TRS-80™ MODEL II DEVELOPMENT SYSTEM

RACET COMPUTES
702 Palmdale, Orange CA 92665

TRS-80™ MODEL II DEVELOPMENT SYSTEM

Written For RACET COMPUTES By
T.S. JOHNSTON

For Use On The Radio Shack® TRS-80™
Model II Microcomputer

TABLE OF CONTENTS

I. GENERAL DISCUSSION

INTRODUCTION

This development package contains the necessary tools needed by a user for assembly language programming. These have been collected in a single package for the convenience of the user. These systems are major enhancements of those previously available for the Model-I system.

This package consists of three major components. Documentation for each component is provided in this manual, along with an additional booklet provided. A summary of the general capabilities available with this package is presented below:

MACASM - This is an enhanced editor/assembler, including such features as macro conditional assembly capabilities, in-memory compiles, dynamic debug facilities, and many others. Source programs can be saved on disk and subsequently loaded back into memory. A range of lines can also be loaded or saved. All the previous editor/assembler features found in the Model-I version have been retained. Uploaded source programs from the Model-I system can be assembled without change by MACASM.

SZAP - This utility provides the capability to read and modify any sector on a diskette. In addition, it provides a generalized facility for copying any number of sectors from one area (or disk) to another. This includes recovery from input/output erros, allowing backup of diskette data not possible with the BACKUP command.

DIS2 - DIS2 is a system for the disassembly of Z80 machine language code. The code to be disassembled can be from memory or from a standard DOS load module from disk. Provisions are made to allow automatic offset for each load address, restart options, and a cross reference list of referenced locations.

All the above systems have been written in assembly language for fast and convenient use.

THE AUTHORS

The three components of this system were implemented by T.S. Johnston. He has written a number of systems marketed by RACET computes. This includes the Disk Sort/Merge system (DSM), the Utility Package (XCOPY, XHIT, XGAT, DCS, XCREATE, DISKID, and DEBUG-II), and the Generalized Subroutine Facility - all available for the Model-II system.

Two components of this package were uploaded from Model-I systems. The DIS2 system is a modified version of DISASSEM provided by APPARAT. The MACASM system is a modified version of the Editor Assembler Plus from Microsoft. The purchase price of the development package includes the cost of the APPARAT DISASSEM package, as well as the Microsoft Editor Assembler Plus system. The additional booklet provided as part of MACASM was written by William Barden, Jr.

II. MACASM

INTRODUCTION

MACASM provides the Model-II interface to the Microsoft "Editor Assembler Plus" software package. This provides the Model-II user with enhanced editor/assembler capabilities including macro and dynamic debug facilities. Listed below are the important features of this package:

Contains all the convenient facilities of the earlier editor/assembler package available on the Model-I. This includes a powerful assembler as well as an in-memory edit facility. This can be used to easily build, edit, and assemble source programs.

Provides the ability to assemble directly into memory. This allows the user to repeatedly assemble and execute a machine language program without reloading the editor assembler.

Includes a powerful MACRO facility capable of automatically generating in-line code. This facility is used to define one or more groups of generalized assembler statements called "macros". The assembler will automatically insert the group of instructions when referenced by a user defined operation code. Conditional assembly of statements is also provided.

Provides the ability to save programs to a disk file and subsequently reload them into memory. Facilities are also provided to save or load portions of the source program. This is very useful for extracting or saving common program segements.

Includes a dynamic debug facility (ZBUG) which can be used to trace, inspect, and change machine language programs. ZBUG displays memory with optional user program symbolic references.

Adds many new editor commands, including substitute, move, copy, and extend commands. New line number range specification formats have also been added to the system.

Allows the user to eliminate either or both the ZBUG or assembler portions of the system if additional memory is required.

Extends the assembler expression capabilities to include multiplication, division, and many logical operators.

Included with MACASM is the "Editor Assembler Plus" instruction manual written by William Barden Jr. Additional features or differences in operation as implemented by RACET computes is described in this manual.

HOW TO GET STARTED

Included in the purchase price of MACASM is the cost of the Microsoft "Editor assember plus" and uploading service to a Model-II diskette. RACET computes also provides the patches and necessary interface to make this system operational on a Model-II system.

MACASM is shipped to the user along with several other programs on a special distribution diskette. Appendix-A outlines the procedure for transfering the contents of the distribution diskette to the users diskette. This procedure will result in a program file "MACASM", which contains both the original "Editor Assembler Plus" package, RACET patches, and interface.

MACASM is a program executed as a DOS mode command similar to COPY, DIR, FREE, etc. The user needs only enter the following command:

MACASM

This will load MACASM into memory, producing the following display:

```
  MACASM - Model-II Macro Editor Assembler -  V3.1
Model-II Interface Copyright 1980 by RACET computes

MICROSOFT EDITOR/ASSEMBLER-PLUS
COPYRIGHT (C) 1979 BY MICROSOFT
VERSION 1.01  CREATED 29-Dec-79

*_
```

The user can then enter the desired assembler commands following the "*" prompt character. These commands are described in detail in the instruction booklet written by William Barden, Jr, and in the section below. The Radio Shack TRS-80 Editor/Assembler User Instruction Manual (Catalog Number 26-2002) provides additional information and a description of the Z80 instructions.

MODEL-II IMPLEMENTATION ENHANCEMENTS AND DIFFERENCES

Most of the facilities described in the Microsoft instruction booklet are available in the MACASM implementation. The user should become familiar with the information in the instruction booklet before proceeding. This section indicates the differences between the cassette version provided by Microsoft and the RACET Model-II implementation.

The major difference between the original version and MACASM is in the use of the escape key. The documentation describes the Model-I escape key as a shifted up-arrow. The MACASM Model-II implementation uses the available escape key found on the keyboard. The "$" will still be displayed when the escape key is pressed.

The next difference is the fact that cassette tape is not supported on the Model-II system, and has been eliminated by the MACASM implementation. The tape commands are substituted by corresponding disk oriented commands. A user, therefore, can save or reload source programs using standard disk data files. Similarly, the assembled object program can be written into a standard DOS program file. The format of the commands required is discussed in detail in the next section.

A "s<u>Y</u>stem" command has been added which allows the user to issue any standard DOS command while in MACASM. This can be used to obtain directory listings, perform an INIT "I" function, list files, rename or kill files, etc.

A "<u>G</u>oto" command has also been added which can be used to transfer control to a specified address. This command can be issued directly in MACASM command mode to execute programs assembled using the in-memory option.

The two ZBUG commands to load and save system format tapes have been eliminated. They have not been replaced by corresponding disk oriented commands. The new "s<u>Y</u>stem" command allows the user to issue DUMP and LOAD commands directly, which provide similar capabilities.

Upper/lower-case has been reenabled in MACASM. However, commands, Z80 operation codes and labels must still be in uppercase. Lower-case can be used as character string arguments within quote(') marks.

The assembler listing output for the "DEFM" operation code has been modified to place 16-bytes on a line. This will produce much smaller listings than the one-byte/line format formerly used.

HOW TO SAVE AND LOAD A SOURCE PROGRAM

The write "W" and load "L" source program tape commands have been replaced by corresponding disk commands. A two line command sequence is now required. The first line indicates a write or load request. This is followed by a prompt from MACASM requesting a corresponding filespec to be used.

Two forms of the "L" command are available. The general formats of these commands are shown below:

```
L   [line1:line2]
```

or

```
LC  [line1:line2]
```

The first form "L" will clear memory of all previous source programs before starting the load operation. The "LC" operation will read a source program from disk and add it to the end of any existing source program in memory (C=concatenate).

The optional line number range refers to the line numbers within the source program being loaded. The lines loaded will **always** be loaded at the end of any existing program in memory. The user may need to renumber the entire program, if using the "LC" format, to ensure non-repeating line numbers. The special symbols "#" and "*" may be used as normal to specify the first and last lines of a source program respectively. Exact line numbers need not be specified. Lines within the range specified will be read into memory.

The general format of the "W" command is shown below:

```
W   [line1:line2]
```

The optional line number range refers to the line numbers of the source program in memory to be written to the disk file. All valid methods of indicating line number ranges can be used. This includes the new "SLN!n" and "Offset" specifications now available. Exact numbers need not be specified. All lines within the range specified will be written to the disk file.

After either a "L" or "W" command the following prompt will be issued:

```
FILESPEC? ........................
```

The user should then enter the appropriate name of the file for the load or write operation. The standard DOS "filespec" format as described in the DOS reference manual should be used.

```
************************************************************
*                                                          *
*         ------>      SPECIAL NOTE     <-----             *
*                 ---------------------                    *
*  MACASM AUTOMATICALLY PERFORMS AN INIT "I" FUNCTION      *
*              BEFORE READING OR WRITING                   *
*            TO A NEW SOURCE OR OBJECT FILE                *
*                                                          *
************************************************************
```

For a "W" command one of the following additional prompt lines is issued by MACASM:

```
FILE ALREADY EXISTS - USE IT(Y/N)? ..
```

or

```
NEW FILE REQUESTED - CREATE IT(C/N)? ..
```

The user should enter the appropriate request. The "N" response will terminate the write operation without modifying the output file specified. The "Y" or "C" response will allow the write operation to continue.

SAVING OBJECT CODE ON DISK

The "A" command initiates an assembly of the source program currently in memory. The options available are described in the Editor/Assembler-Plus manual (pages 49-55). The "AO" option specifies that an object file is to be written. This option is selected by default if none is provided.

After entering an "A" command with an explicit or implied "AO" option, MACASM will respond with the prompt:

```
FILESPEC ?........................
```

The user should then enter the appropriate DOS "filespec" where the assembled object code is to be placed. One of the following two prompt messages will then be issued by MACASM:

```
FILE ALREADY EXISTS - USE IT(Y/N)? ..
```

or

```
NEW FILE REQUESTED - CREATE IT(C/N)? ..
```

The "N" response will terminate the assembly without modifying the output file specified. The "Y" or "C" replies will allow the assembly to continue.

HOW TO USE THE "Y" COMMAND

One additional command has been added to MACASM for user convenience. The system command "Y" can be entered in command mode to execute any available DOS function. The format of the "Y" command is as follows:

```
Y dos-command and required operands
```

The DOS command, such as DIR, LIST PROG1/ASM, etc, must be in the expected format. Optional blank spaces may follow the "Y" before the start of the desired DOS command.

The system command, for example, can be used to obtain a directory listing. The following formats would be acceptable:

```
YI
Y DIR
YDIR
Y DIR :0
YDIR :1
```

Note the use of the INIT command "I" first to make sure the correct disk is initialized by DOS (this may not be needed).

Most DOS commands may be executed in the above manner. MACASM does not use the area from X'2800'-X'3000' used by some of the DOS commands. Some DOS commands, such as COPY, BACKUP, FORMAT, etc. cannot be used since they may use memory above X'3000'.

HOW TO USE THE "G" COMMAND

The "G" command allows the user to transfer from MACASM command mode directly to any address in memory. This command is similar to the "G" command available in the Z-BUG mode of operation, but is issued directly in MACASM command mode. This function is very useful for executing programs which have been assembled directly into memory using the in-memory option (/IM).

The format of the "G" command is shown below:

```
G  hex-address
```

where the hex-address is the starting location to be executed next. The user can return to MACASM by doing a "RET" instruction, or a JP 44CCH.

The user should be cautioned that the existing stack is only about 50-bytes long. A separate stack should be established if a larger area is required. It is not necessary to restore the stack to its original value when returning control back to MACASM.

EXAMPLE USE OF MACASM

The use of MACASM is relatively simple, as shown by the example below. The easiest way to understand how to use MACASM is to actually enter the sequence below. Note that all user input is shown as **boldface underlined** characters.

```
MACASM
  MACASM - Model-II Macro Editor Assembler -  V3.1
Model-II Interface Copyright 1980 by RACET computes

MICROSOFT EDITOR/ASSEMBLER-PLUS
COPYRIGHT (C) 1979 BY MICROSOFT
VERSION 1.01  CREATED 29-Dec-79
*_
```

The "MACASM" command was entered to load and execute the editor/assembler. MACASM then printed the header messages as shown above. At this point the "*" prompt was issued, and MACASM then waited for user input.

```
*I
00100          ORG      8000H
00110  START:  LD       HL,MESG
00120          LD       B,MESGX-MESG
00130          LD       C,0DH
00140          LD       A,9
00150          RST      8
00160          LD       A,36
00170          RST      8
00180  MESG:   DEFM     'This is your first program'
00190          DEFB     0DH
00200          DEFM     'Using MACASM !!!'
00210  MESGX:  EQU      $
00220          END      START
00230 break-key
*_
```

The above short assembly language source program was then entered into memory. Note that the last line was terminated by using the "break-key". This resulted in the "*" prompt requesting the next command.

```
*W
FILESPEC ? PROG1/ASM
NEW FILE REQUESTED - CREATE IT (C/N)? C
*D#:*
*L 100:110
FILESPEC ?PROG1/ASM
*P #:*
00100            ORG     8000H
00110 START:  LD       HL,MESG
*LC 120:*
FILESPEC ?PROG1/ASM
```

The above sequence wrote the source program created above into the file "PROG1/ASM". This was followed by deleting the program from memory and reloading lines 100-110 back into memory. The resulting two lines were printed as shown. The concatentate load "LC" command format was then used to load the remaining lines back into memory. This could be verified by doing another "P #:*" command if desired.

```
*A
FILESPEC ? PROG1
NEW FILE REQUESTED - CREATE IT(C/N)?C
8000           00100          ORG      8000H
8000 210D80    00110 START:   LD       HL,MESG
8003 062B      00120          LD       B,MESGX-MESG
8005 0E0D      00130          LD       C,0DH
8007 3E09      00140          LD       A,9
8009 CF        00150          RST      8
800A 3E24      00160          LD       A,36
800C CF        00170          RST      8
800D 54          00180 MESG:    DEFM     'This is your first
program'
800E 68 69 73 20 69 73 20 79 6F 75 72 20 66 69 72 73
801E 74 20 70 72 6F 67 72 61 6D
8027 0D        00190          DEFB     0DH
8028 55        00200          DEFM     'Using MACASM !!!'
8029 73 69 6E 67 20 4D 41 43 41 53 4D 20 21 21 21
8038           00210 MESGX:   EQU      $
8000           00220          END      START
00000 TOTAL ERRORS

MESG    800D
MESGX   8038
START   8000
```

The assembly was initiated by an "A" command. The "AO" option was used by default since no other options were specified. The output "filespec" for the object code was specified to be written into "PROG1". This was followed by the assembled code as shown above.

```
*B
TRSDOS READY
PROG1
This is your first program
Using MACASM !!!
TRSDOS READY
```

The "B" command was used to exit from MACASM. Entering the command "PROG1" automatically loaded and executed the above program. Next time try something just a little more useful!

III. SZAP — DISK READ AND MODIFY PROGRAM

INTRODUCTION

The purpose of SZAP is to allow the user to access, modify, copy, zero, and print any sector on a Model-II diskette. The features of SZAP include:

All tracks from 0-76 and sectors 1-26 can be read or written using SZAP.

Hexadecimal as well as ASCII representation of the data is shown for each sector.

A convenient screen editor allows the user to easily update any sector on the diskette. This includes complete cursor control and the ability to enter changes in hexadecimal or ASCII.

Sectors can be easily scanned in either a forward or backward direction. The desired starting location can also be specified.

SZAP can function on a one drive system or multiple drive system.

The current sector displayed on the screen can be printed.

A special print mode will automatically print each sector displayed on the screen.

A repeat mode automatically repeats the previous command, allowing viewing and/or printing of a series of sectors.

A powerful copy command is available allowing the user to copy any number of sectors from one location to another, or form one disk to another.

SZAP is a necessary utility for manipulating data directly on the disk. Most important is the ability to examine and modify the directory. For example, this allows the user to recover lost files. SZAP can also be used to backup a disk that the standard BACKUP utility cannot because of I/O errors. In this case SZAP will note the errors but continue.

HOW TO GET STARTED

SZAP, along with several other programs, is shipped to the user on a special distribution diskette. Appendix A. outlines the procedure for transferring the contents of the distribution diskette to a user's system diskette.

Once SZAP has been loaded it may be copied to other diskettes as required.

The user may execute SZAP as a DOS mode command similar to DIR, COPY, FREE, etc. The SZAP command has no other parameters. It is executed by simply entering the command:

SZAP

After entering the above command SZAP will print a header and a menu of commands, and then position the cursor to the CMD ? prompt. The user then enters one of the single character commands shown in parenthesis. Some commands will then request additional information, such as track/sector to display. Figure(III-1) illustrates the contents of the full screen after displaying track=1, sector=1 on drive #0.

NOW IT IS TIME FOR YOU TO TRY IT

The easiest way for you to learn how to use SZAP is to try it. Given below is a sequence of steps that you can try on your own computer. Note that user input is shown in **boldface underlined** characters.

All numeric data (track, sector, and number) MUST BE ENTERED IN HEXADECIMAL.

Select a DOS diskette that does not contain any important data (during the procedure below you might blow it!). This diskette should contain a copy of the SZAP program. Place this diskette in drive #0, boot the system if neccessary or use the INIT "I" function as required. Diskettes may be switched at any time while in SZAP. SZAP could be executed using a master DOS diskette, and then switching to the diskette that needed to be examined or changed. SZAP does not need an INIT "I" function, but this should be performed after SZAP is terminated.

SZAP

The above command should then execute SZAP. This should print the header as shown in Figure(III-1). You will notice that the cursor is located just after the CMD ? _

;
;
-
-

The first commands you should try are ";" and "-". These, you will see, allows you to go backwards and forwards through the disk. Notice that SZAP assumes that you are looking at the diskette on drive #0, starting at track=0. It thinks it is on sector=1, so when you press the first ";" it goes to sector=2.

```
T
TRACK.SECTOR:DRIVE(DISKID)?2.5
```

The next most common command is "T". After you type the "T" SZAP will respond with the prompt line requesting - "What track, sector, drive, and diskid do you want to look at next?". The example above shows the request to look at track=2, sector=5. Notice that it wasn't necessary to specify a drive or diskid. In this case it assumed the diskette already mounted in drive=0 was to be used.

```
T
TRACK.SECTOR:DRIVE(DISKID)?.9
T
TRACK.SECTOR:DRIVE(DISKID)?1
```

The above two requests show that both the track and sector can also be eliminated. In the first case your display should show the contents of track=2, sector=9. The second case should show track=1, sector=9. Notice that SZAP assumes the previous value until changed.

```
T
TRACK.SECTOR:DRIVE(DISKID)?0.1
```

When you enter the above command the data from track=0, sector=1 will be displayed. Notice that this sector contains the diskid of the diskette. Track=0 is special because it is written in single density 128-byte format. SZAP will always show the last 128-bytes as zero, and will be ignored when rewriting back to disk in the edit mode.

```
T
TRACK.SECTOR:DRIVE(DISKID)?2.1
-
;
```

When you execute the above two commands you will note that first track=2, sector=1 is displayed. Pressing the "-" to display the previous sector will then shown track=1, sector=26. Using the ";" command will similarly increment over track boundaries. SZAP will automatically change the track value when necessary. It is also VERY important to note that sector=26 is processed by SZAP. The DOS system does not normally use sector=26. All files on the disk use sectors 1-25, leaving sector=26 unused. The one exception to this is the directory track which uses sector=26.

```
S
```

If your printer was ready, the above command caused the data portion of the screen to be printed. If you have followed the sequence above, you should now have a printed listing of track=2, sector=1.

```
P
;
;
P
```

The "P" command sets SZAP into a mode to automatically print the data portion of the screen when a display changes. This mode of operation is indicated in the upper right portion of the screen. The two ";" commands will display and also print the contents of the next two sectors. Pressing the "P" again will turn off the print mode.

```
=
;
R
any-key
```

The above sequence illustrates use of the "Repeat" function. Using the "R" command continuously repeats the previously entered command until any other key is pressed. You will notice in this case that successive sectors are displayed on the screen. If you were in "P" print mode then the sectors would have also been printed.

```
T
TRACK.SECTOR:DRIVE(DISKID)?40.1
F1          (the F1 key)
-->
dwn         (down arrow)
<--
up          (up arrow)
tab         (tab key)
ent         (enter key)
/
-->
dwn
<--
up
tab
ent
esc
```

Now it is time for you to try the edit mode. The first thing to do is to position SZAP to some unused sector. The example above uses track=40, sector=1. Pressing the "F1" key will then enter the edit mode, and the cursor will be position to the first hex byte of the sector. The next six commands illustrate cursor positioning. The "/" command then sets the cursor in the ASCII portion of the display. Repeating the same six cursor commands should perform similar actions. The last command "esc" will exit from the edit mode. It will also cause the screen to be refreshed.

```
F1
0123456789ABCDEF
/
This should be ASCII data
F2
esc
```

You can see from the above sequence that it is easy to change data on the screen. Pressing the "F1" key enters edit mode. Position the cursor if necessary and enter data either in the hex area or ASCII area. Pressing the "F2" key will write the modified sector to disk. Note that pressing the "esc" key without using the "F2" will cause any changes to be ignored.

```
Z
ZERO TRACK.SECTOR:DRIVE(DISKID)#NUM?40.1#2
Z
ZERO TRACK.SECTOR:DRIVE(DISKID)#NUM?#2
Z
ZERO TRACK.SECTOR:DRIVE(DISKID)#NUM?_
-
-
-
-
-
```

The "Z" command is used to set every byte of one or more sectors to X'00'. The "Z" command will request the track, sector, drive, and diskid similar to the "T" command. The number of sectors to set to zero is also requestd. If the "#NUM" field is not entered only one sector will be set to zero. Appropriate defaults are used if not provided, as shown in the above examples. The sequence above will first set 40.1-2 to zero, followed by 40.3-4, and finally 40.5 to zero. The "-" commands are used to page back through the disk to verify that the operation has been performed. We hope the diskette you used did not contain any important information in this area (it is gone now!).

```
C
COPY-FROM TRACK.SECTOR:DRIVE(DISKID)#NUMBER?40.6#2
COPY-TO TRACK.SECTOR:DRIVE(DISKID)40.1
-
-
```

Now try the "C" command to copy sectors 40.6 and 40.7 back to 40.1 and 40.2 as shown above. Then enter the two "-" commands to verify that data was indeed copied. You will notice that two separate prompts are displayed with the "C" command. The first one tells SZAP where the data is coming from. The second one indicates where the data will be copied to. The from prompt, similar to the "Z" command, has a "#NUMBER" field to indicate how many sectors will be copied.

Break **(break key)**

Pressing the break key will return you to DOS.

Congratulations - if you followed the above sequence you have tried most of the options available in SZAP. You probably won't need to read the rest of the manual. Use of drives 1-3 and specification of the diskid were not illustrated. Common sense will dictate their use. For example, copying data from one diskette to another requires either specification of a different diskid or drive number for the from and to areas.

SZAP can, in fact, be used to backup an entire diskette to another diskette. The command required is:

```
C
COPY-FROM TRACK.SECTOR:DRIVE(DISKID)#NUMBER?0.2:
    (DISKA)#7B8
COPY-TO TRACK.SECTOR:DRIVE(DISKID)?0.2(DISKB)
```

Notice that the copy started at sector=2 on track=0 for a total of 1976 sectors (X'7B8'). The diskid00need to be different on a one drive system since diskettes will need to be switched. If you tried to copy sector=0 first then the diskid would be duplicated and SZAP would get mixed up! The last sector written in the above case will be track=0, sector=0 because SZAP will automatically wrap-around when the end of disk is reached. You could specify #7B7 if you do not want to change the diskid on the to diskette.

The above process for backing up an entire diskette will normally take longer than the standard BACKUP command. SZAP copies every sector even if it is not assigned to a file. BACKUP copies only allocated sectors. SZAP, however, will copy diskettes with bad sectors. This allows the user to at least partially recover a bad diskette. This is described in greater detail in the section "How to Recover Blown Diskettes".

SZAP COMMAND REFERENCE

Given below is a summary of all SZAP commands in alphabetical order. The examples given in the preceeding section provide the necessary additional information needed to effectively use SZAP.

Command Mode

C **Copy**

This command is used to copy any number of sectors from one location to another. SZAP will issue the following two prompts to describe the _from_ and _to_ locations:

```
COPY-TO TRACK.SECTOR:DRIVE(DISKID)#NUMBER?_
COPY-FROM TRACK.SECTOR:DRIVE(DISKID)?_
```

The track, sector, drive, diskid, and number of sectors to copy should be entered with the ".", ":", "(...)", and "#" separator characters as required. At least one of the indicated parameters needs to be entered. The defaults for track, sector, drive, and diskid is the last used value (or 1.1:0 and current diskette if SZAP has just been entered). If "#NUMBER" is not specified SZAP assumes "#1". The _from_ and _to_ diskid's need to be different on a one drive copy.

F1 **Enter or Exit Edit Mode**

This command is used either to enter or exit the edit mode of SZAP. SZAP is initially in the command mode when executed from DOS. Edit mode commands are not valid in command mode, and command mode commands are not valid in edit mode.

; **Next Sector**

This command is used to fetch and display the next sector. The current track and sector locations are used as a base. SZAP will automatically page to the next track if necessary.

- **Previous Sector**

This command is used to fetch and display the previous sector. The current track and sector locations are used as a base. SZAP will automatically page to the previous track if necessary.

P **Print Mode On or Off**

This command will either turn on or off the print mode in SZAP. Every time the display screen is changed the sector data will be printed while in print mode.

R **Repeat Mode**

This command will cause the previous ";" or "-" commands to be automatically repeated. This provides a method for scrolling through a diskette in a forward or reverse direction. This mode is terminated when any key is pressed.

S **Screen Print**

This command will cause the sector data portion of the screen display to be directed to a printer.

T **Track/Sector Display**

This command will set the current track and sector counters, fetch the corresponding sector, and display it on the screen. SZAP will request the following after the "T" is entered:

```
TRACK.SECTOR:DRIVE(DISKID)?
```

The user should respond with the appropriate track, sector, drive, or diskid. The separator characters ".", ":", and "(..)" should be used when entering the sector, drive, or diskid. At least one of the four parameters must be entered. Any parameter not specified will default to the last used value (track=1, sector=1, drive=0, current diskette when SZAP first entered).

Z **Zero Sectors**

This command will set a specified number of sectors to hexadecimal zero (X'00'). The following prompt will be issued by SZAP:

```
ZERO TRACK.SECTOR:DRIVE(DISKID)#NUM?
```

The user should respond with the appropriate track, sector, drive, or diskid. The separator characters ".", ":", "#" and "(..)" should be used when entering the sector, drive, diskid, or number of sectors to be set to zero. At least one of the five parameters must be entered. The track, sector, drive, or diskid will default to the last used value if not specified. The "#NUM" parameter will default to "#1" if not specified.

Edit Mode Commands

/ **ASCII/HEX Mode**

This command changes the cursor position to either the ASCII or hex display area. Data entered must be in the corresponding format as the cursor position.

Down **Down Arrow Key - Position Down One Line**

This command will position the cursor down the current column to the next line. The cursor will wrap around to the top of the screen if necessary.

Enter Enter Key - Down Start Next Line

This command changes the cursor position to the start of the next line down. The cursor will wrap around when the bottom of the screen is reached.

Esc Escape Key - Exit Edit Mode

This command will cause SZAP to exit the edit mode and enter the command mode. The contents of the current track and sector will be reread from the diskette. Any changes entered without subsequently pressing the "F2" key will not be written to the diskette.

Left Left Arrow Key - Position to the Left

This command will position the cursor one position to the left. If the cursor is in the <u>hex</u> area it will position to the previous hex character. If the cursor is in the <u>ASCII</u> area it will position to the previous ASCII character.

F2 Modify Sector Changes On Disk

This command will write the modified sector on the screen back to the current location on disk. Any changes made during edit mode must be explicitly rewritten back to disk for the changes to be made permanent.

Right Right Arrow Key - Position to the Right

This command will position the cursor one positon to the right. If the cursor is in the <u>hex</u> area it will position to the next character. If the cursor is in the <u>ASCII</u> area it will position to the next ASCII character.

Tab Tab Key - Tab to the Next 8-Byte Column

This command will position the cursor to the next eight-byte column, either in the <u>hex</u> or <u>ASCII</u> areas respectively. The cursor will wrap around to the top of the screen if necessary.

Up Up Arrow Key - Position Up to the Next Line

This command will position the cursor up to the next line in the same column. The cursor will wrap around to the bottom of the screen if necessary.

All numeric quantities entered in the commands above must be in <u>Hexadecimal</u> format. The only exception to this is in the ASCII edit mode, where ASCII data may be entered. All track, sector, and number of sectors to be copied or set to zero <u>MUST BE IN HEXADECIMAL</u>.

III-1 SUPERZAP EXAMPLE

```
SUPERZAP-II COPYRIGHT 1979, RACET COMPUTES    -    V2.1 3/2/80·

Command Mode: Next(;) Prev(-) Print(P) Edit(F1) Trk/Sec(T) Repeat(R)
              Screen(S) Copy(C) Zero(Z)
Edit Mode   : Mode(/) Left(<--) Right(-->) Up(UP) Down(DOWN)
              Esc(ESC) Modify(F2) Tab(TAB) Ent(ENT)
CMD ? _
0 01 01 00  01 FE 00 00 3E 24 3E 24  3E 24 3E 24 C3 45 01 FF *....>$>$ >$>$.E..*
        10  FF FF FF FF C9 FF FF FF  FF FF FF FF C9 FF FF FF *........ ........*
        20  FF FF FF FF C9 FF FF FF  FF FF FF FF C9 FF FF FF *........ ........*
        30  FF FF FF FF C9 FF FF FF  FF FF FF FF C9 FF FF FF *........ ........*
        40  FF FF FF FF 20 00 00 00  00 00 00 00 00 00 00 00 *.... ... ........*
        50  00 00 00 00 00 28 00 F8  00 00 FF FF FF FF FF FF *.....(.. ........*
        60  FF FF FF FF 00 00 A2 01  32 03 C5 D5 E5 F5 21 7A *........ 2.....!z*
        70  01 35 21 41 00 CB 5E 20  0E 21 79 01 AF BE 28 07 *.5!A..^  .!y...(.*
        80  35 20 04 3E 4F D3 EF 3E  1D 06 04 21 43 00 23 34 *5 .>O..> ...!C.#4*
        90  BE 30 06 36 00 3E 3B 10  F5 DB FE 21 40 00 CB 7E *.0.6.>;. ...!@..~*
        A0  28 29 DB FF CB 7F 28 23  DB FC 32 D0 03 CB F6 FE *()...(# ..2.....*
        B0  03 20 18 CB B6 CB 66 28  0C 23 23 CB C6 2A D9 03 *. ....f( .##..*..*
        C0  22 80 01 18 06 21 D4 03  E5 18 7C 21 44 00 AF BE *"....!.. ..|!D...*
        D0  20 71 21 40 00 CB 4E 28  3C E5 06 03 11 47 00 21 * q!@..N( <....G.!*
        E0  41 F8 3A 55 00 F6 80 D3  FF 36 14 23 EB 78 32 04 *A.:U.... .6.#.x2.*
        F0  01 7E 06 30 D6 0A 38 03  04 18 F9 EB 70 23 C6 3A *.~.0..8. ....p#.:*
```

IV. DIS2 - DISASSEMBLER

INTRODUCTION

DIS2 is a system for the disassembly of Z80 machine language code. The features of DIS2 are essentially identical to the original version distributed by Apparat. These features include:

Disassembles Z80 machine language code into standard mnemonics.

Disassembles code either directly from memory, or from a standard DOS load module previously saved on disk.

Provides an option to automatically offset each load address of the machine code being disassembled. This allows a module to be disassembled which resides in one location, but actually executes from another location.

Provides a restart option which allows a large disassembly to begin at a specified location.

Produces a cross reference list of referenced locations. This is useful in determining which areas of a program are being referenced by other locations.

Allows the reference table to be written to a file. This can then be processed by other programs.

DIS2 is executed as a standard DOS command followed by additional sub-commands as required.

HOW TO GET STARTED

Included in the purchase price of DIS2 is the cost of the Apparat DISASSEM system and uploading service to a Model-II diskette. RACET computes also provides the patches and necessary interface to make this system operational on a Model-II system.

DIS2 is shipped to the user along with several other programs on a special distribution diskette. Appendix-A outlines the procedure for transferring the contents of the distribution diskette to the users diskette. This procedure will result in the program file "DIS2", which contains both the original "DISASSEM" system, RACET patches, and interface.

DIS2 is a program executed as a DOS mode command similar to COPY, DIR, FREE, etc. The user needs only enter the following command:

```
DIS2
```

This will load DIS2 into memory, producing the following display:

```
APPARAT DISASSEMBLER 2.0

OBJECT FROM MAIN MEMORY OR DISK?_
```

The user then respondes with the appropriate response(M or D). This is followed by other prompts by DIS2 as described in the next section.

DIS2 SUB-COMMANDS

A series of prompts will be issued by DIS2 when executed. These prompts solicit from the user the parameters to be used for the disassembly. The prompt messages and the possible replies are described below.

A. OBJECT FROM MAIN MEMORY OR DISK?...

Purpose: To define the location of the machine code to be disassembled.

Responses available: none, D, or M

1. none - Same as 'D' below.

2. 'D' - Disk module to be disassembled. A separate prompt will be issued for the filespec of the module to be disassembled.

3. 'M' - Module in memory to be disassembled.

B. OBJECT VIRTUAL BASE ADDRESS (HEX)? ...

Purpose: To specify a value where the object code is considered to be executed from. This prompt is issued only for an 'M' response to item-A above. This is necessary when machine code in one location of memory actually is executed from a different location.

Responses available: None or four-digit hex value. If none (just an ENTER) is specified then then the real base address is used.

C. FILESPEC? ...

Purpose: To solicit the name of the file containing the object module to be processed.

Responses available: Filename in standard DOS "filespec" format.

D. OFFSET OBJECT VIRTUAL ADDRESSES BY (HEX)? ...

Purpose: To specify a value which will be added to each load address before processing by DIS2. This parameter is needed when an object module loads to one loaction in main memory, but actually executes from another location.

Responses available: None or a four-digit hex value. Wrap-around is allowed to effectively provide a method for subtracting. The value '0000' is assumed if nothing is entered.

E. OBJECT REAL BASE ADDRESS (HEX)? ...

Purpose: To specify the absolute location in memory where the machine code to be disassembled will be located. This prompt is issued for the "M" in-memory disassembly option (Item-A above).

Responses Available: 1-4 hex digit location value where DIS2 will actually find the object code to be disassembled.

F. OUTPUT TO LINE PRINTER ? ...

Purpose: To specify if a listing of the disassembled code is also to be printed. All disassembled code is displayed on the screen. The cross reference table is printed only if the machine code being disassembled is from a disk file ('D' option from A. above).

Responses Available: none, Y

none (just an ENTER) - If nothing is specified then no printed listing will be produced.

Y - A "Y" response will cause a printed listing to be produced. This will be followed by a "SET PRINTER TO TOP OF PAGE" prompt, allowing the user to adjust the paper accordingly.

G. NORMAL DISPLAY PAUSES? ...

Purpose: To specify to DIS2 whether the display of disassembled code is stop after every full screen. This allows the user a chance to look at the disassembled code before proceding. The "hold" key can also be used at any time to temporarily halt the display.

Responses available: none, Y

none (just an ENTER) - The display of disassembled code will continue until done, the "X" key is pressed to terminate the function, or the "hold" key is pressed to temporarily halt the display. In the latter case pressing the "enter" key will resume the display.

Y - The display of disassembled code will proceed until the screen is full, the "X" key is pressed to terminate the funciton, or the "hold" key is pressed to temporarily halt the display. The display can be resumed in the first and last cases by pressing the "enter" key.

H. ANY OPTIONS ?...

Purpose: The purpose of this prompt is to provide additional options available only if the machine code being disassembled is from a disk file.

Responses available: none, NIP, RTD, REA, RE&, RIA, RI&

1. none (just an ENTER) - No more options.

2. NIP - Do not print or display the disassembled instructions. This does not affect the display of the location cross reference table.

3. RTD - Location reference table is to be stored on disk. After the location cross reference pass of DIS2, the program will query 'REFERENCE TABLE FILESPEC?'. The user should respond with a standard DOS format "filespec" where the cross reference table is to be stored. The format of the cross reference table is described later.

4. REA - Enable listing of all types of references. This is the default condition.

5. RE& - Enable listing of the specified reference type, where "&" is one of the following: L, P, R, S, T, U, V, W, or X. Reference types are defined at the beginning of each location table listing.

6. RIA - Disable list of all types of references.

7. RI& - Disable listing of the specified reference type, where "&" is one of the following: L, P, R, S, T, U, V, W, or X.

OPERATION OF DIS2

DIS2 operates in the following phases:

1. (Object from disk). Build location reference table. If insufficient memory is available, "INSUFFICIENT MEMORY' will be printed and the disassembly terminated.

2. Write reference table file only if the "RTD" option was specified (See H. above).

3. Display and optionally print the disassembled instructions. If specified, display pauses will occur.

4. Print the location reference table only if the object code is from a disk file and the printer use was specified.

If the DOS operating system detects a disk or other error, then an appropriate message will be displayed and the disassembly terminated. The 'DISK OBJECT FILE FORMAT NOT AS EXPECTED' error message will be displayed if DIS2 finds something wrong with the object module format.

While instruction displaying or printing is in process, holding down the "X" or "hold" keys will cause DIS2 to terminate the disassembly, or temporarily pause respectively. The "enter" key can be pressed to continue a disassembly. The "break" key can be used to return to DOS at any time.

Suffixed to each reference location value is a reference type code (defined at the top of each reference listing), giving the type of Z-80 instruction making the reference.

Column-1 of the disassembled instruction print line indicates the number of references to bytes of the instructions. This value is hexadecimal, with "F" meaning fifteen or more references.

Column-2 of the print line indicates which bytes of the instruction have been referenced. If this column is blank and column-1 is non-blank, then only the first byte of the instruction has been referenced. Otherwise, the hex digit represents a 4-bit binary mask indicating which relative byte has been referenced.

FORMAT OF THE REFERENCE TABLE

The "RTD" option will write the reference table created by DIS2 to a specified disk file. The format of this files is as follows:

1. 1-byte = X'CO'. Backward EOF - ignore it.

2. 1 or more entries of the form:

   a. 'Location' low value byte.
   b. 'Location' high value byte.
   c. Control byte, Bits 7-0 (7 is leftmost).

7-6 =11	Dummy last entry in table. Ignore all other bits and bytes of entry.
7-6 =01 5-6 =00	Reference entry: 'Location' referenced by one or more of the following entries.
7-6 =00 5-6 =00-1F	Reference entry: The instruction at this 'Location' referenced 'Location' in the previous reference entry. Bits 5-6 contain a code indicating the type of instruction making the reference: 0=S, 1=T, 2=U, 3=V, 4=W, 5=X, 6-7=none, 8=P, 9=L, A=R, and B-1F=none. See a reference table listing for definitions of above.

APPENDIX A.

DEVELOPMENT PACKAGE DISTRIBUTION DISKETTE

INTRODUCTION

The development package consists of a number of independent programs. These programs are contained on a special distribution diskette in a non-standard format. They must first be transferred to a DOS diskette before they can be used.

```
**************************************************
*                                                *
*        THE DISTRIBUTION DISKETTE IS NOT A      *
*              STANDARD DOS DISKETTE !           *
*     NO ENTRIES WILL BE FOUND IN THE DIRECTORY  *
*                                                *
**************************************************
```

The purpose of this appendix is to describe the procedures required to accomplish this task.

LOADING PROCEDURES

The following procedure should be used to transfer the contents of the distribution diskette to a users DOS diskette. Only drive #0 will be used in the sequence below. Note that all user input is shown below in **boldface underlined** characters.

A. - The development programs will be transferred to a standard DOS diskette. The DOS diskette selected for this purpose should contain all the DOS system modules so that it can be used in drive #0. This diskette should have about 25-granules (125 sectors) of available free space.

B. - Turn the power on if it is not already on.

C. - Insert the development package distribution diskette in drive #0. Toggle the reset switch if the power was already on before Step B. above.

D. - Wait for about 15 seconds. The following message should then be displayed at the top of the screen:

```
RACET computes BOOT System Loaded - Release  6/11/80 -
        REBOOT DOS!!!
```

The release date will change based upon the order date of the product.

E. - Remove the distribution diskette from drive #0

G. - Toggle the reset switch and enter time and date as required to get to TRSDOS READY status.

H. - Enter the following, using an "enter" key after each entry except where specifically noted:

```
DEBUG ON
DEBUG
J              (no "enter" key)
F000           (F zero zero zero zero)
```

I. - This should result in the sequence shown below. When requested you should mount the RACET distribution diskette or your DOS diskette, and then press the enter key.

```
RACET computes Distribution System - Copyright 1980
MOUNT DISKETTE * RACET  IN DRIVE #0_
MOUNT DISKETTE * DOS    IN DRIVE #0_
*CAT LOADED OK
*MACASM LOADED OK
*OPCODE/ASM LOADED OK
*SZAP LOADED OK
*DIS2 LOADED OK
PROCESSING COMPLETED
```

J. - The debug system can then be turned off by entering:

<u>DEBUG OFF</u>

After the above sequence has been completed successfully the development programs should be on your DOS diskette. This can be verified by entering a "DIR" command.

All of the development programs can be immediately used by entering the commands as described in this manual. They can also be copied to other diskettes as required.