User's Reference Guide

For: TELCON'S PORTABLE COMPUTERS

TELCON INDUSTRIES, INC.
1401 N.W. 69th Street
Ft. Lauderdale, Florida
33309

Telephone (305) 971-2250
Telex: 510-956-9412

PRELIMINARY
FCC NOTIFICATION

WARNING: This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. As temporarily permitted by regulation it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interferences. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

On your instrument appears the following label:

This equipment has not been tested to show compliance with new FCC rules (47 CFR Part 15) designed to limit interference to radio and TV reception. Operation of this equipment in a residential area is likely to cause unacceptable interference to radio communication requiring the operator to take whatever steps are necessary to correct the interference.
Parts of this manual where copied, in part or in whole, with the written consent of and are copyrighted by Digital Research, MicroPro International Corporation and Microsoft. All rights to this publication are reserved. No part of this document may be reproduced, transmitted, transcribed, stored in any retrieval system, translated into any other language, in any form, by any means without prior written consent from: Telcon Industries, Inc., 1401 NW 69th Street, Fort Lauderdale, Florida 33309

TRADEMARK

MicroPro, WordStar (Reg.), MailMerge, CalcStar are trademarks of MicroPro International Corporation. CP/M and CBASIC are trademarks of Digital Research. M80, L80, LIB80 and CREF80 are trademarks of Microsoft. SGEN is a trademark of Solutions Technology Inc.

DISCLAIMER

Telcon Industries, Inc. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties or merchantability or fitness for any particular purpose. Further, Telcon Industries, Inc. reserves the right to revise this publication and to make changes from time to time in the content herof without obligation of Telcon Industries, Inc. to notify any person or organization of such revision or changes.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2.0 GENERAL DESCRIPTION OF YOUR SYSTEM</td>
<td>2</td>
</tr>
<tr>
<td>Unpacking and Setup; Attaching Supplemental Devices; Display; Disk Drives; Keyboard; What is on a Disk; Program Definitions</td>
<td></td>
</tr>
<tr>
<td>3.0 GETTING STARTED</td>
<td>14</td>
</tr>
<tr>
<td>Inserting Disk; System Start-up; Formatting; Copying Disk and Operating System Track</td>
<td></td>
</tr>
<tr>
<td>4.0 CP/M</td>
<td>19</td>
</tr>
<tr>
<td>Filenames; Built-in Commands; Transient Commands</td>
<td></td>
</tr>
<tr>
<td>5.0 PRINTING</td>
<td>108</td>
</tr>
<tr>
<td>Copy Screen to Printer; Print all Keystrokes; Print a File</td>
<td></td>
</tr>
<tr>
<td>6.0 COMMUNICATIONS PORT</td>
<td>109</td>
</tr>
<tr>
<td>Receiving Data; Transmitting a File; Transmitting from the Keyboard</td>
<td></td>
</tr>
<tr>
<td>7.0 TERMINAL EMULATION</td>
<td>110</td>
</tr>
<tr>
<td>Heathkit Model H19/Zenith Z19; Cursor Functions; Erasing and Editing</td>
<td></td>
</tr>
<tr>
<td>8.0 AUXILIARY PROM ACCESS</td>
<td>113</td>
</tr>
</tbody>
</table>
## APPENDICES

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Command Summary</td>
<td>114</td>
</tr>
<tr>
<td>B Computer Number Systems</td>
<td>118</td>
</tr>
<tr>
<td>C Line Editing and Output Control</td>
<td>123</td>
</tr>
<tr>
<td>D What is on your System Disk</td>
<td>124</td>
</tr>
<tr>
<td>E Memory and Disk Layout</td>
<td>126</td>
</tr>
<tr>
<td>F Port Addresses and Names</td>
<td>130</td>
</tr>
<tr>
<td>G Keyboard Layout with Address and Values Generated</td>
<td>132</td>
</tr>
<tr>
<td>H Graphic Character Set</td>
<td>135</td>
</tr>
<tr>
<td>I Pin Assignments</td>
<td>138</td>
</tr>
</tbody>
</table>

## GLOSSARY

Page 140

## INDEX

Page 152
HOW TO USE THIS MANUAL
------------------------

THIS MANUAL IS DESIGNED FOR BOTH THE EXPERIENCED USER AND THE BEGINNER.

FOR THE NOVICE, WE RECOMMEND THAT YOU READ THE ENTIRE MANUAL, INCLUDING THE BEGINNING SECTIONS WHICH WILL GUIDE YOU THROUGH SETUP, AND FAMILIARIZE YOU WITH YOUR NEW COMPUTER.

EXPERIENCED USERS MAY GO STRAIGHT TO THE PROGRAM SECTION AFTER SETUP.
1.0 INTRODUCTION

This series of micro-computers has been designed to increase productivity in business and add enjoyment at home by introducing you to the world of micro-computers. It is our aim to make the transition from the pencil and paper mode to computerization as painless as possible.

As you will notice there are no application programs included with your system. This has been done for a reason. We at Telcon Industries, Inc. feel that for us to choose the specific applications packages for you would be unreasonable; instead, we have tried to give you the most flexibility and power for your computing dollar.

Included with your new system is an operating system, CP/M tm, which will handle all of the computer monitoring and internal operations. With CP/M comes a host of utility routines. These routines will make life a lot easier for the newcomer as well as the experienced programmer.

The C/BASIC tm programming language has also been included. This language will allow a person new to computers to start writing programs on day one. If you do not wish to write programs yourself, then commercially written programs are available.

We have prepared this manual to try and answer your questions about the world of micro-computing and about your new Telcon Personal Computer. Please take the time to read this user's guide prior to powering up your new system, it will answer a lot of questions before they occur.
2.0 GENERAL DESCRIPTION OF YOUR SYSTEM

2.1 UNPACKING AND SETUP

The packing material has been designed to conform to the outside contours of your system. If the packaging material is extremely distorted or the external box has been damaged, there is a possibility that your system may have been damaged during shipment. If this is the case, please document any damage to the containers, with pictures if possible, and save should a claim be necessary.

Please save all packing material for future use during moving or shipping, as it will protect your system.

Remove the packing material from around your system and lift it out of the shipping container. Place your system on a suitable work surface. We suggest that your system be located in an area such that it affords proper lighting and viewing.

Grasp the handle and stand the unit up with the legs facing away from you. At this point, the keyboard is at the bottom of the unit. Release the clasps on the side of the unit and pull them clear of the latches. (Refer to figure).

Figure
Place one hand on each side of the unit and lift slightly while rotating the larger unit 90 degrees away from you. The legs on the bottom of the unit should now be extended to the locked position. This will place the unit in the operating position. (Refer to figure).
2.2 ATTACHING SUPPLEMENTAL DEVICES

If you have purchased peripheral devices such as a printer or modem, they may be attached at this time. The communications and printer ports are located below the carrying handle behind the access door. A table of pin placements for the ports is included in Appendix I.
2.3 THE PHYSICAL SYSTEM

This section will describe the physical attributes of the Display, Disk Drives and Keyboard.

2.3.1 THE DISPLAY

The CP/M Operating System (overall controlling body of the computer) communicates with the user through messages formatted and displayed on the CRT. The user responds by typing on the keyboard which also is displayed on the CRT.

The display is a Cathode Ray Tube (CRT) that measures 7" diagonally on the ZORBA, 9" diagonally on the NOMIS IX, and 12" diagonally on the NOMIS XII. This display is 80 characters wide and 25 lines long. Each character is made from a dot matrix that is 5 dots wide by 7 dots long. Graphic characters are made from an 8X10 matrix.

The On/Off switch on the ZORBA also contains the brightness control. Rotating this switch in a clockwise direction will increase the brightness. On the NOMIS IX and XII, a separate rotary knob controls the brightness and the contrast. We suggest that the level of brightness not be overpowering as the CRT phospher could be permanently burned if left on for a long period of time. (Refer to figure).
2.3.2 THE DISK DRIVES

The two disk drives contained in the main unit are the non-volatile storage devices for the system. These disk drives are addressed "Drive A", and "Drive B". (They are also referred to as Drive 0 and Drive 1 respectively). For the ZORBA system, Drive A is on the left and Drive B on the right. For the NOMIS IX and XII systems, Drive A is on the top and Drive B is on the bottom.

It is through these drives that information is written onto and retrieved from the diskette. Like a phonograph, the disk drive has an arm (read/write head) that is lowered onto the diskette to read or write information.

Usually the CP/M disk resides in Drive A and application and data disks reside in Drive B. If different drive assignments are required, then the application program will request the proper assignments.

The physical drive itself has an indicator light and a door that will enclose the disk. When inserting the disk into the drive, the write protect notch must be on the top edge closest to the operator, for the ZORBA system; and for the NOMIS IX and XII systems, the write protect notch is on the left closest to the operator.

2.3.3 THE KEYBOARD

The keyboard case forms the front cover of the portable computer. To place the keyboard in operating position, unlatch the two side catches and place the keyboard in a comfortable typing position. The keyboard is attached to the computer through a standard three foot, four conductor coiled telephone cord. A longer cord, if desired, can be purchased at any electronics or phone store.
The keyboard, shown below, consists of four main areas:

1. Typewriter Area
2. Control Key Area
3. Function Key Area
4. Keypad Area

Figure
2.3.3.1 TYPEWRITER AREA

The center section of the keyboard comprises the typewriter area. The keys here emit codes corresponding to all upper and lower case letters as well as special symbols.

RETURN - This key indicates the termination of an input line to the operating system. In general, this key acts in the same fashion as a standard typewriter.

SHIFT KEYS - When depressed simultaneously with another key in the typewriter area, the SHIFT KEY causes the typing of an upper case letter or the upper symbol on the key cap.

SHIFT LOCK - This lighted key is not part of the primary typing area. Each time you press the key the light will go on. When the SHIFT LOCK light is on, it is as if you were holding down a SHIFT KEY all the time. The SHIFT LOCK light will go off when either of the regular SHIFT keys is depressed.

CAPS LOCK - This lighted key is not part of the primary typing area. Each time you press the key the light will either go on or off. When the CAPS LOCK light is on, only upper case letters (A...Z) will be typed. For all other keys, the lower symbol will normally be typed and the SHIFT KEY must be used to obtain the upper symbol.

2.3.3.2 CONTROL KEY AREA

The control keys surround the main typewriter area and perform special computer related control functions. Except where noted, these keys are generally unaffected by the SHIFT, SHIFT LOCK, and CAPS LOCK keys. Pressing the CTRL key simultaneously with one of these keys may perform an alternate function where noted. The control keys are labeled:

CTRL
ESCAPE
TAB
PRINT  (SHIFT/~) or (CTRL/~)
DEL
BS  (SHIFT/DEL)
HOME
CLEAR  (CTRL/HOME)
left, right, up, down ARROW
BREAK
CTRL - The CTRL key does not issue any code. Rather, this key (when depressed simultaneously with another) causes most other keys to issue a different code than that normally assigned to the key. This is called a control command.

ESCAPE - Pressing the ESCAPE key issues the code for escape. The action taken upon this key is dependant upon the program that is running.

TAB - Pressing the TAB key issues the code for tab. The response to this code is program dependant. Normally the code will cause the display cursor to advance to the next defined tab stop.

BS - Pressing the (SHIFT/DEL) key issues the code for backspace. The response to this code is program dependant. Normally the code will cause the display cursor to move, non-destructively, one character position to the left.

DEL - Pressing the DEL key issues the code for delete. The response to this code is program dependant. Normally, the code will cause the display cursor to move one character position to the left.

PRINT - The PRINT key is a shifted or CTRL key. Pressing the SHIFT (or CTRL) and PRINT key simultaneously will cause the display to be printed on the line printer. No code is generated.

HOME - Pressing the home key will cause the cursor to move to the upper left corner of the terminal screen. No code is generated.

CLEAR - This is not a key that appears on the keyboard; however, pressing the CTRL and HOME keys simultaneously clears the screen from the position of the cursor (including that line), and repositions the cursor to the upper left corner of the terminal screen. No code is generated.

ARROW KEYS - Pressing any of the ARROW keys issues the codes appropriate for that key. The response to this code is program dependant. The ARROW keys may be individually programmed to emit a user definable series of keystrokes. Like Function keys, they do nothing to the screen. (See Function Key Section).

BREAK - Pressing this key causes a 600 millisecond space condition on the communications port (normally called BREAK). This generally signals any telecommunications device transmitting to this computer to cease sending and abort to an idle condition. This is used only when the communications port is busy doing something that knows how to respond to a BREAK.
2.3.3.3 FUNCTION KEY AREA

It is often more advantageous to have a single key emit a series of frequently typed keystrokes rather than a specific code. Along the top of the keyboard are 19 unlabeled keys. These are called programmable function keys. Pressing one of these keys will emit one of the 19 programmed codes. If the CTRL key is depressed simultaneously with one of these keys, a second set of codes will be emitted giving a total of 38 distinct programmable keys.

An example of the use of these keys is that an editor might require two or three separate keystrokes to terminate the current edit, save the file and exit back to the operating system command level. These keystrokes may be stored as a single function key so that when you want to terminate the edit, you press that key and the programmed keystrokes are automatically executed.

The following keys are individually programmable by the use of the SETUP program.

F1...F19
CTRL/F1...CTRL/F19
All the Arrow Keys
CTRL/All key pad keys

The actual number of programmable keys is 55. All are distinct and separate.

2.4 WHAT IS ON A DISK?

A diskette is a thin plastic disk with a magnetic coating where information is magnetically stored. You will notice that the actual diskette is enclosed in a black plastic cover. This is NOT removable. Diskettes are very fragile, and care must be taken with them to avoid any damage which would result in a loss of the information stored on the disk. Do not let the diskette come into contact with any magnetic field, as this would erase the disk. You should also keep the disk in its protective jacket at all times when not in use to keep the exposed parts of the disk protected from dust, scratches, and even the oil on your skin.

The spindle hole is used by the disk drive to turn the diskette (from inside its plastic covering) so that the read/write head may make contact with the diskette through the access slot. It is through this slot that information is read from or written onto the disk.
The index hole is used as the starting position of the diskette surface.

The little notch on the side of the diskette is called the write-protect notch. This notch may be covered with tape to prevent the disk from being written on. Information may still be read from the disk, but nothing may be written on it. This comes in handy if you have a disk with valuable information on it that you want to protect from being accidentally written on or erased.

The diskette is divided into tracks and sectors. Think of a floppy disk as a record, with the grooves on the record being the tracks. On a disk the grooves are there but not visible. A track is an area on a disk that can be accessed by the disk drive without moving the read/write head.

The tracks are allocated as follows:

- **Track 0** - For system use only.
- **Track 1** - Directory (64 segments for the ZORBA, and 128 for the NOMIS), and the balance is used for data.
- **Tracks 2-39** - For data on ZORBA
- **Tracks 2-79** - For data on NOMIS

Since the disk is divided into tracks, then each track must be divided into subsequent areas. Each area is called a sector. These sectors may be of varying length between different manufacturers. We use 20 sectors per track, with 512 bytes per sector, for a total of 400K bytes (ZORBA) and 800K bytes (NOMIS).

There is also a directory on every floppy disk. This directory is like a telephone book for the disk. It will allow the operating system to locate the number of the file when given the name. This number will correspond to the address that the file resides at on the disk. Without the directory there would be no way for the system to locate a file.

A directory segment is a 32 byte field which tells CP/M the name of the file and where the data is located on the disk.
2.5 PROGRAM DEFINITIONS

Up until now a lot of terms have been used that might be meaningless to you. At this point in time some more definitions are required.

2.5.1 WHAT IS AN OPERATING SYSTEM?

Computers are dumb. In order to operate, they must have instructions. These instructions are supplied by a group of programs collectively called the operating system. This operating system is the overall controlling body of the computer. Your computer uses CP/M as its operating system, and through the use of many associated programs and modules, your system will provide high speed data processing.

What does the operating system do for you? The operating system (composed of CCP, BDOS and BIOS, which are names for internal segments of the Operating System) will perform the accessing of the diskettes and the displaying of data on the CRT. The transmission of data through any one of the various ports is also monitored by the operating system.

In case an error occurs, depending on the level of the error, the operating system allows the system to continue functioning. If the error is in the operating system itself, the operating system will notify the user and corrective action may be taken.

2.5.2 WHAT ARE UTILITIES?

Utilities are programs that allow the user to perform the more common tasks required to operate the computer. A prime example of the use of a utility program is when you have to make copies of your diskettes. If utility programs were not provided, then it would be your responsibility to write a program to copy data from one diskette to another. Another example of a utility program is one that allows you to display the contents of the disk directory.

In general, a utility program makes life easier for the computer user or operator.
2.5.3 WHAT IS A LANGUAGE?

In order for you to communicate with your computer you must first learn to speak its language. The base language for your computer is 780 Assembler. This is an assembly type language. This means that before you may execute the program, the program must first be assembled. During the assembly process the assembler will translate the instructions that you wrote, into instructions that the computer can understand. The only problem with an assembly language is that you must first learn the abbreviations for the instructions.

To make life easier for the new computer user, a language called C/BASIC has been included with your system. This is an interpreter type language that is easy to learn, as the instructions are basically English-type words or abbreviations. C/BASIC does not require any steps to convert the language into a machine readable format, as the very nature of the language is that the C/BASIC INTERPRETER will interpret the instructions and convert them to machine instructions as the program runs.

There are also languages that are available for your system that require a compiler to convert them into instructions that your system can understand. These languages are COBOL, FORTRAN, PASCAL, etc. These are more complex and more powerful than C/BASIC, but less powerful than an assembly language.

2.5.4 WHAT IS AN APPLICATION?

An application program is a program that was written by you or for you that will perform a specific task or group of tasks. Application programs range from programs that balance your checkbook to programs that will compute corporate income taxes.

Application programs range in size from only a few instructions to programs that take many years to complete.
3.0 GETTING STARTED

3.1 PLUGGING IT IN

Now that you are familiar with the major components of your new computer, it has come time to plug it in. The system is designed to operate on 110 volts AC plus or minus 10 percent. The unit must use a grounded receptacle and we discourage the use of an extension cord. Before plugging the unit in make sure the On/Off switch is in the Off position. On the ZORBA, the switch is rotated counterclockwise until the click. On the NOMIS IX and XII, the toggle switch on the back of the unit is in the Off position.

BEFORE TURNING THE POWER ON OR OFF, always make sure that the drive doors are open. When power is turned on or off, the power may surge and damage the data on the disk, or the disk itself. When the doors are open, the read/write head cannot make contact with the disk.

Plug it in and turn it On.

At this time, the banner line should be displayed. If it is not displayed, please turn to the troubleshooting section of this guide.
3.2 INSERTING A DISK

To insert the diskette into the disk drive, remove the protective jacket. For the ZORBA system, the manufacturer's label must be on the lower right hand side when held in a vertical position. The write-protect notch will be on the top edge closest to the operator. Open the door on the disk drive and slowly insert the diskette into the drive. When the diskette will go in no further, close and latch the door. There will be a faint click when the door is properly latched.

For the NOMIS IX and XII systems, hold the disk in the horizontal position with the manufacturer's label up and to the right. This will place the write protect notch to the left. Open the disk door and insert the diskette as above. Then close the door until it latches.

NOTE: WHEN THE INDICATOR LIGHT IS ON, THIS SIGNIFIES THAT THE DISK DRIVE IS ACTIVE. DISKS MAY BE INSERTED AND REMOVED WHEN THE COMPUTER IS RUNNING BUT NEVER WHEN THE DRIVE IS ACTIVE.

3.3 SYSTEM START-UP

Insert the system disk (labeled as such) into Disk Drive A: as instructed above, and press the RETURN key. The startup banner will be displayed, and the computer will read the system disk into memory and begin execution. You can tell when the computer is ready for your command when the A> symbol appears.
3.4 FORMATTING A DISK

Prior to using a new disk for the first time it must be formatted. Formatting defines the tracks and sectors and prepares the disk to be written on.

To format a disk, perform the following steps:

* Load the CP/M operating system by depressing the RETURN key. The system will respond with:

  58K CP/M version 2.2

  A>

The symbol A> is the system prompt which indicates that CP/M has been loaded and is waiting for a command.

NOTE: Only upon a cold-start will the CP/M version number message appear.

Insert a blank disk in drive B and respond with:

  A>FORMAT A B G X <CR>

NOTE: <CR> is the symbol for carriage return, this signifies that you should press the return key.

For more detailed information on formatting, refer to the Formatting section of Transient Commands.
3.5 COPYING A DISK

Before using any original disk, it is highly recommended that it be copied to a newly formatted disk, and that this copy be used instead of the original. By doing so you will always have a backup copy of your system, as the diskettes are not an infallible storage media. It is also not uncommon for one to be inadvertently written on, thus destroying one of your systems.

To copy your diskettes, perform the following with CP/M loaded and a formatted diskette in Drive B:

A>PIP B:=A:*.*[V] <CR>

The above command will access the "Peripheral Interchange Program" and copy all files from the diskette in Drive A to Drive B. The following will be a sample listing of what will be displayed by the PIP program:

COPYING-
PIP.COM
TEST.SYS
LINK.COM
etc.

A>

The above will copy all the files on the disk but not the system track. The [V] in the command may be omitted to improve copying time, but the copy may contain undetected errors which may not be found until the programs are used.

[V] means "verify" during COPY.
3.6 COPYING THE OPERATING SYSTEM TRACK

The Operating System Track is the invisible part of the disk (does not show up on the directory) which contains the CCP, BDOS, and BIOS modules comprising the Operating System. Copying files with PIP (see previous procedure) does not copy the Operating System track, therefore, the new disk cannot be "booted". If this ability is desired when making a copy of a "bootable" disk, the operating system track must also be copied. In other words, the operating system track must be copied to bring CP/M to an operational state. To copy the system track respond to the following interactive dialogue from the SGEN utility:

A>SGEN <CR>

SGEN VERSION x.x
SOURCE DRIVE NAME (OR RETURN TO EXIT): A
SOURCE ON A THEN TYPE RETURN <CR> (after source in drive A)

FUNCTION COMPLETE
DESTINATION ON B THEN TYPE RETURN <CR> (after diskette inserted in drive B)

FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO EXIT): <CR>
As described earlier, we are using the CP/M operating system. Included with CP/M are various routines and programs that will make operating your computer as easy as learning to walk. Most of the time spent using your computer will be that of writing and/or running programs, but there will come a time when you will have to do some "system-type" work. This section of the User's Guide will provide you with the working knowledge that will walk you through the steps required to perform these tasks.

The operating system or monitor is made up of many modules. The module that the user interfaces with primarily is referred to as CCP. This module is responsible for reading and interpreting the commands that the user types. It is also the responsibility of CCP to interface with the proper diskette that has been specified by the user.

4.1 THE COMMAND LINE

A disk is "logged in" if the CCP is currently addressing it. In order to clearly indicate which disk is in use, the CCP prompts the user with the disk name followed by the ">" symbol, indicating that CCP is waiting for a command.

After the initial sign-on of the computer, CP/M automatically prompts the user with "A>" (indicating that CP/M is currently addressing Disk "A") and waits for a command. These commands are of two types, Built-in and Transient. These two types of commands will be discussed in detail later in this chapter.
4.1.1 SWITCHING FROM ONE DRIVE TO ANOTHER

The currently logged disk can be switched to the other by typing the disk drive name followed by a (:) when the CCP is waiting for a command. See example:

A>DIR List all files on disk A.
SAMPLE ASM
SAMPLE PRN
A>B: Switch to disk B.
B>DIR *.ASM List all "ASM" files on B.
DUMP ASM
FILES ASM
B>A: Switch back to A.
A>

4.2 FILENAMES

Data is stored on the diskettes in the form of files. The files must be given a name so that they may be referenced.

The file naming convention used in this system conforms to the CP/M file naming convention.

xxxxxxxx.xxx

Where: xxxxxxxx = the primary name
       yyy = the secondary name

The primary name is used to distinguish particular files from another. The primary name may consist of from 1 to a maximum of 8 characters.

The secondary name (also referred to as the extension) is used to differentiate file types (a C/BASIC file name could have the secondary name .BAS). This separates files of one group (C/BASIC, for example) from those of all other files.
The only characters that may not be used in filenames are:

< > . , ; : = ? * [ ]

All other special, numeric and alphabetic characters may be used.

There are two types of filenames used to reference files that reside on the diskettes; unambiguous and ambiguous. The unambiguous file reference specifies exactly the file to be accessed while, the ambiguous file reference will access file names similar to the specified. An example of ambiguous referencing is

A>DIR TE<T.BAS <CR>

This would result in the following files to be displayed:

TEXT.BAS     TEST.BAS

If the command A>DIR TE<T.* were entered, the display would be:

TEXT.BAS     TEST.BAS     TEST.COM
TEXT.DOC

Using ambiguous file references allows the user to access a file when the complete filename is not known.

"Wildcard" (ambiguous name or type) filenames may be used when displaying selective file types; i.e.,

A>DIR *.BAS

would display all files with the secondary name of BAS.

Where as:

A>DIR TEST.*

would display all filenames with the primary name of TEST and having any secondary name.

If the filename is preceded by the drive notation A: or B:, then the file on the drive specified would be accessed.
4.3 INDIVIDUAL COMMANDS

The individual commands available from CP/M are broken down into two major groups:

1) BUILT-IN COMMANDS which are part of the CCP, and nothing is loaded from the disk.

2) TRANSIENT COMMANDS which are loaded from the currently logged disk and executed in the Transient Program Area.

4.3.1 BUILT-IN COMMANDS

Built-in commands are commands that are programmed into the operating system itself. The following commands will access the disk in some fashion.

ERASE - Erase specified file.

DIR - Produce a list of filenames for the specified diskette directory.

RENAME - Rename a specified file.

SAVE - Save memory contents in a file.

TYPE - Copy the contents of the specified file to the CRT.

USER - Allows restriction of certain files from unauthorized users.

All built-in commands (except DIR and USER) require a filename to be specified.

These built-in commands are described in detail in the following paragraphs.
4.3.1.1 DIR - DISPLAY THE DISK DIRECTORY.

The DIR command will allow the user to display the file names of all the files that reside in either the "A" or "B" drive. To perform this function do the following:

A>DIR <CR>

This will display the directory for the currently logged drive (A:).

To display a group of files but not the complete directory, do the following:

A>DIR TEST.* <CR>

This will display all the files with the primary name of TEST and any secondary name.

To display all the files within a specified group, type the following:

A>DIR *.BAS <CR>

This will display all the files with the secondary name of BAS.

To check the validity of a particular file as to whether or not it exists, type the following:

A>DIR file.ext <CR>

If the file exists, the entry in the directory will be displayed. If the file does not exist, an error message will be displayed.
4.3.1.2 ERA - ERASE A FILE

This function will allow the user to erase a specified file do the following:

A>ERA file.ext <CR>

The filename specified above could take the form:

file.ext Where file.ext is the only file erased.

TEST.* Where the files with the primary name TEST and any secondary names are erased.

*.* Where the files with the secondary name of BAS and any primary name are erased.

.* This form would erase all files from the directory.

TE?T.BAS This would erase the files having the extension of .BAS and any character substituted for the ?.

USE CAUTION WHEN USING "WILDCARDS" WITH THE ERASE COMMAND. IF YOU'RE NOT CAREFUL, IT IS POSSIBLE TO DELETE MORE FILES THAN DESIRED.

To specify a file that is in Drive B, prefix the file-name with 'B:'.

4.3.1.3 REN - RENAME A DISK FILE

This function allows the user to change the name of an existing file to a different filename. The form of the entry is:

A>REN NEW.EXT=OLD.EXT <CR>

In the above example, the file OLD.EXT is renamed as NEW.EXT. If either filename was preceded by a drive name, then both files would reside on the specified drive. If both filenames are preceded by drive names, then both drive names must be the same.
4.3.1.4 SAVE - MAKE A FILE FROM CONTENTS OF MEMORY

The form of this command is:

A>SAVE n file.ext

where n is the number of pages (in decimal) to be saved from memory, and file.ext is the name given to the new file.

Memory is saved in 256 byte increments, which is equal to one page of memory. The TPA starts at 100H (hex) which is the second page of memory. (The first page of memory consists of special system related information).

For example, if the user's program occupies the area from 100H (the 256th byte of memory) through 2FFH (the 767th byte of memory) the total number of bytes to be saved would be 511 (767-256), thus requiring 2 pages of memory, at 256 bytes per page.

An easier way to find the number of pages is by using the STAT command. The STAT command will give you the size of the file in kilobytes. Divide this number by 4 to get the approximate page number.

See the following examples:

A>SAVE 3 X.COM Would save 3 pages (at 256 bytes each) to X.COM

A>SAVE 40 Q (.EXT not necessary) Would save 28H pages (at 256 bytes each) to Q. (Note that 28 is the page number, and that 28=2*16+8=40 decimal).

NOTE: SEE APPENDIX B - COMPUTER NUMBER SYSTEMS

An example of the use of the SAVE command would be if you were working on a program and something went wrong, you could save the file and look at it later to make the appropriate corrections.
4.3.1.5 TYPE - DISPLAY OF ASCII DISK FILE

The TYPE command will display on the CRT the specified file. The form of the command is:

A>TYPE FILE.EXT <CR>

The system will access the requested file and display it in ASCII on the CRT. If the file contains TABS, they will be expanded with spaces to every 8 character positions from the left screen edge.

A file from a specific drive may be displayed by preceding the filename with the disk drive name:

A>TYPE B:FILE.EXT <CR>

4.3.1.6 USER

CP/M allows the assignment of up to 16 user areas which are capable of being assigned certain files for security purposes.

The "system files" are accessible to all user areas, whereas certain files may be restricted to a specific user number.

The only form of the command is:

A> USER n

where n is an integer from 0-15. Upon cold-start, the operator is automatically logged into the user area 0, which is accessible by all users.

The operator may issue the user command at any time to move to another area within the same directory; however, only those files authorized by his user number(s) will become available.

The active user number is maintained until changed by another user command or until a cold start operation, where user 0 is assumed.

Note that if no user number is specified, 0 is automatically assumed.
4.3.2 TRANSIENT COMMANDS

Transient commands or functions are files with the extension .COM and load from the currently logged disk drive to execute in the TPA (Transient Program Area). The TPA is actually the main section (about 90%) of memory, dedicated to Transient Programs run from disk via Transient commands. Any program file on disk of the type .COM can be a transient command. The system (CCP) will search the disk for .COM files when a command other than ERA, TYPE, REN, DIR, SAVE or USER (built-in commands) is typed, loading and running the .COM file whose name matches the command given.

Example: A>PIP <CR>

causes the CCP to search the A: disk for PIP.COM, and run it if found. (The extension .COM is assumed, so it need not be specified.)

This system of command names vs .COM files allows an unlimited set of "commands" to be built into CP/M for any imaginable application. Provided with the ZORBA and NOMIS computers are a number of .COM programs, each of which provides a Transient command and set of functions necessary to operate the computer. These transient commands are:

STAT - statistical information about disk drive.
ASM - load the CP/M assembler.
LOAD - load a file that is in machine code and translate to machine executable form which can be loaded into the TPA.
DDT - load the CP/M dynamic debugging tool into TPA and start execution.
PIP - load peripheral interchange program
ED - load CP/M text editor
SGEN - create new CP/M system diskette
SUBMIT - submit file containing commands for batch processing
DUMP - dump the contents of file in hexadecimal
MOVCPM - reconfigure CP/M for different system size
PATCH - allows CP/M to be altered
SETUP - allows the ZORBA user to select the operational characteristics of the computer
FORMAT - prepares a new disk to be written on
Transient commands are executed the same as built-in commands, except that a DRIVE specification (as with filenames) may appear immediately before the command.

Example: A>B:PIP <CR>

This runs PIP from drive B: to allow the transient program to run, though it may not be on the currently logged drive.

4.3.2.1 DUMP - HEXADECIMAL DISPLAY OF A DISK FILE

To obtain a display of a disk file in hexadecimal format type the following:

A>DUMP file.ext <CR>

The system will access the specified file and convert the data to a hexadecimal display format, and then output the data to the CRT or the printer. This function is useful when trying to debug a program and the input or output data does not agree with the format specified.

4.3.2.2 STAT - DISPLAY THE STATISTICS FOR DISK.

The STAT function will display statistics for a specified disk drive. The form of the entry is:

A>STAT A: <CR>

The above request will display the statistics for the requested drive as follows:

BYTES REMAINING ON A: nnnK

Where nnn is the number of kilobytes (thousand) available.

If the request were typed:

A>STAT <CR>

The system would respond:

A: R/W, SPACE: nnnK
or
B: R/O, SPACE: nnnK

This will repeat for all assigned drives with R/W and R/O implying read/write and read/only respectively.
The command line can also specify a set of files to be scanned by STAT by means of an ambiguous file reference.

The files which satisfy the reference are listed in alphabetical order, with storage requirements for each file under the heading. For example:

```
A>STAT *.EXT <CR>
A>STAT FILE.* <CR>
```

Would cause the files satisfying these references to be displayed alphabetically in the form:

```
RECS BYTES EXT ACC A:FILENAME.TYP
  rrrr  bbbK  ee  a  d  FILE.EXT
```

Where:
- rrrr = number of 128 byte records
- bbb = number of kilobytes allocated
- ee  = number of 16K extensions
- a   = read/write access
- d   = drive name
- FILE = primary filename
- EXT  = secondary file name

As a convenience, the drive name may be given ahead of the ambiguous file reference.

```
A>STAT B:FILE.* <CR>
```

The following form sets the drive (x) to read-only, which remains in effect until the next warm or cold boot.

```
A>STAT x:=R/O <CR>
```

The message

```
BDOS ERR ON x: READ ONLY
```

will appear if there is an attempt to write to the read-only disk x. CP/M waits until a key is depressed before performing an automatic warm boot, at which time the disk becomes R/W; therefore, the only reliable way to protect a disk from being written on is to cover the write-protect notch with tape.
The STAT command also allows control over the physical to logical device assignment. In general, there are four logical peripheral devices which are, at any particular instant, each assigned to one of several physical peripheral devices. The four logical devices are named:

- **CON:** The system console device (used by CCP for communication with the operator).
- **RDR:** The paper tape reader device
- **PUN:** The paper tape punch device
- **LST:** The output list device (printer)

The actual devices attached to the ZORBA computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device is actually the main communications serial port input, and PUN: is the same port as output. All SETUP port A selections are effective through the RDR: and PUN: device channels. In order to allow some flexibility in device assignment, several logical devices are defined, as shown below:

**CON: Assignments**

- **CRT:** Local screen and keyboard default.
- **TTY:** Local screen and keyboard OR send and receive data via the main communications port.
- **UCl:** Same effect as CTRL-F toggle, echoes all output to screen through the LST: device. This does not cancel at "warm-boot" time, as does CTRL-F.
- **BAT:** Not used, same as CRT:

**RDR: Assignments**

All assignments (TTY, PTR, UR1, UR2) point to the main communications port for input.

**PUN: Assignments**

All assignments (TTY, PIP, UP1, UP2) point to the main communications port for output.
LST: Assignments

LPT: Output sent to the printer serial port using SETUP configuration.

ULL: Output redirected to PUN: instead of the printer serial port. This allows translation for LST: which is available only on the main communications port via SETUP.

CRT: Not used, same as LPT:.

TTY: Not used, same as LPT:

NOTE that SETUP printer configured as "Centronics parallel" overrides the LST: selections above, always sending printer data via the parallel facility on the printer port connector.

It must be emphasized that the physical device names may or may not actually correspond to devices which the names imply. That is, the PTP: device has been implemented as the communications port. The driving subroutine is defined in the BIOS portion of CP/M.

The possible logical to physical device assignments can be displayed by typing:

A>STAT VAL: <CR>

STAT prints the possible values which can be taken on for each logical device:

CON: = TTY:   CRT: = BAT:   UCl: 
RDR: = TTY:   PTR: = URL:   UR2: 
PUN: = TTY:   PTP: = UPl:   UP2: 
LST: = TTY:   CRT: = LPT:   ULL:

In each case, the logical device shown to the left can take any of the four physical assignments shown to the right on each line. The current logical to physical mapping is displayed by typing the command:

A>STAT DEV: <CR>

which produces a listing of each logical device to the left, and the current corresponding physical device to the right. For example, the list might appear as follows:

CON: = CRT: 
RDR: = URL: 
PUN: = PTP: 
LST: = TTY:
The current logical to physical device assignment can be changed by typing a STAT command of the form

\[ \text{A}\text{>STAT }1d1 = pd1, \ldots, 1dn = pdn \quad \text{<CR>} \]

where \(1d1\) through \(1dn\) are logical device names, and \(pd1\) through \(pdn\) are compatible physical device names (i.e., \(1d1\) and \(pd1\) appear on the same line in the "VAL:" command shown above). The following are valid STAT commands which change the current logical to physical device assignments:

\[ \text{A}\text{>STAT CON:}=\text{CRT:} \quad \text{<CR>} \]

\[ \text{A}\text{>STAT PUN:} = \text{TTY:}, \text{LST:}=\text{LPT:}, \text{RDR:}=\text{TTY:} \quad \text{<CR>} \]

The command form

\[ \text{A}\text{>STAT DSK:} \]

produces a drive characteristics table for all currently active drives. The final STAT command form is:

\[ \text{A}\text{>STAT USR:} \]

which produces a list of the user numbers which have files on the currently addressed disk. The display format is:

\[
\begin{align*}
\text{Active User} & : 0 \\
\text{Active Files} & : 0 \ 1 \ 3
\end{align*}
\]

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In the above case, the active user number is 0 (default at cold start), with three user numbers which have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging-in with USER 1, USER 2 or USER 3 commands, followed by a DIR command at the command level.
4.3.2.3  ASM - LOAD AND EXECUTE CP/M 8080 ASSEMBLER

The ASM command will load and execute the CP/M 8080 assembler. The filename specified on the command line will become the input for the assembler, the output from the assembler will be stored in a file with the same primary name as the input file but with the secondary name of .PRN. A second file is produced by the assembler which contains the machine code and has a secondary name of .HEX. The command line would take the form:

```
A>ASM filename  <CR>
or
A>ASM filename.parms  <CR>
```

In both cases, the assembler assumes there is a file on the diskette with the extension .ASM, so it need not be specified.

The first and second forms shown above differ only in that the second form allows parameters to be passed to the assembler to control source file access and hex and print file destinations.

In either case, the CP/M assembler loads, and prints the message:

```
CP/M ASSEMBLER VER n.n
```

where n.n is the current version number. In the case of the first command, the assembler reads the source file with assumed file type "ASM" and creates two output files:

```
filename.HEX
or
filename.PRN
```

The "HEX" file contains the machine code corresponding to the original program in hex format, and the "PRN" file contains an annotated listing showing generated machine code, error flags, and source lines. If errors occur during translation, they will be listed in the PRN file as well as at the CRT console.

The second command form can be used to redirect input and output files from their defaults. In this case, the "parms" portion of the command is a three letter group which specifies the origin of the source file, the destination of the hex file, and the destination of the print file. The form is:

```
filename.p1p2p3
```

where p1, p2, and p3 are single letters
p1: \( A, B, \ldots, Y \) designates the disk name which contains the source file

p2: \( A, B, \ldots, Y \) designates the disk name which will receive the hex file

\( Z \) skips generation of hex file

p3: \( A, B, \ldots, Y \) Designates the disk name which will receive the print file

\( X \) Places listing at the console CRT.

\( Z \) Skips generation of print file

Thus, the command

ASM X.AAA

Indicates that the source file (X.ASM) is to be taken from disk A, and that the hex (X.HEX) and print (X.PRN) files are to be created also on disk A. This form of the command is implied if the assembler is run from disk A. That is given that the operator is currently addressing disk A, the above command is equivalent to

ASM X

The command

ASM X.ABX

indicates that the source file is to be taken from disk A, the hex file is placed on disk B, and the listing file is to be sent to the console CRT. The command

ASM X.BZZ

takes the source file from disk B and skips the generation of the hex and print files (this command is useful for fast execution of the assembler to check program syntax).

The source program format is compatible with both the Intel 8080 assembler as well as the Processor Technology Software Package #1 assembler. That is, the CP/M assembler accepts source programs written in either format. There are certain extensions in the CP/M assembler which make it somewhat easier to use.

These extensions are described in the following paragraphs.
4.3.2.3.1 PROGRAM FORMAT

An assembly language program acceptable as input to the assembler consists of a sequence of the form

```
line#  label  operation  operand  ;comment
```

where any or all of the fields may be present in a particular instance. Each assembly language statement is terminated by a carriage return and line feed (the line feed is inserted automatically by the ED program), or with the character "!" which is treated as an end-of-line by the assembler (thus, multiple assembly language statements can be written on the same physical line if separated by ! symbols).

The line# is an optional decimal integer value representing the source program line number, which is allowed on any source line to maintain compatibility with the Processor Technology format. In general, these line numbers will be inserted if a line-oriented editor is used to construct the original program, and thus ASM ignores this field if present.

The label field takes the form

```
identifier
or
identifier:
```

and is optional, except where noted in particular statement types. The identifier is a sequence of alphanumeric characters where the first character is alphabetic. Identifiers can be freely used by the programmer to label elements such as program steps and assembler directives, but cannot exceed 16 characters in length. All characters are significant in an identifier, except for the embedded dollar symbol ($) which can be used to improve readability of the name. Further, all lower case alphabettics are treated as if they were upper case. Note that the "!" following the identifier in a label is optional (to maintain compatibility between Intel and Processor Technology). Thus, the following are all valid instances of labels:

```
x         xy      long$name
x:        yxl:    longer$name$data:
X1Y2      X1X2    x234$5678$9012$3456:
```

The operation field contains either an assembler directive, or pseudo operation, or an 8080 machine operation code. The pseudo operations and machine operation codes are described below.
The operand field of the statement, in general, contains an expression formed out of constants and labels, along with arithmetic and logical operations on these elements. Again, the complete details of properly formed expressions are given below.

The comment field contains arbitrary characters following the ";" symbol until the next real or logical end-of-line. These characters are read, listed, and otherwise ignored by the assembler. In order to maintain compatibility with the Processor Technology assembler, the CP/M assembler also treats statements which begin with an "*" in column one as comment statements, which are listed and ignored in the assembly process. Note that the Processor Technology assembler has the side effect in its operation of ignoring the characters after the operand field has been scanned. This causes an ambiguous situation when attempting to be compatible with Intel's language, since arbitrary expressions are allowed in this case. Hence, programs which use this side effect to introduce comments, must be edited to place a ";" before these fields in order to assemble correctly.

The assembly language program is formulated as a sequence of statements of the above form, terminated optionally by an END statement. All statements following the END are ignored by the assembler.

4.3.2.3.2 FORMING THE OPERAND

In order to completely describe the operation codes and pseudo operations, it is necessary to first present the form of the operand field, since it is used in nearly all statements. Expressions in the operand field consist of simple operands (labels, constants, and reserved words), combined in properly formed subexpressions by arithmetic and logical operators. The expression computation is carried out by the assembler as the assembly proceeds. Each expression must produce a 16-bit value during the assembly. Further, the number of significant digits in the result must not exceed the intended use. That is, if an expression is to be used in a byte move immediate instruction, then the most significant 8 bits of the expression must be zero. The restrictions on the expression significance is given with the individual instructions.
4.3.2.3.3 LABELS

As discussed above, a label is an identifier which occurs on a particular statement. In general, the label is given a value determined by the type of statement which it precedes. If the label occurs on a statement which generates machine code or reserves memory space (e.g. a MOV instruction, or a DS pseudo operation), then the label is given the value of the program address which it labels. If the label precedes an EQU or SET, then the label is given the value which results from evaluating the operand field. Except for the SET statement, an identifier can label only one statement.

When a label appears in the operand field, its value is substituted by the assembler. This value can then be combined with other operands and operators to form the operand field for a particular instruction.

4.3.2.3.4 NUMERIC CONSTANTS

A numeric constant is a 16-bit value in one of several bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. The radix indicators are:

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Radix</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Binary (base 2)</td>
</tr>
<tr>
<td>O</td>
<td>Octal (base 8)</td>
</tr>
<tr>
<td>Q</td>
<td>Octal (base 8)</td>
</tr>
<tr>
<td>D</td>
<td>Decimal (base 10)</td>
</tr>
<tr>
<td>H</td>
<td>Hexadecimal (base 16)</td>
</tr>
</tbody>
</table>

Q is an alternate radix indicator for octal numbers since the letter O is easily confused with the digit 0. Any numeric constant which does not terminate with a radix indicator is assumed to be a decimal constant.

A constant is thus composed as a sequence of digits, followed by an optional radix indicator, where the digits are in the appropriate range for the radix. That is, binary constants must be composed of 0 and 1 digits, octal constants can contain digits in the range 0-7, while decimal constants contain decimal digits. Hexadecimal constants contain decimal digits as well as hexadecimal digits A (10D), B (11D), C (12D), D (13D), E (14D) and F (15D). Note that the leading digit of a hexadecimal constant must be a decimal digit in order to avoid confusing a hexadecimal constant with an identifier (a leading 0 will always suffice). A constant composed in this manner must evaluate to a binary number which can be contained within a 16-bit counter, otherwise it is truncated on the right by the assembler. Similar to identifiers, imbedded "$" are allowed within constants to improve their readability.
Finally, the radix indicator is translated to upper case if a lower case letter is encountered. The following are all valid instances of numeric constants.

\[
\begin{array}{cccc}
1234 & 1234D & 1100B & 1111$0000$1111$0000B \\
1234H & 0FFEH & 33770 & 33577$22Q \\
33770 & 0fe3h & 1234d & 0fffh \\
\end{array}
\]

4.3.2.3.5 RESERVED WORDS

There are several reserved character sequences which have predefined meanings in the operand field of a statement. The names of 8080 registers are given below, which, when encountered, produce the value shown to the right.

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>L</td>
<td>5</td>
</tr>
<tr>
<td>M</td>
<td>6</td>
</tr>
<tr>
<td>SP</td>
<td>6</td>
</tr>
<tr>
<td>PSW</td>
<td>6</td>
</tr>
</tbody>
</table>

(Again, lower case names have the same values as their upper case equivalents). Machine instructions can also be used in the operand field, and evaluate to their internal codes. In the case of instructions which require operands, where the specific operand becomes a part of the binary bit pattern of the instruction (e.g., MOV A,B), the value of the instruction (in this case MOV) is the bit pattern of the instruction with zeroes in the optional fields (e.g., MOV produces 40H).

When the symbol "$" occurs in the operand field (not imbedded within identifiers and numeric constants) its value becomes the address of the next instruction to generate, not including the instruction contained within the current logical line.
4.3.2.3.6 STRING CONSTANTS
--------------------------------

String constants represent sequences of ASCII characters, and are represented by enclosing the characters within apostrophe symbols ('). All strings must be fully contained within the current physical line (thus allowing "!") symbols within strings), and must not exceed 64 characters in length. The apostrophe character itself can be included within a string by representing it as a double apostrophe ("'"), which becomes a single apostrophe when read by the assembler. In most cases, the string length is restricted to either one or two characters (the DB pseudo operation is an exception), in which case the string becomes an 8 or 16 bit value, respectively. Two character strings become a 16-bit constant, with the second character as the low order byte, and the first character as the high order byte.

The value of a character is its corresponding ASCII code. There is no case translation within strings, and thus both upper and lower case characters can be represented. Note however, that only graphic (printing) ASCII characters are allowed within strings. Valid strings are

'A', 'AB', 'ab', 'c',
'... ', 'a', '.... ', '.... ',
'Walla Walla Wash.',
'She said 'Hello' to me.',
'I said 'Hello' to her.'

4.3.2.3.7 ARITHMETIC AND LOGICAL OPERATORS
---------------------------------------------

The operands described above can be combined in normal algebraic notation using any combination of properly formed operands, operators, and parenthesized expressions. The operators recognized in the operand field are

\[ a + b \quad \text{unsigned arithmetic sum of } a \text{ and } b \]
\[ a - b \quad \text{unsigned arithmetic difference between } a \text{ and } b \]
\[ + b \quad \text{unary plus (produces } b) \]
\[ - b \quad \text{unary minus (same as } 0 - b) \]
\[ a \times b \quad \text{unsigned magnitude multiplication of } a \text{ and } b \]
\[ a \div b \quad \text{unsigned magnitude division of } a \text{ and } b \]
a MOD b remainder after a / b

NOT b logical inverse of b (all 0's become 1's, 1's become 0's), where b is considered a 16-bit value

a AND b bit-by-bit logical AND of a and b

a OR b bit-by-bit logical OR of a and b

a XOR b bit-by-bit logical EXCLUSIVE OR of a and b

a SHL b the value which results from shifting a to the left by an amount b, with zero fill

a SHR b the value which results from shifting a to the right by an amount b, with zero fill

In each case, a and b represent simple operands (labels, numeric constants, reserved words, and one or two character strings), or fully enclosed parenthesized subexpressions such as

\[
10+20 \quad 10h+37Q \quad L1/3 \quad (L2+4) \quad SHR \ 3
\]

\[
('a' \ and \ 5fh) + '0' \quad ('B' + B) \ OR \ (PSW + M)
\]

\[
(1+(2+c)) \ shr \ (A-(B+1))
\]

Note that all computations are performed at assembly time as 16-bit unsigned operations. Thus, -1 is computed as 0-1 which results in the value 0ffffh (i.e., all 1's). The resulting expression must fit the operation code in which it is used. If, for example, the expression is used in a ADI (add immediate) instruction, then the high order eight bits of the expression must be zero. As a result, the operation "ADI -1" produces an error message (-1 becomes 0ffffh which cannot be represented as an 8 bit value), while "ADI (-1) and 0FFH" is accepted by the assembler since the "AND" operation zeroes the high order bits of the expression.
4.3.2.3.8 PRECEDENCE OF OPERATORS

As a convenience to the programmer, ASM assumes that operators have a relative precedence of application which allows the programmer to write expressions without nested levels of parentheses. The resulting expression has assumed parentheses which are defined by the relative precedence. The order of application of operators in unparenthesized expressions is listed below. Operators listed first have highest precedence (they are applied first in an unparenthesized expression), while operators listed last have lowest precedence. Operators listed on the same line have equal precedence, and are applied from left to right as they are encountered in an expression.

* / MOD SHL SHR
- +
NOT
AND
OR XOR

Thus, the expressions shown to the left below are interpreted by the assembler as the fully parenthesized expressions shown to the right below

a * b + c   (a * b) + c
a + b * c   a + (b * c)
a MOD b * c SHL d   ((a MOD b) * c) SHL d

a OR b AND NOT c + d SHL e = a OR (b AND (NOT (c + (d SHL e)))

Balanced parenthesized subexpressions can always be used to override the assumed parentheses, and thus the last expression above could be written to force application of operators in a different order as

(a OR b) AND (NOT c) + d SHL e

resulting in the assumed parentheses

(a OR b) AND ((NOT c) + (d SHL e))

Note that an unparenthesized expression is well-formed only if the expression which results from inserting the assumed parentheses is well-formed.
4.3.2.3.9 ASSEMBLER DIRECTIVES

Assembler directives are used to set labels to specific values during the assembly, perform conditional assembly, define storage areas, and specify starting addresses in the program. Each assembler directive is denoted by a "pseudo operation" which appears in the operation field of the line. The acceptable pseudo operations are

- **ORG** set the program or data origin
- **END** end program, optional start address
- **EQU** numeric "equate"
- **SET** numeric "set"
- **IF** begin conditional assembly
- **ENDIF** end of conditional assembly
- **DB** define data bytes
- **DW** define data words
- **DS** define data storage area

The individual pseudo operations are detailed below.

4.3.2.3.9.1 The ORG Directive

The ORG statement takes the form

```
label ORG expression
```

where "label" is an optional program label, and expression is a 16-bit expression, consisting of operands which are defined previous to the ORG statement. The assembler begins machine code generation at the location specified in the expression. There can be any number of ORG statements within a particular program, and there are no checks to ensure that the programmer is not defining overlapping memory areas. Note that most programs written for the CP/M system begin with an ORG statement of the form

```
ORG 100H
```

which causes machine code generation to begin at the base of the CP/M transient program area. If a label is specified in the ORG statement, then the label is given the value of the expression (this label can then be used in the operand field of other statements to represent this expression).
4.3.2.3.9.2 The END Directive

The END statement is optional in an assembly language program, but if it is present it must be the last statement (all subsequent statements are ignored in the assembly). The two forms of the END directive are:

```
label   END
label   END   expression
```

where the label is again optional. If the first form is used, the assembly process stops, and the default starting address of the program is taken as 0000. Otherwise, the expression is evaluated, and becomes the program starting address (this starting address is included in the last record of the Intel formatted machine code "hex" file which results from the assembly). Thus, most CP/M assembly language programs end with the statement

```
END 100H
```

resulting in the default starting address of 100H (beginning of the transient program area).

4.3.2.3.9.3 The EQU Directive

The EQU (equate) statement is used to set up synonyms for particular numeric values. The form is

```
label   EQU   expression
```

where the label must be present, and must not label any other statement. The assembler evaluates the expression, and assigns this value to the identifier given in the label field. The identifier is usually a name which describes the value in a more human-oriented manner. Further, this name is used throughout the program to "parameterize" certain functions. Suppose for example, that data received from a Teletype appears on a particular input port, and data is sent to the Teletype through the next output port in sequence. The series of equate statements could be used to define these ports for a particular hardware environment:

```
TTYBASE   EQU   10H   ;BASE PORT NUMBER FOR TTY
TTYIN     EQU   TTYBASE   ;TTY DATA IN
TTYOUT    EQU   TTYBASE+1   ;TTY DATA OUT
```

At a later point in the program, the statements which access the Teletype could appear as:

```
IN       TTYIN    ;READ TTY DATA TO REG-A
OUT      TTYOUT   ;WRITE DATA TO TTY FROM REG-A
```
Making the program more readable than if the absolute i/o ports had been used. Further, if the hardware environment is redefined to start the Teletype communications ports at 7FH instead of 10H, the first statement need only be changed to:

TTYBASE EQU 7FH ;BASE ORT NUMBER FOR TTY

and the program can be reassembled without changing any other statements.

4.3.2.3.9.4 The SET Directive
---------------------------------------

The SET statement is similar to the EQU, taking the form

label SET expression

except that the label can occur on other SET statements within the program. The expression is evaluated and becomes the current value associated with the label. Thus, the EQU statement defines a value which is valid from the current SET statement to the point where the label occurs on the next SET statement. The use of the SET is similar to the EQU statement, but is used most often in controlling conditional assembly.

4.3.2.3.9.5 The IF and ENDIF Directives
---------------------------------------

The IF and ENDIF statements define a range of assembly language statements which are to be included or excluded during the assembly process. The form is:

IF expression
    statement#1
    statement#2
    ...
    statement#n
ENDIF
Upon encountering the IF statement, the assembler evaluates the expression following the IF (all operands in the expression must be defined ahead of the IF statement). If the expression evaluates to a non-zero value, then statement\#1 through statement\#n are assembled; if the expression evaluates to zero, then the statements are listed but not assembled. Conditional assembly is often used to write a single "generic" program which includes a number of possible run-time environments, with only a few specific portions of the program selected for any particular assembly. The following program segments for example, might be part of a program which communicates with either a Teletype or a CRT console (but not both) by selecting a particular value for TTY before the assembly begins:

```
TRUE EQU 0FFFH ;DEFINE VALUE OF TRUE
FALSE EQU NOT TRUE ;DEFINE VALUE OF FALSE
;
TTY EQU TRUE ;TRUE IF TTY, FALSE IF CRT
;
TTYBASE EQU 10H ;BASE OF TTY I/O PORTS
CRTBASE EQU 20H ;BASE OF CRT I/O PORTS
IF TTY ;ASSEMBLE RELATIVE TO TTYBASE
CONIN EQU TTYBASE ;CONSOLE INPUT
CONOUT EQU TTYBASE+1 ;CONSOLE OUTPUT
ENDIF
;
IF NOT TTY ;ASSEMBLE RELATIVE TO CRTBASE
CONIN EQU CRTBASE ;CONSOLE INPUT
CONOUT EQU CRTBASE+1 ;CONSOLE OUTPUT
ENDIF
...
IN CONIN ;READ CONSOLE DATA
...
OUT CONOUT ;WRITE CONSOLE DATA
```

In this case, the program would assemble for an environment where a Teletype is connected, based at port 10H. The statement defining TTY could be changed to

```
TTY EQU FALSE
```

and, in this case, the program would assemble for a CRT based at port 20H.
4.3.2.3.9.6 The DB Directive
--------------------------------------

The DB directive allows the programmer to define initialize storage areas in single precision (byte) format. The statement form is

    label    DB    e#1, e#2, ..., e#n

where e#1 through e#n are either expressions which evaluate to 8-bit values (the high order eight bits must be zero), or are ASCII strings of length no greater than 64 characters. There is no practical restriction on the number of expressions included on a single source line. The expressions are evaluated and placed sequentially into the machine code file following the last program address generated by the assembler. String characters are similarly placed into memory starting with the first character and ending with the last character. Strings of length greater than two characters cannot be used as operands in more complicated expressions (i.e., they must stand alone between the commas). Note that ASCII characters are always placed in memory with the parity bit reset (0). Further, recall that there is no translation from lower to upper case within strings. The optional label can be used to reference the data area throughout the remainder of the program. Examples of valid DB statements are:

    data:    DB    0,1,2,3,4,5       
            DB    data and 0ffh,5,377Q,1+2+3+4
    signon: DB    'please type your name', cr,lf,0
             DB    'AB' SHR 8, 'C', 'DE' AND 7FH

4.3.2.3.9.7 The DW Directive
--------------------------------------

The DW statement is similar to the DB statement except double precision (two-byte) words of storage are initialized. The form is

    label    DW    e#1, e#2, ..., e#n

where e#1 through e#n are expressions which evaluate to 16-bit results. Note that ASCII strings of length one or two characters are not allowed. In all cases, the data storage is consistent with the 8080 processor: the least significant byte of the expression is stored first in memory, followed by the most significant byte. Examples are:

    doub:    DW    0ffeh,doub+4,signon-$,255+255
             DW    'a', 5, 'ab', 'CD', 6 shl 8 or 11b
4.3.2.3.9.8 The DS Directive

The DS statement is used to reserve an area of uninitialized memory, and takes the form

```assembly
label    DS    expression
```

where the label is optional. The assembler begins subsequent code generation after the area reserved by the DS. Thus, the DS statement given above has exactly the same effect as the statement

```assembly
label:   EQU    $    ;LABEL VALUE IS CURRENT CODE LOCATION
         ORG    $+expression  ;MOVE PAST RESERVED AREA
```

4.3.2.4.0 OPERATION CODES

Assembly language operation codes form the principal part of assembly language programs, and form the operation field of the instruction. Labels are optional on each input line and, if included, take the value of the instruction address immediately before the instruction is issued. The individual operators are listed briefly in the following sections for completeness. In each case,

- **e3** represents a 3-bit value in the range 0-7 which can be one of the predefined registers A, B, C, D, E, H, L, M, SP, or PSW.
- **e8** represents an 8-bit value in the range 0-255
- **e16** represents a 16-bit value in the range 0-65535

which can themselves be formed from an arbitrary combination of operands and operators. In some cases, the operands are restricted to particular values within the allowable range, such as the PUSH instruction. These cases will be noted as they are encountered.

In the sections which follow, each operation code is listed in its most general form, along with a specific example, with a short explanation and special restrictions.
### 4.3.2.4.1 Jumps, Calls, and Returns

The JUMP, CALL, and RETURN instructions allow several different forms which set the condition flags in the 8080 microcomputer CPU. The forms are:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP e16</td>
<td>JMP L1</td>
<td>Jump unconditionally to label</td>
</tr>
<tr>
<td>JNZ e16</td>
<td>JMP L2</td>
<td>Jump on non-zero condition to label</td>
</tr>
<tr>
<td>JZ e16</td>
<td>JMP 100H</td>
<td>Jump on zero condition to label</td>
</tr>
<tr>
<td>JNC e16</td>
<td>JNC L1+4</td>
<td>Jump no carry to label</td>
</tr>
<tr>
<td>JC e16</td>
<td>JC L3</td>
<td>Jump on carry to label</td>
</tr>
<tr>
<td>JPO e16</td>
<td>JPO S+8</td>
<td>Jump on parity odd to label</td>
</tr>
<tr>
<td>JPE e16</td>
<td>JPE L4</td>
<td>Jump on even parity to label</td>
</tr>
<tr>
<td>JP e16</td>
<td>JP GAMMA</td>
<td>Jump on positive result to label</td>
</tr>
<tr>
<td>JM e16</td>
<td>JM a1</td>
<td>Jump on minus to label</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL e16</td>
<td>CALL S1</td>
<td>Call subroutine unconditionally</td>
</tr>
<tr>
<td>CNZ e16</td>
<td>CNZ S2</td>
<td>Call subroutine if non-zero flag</td>
</tr>
<tr>
<td>CZ e16</td>
<td>CZ 100H</td>
<td>Call subroutine on zero flag</td>
</tr>
<tr>
<td>CNC e16</td>
<td>CNC S1+4</td>
<td>Call subroutine if no carry set</td>
</tr>
<tr>
<td>CC e16</td>
<td>CC S3</td>
<td>Call subroutine if carry set</td>
</tr>
<tr>
<td>CPO e16</td>
<td>CPO S+8</td>
<td>Call subroutine if parity odd</td>
</tr>
<tr>
<td>CPE e16</td>
<td>CPE S4</td>
<td>Call subroutine if parity even</td>
</tr>
<tr>
<td>CP e16</td>
<td>CP GAMMA</td>
<td>Call subroutine if positive result</td>
</tr>
<tr>
<td>CM e16</td>
<td>CM blc2</td>
<td>Call subroutine if minus flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST e3</td>
<td>RST 0</td>
<td>Programmed &quot;restart&quot;, equivalent to CALL 8*e3, except one byte call</td>
</tr>
</tbody>
</table>

Return from subroutine:
- RET
- RNZ
- RZ
- RNC
- RC
- RPO
- RPE
- RP
- RM

Return if non-zero flag set: RET
Return if zero flag set: RET
Return if no carry: RET
Return if carry flag set: RET
Return if parity is odd: RET
Return if parity is even: RET
Return if positive result: RET
Return if minus flag is set: RET
### 4.3.2.4.2 Immediate Operand Instructions

Several instructions are available which load single or double precision registers, or single precision memory cells, with constant values, along with instructions which perform immediate arithmetic or logical operations on the accumulator (register A).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVI e3,e8</td>
<td>MVI B,255</td>
</tr>
<tr>
<td></td>
<td>Move immediate data to register A,B,C,D,E,H,L or M (memory)</td>
</tr>
<tr>
<td>ADI e8</td>
<td>ADI 1</td>
</tr>
<tr>
<td></td>
<td>Add immediate operand to A without carry</td>
</tr>
<tr>
<td>ACI e8</td>
<td>ACI OFFH</td>
</tr>
<tr>
<td></td>
<td>Add immediate operand to A with carry</td>
</tr>
<tr>
<td>SUI e8</td>
<td>SUI L + 3</td>
</tr>
<tr>
<td></td>
<td>Subtract from A without borrow (carry)</td>
</tr>
<tr>
<td>SBI e8</td>
<td>SBI L AND 11B</td>
</tr>
<tr>
<td></td>
<td>Subtract from A with borrow (carry)</td>
</tr>
<tr>
<td>ANI e8</td>
<td>ANI $ AND 7FH</td>
</tr>
<tr>
<td></td>
<td>Logical &quot;and&quot; A with immediate data</td>
</tr>
<tr>
<td>XRI e8</td>
<td>XRI 1111$0000B</td>
</tr>
<tr>
<td></td>
<td>&quot;Exclusive or&quot; A with immediate data</td>
</tr>
<tr>
<td>ORI e8</td>
<td>ORI L AND 1+1</td>
</tr>
<tr>
<td></td>
<td>Logical &quot;or&quot; A with immediate data</td>
</tr>
<tr>
<td>CPI e8</td>
<td>CPI 'a'</td>
</tr>
<tr>
<td></td>
<td>Compare A with immediate data (same as SUI except register A not changed)</td>
</tr>
<tr>
<td>LXI e3,e16</td>
<td>LXI B,100H</td>
</tr>
<tr>
<td></td>
<td>Load extended immediate to register pair (e3 must be equivalent to B,D,H or SP)</td>
</tr>
</tbody>
</table>
4.3.2.4.3 Increment and Decrement Instructions

Instructions are provided for incrementing or decrementing single and double precision registers. The instructions are:

<table>
<thead>
<tr>
<th>Single precision increment register (e3 produces one of A,B,C,D,E,H,L,M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single precision decrement register (e3 produces one of A,B,D,D,E,H,L,M)</td>
</tr>
<tr>
<td>Double precision increment register pair (e3 must be equivalent to B,D,H, or SP)</td>
</tr>
<tr>
<td>Double precision decrement register pair (e3 must be equivalent to B,D,H, or SP)</td>
</tr>
</tbody>
</table>
4.3.2.4.4 Data Movement Instructions

Instructions which move data from memory to the CPU and from the CPU to memory are given below:

MOV e3,e3  MOV A,B  Move data to leftmost element from rightmost element (e3 produces one of A,B,C,D,E,H,L, or M). MOV M,M disallowed.

LDAX e3  LDAX B  Load register A from computed address (e3 must produce either B or D)

STAX e3  STAX D  Store register A to computed address (e3 must produce either B or D)

LHLD e16  LHLD L1  Load HL direct from location e16 (double precision load to H and L)

SHLD e16  SHLD L5+x  Store HL direct to location e16 (double precision store from H and L to memory)

LDA e16  LDA Gamma  Load register A from address e16

STA e16  STA X3-5  Store register A into memory at e16

POP e3  POP PSW  Load register pair from stack, set SP (e3 must produce one of B,D,H, or PSW)

PUSH e3  PUSH B  Store register pair into stack, set SP (e3 must produce one of B,D,H, or PSW)

IN e8  IN 0  Load register A with data from port e8

OUT e8  OUT 255  Send data from register A to port e8

XTHL  Exchange data from top of stack with HL

PCHL  Fill program counter with data from HL

SPHL  Fill stack pointer with data from HL

XCHG  Exchange DE pair with HL pair
### 4.3.2.4.5 Arithmetic Logic Unit Operations

Instructions which act upon the single precision accumulator to perform arithmetic and logic operations are:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD e3</td>
<td>Add register given by e3 to accumulator without carry (e3 must produce one of A,B,C,D,E, H, or L)</td>
</tr>
<tr>
<td>ADD B</td>
<td>Add register to A with carry, e3 as above</td>
</tr>
<tr>
<td>ADC e3</td>
<td>Add register to A with carry, e3 as above</td>
</tr>
<tr>
<td>ADC L</td>
<td>Subtract reg e3 from A without carry, e3 is defined as above</td>
</tr>
<tr>
<td>SUB e3</td>
<td>Subtract register e3 from A with carry, e3 defined as above</td>
</tr>
<tr>
<td>SUB H</td>
<td>Logical &quot;and&quot; reg with A, e3 as above</td>
</tr>
<tr>
<td>SBB e3</td>
<td>&quot;Exclusive or&quot; with A, e3 as above</td>
</tr>
<tr>
<td>SBB 2</td>
<td>Logical &quot;or&quot; with A, e3 as above</td>
</tr>
<tr>
<td>ANA e3</td>
<td>Compare register with A, e3 as above</td>
</tr>
<tr>
<td>ANA 1+1</td>
<td>Decimal adjust register A based upon last arithmetic logic unit operation</td>
</tr>
<tr>
<td>XRA e3</td>
<td>Complement the bits in register A</td>
</tr>
<tr>
<td>XRA A</td>
<td>Set the carry flag to 1</td>
</tr>
<tr>
<td>ORA e3</td>
<td>Complement the carry flag</td>
</tr>
<tr>
<td>ORA B</td>
<td>Rotate bits left, (re)set carry as a side effect (high order A bit becomes carry)</td>
</tr>
<tr>
<td>CMP e3</td>
<td>Rotate bits right, (re)set carry as side effect (low order A bit becomes carry)</td>
</tr>
<tr>
<td>CPM H</td>
<td>Rotate carry/A register to left (carry is involved in the rotate)</td>
</tr>
<tr>
<td>DAD e3</td>
<td>Rotate carry/A register to right (carry is involved in the rotate)</td>
</tr>
<tr>
<td>DAD B</td>
<td>Double precision add register pair e3 to HL (e3 must produce B,D,H, or SP)</td>
</tr>
</tbody>
</table>
4.3.2.4.6 Control Instructions

The four remaining instructions are categorized as control instructions, and are listed below:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLT</td>
<td>Halt the 8080 processor</td>
</tr>
<tr>
<td>DI</td>
<td>Disable the interrupt system</td>
</tr>
<tr>
<td>EI</td>
<td>Enable the interrupt system</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
</tr>
</tbody>
</table>

4.3.2.4.7 Error Messages

When errors occur within the assembly language program, they are listed as single character flags in the leftmost position of the source listing. The line in error is also echoed at the CRT console so the source listing need not be examined to determine if errors are present. The error codes are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Data error: element in data statement cannot be placed in the specified data area</td>
</tr>
<tr>
<td>E</td>
<td>Expression error: expression is ill-formed and cannot be computed at assembly time</td>
</tr>
<tr>
<td>L</td>
<td>Label error: Label cannot appear in this context (may be duplicate label)</td>
</tr>
<tr>
<td>N</td>
<td>Not implemented: features which will appear in future ASM versions (e.g., macros) are recognized, but flagged in this version</td>
</tr>
<tr>
<td>O</td>
<td>Overflow: expression is too complicated (i.e., too many pending operators) to be computed; simplify it</td>
</tr>
<tr>
<td>P</td>
<td>Phase error: label does not have the same value on two subsequent passes through the program</td>
</tr>
<tr>
<td>R</td>
<td>Register error: the value specified as a register is not compatible with the operation code</td>
</tr>
<tr>
<td>V</td>
<td>Value error: operand encountered in expression is improperly formed</td>
</tr>
</tbody>
</table>
Several error messages are printed which are due to terminal error conditions:

**NO SOURCE FILE PRESENT**
The file specified in the ASM command does not exist on the disk

**NO DIRECTORY SPACE**
The disk directory is full, erase files which are not needed, and retry

**SOURCE FILE NAME ERROR**
Improperly formed ASM file name (e.g., it is specified with "?" fields)

**SOURCE FILE READ ERROR**
Source file cannot be read properly by the assembler; execute a TYPE to determine the point of error

**OUTPUT FILE WRITE ERROR**
Output files cannot be written properly, most likely cause is a full disk; erase and retry

**CANNOT CLOSE FILE**
Output file cannot be closed, check to see if disk is write-protected

4.3.2.5 LOAD - LOAD MACHINE READABLE CODE

The LOAD command reads any .HEX file (typically created by the assembler) and translates it into an executable file. It assumes the extension to be .HEX, so it does not have to be specified. The command for LOAD has the form:

```
A>LOAD FILE <CR>
```

This converts the .HEX file to a .COM file which can be executed from the command line as a transient command.

As with the ASM command, if the file resides in Drive B or the non-logged drive, precede the file name with the drive name.
4.3.2.6 PIP - PERIPHERAL INTERCHANGE PROGRAM

PIP is the CP/M Peripheral Interchange Program which implements the basic media conversion necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms

    PIP <CR>

or

    PIP "command line" <CR>

In both cases, PIP is loaded into the TPA and executed. If the first form is used, PIP reads command lines directly from the CRT console, prompting the user with the "*" asking for information on what to do. When input is completed, the user should respond with a <CR> only.

The second form of the PIP command is equivalent to the first except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting.

In all cases for the second example, the form of the input is:

    Destination = source#1, source#2,..., source#n <CR>

where "destination" is the file or peripheral device to receive the data, and "source#1,...,source#n" represents a series of one or more files or devices which are copied from left to right of the destination.

The destination and source elements can be unambiguous references to CP/M source files, with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name which defines the particular drive where the file may be obtained or stored. When the drive name is not included, the currently logged disk is assumed. Further, the destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If the destination file already exists, it is removed if the command line is properly formed (it is not removed if an error condition arises). The following command lines (with explanations to the right) are valid as input to PIP.

    X = Y

Copy to file X from file Y, where X and Y are unambiguous file names; Y remains unchanged.
\[ X = Y, Z \]

Concatenate files \( Y \) and \( Z \) and copy to file \( X \), with \( Y \) and \( Z \) unchanged.

\[ X.ASM= Y.ASM, Z.ASM, FIN.ASM \]
Create the file \( X.ASM \) from the concatenation of the \( Y.ASM \), \( Z.ASM \) and \( FIN.ASM \) files.

\[ NEW.ZOT = B:OLD.ZAP \]
Move a copy of \( OLD.ZAP \) from drive \( B \) to the currently logged disk; name the file \( NEW.ZOT \).

\[ B:FILE.4=B:FILE.1, A:FILE.2, FILE.3 \]
Concatenate \( FILE.1 \) from drive \( B \) with \( FILE.2 \) from drive \( A \) and \( FILE.3 \) from the logged disk; create the file \( FILE.4 \) on drive \( B \).

For more convenient use, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated forms are:

\[
\begin{align*}
\text{PIP } x:=\text{filename} & \quad \text{(CR)} \\
\text{PIP } x:=y:\text{filename} & \quad \text{(CR)} \\
\text{PIP filename} = y & \quad \text{(CR)} \\
\text{PIP } x:\text{filename} = y & \quad \text{(CR)}
\end{align*}
\]

The first form copies the file from the currently logged disk to the same file name on drive \( x \) (\( x = A..P \)).

The second form is equivalent to the first, where the source for the copy is drive \( y \) (\( y = A..P \)).

The third form is equivalent to the command

\[ x:\text{filename} = y:\text{filename} \]

which copies the file given from drive \( y \) to a file on drive \( x \).

The fourth form is equivalent to the third, where the source disk is explicitly given by \( y \).
Note that the source and destination disks must be different in all of these cases. If the filename is specified, PIP lists it as it is being copied. If a file exists by the same name on the destination drive, it is deleted upon successful completion of the copy, and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

\[
\begin{align*}
B: &= * . COM \\
A: &= B: ZAP . * \\
ZAP . ASM &= B: \\
B: ZOT . COM &= A: ZOT . COM
\end{align*}
\]

Copy all files which have the secondary name "COM" to drive B from the currently logged drive.

Copy all files which have the primary name "ZAP" to drive A from drive B.

Same as \( ZAP . ASM = B: ZAP . ASM \)

Same as \( B: ZOT . COM = A: ZOT . COM \)

PIP also allows reference to physical and logical devices which are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The logical devices given in the STAT command are:

- CON: console
- RDR: reader
- PUN: punch
- LST: list

while the physical devices are:

- TTY: console, reader, punch, or list
- CRT: console, or list
- PTR: reader
- PTP: punch
- LPT: list
- URL: reader
- UPI: punch
- ULI: list
- UC1: console
- UR2: reader
- UP2: punch

(Note that the "BAT:" physical device is not included, since this assignment is used only to indicate that the RDR: and LST: devices are to be used for console input/output.)
The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M, and thus are easily altered for any particular I/O system. The destination device must be capable of generating data (i.e., the LST: device cannot be read).

The additional device names which can be used in PIP commands are:

**NUL:**
Send 40 "nulls" (ASCII 0's) to the device (this can be issued at the end of punched output).

**EOF:**
Send a CP/M end-of-file (ASCII ^Z) to the destination device (set automatically at the end of all ASCII data transfers through PIP).

**INP:**
Special PIP input source which can be "patched" into the PIP program itself: PIP gets the input data character-by-character by CALLing location 103H, with data returned in location 109H (parity bit must be zero).

**OUT:**
Special PIP output destination which can be patched into the PIP program: PIP CALLS location 106H with data in register C for each character to transmit. Note that locations 109H through 1FFH of the PIP memory image are not used and can be replaced by special purpose drivers using DDT.

**PRN:**
Same as LST:, except that tabs are expanded at every eighth character position, lines are numbered, and page ejects are inserted every 60 lines, with an initial eject.

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (^Z for ASCII files, and a real end of file for non-ASCII disk files). Data from each device or file is concatenated from left to right until the last data source has been read. The destination device or file is written using the data from the source files, and an end-of-file character (^Z) is appended to the result for ASCII files. Note if the destination is a disk file, then a temporary file is created (.$$$ secondary name) which is changed to the actual file name only upon successful completion of the copy. Files with the extension .COM are always assumed to be non-ASCII.
The copy operation can be aborted at any time by depressing any key on the keyboard (a delete suffices). PIP will respond with the message "ABORTED" to indicate that the operation was not completed. Note that if any operation is aborted, or if an error occurs during processing, PIP removes any pending commands which were set up while using the SUBMIT command.

It should also be noted that PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the CRT console and waits for corrective action.

Valid PIP commands are shown below:

- **PIP LST: = X.PRN**  
  Copy X.PRN to the LST device and terminate the PIP program.

- **PIP**  
  Start PIP for a sequence of commands (PIP prompts with "*").

- **CON:=X.ASM,Y.ASM,Z.ASM**  
  Concatenate three ASM files and copy to the CON device.

- **X.HEX=CON:,Y.HEX,PTR:**  
  Create a HEX file by reading the CON (until a ^z is typed), followed by data from Y.HEX, followed by data from PTR until ^Z is encountered.

- **<CR>**  
  Single carriage return stops PIP.

- **PIP PUN:=NUL:,X.ASM,EOF:,NUL:**  
  Send 40 nulls to the punch device; then copy the X.ASM file to the punch, followed by an end-of-file (^Z) and 40 more nulls.
The user can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). The valid PIP parameters are listed below.

B  Block mode transfer: data is buffered by PIP until an ASCII x-off character (^S) is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data which can be buffered is dependent upon the memory size of the host system (PIP will issue an error message if the buffers overflow).

Dn  Delete characters which extend past column n in the transfer of data to the destination from the character source. This parameter is used most often to truncate long lines which are sent to a (narrow) printer or console device.

E  Echo all transfer operations to the console as they are being performed.

F  Filter form feeds from the file. All imbedded form feeds are removed. The P parameter can be used simultaneously to insert new form feeds.

H  Hex data transfer: all data is checked for proper Intel hex file format. Non-essential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur.

I  Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).

L  Translate upper case alphabettics to lower case.

N  Add line numbers to each line transferred to the destination starting at one, and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, then leading zeroes are included, and a tab is inserted following the number. The tab is expanded if T is set.
0 Object file (non-ASCII) transfer: the normal CP/M end-of-file is ignored.

Pn Include page ejects at every n lines (with an initial page eject). If n=1, or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form feed suppression takes place before the new page ejects are inserted.

Qs^z Quit copying from the source device or file when the string s (terminated by "z) is encountered.

Ss^z Start copying from the source device when the string s is encountered (terminated by ^z). The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine). The start and quit strings are always included in the copy operation.

NOTE - the strings following the s and q parameters are translated to upper case by the CCP if form (2) of the PIP command is used. Form (1) of the PIP invocation, however, does not perform the automatic upper case translation.

1 PIP <CR>
2 PIP "command line" <CR>

Tn Expand tabs (^I characters) to every nth column during the transfer of characters to the destination from the source.

U Translate lower case alphabets to upper case during the copy operation.

V Verify that data has been copied correctly by rereading after the write operation (the destination must be a disk file).

Z Zero the parity bit on input for each ASCII character.
The following are valid PIP commands which specify parameters in the file transfer:

**PIP X.ASM=B: [v]**  Copy X.ASM from drive B to the current drive and verify that the data was properly copied.

**PIP LPT:=X.ASM [nt8u]**  Copy X.ASM to the LPT: device; number each line, expand tabs to every eighth column, and translate lower case alphabets to upper case.

**PIP PUN:=X.HEX [i], Y.ZOT [h]**  First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; then continue the transfer of data by reading Y.ZOT, which contains hex records, including any ":00" records which it contains.

**PIP PRN:=X.ASM [p50]**  Send X.ASM to the LST: device, with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. Note that nt8p60 is the assumed parameter list for a PRN file; p50 overrides the default value.
4.3.2.7 DDT - Dynamic Debugging Tool

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. The debugger is initiated by typing one of the following commands at the CP/M console command level.

```
DDT
DDT filename.HEX
DDT filename.COM
```

where "filename" is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in the place of the Console Command Processor, and thus resides directly below the Basic Disk Operating System (BDOS) portion of the CP/M. The BDOS starting address, which is located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command shown above perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the sequence of commands

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

Where the I and R commands set up and read the specified program to test (see the explanation of the I and R commands below for exact details).

Upon initiation, DDT prints a sign-on message in the format

```
DDT vers 2.2
```

Following the sign-on message, DDT prompts the operator with the character "#" and waits for input commands from the console. The operator can type any of several single character commands, terminated by a carriage return to execute the command. Each line of input can be line-edited using the standard CP/M controls

- **DELETE**: Remove last character typed
- **CTRL-U**: Remove the entire line, ready for re-typing
- **CTRL-C**: System reboot
Any command can be up to 32 characters in length (an automatic carriage return is inserted as the 33rd character), where the first character determines the command type:

A  enter assembly language mnemonics with operands
D  display memory in hexadecimal and ASCII
F  fill memory with constant data
G  begin execution with optional breakpoints
I  set up a standard input file control block
L  list memory using assembler mnemonics
M  move a memory segment from source to destination
R  read program for subsequent testing
S  substitute memory values
T  trace program execution
U  untraced program monitoring
X  examine and optionally alter the CPU state
The command character, in some cases, is followed by zero, one, two, or three hexadecimal values which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. In all cases, the commands are not executed until the carriage return is typed at the end of the command.

At any point in the debug run, the operator can stop execution of DDT using either a CTRL-C or GO (jmp to location 0000H), and save the current memory image using a SAVE command of the form

```
SAVE n filename.COM
```

where n is the number of pages (256-byte blocks) to be saved on disk. The number of blocks can be determined by taking the high order byte of the top load address and converting this number to decimal. For example, if the highest address in the Transient Program Area is 1234H then the number of pages is 12H, or 18 in decimal. Thus the operator could type a CTRL-C during the debug run, returning to the Console Processor level, followed by

```
SAVE 18 X.COM
```

The memory image is saved as X.COM on the diskette, and can be directly executed by simply typing the name X. If further testing is required, the memory image can be recalled by typing

```
DDT X.COM
```

which reloads the previously saved program from location 100H through page 18 (12FFH). The machine state is not a part of the .COM file, and thus the program must be restarted from the beginning in order to properly test it.
4.3.2.7.1 DDT COMMANDS

The individual commands are given below in some detail. In each case, the operator must wait for the prompt character (-) before entering the command.

In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, where the numbers are represented by lower case letters. These numbers are always assumed to be in a hexadecimal radix, and from one to four digits in length (longer numbers will be automatically truncated on the right).

Many of the commands operate upon a "CPU state" which corresponds to the program under test. The CPU state holds the registers of the program being debugged, and initially contains zeroes for all registers and flags except for the program counter (P) and stack pointer (S), which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded (see the I and R commands).

4.3.2.7.1.1 A (ASSEMBLE) COMMAND

DDT allows inline assembly language to be inserted into the current memory image using the A command which takes the form

As

where s is the hexadecimal starting address for the inline assembly. DDT prompts the console with the address of the next instruction to fill, and reads the console, looking for assembly language mnemonics, followed by register references and operands in absolute hexadecimal form. Each successive load is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, the operator can review the memory segment using the DDT disassembler (see the L command).

Note that the assembler/disassembler portion of DDT can be overlayed by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used.
4.3.2.7.1.2 D (DISPLAY) COMMAND
-----------------------------

The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. The forms are,

D  
Ds  
Ds,f

In the first case, memory is displayed from the current display address (initially 100H), and continues for 16 display lines. Each display line takes the form shown below

aaaa bb bb bb bb bb bb bb ... ccccccccccccccccc

where aaaa is the display address in hexadecimal, and bb represents data present in memory starting at aaaa. The ASCII characters starting at aaaa are given to the right (represented by the sequence of c's), where non-graphic characters are printed as a period (.) symbol. Note that both upper and lower case alphabets are displayed, and thus will appear as upper case symbols on a console device that supports only upper case. Each display line gives the values of 16 bytes of data, except that the first line displayed is truncated so that the next line begins at an address which is a multiple of 16.

The second form of the D command shown above is similar to the first, except that the display address is first set to address s. The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pushing the delete key.

4.3.2.7.1.3 F (FILL) COMMAND
-----------------------------

The F command takes the form

Fs,f,c

where s is the starting address, f is the final address, and c is a hexadecimal byte constant. The effect is as follows: DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f then the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.
4.3.2.7.1.4 G (GO) COMMAND

Program execution is started using the G command, with up to two optional breakpoint addresses. The G command takes one of the forms

\[ \begin{align*}
G \\
Gs \\
Gs,b \\
Gs,b,c \\
G,b \\
G,b,c
\end{align*} \]

The first form starts execution of the program under test at the current value of the program counter in the current machine state, with no breakpoints set. The current program counter can be viewed by typing an X or XP command. The second form is similar to the first except that the program counter in the current machine state is set to address s before execution begins. The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered. The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are subsequently cleared. The last two forms take the program counter from the current machine state, and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. That is, there is no intervention between the starting address and the break address by DDT. Upon encountering a breakpoint, DDT stops execution and types

\[ *d \]

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. The operator must specify breakpoints which differ from the program counter address at the beginning of the G command. Thus if the current program counter is 1234H, then the commands

\[ \begin{align*}
G,1234 \\
G400,400
\end{align*} \]

both produce an immediate breakpoint, without executing any instructions whatsoever.
4.3.2.7.1.5 I (INPUT) COMMAND

The I command allows the operator to insert a file name into the default file control block at 5CH (the file control block created by CP/M for transient programs is placed at this location). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. Note that this file name is also used by DDT for reading additional HEX and COM files. The form of the I command is

I filename

or

I filename.filetype

If the second form is used, and the filetype is either HEX or COM, then subsequent R commands can be used to read the pure binary or hex format machine code (see the R command for further details).

4.3.2.7.1.6 L (LIST) COMMAND

The L command is used to list assembly language mnemonics in a particular program region. The forms are

    L
    Ls
    Ls,f

The first command lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s, and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. Upon encountering an execution breakpoint, the list address is set to the current value of the program counter (see the G and T commands). Again, long typeouts can be aborted using the delete key during the list process.

4.3.2.7.1.7 M (MOVE) COMMAND

The M command allows block movement of program or data areas from one location to another in memory. The form is

    Ms,f,d

where s is the start address of the move, f is the final address of the move, and d is the destination address. Data is first moved from s to d, and both addresses are incremented. If s exceeds f then the move operation stops, otherwise the move operation is repeated.
4.3.2.7.1.8 R (READ) COMMAND
-------------------------------------

The R command is used in conjunction with the I command to
read COM and HEX files from the diskette into the transient
program area in preparation for the debug run. The forms
are:

R
Rb

where b is an optional bias address which is added to each
program or data address as it is loaded. The load operation
must not overwrite any of the system parameters from 000H
through 0FFH (i.e., the first page of memory). If b is
omitted, then b=0000 is assumed. The R command requires a
previous I command, specifying the name of a HEX or COM
file. The load address for each record is obtained from
each individual HEX record, while an assumed load address of
100H is taken for COM files. Note that any number of R
commands can be issued following the I command to re-read
the program under test, assuming the tested program does not
destroy the default area at 5CH. Further, any file specified
with the filetype "COM" is assumed to contain machine code in
pure binary form (created with the LOAD or SAVE command), and
all others are assumed to contain machine code in Intel hex
format (produced, for example, with the ASM command).

Recall that the command

DDT filename.filetype

which initiates the DDT program is equivalent to the
commands

DDT
-filename.filetype
-R

Whenever the R command is issued, DDT responds with either the
either indicator "?" (file cannot be opened, or a checksum
error occurred in a HEX file), or with a load message taking
the form

NEXT PC
nnnn pppp

where nnnn is the next address following the loaded program,
and pppp is the assumed program counter (100H for COM files,
or taken from the last record if a HEX file is specified).
4.3.2.7.1.9 S (SET) COMMAND

The S command allows memory locations to be examined and optionally altered. The form of the command is

\[ \text{Ss} \]

where \( s \) is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in the memory location. If the operator types a carriage return, then the data is not altered. If a byte value is typed, then the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period (.) is typed by the operator, or an invalid input value is detected.

4.3.2.7.2.0 T (TRACE) COMMAND

The T command allows selective tracing of program execution for 1 to 65535 program steps. The forms are

\[ \text{T} \]
\[ \text{Tn} \]

In the first case, the CPU state is displayed, and the next program step is executed. The program terminates immediately, with the termination address displayed as

\[ \text{*hhhh} \]

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for \( n \) steps (\( n \) is a hexadecimal value) before a program breakpoint occurs. A breakpoint can be forced in the trace mode by typing a delete character. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.
Note that program tracing is discontinued at the interface to CP/M, and resumes after return from CP/M to the program under test. Thus CP/M functions which access I/O devices, such as the diskette drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real time since DDT gets control after each user instruction is executed. Interrupt processing routines can be traced. The trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

Note also that the operator should use the delete key to get control back to DDT during trace, in order to ensure that the trace for the current instruction is completed before interruption.

4.3.2.7.2.1 U (UNTRACE) COMMAND

The U command is identical to the T command except that the intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (0FFFFH) steps to be executed in monitored mode, and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

4.3.2.7.2.2 X (EXAMINE) COMMAND

The X command allows selective display and alteration of the current CPU state for the program under test. The forms are:

```
X
Xr
```

where r is one of the 8080 CPU registers

<table>
<thead>
<tr>
<th>register</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Carry Flag</td>
<td>(0/1)</td>
</tr>
<tr>
<td>Z</td>
<td>Zero Flag</td>
<td>(0/1)</td>
</tr>
<tr>
<td>M</td>
<td>Minus Flag</td>
<td>(0/1)</td>
</tr>
<tr>
<td>E</td>
<td>Even Parity Flag</td>
<td>(0/1)</td>
</tr>
<tr>
<td>I</td>
<td>Interdigit Carry</td>
<td>(0/1)</td>
</tr>
<tr>
<td>A</td>
<td>Accumulator</td>
<td>(0-FP)</td>
</tr>
<tr>
<td>B</td>
<td>BC register pair</td>
<td>(0-FFFF)</td>
</tr>
<tr>
<td>D</td>
<td>DE register pair</td>
<td>(0-FFFF)</td>
</tr>
<tr>
<td>H</td>
<td>HL register pair</td>
<td>(0-FFFF)</td>
</tr>
<tr>
<td>S</td>
<td>Stack pointer</td>
<td>(0-FFFF)</td>
</tr>
<tr>
<td>P</td>
<td>Program Counter</td>
<td>(0-FFFF)</td>
</tr>
</tbody>
</table>
In the first case, the CPU register state is displayed in the format

\[ \text{CfZfMfEfIff=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst} \]

where \( f \) is a 0 or 1 flag value, \( bb \) is a byte value, and \( dddd \) is a double byte quantity corresponding to the register pair. The "inst" field contains the disassembled instruction which occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where \( r \) is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed then the flag or register value is not altered. If a value in the proper range is typed, then the flag or register value is altered. Note that BC, DE, and HL are displayed as register pairs. Thus, the operator types the entire register pair when B, C, or the BC pair is altered.

4.3.2.7.2 IMPLEMENTATION NOTES

The organization of DDT allows certain non-essential portions to be overlaid in order to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the Console Command Processor, and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus which, in turn, contains a JMP instruction to the BDOS. Thus, programs which use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process, then the DDT program again alters the address field at 6H to include this module, thus further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a "?" in response), and the trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.
This program allows the user to create, alter, and save text in the CP/M environment. The command line for the text editor is as follows:

A>ED FILENAME,EXT <CR>

If the filename specified does not exist, the system will create a new file and then allow the insertion of text. If the file already exists, the text is passed through a memory buffer where it can be altered from the CRT. Upon completion of the edit, the memory buffer is written onto a temporary file having the same primary name as the original file, but the secondary name of .$$$.

The secondary name of the original file is changed to .BAK so that the most recently edited file may be recovered when necessary. The temporary file is then changed from filename.$$ back to its original name.

If filename.BAK already exists, then it is replaced with the most recent edit.

If a drive name is specified after the filename to be edited, the system will create its temporary file on the drive specified, and rename the original file as .BAK on the original drive.

There are a number of commands provided with the ED program which are useful in creating and editing files, such as character string searching, replacement, and insertion.

Given that `n' is between 0 and 65535, the following ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file:

nA <CR> Add n lines to the end of memory buffer

nW Write the first n lines on the memory buffer to the temporary file free space. Move the remaining lines to the top of the memory buffer.

E End the edit. Copy all buffered text to temporary file, and copy all unprocessed source lines to the temporary file. Rename files as previously described.

H End edit, close and reopen files. (Same as issuing an E command, followed by a reinvocation of ED, using filename.ext as the file to edit.)
Return to original file. The memory buffer is emptied, the temporary file is deleted, all previous editing to the file is wiped out, and the source pointer returns to position 1 of the source file.

Quit edit with NO file alterations.

If 'n' is omitted in any ED command where an integer is allowed, then 1 is assumed. In addition, if a (#) is given in the place of 'n', then the integer 65,535 is assumed (the largest value for n that is allowed). Since most reasonably sized source files can be contained entirely in the memory buffer, the command #A is often issued at the beginning of the edit to read the entire source file to memory. Similarly, the command #W writes the entire buffer to the temporary file. Two special forms of the A and W commands are provided as a convenience. The command 0A fills the current memory buffer to at least half-full, while 0W writes lines until the buffer is at least half-empty. It should also be noted that an error is issued if the memory buffer size is exceeded. The operator may then enter any command (such as W) which does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

4.3.2.8.1 MEMORY BUFFER ORGANIZATION

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an associated (imaginary) character pointer CP which moves throughout the memory buffer under command of the operator. The CP is always located AHEAD of the first character of the first line, BEHIND the last character of the last line, or BETWEEN two characters. The current line (CL) is the source line which contains the CP.
4.3.2.8.2 MEMORY BUFFER OPERATION

Upon initiation of ED, the memory buffer is empty. The operator may either append lines (A command) from the source file, or enter the lines directly from the console with the insert command

A>I <CR>

ED then accepts any number of input lines, where each line terminates with a <CR>, until a CTRL-Z ('Z) is typed by the operator. The sequence

A>I <CR>
NOW IS THE <CR>
TIME FOR <CR>
ALL GOOD MEN <CR>
^Z

leaves the memory buffer as shown below

NOW IS THE <CR> <LF>
TIME FOR <CR> <LF>
ALL GOOD MEN <CR> <LF>

Various commands can then be issued which manipulate the imaginary "character pointer" within the memory buffer. The commands shown below with a preceding \( n \) indicate that an optional unsigned value can be specified. When preceded by +/-, the command can be unsigned, or have an optional preceding plus or minus sign. As before, the pound sign (#) is replaced by 65,535. If an integer \( n \) is optional, but not supplied, then \( n=1 \) is assumed. Finally, if a plus sign is optional, but none specified, then + is assumed.

\[
\begin{align*}
+/- \ B & \quad \text{Move CP (character pointer) to beginning of memory buffer if +, and to bottom if -}. \\
+/- \ nC & \quad \text{Move CP by +/- n characters (toward front of buffer if +), counting the <CR> <LF> as 2 distinct characters}. \\
+/- \ nD & \quad \text{Delete n characters ahead of CP if +, and behind CP if minus}. \\
+/- \ nK & \quad \text{Kill (remove) +/- n lines of source text using CP as the current reference. If CP is not at the beginning of the current line when K is issued, then the characters BEFORE CP remain if + is specified, while the characters AFTER CP remain if - is given.}
\end{align*}
\]
$+/\ nL$

If $n=0$ then move CP to the beginning of the current line. If $n$ is not equal to 0, then first move the CP to the beginning of the current line, and then move it to the beginning of the line which is $n$ lines down (if $+$) or up (if $-$). The CP will stop at the top or bottom of the memory buffer if too large a value of $n$ is specified.

$+/\ nT$

If $n=0$ then type the contents of the current line up to CP. If $n=1$ then type the contents of the current line from CP to the end of the line. If $n>1$ then type the current line along with $n-1$ lines which follow, if $+$ is specified. Similarly, if $n>1$ and $-$ is given, type the previous $n$ lines, up to the CP. The BREAK key can be depressed to abort long type-outs.

$+/\ n$

Equivalent to $+/\ nLT$, which moves up or down and types a single line.

4.3.2.8.3 COMMAND STRINGS

Any number of commands can be typed contiguously (up to the capacity of the CP/M console buffer), and are executed only after the $<\text{CR}>$ is typed. Thus, the operator may use the CP/M console command functions to manipulate the input command:

- **Delete** Remove the last character
- **CONTROL-U** Delete the entire line
- **CONTROL-C** Re-initialize the CP/M system
- **CONTROL-E** Return carriage for long lines without transmitting buffer (max 128 characters)

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. The command strings shown below produce the results to the right:
<table>
<thead>
<tr>
<th>Command String</th>
<th>Effect</th>
<th>Resulting Memory Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. 5C0T &lt;CR&gt;</td>
<td>Move CP 5 characters and type the beginning of the line. &quot;NOW I&quot;</td>
<td>NOW I (CP) S THE</td>
</tr>
<tr>
<td>3. 2L-T &lt;CR&gt;</td>
<td>move two lines down and type previous line &quot;TIME FOR&quot;</td>
<td>NOW IS THE &lt;CR&gt; &lt;LF&gt; TIME FOR &lt;CR&gt; &lt;LF&gt; (CP) ALL GOOD MEN &lt;CR&gt; &lt;LF&gt;</td>
</tr>
<tr>
<td>4. -L#K &lt;CR&gt;</td>
<td>Move up one line, delete 65535 lines which follow</td>
<td>NOW IS THE &lt;CR&gt; &lt;LF&gt;</td>
</tr>
<tr>
<td>5. I &lt;CR&gt;</td>
<td>Insert two lines of text</td>
<td>NOW IS THE &lt;CR&gt; &lt;LF&gt; TIME TO &lt;CR&gt; &lt;LF&gt; INSERT &lt;CR&gt; &lt;LF&gt;</td>
</tr>
<tr>
<td>TIME TO &lt;CR&gt;</td>
<td>INSERT &lt;CR&gt;</td>
<td></td>
</tr>
<tr>
<td>CTRL-Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. &lt;CR&gt;</td>
<td>Move down one line and type one line &quot;INSERT&quot;</td>
<td>NOW IS THE &lt;CR&gt; &lt;LF&gt; TIME TO &lt;CR&gt; &lt;LF&gt; (CP) INSERT &lt;CR&gt; &lt;LF&gt;</td>
</tr>
</tbody>
</table>
4.3.2.8.4 TEXT SEARCH AND ALTERATION

ED also has a command which locates strings within the memory buffer. The command takes the form

nF clc2...ck <CR> or CTRL-Z

where cl through ck represent the characters to match followed by either a <CR> or CTRL-Z. ED starts at the current position of CP and attempts to match all k characters. The match is attempted n times, and if successful, the CP is moved directly after the character ck. If the n matches are NOT successful, the CP is NOT moved from its initial position. Search strings can include CTRL-L, which it replaces by the pair of symbols <CR> <LF>.

The following commands illustrate the use of the F command:

<table>
<thead>
<tr>
<th>Command String</th>
<th>Effect</th>
<th>Resulting Memory Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. B#T &lt;CR&gt;</td>
<td>move to beginning and type entire buffer</td>
<td>(CP) NOW IS THE TIME FOR ALL GOOD MEN</td>
</tr>
<tr>
<td>2. FS T &lt;CR&gt;</td>
<td>find the end of the string &quot;S T&quot;</td>
<td>NOW IS T (CP) HE</td>
</tr>
</tbody>
</table>

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is:

I clc2...cn CTRL-Z
or
I clc2...cn <CR>
where cl through cn are characters to insert. If the insertion string is terminated by a CTRL-Z, the characters Cl through cn are inserted directly following the CP, and the CP is moved directly after character cn. The action is the same if the command is followed by a <CR> except that a <CR> <LF> is automatically inserted into the text following character cn. Consider the following command sequences as examples of the F and I commands:

<table>
<thead>
<tr>
<th>Command String</th>
<th>Effect</th>
<th>Resulting Memory Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTHIS IS `z &lt;CR&gt;</td>
<td>Insert &quot;THIS IS &quot; at the beginning of the text</td>
<td>THIS IS (CP) NOW THE TIME FOR ALL GOOD MEN</td>
</tr>
<tr>
<td>FTIME<code>z-4DIPLACE</code>z &lt;CR&gt;</td>
<td>find &quot;TIME&quot; and delete it; then insert &quot;PLACE&quot;</td>
<td>THIS IS NOW THE PLACE (CP) FOR ALL GOOD MEN</td>
</tr>
<tr>
<td>3F0<code>z-3D5DICHANGES</code>z &lt;CR&gt;</td>
<td>find third occurrence of &quot;O&quot; (ie, the second &quot;O&quot; in GOOD), delete previous 3 characters; then insert &quot;CHANGES&quot;</td>
<td>THIS IS NOW THE PLACE FOR ALL CHANGES (CP)</td>
</tr>
</tbody>
</table>

ED also provides a single command which combines the F and I commands to perform simple string substitutions. The command takes the form

```
n s clc2....ck `z dld2....dm <CR> or CTRL-Z
```

and has exactly the same effect as applying the command string

```
F clc2....ck `z-kDIdld2....dm <CR> or CTRL-Z
```

a total of n times. That is, ED searches the memory buffer starting at the current position of CP and successively substitutes the second string for the first string until the end of buffer, or until the substitution has been performed n times.
As a convenience, a command similar to `F` is provided by ED which automatically appends and writes lines as the search proceeds. The form is

```
n N c1c2...ck <CR> or CTRL-Z
```

which searches the entire source file for the nth occurrence of the string `c1c2...ck` (recall that `F` fails if the string cannot be found in the current buffer). The operation of the `N` command is precisely the same as `F` except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory contents is written (i.e., an automatic `#W` is issued). Input lines are then read until the buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until the string has been found n times, or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command, takes the form

```
n J c1c2...ck ^z d1d2...dm ^z e1e2...eq <CR> or ^z
```

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string `c1c2...ck`. If found, insert the string `d1d2...dm`, and move CP to follow `dm`. Then delete all characters following CP up to (but not including) the string `e1e2...eq`, leaving CP directly after `dm`. If `e1e2...eq` cannot be found, then no deletion is made. If the current line is

```
(CP) NOW IS THE TIME <CR> <LF>
```

Then the command

```
JW ^zWHAT^z^L <CR>
```

Results in

```
NOW WHAT (CP) <CR> <LF>
```

(Recall that `^L` represents the pair `<CR> <LF>` in search and substitute strings).

It should be noted that the number of characters allowed by ED in the `F`, `S`, `N`, and `J` commands is limited to 100 symbols.
4.3.2.8.5 SOURCE LIBRARIES

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is:

```
R flf2...fn ^z
or
R flf2...fn <CR>
```

where flf2...fn is the name of a source file on the disk with an assumed filetype of 'LIB'. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command

```
RMACRO <CR>
```

is issued by the operator, ED reads from the file MACRO.LIB until the end-of-file, and automatically inserts the characters into the memory buffer.

4.3.2.8.6 REPETITIVE COMMAND EXECUTION

The macro command M allows the ED user to group ED commands together for repeated evaluation. The M command takes the form:

```
n M clc2...ck <CR> or ^z
```

where clc2...ck represent a string of ED commands, not including another M command. ED executes the command string n times if n>1. If n=0 or 1, the command string is executed repetitively until an error condition is encountered (e.g., the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line which is changed:

```
MFGAMMA^z-5DIDELTA^z0TT <CR>
```

or equivalently

```
MSGAMMA^zDELTA^z0TT <CR>
```
4.3.2.8.7 ED ERROR CONDITIONS

On error conditions, ED prints the last character read before the error, along with an error indicator:

? unrecognized command

> memory buffer full (use one of the commands D,K,N,S, or W to remove characters), F,N, or S strings too long.

# cannot apply command the number of times specified (e.g., in F command)

0 cannot open LIB file in R command

Cyclic redundancy check (CRC) information is written with each output record under CP/M in order to detect errors on subsequent read operations. If a CRC error is detected, CP/M will type

PERM ERR DISK d

where d is the currently selected drive. The operator can choose to ignore the error by typing any character at the console (in this case, the memory buffer data should be examined to see if it was incorrectly read), or the user can reset the system and reclaim the backup file, if it exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information:

TYPE x.BAK <CR>

where x is the file being edited. Then remove the primary file:

ERA x.y <CR>

and rename the BAK file:

REN x.y=x.BAK <CR>

The file can then be re-edited, starting with the previous version.
## 4.3.2.8.8 CONTROL CHARACTERS AND COMMANDS

The following table summarizes the control characters and commands available in ED:

<table>
<thead>
<tr>
<th>Control Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^c</code></td>
<td>system reboot</td>
</tr>
<tr>
<td><code>^e</code></td>
<td>physical <code>&lt;CR&gt; &lt;LF&gt;</code> (not actually entered in command)</td>
</tr>
<tr>
<td><code>^i</code></td>
<td>logical tab (cols 1,8, 15,...)</td>
</tr>
<tr>
<td><code>^l</code></td>
<td>logical <code>&lt;CR&gt; &lt;LF&gt;</code> in search and substitute strings</td>
</tr>
<tr>
<td><code>^u</code></td>
<td>line delete</td>
</tr>
<tr>
<td><code>^z</code></td>
<td>string terminator</td>
</tr>
<tr>
<td>delete</td>
<td>character delete</td>
</tr>
<tr>
<td>break</td>
<td>discontinue command (e.g., stop typing)</td>
</tr>
<tr>
<td>Command</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>nA</td>
<td>append lines</td>
</tr>
<tr>
<td>+/- B</td>
<td>begin bottom of buffer</td>
</tr>
<tr>
<td>+/- nC</td>
<td>move character positions</td>
</tr>
<tr>
<td>+/- nD</td>
<td>delete characters</td>
</tr>
<tr>
<td>E</td>
<td>end edit and close files (normal end)</td>
</tr>
<tr>
<td>nF</td>
<td>find string</td>
</tr>
<tr>
<td>H</td>
<td>end edit, close and reopen files</td>
</tr>
<tr>
<td>I</td>
<td>insert characters</td>
</tr>
<tr>
<td>nJ</td>
<td>place strings in juxtaposition</td>
</tr>
<tr>
<td>+/- nK</td>
<td>kill lines</td>
</tr>
<tr>
<td>+/- nL</td>
<td>move down/up lines</td>
</tr>
<tr>
<td>nM</td>
<td>macro definition</td>
</tr>
<tr>
<td>nN</td>
<td>find next occurrence with autoscan</td>
</tr>
<tr>
<td>O</td>
<td>return to original file</td>
</tr>
<tr>
<td>+/- nP</td>
<td>move and print pages</td>
</tr>
<tr>
<td>Q</td>
<td>quit with no file changes</td>
</tr>
<tr>
<td>R</td>
<td>read library file</td>
</tr>
<tr>
<td>nS</td>
<td>substitute strings</td>
</tr>
<tr>
<td>+/- nT</td>
<td>type lines</td>
</tr>
<tr>
<td>+/- u</td>
<td>translate lower to upper case if U, no translation if -U</td>
</tr>
<tr>
<td>nW</td>
<td>write lines</td>
</tr>
<tr>
<td>nZ</td>
<td>sleep</td>
</tr>
<tr>
<td>+/- n &lt;CR&gt;</td>
<td>move and type (+/- nLT)</td>
</tr>
</tbody>
</table>
The ED context editor contains a number of commands which enhance its usefulness in text editing. The improvements are found in the addition of line number, free space interrogation, and improved error reporting.

The context editor produces absolute line number prefixes when the "V" (Verify Line Numbers) command is issued. Following the V command, the line number is displayed ahead of each line in the format:

```
nnnnn:
```

where nnnnn is an absolute line number in the range of 1 to 65535. If the memory buffer is empty, or if the current line is at the end of the memory buffer, then nnnnn appears as 5 blanks.

The user may reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. Thus, the command

```
345:T
```

is interpreted as "move to absolute line 345, and type the line." Note that absolute line numbers are produced only during the editing process, and are not recorded with the file. In particular, the line numbers will change following a deleted or expanded section of text.

The user may also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute line number with a colon. Thus, the command

```
:400T
```

is interpreted as "type from the current line number through absolute line 400." Combining the two line reference forms, the command

```
345::400T
```

for example, is interpreted as "move to absolute line 345, then type through absolute line 400." Note that absolute line references of this sort can precede any of the standard ED commands.

A special case of the V command, "0V", prints the memory buffer statistics in the form:

```
free/total
```

where "free" is the number of free bytes in the memory buffer (in decimal), and "total" is the size of the memory buffer.
ED also includes a "block move" facility implemented through the "X" (Xfer) command. The form

\[ nX \]

transfers the next \( n \) lines from the current line to a temporary file called

\[ X$$$$$$\.LIB \]

which is active only during the editing process. In general, the user can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred lines accumulate one after another in this file, and can be retrieved by simply typing:

\[ R \]

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer, although a \( K \) command can be used directly after the \( X \), and the \( R \) command does not empty the transferred line file. That is, given that a set of lines has been transferred with the \( X \) command, they can be re-read any number of times back into the source file. The command

\[ 0X \]

is provided, however, to empty the transferred line file.

Note that upon normal completion of the ED program through \( Q \) or \( E \), the temporary LIB file is removed. If ED is aborted through CTRL-C, the LIB file will exist if lines have been transferred, but will generally be empty (a subsequent ED invocation will erase the temporary file).

Due to common typographical errors, ED 1.4 requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The commands

\[ E \text{ (end), H \ (head), O \ (original), Q \ (quit)} \]

must be typed as single letter commands.

ED 1.4 also prints error messages in the form

\[ \text{BREAK "x" AT c} \]

where \( x \) is the error character, and \( c \) is the command where the error occurred.
4.3.2.9 SGEN - COPY CP/M OPERATING SYSTEM

The SGEN program is used to copy the operating system tracks onto a diskette just as the copy command allows.

If you are not copying from a disk that already contains the system track, the SGEN program can obtain the system track information directly from a CP/M file. (See command form 2 below.)

The SGEN program writes information onto the reserved system tracks only; thus, it will not alter or destroy the disk directory or data already stored on the disk in any way.

The form of the command is:

A>SGEN  <CR>

or

A>SGEN filename.type  <CR>

In the first form, SGEN will prompt for both a source disk name (which must be a "bootable" system disk) and a destination disk name to copy the operating system to. The second form allows a "system image file" to serve as the source, requiring only a destination disk name. In either case, SGEN will write and verify the system information on the destination disk, reporting errors if they occur.

Note that SGEN is transportable to any CP/M v2.2 system and disk format, and is also useful for copying emulation disk system tracks if desired.
4.3.3.0 PATCH

From time to time programs must be corrected or patched to correct errors or deficiencies in the original code. The program named "PATCH.COM" has been supplied by Telecon to facilitate the easy installation of useful system modifications or required system updates. These patches change instructions on the CP/M v2.2 operating system (within the reserved system tracks of disk A), not in any of the named files or other stored information. It is suggested that you do not apply these patches to your original operating system disk, only to your copies. This PATCH is not reversible once installed. To apply the patches to the system, perform the following steps:

A> PATCH <CR>

This loads the "PATCH.COM" program and then starts execution automatically. There are three areas to be patched but only one of which is mandatory. Answer all questions and then make them final as in the following example:

** MANDATORY SYSTEM PATCH **

This patch corrects a problem in the BDOS associated with deblocking. If this patch is not installed, you are likely to encounter write errors. These errors may not be immediately apparent, but may corrupt disk data.

Do you wish to include this patch (Y/N)? Y
Optional CCP patch (for CRT type terminals)

This patch makes the DELETE (RUBOUT on some terminals) behave like the BACKSPACE key. That is, when a character is deleted, the system prints a BACKSPACE-SPACE-BACKSPACE combination to erase the character from the screen. That is the sole effect of this patch.

Do you wish to include this patch (Y/N)? Y
Optional CCP patch (for multi-disk systems)

Normally when a program (.COM file) is executed, the drive name must be specified or the system will look on the currently logged disk drive. This patch causes the CCP to search drive "A:" - if it fails to find the program on the currently logged disk drive. Explicit drive specifications will over-ride the effect of this patch.

Do you wish to include this patch (Y/N)? Y
Do you wish to make these patches final by writing them to disk (Y/N)? Y

Please wait writing to disk

It should be noted that this program may take as long as 15 to 20 seconds to execute. Do not interrupt the execution of this program, as results cannot be guaranteed. It should also be noted that this patch is transportable to any system using CP/M 2.2, using any of the 280 processor family.
4.3.3.1 SETUP

----------

SETUP is a program which allows the ZORBA user to select and save the operational characteristics of the computer which suit his needs. ZORBA is shipped with a default SETUP configuration installed which is remembered even with the power turned off. You may, however, use SETUP to program new characteristics which may be either temporarily installed or replace the default characteristics.

No ZORBA function keys have any programming as shipped. A sample function key file, named SYSTEM.KEY is on the system disk. To install this function key definition, until power is turned off, type the following:

A> SETUP SYSTEM <CR>

To run SETUP in order to change a function key or operational characteristic, enter the following sequence and select the change operation you desire from the main menu.

A> SETUP <CR>

The main menu allows you to select from any of the following functions:

SETUP v1.0

<table>
<thead>
<tr>
<th></th>
<th>PORT A (Primary)</th>
<th>PORT B (LST:)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAUD Rate</td>
<td>9600</td>
<td>9600</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Word width</td>
<td>8 bits</td>
<td>8 bits</td>
</tr>
<tr>
<td>Stop bits</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Protocol</td>
<td>None</td>
<td>XON/XOFF</td>
</tr>
<tr>
<td>Translation</td>
<td>None</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Command ""

Please select one of the following...

A - SET parameters for PORT A (Primary)
B - SET parameters for PORT B (LST:)
C - Specify COMMAND to be executed at cold boot time
K - DEFINE function KEY(S)
R - READ a function KEY FILE
S - SAVE FUNCTION KEYS to a file
X - EXIT (and save parameters to disk A:)
^C - ABORT (Exit without saving parameters)

Selection:

The above menu will be referred to as the main menu.
The main menu has three basic areas:

1. Current port configurations for Ports A & B
2. The current command to be executed on cold boot
3. A menu of available functions

Each area of the menu and each function shall be described below.

4.3.3.1.1 Current Port Configuration

<table>
<thead>
<tr>
<th>SETUP v1.0</th>
<th>--- Current Settings ---</th>
<th>PORT A (Primary)</th>
<th>PORT B (LST:)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAUD Rate</td>
<td>9600</td>
<td>9600</td>
<td></td>
</tr>
<tr>
<td>Parity</td>
<td>Odd</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Word Width</td>
<td>8 bits</td>
<td>8 bits</td>
<td></td>
</tr>
<tr>
<td>Stop bits</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td>None</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>Translation</td>
<td>None</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

This area contains information concerning the current settings for the two ports on the back of the system, the communications port and the printer port.

4.3.3.1.2. Cold Boot Command

Command ""

This is the command that is immediately executed after CP/M has been loaded and control has been transferred to the CCP.

This command WILL NOT appear on the screen, but its results will; if any is generated.
4.3.3.1.3 MENU OF AVAILABLE FUNCTIONS
-----------------------------------------------

Please select one of the following...

A - SET parameters for PORT A (Primary)
B - SET parameters for PORT B (LST:)
C - Specify COMMAND to be executed at cold boot time
K - DEFINE function KEY(S)
R - READ a function KEY FILE
S - SAVE FUNCTION KEYS to a file
X - EXIT (and save parameters to disk A:)
^C - ABORT (Exit without saving parameters)

Selection :

The above menu will allow your system to be configured to
perform any variety of combinations via the keyboard, thus,
eliminatating having to open the system and change DIP switch
settings.

To change any of the displayed parameters select one item
from the menu of functions and type the corresponding letter.
The first example will be that of changing the cold boot
command.

Selection : C

The system will now prompt

COMMAND : DIR

The user may now type the cold-boot command of his choice,
followed by a <CR>. Example:

COMMAND: DIR <CR>

The above command will display the directory of the disk that
was just booted. This command could have been a submit file
which could have performed any number of commands that would
have been loaded in sequence as they appear in the submit or
command file.

Upon completion of the requested function, the main menu will
be displayed with the new data being reflected. To save the
changes just made, select option "X" and the SETUP program will
return to the operating system after the data has been written
onto the disk.
4.3.3.1.4 PORT CONFIGURATION

It should be noted that both ports are configured using nearly the same menus. If there is an area that differs, it will be noted. The following examples will reflect changing the configuration of Port A but could just as well have been Port B.

From the main menu select option "A". The SETUP program will display a menu listing the areas available for change:

Please select one of the following...

A - Parity
B - Baud rate
D - Set this port with DEFAULT settings
P - Protocol
S - Stop bits
T - Translation table
W - Word width

(ESCAPE to cancel command)

Selection:

This menu will be referred to as the PORT menu in continuing text.
4.3.3.1.5 PORT PARITY

Taking each item in order, the Parity of Port A will be changed.

Selection : A <CR>

The SETUP program will now display the available options for parity:

N - No Parity
O - Odd Parity
E - Even Parity

(ESCAPE to cancel command)

Selection :

For this example take

Selection : E <CR>

The main menu will now reflect that the parity for Port A is now even.

NOTE that the ESCAPE key may be pressed from any menu level to exit that menu with NO change made.
4.3.3.1.6 BAUD RATE

The baud rate for the port is changed by selecting option "B" from the PORT menu.

Selection: B <CR>

The baud rate menu is now displayed with alphabetic selection codes listed to the left of each item.

Please select one of the following...

<table>
<thead>
<tr>
<th>Code</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>45.45</td>
</tr>
<tr>
<td>B</td>
<td>50</td>
</tr>
<tr>
<td>C</td>
<td>62.5</td>
</tr>
<tr>
<td>D</td>
<td>75</td>
</tr>
<tr>
<td>E</td>
<td>110</td>
</tr>
<tr>
<td>F</td>
<td>134.5</td>
</tr>
<tr>
<td>G</td>
<td>150</td>
</tr>
<tr>
<td>H</td>
<td>200</td>
</tr>
<tr>
<td>I</td>
<td>300</td>
</tr>
<tr>
<td>J</td>
<td>600</td>
</tr>
<tr>
<td>K</td>
<td>1200</td>
</tr>
<tr>
<td>L</td>
<td>1760</td>
</tr>
<tr>
<td>M</td>
<td>1800</td>
</tr>
<tr>
<td>N</td>
<td>2000</td>
</tr>
<tr>
<td>O</td>
<td>2400</td>
</tr>
<tr>
<td>P</td>
<td>3520</td>
</tr>
<tr>
<td>Q</td>
<td>3600</td>
</tr>
<tr>
<td>R</td>
<td>4800</td>
</tr>
<tr>
<td>S</td>
<td>7200</td>
</tr>
<tr>
<td>T</td>
<td>9600</td>
</tr>
<tr>
<td>U</td>
<td>19200</td>
</tr>
</tbody>
</table>

(ESCAPE to cancel command)

Determine the desired Baud Rate and enter that code.

Selection: K <CR> (1200 Baud)

The main menu is now displayed with the new baud rate being reflected.
4.3.3.1.7 PROTOCOL

The line protocol for the communications port (port A) is different from that of the printer port (port B). For port A the selection will be:

Selection : A <CR>

from the main menu, and then the port menu is displayed; with the port menu select option "P".

Selection : P <CR>

which will result in the protocol menu being displayed for the communications port.

Please select one of the following...

C - Centronics Parallel
N - None
X - XON/XOFF
E - ETX/ACK

(ESCAPE to cancel command)

Selection :

After selecting the required protocol, the main menu will be displayed reflecting the new protocol.

For the communications port (Port A), the Centronics Protocol is not allowed.

4.3.3.1.8 STOP BITS

From the PORT menu the number of stop bits appended to each byte sent may be changed:

Selection : S <CR>

The stop bit menu will be displayed if option "S" is selected. This will result in the following menu:

Please select one of the following...

1 - 1 stop bit
5 - 1.5 stop bits
2 - 2 stop bits

(ESCAPE to cancel command)

Selection :

After selecting the required protocol, the main menu will be displayed reflecting the new protocol.
4.3.3.1.9 TRANSLATION TABLE

From the PORT menu the selection of option "T" will result in the available translation tables to be displayed.

Selection: T <CR>

Please select one of the following...

B - Baudot
C - IBM Correspondence
E - EBCDIC
N - No Translation (ASCII)

(ESCape to cancel command)

Selection:

Translation is not allowed for Port B.

Translation from ASCII (CP/M native code) to any other character code set is specified by answering B, C, or E to this menu. The translation is installed in the system area and is permanent until deselected using SETUP. Send and receive translated data through the communications port using PIP to PUN: or from RDR.
4.3.3.2.0 WORD WIDTH

The final area available for change is that of the number of bits within a character.

Selection: W <CR>

This will result in the following menu:

5 - Five bits
6 - Six bits
7 - Seven bits
8 - Eight bits

(ESCAPE to cancel command)

Selection:

As with all of the other areas, the changes made are immediately reflected on the appropriate menu. If the changes are to be permanent (until SETUP is run again) terminate this function with:

Selection: X <CR>

This will write the configuration to the system tracks and when the system is cold booted, these changes will automatically be remembered. Any changes made during SETUP execution are effective immediately upon selection, regardless of the method of exit. A "cold-boot" will restore the system to the last written configuration (using SETUP X) if SETUP was exited using "C", not saving to disk.

If a SETUP-configured system is copied to another disk using SGEN, the configuration, including the programmable keys, will also be copied.
4.3.2.1 PROGRAMMABLE FUNCTION KEYS

With the main menu being displayed, there are three options available pertaining to programmable function keys.

K - Define function key(s)  
R - Read function key file  
S - Save function keys to a file

4.3.2.1.1 DEFINE FUNCTION KEY

Option K will result in a display of the function keys as they are presently programmed. The keys are displayed as four groups.

Type "D" to examine any of the four key groups as shown below:

Group 1 (Keys F1 - F19)

Selection: (NON-CTRL) FUNCTION KEY DEFINITIONS  
(184 characters unused)

1:  
2:  
3:  
4:  
5:  
6:  
7:  
8: text for any keys programmed will appear here  
9:  
10:  
11:  
12:  
13:  
14:  
15:  
16:  
17:  
18:  
19:  

Options: X - exit  
D - display next key group  
Or press the FUNCTION KEY you wish to define.
Group 2 (CTRL key plus Fl - F19)

(CONTROL) FUNCTION KEY DEFINITIONS

1:
2:
3:
4:
5:
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:

Options: X - exit     D - display next key group
Or press the FUNCTION KEY you wish to define.

Group 3 (CTRL KEY plus numeric key pad)

(NUMERIC) FUNCTION KEY DEFINITIONS

9: 9
8: 8
7: 7
6: 6
5: 5
4: 4
3: 3
2: 2
1: 1
0: 0
.: .
NL: <LF>
-: -

Options: X - exit     D - display next key group
Or press the FUNCTION KEY you wish to define.
Group 4 (ARROW KEYS)

----

(ARROWS) FUNCTION KEY DEFINITIONS

UP:    ^A
DOWN:  ^B
LEFT:  ^C
RIGHT: ^D

Options: X - exit  D - display next key group
Or press the FUNCTION KEY you wish to define.

To define a function key, answer to the following menu:

Options: X - exit  D - display next key group
Or press the FUNCTION KEY you wish to define.

DEPRESS F1.

This signifies that key F1 is being programmed, and the following will be displayed:

Type NEW definition string (up to 80 characters).
END your entry by pressing the FUNCTION KEY again.

user (NON-CTRL) FUNCTION KEY DEFINITIONS

.types--> ?THIS IS A TEST <CR>  (169 characters unused)

It must be noted that when defining a key, all keystrokes are valid and that if any errors are typed, there is no way of correcting them. The key must be reprogrammed.

This is displayed on the menu in the form:

1:  THIS IS A TEST <CR>
2:
3:
4:
5:
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:

Options X - exit  D - display next key group
Or press the FUNCTION KEY you wish to define.
The maximum number of keystrokes allowed is indicated and will range from a maximum of 80 characters to a minimum of one character. In the upper right-hand corner of each group display is the number of characters available for the balance of keys not yet programmed, or available for reprogramming. To have key F1 display the directory for the currently logged disk plus the STAT *.* command to display how much storage is available, the programming would be:

```
DIR <CR> STAT *.* <CR>
```

This string would be terminated by depressing Key F1 again. This would result in the Group 1 keys being displayed and also reflect the string just typed being listed adjacent to 1:

After all keys that are to be programmed have been programmed, option "X" should be selected.

```
Selection: X <CR>
```

This option will save all programmed keys on the system track. When the system responds with "A>" signifying that you are back at the command level, depress F1 and the results should be:

```
A> (key F1 depressed)
A> DIR
Directory of disk 'A'
A> STAT *.*
The status and size of each file followed by the number of free bytes available.
A>
```

All keys that perform a non-printable function will be displayed similar to the way the carriage return was in the previous example. These keys will still perform their normal function when used within a programmed key.
4.3.2.1.2 SAVE FUNCTION KEYS TO A FILE  
----------------------------------------

Once the programmable function keys have been programmed they may be saved as a file. This is option "S" from the main menu.

Selection : S <CR>

The SETUP program will respond with:

   ENTER KEY FILE NAME?  FUNCTION <CR>

The file extension is assumed to be ".KEY" and does not have to be specified.

If at any time in the SETUP program you wish to abort or not apply any of the changes to either the function keys or the ports, type a CTRL-C for selection while at the main menu, and the SETUP program will exit to the command level.

4.3.2.1.3 READ FUNCTION KEY FILE  
-----------------------------------

If the function keys have been previously programmed and saved as a file, they may be reloaded at any time using option "R" from the main menu.

Selection : R <CR>

The SETUP program will respond with:

   ENTER NAME OF KEY FILE : SYSTEM <CR>

The SETUP program will append the file extension of ".KEY" to the primary file name, so the extension does not have to be specified. The file specified is then read and the keys are then programmed as required.

An alternate method of programming the keys from a file is initiated from the command level:

   A> SETUP SYSTEM <CR>

This will have the same results as the menu driven programming, except that the keys are NOT saved to disk and will be forgotten at the next cold boot.
4.3.3.2 SUBMIT - BATCH PROGRAM COMMANDS FILE

This function allows responses and/or requests to be entered into a file and then control passed to the file for execution of a program or group of programs. If any of the programs require responses, they may be included into the file at the appropriate time. This file must be created using the ED program and must have a secondary name of .SUB. The SUBMIT program or function also allows for variables to be replaced within the file. The file variables must be in the form $1 $2 etc. The command lines below show a) the command without variables and then b) with variables:

A> SUBMIT FILENAME.SUB <CR>

or

A> SUBMIT FILENAME.SUB x y z <CR>

Where

x,y,z are variables for the inclusion to the file.
4.3.3.3 MOVCPM - RECONFIGURE CP/M
--------------------------------------

This function allows the user to reconfigure CP/M to operate in an environment other than what has been SGENed. The format of the command line is:

A>MOVCPM x y <CR>

The above format has two variables included, they are:

x - system size (16 to 64K bytes) or * for all available memory, and

y - if system is to be left in memory for following SGEN.

If x is omitted or an * is entered then the maximum size is used. Otherwise, the system will be configured to the size input.

If y is omitted the system is executed but not permanently recorded. If y is an '*' the system is left in memory ready for SGEN.

Upon completion the message is displayed:

"READY FOR SGEN OR
SAVE 34 CPMMxx.COM"

Where xx is the size of the system.

In the ZORBA, MOVCPM is used only during BIOS regeneration from source. The procedure for this is contained in the files GEN.SUB and GEN2.SUB supplied on your "source disk".

4.3.3.4 FORMAT - Prepare New Disk for Writing
-----------------------------------------------

Prior to using a new disk for the first time it must be formatted. A new disk is blank; like a record without grooves. Formatting defines the tracks and sectors, and prepares the disk to be written on. Formatting a used disk will erase the data on the disk and prepare it for new data. Formatting will also inform the user of the fact that there are bad sectors (if any) and which sectors are bad. Another purpose of formatting is that it will initialize the directory segments on the disk. This is required, as the operating system will interrogate the directory prior to performing a read or write function.

To format a disk, perform the following steps:

* Load the CP/M operating system by depressing the RETURN key. The system will respond with:

58K CP/M version 2.2

A>
The above means that CP/M has been loaded and is waiting for a command. Respond with:

A>FORMAT <CR>

The system will now load the Format Program from the disk and transfer control to it. This program will respond:

FORMAT v2.4 for TELCON CP/M
COPYRIGHT (C) 1982, TELCON INDUSTRIES, INC.

Select  Disk Format
-------  ------------
A  -  Telcon DD
B  -  Telcon QD
C  -  Xerox 820-2 DD
D  -  Xerox 820 SD or Cromemco 520 SD
E  -  KayComp II DD
F  -  Osborne SD
G  -  CP/M-86 SD for IBM-PC
H  -  DEC VT-18x series DD
X  -  EXIT TO CP/M, cancel this program.

Please select one letter:

FUNCTION: A (user selects one letter)

A - FORMAT without verification
B - FORMAT AND VERIFY afterward.
C - VERIFY ONLY, do not write on the disk.
X - EXIT TO CP/M, cancel this program.

Select one letter: B (second selection)

INSERT DISK TO BE FORMATTED AND VERIFIED IN DRIVE B (1), TYPE:

G  When ready to GO,
R  To specify a RANGE of tracks first,
CTRL-C  To cancel.

Please select one letter : G (third selection, following insertion of disk into B:)

At this time the system will begin to format the disk and display any errors. If there are errors detected, the format program should be rerun. If the errors persist then the disk should not be used, as possible errors could destroy your data. Try another blank disk to be sure the problem is not in your disk drive.
It is suggested that you format all of your supplemental disks at this time so you will not have to worry whether or not they have been formatted.

Some of the options available during the execution of the "FORMAT" program are: the selection of the type of format to be used (which is self-explanatory); how the program will run it; (format only; format and verify; or verify only).

The final option is whether to perform the requested function on all or part of the disk. If when responding to the third prompt, option "R" is specified, the following prompt will be given:

Enter LOW (beginning) track number : 0  <CR>
Enter HIGH (ending) track number : 10  <CR>

Please wait.

xx WRITING TRACK xx

where xx updates as the disk is formatting.

Once all prompts have been answered and the program is in control, the status will be displayed as to which track is being written on or read.

An important fact should be noted, that once the program starts to format, or format and verify, it should not be stopped; as the state of the disk cannot be determined. If this must be done, however, use CTRL-C to abort the operation.
5.0 PRINTING

If your system is equipped with a printer, the following commands may be applied.

5.1 COPYING THE SCREEN TO THE PRINTER

To obtain a printed copy of the information currently on the screen, depress the "PRINT" key in conjunction with the SHIFT key. The system must be receiving a clear-to-send signal from the printer, or the system will wait until one is received. This could result in the system being hung up, as there is no way to interrupt the printing. Upon completion, you may continue from the point you were at prior to the print request. The screen contents are not disturbed.

If the system becomes hung up waiting for the printer, the only way to restore the system to an operational mode is to press the RESET button, thus initiating a "cold boot".

5.2 PRINT ALL KEYSTROKES

To obtain a hard copy or printed list of all keystrokes and program responses, make sure the printer is on-line and available. Type a CTRL-P. This key acts as a toggle switch. The second time CTRL-P is depressed, the print function may be terminated.

Note that this function is cancelled by the system at "warm-boot" time; typically when a program exits to the CCP prompt A>.

5.3 PRINTING A FILE

To copy a file to the printer, perform the following:

    A> PIP LST: = file.ext <CR>

This will result in the file named file.ext to be copied to the LST: device.
6.0 COMMUNICATIONS PORT

Once the communications port has been configured as described using the SETUP program, data may be received from, or transmitted to the communications port.

6.1 RECEIVING DATA

To receive data from an external device via the communications port (Port A), the following must be done at the command level:

```
A> PIP file.ext:=RDR:  <CR>
```

The data being received must be terminated with a CTRL-Z. The data being received will be written to the file name specified on the command line "file.ext".

6.2 TRANSMITTING A FILE

To transmit a file from the disk to an external device, do the following:

```
A> PIP PUN:=file.ext  <CR>
```

This will take the contents of the file name on the command line and transmit that data over the communications port (port A). This file must also be terminated with or contain a CTRL-Z to terminate transmission.

6.3 TRANSMITTING FROM THE KEYBOARD

To transmit data from the keyboard to the communications port (port A) type the following:

```
A> PIP PUN:=CON:  <CR>
```

The data will then be transmitted to the external device until a CTRL-Z is typed.
7.0 TERMINAL EMULATION

As previously stated, the ZORBA portable computer is capable of emulating other types of terminals. These emulations will result in the ZORBA transmitting and acting upon received control sequences to operate the terminal.

7.1 HEATHKIT MODEL H19 / ZENITH Z19

The built-in H19 emulation software is designed to allow the user to communicate with a host computer and its associate applications programs that are structured to the use of the HEATHKIT H19 type terminals. Most of the important features of this type terminal are incorporated in this emulation.

7.1.0 CURSOR FUNCTIONS

7.1.0.1 Cursor Home - Esc H
move the cursor to the first position of the first line of the display.

7.1.0.2 Direct Cursor Positioning - Esc Y L# C#
This escape sequence allows the computer to position the cursor at any line column combination on the screen. The line and column references are represented by ASCII characters.

Since line 1, column 1 represents the first character on the screen and the first printable character in ASCII, the value of 20H (3210), the line and column numbers must be adjusted accordingly.

The following example will position the cursor to line 5, column 10:

The line number would be computed: 31 + 5 = 36 (base 10 or 24H) which is the ASCII character "$". The column number would be 31 + 10 = 41 (29H) which is the ASCII character "").

The escape sequence would then result in the following:

ESC Y $

})
The maximum value for a line number would be 31 + 25 = 56 (38H) = "8".

The maximum value for a column number would be 31 + 80 = 111 (6FH) = "a"

If a LINE number is specified that is greater than the maximum value, the cursor will remain on the current line. If a COLUMN number is specified greater than the maximum, the cursor will be positioned to the right-most column of the current line.

Note, 25th line support exists for this H19 emulation. Use direct cursor addressing to access the 25th line, per H19 specifications.
7.1.1 ERASING AND EDITING

7.1.1.1 Clear display - ESC E
This function will clear the screen and place the cursor at the home position.

7.1.1.2 Clear to end of screen - ESC J
Erase all information from current cursor position to the end of the screen.

7.1.1.3 Erase line - ESC l (lowercase L)
Erase current line including current cursor position.

7.1.1.4 Erase beginning of line - ESC o
Erase data from the beginning of the line thru the current cursor position.

7.1.1.5 Erase to end of line - ESC K
Erase data from the current cursor position to the end of line. This includes the current cursor position.

7.1.1.6 Insert line - ESC L
All data from the line the cursor is on is moved down one line and a blank line is inserted where the cursor was. The cursor is positioned to be beginning of the blank line.

7.1.1.7 Delete line - ESC M
The contents of the line where the cursor is positioned is deleted and the balance of the screen is moved up one line with a blank line being added for line 25. The cursor is positioned on the first character of the current line.

7.1.1.8 ESC F - enter graphics mode.

7.1.1.9 ESC J - exit.
8.0 AUXILLARY PROM ACCESS

The prom area overlays the first 16k of RAM when selected. This PROM area cannot be accessed while prom is selected, but is also not disturbed. To select/deselect prom area, execute either an in or out instruction to I/O device 4 (for select) or I/O device 5 (to deselect).

The most common usage for prom area is permanently resident routines. Example:

```
OUT (4), A ;select prom area (contents of A immaterial)
CALL ROUTINE ;execute code in prom
OUT (5), A ;deselect prom
```

```
-----------------------|       | FFFF
| RAM                 | 16K  |
-----------------------|       | C000
-----------------------|       | BFFF
| RAM                 | 16K  |
-----------------------|       | 8000
| PROM 3              | 3FFF  |
-----------------------|       | 7FFF
| PROM 2              | 2FFF  |
-----------------------|       | 4000
| PROM 1              | 1FFF  |
-----------------------|       | 3FFF
| PROM 0 *            | 0FFF  |
-----------------------|       | 0000
```

* Used by computer for cold boot/common routines
APPENDIX A

Command Summary

COMMAND

--------

MEANING

--------

ERASE COMMANDS

---------

ERASE COMMANDS

---------

ERASE file.ext

Erase filename file.ext from disk

ERASE file.*

Erase primary name file with any secondary name.

ERASE *.ext

Erase all files with secondary names of .ext

ERASE x?z.ext

Erase all files with primary names that start with x and end with z and with a single character in between with the secondary name of .ext

ERASE *.*

Erase all files

All above filenames may be preceded by drive name.

DIRECTORY COMMANDS

---------

DIRECTORY COMMANDS

---------

DIR

Display all filenames on logged disk.

DIR B:

Display all filenames on Drive B.

DIR file.ext

Display directory entry for file.ext.

DIR *.BAS

Display all files with secondary name of .BAS

DIR TEXT.*

Display all files with primary name of TEXT.

DIR X?Z.BAS

Display all files with primary name of X as the first character and Z as the last character with any single character in between with a secondary name of .BAS

DIR *.*

Display all directory entries.
RENAMES Command

```
RENAME COMMAND

RENAME NEWFILE.EXT=OLDFILE.EXT Rename OLDFILE.EXT to NEWFILE.EXT
RENAME A:X.BAS=B:Z.BAS Rename the file Z.BAS on Drive A to X.BAS also on Drive A
RENAME X:BAS=B:Z.BAS Rename the file Z.BAS on Drive B to X.BAS also on Drive B.
SAVE 4 X.COM Copy 100H through 4FFH to X.COM
SAVE 41 Z Copy 100H through 29FFH to Z

The drive name may prefix the file name in both examples.
```

TYPE Command

```
TYPE COMMAND

TYPE FILE.EXT Copy FILE.EXT to the CRT

Note: If tabs are present in the file, they are expanded to every 8 character positions.
```

STAT Command

```
STAT Display the statistics for all drives
STAT x: Display statistics for specified drive
STAT file.ext Display statistics for specified file
STAT x:file.ext Display statistics for specified file on the specified drive
STAT DEV: Display statistics for specified device

Device abbreviations:
CON: - system console
LST: - output list device

ASM X Assemble file named X
```
ASM COMMANDS

ASM A:file.ext  Assemble file named file.ext on Drive A

PIP COMMANDS

PIP  Interactive mode
PIP file.ext=file1.ext  Copy file1.ext to file.ext
PIP file.ext=f1.ext,f2.ext,f3.ext
  Concatenate files f1,f2,f3 into file.ext
PIP LST:=file.ext  Copy file.ext to list device

In all above cases, a drive name may be specified prior to the
file name.

ED COMMANDS

ED file.ext  Edit file.ext if old file or
  create new file and accept data
ED A:file.ext  Edit file.ext on drive A or create
  new file.ext on drive A.

SGEN COMMANDS

SGEN  Interactive program
SGEN filename
SUBMIT COMMANDS

SUBMIT file.sub
Accept commands from file named file.sub

SUBMIT file.sub a b c
Accept commands from file named file.sub and substitute a for $1, b for $2, and c for $3 in input file.

DUMP COMMANDS

DUMP file.ext
Copy the contents of file file.ext to the console in hexadecimal format.

MOVCPM COMMANDS

MOVCPM
Relocate and execute CP/M using the maximum amount of memory available. After relocation, system is executed but not permanently saved.

MOVCPM n
Configure CP/M to operate using n bytes (16 to 64K) as maximum. System not saved.

MOVCPM n *
Same as above but save system for following SGEN.

MOVCPM **
Configure CP/M to use maximum amount of memory and save for SGEN.
APPENDIX B

COMPANY NUMBER SYSTEMS

CONVERSION OF BASE 10 TO BINARY (BASE 2)

Base 10, that is, the numbers 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9, is the most commonly used number system. However, in digital computers, only two numbers are used, 0 and 1. Therefore, in order for the computer to understand an instruction, it must be converted to 0's and 1's. This is where Base 2 comes in, since this number system is composed of only 2 numbers, 0 and 1.

The procedure for converting numbers from Base 10 to Base 2 is by repeatedly dividing the Base 10 number by 2.

For example, convert the number 42 to a Base 2 number:

1) \[ \frac{42}{2} = 21 \text{ Remainder} = 0 \]
2) \[ \frac{21}{2} = 10 \text{ Remainder} = 0 \]
3) \[ \frac{10}{2} = 5 \text{ Remainder} = 0 \]
4) \[ \frac{5}{2} = 2 \text{ Remainder} = 1 \]
5) \[ \frac{2}{2} = 1 \text{ Remainder} = 0 \]
6) \[ \frac{1}{2} = 0 \text{ Remainder} = 1 \]

Keep dividing until you reach an answer of 0, then stop. Now read the remainders in reverse order for the answer in Base 2.

101010

You can check your answer by:

\[
\begin{array}{cccc}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 2 & 0 \\
0 & 8 & 0 & 0 \\
0 & 32 & 0 & 42
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 2 \\
1 & 0 & 0 & 8 \\
1 & 0 & 0 & 32 \\
1 & 0 & 0 & 42
\end{array}
\]

NOTE: Any number to the power of zero is 1.
CONVERSION OF BASE 10 TO OCTAL (BASE 8)

By dividing the string of 0's and 1's formed by the conversion of Base 10 to Base 2 by 8, the total number of digits is reduced. A Base 8 number represents three Base 2 digits, therefore, it is a more convenient number system for digital computers. The format of this conversion is basically the same as that of Base 10 to Base 2; by repeated division of 8.

For this example we will use the number 42 again.

1) \[ 42/8 = 5 \text{ Remainder } 2 \]

2) \[ 5/8 = 0 \text{ Remainder } 5 \]

Again, read the answer from the bottom up.

52

Check this by:

\[
\begin{array}{c}
5 \\
2 \\
\hline
1 \_ \_ \\
2 \times 8^0 = 2 \\
1 \_ \_ 5 \times 8^1 = 40 \\
\hline
42
\end{array}
\]

As we mentioned before, a single Base 8 digit can represent three Base 2 digits. To illustrate this you must convert the Base 10 numbers 1, 2, 3, 4, 5, 6, and 7 to their equivalent Base 2 and Base 8 numbers. (Notice that 8 and 9 are not included. This is because Base 8 consists of only 8 numbers.)

Use the previously given examples to do these conversions.

The equivalents between the bases are shown below:

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 8</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
</tbody>
</table>
From our previous examples you can see that:

| Base 10 | 42  
|--------|-----|
| Base 2 | 101010 
| Base 8 | 52  |

are equivalent.

Notice that the Base 2 number may be written directly from each Base 8 number or visa versa.

\[
101/010 \\
5 / 2
\]

Using these equivalents, it is much easier to convert a Base 10 number to a Base 2 number by first converting the number to its Base 8 equivalent, and then directly converting it to its Base 2 equivalent.

CONVERSION OF BASE 10 TO BASE 16 (HEXADECIMAL)

We have seen the convenience of the Base 2 and Base 8 number systems. However, there is still another system that proves to be even more convenient. This is the Hexadecimal (Base 16) system.

This system is different than the others in the sense that it uses letters as well as numbers.

The numbers used in the hex system are shown below:

\[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F,\]

where \(A=10, B=11, C=12, D=13, E=14, \) and \(F=15\) (decimal)

The relationship between Base 10, Base 16, and Base 2 is shown below.

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 16</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>
To convert Base 10 to Hex the same conversion method as the others is used.

For example, convert 999 to hex.

1) \[999/16 = 62\text{ Remainder} = 7\]
2) \[62/16 = 3 \text{ Remainder} = 14 \text{ (E)}\]
3) \[3/16 = 0 \text{ Remainder} = 3\]

The answer is 3E7. (Remember, from the bottom up.)

To check this:

\[
\begin{align*}
3 & \times 16^0 = 7 \\
E(14) & \times 16^1 = 224 \\
3 & \times 16^2 = 768 \\
\hline
999 \\
\end{align*}
\]

To further illustrate the convenience of the hex system, convert 999 to a binary number.

1) \[999/2 = 499 \text{ Remainder} = 1\]
2) \[499/2 = 249 \text{ Remainder} = 1\]
3) \[249/2 = 124 \text{ Remainder} = 1\]
4) \[124/2 = 62 \text{ Remainder} = 0\]
5) \[62/2 = 31 \text{ Remainder} = 0\]
6) \[31/2 = 15 \text{ Remainder} = 1\]
7) \[15/2 = 7 \text{ Remainder} = 1\]
8) \[7/2 = 3 \text{ Remainder} = 1\]
9) \[3/2 = 1 \text{ Remainder} = 1\]
10) \[1/2 = 0 \text{ Remainder} = 1\]

The answer is 1111100111.

Now convert 999 to binary by first converting it to hex.

1) \[999/16 = 62 \text{ Remainder} = 7\]
2) \[62/16 = 3 \text{ Remainder} = 14 \text{ (E)}\]
3) \[3/16 = 0 \text{ Remainder} = 3\]
The answer is 3E7 hex. Notice from the above chart that:

\[
\begin{align*}
3 &= 0011 \\
E &= 1110 \\
7 &= 0111
\end{align*}
\]

We can now see that 999 = 3E7 = 1111100111. (The first two 0's can be left off, seeing as they don't mean anything).

The hex system is also convenient for converting a 16 digit binary number to its base 10 equivalent by either first converting it to hex, or by digit by digit conversion.

The first example is:

\[
1111100111
\]

First separate into groups of 4 starting from the left, and give the hex equivalent.

\[
\begin{align*}
0011 & 1110 & 0111 \\
3 & E & 7
\end{align*}
\]

Now convert 3E7 to Base 10.

\[
\begin{align*}
3 & E & 7 \\
\mid & \mid & \mid \\
7 & \times 16^0 &= 7 \\
E & \times 16^1 &= 224 \\
3 & \times 16^2 &= 768 \\
\hline
999 & \text{ Base 10}
\end{align*}
\]

The other example is:

\[
\begin{align*}
0 & 0 \& 1 \& 1 \& 1 \& 1 \& 1 \& 0 \& 0 \& 1 \& 1 \& 1 \& 1 \\
\mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid & \mid \\
1 & \times 2^0 &= 1 \\
1 & \times 2^1 &= 2 \\
1 & \times 2^2 &= 4 \\
0 & \times 2^3 &= 0 \\
0 & \times 2^4 &= 0 \\
1 & \times 2^5 &= 32 \\
1 & \times 2^6 &= 64 \\
1 & \times 2^7 &= 128 \\
1 & \times 2^8 &= 256 \\
1 & \times 2^9 &= 512 \\
0 & \times 2^{10} &= 0 \\
0 & \times 2^{11} &= 0 \\
\hline
999 & \text{ Base 10}
\end{align*}
\]
APPENDIX C
LINE EDITING AND OUTPUT CONTROL

While typing command lines at the CCP prompt ( A> ), certain line editing functions are allowed:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>Delete the last character typed</td>
</tr>
<tr>
<td>CTRL-U</td>
<td>Delete the entire line</td>
</tr>
<tr>
<td>CTRL-X</td>
<td>Same as CTRL-U</td>
</tr>
<tr>
<td>CTRL-R</td>
<td>Retype the current command line: types a &quot;clean line&quot; following character deletion with deletes.</td>
</tr>
<tr>
<td>CTRL-E</td>
<td>Physical end of line: Carriage is returned but line is not sent until a &lt;CR&gt; is executed.</td>
</tr>
<tr>
<td>CTRL-C</td>
<td>CP/M system reboot</td>
</tr>
<tr>
<td>CTRL-Z</td>
<td>End input from the console (used in ED and PIP).</td>
</tr>
</tbody>
</table>

CTRL-P and CTRL-S affect console output as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-P</td>
<td>Copy all subsequent console output to the currently assigned console list device. Output is sent to both the list device and the console device until the next CTRL-P is typed.</td>
</tr>
<tr>
<td>CTRL-S</td>
<td>Temporarily stops the console output. Program execution and output continue when the next character is typed at the console (e.g., another CTRL-S). This feature is used to stop output at high speed consoles, such as CRT's in order to view a segment of output before continuing.</td>
</tr>
</tbody>
</table>
APPENDIX D

A file on your system disk describes what has been included. To display this file, perform the following:

A> TYPE DISK.DOC <CR>

The listing of the file will now be displayed on your CRT. Since this file is longer than one screen, CTRL-S may be used to stop transmission and allow viewing. To resume transmission, type CTRL-Q. This may be repeated as many times as required.

WHAT IS ON YOUR SYSTEM DISK?

Digital Research standard CP/M 2.2 release programs:

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM.COM</td>
<td>8080 assembler, .HEX output</td>
</tr>
<tr>
<td>DDT.COM</td>
<td>.COM file debugger</td>
</tr>
<tr>
<td>ED.COM</td>
<td>Source file editor</td>
</tr>
<tr>
<td>LOAD.COM</td>
<td>Generates .COM file from .HEX file</td>
</tr>
<tr>
<td>MOVCPM.COM</td>
<td>Makes CP/M system image in memory</td>
</tr>
<tr>
<td>PIP.COM</td>
<td>File transfer utility</td>
</tr>
<tr>
<td>STAT.COM</td>
<td>File attributes and status utility</td>
</tr>
<tr>
<td>SUBMIT.COM</td>
<td>Batched command processor</td>
</tr>
<tr>
<td>XSUB.COM</td>
<td>(extension)</td>
</tr>
<tr>
<td>DUMP.COM</td>
<td>ASCII file &quot;hex dump&quot; utility</td>
</tr>
</tbody>
</table>

Microsoft "utilities" and BASIC:

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M80.COM</td>
<td>8080 or Z-80 macro assembler, .REL output.</td>
</tr>
<tr>
<td>L80.COM</td>
<td>Linker for .REL files, generates .COM files</td>
</tr>
<tr>
<td>CREF80.COM</td>
<td>Cross reference utility for M80 assembler.</td>
</tr>
<tr>
<td>LIB80.COM</td>
<td>.REL file librarian</td>
</tr>
<tr>
<td>CBAS2.COM</td>
<td>CBASIC compiler language</td>
</tr>
<tr>
<td>CRVN2.COM</td>
<td>CBASIC compiler language</td>
</tr>
<tr>
<td>XREF.COM</td>
<td>CBASIC compiler language</td>
</tr>
</tbody>
</table>
TELCON supplied programs and sources

SGEN.COM
PATCH.COM
PATCH.Z80
SETUP.COM
SETUP.Z80
FORMAT.COM
FORMAT.Z80
TRACKRD.COM
TRACKRD.Z80
RELOAD.COM
SYSEQU.Z80
CPMBIOS.Z80
CPMBEOU.Z80
GEN.SUB
GENZ.SUB
KEY.MAC
CPMBOOT.Z80
DISK.DOC

SYSGEN.COM replacement, copies
disk "system".
Run this to install system patches
(source for above)
System configurator program
(source for above)
Universal disk formatter
(source for above)
Special disk reading program, for
engineers.
(source for above)
Utility used during BIOS regeneration.
(part of FORMAT, SETUP, PATCH sources)
CP/M BIOS source file.
(part of CPMBIOS source).
SUBMIT procedure for BIOS regression.
(second half of GEN.SUB).
Keyboard PROM contents, can be
assembled.
Monitor PROM contents, can be assembled.
Explains files on disk (you are
reading it).

PROGRAM GENERATION PROCEDURE
for all source files

A> M80 file = file.Z80  ... where 'file' is PATCH,
                          FORMAT, SETUP
                          (EX. A>M80 PATCH=PATCH.Z80)

A> L80 file/N, file/E  ... same rules
                         (EX. A>L80 PATCH/N,PATCH/E)
A>

... now you have file.COM on
your disk.

For generation procedure of 'system track' using CPMBIOS.SRC,
type GEN.SUB and GENZ.SUB which contain the entire assembly
and disk write procedures for creating a system disk from BIOS
source. The GEN files may be executed in their entirety
(typically following a BIOS change) by typing:

A> SUBMIT GEN
APPENDIX E

MEMORY AND DISK LAYOUT

64k 10000H ------------------------ top of 64k.
                      (FF80H interrupts)
                      FREE
                      (FDD0H)

63k FC00H

FA00H

DISPLAY
RAM

62k F800H

F600H

61k F400H

F200H

EXT. DATA
(see SETUP)

60k F000H

EE00H

59k EC00H

EA00H

BIOS
(unloadable part)

58k E800H

E600H

57k E400H

E200H

BIOS
(loadable part)
<table>
<thead>
<tr>
<th>Section Name</th>
<th>FROM addr</th>
<th>TO addr</th>
<th>SIZE</th>
<th>CP/M Sectors</th>
<th>DISK Sectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free Space</td>
<td>FF96H</td>
<td>FFFFH</td>
<td>6AH, 106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt vectors</td>
<td>FF80H</td>
<td>FF95H</td>
<td>16H, 22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free space</td>
<td>FDD1H</td>
<td>FF75H</td>
<td>1APH, 431</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Screen RAM</td>
<td>F600H</td>
<td>FDD0H</td>
<td>7DOH, 2000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ext. data (SETUP)</td>
<td>F200H</td>
<td>F5FFH</td>
<td>400H, 1024</td>
<td>72-79</td>
<td>19,20</td>
</tr>
<tr>
<td>BIOS, unloadable</td>
<td>F000H</td>
<td>F1FFH</td>
<td>200H, 512</td>
<td>44-71</td>
<td>12-18</td>
</tr>
<tr>
<td>BIOS, loadable</td>
<td>E200H</td>
<td>E5FFH</td>
<td>E00H, 3584</td>
<td>16-43</td>
<td>5-11</td>
</tr>
<tr>
<td>EDOS</td>
<td>D400H</td>
<td>E1FFH</td>
<td>E00H, 3584</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP</td>
<td>CC00H</td>
<td>D3FFH</td>
<td>800H, 2048</td>
<td>0-15</td>
<td>1-4</td>
</tr>
<tr>
<td>user RAM (TPA)</td>
<td>100H</td>
<td>FFFFH</td>
<td>C800H, 51200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP/M base page</td>
<td>0</td>
<td>FPH</td>
<td>100H, 256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address Range</th>
<th>CP/M Sector</th>
<th>DISK Sector</th>
<th>Section Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>F580H - F5FFH</td>
<td>79</td>
<td>20</td>
<td>EXT. DATA BLOCK</td>
</tr>
<tr>
<td>F500H - F57FFH</td>
<td>78</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>F480H - F4FFH</td>
<td>77</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>F400H - F47FFH</td>
<td>76</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>F380H - F3FFH</td>
<td>75</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>F300H - F37FFH</td>
<td>74</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>F280H - F2FFH</td>
<td>73</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>F200H - F27FFH</td>
<td>72</td>
<td>19</td>
<td>EXT. DATA BLOCK</td>
</tr>
<tr>
<td>E580H - E5FFH</td>
<td>71</td>
<td>18</td>
<td>BIOS, loadable</td>
</tr>
<tr>
<td>E500H - E57FFH</td>
<td>70</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>E480H - E4FFH</td>
<td>69</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>E400H - E47FFH</td>
<td>68</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>E380H - E3FFH</td>
<td>67</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>E300H - E37FFH</td>
<td>66</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>E280H - E2FFH</td>
<td>65</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>E200H - E27FFH</td>
<td>64</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>EC80H - ECFFH</td>
<td>63</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>EC00H - EC7FFH</td>
<td>62</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>EB80H - EBF0H</td>
<td>61</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>EB00H - EBF7H</td>
<td>60</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>EA80H - EAF0H</td>
<td>59</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>FA00H - FAF7H</td>
<td>58</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>F980H - FE0F0H</td>
<td>57</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>F900H - FE0FH</td>
<td>56</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>E880H - EBF0H</td>
<td>55</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>E800H - EBF7H</td>
<td>54</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>E700H - EBE0H</td>
<td>53</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>E680H - EBE7H</td>
<td>52</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>E600H - EBE7F0H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E580H - E5FFH</td>
<td>51</td>
<td>13</td>
<td>.</td>
</tr>
<tr>
<td>E500H - E57FH</td>
<td>50</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>E480H - E4FFH</td>
<td>49</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>E400H - E47FH</td>
<td>48</td>
<td>13</td>
<td>.</td>
</tr>
<tr>
<td>E380H - E3FFH</td>
<td>47</td>
<td>12</td>
<td>.</td>
</tr>
<tr>
<td>E300H - E37FH</td>
<td>46</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>E280H - E2FFH</td>
<td>45</td>
<td>12</td>
<td>BIOS, loadable</td>
</tr>
<tr>
<td>E200H - E27FH</td>
<td>44</td>
<td>11</td>
<td>BDOS ------ BDOS</td>
</tr>
<tr>
<td>E180H - E1FFH</td>
<td>43</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>E100H - E17FH</td>
<td>42</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>E080H - E0FFH</td>
<td>41</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>E000H - E07FH</td>
<td>40</td>
<td>9</td>
<td>.</td>
</tr>
<tr>
<td>DF80H - DFFFH</td>
<td>39</td>
<td>8</td>
<td>.</td>
</tr>
<tr>
<td>DF00H - DF7FH</td>
<td>38</td>
<td>7</td>
<td>.</td>
</tr>
<tr>
<td>DE80H - DEFFH</td>
<td>37</td>
<td>6</td>
<td>.</td>
</tr>
<tr>
<td>DE00H - DE7FH</td>
<td>36</td>
<td>5</td>
<td>.</td>
</tr>
<tr>
<td>DD80H - DDFH</td>
<td>35</td>
<td>4</td>
<td>.</td>
</tr>
<tr>
<td>DD00H - DD7FH</td>
<td>34</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>DC80H - DCFH</td>
<td>33</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>DC00H - DC7FH</td>
<td>32</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>DB80H - DBFFH</td>
<td>31</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>DB00H - DB7FH</td>
<td>30</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>DA80H - DAFFH</td>
<td>29</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>DA00H - DA7FH</td>
<td>28</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D980H - D9FFH</td>
<td>27</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D900H - D97FH</td>
<td>26</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D880H - D8FFH</td>
<td>25</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D800H - D87FH</td>
<td>24</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D780H - D7FFH</td>
<td>23</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D700H - D77FH</td>
<td>22</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D680H - D6FFH</td>
<td>21</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D600H - D67FH</td>
<td>20</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D580H - D5FFH</td>
<td>19</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D500H - D57FH</td>
<td>18</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D480H - D4FFH</td>
<td>17</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D400H - D47FH</td>
<td>16</td>
<td>5</td>
<td>BDOS ------ BDOS</td>
</tr>
<tr>
<td>D380H - D3FFH</td>
<td>15</td>
<td>4</td>
<td>CCP -------- CCP</td>
</tr>
<tr>
<td>D300H - D37FH</td>
<td>14</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D280H - D2FFH</td>
<td>13</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D200H - D27FH</td>
<td>12</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>D180H - D1FFH</td>
<td>11</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>D100H - D17FH</td>
<td>10</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>D080H - D0FFH</td>
<td>9</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>D000H - D07FH</td>
<td>8</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CF80H - CFFH</td>
<td>7</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CF00H - CF7FH</td>
<td>6</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CE80H - CEFFH</td>
<td>5</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CE00H - CE7FH</td>
<td>4</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>CD80H - CDFH</td>
<td>3</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>CD00H - CD7FH</td>
<td>2</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CC80H - CCFH</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CC00H - CC7FH</td>
<td>0</td>
<td>1</td>
<td>CCP -------- CCP</td>
</tr>
</tbody>
</table>
APPENDIX F

Port Addresses and Names

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D$CTCO</td>
<td>00H</td>
<td>;counter timer (INTEL 8254) counter 0</td>
</tr>
<tr>
<td>D$CTC1</td>
<td>01H</td>
<td>;counter timer (INTEL 8254) counter 1</td>
</tr>
<tr>
<td>D$CTC2</td>
<td>02H</td>
<td>;counter timer (INTEL 8254) counter 2</td>
</tr>
<tr>
<td>D$CTCS</td>
<td>03H</td>
<td>;counter timer (INTEL 8254) command/status</td>
</tr>
<tr>
<td>D$SLPR</td>
<td>04H</td>
<td>;select prom</td>
</tr>
<tr>
<td>D$SLRM</td>
<td>05H</td>
<td>;select ram</td>
</tr>
<tr>
<td>D$CRDT</td>
<td>10H</td>
<td>;crt controller (INTEL 8275) data</td>
</tr>
<tr>
<td>D$CRCs</td>
<td>11H</td>
<td>;crt controller (INTEL 8275) command status</td>
</tr>
<tr>
<td>D$CMDT</td>
<td>20H</td>
<td>;communications usart (INTEL 8251A) data</td>
</tr>
<tr>
<td>D$CMCS</td>
<td>21H</td>
<td>;communications usart (INTEL 8251A) command/status</td>
</tr>
<tr>
<td>D$PRDT</td>
<td>22H</td>
<td>;printer usart (INTEL 8251A) data</td>
</tr>
<tr>
<td>D$PRCS</td>
<td>23H</td>
<td>;printer usart (INTEL 8251A) command/status</td>
</tr>
<tr>
<td>D$KBDT</td>
<td>24H</td>
<td>;keyboard usart (8251A) data</td>
</tr>
<tr>
<td>D$KBCS</td>
<td>25H</td>
<td>;keyboard usart (8251A) command/status</td>
</tr>
<tr>
<td>D$UIMK</td>
<td>26H</td>
<td>;usrart interrupt mask byte</td>
</tr>
<tr>
<td>D$DMIO</td>
<td>30H</td>
<td>;dma (ZILOG 8410) input/output</td>
</tr>
<tr>
<td>D$FCCS</td>
<td>40H</td>
<td>;floppy controller (WD 1793) command/status</td>
</tr>
<tr>
<td>D$FCTR</td>
<td>41H</td>
<td>;floppy controller (WD 1793) track</td>
</tr>
</tbody>
</table>
D$PCSC  42H ;floppy controller (WD 1793) sector
D$FCDT  43H ;floppy controller (WD 1793) data
D$PIDT  50H ;floppy interface pia (MOTOROLA 6821) data
D$FICS  51H ;floppy interface pia (MOTOROLA 6821) command/status
D$PLDT  52H ;parallel interface pia (MOTOROLA 6821) data
D$PLCS  53H ;parallel interface pia (MOTOROLA 6821) command/status
D$IAADT 60H ;IEEE pia a (MOTOROLA 6821) data
D$IAACS 61H ;IEEE pia a (MOTOROLA 6821) command/status
D$IBDT  62H ;IEEE pia b (MOTOROLA 6821) data
D$IBCS  63H ;IEEE pia b (MOTOROLA 6821) command/status
APPENDIX G

Keyboard Layout With Address
and Values Generated

F-1 KEYBOARD NUMBERING SYSTEM

The keyboard is laid out such that there are six horizontal rows of keys. The keys are physically numbered 1 thru 95, with number one being key F1 in the upper left hand corner and key 95, the minus sign in the lower right hand corner.

The return key on the right side of the keyboard, which occupies a position on rows 3 and 4, is numbered by 54.

The first row of keys, F1-F19 are numbered 1 thru 19.
The second row of keys, PRINT - 9 are numbered 20 thru 38.
The third row of keys BREAK - 6 are numbered 39 thru 57.
The fourth row of keys CAPS LOCK - 3 are numbered 58 thru 74.
The fifth row of keys UP ARROW - (decimal point) are numbered 75 thru 90.
The final row of keys, LEFT ARROW - minus sign are numbered 91 thru 95.

The above numbering scheme is used in various tables in this appendix.
F2 KEYBOARD PROM MEMORY MAP (2716)

Each section is 128 bytes in length.

<table>
<thead>
<tr>
<th>3FF</th>
<th>CAPS LOCK</th>
<th>SHIFT/CONTROL MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>380F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>CAPS LOCK</td>
<td>CONTROL MODE</td>
</tr>
<tr>
<td>2FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27F</td>
<td>CAPS LOCK</td>
<td>NORMAL MODE</td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1FF</td>
<td></td>
<td>SHIFT/CONTROL MODE</td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17F</td>
<td></td>
<td>CONTROL MODE</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[OFF]</td>
<td></td>
<td>SHIFT MODE</td>
</tr>
<tr>
<td>080</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07F</td>
<td></td>
<td>NORMAL MODE</td>
</tr>
<tr>
<td>000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F2-1

2716 Memory Layout

This Prom contains 8 segments of 128 bytes each. Each byte will then index into the character generator prom to produce the character representation on the CRT. Included with your system is a file called KEY.MAC which explains the contents of the keyboard prom.
F3 CHARACTER GENERATOR PROM (2732)

To determine the address within the character generator Prom, use the number in ( ) and multiply by 16; as each character occupies 16 bytes of data. The standard characters use a 5X7 dot matrix, and the graphic characters use an 8X10 dot matrix.

For standard character width use bits 0-4.
For standard character height use bytes 0-6.
For lower case decenders use byte 7.
For graphic characters width use bits 0-7.
For graphic characters height use bytes 0-9.
APPENDIX H

G-1 GRAPHIC CHARACTER SET

The following examples are the graphic symbols which will be output when in graphics mode. These symbols are created using an 8X10 dot matrix.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>**</td>
<td>1</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>**</td>
<td>2</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>***</td>
<td>3</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>*****</td>
<td>4</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>*****</td>
<td>5</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>*****</td>
<td>6</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>***</td>
<td>7</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>**</td>
<td>8</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>*</td>
<td>9</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>*</td>
<td>10</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

UP-ARROW
value 5E (94)

CHARACTER -
value 5F (95)

CHARACTER /
value 60 (96)

CHARACTER a
value 61 (97)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>**</td>
<td>1</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>**</td>
<td>2</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>**</td>
<td>3</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>**</td>
<td>4</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>*****</td>
<td>5</td>
<td>****</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>*****</td>
<td>6</td>
<td>****</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>**</td>
<td>7</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>**</td>
<td>8</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>**</td>
<td>9</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>**</td>
<td>10</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CHARACTER b
value 62 (98)

CHARACTER c
value 63 (99)

CHARACTER d
value 64 (100)

CHARACTER e
value 65 (101)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>* *</td>
<td>1</td>
<td>* *</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>2</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>3</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>*****</td>
<td>4</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>*****</td>
<td>5</td>
<td>****</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>*</td>
<td>6</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>*</td>
<td>7</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>*****</td>
<td>8</td>
<td>*****</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>*</td>
<td>9</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>*</td>
<td>10</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CHARACTER f
value 65 (102)

CHARACTER g
value 66 (103)

CHARACTER h
value 67 (104)

CHARACTER i
value 68 (105)
<table>
<thead>
<tr>
<th>CHARACTER j</th>
<th>CHARACTER k</th>
<th>CHARACTER l</th>
<th>CHARACTER m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>* * *</td>
<td>5</td>
<td>5 **</td>
</tr>
<tr>
<td>6</td>
<td>***</td>
<td>6 **</td>
<td>6 **</td>
</tr>
<tr>
<td>7</td>
<td>*</td>
<td>7 **</td>
<td>7 **</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>8 **</td>
<td>8 **</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>9 **</td>
<td>9 **</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>10 **</td>
<td>10 **</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHARACTER n</th>
<th>CHARACTER o</th>
<th>CHARACTER p</th>
<th>CHARACTER q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 **</td>
<td>1 **</td>
<td>1 **</td>
<td>1 **</td>
</tr>
<tr>
<td>2 **</td>
<td>2 **</td>
<td>2 **</td>
<td>2 **</td>
</tr>
<tr>
<td>3 **</td>
<td>3 **</td>
<td>3 **</td>
<td>3 **</td>
</tr>
<tr>
<td>4 **</td>
<td>4 **</td>
<td>4 **</td>
<td>4 **</td>
</tr>
<tr>
<td>5 **</td>
<td>5 **</td>
<td>5 **</td>
<td>5 **</td>
</tr>
<tr>
<td>6 **</td>
<td>6 **</td>
<td>6 **</td>
<td>6 **</td>
</tr>
<tr>
<td>7 **</td>
<td>7 **</td>
<td>7 **</td>
<td>7 **</td>
</tr>
<tr>
<td>8 **</td>
<td>8 **</td>
<td>8 **</td>
<td>8 **</td>
</tr>
<tr>
<td>9 **</td>
<td>9 **</td>
<td>9 **</td>
<td>9 **</td>
</tr>
<tr>
<td>10 **</td>
<td>10 **</td>
<td>10 **</td>
<td>10 **</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHARACTER r</th>
<th>CHARACTER s</th>
<th>CHARACTER t</th>
<th>CHARACTER u</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5 **</td>
<td>5 **</td>
<td>5 **</td>
</tr>
<tr>
<td>6</td>
<td>6 **</td>
<td>6 **</td>
<td>6 **</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>1 **</td>
<td>1 *</td>
<td>1 *</td>
<td>1 *</td>
</tr>
<tr>
<td>2 **</td>
<td>2 **</td>
<td>2 **</td>
<td>2 **</td>
</tr>
<tr>
<td>3 **</td>
<td>3 **</td>
<td>3 **</td>
<td>3 **</td>
</tr>
<tr>
<td>4 **</td>
<td>4 ****</td>
<td>4 **</td>
<td>4 **</td>
</tr>
<tr>
<td>5 *****</td>
<td>5 **</td>
<td>5 **</td>
<td>5 **</td>
</tr>
<tr>
<td>6 *****</td>
<td>6 **</td>
<td>6 **</td>
<td>6 **</td>
</tr>
<tr>
<td>7 **</td>
<td>7 ****</td>
<td>7 **</td>
<td>7 **</td>
</tr>
<tr>
<td>8 **</td>
<td>8 **</td>
<td>8 **</td>
<td>8 **</td>
</tr>
<tr>
<td>9 **</td>
<td>9 **</td>
<td>9 **</td>
<td>9 **</td>
</tr>
<tr>
<td>10 **</td>
<td>10 *</td>
<td>10 *</td>
<td>10 *</td>
</tr>
</tbody>
</table>

**CHARACTER v**
value 75 (118)

**CHARACTER w**
value 76 (119)

**CHARACTER x**
value 77 (120)

**CHARACTER y**
value 78 (121)

<table>
<thead>
<tr>
<th>12345678</th>
<th>12345678</th>
<th>12345678</th>
<th>12345678</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ********</td>
<td>1</td>
<td>1 **</td>
<td>1</td>
</tr>
<tr>
<td>2 ********</td>
<td>2</td>
<td>2 **</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3 **</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4 **</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5 **</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6 **</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7 **</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8 **</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9 ********</td>
<td>9 **</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10 ********</td>
<td>10 **</td>
<td>10</td>
</tr>
</tbody>
</table>

**CHARACTER z**
value 79 (122)

**CHARACTER {**
value 7A (123)

**CHARACTER :**
value 7B (124)

**CHARACTER }**
value 7C (125)

<table>
<thead>
<tr>
<th>12345678</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

**CHARACTER ~**
value 7D (126)
## APPENDIX I

### H-1 PORT PIN ASSIGNMENTS

The following figures are the pin assignments for the Ports located on the back of the unit.

#### H 1.1 Parallel/IEEE 488 Interface

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Service Request (SRQ)</td>
<td>2</td>
<td>Attention (ATN)</td>
</tr>
<tr>
<td>3</td>
<td>Interface Clear (IFC)</td>
<td>4</td>
<td>Not ready for Data (NRFD)</td>
</tr>
<tr>
<td>5</td>
<td>Data Input/Output (DIO7)</td>
<td>6</td>
<td>Data Input/Output (DIO6)</td>
</tr>
<tr>
<td>7</td>
<td>Data Input/Output (DIO5)</td>
<td>8</td>
<td>Data Input/Output (DIO4)</td>
</tr>
<tr>
<td>9</td>
<td>Data Input/Output (DIO3)</td>
<td>10</td>
<td>Data Input/Output (DIO2)</td>
</tr>
<tr>
<td>11</td>
<td>Data Input/Output (DIO1)</td>
<td>12</td>
<td>Data Input/Output (DIO0)</td>
</tr>
<tr>
<td>13</td>
<td>Not used</td>
<td>14</td>
<td>Data Valid (DAV)</td>
</tr>
<tr>
<td>15</td>
<td>Remote Enable (REN)</td>
<td>16</td>
<td>Not Data Accepted (NDAC)</td>
</tr>
<tr>
<td>17</td>
<td>End or Identify (EOI)</td>
<td>18</td>
<td>Ground</td>
</tr>
<tr>
<td>19</td>
<td>Ground</td>
<td>20</td>
<td>Ground</td>
</tr>
<tr>
<td>21</td>
<td>Ground</td>
<td>22</td>
<td>Ground</td>
</tr>
<tr>
<td>23</td>
<td>Ground</td>
<td>24</td>
<td>Ground</td>
</tr>
<tr>
<td>25</td>
<td>Ground</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### H 1.2 EIA RS232C (CCITT V.24/28) Communications Port

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ground</td>
<td>2</td>
<td>Transmit Data (TXD)</td>
</tr>
<tr>
<td>3</td>
<td>Receive Data (RXD)</td>
<td>4</td>
<td>Request to Send (RTS)</td>
</tr>
<tr>
<td>5</td>
<td>Clear to Send (CTS)</td>
<td>6</td>
<td>Data Set Ready (DSR)</td>
</tr>
<tr>
<td>7</td>
<td>Ground</td>
<td>8</td>
<td>Not used</td>
</tr>
<tr>
<td>9</td>
<td>Not used</td>
<td>10</td>
<td>Not used</td>
</tr>
<tr>
<td>11</td>
<td>+12V output *</td>
<td>12</td>
<td>+12V output *</td>
</tr>
<tr>
<td>13</td>
<td>-12V output *</td>
<td>14</td>
<td>+5V output *</td>
</tr>
<tr>
<td>15</td>
<td>External Transmit Clock (TXC)</td>
<td>16</td>
<td>16X TTL Clock *</td>
</tr>
<tr>
<td>17</td>
<td>External Receive Clock (RXC)</td>
<td>18</td>
<td>-12V Output *</td>
</tr>
<tr>
<td>19</td>
<td>+5V output *</td>
<td>20</td>
<td>Data Terminal Ready (DTR)</td>
</tr>
<tr>
<td>21</td>
<td>Audio Signal from TAC-212 +</td>
<td>22</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>modem</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Not used</td>
<td>24</td>
<td>Not used</td>
</tr>
<tr>
<td>25</td>
<td>Not used</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* = To TAC 212 + EX Modem
### H 1.3 SERIAL RS232C (CCITT V.24/28)
Printer Port and 8-Bit TTL Parallel Port

<table>
<thead>
<tr>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground</td>
<td>Transmit data output to printer</td>
</tr>
<tr>
<td>Receive data input from</td>
<td>Request to send output to printer</td>
</tr>
<tr>
<td>printer</td>
<td>Not used</td>
</tr>
<tr>
<td>Clear to send input from</td>
<td>Not used</td>
</tr>
<tr>
<td>printer</td>
<td></td>
</tr>
<tr>
<td>Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>Ground</td>
<td>Ground</td>
</tr>
<tr>
<td>+5V</td>
<td>Ground</td>
</tr>
<tr>
<td>Ground</td>
<td>CB2 Parallel Port</td>
</tr>
<tr>
<td>CB1 Parallel Port</td>
<td>PB7 Parallel Port</td>
</tr>
<tr>
<td>PB6 Parallel Port</td>
<td>PB5 Parallel Port</td>
</tr>
<tr>
<td>PB4 Parallel Port</td>
<td>PB3 Parallel Port</td>
</tr>
<tr>
<td>PB2 Parallel Port</td>
<td>PB1 Parallel Port</td>
</tr>
<tr>
<td>PB0 Parallel Port</td>
<td></td>
</tr>
<tr>
<td><strong>GLOSSARY</strong></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td><strong>A&gt;</strong></td>
<td>Symbol signifying that the computer is ready for a command. CCP Prompt.</td>
</tr>
<tr>
<td><strong>ADDRESS</strong></td>
<td>The location in a computer's storage compartment of an item of information identified by a number or other code.</td>
</tr>
<tr>
<td><strong>ALPHANUMERIC</strong></td>
<td>Containing both alphabetic and numeric symbols.</td>
</tr>
<tr>
<td><strong>AMBIGUOUS FILE REFERENCE</strong></td>
<td>Allows user to access a file when the complete filename is not known. Example: A&gt;DIR TEST.*</td>
</tr>
<tr>
<td><strong>APPLICATION PROGRAM</strong></td>
<td>A program that performs a specific task for the user. In general, the term refers to any task that is not part of the operating system.</td>
</tr>
<tr>
<td><strong>ARITHMETIC OPERATOR</strong></td>
<td>Any symbol or term indicating that a certain process, substitution, etc. is to be carried out.</td>
</tr>
<tr>
<td><strong>ASCII</strong></td>
<td>American Standard Code for Information Interchange. A code used by computers to translate letters, numbers, and symbols from a keyboard into machine code, and vise versa.</td>
</tr>
<tr>
<td><strong>ASM</strong></td>
<td>An 8080 machine language assembler supplied by Digital Research.</td>
</tr>
<tr>
<td><strong>ASSEMBLER</strong></td>
<td>Converts symbolic instructions (mnemonics) into the binary machine code which the 280 can execute.</td>
</tr>
<tr>
<td><strong>BAUD RATE</strong></td>
<td>The speed at which serial data is sent by an RS232 port. The number of bits transmitted per second.</td>
</tr>
<tr>
<td><strong>BDOS</strong></td>
<td>Basic Disk Operating System. Does file management and other programs. (Makes programming tasks simpler).</td>
</tr>
</tbody>
</table>
**BINAR Y MACHINE CODE**
The internal instruction format actually used by the computer. It is called binary because only two characters, 0 and 1, are used in this code.

**BIOS**
Basic Input Output System. Software that interfaces CP/M and other programs to the I/O devices.

**BIT**
A binary digit. The smallest piece of information a computer can handle.

**BOOT**
Loading of the operating system (CP/M) from the system track of the disk in Drive 0. If the machine has just been powered up or reset, all of the system track must be loaded, which is called a cold boot. When most transient programs exit or a user types CTRL-C, a warm boot is performed that reads only the CCP and BDOS.

**BUFFER**
The space in the computer's memory where text is temporarily stored while the computer is on. The buffer serves as the space where part of a text file is stored while it is being edited. The buffer is the space between the computer's main (or core) memory and the disk file.

**BUFFER NAME**
The name assigned to the buffer space. The buffer name consists of the first part of the filename assigned to the disk file of a document.

**BUILTIN COMMAND**
A command that is part of the CCP. The built-in commands are ERA, SAVE, TYPE, USER, and DIR.

**BYTE**
8 bits. The size of a memory location. A computer "word".

**CCP**
The CCP is responsible for reading and interpreting the commands that the user types. It is also responsible for interfacing with the proper diskette that the user specifies.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKSUM</td>
<td>An additional piece of information accompanying a group of information to insure the integrity of the group.</td>
</tr>
<tr>
<td>CODE</td>
<td>Binary information that instructs a microprocessor what to do.</td>
</tr>
<tr>
<td>COMMAND</td>
<td>An instruction or request to the operating system to perform a particular action.</td>
</tr>
<tr>
<td>COMMUNICATIONS PORT</td>
<td>An I/O device that sends and receives data one bit at a time (with the bits being reassembled by the I/O device back into bytes.) Usually conforming to an industry standard number RS232, With RD (Receive Data), TD (Transmit Data) and control signals.</td>
</tr>
<tr>
<td>CONCATENATE</td>
<td>Link more than one command together. CC&amp; (&amp; is the usual symbol for concatenate).</td>
</tr>
<tr>
<td>CONSOLE DEVICE</td>
<td>Logical name for the device in which CP/M receives all commands and sends all responses. In the ZORBA, this normally refers to the keyboard &amp; CRT.</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>Variable that always has the same value.</td>
</tr>
<tr>
<td>CONTROL KEYS</td>
<td>Keys that when depressed simultaneously with the CTRL key, emit a different code than that normally assigned to the key.</td>
</tr>
<tr>
<td>COPY</td>
<td>To create an exact copy of a working disk.</td>
</tr>
<tr>
<td>CP/M</td>
<td>This is the Control Program Monitor that provides a standard operating system for microcomputers. It acts as the interface between the computer hardware and software. The major advantage of CP/M is that it permits software to be interchangeable between computers. In addition, CP/M provides numerous operating utilities to copy disks, rename and erase files, list the contents of disks, and more.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CP/M DISK</td>
<td>Disk supplied with the system by Telcon.</td>
</tr>
<tr>
<td>CP/M OPERATING SYSTEM</td>
<td>See Operating System</td>
</tr>
<tr>
<td>&lt;CR&gt;</td>
<td>Carriage Return. Performs the same function as the return key on a type-writer.</td>
</tr>
<tr>
<td>CRT</td>
<td>A Cathode Ray Tube. The display screen of a computer terminal.</td>
</tr>
<tr>
<td>CTRL</td>
<td>This is the prefix used in the manual to refer to Control commands. These are formed by holding down the CTRL key and depressing another key. The symbol for CTRL is ^ . Example: ^Z</td>
</tr>
<tr>
<td>CTRL-S</td>
<td>Temporarily stops display of text after a TYPE command.</td>
</tr>
<tr>
<td>CTRL-Q</td>
<td>Resumes display of text after a TYPE command.</td>
</tr>
<tr>
<td>CURSOR</td>
<td>Small square on the CRT screen which indicates where the next character will be typed.</td>
</tr>
<tr>
<td>DATA</td>
<td>Term used for any representation of facts, concepts, or instructions in a form suitable for communication, interpretation, or processing.</td>
</tr>
<tr>
<td>DATA FIELD</td>
<td>The basic storage unit of a data record, capable of holding a single piece of information.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>To find and correct errors and defects.</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>A value or operation that is automatically included in a command unless otherwise specified. In most cases, default settings will be what is normal or expected, and you will not even notice they are being used.</td>
</tr>
</tbody>
</table>
DELIMITER
A character that separates, terminates, or organizes elements of a character string, statement or task.

DESTINATION
The place toward which information is to be sent.

DIP SWITCH
Dual In-line Package Switch. Used for configuration of the system.

DIRECTIVE
Tells the assembler how to interpret instructions.

DISKETTE
Thin plastic disk with a magnetic coating where information is stored magnetically.

DISK DRIVE
Reads information from, and writes information onto the diskettes.

DRIVE A
Also referred to as Drive 0.
For Zorba - left drive
For Nomis - top drive

DRIVE B
Also referred to as Drive 1.
For Zorba - right drive
For Nomis - bottom drive

ENVIRONMENT
An environment is a description of how the result should look. Environments control such things as margins, line spacing, whether lines wrap, and how paragraphs should start.

ERROR MESSAGE
Error messages are sent by the system when some action you have requested fails. Each error message identifies the particular part that detected the error.

FIELD
The term field usually refers to a portion of a command or a command element. The name and filetype are two fields of the filespec, for instance.
FIELD LENGTH
The total number of column spaces that a data field occupies. Data entered into a field cannot exceed the length of the field.

FILESPEC
The unique identification given of a file that gives its physical location and generally an indication of its contents.

FORM FEED
Causes the line printer to move past the perforations in the paper to the top of a new form or page.

FORMAT
To write a pattern of data onto a disk and read it back, thus checking for errors, and also creating the disk directory. Prepares disk to be written on.

FUNCTION KEYS
Keys that are able to be programmed to perform a specific function.

HARDWARE
Physical computer equipment, including such mechanical devices as the line printer, the terminals, storage devices, and so on.

HEXADECIMAL
Counting system based on Base 16.

HEX FILE
ASCII method of representing a binary file.

HIGH ORDER BYTE
Same as most significant bit.

HOST
An information processor which provides supporting services and/or guidance to users and/or terminals and other subsidiary devices. A host processor is generally assumed to be self-sufficient, and to require no supervision from other processors.
IDENTIFIER

A sequence of alphanumeric characters that may be freely used by programmers to label elements such as program steps and assembler directives, but cannot exceed 16 characters in length.

I/O

Input/Output

JUXTAPOSITION

Side-by-side or close together.

KILOBYTE

1024 bytes (1024 character spaces).

LABEL

One or more characters used to identify a source language statement or a line in a program.

LANGUAGE

Instructions that the computer can understand.

<LF>

Line Feed

LEAST SIGNIFICANT BIT

In a BYTE, the rightmost (least significant) bit.

LOGICAL DEVICE

Device assigned to one of several peripheral devices.

LOW ORDER BYTE

Same as least significant byte.

MACRO

A single assembly-language instruction that generates a pre-defined set of machine-language instructions.

MAGNITUDE

Indicates that the value of a binary number (unsigned) is twice that of a signed (+/-) number. I.e., bit 7 as a magnitude value indicates a number from 0-255.

MEMORY

Memory is a series of physical locations into which data or instructions can be placed in the form of binary words. Each location in memory can be addressed and its contents altered.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNEMONICS</td>
<td>A technique or system of improving the memory by the use of certain formulas. Symbolic notation for a machine instruction.</td>
</tr>
<tr>
<td>MODEM</td>
<td>A device that converts data to a form that can be transmitted, as by telephone, to data processing equipment, where a similar device reconverts it.</td>
</tr>
<tr>
<td>MODULE</td>
<td>A compact assembly functioning as a component of a larger unit.</td>
</tr>
<tr>
<td>MONITOR</td>
<td>Checks or regulates the performance of the computer.</td>
</tr>
<tr>
<td>MOST SIGNIFICANT BIT</td>
<td>In a byte, the leftmost (most significant) bit.</td>
</tr>
<tr>
<td>OPERAND</td>
<td>That which is operated upon by an operator.</td>
</tr>
<tr>
<td>OPERAND FIELD</td>
<td>Contains an expression formed out of constants and labels along with arithmetic and logical operations on these elements.</td>
</tr>
<tr>
<td>OPERATORS</td>
<td>Any symbol or term indicating that a certain process, substitution, etc. is to be carried out.</td>
</tr>
<tr>
<td>OPERATING SYSTEM</td>
<td>Set of tasks that collectively automate the management of computer resources to provide efficient computer operation. It is used for user communication with the computer, for program development, and for scheduling the use of the central processing unit and its peripherals.</td>
</tr>
</tbody>
</table>
OPERATING SYSTEM TRACK
Track of the diskette containing the operating system.

OUTPUT FILE
File produced from altering an existing file.

PARAMETERS
Any constant, with variable values, used as a referent for determining other variables.

PARITY
One of several processes for data error checking.

PARITY BIT
Single bit error checking for a group of bits. Additional data which confirms a set of data. See Checksum.

PERIPHERAL DEVICE
Piece of equipment that can be used with a computer to make it more useful such as a printer, disk, modem, etc.

PHYSICAL DEVICE
Name referring to a physical device used interactively with logical devices.

PIP
Peripheral Interchange Program. Copies files or any other set of information between peripheral devices on the computer.

PORT
A connection on the CPU board which allows interface to the real world (outside the computer).

PROCEDURE
A series of operations performed in sequence to accomplish a stated purpose.

PROGRAM
A collection of one or more computer-executable procedures. In general, the execution of a program results in the execution of one main procedure and a number of subprocedures.
PROTOCOL
A set of rules governing the exchange of information between two computers, or a computer and its peripherals.

PSEUDO DEVICE
An entity treated as an I/O device by the user or system, although it is not any particular physical device. It is a forwarding address through which actual physical devices can be reached.

READ
When a task is accepting data it is said to be reading. This is a standard term.

READ/WRITE HEAD
Arm in the disk drive which reads data from, or writes data onto the disk.

RADIX INDICATORS
Denotes the base of a constant. B (binary); H (hexadecimal); D (decimal); Q or O (octal).

ROUTINE
A part of a program which performs a complete task.

SECTOR
Subdivision of a track.

SOFTWARE
The collection of tasks, procedures, rules and documentation associated with the operation of a computer system.

SOURCE DISK
Disk that has the ASCII input files to an assembler or high-level language.

SOURCE FILE
Text file containing material suitable for translation into an object module by an assembler or compiler.

STRING
A sequence of characters.

SUBROUTINE
A short set of instructions, often used repeatedly, that directs the computer in the solution of a problem.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNTAX</td>
<td>The form that a command must follow. Misspelled words are the most common syntax errors.</td>
</tr>
<tr>
<td>SYSTEM DISK</td>
<td>Disk containing the operating system programs.</td>
</tr>
<tr>
<td>TASK</td>
<td>The fundamental, executable programming unit. It may include one or more routines written for a particular purpose.</td>
</tr>
<tr>
<td>TEXT FILE</td>
<td>Those files written in ASCII code that can be read by both humans and by software.</td>
</tr>
<tr>
<td>TRACK</td>
<td>Area on a disk that can be accessed by the disk-drive without moving the read/write head.</td>
</tr>
<tr>
<td>TRANSIENT COMMAND</td>
<td>Commands that are loaded from the disk, and executed in the TPA.</td>
</tr>
<tr>
<td>TPA</td>
<td>Transient Program Area. Part of the memory dedicated to Transient Programs run from the disk via Transient commands.</td>
</tr>
<tr>
<td>UNAMBIGUOUS FILE</td>
<td>Specifies the exact file to be accessed.</td>
</tr>
<tr>
<td>REFERENCE</td>
<td></td>
</tr>
<tr>
<td>UNARY</td>
<td>1</td>
</tr>
<tr>
<td>UNSIGNED</td>
<td>Cannot be +/-, only +. Absolute value.</td>
</tr>
<tr>
<td>USER</td>
<td>A person known to the information system by means of a unique identification (name, password, etc.) who can act in one or more roles, and who has a defined set of access rights to system resources.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UTILITY</td>
<td>A general-purpose task included in an operating system to perform common functions, such as editing or file handling.</td>
</tr>
<tr>
<td>WILDCARD</td>
<td>The &quot;*&quot; symbol used in ambiguous file referencing.</td>
</tr>
<tr>
<td>WRITE</td>
<td>When a task is sending output it is said to be writing. This is a standard term.</td>
</tr>
<tr>
<td>WRITE-PROTECT NOTCH</td>
<td>Notch on the side of a diskette that may be covered with tape to prevent writing on the disk. Information may still be read from the disk, but nothing may be written on it.</td>
</tr>
<tr>
<td>Z80</td>
<td>A computer chip in the ZORBA.</td>
</tr>
</tbody>
</table>
INDEX

Application Program, 13

ASM, 33-54
program format, 35
form operand, 36
labels, 37
numeric constants, 37
reserved words, 38
string constants, 39
arithmetic operators, 39
logical operators, 39
precedence of operators, 41
operation codes, 47
error messages, 53

Assembler
directives, 42
labels, 37
operand, 36
strings, 39

Assembler Directives, 42-47
ORG,END,EQU,SET,IF
ENDDIF,DB,DW,DS

Boot
cold, 26,29,91,92,98,103,108
warm, 29,30,108

Built-in Commands, 22-26
DIR,ERA,REN,SAVE,
TYPE,USER

Commands
built-in, 22-26
transient, 22, 27-107

Command line, 19

Communications Port, 109
receive data, 109
transmit file, 109
transmit from keyboard, 109
configuration, 91-93
stop bits, 96
baud rate, 95
protocol, 96
translation, 97
port parity, 94
word width, 98

Cursor Functions, 110

DDT, 63-73
DDT Commands, 64, 66-73
DIR, 23

Directory
file, 11

Disk Drive, 6
drive assignments, 6
location, 6
switching drives, 20

Diskette, 10-11
backup, 17
care and handling, 10
copy, 17-18
format, 16
insertion, 15
protect, 11
read, 6
write, 6

Display, 5
brightness control, 5
on/off switch, 5

DUMP, 28

ED, 74-87
command strings, 77
command summary, 85
CTRL characters, commands, 84
error conditions, 83
source libraries, 82
text search, alteration, 79
enhancements, 86-87

ERA, 24

Erasing and Editing, 112

Error Flags, 33,53

Error Messages, 53

Errors, 17,33,53,83,89,101,106

Filenames, 20-21
ambiguous referencing, 21
characters not allowed, 21
primary, 20
rename, 24
Filenames, cont'd
secondary (extension), 20
wildcard, 21
with drive notations, 21

FORMAT, 105

Keyboard
control key area, 8-9
function key area, 10
setup, 6
typewriter area, 10

Language
assembly, 13
C/BASIC, 13
Z80, 13

LOAD, 54

Logical devices, 30, 57

Memory
buffer operation, 76
buffer organization, 75
mapping, 31

MOVCPM, 105

Operation Codes
arithmetic logic unit
operations, 52
control instructions, 53
data movement instructions, 51
immediate operand instructions, 49
increment/decrement instructions, 50
jumps, calls, returns, 48

PATCH, 89

PIP, 55-62

Physical devices, 30, 57

Print, 108
screen, 108
file, 108
all keystrokes, 108

Programmable Function Key, 99-103
F1-F19, 99
CTRL F1-F19, 100
CTRL keypad, 100
arrows, 101
read function key file, 103
save to a file, 103
define function key, 99-102

REN, 24
SAVE, 25
SETUP, 90-103
current port configuration, 91
cold boot command, 91
menu of available functions, 92
port configuration, 93
port parity, 94
baud rate, 95
protocol, 96
stop bits, 96
translations table, 97
word width, 98
function keys, 99-103

SGEN, 88
STAT, 28-32
SUBMIT, 104

System
attaching supplement devices, 2
operating 18, 19
unpacking and setup, 2

Tracks
allocation, 11
operating system, 18

Transient Commands, 27-107
STAT, ASM, LOAD, DDT, PIP,
ED, SGEN, SUBMIT, DUMP,
MOVCPM, PATCH, SETUP, FORMAT

TYPE, 26

USER, 26

Utilities, 12

Wildcards, 21