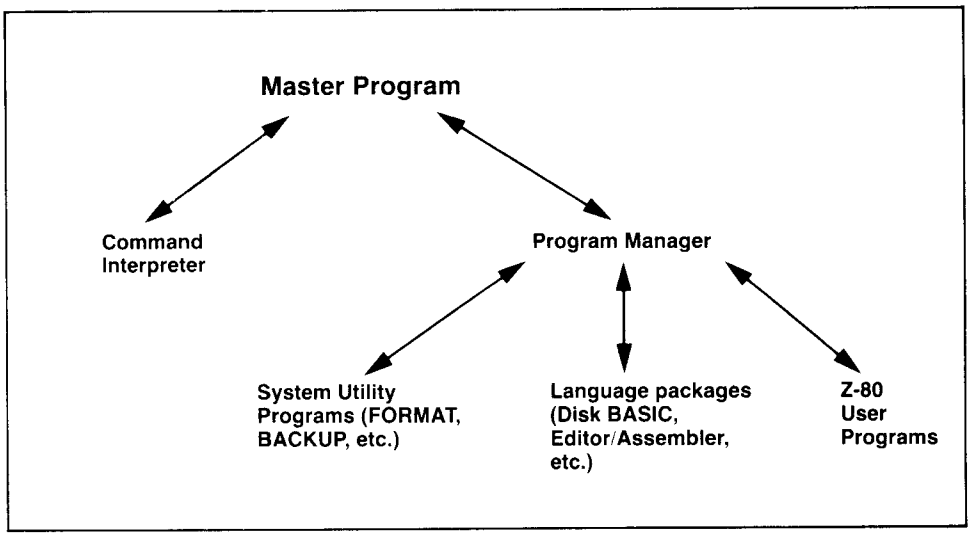



**Figure 2.** TRSDOS Roles.

# How TRSDOS Uses RAM

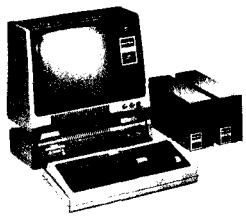
TRSDOS consists of:

- an executive program file
- auxiliary system-routine files
- a library-command file
- and the Disk BASIC file.

The *executive program* is loaded into RAM on power-up, and remains there at all times while TRSDOS is running. For this reason it is called the "*resident*" TRSDOS program. It includes certain system routines, tables, pointers, and Input/Output drivers.

The *auxiliary system files* contain routines and commands which are loaded as needed to execute your commands and programs. These routines load into an "overlay" area of memory. When TRSDOS has executed the routine, another one may be loaded in the same area, or "overlayed." The use of overlays means that execution of system routines will not affect your memory area (addresses above 51FF hex).

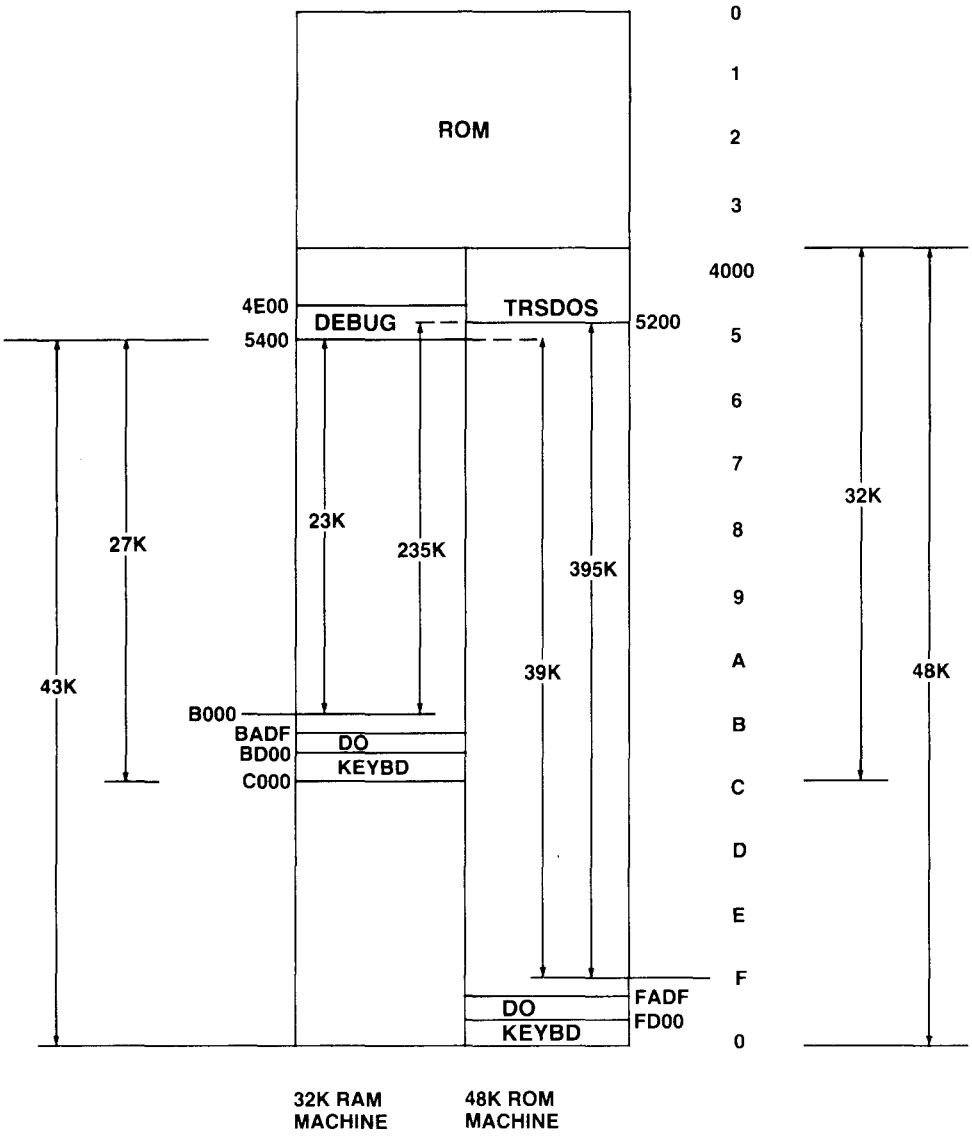
The *library command file* contains the routines for executing most of the operator commands. These routines load into memory addresses from 5200 to



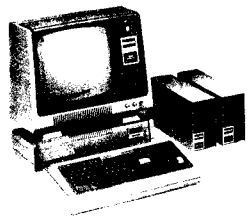
6FFF. Therefore your machine language programs should generally be located above 6FFF. That way they won't be affected by execution of the library commands.

There are three library commands which use all available memory. They are BACKUP, FORMAT and COPY (for single-drive copying.)

*Disk BASIC* is a set of enhancements to Level II BASIC. When you type in its file name, BASIC, it will load into memory beginning at 5200, and begin execution.




TRSDOS Memory Map



# Using TRSDOS

## Entering a Command

Whenever the TRSDOS prompt

```
TRSDOS READY
```

is displayed, you can type in a command, which can be no more than 63 characters in length. You must press (ENTER) to end the line. TRSDOS will then "accept" the command.

For example, type: `CLS` (ENTER) TRSDOS will clear the Display and the TRSDOS READY prompt will reappear.

In general, your commands will require more than one word. For example, to kill (delete) a file named MYNAME, you have to specify the command and the filename.

```
KILL MYNAME (ENTER)
```

tells TRSDOS to find the file named MYNAME, eliminate it from the diskette, and release the space previously occupied by that file.

Whenever you type in a line, TRSDOS follows this procedure:

1. First it checks to see if what you've typed is the name of a TRSDOS command. If it is, TRSDOS executes it immediately.
2. If what you typed is not a TRSDOS command, then TRSDOS will check to see if it's the name of a program file on one of the drives.
3. When searching for a file, TRSDOS looks first through Drive 0, then Drive 1, etc., unless you include a particular **drive specification** with the file name — or specify the MASTER command (see **Library Commands**).

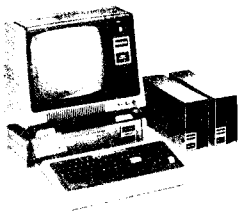
   If TRSDOS finds a specified user file, it will load and execute the file if it is a program file. Otherwise, you'll get an error message.

### System, Program, and Data Files

System files contain the TRSDOS routines necessary to execute your commands. These files include language software released by Radio Shack.

Program files are stored in a special program file format which allows them to be loaded and executed directly from TRSDOS. The BASIC interpreter is an example of a Program file.

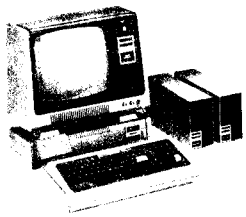
Data files are not stored in Program file format but are stored as data files. Data files cannot be loaded or executed from TRSDOS. Programs saved from BASIC or created by other Program files are stored as data files.





## Repeat Key

The Double-Density Model I has the ability to "repeat" keystrokes. By holding a key down, the keyboard outputs a steady stream of that character.



# 3/TRSDOS Commands

## APPEND
## Append files

**APPEND *source-file destination-file***

***source-file* is the specification for the file which is to be copied onto the end of the other file.**

***destination-file* is the specification for the file which is to receive the appendage (addition).**

**Note: Both *source-* and *destination-files* must be in ASCII format (data files or BASIC programs saved with the A option).**

APPEND copies the contents of the *source-file* onto the end of the *destination-file*. The *source-file* is unaffected, while the *destination-file* is extended to include the *source-file*.

**Note:** The logical record lengths must match. See DIR for more information on logical record lengths.

This command is most useful for data files and is not recommended for BASIC or machine-language programs. Because of an end-of-program marker TRSDOS puts at the end of these programs, if these types of files are appended they would not load (or run) past the end-of-program marker of the first program. In other words, the second program would be ignored. To combine two BASIC program files, use the BASIC command MERGE. Machine-language programs cannot be appended.

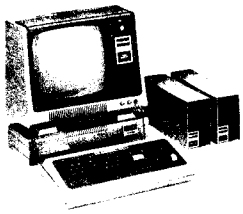
### Examples

```
APPEND WORDFILE/C WORDFILE/D
```

A copy of WORDFILE/C is appended to WORDFILE/D.

```
APPEND REGION1/DAT TO TOTAL/DAT,GUESS
```

A copy of REGION1/DAT is appended to TOTAL/DAT, which is protected with the password GUESS. **Note:** The delimiter "TO" is optional.





## Sample Uses

Suppose you have two data files, PAYROLL/A and PAYROLL/B.

PAYROLL/A

```
Atkins, W.R. .......
Baker, J.B. ........
Chambers, C.P. .....
Dodson, M.W. .......
Kickamon, T.Y. .....
```

PAYROLL/B

```
Lewis, G.E. ........
Miller, L.O. .......
Peterson, B. .......
Rodriguez, F. ......
```

You can combine the two files with the command:

```
APPEND PAYROLL/B PAYROLL/A
```

PAYROLL/A will now look like this:

PAYROLL/A

```
Atkins, W.R. .......
Baker, J.B. ........
Chambers, C.P. .....
Dodson, M.W. .......
Kickamon, T.Y. .....
Lewis, G.E. ........
Miller, L.O. .......
Peterson, B. .......
Rodriguez, F. ......
```

PAYROLL/B will be unaffected. To see the APPENDed file, type `LIST PAYROLL/A (ASCII)`.

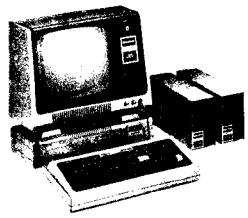
# ATTRIB
# Change a File's Password

**ATTRIB *filespec (visibility*,ACC = *name*,UPD = *name*,PROT = *level)***

***visibility* must be I or N. Tells TRSDOS whether the file is Invisible (I) or Non-invisible (N) (see DIR). If omitted, *visibility* is unchanged.**

**ACC = *name* tells TRSDOS the access word. If omitted, the access word is unchanged. If ACC =, is used, the access word is set to blanks.**

**UPD = *name* tells TRSDOS the update word. If omitted, the update word is unchanged. If UPD =, is used, the update word is set to blanks.**



**PROT = *level* tells TRSDOS the protection level for access. If omitted, *level* is unchanged.**

| Level | Degree of access granted by access word |
|---|---|
| FULL | Full access, no protection. |
| KILL | Kill, rename, read, execute, and write (gives total access, i.e., the least-protected). |
| NAME | Rename, read, execute, and write. |
| WRITE | Read, execute, and write. |
| READ | Read and execute. |
| EXEC | Execute only. |

**Note: Each level allows access to itself plus all lower levels.**

ATTRIB lets you change the passwords to an existing file or makes the file invisible or non-invisible. Passwords are initially assigned when the file is created. At that time, the update and access words are set to the same value (either the password you specified or a blank password).

## Examples

```
ATTRIB DATAFILE (I,ACC=JULY14,UPD=MOUSE,PROT=READ)
```

Makes the file invisible, sets the access password to JULY14 and the update password to MOUSE. Use of the access word will allow only reading and executing the file.

```
ATTRIB PAYROLL/BAS,SECRET (N,ACC=,)
```

Sets the access word to blanks. The file is made non-invisible and the protection level assigned to the update word is left unchanged.

```
ATTRIB OLD/DAT,APPLES (UPD=,)
```

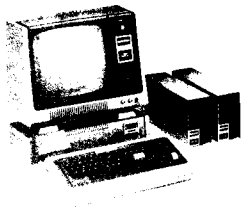
Sets the update word to blanks.

```
ATTRIB PAYROLL/BAS,PW (PROT=EXEC)
```

Leaves the access and update words unchanged, but changes the level of access.

```
ATTRIB PAYROLL/BAS,PW (N,ACC=, UPD=, PROT=FULL)
```

Sets both the passwords to blanks and removes all protection.



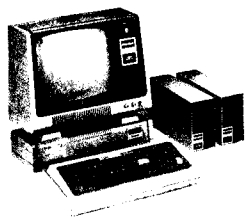


## Sample Uses

Suppose you have a data file, PAYROLL, and you want an employee to use the file in preparing paychecks. You want the employee to be able to read the file but not to change it. Then use a command like:

```
ATTRIB PAYROLL (I,ACC=PAYDAY,UPD=AVOCADO,PROT=READ)
```

Now tell the clerk to use the password PAYDAY (which allows read only); while only you know the password, AVOCADO, which grants total access to the file.



## Protecting BASIC Programs

You may give a BASIC program execute-only protection using the ATTRIB command. For example, suppose the program is named TEST (no password). Under TRSDOS READY, execute this command:

```
ATTRIB TEST (ACC=,UPD=VALLEY,PROT=EXEC)
```

Now TEST has a blank access password, an update password of VALLEY, and an access level of execute only. Without using the update password, there is now only one way to execute the program:

1. Start BASIC.
2. Type: `RUN "TEST"`

   (This is the only way to access the program. If the operator attempts to LOAD it instead, BASIC will erase the program from memory before returning with READY.)

After the RUN "TEST" command, BASIC will load and execute the program. If the operator presses (BREAK) or if the program ends normally, BASIC will erase the program before returning with the READY message. This makes it impossible to obtain a listing of the program — unless the update password is used.

Of course, if you use the update password, you may gain full access to the program.

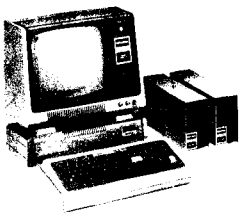
# AUTO
# Automatic Command after System Start-up

**AUTO *:d command-line***

***:d* specifies a drive number where the AUTO is to be written. *:d* is optional; if omitted Drive 0 is used.**

***command-line* gives TRSDOS a command or the name of an executable program file created by BUILD. If *command-line* is given, the command will be executed on Reset/Power-up. If *command-line* is omitted, the previous AUTO command is erased from the diskette. *command-line* must not exceed 30 characters plus the carriage return.**

This command lets you provide a command to be executed whenever TRSDOS is started (Power-up or Reset). You can use it to get a desired program running without any operator action required, except for typing in the date and time.





When you enter an auto command, TRSDOS writes the command line into the start-up procedure of the diskette in the drive you specified (provided it is a TRSDOS diskette). TRSDOS does not check for valid commands: if the command line contains an error, it will be detected the next time the System is started up.

You may install an automatic command on a diskette in a drive other than Drive 0. However, an AUTO diskette must contain an operating system and must be used in Drive 0. (An AUTO command written on a data diskette will not be used during Reset or Power-up).

## Sample

```
AUTO DIR (SYS)
```

The Screen will display `AUTO = 'DIR (SYS)'` as an acknowledgement. The next time the computer is Reset or Powered-up, the message `--AUTO FUNCTION ENGAGED--` will be displayed followed by the System Directory.

## Example

```
AUTO BASIC
```

Tells TRSDOS to load and execute BASIC each time the System is started up.

```
AUTO PAYROLL/CMD
```

Tells TRSDOS to load and execute PAYROLL/CMD (must be a machine-language program) after each System start-up.

```
AUTO DO STARTER
```

Tells TRSDOS to take automatic key-ins from the file named STARTER after each System start-up. See BUILD and DO.

```
AUTO DIR, FREE, LIB
```

TRSDOS executes DIR only, FREE and LIB are ignored.

## To Erase an AUTO Command

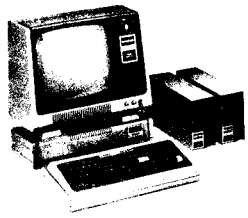
Type: `AUTO` (ENTER)

This tells TRSDOS to erase any automatic command. The command will be deleted the next time you power-up or RESET the System.

The acknowledgement: `AUTO = ''` is displayed after an erasure.

## To Override an AUTO Command

You can bypass any automatic command by holding down any key except (BREAK) while pressing RESET. You must continue holding down the key until you are prompted for the date or `TRSDOS READY` is displayed during the initialization process.



for the date or TRSDOS READY is displayed during the initialization process.

# BACKUP
# Create an Exact Copy of an Original Disk

**BACKUP *:source:destination***

***:source* specifies the drive containing the original diskette. If omitted, TRSDOS will prompt you for this information.**

***:destination* specifies the drive containing the diskette to receive the copy. If omitted, TRSDOS will prompt for it.**

***:source* and *:destination* may reference the same drive.**

BACKUP copies the contents of the *source* diskette to the *destination* diskette. BACKUP will make a "mirror" image of a diskette. This gives you a "safe" copy of the diskette. Always keep an extra copy of data or programs you have stored on your diskettes.

**Note:** The destination diskette must not be write-protected. Placing a write protect tab on the source diskette is recommended until a successful BACKUP is made.

TRSDOS will prompt you at each step after you type: BACKUP (ENTER)

If you omitted the source/destination-drive numbers, TRSDOS will begin with the prompt:

SOURCE DRIVE NUMBER?

Type in the number of the drive that contains the source diskette and press (ENTER).

DESTINATION DRIVE NUMBER?

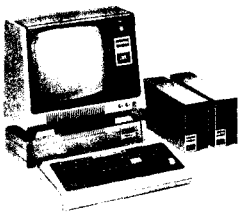
Type in the number of the drive that will contain the destination diskette and press (ENTER).

SOURCE DISK MASTER PASSWORD?

Type in the password assigned to your source diskette. Then press (ENTER). The TRSDOS default password is PASSWORD. If this password is assigned to your source disk, you may press (ENTER) to this prompt.

If you are doing a BACKUP with more than one drive, TRSDOS will ask:

QUICK OR FULL VERIFY?





Type Q (Quick) or F (Full) and press (ENTER). If this optional response is not answered, Q is used. If you select full, TRSDOS will do a byte to byte comparison of all allocated portions of the diskette (those with information stored on them). If you select quick this comparison will not be done.

TRSDOS will now display the message: ANALYZING DISKETTE. During this time TRSDOS checks to see if the diskette contains any files, if the diskette is already formatted, and if the source diskette is single or double-density.

It doesn't matter what the density of the destination diskette is. If the source diskette is double-density and destination diskette is single, BACKUP will re-format the destination diskette to double-density.

**Important Note:** BACKUP cannot copy from double-density to single-density or single-density to double-density.

To move files or programs from a single-density diskette to a double-density diskette (or vice versa) use COPY.

If the diskette contains files, TRSDOS will display:

```
DISK CONTAINS DATA, USE DISK OR NOT?
```

Type in Y (Yes) or N (No).

If the diskette was already formatted, the following will be displayed:

```
DO YOU WISH TO RE-FORMAT THE DISK?
```

Type in Y (Yes) or N (No).

If you specified the source/destination drives, TRSDOS will request the PASSWORD, skipping the first two steps.

TRSDOS will then take charge of formatting and verifying the destination disk as well as letting you know if there are any errors or flawed tracks.
If the source-drive and destination-drive are the same, TRSDOS will display INSERT SOURCE DISK or to INSERT DESTINATION DISK when necessary. Wait until the drive motor is stopped before doing so.

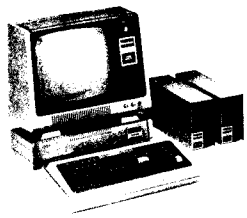
After a single-drive BACKUP on Drive 0, TRSDOS will display:

```
** BACKUP COMPLETE **
INSERT SYSTEM DISKETTE <ENTER>
```

Be sure there is a TRSDOS diskette in Drive 0 and press (ENTER). You will then return to TRSDOS READY.

Be careful when making BACKUPS on drives that are CONFIGured differently. There will be abnormal results if a BACKUP is attempted from a 40 track diskette to a 35 track diskette.

(See CONFIG for more information.)



# BLINK
## Turn Blinking Cursor ON and OFF

**BLINK *(switch)***

***switch* is ON or OFF. *switch* is optional; if omitted TRSDOS uses ON.**

This command allows you to turn off the blinking cursor if you prefer a steady non-blinking cursor. It also allows you to turn the blinking cursor back on.

### Example

```
BLINK (OFF)
```

This will turn off the blinking effect of the cursor.

```
BLINK (ON)
BLINK
```

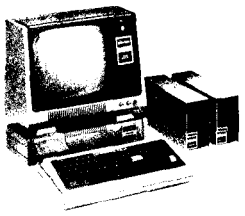
This returns the cursor to its original blinking mode.

# BUILD
## Create an Automatic Command Input File

**BUILD *filespec***

***filespec* is a file specification which can include an extension. If an extension does not exist, BUILD adds its own extension ( /BLD).**

This command lets you create an automatic command input file which can be executed via the DO command. The file must contain data that would normally be typed in from the keyboard to the TRSDOS READY mode.





BUILD files are intended for passing command lines to TRSDOS just as if they'd been typed in at the TRSDOS READY level. **Note:** CLEAR *cannot* be used in a DO file.

When you enter the BUILD command, BUILD creates the file and immediately prompts you to begin inserting lines. Each time you complete a line, press (ENTER). (While typing in a line, you can use the usual cursor control keys for erasures and corrections.)

To end the BUILD file, simply press (BREAK) at the beginning of a line.

First type: BUILD *filespec*

## Example

First type: BUILD STARTUP (ENTER)

You will then be prompted to type in the command text. You may type in up to 63 characters. Press (ENTER) after each command line:

```
MASTER (DRIVE=1) (ENTER)
WP (DRIVE=Ø) (ENTER)
DIR (ENTER)
(BREAK)
```

This builds the DO file called STARTUP. To run, type DO STARTUP (ENTER).

# CLEAR
# Clear User Memory

**CLEAR (START = *aaaa*,END = *bbbb*,MEM = *cccc*)**

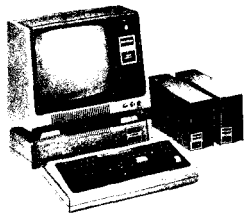
**START = *aaaa* tells TRSDOS where to start clearing user memory. *aaaa* is a four-digit hexadecimal number from 7000 to the end of user memory. If this option is omitted, 7000 is used. If this option is used, END = *bbbb* must also be used.**

**END = *bbbb* tells TRSDOS to clear user memory to a specified end. *bbbb* is a four-digit hexadecimal number no less than the START number and no greater than the top of memory. If this option is used, START = *aaaa* must also be used.**

**MEM = *cccc* sets the memory protect address. *cccc* is a four-digit hexadecimal number from 0000 to FFFF. If this option is omitted, the memory protect address is reset to end of user RAM.**

**If all options are omitted, all available RAM is cleared, memory-protect is reset to end of memory, the Display is cleared.**

**Note: START and MEM cannot be set below X'7000'. END and MEM cannot be set above X'FCFF'.**



This command gets you off to a fresh start.

Depending on the options you select, this command will:

- Zero user memory (load binary zero into each memory address above 7000H)
- Clear the Display
- Un-protect all memory

**Note:** CLEAR *cannot* be used in a DO file.

## Example

```
CLEAR (START=9000,END=0A000)
```

**Note:** Hexadecimal numbers which begin with a letter must be prefaced by zero (see above example).

```
CLEAR (MEM=7000)
```

# CLOCK
# Turn On Clock Display

**CLOCK (*switch*)**

***switch* gives TRSDOS one of two options, ON or OFF. If *switch* is omitted, TRSDOS uses ON.**

This command controls the real-time clock display in the upper right corner of the Video Display. When it is on, the 24-hour time will be displayed and updated once each second, regardless of what program is executing.

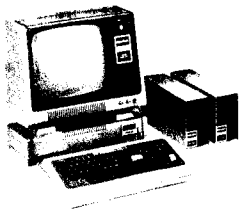
Clock display is OFF at TRSDOS start-up.

**Note:** Except during cassette and disk I/O, the real-time clock is always running, *regardless* of whether the clock display is on.

## Examples

```
CLOCK
```

Turns on the clock display.





CLOCK (OFF)
Turns the clock display off.

See TIME and DATE.

# CLS
# Clear the Screen

**CLS**

This command clears the Display and puts it in the 64 character/line mode.

## Example

```
CLS
```

# CONFIG
# Change Stepping Rate and Specify Number of Tracks

**CONFIG *:d* (STEP = *x*, TRACK = *y*)**

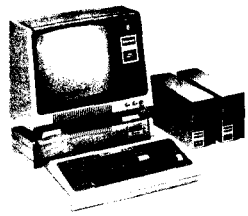
***:d* specifies the drive that is to be configured. The drive number must be specified.**

**TRACK = *y* specifies the number of tracks for that drive where *y* = 35 or 40. TRACK is optional; if omitted, 35 is used.**

**STEP = *x* specifies the stepping rate of that drive where *x* =:**

**0 = 6 MS**
**1 = 12 MS**
**2 = 20 MS**
**3 = 30 MS**

**STEP is optional; if omitted, 30 is used.**
**TRSDOS starts up at the slowest rate 30 MS and is set for 35 tracks.**



This command allows you to change the configuration of a disk drive by changing the stepping rate (speed of accessing a file) for that drive and specifying the number of tracks for that drive.

There are two different models of drives for the TRS-80 Model I. (This should not be confused with the two different catalog numbers. This command applies to both 26-1160 and 26-1161.) These drives can be distinguished by the serial numbers. For ease of explanation, the drives shall be referred to as SUFFIXED and NON-SUFFIXED.

SUFFIXED — any drive whose serial number (on bottom of drive) has a suffix of "-1"

NON-SUFFIXED — any drive whose serial number does not have a suffix.

The stepping rate can only be changed on SUFFIXED drives. Do not change the stepping rate on NON-SUFFIXED drives or drives without serial numbers.

CONFIG is not a permanent change. If the computer is turned OFF or reset, TRSDOS will start up at 30 MS for the stepping rate and 35 for the number of tracks.

**Note:** If you should configure your drive to a rate other than 30 MS and receive excessive I/O errors or can't read the directory, then your drive probably cannot run at any rate other than 30 MS.

CONFIG also allows the number of tracks to be changed. Only the SUFFIXED drives can, however, be operated at 40 tracks per diskette. NON-SUFFIXED drives should be operated at 35 tracks per diskette.

**Note:** If you do not know which drive you have, don't take a chance — don't change the stepping rate or the number of tracks per disk. Be sure to make a backup of your TRSDOS diskette before trying to make any changes.

If any problem occurs, reset the Computer and try again. If the problem still exists, your drive probably cannot be configured to that stepping rate or number of tracks. Try using a different value or don't use CONFIG at all (stepping rate and number of tracks will stay at their default values).

FORMAT and BACKUP will format diskettes according to the configuration of the drive. Be extremely careful when mixing drive configurations (tracks especially). This could cause abnormal results. (i.e., if a BACKUP is attempted from a 40 track diskette/drive to a 35 track diskette/drive.)

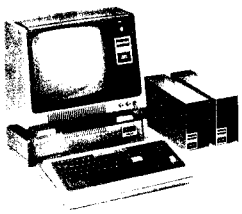
## Example

```
CONFIG :1 (STEP=2)
```

This sets Drive 1 at a stepping rate of 20 MS.

```
CONFIG :2 (STEP=1, TRACK=40)
```

This sets Drive 2 at a stepping rate of 12 MS and the number of tracks to 40.



# TRS-80 MODEL I DISK SYSTEM

## Permanent Configuration Change

This PATCH should be applied to Working Copy of System Diskette, not the Master Copy.

**PATCH *0:*d* (R=2, B=*bbb*, F=*ff*, C=*cc*)**

***d*=Drive # of Working Copy**

***bbb*=Byte Offset (See Table Below)**

***ff*=Find Byte (See Table Below)**

***cc*=Change Byte to (See Table Below)**

| DRIVE # | STEPPING RATE | # OF TRACKS |
|---|---|---|
| 0 | 110 | 114 |
| 1 | 111 | 115 |
| 2 | 112 | 116 |
| 3 | 113 | 117 |

To find *bbb*, cross-index the drive that is to be changed.

Stepping rate can be one of four values:

| STEPPING RATE | | BYTE VALUE | |
|---|---|---|---|
| 6 MS | = | 00 | |
| 12 MS | = | 01 | |
| 20 MS | = | 02 | { This value is set on original diskette |
| 30 MS | = | 03 | |

# of tracks can be either of two values:

| # OF TRACKS | | BYTE VALUE | |
|---|---|---|---|
| 35 | = | 23 | { This value set on original diskette |
| 40 | = | 28 | |

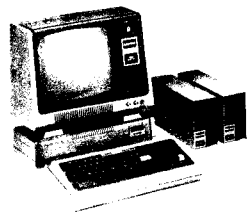
## Example

To change Stepping Rate on Drive #2 from 30 MS to 12 MS and Working Copy is on Drive #1.

```
PATCH *0:1 (R=2, B=112, F=03, C=01)
```

To change # of Tracks on Drive #1 from 35 tracks to 40 tracks and Working Copy is on Drive #2.

```
PATCH *0:2 (R=2, B=115, F=23, C=28)
```



# COPY
## Copy a File or Files

**Three syntaxes:**

**A)** COPY ***source-file:d destination-file:d***

***source-file* is a file specification of the file to be copied.**

***destination-file* is a file specification for the name and drive of the duplicate file.**

***:d* is the drive number where that file is found.**

**B)** COPY ***source-file:d :d***

***source is defined above.***

***:d* tells TRSDOS to copy the file onto drive *d*, using the same filename.**

**C)** COPY ***/ext :d :d***

***/ext* is a wild-card file specification in which the file name is omitted and the extension is given. TRSDOS will copy all files except password protected files which have a matching extension, regardless of the filename.**

**Note: Drive numbers are required except on syntax A. For syntax A, *destination-file* defaults to Drive 0.**

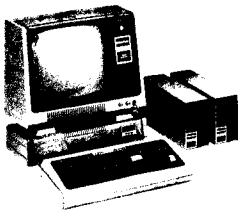
This command copies *source-file* into the new file defined by *destination-file*. This allows you to copy a file from one disk to another, using a single-drive if necessary. In the latter case, you must specify drive number in both file specifications. You will be prompted when to swap diskettes.

COPY also allows you to copy files from a single-density diskette to a double-density diskette and vice versa. TRSDOS examines each diskette (source and destination), determines the density of each and does the COPY automatically. For example, if you have a double-density TRSDOS diskette in Drive 0, and a single-density diskette in Drive 1, then the command:

```
COPY DATAFIL/TXT:1 DATAFIL/TXT:0
```

would copy the file DATAFIL/TXT from the single-density diskette to the double-density diskette. The command:

```
COPY DATAFIL/TXT:0 DATAFIL/TXT:1
```





would also work. It doesn't matter whether a single or double-density diskette is in a drive, TRSDOS will know.

**Important Note:** Wild card COPY (syntax C) does not allow copying to the same disk.

### Example

```
COPY OLDFILE/BAS:0 NEWFILE/BAS:1
```

Copies OLDFILE/BAS on Drive 0 into a new file named NEWFILE/BAS on Drive 1.

```
COPY NAMEFILE/TXT:2 :1
```

This command specifies a file named NAMEFILE/TXT on Drive 2 to be copied to Drive 1 using the same filename.

```
COPY FILE/EXT:0 :1
```

This command copies FILE/EXT from Drive 0 to Drive 1.

```
COPY /BAS:0 :1
```

tells TRSDOS to copy all Drive 0 files which have the extension /BAS. The files will be copied onto Drive 1, using their present file names and extensions.

**Note:** Whenever a file is updated, use COPY to make a backup file on another diskette. You can also use COPY to restructure a file for faster access. Be sure the destination diskette is already less segmented than the source diskette; otherwise, the new file could be more segmented than the old one. (See FREE for information on file segmentation.)

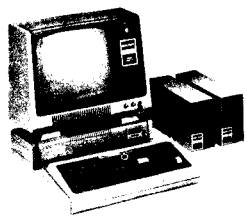
To rename a file on the same diskette, use RENAME, not COPY.

# CREATE
# Create a Pre-allocated File

**CREATE *filespec* (LRL = *aaa*,REC = *bbb*)**

**LRL = *aaa* is the logical record length. *aaa* is a decimal number between zero and 255. If omitted, 256 is assumed.**

**REC = *bbb* is the number of records desired. If omitted, no records are allocated.**



This command lets you create a file and pre-allocate (set aside) space for its future contents. This is different from the default (normal) TRSDOS procedure in which space is allocated to a file dynamically, i.e., as necessary when data is written into the file.

If you open the file for sequential writes and write data to it, TRSDOS will de-allocate (recover) any unused granules when the file is closed. If you open the file for random access, TRSDOS will not de-allocate space when the file is closed.

You may want to use CREATE to prepare a file which will contain a known amount of data. This will usually speed up file write operations. File reading will also be faster, since pre-allocated files are less segmented or dispersed on the disk — requiring less motion of the read/write mechanism to locate the records.

If you create a file and the last record is on a granule boundary, an extra granule may be allocated to the file.

## Examples

```
CREATE DATAFILE/BAS (REC=300, LRL=0)
```

Creates a file named DATAFILE/BAS, and allocates space for 300 256-byte records.

```
CREATE NAMES/TXT,IRIS (LRL=64,REC=50)
```

Creates a file named NAMES/TXT protected by the password IRIS. The file will be large enough to contain 50 records, each 64 bytes long.

```
CREATE PAYROLL/BAS
```

Creates a file named PAYROLL/BAS but allocates no space to it.

## Sample Use

Suppose you are going to store personnel information on no more than 250 employees, and each data record will look like this:

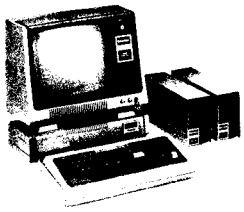
Name (Up to 25 letters)
Social Security Number (11 characters)
Job Description (Up to 92 characters)

Then your records would need to be 25 + 11 + 92 = 128 bytes long.

You could create an appropriate file with this command:

```
CREATE PERSONNL/TXT (REC=250,LRL=128)
```

Once created, this pre-allocated file would allow faster writing than would a dynamically allocated file, since TRSDOS would not have to stop writing periodically to allocate more space (unless you exceed the pre-allocated amount).





# DATE
## Reset or Get Today's Date

**DATE *mm/dd/yy***

***mm/dd/yy* is the specification for the month (*mm*), day (*dd*) and year (*yy*).**

**Each must be a two-digit decimal number between the following ranges:**

| | |
|---|---|
| ***mm*** | **01-12** |
| ***dd*** | **00-31** |
| ***yy*** | **00-99** |

**The specifications are an option; however, if one specification is used, they all must be used.**

**If *mm/dd/yy* is omitted, TRSDOS displays the current date.**

**If *mm/dd/yy* is given, TRSDOS resets the date.**

This command lets you reset the date or display the date.

When you request the date, TRSDOS displays it in the format: `07/25/80` for July 25, 1980.

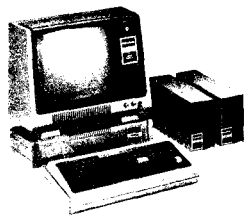
## Examples

`DATE`

Displays the current date.

`DATE 07/18/80`

Resets the date to July 18, 1980.



# DEBUG
## Start Debug Monitor

**DEBUG (*switch*)**

***switch* gives TRSDOS one of two options, ON or OFF. If option is omitted, TRSDOS uses ON.**

This command starts the debug monitor, which allows you to enter, test, and debug machine-language programs. (**Note:** BASIC programs cannot be debugged since portions of BASIC and DEBUG occupy the same memory locations.)

Its features include:

- Full- or half-screen displays of memory contents
- Commands for modifications to RAM and register contents
- Single-step execution of programs
- Breakpoint interruption of program execution
- Transfer of control (Jump)
- "Editing" of disk-files

DEBUG uses the memory area from X'4E00' to X'53FF' (see **TRSDOS Memory Map**). DEBUG can only be used on programs in the user area X'5400' to TOP.

Type: DEBUG (ENTER)

This turns DEBUG ON. The command stays in the TRSDOS READY mode but the DEBUG program is set to execute under any of the following conditions:

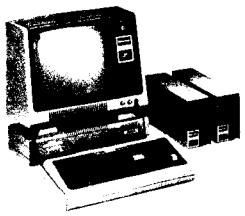
- When (BREAK) is pressed
- After a program is loaded and before its first instruction is executed.
- Upon detection of an error.

Press: (Q)

to exit DEBUG and turn off the "debugger."

## Command Description

Debug commands are usually entered by pressing a single key. In most cases, you do not have to press (ENTER) after the command has been typed. Either a prompt will immediately be displayed or DEBUG will execute the operation without further instruction.





In some cases, you will have to enter a specific hexadecimal value or address (see R and J commands, for instance). Instead of pressing (ENTER) after the address is typed in, you will have to press (SPACEBAR).

Once you have entered the DEBUG program, you may use any of the following special commands:

## D (Display Memory Contents)

Press (D) to display the contents of memory. TRSDOS will respond with the prompt: `D ADDRESS =` You should type in the hexadecimal address of the memory location you wish to see.

The display will be either half- or full-screen, depending on the format you are currently using (see below).

**Note:** If memory address displayed on the Screen is between 0 and 5400, then the D command must be used to move the display into the user area. This will allow the use of (−) and (;).

## X (Half-Screen Display)

Press (X) to put the Display in the half-screen format. A 128-byte block of memory will be displayed starting with the next lowest address which is a factor of 16.

**Figure 4** shows a typical half-screen format.

## S (Full-Screen Display)

Press (S) to display the contents of a 256-byte block of memory starting with the next lowest address which is a factor of 256.

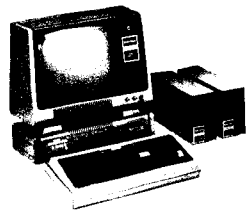
**Note:** The last 16 bytes on the Display will be overlaid by any command line typed in after the full-screen display is updated.

## M (Modify RAM)

Press (M) to change to the disk utility display format (see the F command). TRSDOS will respond with the prompt: `M ADDRESS =` You should type in the four-digit hexadecimal address of the memory location you wish to modify, followed by a blank space (anything other than a space will abort the command).

The display will change to the memory edit format. The cursor will appear as a blinking character at the specified location.

To exit the modify mode, press (ENTER) to keep all changes made.



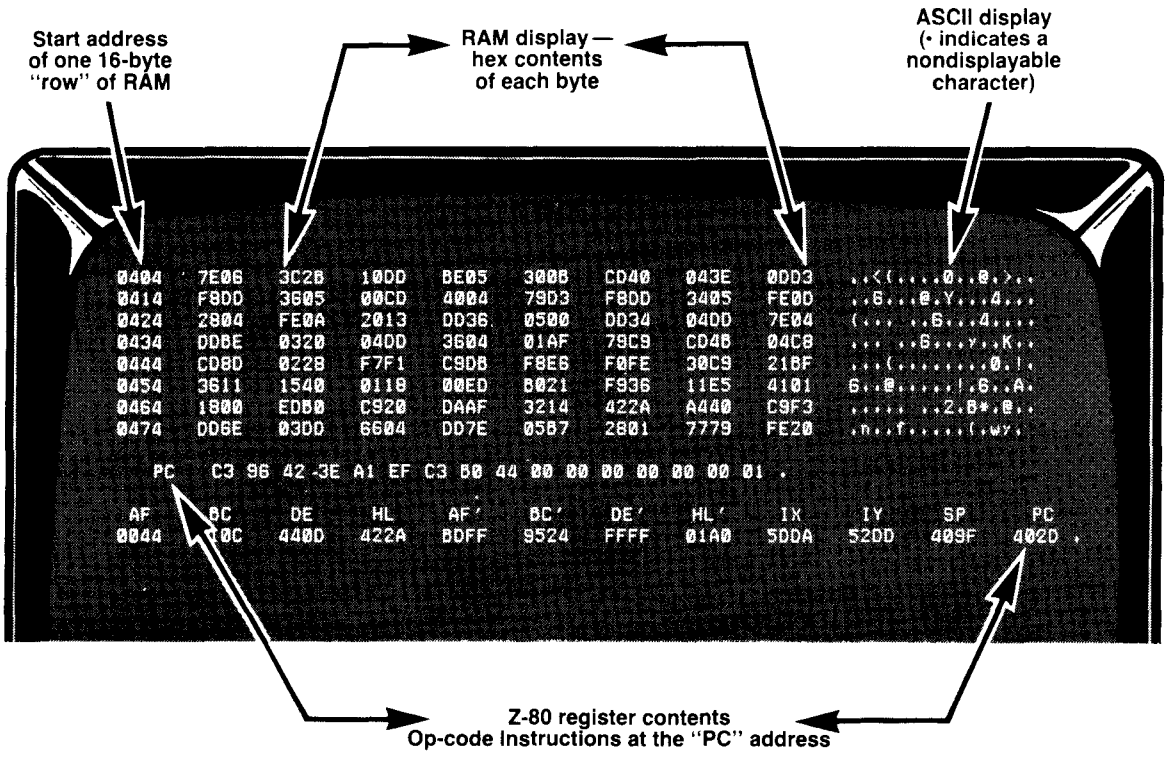


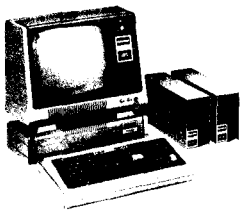

**Figure 4.** Half-Screen Format.

## R (Change Register Contents)

**R*aa,bbbb*** (SPACEBAR)

***aa*** **is the name of one of the register pairs AF, BC, DE, HC, or PC.**

***bbbb*** **is the four-digit hexadecimal value which will be loaded into *aa*. If fewer than four digits are typed in before pressing (SPACEBAR), leading zeros are assumed.**



# TRS-80 MODEL I DISK SYSTEM

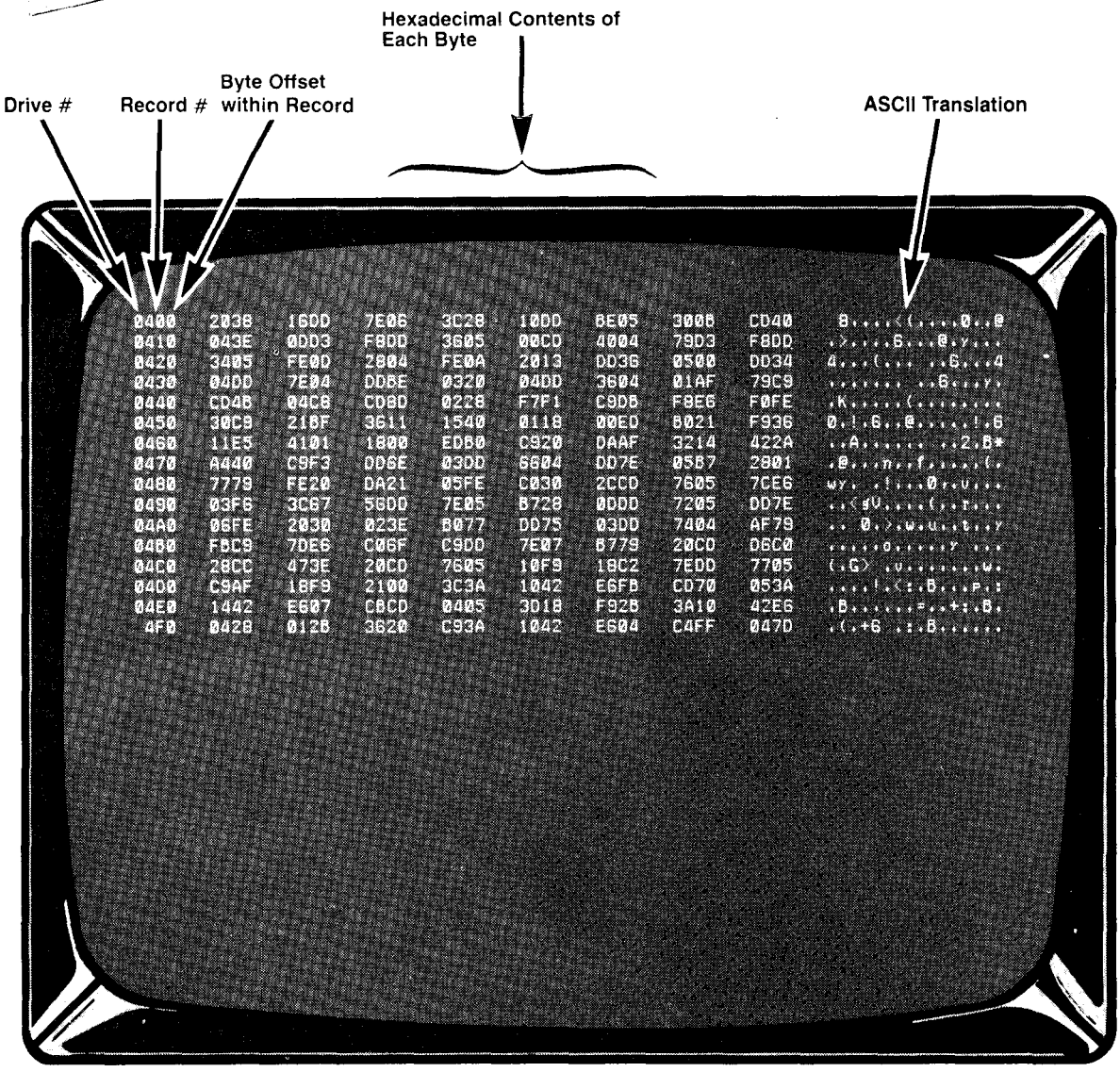
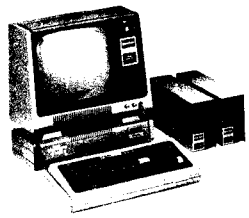



**Figure 5.** Full-Screen Format



## I (Instruction Single-Step)

Pressing (I) will allow the Computer to execute a single Z-80 instruction. The display will then be updated.

The instruction in the memory contents referenced by the program counter is executed. The program counter is increased by the appropriate value, and the control is returned to DEBUG.

DEBUG will not, however, step through a call or jump into a ROM address.

## C (Call Single-Step)

If you wish to complete an entire call/return sequence, press (C). The call is then executed and control is returned to DEBUG when the subroutine returns. Otherwise, this instruction acts just like the I command.

You will not be able to step through a call or jump into a ROM address.

## ; (Increment Display Address)

If the Display is half-screen, the first location shown is incremented by 16 when you press (;). If the full-screen format is displayed, the starting address will be incremented by 256.

## – (Decrement Display Address)

If the Display is half-screen, the first location is decremented by 16 when you press (–). If the full-screen format is displayed, the starting address will be decremented by 256.

## J (Jump)

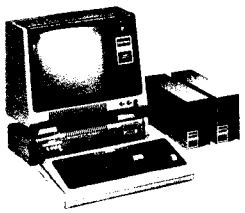
Press (J) to transfer control to a machine-language program, setting optional breakpoints.

Debug will respond with the prompt: `J ADDRESS? =`

You may type in a jump address and a breakpoint address. The command is terminated when you press (ENTER). Type in the addresses in one of three formats:

**J ADDRESS? = *aaaa,bbbb* (ENTER)**
**J ADDRESS? = *aaaa* (ENTER)**
**J ADDRESS? = *,bbbb* (ENTER)**

***aaaa* is a four-digit hexadecimal address specifying the jump destination. If omitted, the address in the PC register is used.**





***bbbb* is a four-digit hexadecimal address specifying a breakpoint. Before the Computer executes an instruction at this address, it will return control to DEBUG. If this address is omitted, control will not return to DEBUG.**

**Notes:** Breakpoints must be set at the *beginning* of Z-80 instructions. You may not set breakpoints in ROM addresses. The breakpointed address will contain an X'F7' until the breakpoint is encountered. Then the original contents will be restored and DEBUG will take control again.

## Q (Quit)

Pressing (Q) turns off DEBUG and returns control to TRSDOS.

# DIR
# List the Diskette Directory

**DIR :*d* (INV,SYS,PRT)**

**:*d* is the desired drive directory. If omitted, Drive 0 is assumed.**

**I or INV lists the invisible user files. If omitted, non-invisible user files are listed.**

**S or SYS lists system and user files. If omitted, only non-invisible user files are listed.**

**P or PRT lists the directory to the Printer. If omitted, the directory will be listed on the Video Display only.**

**If option is not given, TRSDOS lists non-invisible user files in Drive 0.**

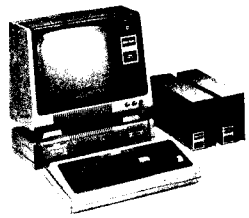
This command gives you information about a disk and the files it contains. It will work on both single and double-density diskettes.

To pause the listing, press (@). To continue, press (ENTER). To terminate the listing, press (BREAK). When the screen becomes full, DIR will automatically pause. To continue, press (ENTER).

## Examples

```
DIR
```

Displays the directory of non-invisible user files in Drive 0.



```
DIR :1 (PRT)
```

Lists the directory of the user files in Drive 1 to the Printer.

## Sample Directory Listing

(See **Figure 6**.)

**Definition of column headings**

① File Name — The name and extension assigned to a file when it was created. The password (if any) is not shown.

② Attributes — A four-character field.

The first character is either I (Invisible) or N (Non-invisible).

The second character is S (System) or * (User) file.

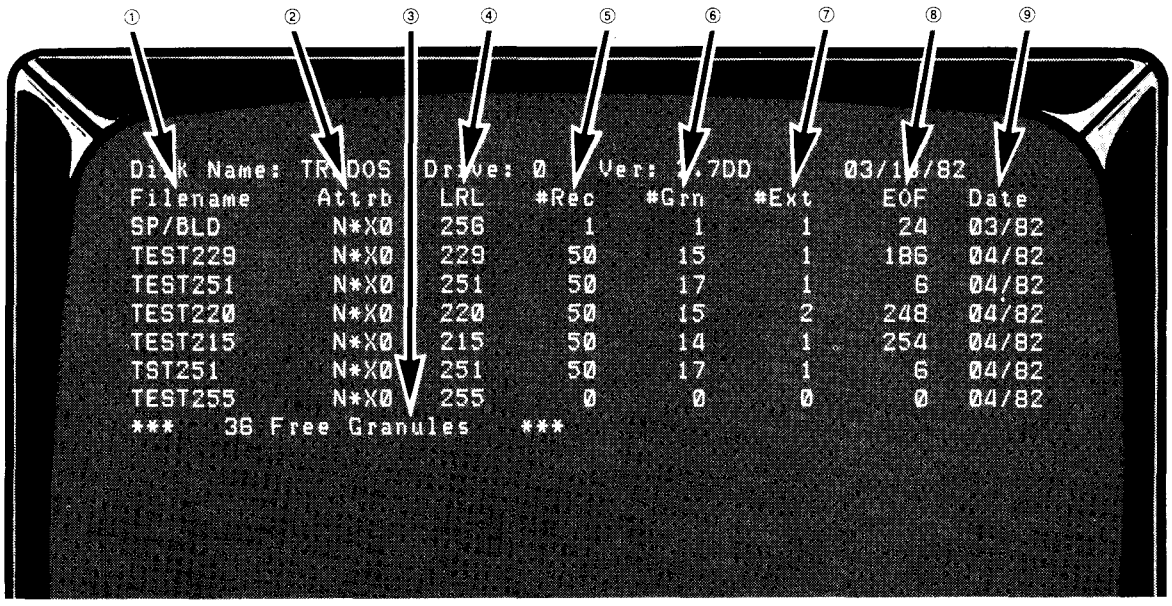
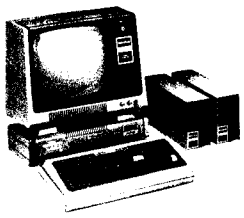



**Figure 6.** Directory Listing.





The third character gives the password protection status:

X The file is unprotected (no password).

A The file has an access word but no update word.

U The file has an update word but no access word.

B The file has both update and access words.

The fourth character specifies the level of access assigned to the access word:

0 Total access

1 Kill file and everything listed below.

2 Rename file and everything listed below.

3 This designation is not used.

4 Write and everything listed below.

5 Read and everything listed below.

6 Execute only.

7 No access.

③ Number of Free Granules — How many free granules remain on the diskette.

④ Logical Record Length — Assigned when the file was created.

⑤ Number of Records — How many logical records have been written.

⑥ Number of Granules — How many granules have been used in that particular file.

⑦ Number of Extents — How many segments (contiguous blocks of up to 32 granules) of disk space are allocated to the file.

⑧ End of File (EOF) — Shows the last byte number of the file.

⑨ Creation Date — When the file was created. This column does not appear on the directory of single-density diskettes.

# DO
# Begin Auto Command Input from a BUILD-File

**DO *filespec***

***filespec* is the name of file created with BUILD. If an extension is not included, the file will automatically be given the extension /BLD.**

This command reads and executes the lines stored in a special-format file created with the BUILD command. The System executes the commands just as if they had been typed in from the Keyboard.

Command lines in a BUILD file may include library commands or file specifications for user programs.

When DO reaches the end of the automatic command input file, it returns control to TRSDOS.

The DEBUG, DO and CLEAR command *cannot* be included in a BUILD file.

In addition to executing TRSDOS library commands, you can load and execute user programs from a DO-file. You will probably want to make your program name be the last line in the DO-file.

## Examples

```
DO STARTER
```

TRSDOS will begin automatic command input from STARTER, after the operator answers the Date and Time prompts.

```
AUTO DO STARTER
```

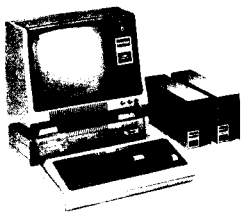
Whenever you start TRSDOS, it will begin automatic command input from STARTER.

## Sample Uses

Suppose you want to set up the following TRSDOS functions automatically on start-up:

```
CLOCK (ON)
WP (DRIVE=0)
DUAL (ON)
```

Then use BUILD to create such a file. If you called it BEGIN, then use the command: `AUTO DO BEGIN` to perform the commands each time TRSDOS starts up.





# DUAL
# Duplicate Output to Video and Printer

**DUAL (*switch*)**

***switch* is one of two options, ON or OFF. If *switch* is omitted, TRSDOS uses OFF.**

This command duplicates all video output to the Printer, and vice versa. It takes effect immediately.

**Notes:**

- Video and printer output may be different because of intrinsic differences between output devices and output software.
- Using the DUAL command will slow down the video output process.
- The printer should be ready when you execute the command.

## Example

For a printed copy of all system/operator dialog, type: `DUAL (ON)` (ENTER)

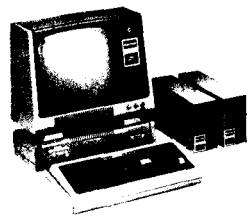
To turn off the DUAL process, type: `DUAL (OFF)` (ENTER)

# DUMP
# Store a Machine Language Program Into a Disk File

**DUMP *file* (START = *aaaa*,END = *bbbb*,TRA = *cccc*,RELO = *dddd*)**

***file* is the file specification**

**START = *aaaa* is the start address of memory block. *aaaa* must be a four-digit hexadecimal number greater than or equal to X'6300.'**



**END = *bbbb* is the end address of the memory block. *bbbb* must be a four-digit hexadecimal number and be greater than start.**

**TRA = *cccc* is the transfer address where execution starts when the program is loaded. *cccc* must be a four-digit hexadecimal number. If this option is omitted, the command will default to TRSDOS re-entry.**

**RELO = *dddd* is the start address for relocating or loading the program back into memory. *dddd* must be a four-digit hexadecimal number. If this option is omitted, no relocation will take place.**

**Note: Addresses must be in hexadecimal form, without the X' ' notation. You must add the prefix "0" to any hex number which begins with a letter.**

This command copies a machine-language program from memory into a program file. You can then load and execute the program at any time by entering the file name in the TRSDOS READY mode.

## Examples

```
DUMP LISTER (START=7000,END=7100,TRA=7004)
```

Creates a program file named LISTER/CMD containing the program in memory locations X'7000' to X'7100'. When loaded, LISTER/CMD will occupy the same addresses, and TRSDOS will protect memory beginning at X'7000'. The program is executable for the TRSDOS READY mode.

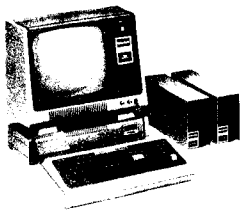
```
DUMP PROG2 (START=7000,END=7F00,TRA=8010,RELO=8000)
```

Creates a program file named PROG2/CMD containing the program in addresses X'7000' to X'7F00'. When loaded, PROG2/CMD will reside from X'8000' to X'8F00'. Execution will start at X'8010'.

# ERASE
# Erase File From Diskette

**ERASE *filename***

***filename* is a TRSDOS file specification.**





This command is similar to the KILL command in that it deletes a file from the Directory, but it also clears (changes the bytes to zeros) the data area where the file was stored. Using ERASE instead of KILL will insure that your file may not be recovered.

**Note:** A file that has been ERASEd cannot be recovered by UNKILL, whereas a file that has been killed may be recovered.

### Example

```
ERASE TEST/BAS
```

Erases the filename TEST/BAS off the directory and clears that area of the diskette where the file was stored.

(Also see KILL and UNKILL)

# ERROR
# Display Error Message

**ERROR *number***

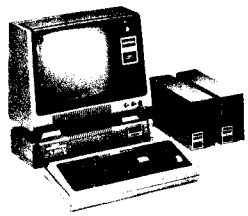
***number* is a decimal number for a TRSDOS error code.**

This command displays a descriptive error message. For example, after receiving the message, `* * ERROR 11 * *` you may respond with the command: `ERROR 11` (ENTER) and TRSDOS will display the full error message.

For a complete list of error codes and messages, see the **Technical Information** section of this manual.

# FILFIX
# Load and Modify Contents of File

**FILFIX**



This command lets you load and modify the contents of a diskette file.

Type: FILFIX (ENTER)

TRSDOS will prompt: FILESPEC?

Enter the name of the file to be patched.

FILFIX will set up a full-screen display showing the first 256 bytes in the file. You can "page" through the file using the (;) and (-) keys.

The display is similar to a full page display of DEBUG.

In this file-display mode, both hexadecimal and ASCII are given for each byte. If a code has no displayable character, a period is shown in the ASCII area.

The display control commands are like those for the normal file-display mode:

(;) Next page
(-) Previous page
(P) Output to printer

To change the file contents, press (M). This puts you in a modify-memory mode like the one previously described. Use the arrow keys to position the cursor, then type in the correct contents as a hexadecimal value. When you are through changing a page on the display, press (ENTER). The diskette file will be updated and you will be returned to the file-display mode.

To cancel changes made, do not press (ENTER), press (BREAK). This will put you back in the file-display mode without updating the diskette file. You may press (;) then (-) to restore the page display to its actual contents.

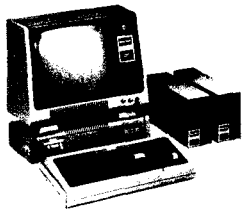
To quit patching a file, press (BREAK) while in the file display mode and you will be returned to TRSDOS READY.

# FORMAT
# Prepare a Data Diskette

**FORMAT :*d***

**:*d* specifies the disk drive which contains the diskette to be formatted. If :*d* is omitted, TRSDOS will prompt you for this information.**

This command lets you prepare data diskettes (either new or diskettes which contain undesired data or programs), leaving a maximum amount of space for your program and data files.





**Note:** Data diskettes may only be used in Drives 1, 2, and 3 except during a BACKUP.

FORMAT takes a blank (new or magnetically erased) diskette, records track/sector boundaries on it, then initializes it and creates a directory.

## Example

Type:

```
FORMAT (ENTER)
```

to execute the FORMAT utility. If you did not specify a drive in the command line, TRSDOS will display the message:

```
FORMAT WHICH DRIVE?
```

Type the drive number that contains the diskette to be formatted and press (ENTER).

```
DISKETTE NAME?
```

The name serves as an internal label for the diskette. TRSDOS will default to TRSDOS by simply pressing (ENTER). If another name is desired, type in any appropriate name of one to eight letters and numbers, beginning with a letter. Press (ENTER) at the end of that name.

```
MASTER PASSWORD?
```

The password may be from one to eight alphanumeric characters. The first character must be alphabetic and you must press (ENTER). Pressing (ENTER) without typing a password will default to PASSWORD.

TRSDOS will now display the message: `ANALYZING DISKETTE` During this time TRSDOS checks to see if the diskette contains any files and if the diskette is single or double density.

If the diskette is unformatted or formatted in single density, TRSDOS will automatically begin formatting. If the diskette contains data, you will be asked:

```
DISK CONTAINS DATA, USE DISK OR NOT?
```

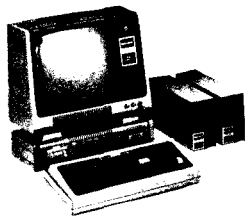
Type `Y` (Yes) and press (ENTER) if you do want to reformat, `N` (No) and press (ENTER) if you want to save the disk information. You will be returned to TRSDOS READY.

**Notes:** When formatting is complete TRSDOS READY will appear on the screen.

If CONFIG has been used to change the number of tracks on the diskette being formatted, it will be formatted to have the number of tracks specified in CONFIG.

FORMAT will lock out any defective tracks to prevent data being lost in these areas.

If you get READ errors during access, reformat the diskette.



# FREE
# Display Disk Allocation Map

**FREE :*d* (PRT)**

**:*d* is the drive specification. If omitted, Drive 0 is used.**

**(PRT) tells TRSDOS to send the map to the Printer.**

**If omitted, TRSDOS sends the map to the Video Display only.**

This command gives you a map of granule allocation on a double-density diskette. (A granule, 768 bytes, is the unit of space allocation.) It also shows the location of the directory and any flawed sectors.

When a diskette has been used extensively (file updates, files killed, extended, etc.), files often become segmented (dispersed or fragmented). This slows the access time, since the disk read/write mechanism must move back and forth across the diskette to read and write to a file.

FREE helps you determine just how segmented your disk files are. If you decide you'd like to re-organize a particular file to allow faster access, you can then COPY it onto a relatively "clean" diskette.

## Example

```
FREE
```

Displays a free space map of the diskette in Drive 0.

```
FREE (PRT)
```

Lists the free space for Drive 0 to the Printer.

```
FREE :1 (PRT)
```

Lists the Drive 1 map to the Printer.

## A Typical FREE Display

Four special symbols are used in the FREE map.

| | |
|---|---|
| • | Unused Granule |
| Direct | Directory Information |
| X | Allocated Granule |
| Flawed | Track Contains a Flawed Sector (Unusable) |

A typical free map display is shown in **Figure 7**.

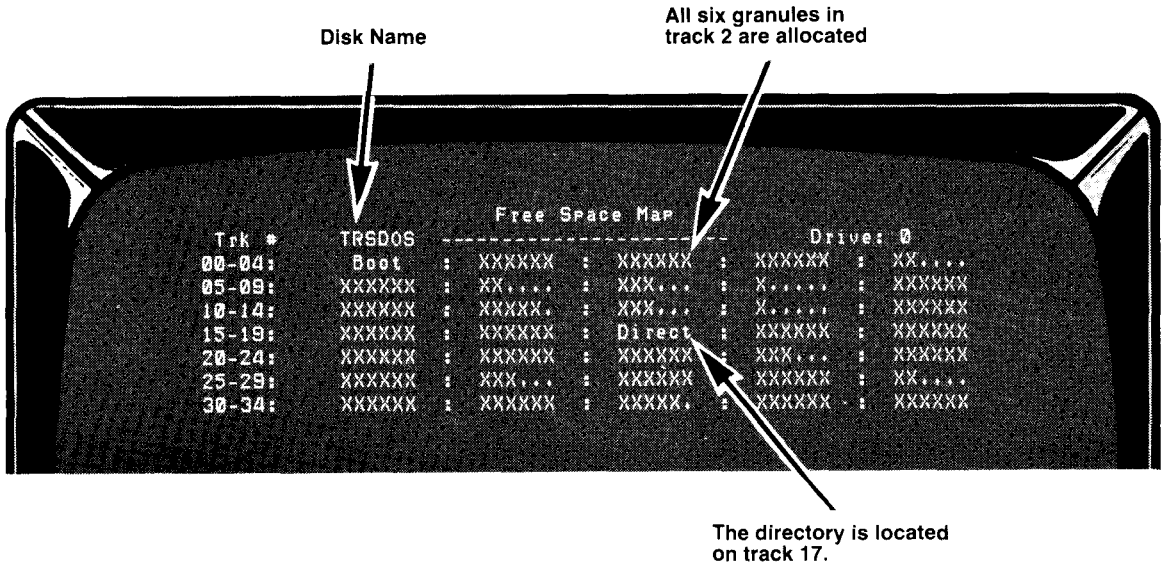
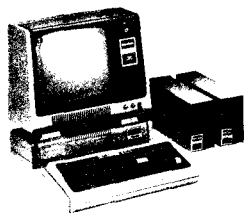



**Figure 7.** Free Map.

# HELP
# Explanation of TRSDOS Command

**HELP *command***

***command* is the specific TRSDOS command or subject on which you need help. If *command* is omitted or if an invalid subject is given, TRSDOS will list all available subjects.**



## Example

If you type in the following: HELP BACKUP (ENTER) TRSDOS will respond with the syntax format, a definition of the command, and an explanation of the abbreviation.

HELP COMMAND tells TRSDOS to display the syntax for the command.

If HELP * or HELP command * (ENTER) is used, the message will go to the printer instead of the display.

**Note:** Typing HELP (ENTER) or specifying a command that does not exist in the library will cause the HELP list to be displayed or printed.

# KILL
# Delete a File or Group of Files

**Two syntaxes:**

**A) KILL *filespec***

**B) KILL */ext:d***
***/ext* is a file extension that *must* contain three characters.**
***:d* is a drive specification. It *must* be provided.**

This command deletes one file or a group of files from the directory, depending on which form is used. Form A deletes the specified file. If no drive specification is given, TRSDOS deletes the file from the first diskette that contains it.

Form B deletes all files with a specified extension, regardless of the file name of each file. A drive number must be included.

**Note:** A file that has been KILLed may be recovered by using UNKILL.

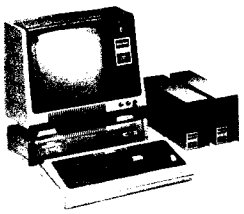
## Examples

```
KILL TESTPROG/BAS
```

Deletes the named file from the first drive that contains it.

```
KILL JOBFILE/IDY,PASSWORD:1
```

Deletes the named file from Drive 1. The file has a password of PASSWORD.





```
KILL /BAS:0
```

Deletes from Drive 0 all files having the extension /BAS.

(For more information, see ERASE.)

# LIB
# Display Library Commands

**LIB (PRT)**

**(PRT) tells TRSDOS to output to the Printer. PRT is optional; if omitted, output is to the Video Display.**

This command lists to the Display all the library commands. For help with a command, use HELP.

## Example

```
LIB
```

# LIST
# List Contents of a File

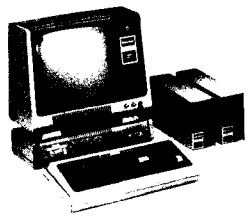
**LIST *filespec* (PRT,SLOW,ASCII)**

***filespec* is a TRSDOS file specification and must be specified.**

**PRT tells TRSDOS to list to the Printer. If omitted, only the Video Display is used.**

**SLOW tells TRSDOS to pause briefly after each line. If omitted, the listing is continuous.**

**ASCII tells TRSDOS to list the file in ASCII format. If omitted, hexadecimal format is used.**



This routine lists the contents of a file. The listing shows both the hexadecimal contents and the ASCII characters corresponding to each value. For values outside the range (X'20', X'7F'), a period is displayed.

Use the ASCII option for text files and BASIC programs saved with the A option.

**Note:** Only ASCII codes X'00'-X'7F' are sent to the Printer. Bit 7 is always set to 0.

During the listing, press @ to pause, ENTER to continue, or BREAK to exit.

## Examples

```
LIST DATA/TXT (ASCII)
```

Lists the contents of DATA/TXT in ASCII format.

```
LIST FILE/A (SLOW)
```

Lists the contents of FILE/A, pausing after each line.

```
LIST PROGRAM/CMD (PRT)
```

Lists the file PROGRAM/CMD to the Printer.

# LOAD
# Load a Program File

**LOAD *filespec***

***filespec* is a file specification that is created by the DUMP command.**

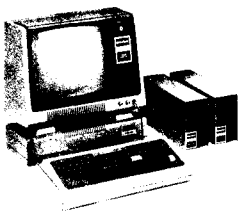
This command loads a machine-language program file into memory. After the file is loaded, TRSDOS returns to the TRSDOS READY mode.

You cannot use this command to load a BASIC program or any file created by BASIC. See the BASIC Reference Manual for instructions on loading BASIC programs.

**Note:** The file must load into RAM above X'5200. (X'5200-TOP).

## Examples

```
LOAD PAYROLL/PT1
```





## Sample Use

Often several program modules must be loaded into memory for use by a master program. For example, suppose PAYROLL/PT1 and PAYROLL/PT2 are modules, and MENU is the master program. Then you could use the commands:

```
LOAD PAYROLL/PT1
LOAD PAYROLL/PT2
```

to get modules into memory, and then type: MENU to load and execute MENU.

# LPC
# Line Printer Control

**LPC**

The LPC utility program allows TRSDOS to ignore multiple carriage return commands. Without LPC, a top-of-form (LPRINT CHR$(12)) command will add an extra carriage return/line feed each time it is executed. Also, LPC masks the high bit of each data byte, allowing you to send certain intercepted codes to the printer. For instance, the BASIC statement LPRINT CHR$(140) will send code 140-128 = 12 (LPRINT CHR$(12)) to the Printer.

The printers that require LPC are:

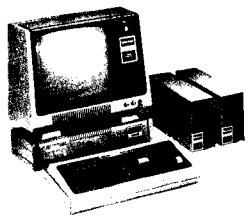
- Line Printer III (26-1156)
- Line Printer VI (26-1166)
- Daisy Wheel WP50 (26-1157)
- Qume Daisy Wheel (26-1157A)
- Daisy Wheel II (26-1158)

and all future printers.

Printers that do not require LPC:

26-1150, 1152, 1153, 1154, 1159, and the A version of LPIII (26-1156A).

You must load the LPC program before you load an application program. The easiest way to do this is to copy LPC onto your data/program diskette and then



use the AUTO command to load LPC automatically each time you use the system. For instance, type:

```
COPY LPC/CMD:1 :0 (ENTER)
```

Then, to make LPC an AUTO command on the diskette, type:

```
AUTO LPC/CMD (ENTER)
```

Whenever you use your program diskette, LPC will automatically load into memory and you can use the program as usual. Since LPC has the extension /CMD it is not necessary to type the extension when executing.

**Note:** LPC must be loaded before using SPOOL.

LPC locates into the highest available memory. There is no need to set MEMORY SIZE to protect LPC. ULC resets itself in high memory. However, you still need to set memory if required by your application program. LPC will be killed if the CLEAR command is used.

**Warning:** Once the LPC utility program is loaded and installed, you should not reload it except after a reset. Reloading re-installs the program and uses up more space each time!

# MASTER
# Set Master Read/Write Drive

**MASTER (DRIVE = *a*)**

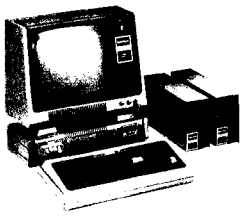
***a* is the drive specification. If omitted, Drive 0 is set as the master drive.**

This command allows you to assign a specified drive as the Master Read or Write drive in the system. When searching for a file, TRSDOS will start with the master drive.

If the file is not found on the specified drive, TRSDOS will continue searching on the next higher-numbered drive. If the file is still not found, TRSDOS will return to caller.

## Example

After you enter the command: `MASTER (DRIVE=1)` Drive 1 becomes the master drive.





# MEMTEST
## Test Memory

**MEMTEST**

This program tests your Model I's memory (read only and random access). In TRSDOS READY, just type MEMTEST and press (ENTER).

The program automatically tests all memory locations, no matter what memory size you have. First it checks read only memory; if everything is okay, it automatically goes on and checks random access memory. If all RAM checks out okay, the program continues. To return to TRSDOS, you must reset.

If the program detects a ROM or RAM error, it will display a detailed message. Repeat the test to make sure it is a valid error condition. Write the message down and contact your nearest Radio Shack for assistance.

**Note:** MEMTEST changes the entire contents of RAM. Before running it, be sure you have saved any valuable code you may have in RAM. Other checksums include ROM A (BO78) and ROM C (4006).

# PATCH
## Change the Contents of a Disk File

**Four syntaxes:**

**A)** **PATCH (ENTER)**

**This form will cause TRSDOS to prompt you for all entries described below.**

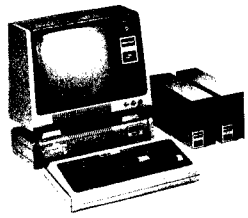
**B)** **PATCH *programfile:d***

**This form will cause TRSDOS to prompt you for all information except the filename.**

**C)** **PATCH *programfile* (A = *nnnn*,F = *cc*,C = *dd*)**

**This form is used for machine-language programs.**

***programfile* is the file specification of the program to be changed.**



**A = *nnnn* specifies the starting address at which the data is found. *nnnn* is a four-digit hexadecimal number.**

**F = *cc* specifies the string you wish to find (or compare). *cc* may be a hexadecimal sequence or a ASCII string enclosed in quotes.**

**C = *dd* specifies the new contents for the byte(s). *dd* may be either a hexadecimal sequence or an ASCII string enclosed in quotes. The change string must contain the same number of bytes as the find string.**

**D) PATCH *datafile (R = aa,B = bb,F = cc,C = dd)***

**This form is used for data files. (It can also be used for BASIC programs.)**

***datafile* is the file specification of the file to be changed.**

***R = aa* specifies the record which contains the data to be changed, and is a decimal number from 1 to 65535.**

***B = bb* specifies the position of the first byte to be changed. It is a decimal number from 1 to 256.**

***F = cc* is defined above.**

***C = dd* is defined above.**

This command lets you make minor corrections in any disk file, provided that:

1. You know the existing contents and location of the data you want to change.
2. You want to replace one string of code or data with another string of the same length.

You can use PATCH to make minor changes to your own machine-language programs; you won't have to change the source code, re-assemble it, and re-create the file. You can also use it to make minor changes in data files.

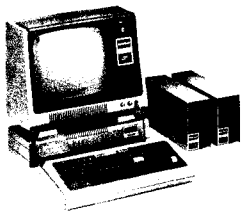
PATCH enables you to specify F (FIND) as an ASCII string such as "JOHN" and replace it with a hexadecimal sequence (of same length) such as 4F48494F.

Another application for PATCH is to allow you to implement any modifications to TRSDOS that may be supplied by Radio Shack. That way, you do not have to wait for a later release of the operating system.

## Example

Using PATCH on a data file (including BASIC programs).

PATCH areas are specified in terms of the logical record which contains the data, and the starting byte of the data in that record. (The TRSDOS LIST command gives this information.)





For example suppose you need to change a 4-byte sequence in a file called NAMEFILE. When you list the file, you find that the sequence is located in record 128, and that the sequence starts at byte 14. Write down the information like this:

| | |
|---|---|
| File to be changed | NAMEFILE |
| Record number | 128 |
| Starting Byte | 14 |
| Sequence of text to be changed | "JOHN" |
| Replacement text | 4F48494F |

Then use the following command:

```
PATCH NAMEFILE (R=128,B=14,F="JOHN",C=4F48494F)
```

or the command PATCH (ENTER), where TRSDOS would then ask what the filespec is for the file to be patched, whether to patch an address or record, etc.

**Note:** The string you are changing must be entirely contained within the specified record. If it spans two records, you will have to perform the patch operations twice, once for each record.

The following command could be used to change a program file:

```
PATCH VREAD (A=5200,F=0CD2D25E5,C=000000009)
```

# PAUSE
# Pause Execution for Operator Action

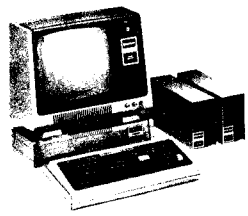
**PAUSE *message***

***message* is the message to be displayed during the pause execution. This is optional. If omitted, PAUSE will be displayed by itself.**

This command is intended for use inside a DO file so TRSDOS can print a message or reminder.

To continue after the pause, TRSDOS prompts you with the message:

```
PRESS <ENTER> TO CONTINUE
```



## Example

```
PAUSE INSERT DISKETTE #21
```

TRSDOS displays PAUSE, next the message and then prompts you to press (ENTER) to continue execution.

```
PAUSE
PRESS <ENTER> TO CONTINUE
```

TRSDOS displays PAUSE and then next prompts you to press (ENTER) to continue. See BUILD and DO for sample uses.

# PROT
# Use or Change a Diskette's Master Password

**PROT :*d* (PW,LOCK)**

**:*d* is an optional drive specification. If omitted, Drive 0 is used.**

**PW tells TRSDOS you want to change the master password.**

**LOCK tells TRSDOS to assign the master password to all unprotected user files. If omitted, the unprotected files remain unprotected.**

PROT lets you use the master password to protect all unprotected files at once, or to change the master password.

The master password will be needed to BACKUP the diskette, so be sure to remember it!

**Note:** The master password on the TRSDOS factory-release diskette is PASSWORD.

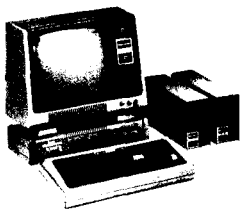
## Examples

```
PROT :0 (PW)
```

Tells TRSDOS to change the master password on the Drive 0 diskette. TRSDOS will prompt you first for the old master password, then for the new master password.

```
PROT :1 (LOCK)
```

Tells TRSDOS to assign the master password to all unprotected user files. TRSDOS will first prompt you for the master password.





# PURGE
## Delete Files

**PURGE :*d (file-type)***

**:*d* is the drive which contains the disk to be purged. *d* is optional; if omitted, Drive 0 is used.**

***file-type* must be one of the following:**

**SYS All system plus user files.**
**INV All invisible plus user files (No system files).**
**ALL All files on diskette except TRSDOS.**
**MIN All files except those necessary for minimal system diskette.**

**(If *file-type* is omitted, TRSDOS will list user files.)**

This command allows quick deletion of files from a particular drive. When the command is entered, TRSDOS will ask for the diskette's master password. (TRSDOS System diskettes are supplied with the password PASSWORD). After the password is supplied, press (ENTER) and TRSDOS will display filenames one at a time, prompting you to kill, leave the file or to quit (return to TRSDOS).

`PURGE * :d (file-type)`

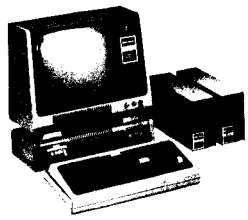
The asterisk (*) tells TRSDOS to ask you if you want to delete the TRSDOS System files. These files are not shown on any of the directory listings and may be deleted with this special form of purge. If you delete all of the files, the diskette becomes a data diskette and may only be used in Drive 1, 2, or 3.

The other parts of this command are as explained previously. However, be sure to do the PURGE using Drive 1, 2 or 3, since the diskette will become "non-system" during PURGE.

`PURGE * :d (MIN)`

This command allows you to purge all of a TRSDOS system except those necessary for a minimal system diskette. User files will be listed at this time. SYStem and INVisible files must be purged using previous options. On a single diskette system these systems (such as BASIC) must be purged prior to using the MIN command.

A TRSDOS minimal system file includes I/O drivers, Error handler and four TRSDOS I/O calls ($OPEN, $INIT, $CLOSE and $KILL). The library commands that are



included are DIRectory, clear screen (CLS), blinking cursor (BLINK), TRACE, CLOCK, and VERIFY.

**CAUTION:** If an attempt is made to use a non-available Library command, a no system warning (0 NS) will appear on the screen. It may be in double character size mode. Press (BREAK) or any key and control will be returned to TRSDOS.

| FILES LISTED FOR PURGE | (FILE-TYPE) | | | | | |
|---|---|---|---|---|---|---|
| | DEFAULT | (SYS) | (INV) | (ALL) | *(XXX) | *(MIN) |
| TRSDOS System Files | | | | | | |
| All Library Commands | | | | | X | |
| All Library Commands (Except DIRectory) | | | | | | X |
| Error Handler | | | | | X | |
| Machine Language Subroutines ($OPEN, $INIT, $CLOSE & $KILL) | | | | | X | |
| ULC (System, Invisible) | | X | | X | | |
| ULP (System, Invisible) | | X | | X | | |
| BASIC (System, Invisible) | | X | | X | | |
| BASIC Overlays | | X | | X | | |
| LPC (Invisible) | | | X | X | | |
| MEMTEST (Invisible) | | | X | X | | |
| USER (Non-Invisible) | X | X | X | X | X | X |

BASIC overlays can only be purged with (SYS) or (ALL) commands prior to using (MIN).

## Example

```
PURGE :1
```

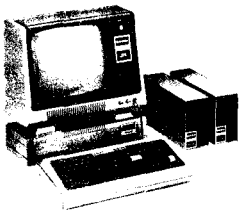
TRSDOS will purge user files from Drive 1. This would include BASIC programs.

```
PURGE :1 (INV)
```

TRSDOS will purge all invisible files in Drive 1.

## How to use the (MIN) TRSDOS System Diskette to write BASIC programs

A minimum TRSDOS System diskette can be easily made with a single-disk drive system and is very useful for writing BASIC programs that need a maximum of disk space.





You must use one of your BACKUPS of the full System diskette to make a (MIN) System diskette. For additional copies of (MIN) System diskette, use BACKUP. To make the original diskette, use the PURGE command twice. First, purge from your diskette all of the files, except TRSDOS. You may use the following command:

PURGE :*d* (ALL), or

PURGE * :*d* (ALL) (Your response must be N, when asked to delete System files.

All files, including BASIC may be PURGEd from the diskette. The remaining files on the diskette are the TRSDOS System files. Type PURGE * :d (MIN) followed by (ENTER) will complete the minimum System diskette. **Note:** The BASIC overlays are not PURGEd when BASIC is deleted but are loaded with TRSDOS and will remain as part of the (MIN) System.

A TRSDOS minimal system file includes I/O drivers, Error handler and four BASIC overlays or TRSDOS I/O calls ($OPEN, $INIT, $CLOSE and $KILL). The Library commands that are included are DIRectory, clear screen (CLS), blinking cursor (BLINK), TRACE, CLOCK, and VERIFY.

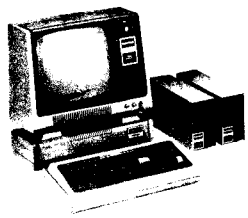
To use this diskette, load BASIC into RAM using the full System diskette. Remove the full System diskette from your drive and insert your (MIN) System diskette. You can now use BASIC to write and save the programs on your (MIN) diskette. If you exit BASIC, it must be reloaded from a full System disk. If you exit BASIC before saving your program, you may be able to restore it by loading BASIC * from the full System diskette. (Do not load a TRSDOS command or power-down prior to using BASIC *).

If you have more than one disk Drive, you can insert the full System diskette in Drive 0 and the (MIN) System diskette in another drive.

# RELO
# Change Where Program Loads into Memory

**RELO *filespec* (ADD = *aaaa*)**

**ADD = *aaaa* specifies the new load address. *aaaa* is a four-digit hexadecimal number referring to an address in the user memory. *aaaa* must be in the user area of RAM.**



This command allows you to change the address at which a machine-language program loads into memory. It does not change the program itself.

**Note:** This command may be useful in conjunction with DUMP.

## Example

```
RELO PROGRAM/CMD (ADD=6578)
```

TRSDOS will load the program PROGRAM/CMD at the new memory address of 6578.

# RENAME
# Rename a File

**RENAME *oldname newname***

***oldname* is the old file name.**

***newname* is the new file name.**

**The file name may include a drive specification and or password.**

**The new file name should not include a drive specification or password.**

This command lets you rename a file or program. Only the name/extension is changed; the password data in the file and its physical location on the diskette are unaffected.

RENAME cannot be used to change a file's password protection. Use ATTRIB to do that.

RENAME also checks to see that the intended new name does not duplicate a filename currently on the same diskette. If it does, the command is cancelled and an error message is displayed.

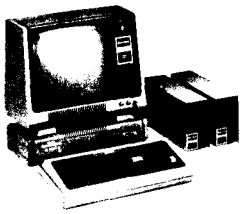
## Examples

```
RENAME MATHPAK MATHPAK/BAS
```

Tells TRSDOS to add the extension to the filename.

```
RENAME ABCDE/DAT ABCDEF/DAT
```

Tells TRSDOS to change the filename only.





```
RENAME PAYROLL1/TXT:GSR PAYROLL2/TXT
```

Tells TRSDOS to change the filename; the password is retained automatically.

```
RENAME FILE1:3 FILE2
```

Tells TRSDOS to change the filename of the file on Drive 3.

# SETCOM
# Set Up RS-232-C Communications

**SETCOM (WORD = *a*,BAUD = *b*,STOP = *c*,PARITY = *d*,)**

**WORD = *a* is the number of bit/byte desired. *a* must be either 5, 6, 7, or 8, depending on your needs. If omitted, the word length is not changed.**

**BAUD = *b* specifies the baud rate. *b* must be one of the following (110, 150, 300, 600, 1200, 2400, 4800, or 9600). If omitted, the baud rate is not changed.**

**STOP = *c* specifies the number of stop bits. *c* must be either 1 or 2. If omitted, stop bits are not changed.**

**PARITY = *d* determines whether the parity is odd, even, or none. *d* must be 3 (none), 1 (odd), or 2 (even). If omitted, parity is not changed.**

**Settings for SETCOM at Power-up and after Reset are BAUD = 300, WORD = 7, STOP = 1, and PARITY = EVEN. Typing SETCOM (ENTER) will display the current RS-232-C settings. Typing SETCOM ( ) (ENTER) will cancel all settings and return to default values.**

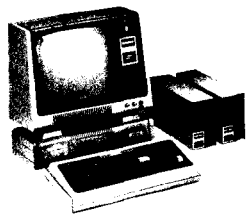
This command initializes RS-232-C communications via the serial channel, providing that your Expansion Interface has a RS-232-C Serial Interface Board (26-1145) installed. Before executing this command, you should connect the communications device to the Model I.

**Note:** SETCOM is not a serial driver. It is simply a way to change the default parameters of the RS-232-C.

## Example

```
SETCOM (WORD=7,BAUD=300,STOP=1,PARITY=3)
```

This would set the RS-232-C to seven bit words, 300 baud, one stop bit, and no parity.



SETCOM

The command without specifications will display the current settings.

For further information, see the TRS-80 RS-232-C Interface Manual. (Catalog Number 26-1145).

# SPOOL
# Turn Print Spooler On

**A) SPOOL *nnnn***

***nnnn*—number of decimal bytes to use for the spool buffer. This decimal length is subtracted from the current top of memory.**

**B) SPOOL, *x'nnnn'***

***nnnn*—hexadecimal address in memory. Spool buffer will use the memory from *nnnn* to the top of memory.**

**C) SPOOL—with no option, turns SPOOL OFF.**

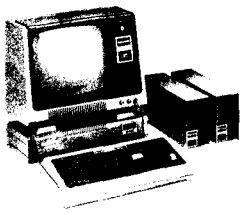
This command increases the efficiency of the system by allowing you to use the system while a printer operation is in progress. SPOOL (Simultaneous Peripheral Operations On Line) does this by using the CPU "wait" times, (when the CPU is idle, i.e., waiting for an input), to send information out to the buffer of the Printer. This enables the CPU to continue processing, rather than wait until the Printer operation is complete.

The information to be sent to the Printer is stored at the top of memory in a special buffer set up by you when SPOOL is initialized. For example, SPOOL 7000H sets aside 7000H bytes at the top of memory for the buffer. Thus instead of the text being sent directly to the Printer (which ties up the CPU until the printing is complete), SPOOL routes this information to its special buffer. Then, as regular CPU and program operations continue, SPOOL will use the CPU's idle periods to send the information to the Printer.

When SPOOL is active, the end of memory is automatically changed for you so a BASIC program will not interact with this memory. You may press (ENTER) to the prompt:

MEMORY SIZE?

because this memory is already protected by the system.





SPOOL must be off before commands SPOOL *aaaa* or SPOOL *X'bbbb'* can be entered or the message:

```
NO CHANGE MADE, SPOOL IS ALREADY ON.
```

will be displayed. In this case, turn SPOOL off by entering:

SPOOL (ENTER)

and the commands will be accepted. Use SPOOL (ENTER) to abort the SPOOLing of text to the printer.

**Note:** SPOOL will work with the LPC (Line Printer Control) driver, but LPC must be installed before the SPOOL command. See LPC in this manual for further information. Turning SPOOL "off" (see above paragraph) will also kill the LPC driver.

## Example

When TRSDOS READY is displayed, type: SPOOL 10000 (ENTER). This will allocate the top 10,000 bytes of memory for the SPOOL buffer.

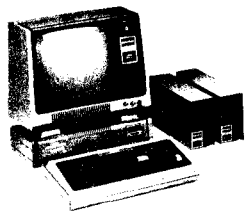
Now load and execute BASIC typing: BASIC (ENTER). Press (ENTER) for MEMORY SIZE? and HOW MANY FILES?. Now run this program (be sure your Printer is ready):

```
10 FOR X% = 1 TO 1000
20 LPRINT "ABCDEFGHIJ";
30 NEXT X%
```

When the program ends, 10,000 characters have been sent to the printer. This means that 10,000 characters have been stored up to be printed in a matter of a few seconds.

Now exit BASIC with CMD"S:" Other TRSDOS LIBRARY commands (such as DIR) may now be executed while the printing continues as a background task.

**Note:** Any continuous disk input/output such as FORMAT, BACKUP, or heavy file access can completely stop printing. However, printing will resume as soon as these disk operations are finished. Commands that use all of memory, such as FORMAT, BACKUP, and COPY should not be used while the spooler is in operation.



# TAPE
# Tape/Disk Transfer

**TAPE (S = *source*,D = *destination*)**

***source* and *destination* are abbreviations for the storage devices to be used:**

- **T Tape**
- **D Disk**
- **R Random access memory**

**Note: TAPE can only be used with machine-language programs. BASIC programs must be CLOADed and CSAVEed.**

This command transfers Z-80 machine-language programs from one storage device to another. The following transfers are possible:

- Tape to disk
- Disk to tape
- Tape to RAM

## Examples

```
TAPE (S=T,D=D)
```

Starts a tape-to-disk transfer. TRSDOS will prompt you `DEVICE = TAPE TO DISK`. TRSDOS will then prompt you to press any key when the recorder is ready to send information to the Computer (recorder should be in Play mode). When you press a key, the tape will begin loading.

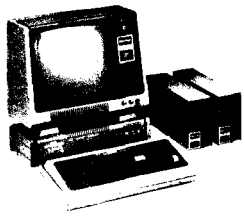
**Note:** If no asterisks flash, the recorder volume may need adjustment.

TRSDOS will read the file name from the tape and use that name for the disk file and append the extension CMD. It will copy the program to the first write-enabled diskette, starting with the master drive (see **MASTER**).

```
TAPE (S=D,D=T)
```

Starts a disk-to-tape transfer. TRSDOS will prompt you

```
DEVICE = DISK TO TAPE - FILESPEC,,,,,
```





Enter filespec and then it will tell you to press (ENTER) when the Cassette Recorder is ready to record from the Computer.

```
TAPE (S=T,D=R)
```

Starts a tape-to-RAM transfer. TRSDOS will prompt you `TAPE INTO RAM`, and will tell you to press any key when the recorder is ready to send information to the Computer's memory. After loading the program, TRSDOS will begin execution at the transfer address specified on the tape.

**Note:** Anytime the TAPE command is used, your machine-language program must reside above 6FFF HEX if you wish to run the program.

# TIME
# Reset or Get the Time

**TIME *hh:mm:ss***

***hh:mm:ss* specifies the hour *hh*, minute *mm*, and second *ss*.**

**Each must be a two-digit decimal number between the following ranges:**

| | |
|---|---|
| ***hh*** | **0-23** |
| ***mm*** | **0-59** |
| ***ss*** | **0-59** |

**If *hh:mm:ss* is given, TRSDOS resets the time.**

**If *hh:mm:ss* is not given, TRSDOS displays the current time.**

This command lets you reset or display the time.

TIME uses a 24-hour clock. For example, 1:00 P.M. is displayed as 13:00.

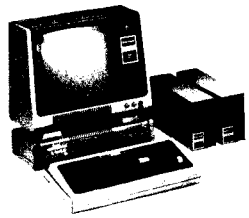
TRSDOS automatically updates the time using its built-in clock.

When you request the time, TRSDOS displays it in this format: 14:15:31 for 2:15:31 P.M.

## Example

```
TIME
```

Displays the current time.



```
TIME 13:20:00
```

Resets the time to 13:20:00 P.M.

**Note:** If the clock is allowed to run past 23:59:59, it will re-cycle to zero, but the date will not be incremented. The clock will continue to run.

See CLOCK and DATE.

# TRACE
# Dynamic Display of PC Register

**TRACE (*switch*)**

***switch* is either ON or OFF. If *switch* is omitted, ON is used.**

The TRACE command enables a foreground task which displays the contents of your program instruction counter (PC Register) in the upper right of the Video Display. The four-digit hexadecimal value will be updated every eight milliseconds with the current background program's execution address. Since it is a foreground task, TRACE operates at all times. To temporarily disable TRACE, disable all interrupts (CMD"T" in Disk BASIC). When interrupts are re-enabled (CMD"R" in Disk BASIC), TRACE will start up again.

When used with the DEBUG program, TRACE can be very useful in debugging machine-language programs, but it won't be of much use during BASIC program execution.

## Example

```
TRACE (ON)
```

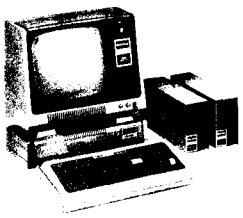
Turns the TRACE feature ON.

```
TRACE (OFF)
```

Turns the TRACE feature OFF.

# ULC
# Upper Lower Case Driver

On start-up, TRSDOS checks to see if the Lower Case Kit (26-1104) is installed. If it is, TRSDOS automatically loads and executes this driver. If the modification is not present, the driver will not be loaded.





No other drivers are necessary for the Lower Case Kit.

**Note:** This program cannot be user loaded. It may be loaded by TRSDOS only.

If the driver is not desired, simply kill the file on the diskette. ULC is not password protected.

**Note:** ULC loads into highest available memory. There is no need to set `MEMORY SIZE?`. ULC "hides" itself. However, you must set `MEMORY SIZE` if required by your application program.

# UNKILL
# Recover a KILLed File

**UNKILL *filespec :d***

***filespec:d* is a file specification that includes a drive number.**

This command recovers a file that was previously KILLed provided that:

- Filename has not been written over by a new filename. (i.e., if files have been opened since the file was KILLed, UNKILL may not be able to recover the file). If filename cannot be found because of this, ERROR 24 (file not found) will be displayed.
- The diskette area where the file was stored has not been written over. If this occurs, ERROR 33 (file unrecoverable) will be displayed.

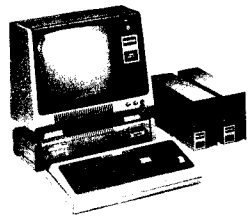
When using UNKILL, if the filename was password protected, then the password must be specified to recover the file.

**Note:** A file that was ERASEd cannot be recovered.

## Example

```
UNKILL PAYROLL/DAT:1
```

Recovers file that has been previously KILLed, provided it has not been written over.



# USER
# User-Defined Library Command

**USER *command***

***command* is a command you specify and must begin with an alphabetic character. Only the first six characters will be recognized and must be unique. If a new USER command is defined, it will replace the previous command.**

This command allows you to create one (and only one) library command. This command will become a permanent feature of the TRSDOS diskette it is stored on and will be displayed when you use LIB.

USER must be a machine-language routine created using DUMP/DEBUG or Editor/Assembler. It must be saved using the filename LIB without any extension. The program must not reside lower than 5200H.

The User-Defined Library command is used just like TRSDOS Library commands. Upon entering your program, HL register points to the next non-blank character on the command line and DE contains the end of memory. This will enable you to pass options to your routine. If options are not specified, HL will point to the (ENTER).

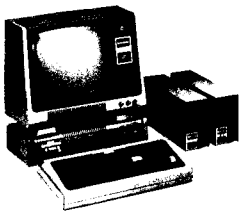
There are also some program security features such as:

- The program file LIB may be fully password protected. TRSDOS will override these passwords to execute the routine, but this will allow you to protect LIB from being listed.
- When the user library command is being executed, DEBUG will be disabled. This will prevent access to the program through DEBUG.
- After execution of the users library command, when TRSDOS READY is returned, all memory will be cleared, so the program will not remain in memory.

To kill or delete the user-command from the library, type:

USER (ENTER)

**Note:** This does not kill the file.





## Example

In the following example, a Library command will be created. It will display a simple message to the video and return to TRSDOS READY. This Library command will be called MYPROG.

Using DEBUG, enter the machine-code and save it to diskette using DUMP:

Type:

```
DEBUG (ENTER)
```

and press (BREAK). You are now ready to use DEBUG.

Type:

```
M
```

DEBUG will prompt `ADDRESS = ;`

Type:

```
7000 (SPACEBAR)
```

Now enter the following machine-code without spaces between each pair.

```
21 09 70 CD 67 44 C3 2D 40 48 45 4C 4C 4F 0D
```

After entering all the code press (ENTER). Then press (Q) to exit the DEBUG monitor.

Type in the following command to save this program to the TRSDOS diskette in Drive 0:

```
DUMP LIB (START=7000,END=7010,TRA=7000)
```

Since DUMP adds the extension /CMD to its file, we must rename LIB/CMD to LIB, because USER will not accept the extension. Type the following to Rename the file:

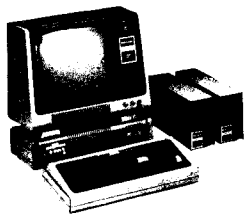
```
RENAME LIB/CMD to LIB (ENTER)
```

Now you are ready to make this new program part of the library commands in TRSDOS.

Type in the following:

```
USER MYPROG (ENTER)
```

If you display the library commands (LIB), you will see MYPROG has been added to the list. By typing MYPROG into the TRSDOS READY prompt, the program can be executed.



# VERIFY
## Verify Disk Writes

**VERIFY (*switch*)**

***switch* is either ON or OFF. *switch* is optional; if omitted TRSDOS uses ON.**

This command causes TRSDOS to verify all user disk writes (for example, file-writes from Disk BASIC). This will be useful when you want to be sure that no data is lost or altered during a disk write. For example, before you COPY a file, you may want to enable VERIFY.

However, when VERIFY is ON, disk accesses are only about 50 percent as fast as normal.

VERIFY does not affect system table and directory writes; they are always verified.

### Example

```
VERIFY (ON)
```

Tells the Computer to verify all disk writes.

```
VERIFY (OFF)
```

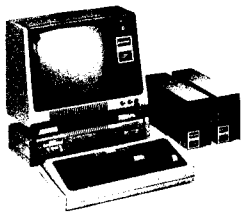
This disables the automatic read-after-write verification.

**Note:** VERIFY is ON when TRSDOS powers-up.

# WP
## Write-Protect Via Software

**WP (DRIVE = *d*)**

***d* specifies the disk drive to be protected. If omitted, all drives will be unprotected.**





Diskettes can be protected from being overwritten. It is a software write-protect rather than a hardware write-protect (such as the write-protect tab on the diskette).

Only one drive may be protected at a time.

To unprotect a drive, making it accessible to writing, simply enter the command WP without options or with a different drive number specified. The WP command will not override a write-protect tab.

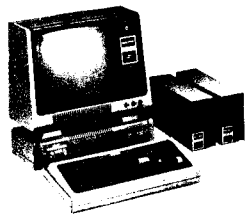
## Examples

```
WP (DRIVE=1)
```

TRSDOS will write-protect the disk in Drive 1.

```
WP
```

TRSDOS will eliminate write-protection on all drives.



# 4/Technical Information

## Memory Organization

The TRS-80 Double-Density Disk Operating System is contained in 1K of ROM resident drivers and 6K of RAM drivers, schedulers, tables, pointers, etc. The ROM resident drivers are also used by LEVEL II BASIC and, therefore, are part of its 12K ROM requirement.

Since LEVEL II is upward compatible with Disk BASIC, an additional 0.5K of RAM is required for both versions of BASIC. This means that user memory starts at hex 5200, resulting in 11.5K of user RAM in a 16K machine.

**Note:** The memory which is completely untouched by both TRSDOS and Disk BASIC begins at hex 7000.

TRSDOS is comprised of a resident system and several overlays which are loaded from disk as the need arises (for example, to open or close a file).

## Disk Organization

Each TRSDOS system diskette contains a TRSDOS system, a utility, command library, and a file directory.

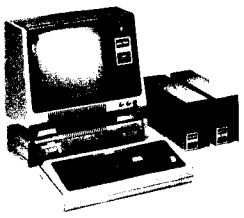
Each diskette is single-sided and has 35 or 40 tracks of information (See CONFIG). Each track contains 18 sectors of 256 bytes.

Normally, data read/write operations may be initiated only at sector boundaries, and must consist of exactly 256 bytes. However, TRSDOS allows the user to have maximum flexibility with minimal effort by automatically blocking and de-blocking all file accesses to user-specified logical record lengths, even if this requires "spanning" of two sectors.

The system disk file structure allows maximum use of disk file space by automatically segmenting files across a diskette in several small pieces. These pieces are correlated into one logically contiguous file by the system without your needing to know the physical file location. This structure eliminates time-consuming disk-packing operations.

## File Structure

A TRSDOS file is composed of one or more segments of storage space. Each segment consists of from one to 32 physically contiguous granules of storage. A granule is the minimum allocatable unit of storage, and consists of three sectors (768 bytes).





Since a file is always lengthened by granules, a small amount of free storage is generally present at the end of every file. This free storage allows minor file additions to be made in space which is physically contiguous to the file.

Every time a disk file is extended (either initialized or lengthened), extra granules may be allocated to that file, depending on the file's accumulated length, diskette space, saturation, etc. These extra granules, along with all granules after the one containing the file's EOF mark, are recovered and returned to the system when the file is closed.

# Units of Allocation

The minimum allocatable unit of storage is the "granule." A granule is defined as three sectors (768 bytes). There are six granules per track.

**Storage Capacity for User Files (Double-Density)**

| | Tracks | Granules | Sectors | Bytes |
|---|---|---|---|---|
| Full System Disk | 35 | 114 | 342 | 87,552 |
| | 40 | 144 | 432 | 110,592 |
| TRSDOS Plus | 35 | 120 | 360 | 92,160 |
| BASIC | 40 | 150 | 450 | 115,200 |
| TRSDOS Only | 35 | 128 | 384 | 98,304 |
| | 40 | 158 | 474 | 121,344 |
| MIN-TRSDOS | 35 | 174 | 522 | 133,632 |
| System | 40 | 204 | 612 | 156,672 |
| Data Disk | 35 | 198 | 594 | 152,064 |
| | 40 | 228 | 684 | 175,104 |

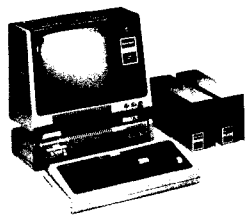
# Methods of File Allocation

TRSDOS provides two ways to allocate disk space for file: Dynamic Allocation and Pre-Allocation.

## Dynamic Allocation

With Dynamic Allocation, the System allocates granules only at the time of write. For example, when a file is first opened for output, space is not allocated. The first space allocation is done at the first write. Additional space is added as required by subsequent writes.

With dynamically allocated files, unused granules are de-allocated (recovered) when the file is closed.



## Pre-Allocation

With Pre-Allocation, the file is allocated a specified number of granules when it is created. Pre-Allocated files can only be created by the operator command CREATE (See TRSDOS section of this manual).

TRSDOS will dynamically extend (enlarge) a Pre-Allocated file as needed for subsequent write operation. TRSDOS will also de-allocate unused granules when a pre-allocated file is opened for sequential writes, is written to and is closed.

**Even though disk space may be available, disk-full errors may occur if you attempt to create more files than are available tracks. For instance, if 3 tracks are available and you attempt to open 4 files, errors may occur.**

# Physical and Logical Records in TRSDOS

A physical record is defined as one sector of a disk. One sector of a disk contains 256 user data bytes. The artificial term "granule" is defined to be 3 sectors of disk space. There are 6 granules on each of the tracks on the disk. A granule is the least amount of space allocated by TRSDOS. For programming purposes, the physical records in a file are numbered from 0 to N. The largest physical record number (N) in a file will then be three times the number of granules allocated minus one ((3*G) − 1). All TRSDOS granule allocations are made as needed *at the time of a write, not when the file is created.*

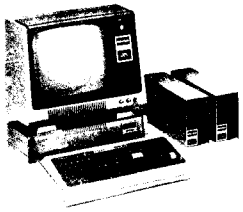
A logical record is defined by the user of TRSDOS. It may be anywhere from 1 to 256 bytes in length. Once a file is opened with a specific LRL (Logical Record Length), the length is fixed until the file is closed.

Each opening of the file sets a single, fixed record-length. TRSDOS will "block" logical records into (or from) one physical record for maximum space utilization on the disk.

**Blocking** is putting more than one logical record into one physical record. For instance, four 64-byte logical records will fit into one 256-byte physical record. A logical record may be broken into two parts by TRSDOS in order to fill the last portion of one physical record entirely before beginning to use the next physical record (i.e. records are spanned). This occurs when the physical record length is not an even multiple of the logical record length.

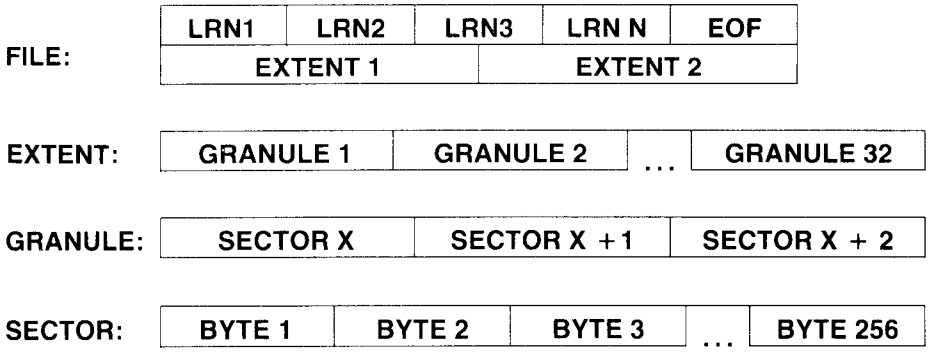
If the user wishes to do his own blocking, he may specify a logical record length of 0 bytes at the time of INIT/OPEN and must himself manage the contents of the physical record buffer area of 256 bytes. TRSDOS *will not move* a logical record for the user if LRL = 0; in this particular case it will only read/write the physical record to/from the buffer. Once control is shifted to your program, you will have about 20 bytes of stack size left.

**Spanning** is spanning across sectors. If the record length is not an even divisor of 256, the records will automatically be spanned across sectors. For example, if the record length is 200, Sectors 1 and 2 will look like this:





## A TRSDOS file




**LRN:** Logical Record Number, used to specify an individual, user-defined logical record. Such a logical record is the smallest unit of information which can be addressed during disk input/output (a physical record is the unit which is actually read from or written to disk).

**File:** A group of logical records; the largest unit of information which can be addressed by a TRSDOS command.

**Sector:** A physical record, composed of 256 contiguous bytes.

**Granule:** The minimum allocatable unit of storage for any file.

**Extent:** One contiguous allocation of granules.

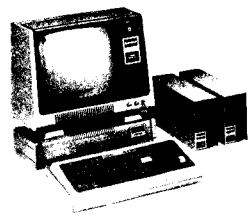
# System Routines for Assembly-Language I/O

This information is provided for customers who wish to write their own assembly level I/O routines. An explanation of the calling sequence and parameters for each necessary I/O routine is given. A knowledge of Z-80 machine code is assumed.

The following notations are standard in this section:

(HL) = *xxxx* — Registers HL contain the address of (point to) *xxxx* in machine format. (If address of *xxxx* = 34B2H then the values in the registers are: H = 34; L = B2). Other register pairs will also be indicated this way.

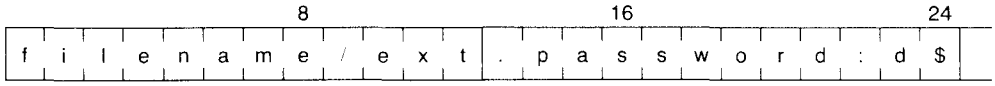
A = *xx* — Register A contains the numeric value of *xx* in binary form. Register A is used to return the TRSDOS error code for I/O calls. A complete list of error codes and their meanings appears at the end of this chapter. Other registers will also be indicated this way.



| | |
|---|---|
| Z = OK | Zero flag is set (OK) if successful return from the system routines. |
| X'*nnnn*' or *nnnn*H | Hard RAM address in hex notation (e.g., 402D is X'402D'). |
| LRL | Logical Record Length, 1-255 bytes only. You can define records any length you wish up to 255 bytes maximum. A length of zero is a special case for physical records only, and indicates the LRL = 256 bytes. |
| BUFFER | 256 user-designated bytes in RAM for TRSDOS to read sectors from or write sectors into. If LRL = 0, this area is the responsibility of the user to manage before and after I/O. TRSDOS manages this area if LRL is between 1 and 255 bytes. Do not alter this area when using logical record processing. |
| UREC | (User Record) The address of the contiguous RAM byte-string assigned by the user as his logical record area. Its length must be equal to LRL. It is a different area from BUFFER. |
| LSB/MSB | Least-significant byte followed by most significant byte. This is the standard Z-80 format for addresses. |
| $*name* | The "$" is prefixed to all system locations and call routines, so they will not be confused with TRSDOS commands or utilities. For example, $OPEN. |

## DCB before $OPEN and after $CLOSE

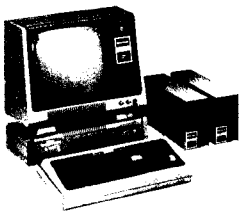
The DCB (device control block) is defined as 32 contiguous bytes of RAM designated by the user. Before $OPEN and after $CLOSE, it is a left-justified, compressed (no spaces) ASCII string, as in a standard TRSDOS filespec. The string is terminated with a carriage return.




Notes: /ext, .password, and :d are optional.
$ stands for a carriage return (X'OD')

## Explanation of DCB while OPEN:

*lsb/msb* is least significant byte followed by most significant byte in Z80 RAM format (i.e. addr = 7CC8 in RAM is C8 7C).





| Address | Length | Explanation |
|---|---|---|
| DCB+0 | 3 | Reserved |
| +3 | 2 | Physical Buffer address (lsb/msb) |
| +5 | 1 | Offset to delimiter at end of current record (ODECR) |
| +6 | 1 | File drive number residence |
| +7 | 1 | Reserved |
| +8 | 1 | EOF offset of last delimiter in last physical record |
| +9 | 1 | LRL (logical record length) |
| +10 | 2 | NRN (next record no. – open sets = X'0000' – lsb/msb) |
| +12 | 2 | ERN (ending record no. – (last in file) – lsb/msb) |
| +14 | 18 | Reserved |

**NRN** Next Record Number defines which record is to be read or written by the next system call for READ or WRITE. It is automatically incremented by one after each system call. In order to process random files, use the POSN call to direct TRSDOS to the record you wish to transfer next.

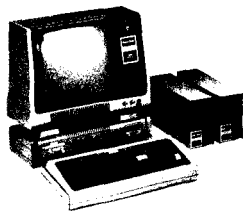
**ERN** Ending Record Number is the last record number currently in the file. It is put into the directory at CLOSE time, so if it is expected to be correct, the user *must* close his files after adding records to a file. This value may also be used to position to end of file so that new records may be added to the end of the file. To position to the end of file use a call to POSN with a record number of ERN + 1. POSN is described later.

# Fundamental TRSDOS I/O Calls

There are 20 fundamental TRSDOS routines involved in handling file I/O. These are:

| | |
|---|---|
| $BKSPC | $POSN |
| $CLOSE | $PRLINE |
| $DIVIDE | $PUTEXT |
| $DMULT | $RAMDIR |
| $DOSABRT | $READ |
| $FILPTR | $REWIND |
| $INIT | $SYNTAX |
| $KILL | $VDLINE |
| $OPEN | $VERF |
| $POSEOF | $WRITE |

The detailed calling sequences and discussions for each of these routines follow. Note that *all* of these system calls use register F and do not restore its value before return. In order to apply this data properly, you should read through all of these descriptions and clear up all of the points that are not obvious to you by using other reference materials. If you are successful in doing this you will find that TRSDOS is a workable tool for your programming ideas.



# $INIT — 17440/X'4420'

$INIT is provided as an entry point to TRSDOS which will create a new file entry in the directory and open the DCB for this file. $INIT scans the directory for the filespec name given in the DCB. If the filespec name is found, $INIT simply opens the file for use. If the name is not found, a new file is created with the filespec name.

## Entry Conditions

(HL) = BUFFER (see beginning of this section for notation)
(DE) = DCB
B = LRL
CALL $INIT

## Exit Conditions

IY = changed
Z = OK
C carry flag is ON if a new file was created
A = TRSDOS error code. (Error codes listed at end of this chapter)

# $OPEN — 17444/X'4424'

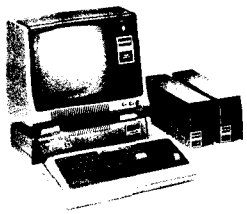
$OPEN provides a way to open the DCB of a file which already exists in the directory. The DCB *must* contain the filespec of the file to be opened *before* entry to $OPEN. The file will be opened using the LRL from the directory and ignore what the user supplies (Register B). In other words, the file will be opened as it is created.

## Entry Conditions

(HL) = BUFFER
(DE) = DCB
B = LRL
CALL $OPEN

## Exit Conditions

Z = OK
Z = 0 if file does not exist.
A = TRSDOS error code.
IY = changed





# $POSN—17474/X'4442'

$POSN positions a file to read or write a randomly selected logical record. Since it deals with logical records, the proper computation is done to locate which physical record(s) contain the data. Following a $POSN with a $READ or $WRITE will transfer the record to/from RAM.

Note that positioning to logical record zero sets the file to read the first logical record in the file. To position to end of file in order to add new records onto the end, use the record number ERN + 1.

## Entry Conditions

(DE) = DCB (must have been opened previously)
BC = Logical record number to position for.
CALL $POSN

## Exit Conditions

Z = OK
A = TRSDOS error code.

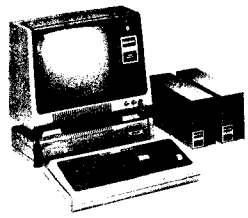
# $READ—17462/X'4436'

If LRL not equal to zero, then $READ transfers the logical record whose number was placed in the DCB by $POSN into the RAM area addressed as UREC. The value of LRL is defined at open time. The record comes from "BUFFER" defined at open time. If TRSDOS must read a new physical record to satisfy the request, it will do so. "Spanned" logical records will be re-assembled as necessary. $READ will automatically increment in the DCB the offset to delimiter at end of current record (ODECR) by the value of LRL for each logical record and NRN by one for each physical record after each transfer is completed. $INIT/$OPEN will set NRN = X'0000 and ODECR = X'00 in order to read the first record with the first $READ.

If LRL = 0, $READ transfers one physical record into BUFFER, defined at open time, from the disk file. Registers HL are ignored. $READ increments NRN as above.

## Entry Conditions

(HL) = UREC if LRL is not zero. Unused if LRL = 0.
(DE) = DCB
CALL $READ



### Exit Conditions

Z = OK
A = TRSDOS error code. (EOF = X'1C' or X'1D')
(see errors 28,29 for EOF or NRF)

## $WRITE—17465/X'4439'

If LRL not equal to zero, then $WRITE transfers the one logical record from the RAM area addressed by UREC with the length LRL as defined at open time. The record goes into the "BUFFER" which was defined at open time. If TRSDOS must write a physical record in order to satisfy the request, it will do so. "Spanning" will be handled by TRSDOS as necessary. At $INIT/$OPEN time the DCB value of NRN = X'0000 and the offset to delimiter at end of current record (ODECR) = X'00 so the first record can be written. After each logical record is transferred, the ODECR value in the DCB will be incremented by the value of LRL. After each physical record is transferred, the NRN value in the DCB will be incremented by one.

If LRL = 0, $WRITE transfers one physical record from BUFFER into the disk file using the NRN in the DCB. BUFFER is defined at $INIT/OPEN time only. The DCB value NRN is updated as above, after the WRITE.

### Entry Conditions

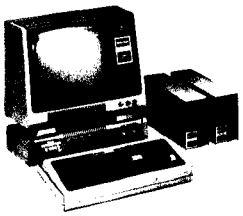
(HL) = UREC if LRL is not zero. Unused if LRL = 0
(DE) = DCB
CALL $WRITE

### Exit Conditions

Z = OK
A = TRSDOS error code.

## $VERF—17468/X'443C'

The only difference between $VERF and $WRITE is that $VERF writes one physical record to disk and then reads it back into a special TRSDOS RAM area not defined by the user. This special area and the original write buffer are then compared byte by byte to assure that the record was successfully written.





### Entry Conditions

(HL) = Same as $WRITE above.
(DE) = DCB
CALL $VERF

### Exit Conditions

Z = OK
A = TRSDOS error code.

## $PUTEXT — 17523/X'4473'

This routine will add an extension to a filename if an extension does not already exist. An extension to a filename may be useful for identifying the type of data in the file.

### Entry Conditions

(DE) = DCB
(HL) = The extension to be added to the file
CALL $PUTEXT

### Exit Conditions

None

## $BKSPC — 17477/X'4445'

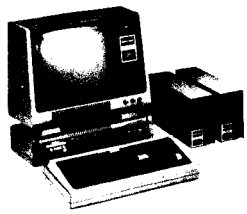
This routine positions the file record pointer to the previous record.

### Entry Conditions

(DE) = DCB
CALL $BKSPC

### Exit Conditions

Z = Valid position
NZ = Invalid position in file



# $REWIND—17471/X'443F'

Point to the beginning of the file. This routine positions the file pointer to the first record in the file. This is useful when the same file must be processed more than once.

## Entry Conditions

(DE) = DCB
CALL $REWIND

## Exit Conditions

Z = Good file specification
NZ = Bad file specification

# $POSEOF—17480/X'4448'

Point to the end of file. This routine positions the file pointer to the last record in the file. This may be used to extend a sequential access file.

## Entry Conditions

(DE) = DCB
CALL $POSEOF

## Exit Conditions

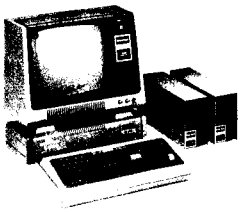
Z = Good file specification
NZ = Bad file specification

# $SYNTAX—17436/X'441C'

Move a file specification to DCB. This routine takes a file specification and checks it for validity and moves it to a DCB so that the file may be opened.

## Entry Conditions

(HL) = Filename
(DE) = DCB
CALL $SYNTAX





### Exit Conditions

Z = Good file specification
NZ = Bad file specification

## $DIVIDE—17486/X'444E'

The divide routine takes a 16-bit dividend and an eight-bit divisor. After division, the quotient replaces the 16-bit dividend and the remainder replaces the eight-bit divisor.

### Entry Conditions

HL = Dividend
A = Divisor
CALL $DIVIDE

### Exit Conditions

HL = Quotient
A = Remainder (0 indicates no remainder).

## $DMULT—17483/X'444B'

The multiply routine uses a 16-bit multiplicand and an eight-bit multiplier. After multiplication takes place, the product replaces the 16-bit multiplicand.

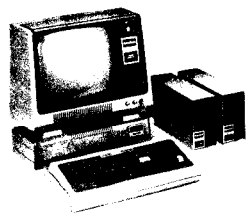
### Entry Conditions

HL = Multiplicand
A = Multiplier
CALL $DMULT

### Exit Conditions

H = High order byte
L = Middle order byte
A = Low order byte

| H | L | A |
|---|---|---|
| High | Middle | Low |



# $RAMDIR—17550/X'448E'

This routine allows you to examine a diskette directory (one entry or the entire directory) or the diskette's free space. The information is written into a user specified RAM buffer.

Only non-system files will be included in the RAM directory.

## Entry Conditions

HL = RAM Buffer. If C=0, size = 2817 [max #*22+1]. If C=1 to 128, size = 22. If C=255, size = 64.
B = Specified drive number
C = Function switch:

| Contents of C | Results |
|---|---|
| 0 | Gets entire directory into RAM. (See RAM Directory Format). |
| 1-128 | Gets one specified directory record into RAM, if it exists. (See RAM Directory Format). |
| 255 | Gets free-space information (See RAM Directory Format). |

```
CALL $RAMDIR
```

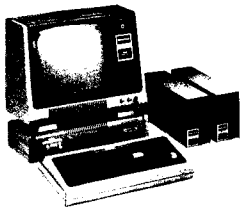
## Exit Conditions

NZ = Error occurred.
Z = No error. (HL) = directory or free-space information.

## RAM Directory Format

The directory is made up of records, one per file. All values are hexadecimal. Each record placed in user RAM is in the following format:

| Byte Number | Contents |
|---|---|
| 0-13 | *filename/ext:d* (left-justified followed by spaces) |
| 14 | Reserved for future use |
| 15 | Protection Level, binary 0-6 |
| 16 | Byte EOF, binary 0-255 |
| 17 | Logical record length, binary 0-255 |
| 18-19 | Last sector number in file, binary LSB, MSB |
| 20-21 | Number of Granules allocated (LSB,MSB) binary<br>After last record of directory (or single record). |
| 22 | "+" (marks the end of directory list after entire directory.) |





## Free Space

HL = address of four bytes containing:
1) the number of granules used
2) the number of free granules

The information appears in the following form:

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| LSB | MSB | LSB | MSB |
| # grans. used | | # grans. free | |
| 62 | 00 | 8E | 00 |

# $FILPTR—17547/X'448B'

This routine provides information on any user file that is currently open. It enables you to obtain the drive number and the logical file number for any file and should be used in conjunction with $RAMDIR.

## Entry Condition

(DE) = Data Control Block (DCB) defined when file was opened.
CALL $FILPTR

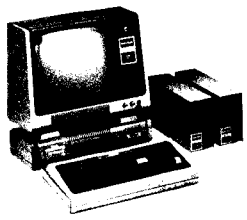
## Exit Conditions

NZ = Error occurred.
Z = No error. The following registers are set up:
B = Which drive contains the file (0,1,2, or 3).
C = Logical file number (1-128)

**Note:** This operates with user files only.

# $CLOSE—17448/X'4428'

$CLOSE closes a file from the last processing done. *It is very important to do a $CLOSE on every file opened before the program ends.* If you do not close a file, the directory entry for this file is incorrect if any new records have been written into the file. Other cases are not given here, but it is very important to TRSDOS that all of the "housekeeping" be complete for file management.



### Entry Conditions

(DE) = DCB
CALL $CLOSE

### Exit Conditions

Z = OK
A = TRSDOS error code.

## $KILL—17452/X'442C'

$KILL deletes the directory entry for a file and releases the disk storage. The file must be open.

### Entry Conditions

(DE) = DCB
CALL $KILL

### Exit Conditions

Z = OK
A = TRSDOS error code.

## $DOSABRT—16432/X'4030'

This routine displays the message: `OPERATION ABORTED`, then transfers control to TRSDOS READY.

### Entry Conditions

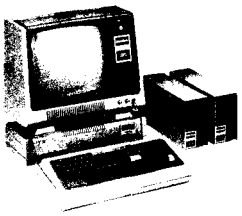
CALL $DOSABRT

### Exit Conditions

None

## $PRLINE—17514/X'446A'

This subroutine prints a line to the Printer. The line must be terminated with an ASCII ETX (X'03') or a carriage return (X'0D'). If the terminator is a carriage return, it will be printed; if it is an ETX, it will not be printed. This allows PRLINE to position print to the beginning of the next line or leave it at the position after the last text character.





### Entry Conditions

(HL) = Output text, terminated by X'03' or X'0D'
CALL $PRLINE

### Exit Conditions

(HL) = the terminator
DE is altered

## $VDLINE—17511/X'4467'

This routine displays a line. The line must be terminated with an ASCII ETX (X'03') or carriage return (X'0D'). If the terminator is a carriage return, it will be printed; if it is an ETX, it will not be printed. This allows VDLINE to position the cursor to the beginning of the next line or leave it at the position after the last text character.

### Entry Conditions

(HL) = Output text, terminated by X'03' or X'0D.'
CALL $VDLINE

### Exit Conditions

(HL) = the terminator
DE is altered.

# Additional Routines and Storage Addresses

## $JP2DOS—16429/X'402D'

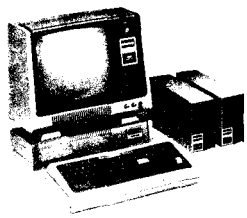
This routine transfers control to TRSDOS READY.

### Entry Conditions

JP $JP2DOS

### Exit Conditions

None



# $DATE—17520/X'4470'
# $TIME—17517/X'446D'

These routines return the date and time in ASCII format:

Date: MM/DD/YY
Time: HH/MM/SS

## Entry Conditions

(HL) = Eight-byte buffer to receive the date/time text
CALL $DATE
CALL $TIME

## Exit Conditions

(HL) = Date or time text

# $DATLOC—16452/X'4044'
# $TIMLOC—16449/X'4041'

These locations store the date and time in binary format:

$DATLOC (Three bytes): MM DD YY
$TIMLOC (Three bytes): SS MM HH

# $ERRDSP—17417/X'4409'

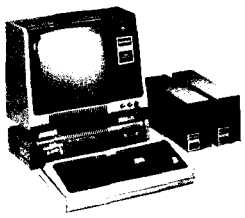
This routine displays a TRSDOS error message determined by the contents of the accumulator (A). This register contains an error code (0 = no error) after completion of any system routine.

## Entry Conditions

A = TRSDOS error code (see Table at the end of this section). In a TRSDOS error code, bits 6 and 7 are normally reset (off). So $ERRDSP interprets them as controls.

| Bit # | Set | Not Set (Normal Condition) |
|---|---|---|
| 7 | Return to caller upon completion | Return to TRSDOS upon completion |
| 6 | Give detailed error message | Give error number only |

CALL $ERRDSP





### Exit Conditions

None

### Sample Use

```
     CALL    $SYSRTN          ; ANY SYSTEM ROUTINE
     JR      Z,OKGO           ; CHECK FOR ERROR
; THE FOLLOWING INSTRUCTION SETS BIT 6 (DETAILED
; ERROR MESSAGE) AND BIT 7 (RETURN TO CALLER AFTER
; DISPLAYING MESSAGE
     OR      C0H              ; BINARY 11000000
     CALL    $ERRDSP          ; NOW CALL ERROR DISPLAY
; CONTINUE HERE UPON RETURN FROM ERROR DISPLAY
; YOUR ERROR HANDLER MAY GO HERE
;            :
;            :
;            :
OKGO  ;CONTINUATION AFTER $SYSRTN WITH NO ERROR
```

## $DSPDIR—17577/X'44A9'

This command displays the directory listing of non-protected visible double-density files in the specified drive.

### Entry Conditions

(X'442B') = ASCII-coded drive number "0," "1," "2," or "3"
CALL $DSPDIR

### Exit Conditions

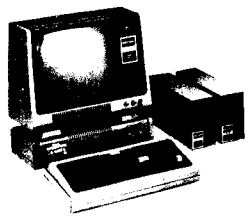
All registers are changed.

## $COMDOS—17553/X'4491'

This routine executes a TRSDOS command and returns to TRSDOS READY.

### Entry Conditions

(HL) = Text of TRSDOS command, terminated by X'0D'
JP $COMDOS



### Exit Conditions

None

## $CMDDOS—17556/X'4494'

This routine executes a TRSDOS command and returns to the caller.

### Entry Conditions

(HL) = Text of TRSDOS command, terminated by X'0D'
CALL $CMDDOS

### Exit Conditions

All registers are changed.

**Caution:** TRSDOS commands will overlay RAM up to X'6FFF'.

## $CMDTXT—17626/X'44DA'

This is the start address of a buffer containing the last command line entered under TRSDOS READY. Using this buffer, your program may recover parameters that were included in the last command line.

For example, given a program named EDITOR/CMD, we want the operator to select an input file name when the program is loaded and executed from TRSDOS READY:

```
TRSDOS READY
EDITOR MYFILE
```

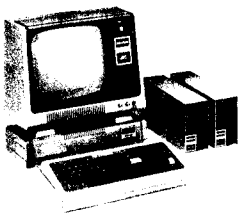
The program, EDITOR, can recover the name of the file in the $CMDTXT buffer.

**Note:** On entry to a program, (HL) = First non-blank character following the program name.

## $MEMEND—16457/X'4049'

This storage location contains the highest address available. It is normally the same as the physical end of RAM, but you may change it for special purposes.

The address is in LSB, MSB sequence.





# TRSDOS Error Codes/Messages

1 CRC **Error During Disk** I/O Diskette contains bad data or diskette had trouble during the input/output self-check. (CRC stands for Cyclic Redundancy Check.)

2 **Disk Drive Not in System** You are attempting to access a disk drive that is not installed in your System.

3 **Lost Data During Disk** I/O An input/output program has occurred and data has been lost because of a problem with the Computer. Have your Computer checked; clean the heads on the disk drives.

4 CRC **Error During Disk** I/O Diskette contains bad data or diskette had trouble during intput/output self-check. (CRC stands for Cyclic Redundancy Check.)

5 **Diskette Sector Not Found** The Computer cannot find recognizable data on the specified diskette. Reformat the diskette.

6 **Disk Drive Hardware Fault** Not used for Model I double-density. (Note: This error, even though not used, is listed in the directory.)

7 **File Already in Directory** A file with the same filename already exists on the specified diskette's directory.

8 **Disk Drive Not Ready** You are attempting to access a disk drive that is not ready for input/output. Probable causes: Drive not connected to System or disk drive's door is open.

9 **Illegal** I/O **Attempt** Not used for Model I double-density. (Note: This error, even though not used, is listed in the directory.)

10 **Required Command Parameter Not Found** The Computer must have additional required information to perform the specified input/output function.

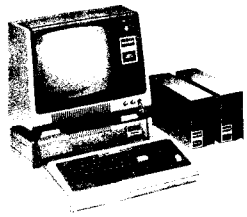
11 **Illegal Command Parameter** You are attempting to use an unrecognized command parameter.

12 **Time Out on Disk Drive** Not used for Model I double-density. (Note: This error, even though not used, is listed in the directory.)

13 I/O **Attempt to Non-System Disk** You are attempting to access a diskette that has not been formatted as a double-density diskette.

14 **Write Fault on Disk** I/O An input problem has occurred because of a problem with the Computer. Check Computer; clean the heads on the disk drives.

15 **Write-Protected Disk** You are attempting to store information on a diskette that is write-protected.



16 **Illegal Logical File Number** You are attempting to input a file number that is greater than the number of files the directory can hold. Use another diskette.

17 **Directory Read Error** A diskette I/O error occurs as the Computer attempts to read the diskette's directory.

18 **Directory Write Error** A diskette I/O error occurs as the Computer attempts to write to the diskette's directory.

19 **Invalid Filename** The file specification does not conform to the syntax rules for valid file specification.

20 **GAT Read Error** Diskette output error during attempt to Read double-density system information from directory track. (GAT stands for Granule Allocation Table which is located on the directory.)

21 **GAT Write Error** Diskette input error during attempt to Write double-density system information onto directory track. (GAT stands for Granule Allocation Table which is located on the directory.)

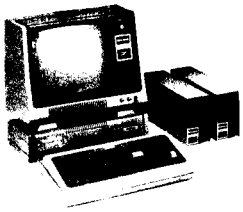
22 **HIT Read Error** Diskette output error during attempt to Read double-density system information from directory track. (HIT stands for Hash Index Table which is located on the directory.)

23 **HIT Write Error** Diskette input error during attempt to Write double-density system information onto directory track. (HIT stands for Hash Index Table which is located on the directory.)

24 **File Not Found** Reference was made in a LOAD, KILL or OPEN statement to a file which did not exist on the specified diskette.

25 **File Access Denied** You are attempting to access a file with the incorrect password (or you're using no password at all). To acces the file, specify the correct password.

26 **Directory Space Full** All directory storage space on the diskette has been used.





27 **Disk Space Full** All storage space on the diskette has been used. KILL unneeded files or use a formatted, non-full diskette.

28 **Attempt to Read Past EOF** You are attempting to read past the end of file.

29 **Attempt to Read Outside of File Limits** You are attempting to put more data in the file than the file can hold.

30 **No More Extents Available** Not used for Model I double-density. (Note: This error, even though not used, is listed in the directory.)

31 **Program Not Found** The Computer cannot find the specified program.

32 **Invalid Drive Number** You have specified an incorrect diskette drive number.

33 **File Unrecoverable** This error is called "unkill" error because you are attempting to access a file that is no longer recoverable since the previously allocated space for this file has now been written to.

34 **Attempt to Use Non-Program File as a Program** You have entered an illegal program format or bad load memory address.

35 **Memory Fault During Program Load** Data is being loaded into a bad load memory address or being loaded to non-existing RAM.

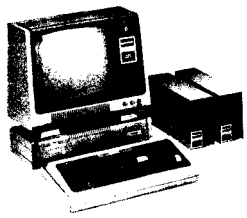
36 ****Undefined Error**** Reserved for future use.

37 **Improper Density Diskette** You are attempting to use a single-density diskette in your double-density system.

38 I/O **Attempt to Unopen File** You are attempting to read or write to a file that is not Opened.

39 **Invalid Command Parameter** You are attempting to use a library command that has wrong parameters for one of the library's options.

40 **Attempt to Open File Already Open** An attempt was made to Open a file that was already Open. This error is also output if a KILL statement is given for an Open file.



# ROM Subroutines

The Level II BASIC ROM contains many subroutines that can be called by a Z-80 program; many of these can be called by a BASIC program via the USR function. Each subroutine will be described in the format given below.

**1. $NAME — Entry Address**

**2. Function Summary**

**3. Description of Function**

**4. Entry Conditions**

**5. Exit Conditions**

**6. Sample Program**

**Notes:**

**1.** The subroutine name is only for convenient reference. It is not recognized by the Computer. The $- prefix reminds you that it is a convenience name only.

The entry address is given in decimal/hexadecimal form. (The hexadecimal address will be given in this form: X'0000.') This is the address you use in a Z-80 CALL. BASIC programmers store this address in the USR definition address (16526-16527).

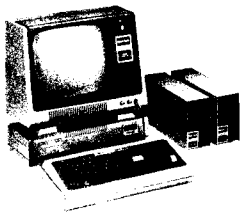
**4, 5.** Entry and exit conditions are given for Z-80 programs. If a Z-80 register is not mentioned here, then you can assume it is unchanged by the subroutine.

**6.** Sample Program fragments are given in Z-80 Assembly Language and, where appropriate, in BASIC.

Here are the subroutines, arranged according to function. In the following pages, they are arranged alphabetically.

## System Control

| | |
|---|---|
| $DELAY | Delay for a specified interval |
| $INITIO | Initialize all I/O Drivers |
| $READY | Jump to Level II "Ready" |
| $RESET | Reset Computer |





## Cassette I/O

| | |
|---|---|
| $ASTBLK | Blinks asterisk in upper-right corner of video |
| $CASSON | Turn on cassette drive |
| $CSHIN | Search for leader and sync byte |
| $CSHWR | Write leader and sync byte |
| $CSIN | Input a byte |
| $CSOFF | Turn off cassette drive |
| $CSOUT | Write a byte to cassette |

## Keyboard Input

| | |
|---|---|
| $KBCHAR | Get a character if available |
| $KBLINE | Wait for a line |
| $KBWAIT | Wait for a character |

## Printer Output

| | |
|---|---|
| $PRCHAR | Print a character |

## Video Display Output

| | |
|---|---|
| $VDCHAR | Display a character |
| $VDCLS | Clear the screen |

# $ASTBLK—556/X'022C'

When called, this subroutine causes the asterisks in the upper-right corner to blink. The blinking asterisks indicate a "good" load from cassette.

## Entry Conditions

Reg. A is destroyed.

## Exit Conditions

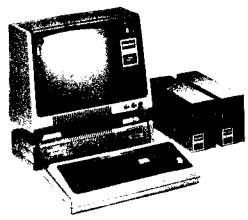
None

# $CASSON—530/X'0212'

This subroutine is used to turn on the cassette drive before reading from or writing to the cassette.

## Entry Conditions

A = Binary-coded cassette drive numbers



0 = cassette drive #1
1 = cassette drive #2

## Exit Conditions

None

**Note:** For Z-80 sample program, see $CSHIN.

# $CSHIN—662/X'0296'

## Search for Cassette Header and Sync Byte

Each cassette "record" begins with a header consisting of a leader sequence, synchronization byte and displays two asterisks in the upper-right corner of the video. $CSHIN begins searching for this header information. The subroutine returns to the calling program after the sync-byte has been read.

## Entry Conditions

None

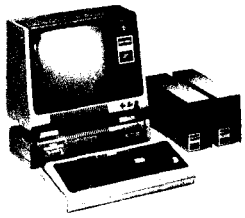
## Exit Conditions

A is altered. All other registers are unchanged.

## Sample Z-80 Programming

The following program reads the tape created by the $CSHWR sample program.

```
01C9          00100 VDCLS   EQU   01C9H    ;VDCLS ADDRESS
0033          00110 VDCHAR  EQU   0033H    ;VDCHAR ADDRESS
4467          00120 VDLINE  EQU   4467H    ;VDLINE TRSDOS ADDRESS
0212          00130 CASSON  EQU   0212H    ;CASSON ADDRESS
0049          00140 KBWAIT  EQU   0049H    ;KBWAIT ADDRESS
0296          00150 CSHIN   EQU   0296H    ;CSHIN ADDRESS
0235          00160 CSIN    EQU   0235H    ;CSIN ADDRESS
402D          00170 JP2DOS  EQU   402DH    ;JUMP TO DOS ADDRESS
01FB          00180 CSOFF   EQU   01FBH    ;CSOFF ADDRESS
              00190 ;READ A MESSAGE FROM TAPE & STOP ON CAR-RET'N
8000          00200         ORG   8000H
8000 CDC901   00210         CALL  VDCLS    ;CLEAR SCREEN
8003 3E0D     00220         LD    A,0DH
8005 CD3300   00230         CALL  VDCHAR   ;SKIP A LINE
8008 213180   00240         LD    HL,MSG0  ;(HL)=CASSETTE PROMPT
```





```
800B CD6744 00250          CALL VDLINE
800E CD4900 00260          CALL KBWAIT   ;WAIT FOR ANY KEY
8011 215880 00270          LD   HL,TXT   ;(HL)=256 BYTE BUFFER
8014 F3     00280          DI            ;DISABLE INTERUPTS
8015 CD1202 00290          CALL CASSON   ;TURN ON CASSETTE PLAYER
8018 CD9602 00300          CALL CSHIN    ;FIND START OF RECORD
801B CD3502 00310 LOOP     CALL CSIN     ;INPUT A BYTE
801E 77     00320          LD   (HL),A   ;STORE IT
801F 23     00330          INC  HL       ;POINT TO NEXT LOC.
8020 FE0D   00340          CP   0DH      ;WAS LAST BYTE=CAR RET'N
8022 20F7   00350          JR   NZ,LOOP  ;IF NOT, GET NEXT BYTE
8024 CDF801 00360          CALL CSOFF    ;IF YES, TURN OFF CASSETTE
8027 FB     00370          EI            ;ENABLE INTERUPTS
8028 215880 00380          LD   HL,TXT   ;DISPLAY THE MESSAGE
802B CD6744 00390          CALL VDLINE
802E C32D40 00400          JP   JR2DOS   ;JUMP TO TRSDOS READY
8031 50     00410 MSG0     DEFM 'PREPARE TAPE TO PLAY AND PRESS ANY KEY'
8057 0D     00420          DEFB 0DH
0100        00430 TXT      DEFS 256      ;STORAGE FOR TAPED MESSAGE
8000        00440 END      8000 H
```

# $CSHWR—647/X'0287'

## Write Leader and Sync Byte

Each cassette "record" begins with a header consisting of a leader sequence and a synchronization byte. $CSHWR writes out this header.

## Entry Conditions

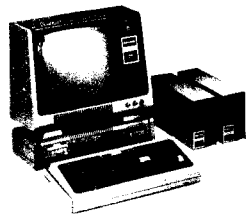
None

## Exit Conditions

A is altered.

## Sample Z-80 Programming

```
01C9        00100 VDCLS  EQU  01C9H    ;VDCLS ADDRESS
0033        00110 VDCHAR EQU  0033H    ;VDCHAR ADDRESS
4467        00120 VDLINE EQU  4467H    ;VDLINE ADDRESS
0049        00130 KBWAIT EQU  0049H    ;KBWAIT ADDRESS
0040        00140 KBLINE EQU  0040H    ;KBLINE ADDRESS
0212        00150 CASSON EQU  0212H    ;CASSON ADDRESS
0264        00160 CSOUT  EQU  0264H    ;CSOUT ADDRESS
01F8        00170 CSOFF  EQU  01F8      ;CSOFF ADDRESS
```



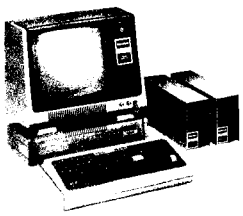
```
402D          00180 JP2DOS  EQU   402DH     ;JUMP TO DOS ADDRESS
0287          00190 CSHWR   EQU   0287H     ;CSHWR ADDRESS
              00200 ;INPUT A KEYBOARD MESSAGE AND WRITE IT TO CASSETTE
8000          00210         ORG   8000H
8000 CDC901   00220         CALL  VDCLS
8003 3E0D     00230 LOOP1   LD    A,0DH     ;CARRIAGE RETURN
8005 CD3300   00240         CALL  VDCHAR    ;SKIP TO NEXT DISPLAY LINE
8008 214080   00250         LD    HL,MSG1   ;PROMPT MESSAGE
800B CD6744   00260         CALL  VDLINE    ;DISPLAY IT
800E 218480   00270         LD    HL,TXT1   ;256 BYTE BUFFER
8011 06FF     00280         LD    B,255     ;MAX OF 255 CHARACTERS
8013 CD4000   00290         CALL  KBLINE    ;GET A LINE FROM KYBD
8016 38EB     00300         JR    C,LOOP1   ;LOOP IF <BREAK> WAS PRESSED
8018 3E0D     00310         LD    A,0DH
801A CD3300   00320         CALL  VDCHAR    ;SKIP A LINE
801D 215280   00330         LD    HL,MSG2   ;CASSETTE PROMPT
8020 CD6744   00340         CALL  VDLINE
8023 CD4900   00350         CALL  KBWAIT    ;WAIT UNTIL A KEY IS PRESSED
8026 F3       00360         DI              ;DISABLE INTERUPTS
8027 CD1202   00370         CALL  CASSON    ;TURN ON CASSETTE
802A CD8702   00380         CALL  CSHWR     ;WRITE CASSETTE HEADER
802D 218480   00390         LD    HL,TXT1   ;(HL)=MESSAGE
8030 7E       00400 LOOP2   LD    A,(HL)    ;A=ASCII BYTE
8031 23       00410         INC   HL        ;POINT TO NEXT BYTE
8032 CD6402   00420         CALL  CSOUT     ;WRITE LAST BYTE TO TAPE
8035 FE0D     00430         CP    0DH       ;WAS IT A CARRIAGE RET'N?
8037 20F7     00440         JR    NZ,LOOP2  ;IF NO, THEN GET NEXT BYTE
8039 CDF801   00450         CALL  CSOFF     ;IF YES, TURN OFF CASSETTE
803C FB       00460         EI              ;ENABLE INTERUPTS
803D C32D40   00470         JP    JP2DOS    ;JUMP TO DOS READY
8040 54       00480 MSG1    DEFM  'TYPE IN A MESSAGE'
8051 0D       00490         DEFB  0DH
8052 4D       00500 MSG2    DEFM  'MESSAGE STORED, PRESS AY KEY WHEN READY
                                    TO RECORD'
8083 0D       00510         DEFB  0DH
0100          00520 TXT1    DEFS  256
8000          00530         END   8000H
```

For a program to read the tape in, see $CSHIN.

# $CSIN—565/X'0235'

## Input a Byte

After completion of $CSHIN, use $CSIN to begin inputting data, one byte at a time.





**Note:** You must call $CSIN often enough to keep up with the baud rate.

### Entry Conditions

None

### Exit Conditions

A = Data byte

### Sample Z-80 Programming

See $CSHIN.

## $CSOFF—504/X'01F8'

### Turn Off Cassette

After writing data to cassette, call this subroutine to turn off the cassette drive.

### Entry Conditions

None

### Exit Conditions

None

### Sample Z-80 Programming

See $CSHWR.

## $CSOUT—612/X'0264'

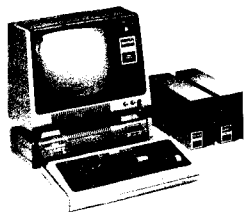
### Output a Byte to Cassette

After writing the header with $CSHWR, use $CSOUT to write the data, one byte at a time.

**Note:** You must call $CSOUT often enough to keep up with the baud rate (either 500 or 1500 baud).

### Entry Conditions

A = Data byte.



## Exit Conditions

None

## Sample Z-80 Programming

See $CSHWR.

# $DELAY—96/X'0060'

## Delay for a Specified Interval

This is a general-purpose routine to be used whenever you want to pause before continuing with a program.

## Entry Conditions

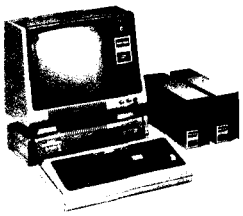
BC = Delay multiplier. Actual delay will be:
BC = N, 21.9MS + (14.6*N)
This includes CALL and RETURN times
When BC = 0000, 65536 is used. This is
the maximum delay
(about one second).

## Exit Conditions

BC and A are altered.

## Sample Z-80 Programming

```
3E20          00100 CENTER EQU  3E20H      ;ROW8; COLUMN 32 OF VIDEO
01C9          00110 VDCLS  EQU  01C9       ;VDCLS ADDRESS
0060          00120 DELAY  EQU  0060H      ;DELAY ADDRESS
402D          00130 JP2DOS EQU  402DH      ;JUMP TO DOS ADDRESS
8000          00140        ORG  8000H
8000 0DC901   00150        CALL VDCLS      ;FIRST CLEAR SCREEN
8003 3E00     00160        LD   A,0H
8005 01FF7F   00170        LD   BC,7FFFH   ;SET 1/2 SEC DELAY FACTOR
8008 32203E   00180 LOOP3  LD   (CENTER),A ;WRITE CHAR. TO VIDEO
800B F5       00190        PUSH AF         ;SAVE LAST CHAR. CODE
800C C5       00200        PUSH BC         ;AND DELAY FACTOR
800D CD6000   00210        CALL DELAY
8010 C1       00220        POP  BC
8011 F1       00230        POP  AF
8012 3C       00240        INC  A          ;NEXT CHAR. CODE
8013 20F3     00250        JR   NZ,LOOP3   ;IF NOT ZERO, DISPLAY IT
```





```
8015 C32D40 00260         JP    JP2DOS     ;ELSE END
8000        00280         END   8000H
```

# $KBCHAR—43/X'002B'

## Get a Keyboard Character if Available

This subroutine checks the keyboard for a character. The character (if any) is not displayed.

## Entry Conditions

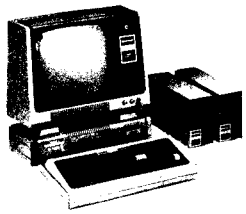
None

## Exit Conditions

A = ASCII Character. If A = 0, no character was available.
DE is altered.

## Sample

```
KBCHAR    EQU    2BH
LOOP      CALL   KBCHAR
          OR     A
          JR     Z,LOOP
```



# $KBLINE—64/X'0040'

## Wait for a Line from the Keyboard

This routine gets a full line from the Keyboard. The line is terminated by a carriage return (X'0D') or (BREAK) (X'01'). Characters typed are echoed to the display.

## Entry Conditions

B = Maximum length of line. When this many characters are typed, no more will be allowed except for (ENTER) or (BREAK)
(HL) = Storage buffer. Length should be B + 1.

## Exit Conditions

C Status = (BREAK) was the terminator.
B = Number of characters entered.
(HL) = Line from keyboard, followed by terminating character.
DE is altered.

## Sample Z-80 Programming

See $CSHWR.

# $KBWAIT—73/X'0049'

## Wait for a Keyboard Character

This routine scans the keyboard until a key is pressed. If (BREAK) is pressed, it will be returned in A like any other key. The character typed is not echoed to the Display.

## Entry Conditions

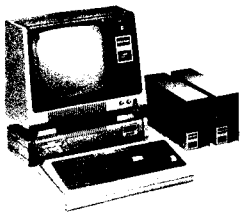
None

## Exit Conditions

A = Keyboard character
DE is altered.

## Sample Z-80 Programming

See $CSHWR.





# $PRCHAR—59/X'003B'

## Output a Character to the Printer

$PRCHAR outputs a byte to the Printer. If the Printer is not available, the system will not return to TRSDOS and must be RESET.

## Entry Conditions

A = ASCII character

## Exit Conditions

DE is altered.

## Sample Z-80 Programming

```
003B          00100 PRCHAR EQU  003BH     ;PRCHAR ADDRESS
402D          00110 JP2DOS EQU  402DH     ;JUMP TO DOS
8000          00120        ORG  8000H
              00130 ;PRINTER DEMO
8000 210F80   00140        LD   HL,TXT4   ;(HL)=SAMPLE TEXT
8003 7E       00150 LOOP5  LD   A,(HL)    ;GET CHAR. INTO A
8004 23       00160        INC  HL        ;POINT TO NEXT CHAR.
8005 CD3B00   00170        CALL PRCHAR    ;PRINT CHAR IN A
8008 FE0D     00180        CP   0DH       ;WAS IT A CARRIAGE RETURN
800A 20F7     00190        JR   NZ,LOOP5  ;IF NO, GET NEXT CHAR.
800C C32D40   00200        JP   JP2DOS    ;IF YES, QUIT
800F 54       00210 TXT4   DEFM 'THIS SENTENCE WILL BE PRINTED'
802C 0D       00220        DEFB 0DH
8000          00230        END  8000H
```

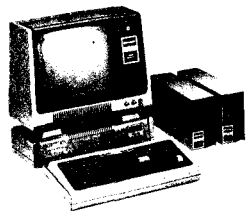
# $READY—6681/X'1A19'

## Jump to Level II BASIC "Ready"

To exit from a machine-language program into BASIC's immediate mode, jump to $READY (don't call it).

## Entry Conditions

None



### Exit Conditions

None

## $RESET—0/X'0000'

### Jump to RESET

Jump to this address to re-initialize the entire system. If a disk controller is present, the Computer will attempt to load TRSDOS. To prevent this from happening, the operator must hold down (BREAK) before this jump is executed.

### Entry Conditions

None

### Exit Conditions

None

## $VDCHAR—51/X'0033'

### Display a Character

This subroutine displays a character at the current cursor location.

### Entry Conditions

A = ASCII character

### Exit Conditions

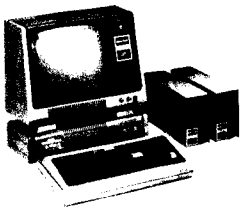
DE is altered.

### Sample Z-80 Programming

See $CSHIN.

## $VDCLS—457/X'01C9'

### Clear the Video Display Screen



# TRS-80 MODEL I DISK SYSTEM

## Entry Conditions

None

## Exit Conditions

All registers are altered.

## Sample Z-80 Programming

See $CSHWR.

## Technical Note

The Sector Register (Address 37EEH) on the Floppy Disk Controller not only allows double-density operation without modification to the Expansion Interface, but it also serves as a control register for the Double-Density Adapter. The control register is enabled by data bits 7, 6, and 5 whenever these bits are not 0. The following combinations allow configuration changes as follows:

| D7 | D6 | D5 | Function |
|---|---|---|---|
| 1 | 0 | 0 | Set Double-Density Mode |
| 1 | 0 | 1 | Set Single-Density Mode |
| 1 | 1 | 0 | Disable Precompensation |
| 1 | 1 | 1 | Enable Precompensation (typically used on tracks greater than 22) |

For Assembly Language:

```
LDL A,80H          Set Double-Density Mode
LD (37EEH),A       Write to Control Register
```

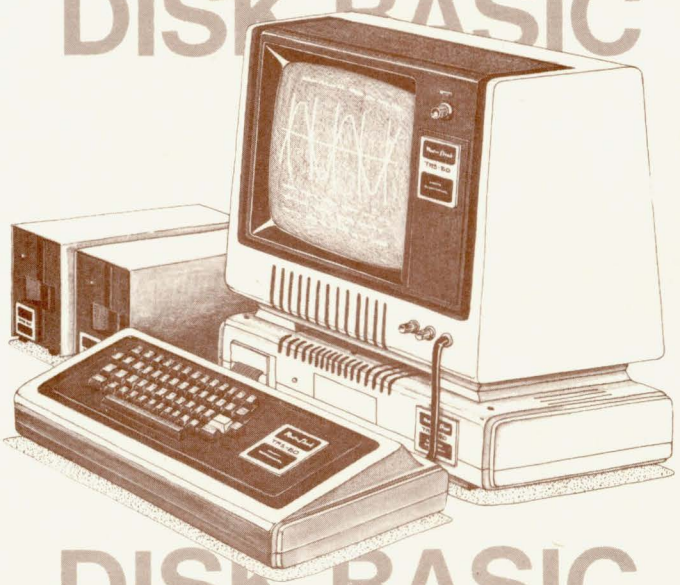
For Basic:

```
POKE 14318,160     Set Single-Density Mode
```

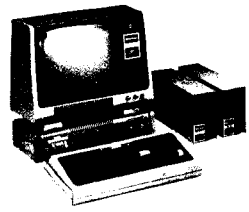
Any data previously written to the Sector register will be destroyed and must be rewritten. Any value written to the Sector Register less than 80H will have no effect on the Double-Density Adapter. Upon Power-up or Reset, the board is configured to Single-Density and precompensation is disabled.

For more information on Single-Density FDC programming, refer to the Model I Expansion Interface Service Manual. For Double-Density FDC programming information, refer to the Model II Technical Reference Manual.

DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC



DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC



# 5/Disk BASIC

## Start-Up

Under TRSDOS READY, type:

```
BASIC (ENTER)
```

TRSDOS will load BASIC and begin the "initialization dialog."

If you want to *recover* a Disk BASIC program after returning to TRSDOS for a DIR or other TRSDOS command, use this command under TRSDOS READY:

```
BASIC * (ENTER)
```

You will go directly to BASIC's READY mode without any initialization dialog. If you had a program in memory, it should still be there. You may not be able to run the program. To be safe, you should immediately save the program, go to TRSDOS, then start BASIC again (no asterisk).

**Note:** If you have overlaid user memory while in TRSDOS, your program will be erased. In such a case, you should not restart BASIC, but should use the normal BASIC start-up procedure.

## Initialization

When you start Disk BASIC, you are first asked, `HOW MANY FILES?`. This lets you specify the maximum number of files that will be "open" or in use at one time. (See OPEN.) Type in an appropriate number and press (ENTER), or simply press (ENTER) and BASIC will provide for three files.

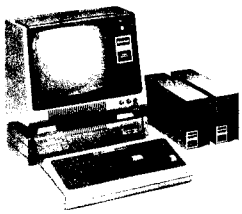
For example, if your program requires one input file and one output file, you should ask for two files.

**Note:** Normally, BASIC will give all your data files a record length of 256. (See **File Access Techniques**.) If you wish to set the record length of each file *individually*, use the suffix V for "Variable" after the number of files. For example,

```
HOW MANY FILES? 3V (ENTER)
```

tells BASIC to give you three **file-buffers**, and to let you set the record length of each file when that file is first opened.

**Note:** Disk BASIC automatically creates a buffer for loading, saving, and merging BASIC programs. This buffer exists in RAM below any data file buffers you may request. It is always available for program I/O, regardless of how you answer the FILES? question.





After you answer the FILES question, BASIC will ask: MEMORY SIZE? Simply press (ENTER) without typing a number. You will then have the maximum amount of RAM available for use by BASIC.

If you will want to load and use machine-language programs or routines, you will have to protect your BASIC memory from these machine-language programs.

In such a case, respond with the highest memory address (in decimal form) you want BASIC to use for storing and executing your BASIC programs. Addresses above the number you specify will then be protected from use by BASIC.

## Example:

```
MEMORY SIZE? 32000 (ENTER)
```

causes BASIC to protect addresses above 32000. If you have 16K of RAM, this means that you'll have 32767-32000 = 767 bytes protected for storing your machine-language routines.

After you answer the MEMORY SIZE question, Disk BASIC will display the following information:

1. Which version of Disk BASIC you are using
2. Copyright information and creation date
3. The number of free bytes available
4. The number of concurrent files you have requested

To exit from Disk BASIC and return to the TRSDOS READY mode, type:

```
CMD"S" (ENTER)
```

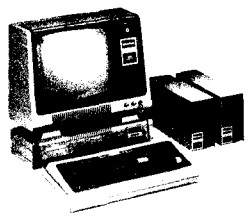
This results in a normal return to TRSDOS, without re-initialization of the system. You may recover your program if you haven't changed user memory while in TRSDOS by typing BASIC *.

# BASIC *

Once you are in Disk BASIC, it is sometimes necessary to exit from it in the middle of developing or running a program. For example you may want to use TRSDOS to find out what files you have saved using DIR, then return to your BASIC program.

To do this you will need to use the BASIC * command to return to Disk BASIC. By typing the asterisk after the command, you will be able to return to Disk BASIC. If you had a program in memory, it should still be there. You may not be able to run the program. To be safe, you should save your program before going to TRSDOS and start BASIC (without an asterisk) again.

**Note:** A space is required between BASIC and the asterisk (*).



## Example

Suppose you are typing in a BASIC program, and you want to return to TRSDOS to examine the diskette directory. Then type:

```
CMD"S"  (ENTER)
DIR     (ENTER)
```

After examining the directory, you can return to BASIC and recover your program by typing:

```
BASIC *  (ENTER)
```

BASIC will skip the FILES? and MEMORY SIZE? questions and return to the prompt:

```
READY
```

You can now LIST the program to be sure it was recovered. Again, we suggest you save your program before going to TRSDOS, as a safeguard for your programs.

**Note:** Do not use the BASIC * command when you have no BASIC program in memory. This might make the System "hang-up," requiring a Reset or power-off, power-on. Also, if you have overlaid user memory while in TRSDOS, your program will be erased. In such a case you should not restart BASIC, but should use the normal BASIC start-up procedure.

## Options for Loading BASIC

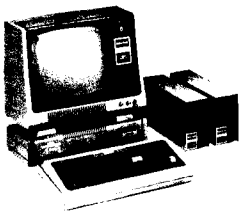
There are several other ways to start up BASIC, as summarized below:

**BASIC *program* –F:*files* –M:*address***

***program* is a TRSDOS file specification for a BASIC program. After start-up, BASIC will run it. If *program* is omitted, BASIC will start-up in the command mode.**

**–F:*files* tells BASIC the maximum number of files that may be Open at once. *files* is a number from 0 to 15. If –F:*files* is omitted, maximum is set to 3.**

**–M:*address* tells BASIC not to use memory above *address*. *address* is a decimal number. If –M:*address* is omitted, BASIC uses all memory up to TOP.**





The options allow you to specify any or all of the following:

- A program to run after BASIC is started.
- Maximum number of data files that may be Open at once. The larger the number of files, the less area available for storing and executing your programs. Each fixed-length file uses 360 bytes and each variable-length file uses 616 bytes of memory.
- Highest address to be used by BASIC during program execution. Omit this unless you are going to call machine-language subroutines.

## Examples

```
TRSDOS READY
BASIC
```

Tells BASIC to prompt the user for the number of files and the amount of memory to be used.

```
TRSDOS READY
BASIC -F:1
```

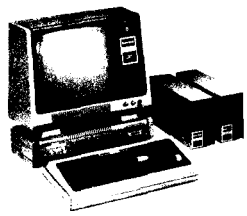
Tells BASIC to allow one file to be open at an given time and to allow all available memory for use.

```
TRSDOS READY
BASIC -M:32000
```

BASIC allows three fixed length files to be open, and 32000 is the highest address it will use during program execution.

```
TRSDOS READY
BASIC PAYROLL -F:3
```

BASIC will start up, load and run the BASIC program PAYROLL; three data files can be Opened, and BASIC can use all available memory.



# Enhancements to Model I BASIC

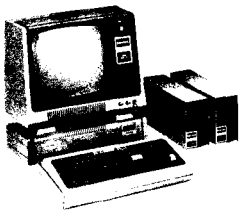
Disk BASIC adds many features which are not disk-related. They are listed below along with abbreviated descriptions. Detailed descriptions follow in alphabetical order.

| | |
|---|---|
| &H | Hexadecimal-constant prefix |
| &O | Octal-constant prefix |
| Abbreviations | Many commands have abbreviations |
| CMD"A" | Return to TRSDOS with error message |
| CMD"B" | Enable/Disable (BREAK) |
| CMD"C" | Delete spaces and remarks from a program (compression) |
| CMD"D" | Display directory for specified drive |
| CMD"E" | Display previous TRSDOS error |
| CMD"I" | Return a command to TRSDOS |
| CMD"J" | Convert calendar date |
| CMD"K" | Turn ON/OFF clock display |
| CMD"L" | Load Z-80 subroutine or program file into RAM |
| CMD"O" | Alphabetizes (sorts) a string array only |
| CMD"P" | Check printer status |
| CMD"R" | Start real-time clock interrupts |
| CMD"S" | Normal return to TRSDOS (jump to EXIT routine) |
| CMD"T" | Turn off real-time clock interrupts |
| CMD"X" | Cross-reference of reserved words, string variables, or strings in a program |
| CMD"Z" | Duplicate output to Display and Printer |
| DEF FN | Define BASIC-statement function |
| DEF USR | Define the entry point for an external machine-language routine |
| INSTR | Instring function; find the substring in the target string |
| LINE INPUT | Input a line from keyboard |
| MID$ = | Replace portion of the target string (used on left of equals sign) |
| NAME | Renumber a program in RAM |
| TIME$ | Get value of real-time clock |
| USR*n* | Call external routine ($n = 0,1,2, \ldots ,9$) |

## &H and &O (hex and octal constants)

Often it is convenient to use hexadecimal (base 16) or octal (base 8) constants rather than their decimal counterparts. For example, memory addresses and byte values are easier to manipulate in hex form. &H and &O let you introduce such constants into your program.

&H and &O are used as prefixes for the numerals that immediately follow them:





**&H*dddd***

***dddd* is a 1 to 4 digit sequence composed of hexadecimal numerals 0,1,...9,A,B,...,F.**

**&O*ddddd***

***ddddd* is a sequence of octal numerals 0,1,...,7 and &O*ddddd*< = 177777 octal.**

**Note: The O can be omitted from the prefix &O. Therefore &O*ddddd* = &*ddddd*.**

The constants always represent signed integers. Therefore any hex number greater than &H7FFF, or any octal number greater than &O77777, will be interpreted as a negative quantity. The following table illustrates this:

| Octal | Hex | Decimal |
|---|---|---|
| &1 | &H1 | 1 |
| &2 | &H2 | 2 |
| &77777 | &H7FFF | 32767 |
| &100000 | &H8000 | −32768 |
| &100001 | &H8001 | −32767 |
| &100002 | &H8002 | −32766 |
| &177776 | &HFFFE | −2 |
| &177777 | &HFFFF | −1 |

Hex and octal constants cannot be typed in as responses to an INPUT prompt or be contained in a DATA statement. Often the hex or octal constant must be enclosed in parentheses to prevent a syntax error from occurring.

## Examples

```
PRINT &H5200, &051000
```

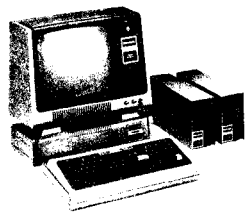
prints the decimal equivalent of the two constants (both equal 20992).

```
POKE &H3C00, 42
```

puts decimal 42 (ASCII code for an asterisk) into video memory address hex 3C00.

# Model I Disk BASIC Abbreviations

| Abbreviation | Meaning |
|---|---|
| ↑ | List Previous Program Line |
| ↓ | List Next Program Line |



| | |
|---|---|
| . | List Current Program Line |
| , | Edit Current Program Line |
| SHIFT ← | List First Program Line |
| SHIFT → Z | List Last Program Line |
| L*xx* | List Program Line *xx* |
| E*xx* | Edit Program Line *xx* |
| D*xx* | Delete Program Line *xx* |
| A*xxx,xxxx* | Automatic Line Numbering Beginning at Line *xxx*, Incrementing by *xxxx*. |

# CMD "A"
# Return to TRSDOS

**CMD"A"**

This command allows you to return to TRSDOS with an error message:

```
OPERATION ABORTED
```

## Example

After an input/output error occurs in a BASIC program, you might want to exit to TRSDOS and print a message.

```
CMD"A"
```

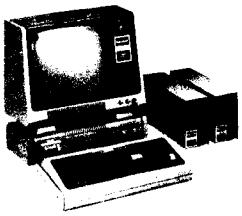
the following will be displayed:

```
OPERATION ABORTED
TRSDOS READY
*****************
```

# CMD "B"
# Enable/Disable BREAK Key

**CMD"B", *"switch"***

***switch* is either ON or OFF. *switch* must be enclosed in quotation marks.**





This command enables or disables the (BREAK) key. While the function is "OFF," the (BREAK) key will be ignored except during cassette or printer output or during serial input/output.

The (BREAK) key will remain disabled even after the program has ended. To enable the (BREAK) key, use the CMD"B","ON" command. Returning to TRSDOS via the CMD"S" or CMD"I" commands will also enable the (BREAK) key.

**Note:** AUTO command automatically enables (BREAK) key.

## Examples

```
CMD"B","OFF"
```

Disables the (BREAK) key.

```
CMD"B","ON"
```

Returns the (BREAK) key to its normal function.

# CMD "C"
# Compress Program

**CMD "C", *options***

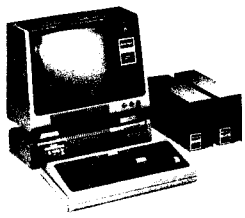
***options* may be either R (delete remarks) or S (delete spaces). If both options are omitted, remarks and spaces are deleted. If only one is omitted, only the specified action is taken.**

This command allows you to compress a program so that it requires less RAM and less storage space on diskette. You can elect to remove all remark statements (beginning with REM or ') or to delete all spaces between BASIC keywords. Spaces inside quotes will *not* be deleted. The REM will not be removed if it is the only command on a line.

## Example

Your program reads as follows:

```
850 RESTORE: ON ERROR GOTO 800  'DOG PROGRAM
860 READ COMPANY$               'PET STORE
```



```
870 PRINT RIGHT$(COMPANY$,2),: GOTO 860
880 END
```

If you want to delete the Remarks (lines 850 and 860), type in the command:

```
CMD"C",R
```

and the program will now read:

```
850 RESTORE: ON ERROR GOTO 800
860 READ COMPANY$
870 PRINT RIGHT$(COMPANY$,2),:GOTO 860
880 END
```

If you then wanted to delete the spaces, type in:

```
CMD"C",S
```

and the program would read:

```
850 RESTORE:ONERRORGOTO800
860 READCOMPANY$
870 PRINTRIGHT$(COMPANY$,2),:GOTO860
880 END
```

You could obtain the same results by typing:

```
CMD"C"
```

**Note:** Always provide the closing quotes on string literals in your BASIC program. Otherwise CMD"C" may not function properly. For example, in

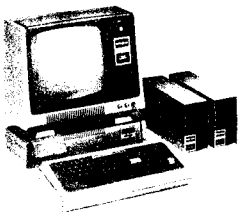
```
10 PRINT "THIS IS A TEST"
```

the second quote should be used even though omitting it will not cause an error.

# CMD"D"
# Display the Directory Listing of a Specified Drive

**CMD"D:*d*"**

***d* is the drive specification**





By entering the command CMD"D:*d*", you can obtain a specified diskette's directory listing from BASIC without returning to TRSDOS. Only unprotected, visible files in double density will be displayed. The drive specification is not optional and must be specified for all drives, including Drive 0. The directory that appears using CMD"D" is not the same format as the TRSDOS library command DIR, and there is no option for the printer.

### Example

If you type in the command:

```
CMD"D:1"
```

the directory listing for Drive 1 will be displayed.

# CMD"E"
# Display Previous TRSDOS error

**CMD"E"**

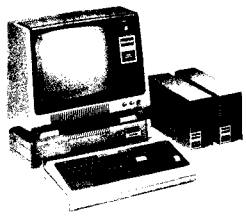
This command displays the last TRSDOS error message. If no errors have occurred prior to the command, the message NO ERROR FOUND will be displayed.

### Example

If you have a two-drive system (0 and 1) and you type:

```
SAVE "PROGRAM:3"
```

Disk BASIC will return a DISK I/O ERROR. To find out what kind of I/O error occurred, type: CMD"E" (ENTER) and Disk BASIC will return with DISK DRIVE NOT IN SYSTEM.



# CMD"I"
# Execute TRSDOS Commands from Disk BASIC

**CMD"I", *command***

***command* is a string expression containing a TRSDOS command or a Z-80 program file name. If it is a string constant, it must be enclosed in quotes.**

You may execute TRSDOS commands directly from BASIC by using CMD"I".

This is similar to CMD"S", except that it lets you include a command or Z-80 program for TRSDOS to execute.

## Example

```
CMD"I","PROGRAM"
```

returns you to TRSDOS and executes the program file PROGRAM.

```
CMD"I",A$
```

returns you to TRSDOS and executes the command contained in A$.

# CMD"J"
# Calendar Date Conversion

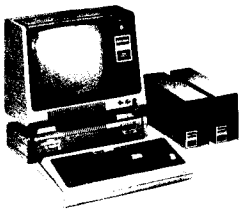
**CMD"J", *input, output***

***input* is a string expression containing the date to be converted. Its contents may be in either of two formats:**

**A) *mm/dd/yy***

**B) *-yy/ddd***

**Format A gives the date in month-day-year sequence. Format B gives the day of the year (from 1 to 365 or 366 for leap years). In format B, the hyphen is required.**





***output* is a string variable to contain the *converted* date. If *input* is in format A, *output* will contain the day of year. If *input* is in format B, *output* will contain the date in format A.**

This command converts dates back and forth between two formats: the standard month, day, year, sequence; and a year, day of year, sequence. The content of the source string determines which way the conversion goes.

## Example

```
CMD"J", "11/30/80", D$
```

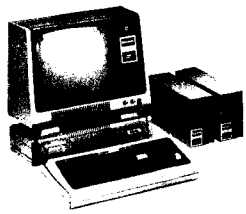
Returns the day of the year in D$.

```
CMD"J", "-79/300", D$
```

Returns the month, day, year, equivalent in D$ (the date for the 300th day of 1979).

## Sample Program

```
10 CLEAR 50
20 LINE INPUT"ENTER FIRST DATE (MM/DD/YY) "; FD$
30 LINE INPUT"ENTER SECOND DATE (MM/DD/YY) ";SD$
40 CMD"J", FD$, D1$
50 CMD"J", SD$, D2$
60 Y1 = VAL(RIGHT$(FD$,2))
70 Y2 = VAL(RIGHT$(SD$,2))
80 J1 = VAL(RIGHT$(D1$,3))
90 J2 = VAL(RIGHT$(D2$,3))
100 S1 = Y1*365 + J1
110 S2 = Y2*365 + J2
120 PRINT "THE INTERVAL BETWEEN DATES IS";
130 PRINT ABS(S1-S2); "DAYS ";
140 PRINT "(IGNORING LEAP-YEARS)."
150 INPUT "<ENTER> TO CONTINUE"; A$
160 GOTO 20
```



# CMD"K"
# Turn Clock Display ON/OFF

**CMD"K","*switch*"**

***switch* is either ON or OFF. *switch* must be enclosed in quotes.**

This command is used to turn ON and OFF the Real-Time Clock display in the upper-right corner of the Video Display. When it is ON, the 24-hour time will be displayed and updated once each second, regardless of what program is executing.

**Note:** The Real-Time Clock is always running (except during cassette I/O, see CMD"R" and CMD"T"; or during disk I/O), regardless of whether the display is ON or OFF.

## Example

```
CMD"K",ON"
```

Turns the clock display ON.

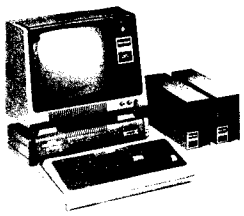
```
CMD"K",OFF"
```

Turns the clock display OFF.

# CMD"L"
# Load Z-80 Routine into RAM

**CMD"L",*routine***

***routine* is a string expression containing a file specification for a Z-80 routine or program created by the DUMP command. If *routine* is a string constant, it must be enclosed in quotes.**





CMD"L" loads a Z-80 (machine-language) routine into RAM. It would normally be used to load a Z-80 subroutine which is to be accessed directly from BASIC.

The Z-80 routine should load into high-RAM and must not overlay the memory protect area reserved when you first entered BASIC (i.e., the MEMORY SIZE? prompt). If you do not overlay BASIC or TRSDOS, control will return to BASIC after the program is loaded.

### Example

The command:

```
CMD"L",PROG"
```

will load a program file named PROG into RAM.

```
CMD"L",P$
```

will load a program which has been specified as P$.

# CMD"O" Sort ("Order") an Array

**CMD"O",*x*,*array (start)***

***x* is an integer variable containing the number of items to be sorted.**

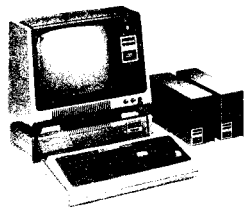
***array (start)* specifies an array element. The *array* contains the data to be sorted, and *start* is the subscript of the first element to be sorted. The *array* must be one-dimensional, string type. The string elements in *array* may be of any length.**

**Note: If *x* is larger than the size of the array, then the results will be unpredictable.**

This command sorts (orders) a one-dimensional string array, i.e., a list. You may sort all or part of the array, depending on the values you give to *x* and *start*.

### Example

```
10 CLEAR 10 * 25 + 50   'ROOM FOR 10 WORDS + EXTRA
20 DIM A$(9)            'LIST OF TEN (0-9)
30 FOR WD = 0 TO 9
```



```
40 PRINT "ENTER WORD #"; WD+1
50 INPUT A$(WD)
60 NEXT WD
70 N%=10: CMD"O", N%, A$(0)
80 PRINT "HERE IS THE SORTED LIST"
90 FOR WD=0 TO 9
100 PRINT A$(WD)
110 NEXT WD
```

# CMD"P"
# Check Printer Status

**CMD"P",*string***

***string* is a string variable**

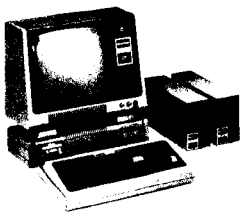
CMD"P" makes it possible for Disk BASIC to check the status of the Printer.

Unlike the video display, the Printer is not always available. It may be disconnected, offline, out of paper, etc. In such cases, when you try to output information to the Printer, the Computer will wait until the Printer becomes available. It will appear to "hang up." To regain control (and cancel the printer operation), press RESET.

Suppose you have a program which uses printer output. If a printer is not available, you don't want the Computer to stop and wait for it to become available. Instead, you may want to print a message such as PRINTER UNAVAILABLE and go on to some other operation.

To accomplish this, you need to check the printer status. CMD"P" can be used to check the printer's status at any time. It returns the contents as an ASCII-coded decimal number. The specific value of this number depends upon the type of printer you are using as well as its status at any particular time. The value may then be printed or examined by the program.

Only the four most significant bits are used in this "status byte." In binary, these must be: "0011" or else the print operation will not be attempted. To check for this "go" condition, AND the status byte with 240 and compare the result with 48. The meaning of each status bit depends on which printer you use. See the Printer's owner's manual for bit designations.





### Example

```
10 CMD"P",X$
20 ST% = VAL(X$) AND 240
30 IF ST% <> 48 THEN PRINT "PRINTER UNAVAILABLE": STOP
40 PRINT "PRINTER AVAILABLE"
50 REM PROGRAM MAY NOW CONTINUE
```

## CMD"R"
## Start Clock (Enable Interrupts)

**CMD"R"**

This command is used to start the Real-Time Clock after a tape input/output operation has been performed. See CMD"T".

### Example

To turn the clock on, type: `CMD"R"` To turn the clock off, type: `CMD"T"`

## CMD"S"
## Return to TRSDOS

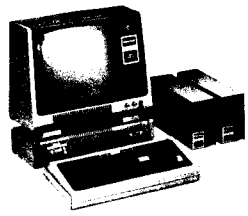
**CMD"S"**

To exit from Disk BASIC, returning control to TRSDOS, simply type the command: `CMD"S"`(ENTER)

To return to BASIC and recover your program, use `BASIC *`. However, recovery will not always be possible. See **BASIC** *.

### Example

The BASIC prompt lets you know you are in Disk BASIC.



```
READY
>
```

To exit, type: `CMD"S"` (ENTER)

and the TRSDOS prompt will appear.

```
TRSDOS READY
```

# CMD"T"
# Stop Clock (Disable Interrupts)

**CMD"T"**

This command turns off the real-time clock. You must execute this command immediately before any BASIC tape input/output operation. Such operations are timing sensitive and cannot allow the interrupt-driven tasks (such as real-time clock, TRACE, and CLOCK-display) to "steal" time.

Here are the commands which must be preceded by execution or CMD"T":

| | |
|---|---|
| CLOAD | CLOAD? |
| INPUT#-1 | CSAVE |
| INPUT#-2 | PRINT#-1 |
| SYSTEM | PRINT#-2 |

After completion of these operations, you can execute a CMD"R" to re-enable interrupts.

## Example

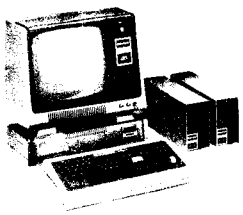
Suppose you want to load in a Level II BASIC program from cassette called "TEST". Before attempting to load it, type in:

`CMD"T"` (ENTER)

to disable the interrupts. You may now load the program by:

`CLOAD "TEST"` (ENTER)

When the program is loaded, type: `CMD"R"` to turn on the clock.





# CMD "X"
# Cross-reference of Program Lines

**CMD "X", *target***

***target* is either a BASIC reserved word (such as PRINT) or a string-literal. If it is a reserved word, it must not be enclosed in quotes; if it is a string-literal, it must be enclosed in quotes.**

This command finds all occurrences of a reserved word or other string literal in the resident program. The "finds" are listed on the display as line numbers.

To search for any BASIC reserved word (including reserved arithmetic operators), use the keyword as-is. To search for anything else (including variable-names and text), enclose the text inside quotes.

For example, suppose you have the following program in memory:

```
10 PRINT "THIS IS A TEST"
20 INPUT "PRESS <ENTER> FOR THE NEXT PRINT MESSAGE"; Z$
30 A = A + 1
40 PRINT "+++++++"
```

`CMD "X", PRINT` will find all occurrences of PRINT, except for cases where PRINT was part of a quoted string: lines 10 and 40.

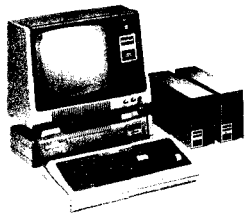
`CMD "X", "PRINT"` will find all occurrences of "PRINT" as a string literal: line 20.

`CMD "X", +` will list line 30, but `CMD "X", "+"` will list line 40. `CMD "X", "A"` will list lines 10, 20, and 30. Notice that variables and text are both treated as string literals.

# CMD "Z"
# Duplicate Output to Video and Printer

**CMD"Z", *"switch"***

***switch* is either ON or OFF. *switch* must be enclosed in quotation marks.**



This command enables or disables dual video/printer output. While the function is "ON," all video output is copied to the printer, and all printer output is copied to the video. (The printer *should* be on-line when you turn dual output "ON.")

Video and printer output may differ due to intrinsic differences in the Printer and video devices.

**Note:** SPOOL must be off or an `ILLEGAL FUNCTION CALL` message will be displayed.

## Examples

```
CMD "Z", "ON"
```

Turns dual video/printer output on.

```
CMD"Z", "OFF"
```

Turns dual video/printer output off.

# DEF FN
# Define Function

**DEF FN *function name (argument-1, . . .) = formula***

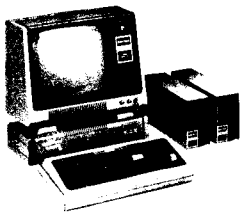
***function name* is any valid variable name.**

***argument-1* and subsequent arguments are used in defining what the function does.**

***formula* is an expression usually involving the argument(s) passed on the left side of the equals sign.**

The DEF FN statement lets you create your own function. That is, you only have to call the new function by name, and the associated operations will automatically be performed. Once a function has been defined with the DEF FN statement, you can call it simply by inserting FN in front of *function name*. You can use it exactly as you might use one of the built-in functions, like SIN, ABS, and STRING$.

The type of variable used for *function name* determines the type of value the function will return. For example, if *function name* is single precision, then that function will return a single-precision value, regardless of the precision of the arguments.





The particular variables you use as arguments in the DEF FN statement *(argument-1, . . .)* are not assigned to the function. When you call the function later, any variable name of the same type can be used.

Furthermore, using a variable as an argument in a DEF FN statement has no effect on the value of that variable. So you can use that particular variable in another part of your program without worrying about interference from DEF FN.

The function can be defined with no arguments at all, if none are required. For example:

```
DEF FNR = RND (90) + 9
```

defines a function to return a random value between 10 and 99.

## Examples

```
DEF FNR(A,B) = A + INT((B - (A - 1)) * RND(0))
```

This statement defines function FNR which returns a random number between integers A and B. The values for A and B are passed when the function is "called," i.e., used in a statement like:

```
Y = FNR(R1, R2)
```

If R1 and R2 have been assigned the values 2 and 8, this line would assign a random number between 2 and 8 to Y.

```
DEF FNL$(X) = STRING$(X, "-")
```

Defines function FNL$ which returns a string of hyphens, X characters long. The value for X is passed when the function is called:

```
PRINT FNL$(30)
```

This line prints a string of 30 hyphens.

Here's an example showing DEF FN used for a complex computation — in double-precision.

```
DEF FNX#(A#, B#) = (A# - B#) * (A# - B#)
```

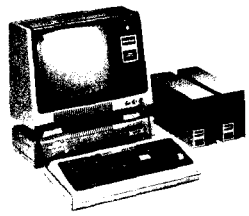
Defines function FNX# which returns the double-precision value of the square of the difference between A# and B#. The values for A# and B# are passed when the function is called:

```
S# = FNX#(A#, B#)
```

We assume that values for A# and B# were assigned elsewhere in the program.

## Sample Program

```
710 DEF FNV(T) = (1087 + SQR(273 + T))/16.52
720 INPUT "AIR TEMPERATURE IN DEGREES CELSIUS"; T
730 PRINT "THE SPEED OF SOUND IN AIR OF" T "DEGREES
    CELSIUS IS" FNV(T) "FEET PER SECOND."
```



# DEFUSR
# Define Point of Entry for USR Routine

**DEFUSR*n* = *address***

***n* equals one of the digits 0, 1,...,9; if *n* is omitted, 0 is assumed. *address* specifies the entry address to a machine-language routine. *address* must be in the range [−32768,32767]. *address* may be any numeric expression or constant from −32768 to 32767.**

DEFUSR lets you define the entry points for up to 10 machine-language routines. In non-Disk BASIC, the addresses were POKEd into RAM. This POKE method cannot be used in Disk BASIC.

## Examples

```
DEFUSR3 = &H7D00
```

assigns the entry point X'7D00', 32000 decimal, to the USR3 call. When your program calls USR3, control will branch to your subroutine beginning at X'7D00'.

```
DEFUSR = (BASE + 16)
```

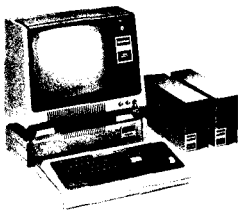
assigns start address (BASE + 16) to the USR0 routine.

**Note:** When decimal addresses are given, they are evaluated as signed two-byte integers. So, for addresses above 32767, use *desired decimal address* − *65536*. See USR*n*.

# INSTR
# Search for Specified String

**INSTR *(position, string 1, string 2)***

***position* specifies the position in *string 1* where the search is to begin. *position* is optional; if it is not supplied, search automatically begins at the first character in *string 1*. (Position 1 is the first character in *string 1*.)**





***string 1* is the string to be searched.**

***string 2* is the substring you want to search for.**

This function lets you search through a string to see if it contains another string. If it does, INSTR returns the starting position of the substring in the target string; otherwise, zero is returned. Note that the entire substring must be contained in the search string, or zero is returned. Also, note that INSTR only finds the first occurrence of a substring at the position you specify.

## Examples

In these examples, A$ = "LINCOLN":

```
INSTR(A$, "INC")
```

returns a value of 2.

```
INSTR (A$, "12")
```

returns a zero.

```
INSTR(A$, "LINCOLNABRAHAM")
```

returns a zero. For a slightly different use of INSTR, look at

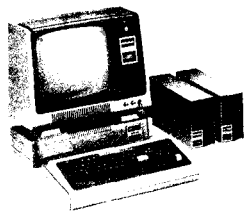
```
INSTR (3, "1232123", "12")
```

which returns 5.

## Sample Program

This program gets search and target text from the keyboard, then locates all occurrences of the target text in the search text. Line 90 is just for "show."

```
10 CLEAR 1000
20 CLS
30 INPUT "SEARCH TEXT"; S$
40 INPUT "TARGET TEXT"; T$
45 CLS
50 C = 0 : P = 1 'P = POSITION, C = COUNT
60 F = INSTR(P,S$,T$)
70 IF F = 0 THEN 120
80 C = C + 1
```



```
90 PRINT @0,LEFT$(S$,F-1) + STRING$(LEN(T$),191) +
   RIGHT$(S$,LEN(S$)-F-LEN(T$)+1)
100 P = F + LEN(T$)
110 IF P <= LEN(S$) - LEN(T$) + 1 THEN 60
120 PRINT "FOUND "; C; "OCCURRENCES"
```

# LINE INPUT
# Input a Line from Keyboard

**LINE INPUT *"prompt" ;variable***

***prompt* is a prompting message.**

***variable* is the name that will be assigned to the line you type in.**

LINE INPUT (or LINEINPUT — the space is optional) is similar to INPUT, except:

- The Computer will not display a question mark when waiting for your operator's input.
- Each LINE INPUT statement can assign a value to just one variable.
- Commas and quotes your operator can use as part of the string input.
- Leading blanks are not ignored — they become part of *variable*.
- The only way to terminate the string input is to press (ENTER).

LINE INPUT is a convenient way to input string data without having to worry about accidental entry of delimiters (commas, quotation marks, colons, etc.). The (ENTER) key serves as the only delimiter. If you want anyone to be able to input information into your program without special instructions, use the LINE INPUT statement.

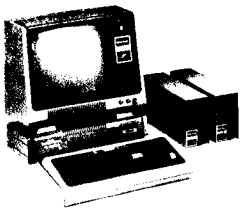
Some situations require that you input commas, quotes and leading blanks as part of the data. LINE INPUT serves well in such cases.

## Examples

```
LINE INPUT A$
```

Input A$ without displaying any prompt.

```
LINE INPUT "LAST NAME, FIRST NAME? ";N$
```





Displays a prompt message and inputs data. Commas will not terminate the input string, as they would in an input statement.

### Sample Program

```
200 REM  CUSTOMER SURVEY
205 CLEAR 1000
207 PRINT
210 LINE INPUT "TYPE IN YOUR NAME "; A$
220 LINE INPUT "DO YOU LIKE YOUR COMPUTER? "; B$
230 LINE INPUT "WHY? "; C$
235 PRINT
240 PRINT A$ : PRINT
250 IF B$= "NO" THEN 270
260 PRINT "I LIKE MY COMPUTER BECAUSE "; C$ :END
270 PRINT "I DO NOT LIKE MY COMPUTER BECAUSE "; C$
```

Notice that when line 210 is executed, a question mark is not displayed after the statement, "Type in your name." Also, notice on line 230 you can answer the question "Why" with a statement full of delimiters, commas and quotes.

# MID$ =
# Replace Portion of String

**MID$ *(oldstring, position, length) = replacement-string***

***oldstring* is the variable-name of the string you want to change.**

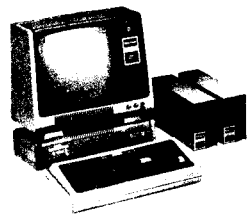
***position* is the numeric expression specifying the position of the first character to be changed.**

***length* is a numeric expression specifying the number of characters to be replaced.**

***replacement-string* is a string expression to replace the specified portion of oldstring.**

**Note: If *replacement-string* is shorter than *length*, then the entire *replacement-string* will be used.**

This statement lets you replace any part of a string with a specified new string, giving you a powerful string editing capability.



Note that the length of the resultant string is always the same as the original string.

## Examples

A$ = "LINCOLN" in the examples below:

```
MID$(A$, 3, 4) = "12345": PRINT A$
```

which returns LI1234N.

```
MID$(A$, 1, 2) = "": PRINT A$
```

which returns LINCOLN.

```
MID$(A$, 5) = "12345": PRINT A$
```

returns LINC123.

```
MID$(A$, 5) = "Ø1": PRINT A$
```

returns LINC01N.

```
MID$(A$, 1, 3) = "***": PRINT A$
```

returns ***COLN.

## Sample Program

```
77Ø CLS: PRINT: PRINT
78Ø LINE INPUT "TYPE IN A MONTH AND DAY MM/DD, "; S$
79Ø P = INSTR(S$, "/")
8ØØ IF P = Ø THEN 78Ø
81Ø MID$(S$, P, 1) = CHR$(45)
82Ø PRINT S$ " IS EASIER TO READ, ISN'T IT?"
```

This program uses INSTR to search for the slash ("/"). When it finds it (if it finds it), it uses MID$ = to substitute a "–" (CHR$(45)) for it.

# NAME
# Renumber the Current Program

**NAME *newline, startline, increment***

***newline* specifies the new line number of the first line to be renumbered. If omitted, 10 is used.**

***startline*** **specifies the line number in the original program where renumbering will start. If omitted, the entire program will be renumbered.**

***increment*** **specifies the increment to be used between each successive line number. If omitted, 10 is used.**

## Examples

```
NAME
```

Renumbers the entire program: 10, 20, 30, . . .

```
NAME 6000,5000,100
```

Renumbers all lines numbered from 5000 up; the first renumbered line will become 6000, and the following lines will be incremented by 100. All line references within your program will be renumbered also.

# TIME$
# Get Value of Real-Time Clock

**TIME$**

TIME$ is a function with no arguments — when executed, it returns a string-value composed of the date and time currently stored in the Real-Time Clock memory area. The string is always 17 characters long and has the following format:

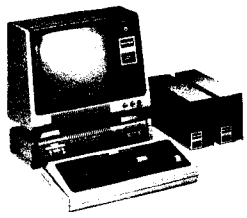
MM/DD/YY␣HH:MM:SS (month/day/year hr:min:sec)

The hour appears in 24-hour form, e.g., 1:30 PM appears as 13:30.

The TIME and DATE are initially set at power-up in TRSDOS.

To reset the time and date, get into the TRSDOS READY mode and use the TRSDOS commands, TIME and DATE, as follows (assume it's 3:30 PM on January 1, 1979):

```
TIME 15:30:00 (ENTER)
DATE 01/01/79 (ENTER)
```



TIME$ can be printed or used internally by your program in dedicated applications.

## Examples

```
1000 IF LEFT$(TIME$,15)="07/04/79 20:00"THEN 2000
1010 GOTO 1000
2000 REM...IT'S 8PM ON JULY 4TH, 1979
2010 REM...START FIREWORKS DISPLAY
.
.
.
```

The following program, Clock, will display the time and date until you press the @ key.

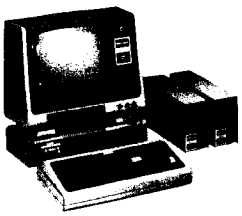
```
100 '          PROGRAM: CLOCK
110  EXAMPLE OF TIME$
120 '
130 CLS: PRINT CHR$ (23)  'GET INTO 32 CHARACTER MODE
140 '
150 ' ****** PRINT TIME AND DATE *******
160 '
170 PRINT @ 264, "THE TRS-80 TIME IS";
180 PRINT @ 458, "DATE: "; LEFT$ (TIME$, 8);
190 PRINT @ 586, "TIME: "; RIGHT$ (TIME$, 8);
200 '
210 ' ***** STOP IF "@" KEY IS DEPRESSED *****
220 '
230 A$=INKEY$: IF A$ = "@" THEN END ELSE 180
```

# USR*n*
# Call to User's External Subroutine

**USR*n* (*nmexp*)**

**where *n* specifies one of ten available USR calls, *n*=0,1,2,...,9. If *n* is omitted, zero is assumed.**

***nmexp* is an integer from −32768 to 32767 and is passed as an integer argument to the routine.**





These functions (USR0 through USR9) transfer control to machine-language routines previously defined with DEFUSR*n* statements.

When a URS call is encountered in a statement, control goes to the address specified in the DEFUSR*n* statement. This address specifies the entry point to your machine-language routine.

**Note:** If you call a USR*n* routine before defining the routine entry point with DEFUSR*n*, an ILLEGAL FUNCTION CALL error will occur.

You can pass one argument and retrieve one output value directly via the USR argument; or you can pass and retrieve arguments indirectly via POKE and PEEK statements.

## Example

```
10 DEFUSR1=&H7D00
20 REM...MORE PROGRAM LINES HERE
100 A=USR1(X)
```

The effect of this sequence is to:

1. Define USR as a routine with an entry point at hex 7D00 (line 10).
2. Transfer control to the routine; the value X can be passed to the routine if the routine makes the CALL described below (line 100).
3. When the routine returns to BASIC, the variable A may contain the value passed back from the routine (if your routine makes the JUMP described below); otherwise A will be assigned the value of X (line 100).

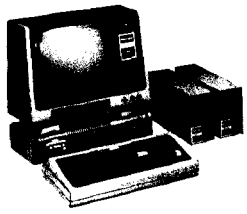
## Passing arguments to and from USR routines

There are several ways to pass arguments back and forth between your BASIC main program and your USR routines: the two major ways are listed below.

1. POKE the argument(s) into fixed RAM locations. The machine-language routine can then access these values and place results in other RAM locations. When the routine returns control to BASIC, your program can PEEK into these addresses to pick up the "output" values. **This is the only way to pass two or more arguments back and forth.**
2. Pass one argument to the routine as the argument in the USRn call, then use special ROM calls to access this argument and return a value to BASIC. **This method is limited to sending one argument and returning one value** (both are integers).

### ROM Calls

CALL 0A7FH Puts the USR argument into the HL register pair; H contains MSB, L contains LSB. This CALL should be the first instruction in your USR routine.



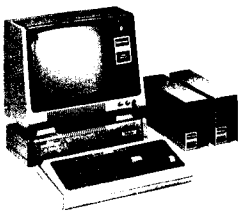
JP 0A9AH — Use this JUMP to return to BASIC; the integer in HL becomes the output of the USR call. If you don't care about returning HL, then execute a simple RETurn instruction instead of this JUMP.

Listed below is an assembled program to white out the display (an "inverse" (CLEAR) key!). Don't type it in. Type in the BASIC program that follows it.

```
                00100 ;
                00110 ; ZAP OUT SCREEN USR FUNCTION
                00120 ;
7D00            00130           ORG     7D00H
                00140 ;
                00150 ; EQUATES
                00160 ;
3C00            00170 VIDEO     EQU     3C00H           ;START OF VIDEO RAM
00BF            00180 WHITE     EQU     0BFH            ;ALL WHITE GRAPHICS BYTE
03FF            00190 COUNT     EQU     3FFH            ;NUMBER OF BYTES TO MOVE
                00200 ;
                00210 ; PROGRAM CHAIN MOVES X'BF' INTO ALL OF VIDEO RAM
                00220 ;
7D00 21003C     00230 ZAP       LD      HL,VIDEO        ;SOURCE ADDRESS
7D03 36BF       00240           LD      (HL),WHITE      ;PUT OUT 1ST BYTE
7D05 11013C     00250           LD      DE,VIDEO+1      ;DESTINATION ADDRESS
7D08 01FF03     00260           LD      BC,COUNT        ;NUMBER OF ITERATIONS
7D0B EDB0       00270           LDIR                    ;DO IT TO IT!!!
                00280 ;
7D0D C9         00290           RET                     ;RETURN TO BASIC
7D00            00300           END     ZAP
```

This routine can be POKEd into RAM and accessed as a USR routine. First start BASIC and answer the MEMORY SIZE question with 31999. Then run the program.

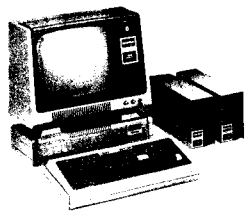
```
100 '          PROGRAM: USR1
110 ' EXAMPLE OF A USER MACHINE LANGUAGE FUNCTION
115 ' DEPRESS THE '@' KEY WHILE NUMBERS ARE PRINTING TO STOP
120 '
130 ' ******* POKE MACHINE PROGRAM INTO MEMORY *******
140 '
150 DEFUSR1 = &H7D00
160 FOR X = 32000 TO 32013   '7D00 HEX EQUAL 32000 DECIMAL
170     READ A
180     POKE X, A
190 NEXT X
192 '
194 ' ******* CLEAR SCREEN & PRINT NUMBERS 1 THRU 100 *******
196 '
200 CLS
205 PRINT TAB(15); "WHITE-OUT USER ROUTINE": PRINT
```





```
210 FOR X = 1 TO 100
220     PRINT X;
225     A$ = INKEY$: IF A$ = "@" THEN END
230 NEXT X
240 '
250 ' ******* JUMP TO WHITE-OUT SUBROUTINE *******
260 '
270 X = USR1 (0)
280 FOR X = 1 TO 1000: NEXT X   'DELAY LOOP
290 GOTO 200
300 '
310 ' ******* DATA IS DECIMAL CODE FOR HEX PROGRAM *******
320 '
330 DATA 33,0,60,54,191,17,1,60,1,255,3,237,176,201
```

Run the program. An equivalent BASIC white out routine takes a long time by comparison!



# 6/Disk-Related Features

Disk BASIC provides a powerful set of commands, statements and functions relating to disk I/O under TRSDOS. These fall into two categories:

1. File manipulation: dealing with files as units, rather than with the distinct records the files contain.
2. File access: preparing data files for I/O; reading and writing to the files.

Under the heading, **File Manipulation**, we will discuss the following commands.

| | |
|---|---|
| KILL | Delete a program or data file from the disk |
| LOAD | Load a BASIC program from disk |
| MERGE | Merge an ASCII-format BASIC program on disk with one currently in RAM |
| RUN"*program*" | Load and execute a BASIC program stored on disk |
| SAVE | Save the resident BASIC program on disk |

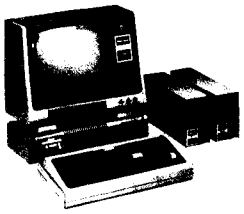
Under the heading, **File Access**, we will discuss the following statements and functions.

**Statements**

| | |
|---|---|
| OPEN | Open a file for access (create the file if necessary) |
| CLOSE | Close access to the file |
| INPUT # | Read from disk, sequential mode |
| LINE INPUT# | Read a line of data, sequential mode |
| PRINT# | Write to disk, sequential mode |
| FIELD | Assign field sizes and names to random-access file buffer |
| GET | Read from disk, random access mode |
| PUT | Write to disk, random access mode |
| LSET | Place value in specified buffer field, add blanks on the right to fill field |
| RSET | Place value in specified buffer field, add blanks on the left to fill field |

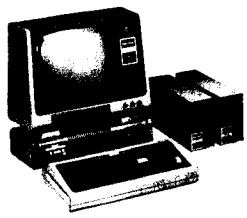
**Functions**

| | |
|---|---|
| CVD | Restore double-precision number to numeric form after GETting from disk |
| CVI | Restore integer to numeric form after GETting from disk |
| CVS | Restore single-precision number to numeric form after GETting from disk |
| EOF | Check to see if end of file encountered during read |



# TRS-80 MODEL I DISK SYSTEM

| | |
|---|---|
| LOC | GET current record number. |
| LOF | Return number of last record in file |
| MKD$ | Convert double-precision number to string so it can be PUT on disk |
| MKI$ | Convert integer to string so it can be PUT on disk |
| MKS$ | Convert single-precision number to string so it can be PUT on disk |



# File Manipulation

## KILL
## Delete a File from the Disk

**KILL *exp$ :d***

***exp$* defines a file specification for an existing file.**

***:d* is a drive specification.**

This command deletes a specified file. *:d* is optional, if none is given the file is deleted from the first diskette that contains the file.

### Example

```
KILL"OLDFILE/BAS.PSW1"
```

deletes the file specified from the first drive which contains it.

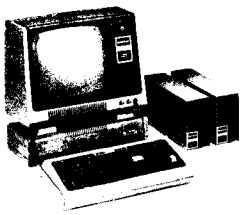
Do not KILL an open file, or you may destroy the contents of the diskette. (First, CLOSE the open file.)

## LOAD
## Load BASIC Program File from Disk

**LOAD *exp$* [,R]**

**where *exp$* defines a filespec for a BASIC program file stored on disk.**

**R tells BASIC to RUN the program after it is loaded.**





This command loads a BASIC program file into RAM; if the R option is used, BASIC will proceed to RUN the program automatically; otherwise, BASIC will return to the command mode.

LOAD without the R option clears all variables and closes all open files. LOAD with the R option clears all variables but does not close the open files.

LOAD with the R option is equivalent to the command RUN *exp$*,R. Either of these commands can be used inside programs to allow program chaining — one program calling another, etc.

## Example

```
LOAD"PROG1/BAS:2"
```

Clears resident BASIC program and loads PROG1/BAS from Drive 2; returns to BASIC command mode.

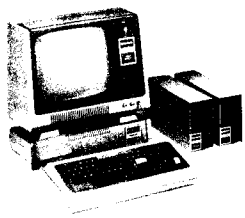
# MERGE
# Merge Disk Program with Resident Program

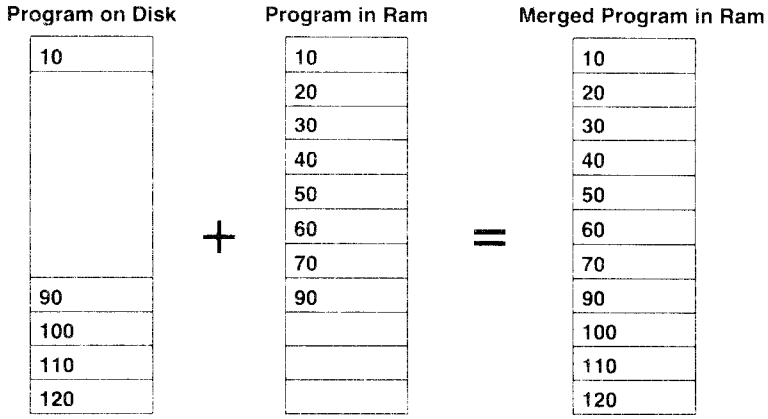
**MERGE *exp$***

***exp$* is file specification for an ASCII-format BASIC disk file, e.g., a program saved with the A-option.**

MERGE is similar to LOAD — except that the resident program is not erased before the new program *exp$* is loaded. Instead, the new program is merged into the resident program.

That is, program lines in *exp$* will simply be inserted into the resident program in sequential order. If line numbers in *exp$* coincide with line numbers in the resident program, the resident lines will be replaced by those from *exp$*.






## Sample Use

Save this program in ASCII format.

```
1000 REM . . . SUBROUTINE TO SAY HELLO
1010 PRINT "HELLO!"
1020 RETURN
```

Type NEW (ENTER), then type in this program.

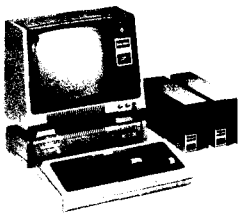
```
100 CLS
110 PRINT "LET'S CALL THE SUBROUTINE . . ."
120 PRINT "DIALING NOW . . ."
130 FOR I=1 TO 1000 : NEXT
140 GOSUB 1000
150 PRINT "BACK FROM SUBROUTINE."
160 END
```

Now type MERGE *"file"* using the file name given to the first file. List the program. Then run it.

# RUN"program"
# Load and Execute a Program from Disk

**RUN *exp$*[,R]**

***exp$* is the name of a BASIC program file. It is a string expression. (If a string constant is used, it must be enclosed in quotes.) The ,R option causes BASIC to leave open files open. If omitted, open files are closed before the program is run.**





This command loads and executes a BASIC program stored on disk. It may be used inside a program to allow chaining (one program calling another).

## Examples

```
RUN "PROG"
```

Loads and executes PROG (all open files are closed first).

```
A$="NEWPROG"
RUN A$, R
```

Loads and executes NEWPROG (all open files remain open).

# SAVE
# Save Program onto Disk

**SAVE *exp$* [,A]**

***exp$* is the name of a BASIC program file. It is a string expression. (If a string constant is used, it must be enclosed in quotes.)**

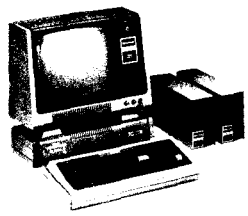
**A causes the file to be stored in ASCII rather than compressed format.**

This command lets you save your BASIC programs on disk. You can save the program in compressed or ASCII format.

Using compressed format takes up less disk space and is faster during both SAVES and LOADS. Using the ASCII option makes it possible to do certain things that cannot be done with compressed format BASIC files.

For example:

- The MERGE command requires that the disk file be in ASCII form.
- Programs which read in other programs as data will typically require that the data programs be stored in ASCII.
- The TRSDOS command APPEND also requires that disk files be in ASCII form.



### Examples

```
SAVE"FILE1/BAS,JOHNQDOE:3"
```

saves the resident BASIC program in compressed format with the file name FILE1, extension /BAS, password JOHNQDOE; the file is placed on Drive :3.

```
SAVE"MATHPAK/TXT",A
```

saves the resident program in ASCII form, using the name MATHPAK/TXT, on the first nonwrite-protected diskette.

Upon completion of a SAVE, BASIC returns in the command mode.

# File Access

This section is divided into four parts:

1. Creating files and assigning buffers — OPEN and CLOSE
2. Statements and functions
3. Sequential I/O techniques
4. Random I/O techniques

If this is your first experience with disk file access, you should concentrate on parts 1, 3 and 4, perhaps just skimming through part 2 to get a general idea of how the functions and statements work. Later you can go back to part 2 and learn the details of statement and function syntax.

## Creating Files and Assigning Buffers

During the initialization dialog, you type in a number in response to HOW MANY FILES? The number you type in tells BASIC how many buffers to create to handle your disk accesses (reads and writes).

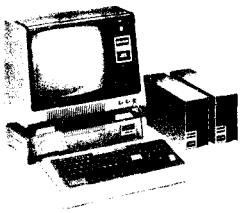
Each buffer is given a number from 1 to 15. If you type:

```
HOW MANY FILES? 3 (ENTER)
```

BASIC sets aside 3 buffers, numbered 1,2,3.

You can think of a buffer as a waiting area that data must pass through on the way to and from the disk file. When you want to access a particular file, you must tell BASIC which buffer to use in accessing that file. You must also tell BASIC what kind of access you want — sequential output, sequential input, or random input/output.

All this is done with the OPEN statement, and "undone" with the CLOSE statement.





# OPEN
# Open a File

OPEN ***mode, buffer, exp$, record-length***

***mode*** **is a string expression containing one of the following:**

- I **Sequential input starting at the first record. If the file is not found, an error will occur.**
- O **Sequential output starting at the first record. If the file is not found, it will be created.**
- E **(Extend) Sequential output starting at end of file. If the file is not found, an error will occur.**
- R **Random input/output. If the file is not found, it will be created.**

**If *mode* is a constant, it must be enclosed in quotes.**

***buffer*** **is a numeric expression specifying which buffer is to be used.**

***exp$*** **is a string expression containing the file specification. If a constant is used, it must be enclosed in quotes.**

***record-length*** **is a numeric expression from 0 to 256 specifying the logical record length. 0 is the same as 256. This option may only be used if variable-length records were requested during initialization (How Many Files?). If record-length is omitted, 256 is used. *record-length* is used with Random access *only.***

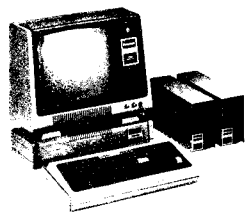
This statement lets you create a file, write data into it, update it, and read it. For details on file access, see **Methods of Access** later in this section.

If *file* includes a drive specification, BASIC will use only the specified drive. If no drive is specified, BASIC will search for a matching file, starting with the master drive (usually Drive 0).

## Examples

```
OPEN "O", 1, "DATAFILE"
```

Opens DATAFILE (creates it if it doesn't already exist) for sequential output. Output will be done through buffer #1. Records will be 256 bytes long. Since the "O" mode is specified, output will start at the first record in the file. If "E" is used instead of "O", output will start at the end of the file.



```
OPEN "R", 2, "PAYROLL/A:1", 64
```

Opens/creates PAYROLL/A for random input/output. Access will be through buffer #2. Records will be 64 bytes long (if BASIC was initialized for variable-length records).

```
BUFFER = 3: FILE$ = "DATA": RECLN = 128
OPEN "R", BUFFER, FILE$, RECLN
```

Opens/creates DATA for random input/output. Access will be through buffer #3. Records will be 128 bytes long (if BASIC was initialized for variable-length records).

# CLOSE
# Close Access to the File

**CLOSE [*nmexp* [,*nmexp*. . .] ]**

***nmexp* has a value from 1 to 15, and refers to the file's buffer number (assigned when the file was opened). If *nmexp* is omitted, all open files will be closed.**

This command terminates access to a file through the specified buffer(s). If *nmexp* has not been assigned in a previous OPEN statement, then

CLOSE *nmexp*

has no effect.

## Examples

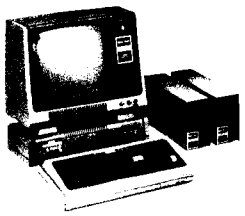
```
CLOSE 1,2,8
```

Terminates the file assignments to buffers 1, 2 and 8. These buffers can now be assigned to other files with OPEN statements.

```
CLOSE FIRST%+COUNT%
```

Terminates the file assignment to the buffer specified by the sum (FIRST% + COUNT%).

*Do not remove a diskette which contains a file opened for writing (mode = O, E, or R). First close the file.* This is because the last 256 bytes of data may not have been written to disk yet. Closing the file will write the data, if it hasn't already been written.





Any modification to the resident program (NEW, editing, LOAD, MERGE, etc.) will cause open files to be closed.

# INPUT#
# Sequential Read from Disk

**INPUT# *nmexp, var[,var . . .]***

**where *nmexp* specifies a sequential input file buffer, *nmexp*=1,2,...,15.**

***var* is the variable name to contain the data from the file.**

This statement inputs data from a disk file. The data is input sequentially. That is, when the file is first opened, a pointer is set to the beginning of the file. Each time data is input, the pointer advances. To start over reading from the beginning of the file, you must close the file and re-open it.

INPUT# doesn't care how the data was placed on the disk — whether a single PRINT# statement put it there, or whether it required 10 different PRINT# statements. What matters to INPUT# are the positions of the terminating characters and the EOF marker.

To INPUT# data successfully from disk, you need to know ahead of time what the format of the data is. Here is a description of how INPUT# interprets the various characters it encounters when reading data.

When inputting data into a variable, BASIC ignores leading blanks; when the first non-blank character is encountered, BASIC assumes it has encountered the beginning of the data item.

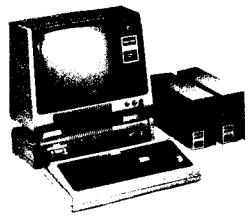
The data item ends when a terminating character is encountered or when a terminating condition occurs. The particular terminating characters vary, depending on whether BASIC is inputting to a numeric or string variable.

**Special Note**

Here's an important exception to keep in mind in reading the following material.

When ENTER (a carriage return) is preceded by ↓ (a line feed), the ENTER is not taken as a terminator. Instead, it becomes a part of the data item (string variable) or is simply ignored (numeric variable).

(To enter the ↓ character from the keyboard, press the down-arrow character. To enter the ENTER character, press ENTER.)



This exception applies to all cases noted below where (ENTER) is said to be a terminator.

## Numeric Input

Suppose the data image on disk is

```
1.234 -33 27 (ENTER)
```

(ENTER) denotes a carriage-return character (ASCII code decimal 13).

Then the statement

```
INPUT#1, A,B,C
```

or the sequence of statements

```
INPUT#1,A: INPUT#1,B: INPUT#1,C
```

will assign the values as follows:

A = 1.234

B = −33

C = 27

This works because blanks and (ENTER) serve as terminators for input to numeric variables. The blank before 1.234 is a "leading blank," therefore it is ignored. The blank after 1.234 is a terminator; therefore BASIC starts inputting the second variable at the − character, inputs the number −33, and takes the next two blanks as terminators. The third input begins at the 2 and ends with the 7.

## String Input

When reading data into a string variable, INPUT ignores all leading blanks; the first non-blank character is taken as the beginning of the data item.

If this first character is a double-quote ("), then INPUT will evaluate the data as a quoted string: it will read in all subsequent characters up to the next double-quote. Commas, blanks, and (ENTER) characters will be included in the string. The quotes themselves do not become a part of the string.

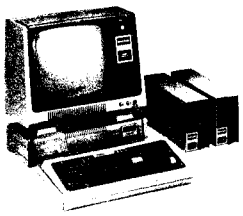
If the first character of the string item is not a double-quote, then INPUT will evaluate the data as an unquoted string: it will read in all subsequent characters up to the first comma, or (ENTER). If double-quotes are encountered, they will be included in the string.

For example, if the data on disk is:

PECOS, TEXAS"GOOD MELONS"

Then the statement

```
INPUT#1, A$,B$,C$
```





would assign values as follows:

A$ = PECOS

B$ = TEXAS "GOOD MELONS"

C$ = null string

If a comma is inserted in the data image before the first double quote, C$ will get the value, GOOD MELONS.

These are very simple examples just to give you an idea of how INPUT works. However, there are many other ways to input data—different terminators, different target variable types, etc.

Rather than taking a shotgun approach and trying to cover them all, we'll give a generalized description of how input works and what the terminating characters and conditions are, and then provide several examples.

When BASIC encounters a terminating character, it scans ahead to see how many more terminating characters it can include with the first terminator. This ensures that BASIC will begin looking for the next data item at the correct place.

The list below defines the various terminating sets INPUT# will look for. It will always try to take-in *the largest set possible*.

**Numeric-input terminator sets**

end of file encountered
255th data character encountered
,(comma)
(ENTER)
(ENTER) (↓)
[ . . . ][ (ENTER) ]
[ . . . ][ (ENTER) (↓) ]

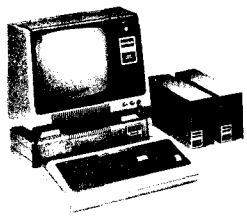
**Quoted-string terminator sets**

end of file encountered
255th data character encountered
" (double quote)
" [ . . . ][,]
" [ . . . ][ (ENTER) ]
" [ . . . ][ (ENTER) (↓) ]

**Unquoted-string terminator sets**

end of file encountered
255th data character encountered
,
(ENTER) [(↓)]

**Figure 8** describes how INPUT# assigns data to a variable.



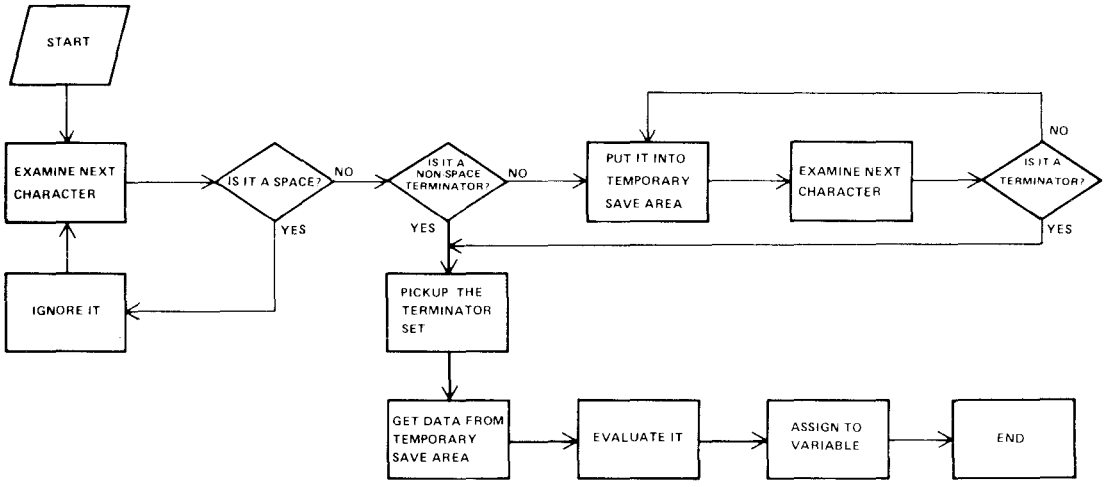


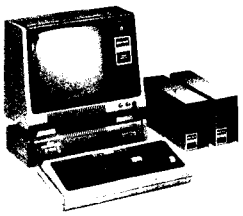

**Figure 8.** Input process.

The following table shows how various data images will be read-in by the statement:

```
INPUT#1,A,B,C
```

| Ex.# | Image on disk | Values assigned |
|---|---|---|
| 1 | 123.45 [ENTER][↓] 8.2E4 7000[ENTER] | A=123.45<br>B=82000<br>C=7000 |
| 2 | 3[↓][ENTER] 4 [ENTER]5 [ENTER] A12 eof | A=34<br>B=5<br>C=0 |
| 3 | 1,,2,3,4 [ENTER] | A=1<br>B=0<br>C=2 |
| 4 | 1,3, eof | A=1<br>B=3<br>C=0 eof error |

(eof = end of file):





In Example 2 above, why does variable C get the value 0? When the input reaches the end of file, it terminates that last data item, which then contains "A12." This is evaluated by a routine just like the BASIC VAL function — which returns a zero since the first character of "A12" is a non-numeric.

In Example 3, when INPUT# goes looking for the second data item, it immediately encounters a terminator (the comma); therefore, variable B is given the value zero.

The following table shows how various data images on disk will be read by the statement:

```
INPUT#1,A$,B$
```

| Ex.# | Image on disk | Values assigned |
|---|---|---|
| 1 | "ROBERTS,J."ROBERTS,M.N eof | A$:ROBERTS,J.<br>B$:ROBERTS,M.N. |
| 2 | ROBERTS,J.,  ROBERTS,M.N. (ENTER) | A$:ROBERTS<br>B$:J. |
| 3 | THE WORD "QUO",12345.789 (ENTER) | A$:THE WORD "QUO"<br>B$:12345.789 |
| 4 | BYTE(↓) (ENTER) UNIT OF MEMORY eof | A$:BYTE(↓)(ENTER)<br>UNIT OF MEMORY<br>B$:null (eof error) |

In Example 3, the first data item is an unquoted string, therefore, the double-quotes are not terminators, and become part of A$.

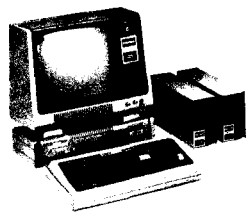
In Example 4, the (ENTER) is preceded by an (↓), therefore it does not terminate the first string; both (↓) and (ENTER) are included in A$.

# LINE INPUT#
# Read a Line of Text from Disk

**LINE INPUT#*nmexp*,*var$***

**where *nmexp* specifies a sequential output file buffer, *nmexp*=1,2,...,15.**

***var$* is the variable name to contain the string data.**



Similar to LINE INPUT from keyboard, this statement reads a "line" of string data into *var$*. This is useful when you want to read an ASCII-format BASIC program file as data, or when you want to read in data without following the usual restrictions regarding leading characters and terminators.

LINE INPUT (or LINEINPUT — the space is optional) reads everything from the first character up to:

1. an (ENTER) character which is not preceded by (↓)
2. the end of file
3. the 255th data character (this 255 character is included in the string)

Other characters encountered — quotes, commas, leading blanks, (↓) (ENTER) pairs — are included in the string.

For example, if the data looks like:

```
10 CLEAR 500 (ENTER)
20 OPEN"I",1,"PROG" (ENTER)
.
.
.
```

then the statement

```
LINEINPUT#1,A$
```

could be used repetitively to read each program line, one line at a time.

# PRINT#
# Sequential Write to Disk File

**PRINT#*nmexp*,[USING *format$;*] *exp*[*p exp* . . .]**

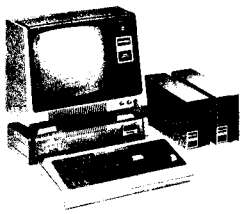
**where *nmexp* specifies a sequential output file buffer, *nmexp* = 1,2,...,15.**

***format$* is a sequence of field specifiers used with the USING option.**

***p* is a delimiter placed between every two expressions to be PRINTed to disk; either a semi-colon or comma can be used (semi-colon is preferable).**

***exp* is the expression to be evaluated and written to disk.**

This statement writes data sequentially to the specified file. When you first open a file for sequential output, a pointer is set to the beginning of the file, therefore





your first PRINT# places data at the beginning of the file. At the end of each PRINT# operation, the pointer advances, so the values are written in sequence.

A PRINT# statement creates a disk image similar to what a PRINT to display creates on the screen. Remember this, and you'll be able to set up your PRINT# list correctly for access by one or more INPUT statements.

PRINT# does not compress the data before writing it to disk; it writes an ASCII-coded image of the data.

For example, if A = 123.45

```
PRINT#1,A
```

will write a nine-byte character sequence onto disk:

```
 123.45  (ENTER)
```

The punctuation in the PRINT list is very important. Unquoted commas and semi-colons have the same effect as they do in regular PRINT to display statements.

For example, if A = 2300 and B = 1.303, then

```
PRINT#1,A,B
```

places the data on disk as

```
 2300            1.303  (ENTER)
```

The comma between A and B in the PRINT# list causes 10 extra spaces in the disk file. Generally you wouldn't want to use up disk space this way, so you should use semi-colons instead of commas.

```
PRINT#1,A;B
```

writes the data as:

```
 2300   1.303 (ENTER)
```

PRINT# with numeric data is quite straightforward—just remember to separate the items with semi-colons.

PRINT# with string data requires more care, primarily because you have to insert delimiters so the data can be read back correctly. In particular, you must separate string items with explicit delimiters if you want to INPUT# them as distinct strings.

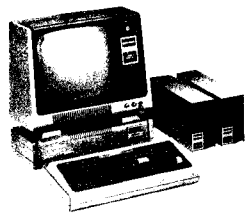
For example, suppose:

```
A$="JOHN Q. DOE" and B$="100-01-001"
```

Then:

```
PRINT#1, A$;B$
```

would produce this image on disk:



```
JOHN Q. DOE100-01-001 (ENTER)
```

which could not be INPUT back into two variables.

The statement:

```
PRINT#1, A$;",";B$
```

would produce:

```
JOHN Q. DOE, 100-01-001
```

which could be INPUT# back into two variables.

This method is adequate if the string data contains no delimiters — commas or (ENTER) — characters. But if the data does contain delimiters or leading blanks that you don't want to ignore, then you must supply explicit quotes to be written along with the data. For example, suppose `A$="DOE, JOHN Q."` and `B$="100-01-001"`

If you use

```
PRINT#1,A$;",";B$
```

the disk image will be:

```
DOE, JOHN Q.,100-01-001 (ENTER)
```

When you try to input this with a statement like

```
INPUT#2,A$,B$
```

A$ will get the value DOE, and B$ will get JOHN Q. — because of the comma after DOE in the disk image.

To write this data so that it can be input correctly, you must use the CHR$ function to insert explicit double quotes into the disk image. Since 34 is the decimal ASCII code for double quotes, use CHR$(34) as follows:

```
PRINT#1,CHR$(34);A$;CHR$(34);B$
```

this produces the disk image

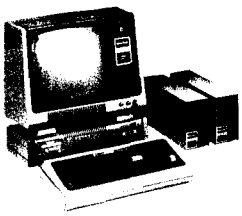
```
"DOE, JOHN Q."100-01-001 (ENTER)
```

which can be read with a simple

```
INPUT#2,A$B$
```

**Note:** You can also use the CHR$ function to insert other delimiters and control codes into the file, for example:

| | |
|---|---|
| CHR$(10) | (↓) Line Feed |
| CHR$(13) | carriage return ((ENTER)character) |
| CHR$(11) or CHR$(12) | line-printer top-of-form |





# USING Option

This option makes it easy to write files in a carefully controlled format.

For example, suppose:

```
A$="LUDWIG"
B$="VAN"
C$="BEETHOVEN"
```

Then the statement

```
PRINT#1,USING"!, !, % %";A$;B$;C$
```

would write the data in nickname form:

```
L,V,BEET <ENTER>
```

(In this case, we didn't want to add any explicit delimiters.) See the PRINT USING description in the *LEVEL II BASIC Reference Manual* for a complete explanation of the field-specifiers.

# Random Access Statements

# FIELD
# Organize a Random File-Buffer into Fields

**FIELD *nmexp,nmexp1* AS*var1$* [,*nmexp2* AS *var2$*. . .]**

***nmexp* specifies a random access file buffer, *nmexp*=1,2,...,15.**

***nmexp1* specifies the length of the first field.**

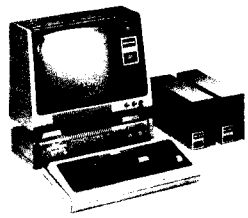
***var1$* defines a variable name for the first field.**

***nmexp2* specifies the length of the second field.**

***var2$* defines a variable name for the second field.**

**. . . Subsequent *nmexp* AS *var$* pairs define other fields in the buffer.**

**Note: The sum of all the field-lengths must not exceed the record length, and should equal the record length.**



Before FIELDing a buffer, you must use an OPEN statement to assign that buffer to a particular disk file (you must use random access mode). Then use the FIELD statement to organize a random file buffer so that you can pass data from BASIC to disk storage and vice-versa.

Each random file buffer has up to 256 bytes which can store data for transfer from disk storage to BASIC or from BASIC to disk. (When variable-length files are used, maximum may be from 1 to 256.) However, you need a way to access this buffer from BASIC so that you can either read the data it contains or place new data in it. The FIELD statement provides the means of access.

You may use the FIELD statement any number of times to "re-organize" a file buffer. FIELDing a buffer does not clear the contents of the buffer; only the means of accessing the buffer (the field names) are changed. Furthermore, two or more field names can reference the same area of the buffer.

## Examples

```
FIELD 1, 128 AS A$, 128 AS B$
```

This statement tells BASIC to assign the first 128 bytes of the buffer to the string variable A$ and the remaining 128 bytes to B$. If you now print A$ and B$, you will see the contents of the buffer. Of course, this value would be meaningless unless you have used GET to read a 256-byte record from disk.

**Note:** All data — both strings and numbers — must be placed into the buffer in string form. There are three pairs of functions (MKI$/CVI,MKS$/CVS,MKD$/CVD) for converting numbers to strings and vice-versa. See "Functions" below.

```
FIELD 3, 16 AS NM$, 25 AS AD$, 10 AS CY$, 2 AS ST$,7 AS ZP$
```

The first 16 bytes of buffer 3 are assigned the buffer name NM$; the next 25, AD$; the next 10, CY$; the next 2, ST$ and the next 7, ZP$. The remaining 196 bytes of the buffer are not fielded at all.

## More on field names

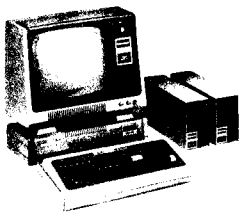
Field names, like NM$,AD$,CY$,ST$, and ZP$, are not string variables in the ordinary sense. They do not consume the string space available to BASIC.

Instead, they point to the buffer field which you assigned with the FIELD statement. That's why you can use:

```
100 FIELD 1,255 AS A$
```

without worrying about whether 255 bytes of string space are available for A$.

If you use a buffer field name on the left side of an ordinary assignment statement, that name will no longer point to the buffer field; therefore, you won't be able to access that field using the previous field name.





For example,

```
A$=B$
```

nullifies the effect of the FIELD statement above (line 100).

During random input, the GET statement places data into the 255-byte buffer, where it can be accessed using the field names assigned to that buffer. During random output, LSET and RSET place data into the buffer, so you can then PUT the buffer contents into a disk file.

Often you'll want to use a dummy variable in a FIELD statement to "pass over" a portion of the buffer and start fielding it somewhere in the middle. For example:

```
FIELD 1, 16 AS CLIENT$(1), 112 AS HIST$(1)
FIELD 1, 128 AS DUMMY$, 16 AS CLIENT$(2), 112 AS HIST$(2)
```

In the second FIELD statement, DUMMY$ serves to move the starting position of CLIENT$(2) to position 129. In this manner, two identical "subrecords" are defined on buffer number 1. We won't actually use DUMMY$ to place data into the buffer or retrieve it from the buffer.

The buffer now looks like this:

| 16 | 112 | 16 | 112 |
|---|---|---|---|
| CL$ (1) | HIST$ (1) | CL$ (2) | HIST$ (2) |

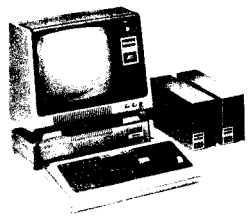
DUMMY$

# GET
# Read a Record from Disk—Random Access

**GET *nmexp1*[,*nmexp2*]**

***nmexp1* specifies a random access file buffer, *nmexp1*=1,2,...,15.**

***nmexp2* specifies which record to GET in the file; if omitted, the current record will be read.**



This statement gets a data record from a disk file and places it in the specified buffer. Before GETting data from a file, you must open the file and assign a buffer to it. That is, a statement like:

OPEN "R",*nmexp1*,*filespec*

is required *before* the statement:

GET *nmexp1*,*nmexp2*

GET tells BASIC to read record *nmexp2* from the file and place it into the *nmexp1* buffer. If you omit the record number in GET, BASIC will read the current record.

The "current record" is the record whose number is one higher than that of the last record accessed. The first time you access a file via a particular buffer, the current record is set equal to 1.

For example:

| Program statement | Effect |
|---|---|
| 1000 OPEN"R",1,"NAME/BAS" | Open NAME/BAS for random access using buffer 1 |
| 1010 FIELD 1,... | Structure buffer |
| 1020 GET 1 | GET record 1 into buffer 1 |
| 1025 REM...ACCESS BUFFER<br>1030 GET 1,30 | GET record 30 into buffer 1 |
| 1035 REM...ACCESS BUFFER<br>1040 GET 1,25 | GET record 25 into buffer 1 |
| 1046 REM...ACCESS BUFFER<br>1050 GET1 | GET record 26 into buffer 1 |

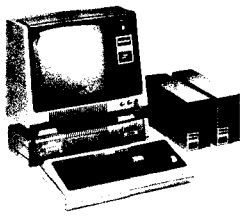
If you are using variable-length records (not fixed-length), an attempt to GET past the end of file will produce an error.

If you are using fixed-length records, the same attempt will return a null record and no error will occur. To prevent this from occurring, you can use the LOF function to determine the number of the highest numbered record.

# PUT
# Write a Record to Disk—Random Access

**PUT *nmexp1*[,*nmexp2*]**

***nmexp1* specifies a random access file buffer, *nmexp*=1,2,...,15.**





***nmexp2* specifies the record number in the file, *nmexp2* is the record you want to write. If *nmexp2* is omitted, the current record number is assumed.**

This statement moves data from a file's buffer into a specified place in the file. Before PUTting data in a file, you must:

1. OPEN the file, thereby assigning a buffer and defining the access mode (must be R);
2. FIELD the buffer, so you can
3. place data into the buffer with LSET and RSET statements.

When BASIC encounters the statement:

PUT *nmexp,nmexp2*

it does the following:

- Gets the information needed to access the disk file
- Checks the access mode for this buffer (must be R)
- Acquires more disk space for the file if necessary to accommodate the record indicated by *nmexp2*
- Copies the buffer contents into the specified record of the disk file
- Updates the current record number to equal *nmexp2 + 1*

The "current record" is the record whose number is one higher than the last record accessed. The first time you access a file via a particular buffer, the current record is set equal to 1.

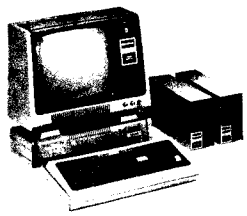
If the record number you PUT is higher than the end-of-file record number, then *nmexp2* becomes the new end-of-file record number.

# LSET and RSET
# Place Data in a Random Buffer Field

**LSET *var$* = *exp$* and RSET *var$*= *exp$***

***var$* is a field name.**

***exp$* contains the data to be placed in the buffer field named by *var$*.**



These two statements let you place character-string data into fields previously set up by a FIELD statement.

For example, suppose NM$ and AD$ have been defined as field names for a random file buffer. NM$ has a length of 18 characters, and AD$ has a length of 25 characters.

Now we want to place the following information into the buffer fields so it can be written to disk:

name: JIM CRICKET, JR.
address: 2000 EAST PECAN ST.

This is accomplished with the two statements:

```
LSET NM$="JIM CRICKET,JR. "
LSET AD$="2000 EAST PECAN ST. "
```

This puts the data in the buffer as follows:

| JIM CRICKET,JR. | 2000 EAST PECAN ST. |
|---|---|
| **NM$** | **AD$** |

Note that filler spaces were placed to the right of the data strings in both cases. If we had used RSET instead of LSET statements, the filler spaces would have been placed on the left. This is the **only** difference between LSET and RSET.

For example:

```
RSET NM$="JIM CRICKET,JR. "
RSET AD$="2000 EAST PECAN ST. "
```

places data in the fields as follows:

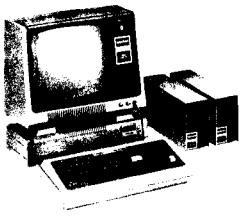
| JIM CRICKET,JR. | 2000 EAST PECAN ST. |
|---|---|
| **NM$** | **AD$** |

If a string item is too large to fit in the specified buffer field, it is always truncated on the right. That is, the extra characters on the right are ignored.

# CVD, CVI and CVS Restore String to Numeric Form

**CVD*(exp$)***

***exp$* defines an eight-character string; *exp$* is typically the name of a buffer field containing a numeric string. If LEN*(exp$)*<8, an ILLEGAL**





**FUNCTION CALL error occurs; if LEN*(exp$)*>8, only the first eight characters are used.**

**CVI*(exp$)***

***exp$* defines a two-character string; *exp$* is typically the name of a buffer field containing a numeric string. If LEN*(exp$)*<2, an ILLEGAL FUNCTION CALL error occurs; if LEN*(exp$)*>2, only the first two characters are used.**

**CVS*(exp$)***

***exp$* defines a four-character string; *exp$* is typically the name of a buffer field containing a numeric string. If LEN*(exp$)*<4, an ILLEGAL FUNCTION CALL error occurs; if LEN*(exp$)*>4, only the first four characters are used.**

These functions let you restore data to numeric form after it is read from disk. Typically the data has been read by a GET statement, and is stored in a random-access file buffer.

The functions CVD, CVI, and CVS are inverses of MKD$, MKI$, and MKS$, respectively.

For example, suppose the name GROSSPAY$ references an eight-byte field in a random-access file buffer, and after GETting a record, GROSSPAY$ contains a MKD$ representation of the number 13123.38.

Then the statement:

```
PRINT CVD(GROSSPAY$)-TAXES
```

prints the result of the difference, 13123.38 − TAXES. Whereas the statement:

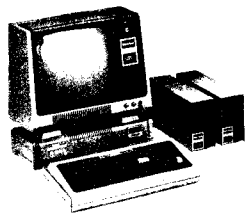
```
PRINT GROSSPAY$-TAXES
```

will produce a TYPE MISMATCH error, since string values cannot be used in arithmetic expressions.

Using the same example, the statement

```
A#=CVD(GROSSPAY$)
```

assigns the numeric value 13123.38 to the double-precision variable A#.



# EOF
# End-Of-File Detector

**EOF*(nmexp)***

***nmexp* specifies a file buffer, *nmexp*=1,2,...,15.**

This function checks to see whether all characters up to the end-of-file marker have been accessed, so you can avoid INPUT PAST END errors during sequential input.

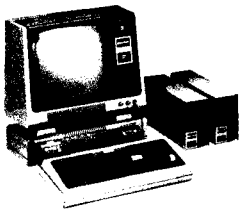
Assuming *nmexp* specifies an open file, then EOF*(nmexp)* returns 0 (false) when the EOF record has not yet been read, and −1 (true) when it has been read.

## Examples

```
IF EOF(5) THEN PRINT"END OF FILE"FILENM$
IF EOF(NM%) THEN CLOSE NM%
```

The following sequence of lines reads numeric data from DATA/TXT into the array A( ). When the last data character in the file is read, the EOF test in line 30 "passes," so the program branches out of the disk access loop, preventing an INPUT PAST END error from occurring. Also note that the variable I contains the number of elements input into array A( ).

```
5 DIM A(100) 'ASSUMING THIS IS A SAFE VALUE
10 OPEN "I",1, "DATA/TXT"
20 I%=0
30 IF EOF(1) THEN 70
40 INPUT#1,A(I%)
50 I%=I%+1
60 GOTO 30
70 REM PROGRAM CONTINUES HERE AFTER DISK INPUT
```





# LOC
# Get Current Record Number

**LOC*(file number)***

***file number* is a numeric expression specifying the buffer for a currently-open file.**

LOC is used to determine the current record number, i.e., the number of the last record read since the file was opened. LOC is only valid after a GET.

## Example

```
PRINT LOC(1)
```

## Sample Program

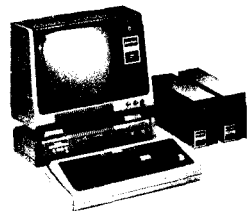
```
1310 A$ = "WILLIAM WILSON"
1320 GET 1, X: X=X+1
1330 IF N$ = A$ THEN PRINT "FOUND IN RECORD" LOC(1): CLOSE:
     END
1340 GOTO 1320
```

This is a portion of a program. Elsewhere the file has been opened and fielded. N$ is a field variable. If N$ matches A$ the record number in which it was found is printed.

# LOF
# Get End-Of-File Record Number

**LOF*(nmexp)***

***nmexp* specifies a random access buffer *nmexp*=1,2,...,15.**



This function tells you the number of the last, i.e., highest numbered, record in a file. It is useful for both sequential and random access.

For example, during random access to a pre-existing file, you often need a way to know when you've read the last valid record. LOF provides a way.

LOF is valid as soon as a previously created file is opened. If a file is extended, LOF is not valid until a GET is executed.

## Examples:

```
10 OPEN "R",1,"UNKNOWN/TXT"
20 FIELD 1,255 AS A$
30 FORI%=1 TO LOF(1)
40 GET 1,I%
50 PRINT A$
60 NEXT
```

In line 30, LOF(1) specifies the highest record number to be accessed.

**Note:** If you attempt to GET record numbers beyond the end-of-file record, BASIC simply fills the buffer with hexadecimal zeros, and no error is generated.

When you want to add to the end of a file, LOF tells you where to start adding:

```
100 I%=LOF(1)+1 'HIGHEST EXISTING RECORD
110 PUT 1,I%    'ADD NEXT RECORD
```

# MKD$, MKI$, and MKS$ Convert Data, Numeric-to-String

**MKD$(*nmexp*)**

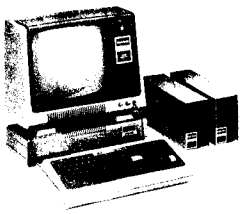
***nmexp* is evaluated as a double-precision number.**

**MKI$(*nmexp*)**

***nmexp* is evaluated as an integer, $-32768 <= nmexp <= 32768$; if *nmexp* exceeds this range, an ILLEGAL FUNCTION CALL error occurs. Any fractional component in *nmexp* is truncated.**

**MKS$(*nmexp*)**

***nmexp* is evaluated as a single-precision number.**





These functions change a number to a "string." Actually the byte values which make up the number are not changed; only one byte, the internal data-type specifier, is changed, so that numeric data can be placed in a string variable.

That is:

MKD$ returns an eight-byte string.
MKI$ returns a two-byte string.
MKS$ returns a four-byte string.

## Examples

```
LSET TALLY$=MKI$(I%)
```

Field name TALLY$ would now contain a two-byte representation of the integer I%.

```
A$=MKI$(8/I)
```

A$ becomes a two-byte representation of the integer portion of 8/I. Any fractional portion is ignored. Note that A$ in this case is a normal string variable, not a buffer-field name.

Suppose BASEBALL/BAT (a non-standard file extension) has been opened for random access using buffer 2, and the buffer has been FIELDed as follows:

| field: | NM$ | YRS$ | AVG$ | HR$ | AB$ | ERNING$ |
|---|---|---|---|---|---|---|
| length: | 16 | 2 | 4 | 2 | 4 | 4 |

NM$ is intended to hold a character string; AVG$, AB$ and ERNING$, converted single-precision values; YRS$ and HR$, converted integers.

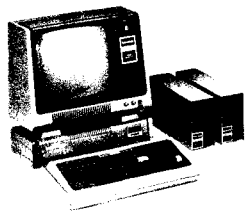
Suppose we want to write the following data record:

SLOW LEARNER played 38 years; lifetime batting average .123; career homeruns, 11; at bats, 32768;...,earnings − 13.75.

Then we'd use the make-string functions as follows:

```
1000 LSET NM$="SLOW LEARNER"
1010 LSET YRS$=MKI$(38)
1020 LSET AVG$=MKS$(.123)
1030 LSET HR$=MKI$(11)
1040 LSET AB$=MKS$(32768)
1050 LSET ERNING$=MKS$(-13.75)
```

After this sequence, you can write SLOW LEARNER's information to disk with the PUT statement. When you read it back from disk with GET, you will need to restore the numeric data from string to numeric form, using CVI and CVS functions.



# Methods of Access

Disk BASIC provides two means of file access:

- Sequential — in which you start reading or writing data at the beginning of a file; subsequent reads or writes are done at following positions in the file.
- Random — in which you start reading or writing at any record you specify. (Random access is also called direct access.)

Sequential access is stream-oriented; that is, the number of characters read or written can vary, and is usually determined by delimiters in the data. Random access is record-oriented; that is, data is always read or written in fixed-length blocks called records.

To do any input/output to a disk file, you must first open the file. When you open the file, you specify what kind of access you want:

- O for sequential output
- I for sequential input
- R for random input/output
- E (Extend) for sequential output starting at the end of file.

You also assign a file buffer for BASIC to use during file accesses. This number can be from 1 to 15, but must not exceed the number of concurrent files you requested when you started BASIC from TRSDOS. For example, if you started BASIC with 3 files, you can use buffer numbers 1, 2, and 3. Once you assign a buffer number to a file, you cannot assign that number to another file until you Close the first file.

## Examples

```
OPEN "O", 1, "TEST"
```

Creates a sequential output file named TEST on the first available drive; if TEST already exists, its previous contents are lost. Buffer 1 will be used for this file.

```
OPEN "I", 2, "TEST"
```

Opens TEST for sequential input, using buffer 2.

```
OPEN "R", 1, "TEST"
```

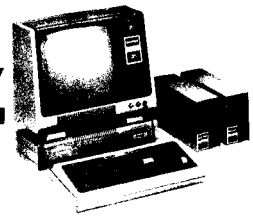
Opens TEST for direct access, using buffer 1. If TEST does not exist, it will be created on the first available drive. Since record length is not specified, 256-byte records will be used.

```
OPEN "R", 1, "TEST", 40
```

Same as preceding example, but 40-byte records will be used.

```
OPEN "E", 1, "TEST"
```

Opens TEST sequentially for write and positions to EOF.



# Sequential Access

This is the simplest way to store data in and retrieve it from a file. It is ideal for storing free-form data without wasting space between data items. You read the items back in the same order in which they were written.

There are several important points to keep in mind.

1. You must start writing at the beginning of the file. If the data you are seeking is somewhere inside, you have to read your way up to it.
2. Each time you Open a file for sequential output, the file's previous contents are lost, unless you use "E" instead of "O" for the mode.
3. To update (change) a sequential file, read in the file and write out the updated data to a *new* output file.
4. Data written sequentially usually includes delimiters (markers) to signify where each data item begins and ends. To read a file sequentially, you must know ahead of time the format of the data. For example: Does the file consist of lines of text terminated with carriage returns? Does it consist of numbers separated by blank spaces? Does it consist of alternating text and numeric information?
5. Sequential files are always written as ASCII-coded text, one byte for each character of data. For example, the number:

   ```
   1.2345
   ```

   requires 8 bytes of disk storage, including the leading and trailing blanks that are supplied. The text string:

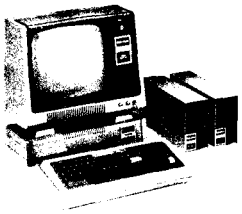
   ```
   JOHNSON, ROBERT
   ```

   requires 15 bytes of disk storage.
6. Sequential files are always written with a record length of 256.

# Sequential Output: An Example

Suppose we want to store a table of English-to-metric conversion constants:

| English unit | Metric equivalent |
|---|---|
| 1 inch | 2.54001 centimeters |
| 1 mile | 1.60935 kilometers |
| 1 acre | 4046.86 sq. meters |
| 1 cubic inch | 0.01638716 liter |
| 1 U.S. gallon | 3.785 liters |
| 1 liquid quart | 0.9463 liter |
| 1 lb (avoir) | 0.45359 kilogram |





First we decide what the data image is going to be. Let's say we want it to look like this:

*English unit* ♦ *metric unit, factor* (ENTER)

For example, the stored data would start out:

```
IN->CM, 2.54001   (ENTER)
```

The following program will create such a data file.

**Note:** X'0D' represents a carriage return.

```
10 OPEN "O",1,"METRIC/TXT"
20 FOR I%=1 TO 7
30     READ UNIT$, FACTR
40     PRINT#1, UNIT$; ",";  FACTR
50 NEXT
60 CLOSE
70 DATA IN->CM, 2.54001, MI->KM, 1.60935, ACRE->SQ.KM,
   4046.86 E-6
80 DATA CU.IN->LTR, 1.638716E-2, GAL->LTR, 3.785
90 DATA LIQ.QT->LTR, 0.9463, LB->KG, 0.45359
```

Line 10 creates a disk file named METRIC/TXT, and assigns buffer 1 for sequential output to that file. The extension /TXT is used because sequential output always stores the data as ASCII-coded text.

**Note:** If METRIC/TXT already exists, line 10 will cause all its data to be lost. Here's why: Whenever a file is opened for sequential output, the end-of-file (EOF) is set to the beginning of the file. In effect, TRSDOS "forgets" that anything has ever been written beyond this point. To avoid this, you could use E instead of O in line 10.

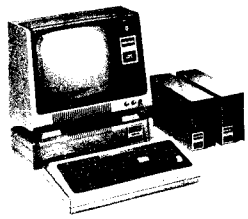
Line 40 prints the current contents of UNIT$ and FACTR to the file. Since the string items do not contain delimiters, it is not necessary to print explicit quotes around them. The explicit comma is sufficient.

Line 60 closes the file. The EOF is at the end of the last data item, i.e., 0.45359, so that later, during input, BASIC will know when it has read all the data.

# Sequential Input: An Example

The following program reads the data from METRIC/TXT into two "parallel" arrays, then asks you to enter a conversion problem.

```
5 CLEAR 500
10 DIM UNIT$(9), FACTR(9)    'allows for up to 10 data pairs
20 OPEN"I",1,"METRIC/TXT"
25 I%=0
30 IF EOF(1) THEN 70
```



```
40 INPUT#1, UNIT$(I%),FACTR(I%)
50 I%=I%+1
60 GOTO 30
70 CLOSE    '   Conversion factors have been read-in
100 CLS: PRINT TAB(5)"*** English to Metric Conversions ***"
110 FOR ITEM%=0 TO I%-1
120    PRINT TAB(9);USING"(## )   %          %   ";ITEM%,
    UNIT$(ITEM%)
130 NEXT
140 PRINT @ 704, "Which conversion (0-6)";
150 INPUT CHOICE%
160 INPUT"Enter English quantity";V
170 PRINT"The Metric equivalent is" V*FACTR(CHOICE%)
180 INPUT"Press <ENTER> to continue";X
190 PRINT @ 704, CHR$(31)    'clear to end of frame
200 GOTO 140
```

Line 20 opens the file for sequential input. Input begins at the beginning of the file.

Line 30 checks to see that the end-of-file record hasn't been reached. If it has, control branches from the disk input loop to the part of the program that uses the newly acquired data.

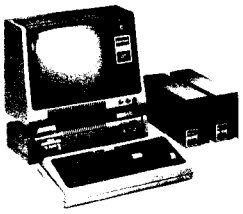
Line 40 reads a value into the string array UNIT$( ), and a number into the single-precision array FACTR( ). Note that this INPUT list parallels the PRINT# list that created the data file (see the section "Sequential Output: An Example"). This parallelism is not required, however. We could just as successfully have used:

```
40 INPUT#1, UNIT$(I%): INPUT#1,FACTR(I%)
```

# How to update a file

Suppose you want to add more entries into the English-Metric conversion file. You could simply re-Open the file with mode = E and PRINT# the extra data. Or, you might want to leave the old file intact and output a new file:

1. Open the file for sequential input (Mode = I)
2. Open another new data file for sequential output (Mode = O)
3. Input a block of data and update the data as necessary
4. Output the data to the new file
5. Repeat steps 3 and 4 until all data has been read, updated, and output to the new file; then go to step 6
6. Close both files





# Sequential Line Input: An Example

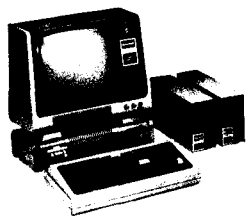
Using the line-oriented input, you can write programs that edit other BASIC program files: renumber them, change LPRINTS to PRINTS, etc. — as long as these "target" programs are stored in ASCII format.

The following program counts the number of lines in any ASCII — format BASIC disk file with the extension /TXT.

```
10 CLEAR 300
20 INPUT"WHAT IS THE NAME OF THE PROGRAM"; PROG$
30 IF INSTR(PROG$,"/TXT")=0 THEN 110 'require /TXT extension
40 OPEN"I", 1, PROG$
50 I%=0
60 IF EOF(1) THEN 90
70 I%=I%+1: LINE INPUT#1, TEMP$
80 GOTO 60
90 PRINT PROG$" IS" I% "LINES LONG."
100 CLOSE: GOTO 20
110 PRINT "FILESPEC MUST INCLUDE THE EXTENSION '/TXT'"
120 GOTO 20
```

For BASIC programs stored in ASCII, each program line ends with a carriage return character not preceded by a line feed. So the LINE INPUT in line 70 automatically reads one entire line at a time, into the variable TEMP$. Variable I% actually does the counting.

To try out the program, first save any BASIC program using the A (ASCII) option (See SAVE). Use the extension /TXT.



# Random Access Techniques

Random access offers several advantages over sequential access:

- Instead of having to start reading at the beginning of a file, you can read any record you specify.
- To update a file, you don't have to read in the entire file, update the data, and write it out again. You can rewrite or add to any record you choose, without having to go through any of the other records.
- Random access is more efficient — data takes up less space and is read and written faster.
- Opening a file for direct access allows you to write and read from the file via the same buffer.
- Random access provides many powerful statements and functions to structure your data. Once you have set up the structure, direct input/output becomes quite simple.

The last advantage listed above is also the "hard part" of direct access. It takes a little extra thought.

For the purposes of direct access, you can think of a disk file as a set of boxes — like a wall of post-office boxes. Just like the post office receptacles, the file boxes are numbered. We call these boxes "records."

You can place data in any record, or read the contents of any record, with statements like:

```
PUT 1,5  write buffer-1 contents to record 5
GET 1,5  read the contents of record 5 into buffer-1
```

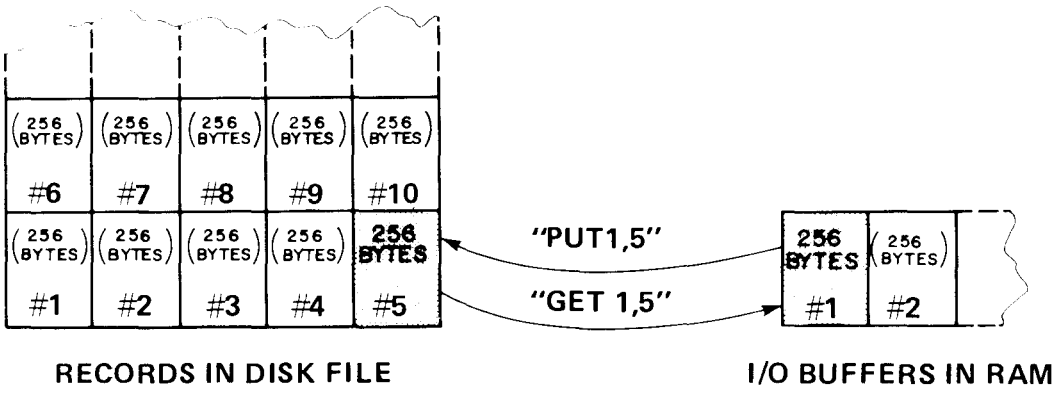
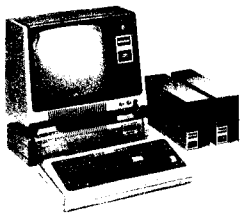



**Figure 9.** GET and PUT.





The buffer is a waiting area for the data. Before writing data to a file, you must place it in the buffer assigned to the file. After reading data from a file, you must retrieve it from the buffer.

As you can see from the sample PUT and GET statements above, data is passed to and from the disk in records. The size of each record is determined by an Open statement.

## Storing Data in a Buffer

You must place the entire record into the buffer before putting its contents into the disk file.

This is accomplished by 1) dividing the buffer up into fields and naming them, then 2) placing the string or numeric data into the fields.

For example, suppose we want to store a glossary on disk. Each record will consist of a word followed by its definition. We start with:

```
100 OPEN"R", 1, "GLOSSARY/BAS"
110 FIELD 1, 16 AS WD$, 240 AS MEANING$
```

Line 100 opens a file named GLOSSARY/BAS (creates it if it doesn't already exist); and gives buffer 1 direct access to the file.

Line 110 defines two fields onto buffer 1:

WD$ consists of the first 16 bytes of the buffer;
MEANING$ consists of the last 240 bytes.

WD$ and MEANING$ are now **field-names**

**What makes field names different?** Most string variables point to an area in memory called the string space. This is where the value of the string is stored.

Field names, on the other hand, point to the buffer area assigned in the FIELD statement. So, for example, the statement:

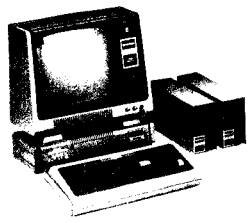
```
10 PRINT WD$; ":"; MEANING$
```

displays the contents of the two buffer fields defined above.

These values are meaningless unless we first place data in the buffer. LSET, RSET and GET can all be used to accomplish this function. We'll start with LSET and RSET, which are used in preparation for disk output.

Our first entry is the word "left-justify" followed by its definition.

```
100 OPEN"R", 1, "GLOSSARY/BAS"
110 FIELD 1, 16 AS WD$, 240 AS MEANING$
120 LSET WD$="LEFT-JUSTIFY"
130 LSET MEANING$="TO PLACE A VALUE IN A FIELD FROM LEFT TO
```



```
RIGHT; IF THE DATA DOESN'T FILL THE FIELD, BLANKS ARE
ADDED ON THE RIGHT; IF THE DATA IS TOO LONG, THE EXTRA
CHARACTERS ON THE RIGHT ARE IGNORED. LSET IS A LEFT-
JUSTIFY FUNCTION."
```

Line 120 left-justifies the value in quotes into the first field in buffer 1. Line 130 does the same thing to its quoted string.

**Note:** RSET would place filler-blanks to the *left* of the item. Truncation would still be on the right.

Now that the data is in the buffer, we can write it to disk with a simple PUT statement:

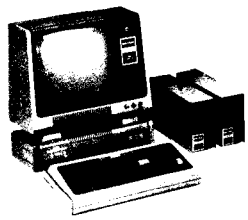
```
140 PUT 1,1
150 CLOSE
```

This writes the first record into the file GLOSSARY/BAS.

To read and print the first record in GLOSSARY/BAS, use the following sequence:

```
160 OPEN"R", 1, "GLOSSARY/BAS"
170 FIELD 1, 16 AS WD$, 240 AS MEANING$
180 GET 1,1
190 PRINT WD$: PRINT MEANING$
200 CLOSE
```

Line 160 and 170 are required only because we closed the file in line 150. If we hadn't closed it, we could go directly to line 180.



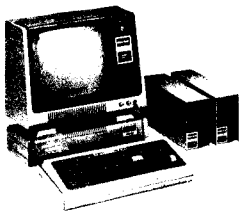
# Random Access: A General Procedure

The previous example shows the necessary sequences to read and write using random access. But it does not demonstrate the primary advantages of this form of access — in particular, it doesn't show how to update existing files by going directly to the desired record.

The program below, GLOSSACC/BAS, develops the glossary example to show some of the techniques of random access for file maintenance. But before looking at the program, study this general procedure for creating and maintaining files via random access.

| Step | See GLOSSACC/BAS, Line Number |
|---|---|
| 1. Open the file | 110 |
| 2. Field the buffer | 120 |
| 3. Get the record to be updated | 140 |
| 4. Display current contents of the record (use CVD, CVI, CVS before displaying numeric data) | 145-170 |
| 5. LSET and RSET new values into the fields (use MKD$, MKI$, MKS$ with numeric data before setting it into the buffer) | 210-230 |
| 6. PUT the updated record | 240 |
| 7. To update another record, continue at step 3. Otherwise, go to step 8. | 250-260 |
| 8. Close the file | 270 |

```
10 REM ..... GLOSSACC/BAS ...
100 CLS : CLEAR 300
110 OPEN "R", 1, "GLOSSARY/BAS"
120 FIELD 1, 16 AS WD$, 238 AS MEANING$, 2 AS NX$
130 INPUT "WHAT RECORD DO YOU WANT TO ACCESS"; R%
140 GET 1, R%
145 NX% = CVI(NX$)    'SAVE LINK TO NEXT ALPHABETICAL ENTRY
150 PRINT "WORD :     "WD$
160 PRINT "DEF'N : " : PRINT MEANING$
170 PRINT "NEXT ALPHABETICAL ENTRY: RECORD #:" NX% : PRINT
180 W$ = "" : INPUT "TYPE NEW WORD <ENTER> OR <ENTER> IF OK";
    W$
190 D$ = "" : PRINT "TYPE NEW DEF'N <ENTER> OR <ENTER> IF
    OK?" : LINE INPUT D$
200 INPUT "TYPE NEW SEQUENCE NUMBER OR <ENTER> IF OK"; NX%
210 IF W$ <> "" THEN LSET WD$ = W$
```





```
220 IF D$ <> "" THEN LSET MEANING$ = D$
230 LSET NX$ = MKI$ (NX%)
240 PUT 1, R%
245 R% = NX% 'USE NEXT ALPHA, LINK AS DEFAULT FOR NEXT RECORD
250 CLS : PRINT " TYPE <ENTER> TO READ NEXT ALPHA, ENTRY,":
    PRINT" OR RECORD # <ENTER> FOR SPECIFIC ENTRY,": INPUT "
    OR 0 <ENTER> TO QUIT"; R%
260 IF 0<R% THEN 140
270 CLOSE
280 END
```

Notice we've added a field, NX$, to the record (line 120). NX$ will contain the number of the record which comes next in alphabetical sequence. This enables us to proceed alphabetically through the glossary, provided we know which record contains the entry which should come first.

For example, suppose the glossary contains:

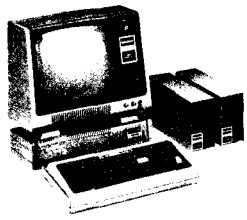
| record# | word (WD$) | defn, | pointer to next alpha. entry (NX$) |
|---|---|---|---|
| 1 | LEFT-JUSTIFY | ... | 3 |
| 2 | BYTE | ... | 4 |
| 3 | RIGHT-JUSTIFY | ... | 0 |
| 4 | HEXADECIMAL | ... | 1 |

When we read record 2 (BYTE), it tells us that record 4 (HEXADECIMAL) is next, which then tells us record 1 (LEFT-JUSTIFY) is next, etc. The last entry, record 3 (RIGHT-JUSTIFY), points us to zero, which we take to mean "The End."

Since NX$ will contain an integer, we have to first convert that number to a two-byte string representation, using MKI$ (line 230 above).

The following program displays the glossary in alphabetical sequence:

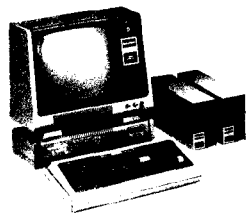
```
300 REM ... GLOSSOUT/BAS ...
310 CLS : CLEAR 300
320 OPEN "R", 1, "GLOSSARY/BAS"
330 FIELD 1, 16 AS WD$, 238 AS MEANING$, 2 AS NX$
340 INPUT "WHICH RECORD IS FIRST ALPHABETICALLY"; N%
350 GET 1, N%
360 PRINT : PRINT WD$
370 PRINT MEANING$
380 N% = CVI(NX$)
390 INPUT "PRESS <ENTER> TO CONTINUE"; X
400 IF N% <> 0 THEN 350
410 CLOSE
420 END
```



# Appendix A / Disk BASIC Error Codes/Messages

51 Field overflow
52 Internal error
53 Bad file number
54 File not found
55 Bad file mode
58 Disk I/O error
62 Disk full
63 Input past end
64 Bad record number
65 Bad file name
67 Direct statement in file
68 Too many files
69 Disk write-protect
70 File access

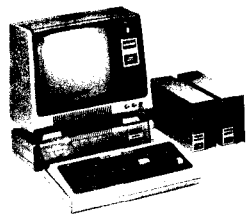
**Note:** Disk errors cannot be simulated via the ERROR statement.



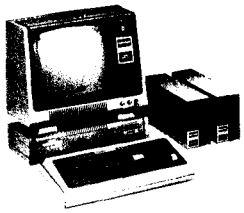
# Appendix B / Model I BASIC Reserved Words

| | | | |
|---|---|---|---|
| @ | ELSE | LLIST | RENAME |
| ABS | END | LPRINT | RESET |
| AND | EOF | LOAD | RESTORE |
| ASC | ERL | LOC | RESUME |
| ATN | ERR | LOF | RETURN |
| AUTO | ERROR | LOG | RIGHT$ |
| CDBL | EXP | MEM | RND |
| CHR$ | FIELD | MERGE | RSET |
| CINT | FIX | MID$ | RUN |
| CLEAR | FN | MKD$ | SAVE |
| CLOCK | FOR | MKI$ | SET |
| CLOSE | FORMAT | MKS$ | SGN |
| CLS | FRE | NAME | SIN |
| CMD | FREE | NEW | SQR |
| CONT | GET | NEXT | STEP |
| COS | GOSUB | NOT | STOP |
| CSNG | GOTO | ON | STRING$ |
| CVD | IF | OPEN | STR$ |
| CVI | INKEY$ | OR | SYSTEM |
| CVS | INP | OUT | TAB |
| DATA | INPUT | PEEK | TAN |
| DEFDBL | INSTR | POINT | THEN |
| DEFFN | INT | POKE | TIME$ |
| DEFINT | KILL | POS | TO |
| DEFSNG | LEFT$ | POSN | TROFF |
| DEFUSR | LET | PRINT | TRON |
| DEFSTR | LSET | PUT | USING |
| DELETE | LEN | RANDOM | USR |
| DIM | LINE | READ | VAL |
| EDIT | LIST | REM | VARPTR |
| | | | VERIFY |

None of these words can be used inside a variable name. You'll get a syntax error if you try to use these words as variables.



# Index



# IMPORTANT NOTICE

ALL RADIO SHACK COMPUTER PROGRAMS ARE LICENSED ON AN "AS IS" BASIS WITHOUT WARRANTY.

Radio Shack shall have no liability or responsibility to customer or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer equipment or programs sold by Radio Shack, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer or computer programs.

**NOTE:** Good data processing procedure dictates that the user test the program, run and test sample sets of data, and run the system in parallel with the system previously in use for a period of time adequate to insure that results of operation of the computer or program are satisfactory.

This Warranty gives the original purchaser specific legal rights, and the original purchaser may have other rights which vary from state to state.

# RADIO SHACK SOFTWARE LICENSE

A. Radio Shack grants to CUSTOMER a non-exclusive, paid up license to use on CUSTOMER'S computer the Radio Shack computer software received. Title to the media on which the software is recorded (cassette and/or disk) or stored (ROM) is transferred to the CUSTOMER, but not title to the software.

B. In consideration for this license, CUSTOMER, shall not reproduce copies of Radio Shack software except to reproduce the number of copies required for use on CUSTOMER'S computer (if the software allows a backup copy to be made,) and shall include Radio Shack's copyright notice on all copies of software reproduced in whole or in part.

C. CUSTOMER may resell Radio Shack's system and applications software (modified or not, in whole or in part), provided CUSTOMER has purchased one copy of the software for each one resold. The provisions of this software License (paragraphs A, B, and C) shall also be applicable to third parties purchasing such software from CUSTOMER.

# LIMITED WARRANTY

For a period of 90 days from the date of delivery. Radio Shack warrants to the original purchaser that the computer hardware unit shall be free from manufacturing defects. This warranty is only applicable to the original purchaser who purchased the unit from Radio Shack company-owned retail outlets or duly authorized Radio Shack franchisees and dealers. This warranty is voided if the unit is sold or transferred by purchaser to a third party. This warranty shall be void if this unit's case or cabinet is opened, if the unit has been subjected to improper or abnormal use, or if the unit is altered or modified. If a defect occurs during the warranty period, the unit must be returned to a Radio Shack store, franchisee, or dealer for repair, along with the sales ticket or lease agreement. Purchaser's sole and exclusive remedy in the event of defect is limited to the correction of the defect by adjustment, repair, replacement, or complete refund at Radio Shack's election and sole expense. Radio Shack shall have no obligation to replace or repair expendable items.

Any statements made by Radio Shack and its employees, including but not limited to, statements regarding capacity, suitability for use, or performance of the unit shall *not* be deemed a warranty or representation by Radio Shack for any purpose, nor give rise to any liability or obligation of Radio Shack.

EXCEPT AS SPECIFICALLY PROVIDED IN THIS WARRANTY OR IN THE RADIO SHACK COMPUTER SALES AGREEMENT, THERE ARE NO OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS OR BENEFITS, INDIRECT, SPECIAL, CONSEQUENTIAL OR OTHER SIMILAR DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR OTHERWISE.


**RADIO SHACK, A DIVISION OF TANDY CORPORATION**

**U.S.A.: FORT WORTH, TEXAS 76102**
**CANADA: BARRIE, ONTARIO L4M 4W5**

**TANDY CORPORATION**

| AUSTRALIA | BELGIUM | U. K. |
|---|---|---|
| 280-316 VICTORIA ROAD<br>RYDALMERE, N.S.W. 2116 | PARC INDUSTRIEL DE NANINNE<br>5140 NANINNE | BILSTON ROAD WEDNESBURY<br>WEST MIDLANDS WS10 7JN |

8749342

**PRINTED IN U.S.A.**

# Important Information for Cassette Users

**Note:** Model III BASIC on the TRS-80 Model III is essentially the same as Level II BASIC on the TRS-80 Model I. All of the following references to Level II BASIC also refer to Model III BASIC. The only difference is that a higher baud rate for saving onto tape can be set if you have a Model III with Model III BASIC (high = 1500 and low = 500). Both low and high baud rate use the same volume setting on the Model III.

## Using Your Cassette Deck

Many factors affect the performance of a cassette system. The most significant one is volume. Too low a volume may cause some of the information to be missed. Too high a volume may cause distortion and result in the transfer of background noise as valid information.

Four different cassette models have been supplied with the TRS-80 system—the CTR-40, CTR-41, CTR-80, and CTR-80A. Each model has its own loading characteristics. The table below gives the suggested volume ranges for each of the CTR models.

Notice that the volume ranges for Level I and Level II are different. This is because the Level II data transfer rate is faster (500 baud vs. 250 baud). Also, notice that for the TRS-80 Model I, pre-recorded Radio Shack programs need a slightly higher volume setting than that required by your own CSAVEd tapes. This is because the pre-recorded tapes are produced with high-speed audio equipment at a slightly lower volume level than the CSAVE process provides. The Model III records at a lower volume than the pre-recorded tapes are recorded at, so the volume setting for user-generated tapes is higher than for pre-programmed tapes. You will need to take this into account when CLOADing Level II programs into a Model III.

| Recorder Model | User-Generated Tapes | | Pre-Recorded Radio Shack Tapes | |
|---|---|---|---|---|
| | LEVEL I | LEVEL II | LEVEL I | LEVEL II |
| CTR-40 | YELLOW LINE | RED LINE | YELLOW LINE | RED LINE |
| CTR-41 | 6-8 | 4-6 | 6.5-8.5 | 5-7 |
| CTR-80 & CRT-80A | 4.5-6.5 | 3-5 | 5.5-7.5 | 2.5-5 |

**Recommended Volume Settings for Radio Shack Cassette Decks When Used with the TRS-80 Model I**

| Recorder Model | User-Generated Tapes | Pre-Recorded Radio Shack Tapes |
|---|---|---|
| CTR-80, CTR-80A | 5-7 | 4-6 |

**Recommended Volume Settings for Radio Shack Cassette Decks When Used with TRS-80 Model III**

(With the CTR-40, CTR-80, and CTR-80A, turn the control to the left to increase volume. With the CTR-41, turn the control to the right.)

When information is being loaded from the cassette tape, two asterisks will appear on the screen. The one on the right will flash on or off as the program is read in. If the asterisks do not appear, or the one on the right does not flash, then the volume setting is probably too low. Increase the volume and try again. If you have a Model III this may be an indication that the tape's baud rate is different than the Computer's baud rate. (All Radio Shack Model I Level II pre-recorded cassettes are recorded at 500 baud rate, so Low baud rate must be selected when they are loaded on the Model III.) Try resetting the baud rate from high to low or vice versa (See your Operation Manual).

Use the reset button to stop the cassette and return control to you if loading problems occur.

Radio Shack programs are recorded at least twice on each tape. Following this practice when you record programs on tape will give you a back-up if one does not load properly or if it becomes damaged.

**Important Note:** The CTR-41 requires that you keep the supplied "dummy plug" in the **MIC** jack at all times. However, the other models should never be used with the "dummy plug."

## Level I

Sometimes you will get an error message during an attempted CLOAD. This means that some information was lost or garbled. Adjust the volume level slightly and try again.

## Level II (Also Model III BASIC)

In case of an error message, proceed as above. In Level II, there is also a rare case in which the program is not loaded correctly even though no error message is generated. So, after CLOADing a program, be sure to LIST it. If some data was garbled, then at some point in the listing the display will be filled with meaningless words and characters. Adjust the volume and try again.

# Hints and Tips

Computer tapes should be stored in a relatively dust-free area (a cassette case is recommended) and protected from high temperatures. Magnetic and electrical fields may alter recorded information, so avoid placing the tape near them

(i.e. household appliances, power sources such as transformers and television sets, etc.).

The cassette deck supplied with the TRS-80 is very compatible with the system and will perform its duties with great success. To keep the cassette deck in top condition and thus minimize your problems, you should periodically perform some routine maintenance on it. Dirty heads can cause as much as a 50% loss of volume. Also, heads become magnetized with use and may cause distortion. We recommend that you clean the head, capstan, and pinch roller after every four hours of operation. Heads on new recorders should always be cleaned before use.

**Note:** Cassette cleaning and demagnetizing accessories are available from your local Radio Shack store.

# IMPORTANT NOTICE

ALL RADIO SHACK COMPUTER PROGRAMS ARE LICENSED ON AN "AS IS" BASIS WITHOUT WARRANTY.

Radio Shack shall have no liability or responsibility to customer or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer equipment or programs sold by Radio Shack, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer or computer programs.

NOTE: Good data processing procedure dictates that the user test the program, run and test sample sets of data, and run the system in parallel with the system previously in use for a period of time adequate to insure that results of operation of the computer or program are satisfactory.

# RADIO SHACK SOFTWARE LICENSE

A. Radio Shack grants to CUSTOMER a non-exclusive, paid up license to use on CUSTOMER'S computer the Radio Shack computer software received. Title to the media on which the software is recorded (cassette and/or disk) or stored (ROM) is transferred to the CUSTOMER, but not title to the software.

B. In consideration for this license, CUSTOMER shall not reproduce copies of Radio Shack software except to reproduce the number of copies required for use on CUSTOMER'S computer (if the software allows a backup copy to be made), and shall include Radio Shack's copyright notice on all copies of software reproduced in whole or in part.

C. CUSTOMER may resell Radio Shack's system and applications software (modified or not, in whole or in part), provided CUSTOMER has purchased one copy of the software for each one resold. The provisions of this software License (paragraphs A, B, and C) shll also be applicable to third parties purchasing such software from CUSTOMER.

**RADIO SHACK A DIVISION OF TANDY CORPORATION**

**U.S.A.: FORT WORTH, TEXAS 76102**
**CANADA: BARRIE, ONTARIO L4M 4W5**

**TANDY CORPORATION**

| AUSTRALIA | BELGIUM | U.K. |
|---|---|---|
| 280-316 VICTORIA ROAD<br>RYDALMERE, N.S.W. 2116 | PARC INDUSTRIEL DE NANINNE<br>5140 NANINNE | BILSTON ROAD WEDNESBURY<br>WEST MIDLANDS WS10 7JN |

8759015 – 381

PRINTED IN U.S.A.