TRS-80 EDITOR/ASSEMBLER

OPERATION
AND
REFERENCE MANUAL

TABLE OF CONTENTS

Introduction

The TRS-80 Editor/Assembler is a RAM-resident text editor and assembler for the TRS-80 microcomputer system. The Editor/Assembler was designed to provide the ease of use required by the novice, while providing capabilities powerful enough for the expert. LEVEL II BASIC is capable of directly loading the Editor/Assembler cassette tape. LEVEL I BASIC must read-in the Editor/Assembler using the SYSTEM tape (included).

The text editing features of the Editor/Assembler facilitate the manipulation of alphanumeric text files. The most common use of the editing capability is in the creation and maintenance of assembly language source programs.

The assembler portion of the Editor/Assembler facilitates the translation of symbolic language source programs into machine executable code. This object code may then be executed with the SYSTEM tape for LEVEL I BASIC or directly with the SYSTEM command under LEVEL II BASIC. Previous knowledge of machine language and the hexidecimal number system is assumed throughout this manual.

The Assemble command (A) supports the assembler language specifications set forth in the Zilog Z80-Assembly Language Program Manual, 3.0 D.S., REL.2.1, FEB 1977, with the following exceptions.

Macros are not supported.

Operand expressions may only contain the + and −, & (logical AND), and 〈 (shift) operators, and are evaluated on a strictly left to right basis. Parentheses are not allowed!

Conditional assembly commands, where a programmer may control which portions of the source code are assembled, are not supported.

Constants may only be decimal (D), hexidecimal (H), or octal (O). See section under operands.

The only Assembler commands supported are *LIST OFF and *LIST ON.

A label can contain only alphanumeric characters. (Use of the – and ? is not supported.) A label can be up to 6 characters long. The first character must be alphabetic. The other characters must be alphanumeric.

NOTATION CONVENTIONS

[ ]	Square brackets enclose optional information: P[line1[:line2]] The :line2 is optional, and the P need not be followed by anything at all since all options following P are enclosed in brackets. The brackets are never actually typed.
. . .	The ellipses represent repetition of a previous item: A[[ɓ filename] [/switch[/switch]. . .]] The /switch may be repeated several times.
CAPITALS	Capital letters must be as shown for input, and will be as shown in examples of output.
lowercase	The user must substitute in his own values (eg: inc. filename, line)
underscore	Underscored information is output printed by the Editor/Assembler unless specified otherwise. This distinguishes user input from computer output but is never actually typed by the user.
ɓ	A lowercase B with slash specifies a mandatory blank(space).
line	Any decimal number from Ø to 65529
line1:line2	Numbers specify two different line numbers (line #1 is usually less than line #2)
•	A period may be used in place of any line number. It represents a pointer to the current line of source code being assembled, printed, or edited.
#	A pound sign may be used in place of any line number. It represents the first (lowest line number) source code line in the text buffer.
*	An asterisk may be used in place of any line number. It represents the last (highest line number) source code line in the text buffer.
inc	A number representing an increment between successive line numbers.
filename	A character string specifying the name of a cassette file. See section on Cassette Tapes.

Editor/Assembler

In brief the Editor/Assembler is designed for a user to type in source assembler code. This source code is assembled and the resulting object code may be recorded onto tape. The Editor/Assembler may also read-in, record, and edit other source code files stored on tape. Of course, the source files manipulated by the Editor/Assembler need not be assembly programs only. The files may be any text information created by the Editor/Assembler. BASIC program tapes may NOT be edited by the Editor/Assembler.

The limit to the size of an assembly language program is the amount of RAM memory in the user's computer system. The Editor/Assembler maintains a "text buffer." This buffer starts at the end of the Editor/Assembler program and continues to the end of memory. This usually leaves around 7K of memory for the text buffer which will contain the source file.

LOADING

LEVEL II BASIC

Since the Editor/Assembler is a machine language program, it may only be loaded using the SYSTEM command. Place the Editor/Assembler tape into the cassette recorder and depress PLAY. The volume should be set to 5 or 6 (this is a 500 baud tape).

Type SYSTEM and then press ENTER. The computer will respond by typing:

```
*?
```

Now type EDTASM, the filename of the Editor/Assembler, and the tape will be read into memory. Once loading is completed, type a / (slash) and press ENTER, the monitor screen is clear and the message:

```
TRS-80 EDITOR/ASSEMBLER 1.1

*
```

is printed. The asterisk is the Editor/Assembler prompt symbol. This is its way of requesting a command. Depressing the BREAK key will always return you to an asterisk except when reading a tape, writing a tape, or editing a line. The BREAK key may be used to abort an assembly or a printout in progress.

LEVEL I BASIC

Since the Editor/Assembler is recorded on tape at 500 baud, LEVEL I BASIC CAN NOT DIRECTLY read-in the tape. You must first load the SYSTEM tape provided. This program can then read-in the 500 baud Editor/Assembler tape.

Load the SYSTEM tape into the cassette recorder. Set volume to 8 or 9 (this is a 250 baud tape). Type CLOAD and BASIC I will read-in the SYSTEM tape. The program will start as soon as loading is finished.

The computer will type:

```
*
```

Now load your cassette with the Editor/Assembler tape. Set volume to 5-6 (this is a 500 baud tape). Type EDTASM and press ENTER. The Editor/Assembler will be read-in. When the reading is complete, another * will be typed. Now type a slash (/) and then the number 18058. Press ENTER to execute the Editor/Assembler. The number 18058 is the entry address of the Editor/Assembler.

```
TRS-80 EDITOR/ASSEMBLER 1.0

*
```

You may now use the Editor/Assembler as described under the section on Assembly Language.

The BREAK key works the same way as described in the third paragraph of this section.

COMMANDS

The TRS-80 Editor/Assembler can perform the following commands. These commands may be typed after the prompt symbol * where applicable. The asterisk indicates the "command level" of the Editor/Assembler. The following list contains all command level instructions recognized by the Editor/Assembler with a brief description of each.

A	Assemble source currently in text buffer
B	Return to BASIC in ROM
D	Delete specified line(s)
E	Edit a specified command; almost exactly like LEVEL II BASIC's EDIT command
F	Find a specified string of characters in the text buffer
H	Same as P command except that output goes to lineprinter
I	Insert source line(s) at a specified line with a specified increment
L	Load a source file from cassette tape into text buffer
N	Renumber source lines in the text buffer
P	Print specified range of source code currently in the text buffer
R	Replace lines currently in text buffer. Like the Insert command only lines are overwritten
T	Same as H only no line numbers are printed – text only.
↑ or ↓	Scroll up or down. Will print the next or previous source line
→	Horizontal tab
W	Write current text buffer onto tape

Assemble (A)

form: *A[[filename] [/switch[/switch] . . .]]

switch may be any of the following four options

NL	No listing written to screen. Errors and bad source lines are still typed.
NO	No object code. Inhibits recording of an object code tape.

NS	No symbol table is to be printed
LP	Send listing, errors, and symbol table to the TRS-80 LINEPRINTER
WE	Cause assembly to wait when an error occurs. Depressing any key will continue assembly until another error is found. Depressing the "C" key will cause assembly to continue without stopping for errors. Pressing BREAK returns to command level at any time.

The contents of the edit buffer are assembled. The object code is written to cassette tape under the specified filename (if no filename is specified the filename is automatically set to NONAME.) An assembly error is usually written to the monitor screen immediately before the line the error occurred on.

After the assembly is completed the total number of errors is printed. Finally, the symbol table is printed. The computer then types:

READY CASSETTE

Prepare your object tape for recording and press ENTER. If you don't want the object code, simply press BREAK and an asterisk (command level) will be returned to you. This is the default procedure which may be altered with the proper switches.

Examples:

*A	Assemble with filename of NONAME; list on screen
*Aϕ IKKY	Same as above; object file is IKKY
*A/NS	Assemble with filename of NONAME, no symbol table
*A/NS/LP	Same as above yet all output is to line-printer
*Aϕ Q/NL	Assemble with filename Q; no listing ϕ is a mandatory blank

Basic (B)

form: *B

Typing a B and then ENTER will return you to a MEMORY SIZE (power up) condition in LEVEL II BASIC or a READY state in LEVEL I BASIC.

Example:

*B

MEMORY SIZE?

Delete (D)

form: *D[line1[:line2]]

Deletes the line or lines specified from the text buffer.

Examples:

*D1ØØ:5ØØ	Deletes lines 1ØØ through 5ØØ (inclusive) from the text buffer
D#:	Deletes entire text buffer. Clears text buffer
*D.	Deletes line currently pointed to by period (.).
*D1Ø5	Deletes the single line 1Ø5

Edit (E)

form: *E[line]

Allows user to edit/modify source lines just like the EDIT command in LEVEL II BASIC. The only difference is that the Delete command does not enclose deleted information in exclamation points(!).

Examples:

*E.	Edits current line pointed to by period (.).
*E211	Edit line 211

Sub-commands for Edit are A,C,D,E,H,I,K,L,Q,S,X.

Edit Subcommands

A	Restart edit
nC	Change n characters
nD	Delete n characters
E	End editing and enter changes
H	Delete remainder of line and insert string. The H command should not be used to delete an entire line of text. There must always be at least one character on a line, or future use of that line will cause problems.
I	Insert string
nKx	Kill all characters up to the nth occurrence of X
L	Print the rest of the line and go back to starting position
Q	Quit and ignore all editing
nSx	Search for the nth occurrence of X
X	Move to the end of the line and insert
Backspace	Move edit pointer back one space
(SHIFT) (↑)	Escape from any edit mode subcommand
ENTER	ENTER the line in its present (edited) form

The user should experiment with these or refer to the LEVEL II BASIC Manual.

Find (F)

form: *F[string]

where string is a sequence of 16 characters or less

The edit buffer is searched starting at .+1 for the first occurrence of the specified string. If no string is specified, the search is the same as that of the last F command in which a string was specified. If the search string is found the line containing it is printed and period (.) is updated to the printed line. If the string is not found STRING NOT FOUND is printed and period (.) remains unchanged. P# is often used to move period (.) to the beginning of the buffer prior to a search.
Example:

```
*P#
00100              ORG          7000H
*F3C00
00100 VIDEO        ORG          3C00H
*F
00211              LD           HL,3C00H
*
```

Hardcopy (H)

form: *H[line1[:line2]]

Prints a line or group of lines onto the TRS-80 LINEPRINTER. Period (.) is updated to point to the last line printed. This command is exactly like the P command.

Example:

H#:	Sends all lines in the text buffer to printer
*H100:500	Sends lines 100 through 500 to printer
*H.	Send current line pointed to by period (.) to the lineprinter.
*H	Prints 15 lines starting with the current line to the printer. Not very useful for lineprinter use.

Insert (I)

form: * I line [,inc]

The I command is used to insert lines of text into the edit buffer. All lines of source are usually entered with the I command. After the I command is issued, line numbers are generated and lines of text are inserted into the edit buffer until one of the following conditions occurs:

a BREAK is typed (usually way to exit)

the edit buffer is full

The line number of the next line to be inserted would be greater than or equal to the next exit line in the buffer. The NO ROOM BETWEEN LINES message is typed.

The line number of the next line to be inserted would be greater than 65529.

If inc is not specified it is assumed to be the last specified value. Period (.) is updated to point to the last line inserted. See section, Sample Use of the I command.

Note: Source lines may be up to 128 characters long. This size line is usually not needed. It is recommended that you use lines of approximately 60 characters each (printout and listings will be neater).

Load (L)

form: *L[bfilename]

The tape is searched for the file specified by filename. If the specified file is found, its contents are added to the current contents of the edit buffer. Note that this may result in improperly sequenced line numbers which must be corrected by use of the N command for proper operation. If the user does not wish to add to the current text buffer, then the buffer must be cleared by the D#:* command.

If filename is not given, the next file on the tape is loaded.

When reading, the familiar asterisks will flash in the upper right corner of the screen. The L command can only read source files created by the Editor/Assembler.

Example:

*L	Loads next source file
*LbMYPROG	Searches for and loads source file named MYPROG. b is a mandatory blank

Number (N)

form: *N[line[,inc]]

The N command is used to renumber the lines in the edit buffer. The first line in the buffer is assigned the number specified or the default 0100 if line is not specified. The remaining lines in the buffer are renumbered according to the increment (inc) or the previous inc in an N,R, or I command if inc was not specified. Period (.) points to the same line it did before the N command was used, but the number of this line may be changed.

Examples:

*N	Renumbers from 100 with previous increment
*N5	Renumbers from 5 with previous increment
*N10,5	Renumber from 10 in steps of 5

Print (P)

form: *P[line1[:line2]]

Prints a line or group of lines on the monitor screen. Period (.) is updated to point to the last line printed.

Example:

P=:	Prints all lines in the text buffer
*P100:500	Prints lines 100 through 500 inclusive
*P.	Prints current line pointed to by period (.)
*P	Prints 15 lines starting with the current line. Repeated use of P allows printout of source without lines being scrolled off the screen

Replace (R)

form: *R[line[,inc]]

The R command only replaces one line and goes into insert mode. If line exists, it is deleted then inserted. If line doesn't exist it is inserted as with the I command. If inc is not specified, the last inc specified by an I, R or N command is used. Period (.) is always updated to the current line.

Example:

*R.	Replace current line
*R100,10	Start replacing lines beginning at line 100 and incrementing with 10.
*R100	Start replacing at line 100 using last specified increments.

Type (T)

form: *T[line1[:line2]]

Prints a line or group of lines onto the TRS-80 LINE PRINTER. Period (.) is updated to point to the last line printed. This command is much like the H command, only no line numbers are printed. Only the source text is printed.

Example:

T=:	Sends all lines in the text buffer to printer
*T100:500	Sends text for lines 100 through 500 to printer
*T.	Sends current line pointed to by period (.) to the lineprinter.

Scroll and Tab

The Editor/Assembler recognizes the following special characters:

Scroll up

The ↑ command prints the line preceding the current line and updates period (.) to the printed line. (If the current line is the first line in the edit buffer, it is printed and period (.) remains unchanged.)

Scroll down

The ↓ command prints the line following the current line and updates period (.) to point to the printed line. (If the current line is the last line in the buffer, it is printed and period (.) remains unchanged.)

Note: Both ↑ and ↓ must be the first character of the command line or they will be ignored.

Tab

Typing → tabs right to the next 8 character field. Calling the first position of a source line 1 (line number not counted), the tabs are at positions 9,17,25,33,41,49,51 and continue on in increments of 8 up to 255. Tabs should always be used instead of spaces to conserve text buffer space. A tab (09 hex) only takes up one byte.

Delete character

Backarrow (←) will delete the last character typed. If the last character was a tab, the cursor jumps backwards to the next non-blank character.

(Shift ←) Delete Line

A (Shift ←) will delete all of the line currently being entered. This is true for both source lines and commands.

(Shift @) Pause

At any time during an Assembly or printout a (Shift @) may be typed to halt the computer. Pressing ENTER will continue the process. The (Shift @) will not be accepted while a line is being printed or assembled; only between lines. A pause received while assembling will not be recognized

```
TEXT DEFM 'TRS-80 MICROCOMPUTER'
```

while bytes of the text string are being assembled. Another pause must be typed after this line is finished being assembled.

Write (W)

form: *W[␣filename]

The contents of the edit buffer are written onto a cassette file under the name filename. If filename is not specified no file name is used. Period (.) is always left unchanged.

Example:

*W	Records text buffer to tape with no file-name
*W␣DEMO	Records text buffer to tape with a file-name of DEMO. ␣ is a mandatory blank.

Cassette Tapes

All cassette tapes created by the Editor/Assembler are written at 500 baud. The cassette tape containing the Editor/Assembler is also at 500 baud. Whenever reading a 500 baud tape the VOLUME LEVEL MUST BE BETWEEN 5 AND 6.

The SYSTEM tape, included with the Editor/Assembler, allows LEVEL I BASIC to read-in the 500 baud Editor/Assembler tape. First read-in the 250 baud SYSTEM tape (with volume at 8 to 9), and then load in the Editor/Assembler (at volune 5 to 6) as specified in section on Loading.

LEVEL II BASIC may directly read-in the 500 baud Editor/Assembler tape.

Execution of object code programs stored on tapes is performed with the SYSTEM command in LEVEL II BASIC. LEVEL I BASIC must again use the SYSTEM tape to read-in and then execute object code from a 500 baud tape. Examples of creating object code and then executing it are given in section on Sample Use.

Filenames

Cassette filenames must begin with an alphabetic character. The remaining characters must be alphanumeric. The length may not exceed 6 characters. Filenames need not be specified for the A or W commands and in the event that a name is not specified, the file is assigned the NONAME filename. If a filename is not specified when using the L command, the first file encountered on the tape is loaded into memory.

Sample Use

The following is a sample session using the Editor/Assembler to write a program. Comments to the reader are enclosed in [ ] and are not part of the program.

```
TRS-80 EDITOR/ASSEMBLER

*I100,10

00100   [→]   ORG     5000H              [→IS A TAB]
00110   VIDEO EQU     3C00H
00120         LD      HL,VIDEO      ;SOURCE ADDRESS
00130         LD      DE,VIDEO+1    ;DEST. ADDRESS
00140         LD      BC,400H       ;BYTE COUNT
00150         LD      (HL),0BFH     ;GRAPHICS BYTE
00160         LDIR                  ;WHITE OUT SCREEN
00170   ;DELAY LOOP TO KEEP WHITE OUT SCREEN ON
00180         LD      B,5
00190   LP1   LD      HL,0FFFFH     :VALUE TO DECREMENT
00200   LP2   DEC     HL
00210         LD      A,H
00220         OR      L             ;HL=0?
00230         JP      NZ,LP2        ;IF NO DEC AGAIN
00240         DJNZ    LP1           ;DEC.B--B=0?
00250         JP      0H            ;RETURN TO BASIC
00260         END
00270 [BREAK]
```

*A XXX [Assemble] [All the following is computer output]

```
5000            00100     ORG          5000H

3C00            00110     EQU          3C00H

5000 21003C     00120     LD           HL,VIDEO    ;SOURCE ADDRESS

5003 11013C     00140     LD           DE,VIDEO+1  ;DEST. ADDRESS

5006 010004     00040     LD           BC,400H     ;BYTE COUNT

5009 36BF       00150     LD           (HL),0BFH   ;GRAPHICS BYTE

500B ED80       00160     LDIR                     ;WHITE OUT SCREEN

                00170 ;DELAY LOOP TO KEEP WHITED OUT SCREEN ON

500D 0605       00180     LD           B,5

500F 21FFFF     00190 LP1 LD           HL,0FFFFH   ;VALUE TO DECREMENT

5012 2B         00200 LP2 DEC          HL

5013 7C         00210     LD           A,H

5014 B5         00?20     OR           L           ;HL=0?

5015 C21250     00230     JP           NZ,LP2      ;IF NO DEC AGAIN

5018 10F5       00240     DJNZ         LP1         ;DEC.B--B=0?

501A C30000     00250     JP           0H          ;RETURN TO BASIC

0000            00260     END

00000  TOTAL ERRORS

LP2   5012                    [Symbol table]

LP1   500F

VIDEO  3C00

READY CASSETTE          [Load tape; set to RECORD]

[ENTER]                 [Press ENTER to record object code]

*
```

Now you can save the information in the text buffer (YOUR SOURCE PROGRAM) onto another tape.

*W MYPROG

The tape file MYPROG may be read in by the Editor/ Assembler's L command.

Any assembler errors are printed immediately before the line the error occurred in.

Execution in LEVEL I BASIC

First load the SYSTEM tape (included with your Editor/ Assembler). Put the SYSTEM tape into your cassette. Be sure volume is between 8 and 9. Type CLOAD, to load in the SYSTEM tape. The program will execute as soon as loading is completed and will type:

*

Now enter the filename of your object tape, which was created by the Editor/Assembler. Note that you must

use the filename NONAME if a filename was not specified. With the example program type XXX, the filename of the object tape.

```
*  XXX
```

At this point put the object tape XXX into the cassette recorder and press PLAY. The volume must be at 5 to 6 (this is a 5ØØ baud tape). Asterisks will flash in the upper right screen corner. Once loading is complete the computer will type * again. Now you must enter the starting address of the machine code program. The starting address (ORG) was 5ØØØH which is a decimal 2Ø48Ø. Now type this decimal number preceded with a slash (/). The command looks like this:

```
*  /2Ø48Ø
```

Press ENTER, of course, and the machine code program will execute. The sample program will white-out the video screen with solid graphics characters. This will stay on the screen for about 5 seconds. The program will then return to a READY condition in BASIC.

Executing in LEVEL II BASIC

Execution is much simpler in LEVEL II BASIC. The object tape is again loaded at 5 to 6 volume (as are all 5ØØ baud tapes). The typing is as follows; comments are in brackets[ ]:

```
READY

> SYSTEM

*? XXX          [read in object tape]

*? /2Ø48Ø
```

The program will now execute and then return to a power up condition (ENTER MEMORY SIZE?).

Multiple Modules

You may load several machine language programs into memory, one after the other. The ORG addresses of these instructions must be such that each object program does not conflict with other modules. If you have the following files:

XXX	7ØØØ to 7ØFF hexidecimal
YYY	71ØØ to 71FF hexidecimal
ZZZ	72ØØ to 72FF hexidecimal

You may then enter the three programs as follows:

```
*? XXX

*? YYY

*? ZZZ

*? /28672       [jump to XXX program]
```

ASSEMBLY LANGUAGE

Syntax

The basic format of an assembly command is:

[LABEL] OPCODE [OPERAND(S)] [COMMENT]

Examples:

```
          ORG     7ØØØH

VIDEO     EQU     3CØØH

          LD      DE,VIDEO+1 ;DESTINATION
```

LABELS

A label is a symbolic name of a line code. Labels are always optional. A label is a string of characters no greater than 6 characters. The first character must be a letter. A label may not contain the $ character. $ is reserved for the value of the reference counter of the current instruction.

The following labels are reserved for referring to registers only and may not be used for other purposes: A,B,C,D,E,H,L,I,R, IX,IY,SP,PC,AF,BC,DE, and HL.

The following 8 labels are reserved for branching conditions and may not be used for other purposes (these conditions apply to status flags):

FLAG	CONDITION SET	CONDITION NOT SET
Carry	C	NC
Zero	Z	NZ
Sign	M(minus)	P(plus)
Parity	PE(even)	PO(odd)

Example: JP NZ, LOOP

If the zero flag is clear (not set), the above instruction jumps to the instruction specified by LOOP.

OPCODES

The opcodes for the TRS-80 Editor/Assembly exactly correspond to those in the Z-80-Assembly Language Programming Manual, 3.0 D.S., REL. 2.1, FEB 1977. See section Index to Instruction Set for the instruction in question.

OPERANDS

Operands are always one or two values separated by commas. Some instructions require no operands at all.

Examples:

```
LD        HL, 3CØØH
```

```
XOR     A

LD      (HL), 20H
```

A value in parentheses ( ) specifies indirect addressing when used with registers, or "contents of" otherwise.

Constants may end in any of the following letters:

H – hexidecimal

D – decimal

O – octal

A constant not followed by one of these letters is assumed to be a decimal. A constant must begin with a digit. Thus FFH is illegal, while ØFFH is legal.

Expressions using the +, –, &, operations are described in section, Expressions.

COMMENTS

All comments must begin with a semicolon (;). If a source line starts with a semicolon in column 1 of the line, the entire line is a comment.

Expressions

A value of an operand may be an expression consisting of +,–,&, or 〈 symbols. These operations are executed in a strictly left to right order. No parentheses are allowed. All four operators are binary. Both + and – have unary uses also.

Addition (+)

The plus will add two constants and/or symbolic values. When used as a unary operator, it simply echoes the value.

Example:

```
ØØ1E    CON3Ø   EQU   3Ø
ØØ1Ø    CON16   EQU   1ØH
ØØØ3    CON3    EQU   3
3CØØ    VIDEO   EQU   3CØØH
3CØ3    A1      EQU   VIDEO + CON3
ØØ2E    A2      EQU   CON3Ø + CON16
3CØØ    A3      EQU   + VIDEO
```

Subtraction (–)

The minus operator will subtract two constants and/or symbolic values. Unary minus produces a 2's complement.

Examples:

```
3BFD    A1      EQU   VIDEO–CON3
ØØØE    A2      EQU   CON3Ø–CON16
C4ØØ    A3      EQU   –VIDEO
```

Logical AND (&)

The logical AND operator logically adds two constants and/or symbolic values.

Examples:

```
3CØØ    A1      EQU   3CØØH & FFH
ØØØØ    A2      EQU   Ø & 15
ØØØØ    A3      EQU   ØAAAAH & 5555H
```

Shift (〈)

The shift operator can be used to shift a value left or right. The form is:

VALUE 〈 AMOUNT

If AMOUNT is positive, VALUE is shifted left. If AMOUNT is negative, VALUE is shifted right.

Examples:

```
CØØØ    A1      EQU   3CØØH 〈 4
Ø3CØ    A2      EQU   3CØØH 〈 –4
BBFF    A3      EQU   3CBBH 〈 8 + 255
Ø3CØ    A4      EQU   15 + 3CØØH 〈 –4
```

Z80 STATUS INDICATORS (FLAGS)

The flag register (F and F') supplies information to the user regarding the status of the Z80 at any given time. The bit positions for each flag are shown below:

7	6	5	4	3	2	1	Ø
S	Z	X	H	X	P/V	N	C

WHERE:

C = CARRY FLAG
N = ADD/SUBTRACT FLAG
P/V = PARITY/OVERFLOW FLAG
H = HALF-CARRY FLAG
Z = ZERO FLAG
S = SIGN FLAG
X = NOT USED

Each of the two Z-80 Flag Registers contains 6 bits of status information which are set or reset by CPU operations. (Bits 3 and 5 are not used.) Four of these bits are testable (C,P/V, Z and S) for use with conditional jump, call or return instructions. Two flags are not testable (H,N) and are used for BCD arithmetic.

CARRY FLAG (C)

The carry bit is set or reset depending on the operation being performed. For 'ADD' instructions that generate a carry and 'SUBTRACT' instructions that generate no borrow, the Carry Flag will be set. The Carry Flag is reset by an ADD that does not generate a carry and a 'SUBTRACT' that generates a borrow. This saved carry facilitates software routines for extended precision arithmetic. Also, the 'DAA' instruction will set the Carry Flag if the conditions for making the decimal adjustment are met.

For instructions RLA, RRA, RLS and RRS, the carry bit is used as a link between the LSB and MSB for any register or memory location. During instructions RLCA, RLC s and SLA s, the carry contains the last value shifted out of bit 7 of any register or memory location. During instructions RRCA, RRC s, SRA s and SRL s the carry contains the last value shifted out of bit Ø of any register or memory location.

For the logical instructions AND s, OR s and XOR s, the carry will be reset.

The Carry Flag can also be set (SCF) and complemented (CCF).

ADD/SUBTRACT FLAG (N)

This flag is used by the decimal adjust accumulator instruction (DAA) to distinguish between 'ADD' and 'SUBTRACT' instructions. For all 'ADD' instructions, N will be set to a 'Ø'. For all 'SUBTRACT' instructions, N will be set to a "1".

PARITY/OVERFLOW FLAG

This flag is set to a particular state depending on the operation being performed.

For arithmetic operations, this flag indicates an overflow condition when the result in the Accumulator is greater than the maximum possible number (+127) or is less than the minimum possible number (−128). This overflow condition can be determined by examining the sign bits of the operands.

For addition, operands with different signs will never cause overflow. When adding operands with like signs and the result has a different sign, the overflow flag is set. For example:

+120 =	0111 1000	ADDEND
+105 =	0110 1001	AUGEND
+225	1110 0001	(−95) SUM

The two numbers added together has resulted in a number that exceeds +127 and the two positive operands has resulted in a negative number (−95) which is incorrect. The overflow flag is therefore set.

For subtraction, overflow can occur for operands of unlike signs. Operands of like sign will never cause overflow. For example:

+127	Ø111 1111	MINUEND
(−) −64	1ØØ ØØØØ	SUBTRAHEND
+191	1Ø11 1111	DIFFERENCE

The minuend sign has changed from a positive to a negative, giving an incorrect difference. Overflow is therefore set.

Another method for predicting an overflow is to observe the carry into and out of the sign bit. If there is a carry in and no carry out, or if there is no carry in and a carry out, then overflow has occurred.

This flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of '1' bits in a byte are counted. If the total is odd, 'ODD' parity (P=Ø) is flagged. If the total is even, 'EVEN' parity is flagged (P=1).

During search instructions (CPI,CPIR,CPD,CPDR) and block transfer instructions (LDI,LDIR,LDD,LDDR) the P/V flag monitors the state of the byte count register (BC). When decrementing, the byte counter results in a zero value, the flag is reset to Ø, otherwise the flag is a Logic 1.

During LD A,I and LD A,R instructions, the P/V flag will be set with the contents of the interrupt enable flip-flop (IFF2) for storage or testing.

When inputting a byte from an I/O device, IN r,(C), the flag will be adjusted to indicate the parity of the data.

THE HALF CARRY FLAG (H)

The Half Carry Flag (H) will be set or reset depending on the carry and borrow status between bits 3 and 4 of an 8-bit arithmetic operation. This flag is used by the decimal adjust accumulator instruction (DAA) to correct the result of a packed BCD add or subtract operation. The H flag will be set (1) or reset (Ø) according to the following table:

H	ADD	SUBTRACT
1	There is a carry from Bit 3 to Bit 4	There is no borrow from bit 4
Ø	There is no carry from Bit 3 to Bit 4	There is a borrow from Bit 4

THE ZERO FLAG (Z)

The Zero Flag (Z) is set or reset if the result generated by the execution of certain instructions is a zero.

For 8-bit arithmetic and logical operations, the Z flag will be set to a '1' if the resulting byte in the Accumulator is zero. If the byte is not zero, the Z flag is reset to 'Ø'.

For compare (search) instructions, the Z flag will be set to a '1' if a comparison is found between the value in the Accumulator and the memory location pointed to by the contents of the register pair HL.

When testing a bit in a register or memory location, the Z flag will contain the complemented state of the indicated bit (see Bit b,s).

When inputting or outputting a byte between a memory location and an I/O device (INI;IND;OUTI and OUTD), if the result of B-1 is zero, the Z flag is set, otherwise it is reset. Also for byte inputs from I/O devices using IN r,(C), the Z Flag is set to indicate a zero byte input.

THE SIGN FLAG (S)

The Sign Flag (S) stores the state of the most significant bit of the Accumulator (Bit 7). When the Z80 performs arithmetic operations on signed numbers, binary two's complement notation is used to represent and process numeric information. A positive number is identified by a 'Ø' in bit 7. A negative number is identified by a '1'. The binary equivalent of the magnitude of a positive number is stored in bits Ø to 6 for a total range of from Ø to 127. A negative number is represented by the two's complement of the equivalent positive number. The total range for negative numbers is from −1 to −128.

When inputting a byte from a I/O device to a register, IN r,(C), the S flag will indicate either positive (S=Ø) or negative (S=1) data.

PSEUDO-OPS

There are nine pseudo-op (assembler directives) which the assembler will recognize. These assembler directives, although written much like processor instructions, are commands to the assembler instead of the processor. They direct the assembler to perform specific tasks during the assembly process but have no meaning to the Z80 processor. These assembler pseudo-ops are:

ORG nn	Sets address reference counter to the value nn.
EQU nn	Sets value of a label to nn in the program: can occur only once for any label.
DEFL nn	Sets value of a label to nn and can be repeated in the program with different values for the same label.
END	Signifies the end of the source program so that any following statements are ignored. If no END statement is found, a warning is produced. The END statement can specify a start address i.e. END LABEL, END 6ØØØH. This address is used by the system program if no start address is given with the slash (/).
DEFB n	Defines the contents of a byte at the current reference counter to be n.
DEFB 's'	Defines the content of one byte of memory to be the ASCII representation of character s.
DEFW nn	Defines the contents of a two-byte word to be nn. The least significant byte is located at the current reference counter while the most significant byte is located at the reference counter plus one.
DEFS nn	Reserves nn bytes of memory starting at the current value of the reference counter.
DEFM 's'	Defines the content of n bytes of memory to be the ASCII representation of string s, where n is the length of s and must be in the range Ø < =n < = 63.

Assembler Commands

The TRS-80 Editor/Assembler supports only two assembler commands. Each command must start in column one of a source line, and must start with an asterisk (*). The assembler commands are:

*LIST OFF	Causes the assembler listing to be suspended, starting with the next line. Errors and bad source lines will still be printed.
*LIST ON	Causes assembler listing to resume, starting with this line.

Z80 INDEX TO INSTRUCTION SET

NOTE: Execution time (E.T.) for each instruction is given in microseconds for an assumed 4 MHZ clock. Total machine cycles (M) are indicated with total clock periods (T States). Also indicated are the number of T States for each M cycle. For example:

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

indicates that the instruction consists of 2 machine cycles. The first cycle contains 4 clock periods (T States). The second cycle contains 3 clock periods for a total of 7 clock periods or T States. The instruction will execute in 1.75 microseconds.

Register format is shown for each instruction with the most significant bit to the left and the least significant bit to the right.

INSTRUCTION SET
TABLE OF CONTENTS

OPERAND NOTATION

The following notation is used in the assembly language:

1) r specifies any one of the following registers: A,B,C,D, E,H,L.
2) (HL) specifies the contents of memory at the location addressed by the contents of the register pair HL.
3) n specifies a one-byte expression in the range (Ø to 255) nn specifies a two-byte expression in the range (Ø to 65535)
4) d specifies a one-byte expression in the range (–128, 127).
5) (nn) specifies the contents of memory at the location addressed by the two-byte expression nn.
6) b specifies an expression in the range (Ø,7).
7) e specifies a one-byte expression in the range (–126, 129).
8) cc specifies the state of the Flags for conditional JR and JP instructions.
9) qq specifies any one of the register pairs BC, DE, HL or AF.
10) ss specifies any one of the following register pairs: BC, DE, HL, SP.
11) pp specifies any one of the following register pairs: BC,DE,IX,SP.
12) rr specifies any one of the following register pairs: BC,DE,IY,SP.
13) s specifies any of r,n,(HL),(IX+d),(IY+d).
14) dd specifies any one of the following register pairs: BC,DE,HL,SP.
15) m specifies any of r,(HL),(IX+d),(IY+d).