TRS-80™
Model II

EDITOR ASSEMBLER

EDAS 4.0

by Galactic Software Ltd.

GALACTIC SOFTWARE LTD. MODEL II EDITOR ASSEMBLER 4.0

by Roy Soltoff, MISOSYS and Bill Schroeder, GALACTIC
Copyright 1980 by Galactic Software ltd.

GALACTIC's MODEL II EDITOR ASSEMBLER includes the following features:

1> Depression of <ENTER> without a command immediately provides a summary listing of the Editor Assembler's available commands.

2> "Load" and "Write" text buffers from/to the disk, as well as assembled object code filed to disk as a directly executable program file.

3> "Move" block allows the user to move lines of text from one location to another location in the text file.

4> "Find" will locate a designated string within the text. It will stop at each location and may be manually continued to the next occurence of the string.

5> "Global" replace allows the user to change a sequence of characters (STRING1) to a different sequence of characters (STRING2) throughout the entire text buffer.

6> "System" allows the user to perform any TRSDOS library command from the editor and return to the editor. You may do a DIR or maybe a KILL or LIST and even enter the DEBUGGER and never leave the environment of the Editor Assembler. Use this command to set FORMS, perform disk INIT, or CREATE files

7> "Edit" provides straightforward editing of designated text lines. The Editor maintains command syntax identical to the Model II BASIC editor while all editing is done in reverse video providing excellent user interface. Line insert, replace, and renumber round out the Editor's complement of commands.

8> "Assemble" with numerous switches is provided to allow for many different types of assembled output. PLUS the unique assemble to memory system, so the user may effectivly debug the object code before leaving the Editor Assembler environment.

9> "Jump" allows you to execute your program, that has been assembled to ram, and then return to the Editor Assembler.

10> "Usage" allows quick reference to the present usage of available ram by printing the number of bytes remaining in your text buffer, how many are in use, plus the address of the first free byte after the text. This last address is useful when using the ASSEMBLE TO MEMORY feature.

11> "Hardcopy" will print all or part of the text buffer.

12> "Type" will print the text buffer without line numbers.

13> The <F1> key is employed as a functional <CLEAR> key.

14> The symbol table is sorted in ascending alphanumeric order and output 5-across in 80 column format.

15> The <UP ARROW> and <DOWN ARROW> keys provide instant scroll up or down, one line at a time, or repeated, with the repeat key.

16> The <F2> key is employed to provide instant advance of a entire text page (23 lines). This "PAGE" function may also be used with the repeat key.

17> The <HOLD> key is employed as a functional Pause key.

18> All Editor Assembler commands may be entered in either upper case or lower case providing ease of operation as a text editor.

19> Great amounts of time and effort were expended to give the user of this Editor Assembler the absolute best in ease of operation and functional efficiency. Optimize assembly programing time, with the EDITOR ASSEMBLER designed with the programmer in mind.

W A R R A N T Y

ALL GALACTIC SOFTWARE IS WARRANTED FOR ONE YEAR FROM THE DATE OF PURCHASE TO BE FREE FROM CODING DEFECTS. SHOULD A PROBLEM BE FOUND BY THE USER, GALACTIC WILL CORRECT IT AT NO CHARGE, DURING THE WARRANTY PERIOD. THIS SERVICE SHALL ONLY BE PROVIDED TO CUSTOMERS WHO REGISTER WITH GALACTIC, WITHIN 30 DAYS OF PURCHASE, BY RETURNING THE REGISTRATION CARD PROVIDED IN THE REAR OF THE MANUAL. GALACTIC'S LIABILITY SHALL NOT, IN ANY WAY, EXTEND BEYOND CORRECTION OF THE PROGRAM ITSELF. THE USER IS TOTALLY RESPONSIBLE FOR ALL DATA ENTRUSTED TO THE PROGRAM AND HIS HARDWARE.

INTRODUCTION

The Galactic Software Editor Assembler is a RAM-resident text editor and assembler for the Model II microcomputer system. The Editor Assembler was designed to provide the maximum in user interface and ease of use while providing capabilities powerful enough for the expert Z80 assembly language programmer.

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 code programs into machine executable code. This object code may then be executed directly from TRSDOS (tm) as a program file. Previous knowledge of machine language and the hexadecimal 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 "+", "-", "&" (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), hexadecimal (H), or octal (O).

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

()	Parentheses enclose optional information. They are never input in Editor Assembler commands.
...	The ellipses represents repetition of a previous item.
line	Any decimal number from 1 to 65529.
.	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.
T	The character <T> may be used in place of any line number. It represents the top of the text buffer.
B	The character <B> may be used in place of any line number. It represents the bottom of the text buffer.
inc	A number representing an increment between successive line numbers.

All Editor Assembler commands may be entered in lower case as well as upper case to facilitate its use as a general purpose text editor. Assembler source code must be entered in upper case only. It is suggested that <CAPS LOCK> be used to enter source code.

A file called "Z80CODE/SOR" has been written to the diskette containing the Galactic Software Editor Assembler. This file is a Z-80 code source file containing the entire Z-80 code instruction set which can be loaded into the Editor Assembler. If assembled, it will produce Z-80 object code in numeric order. The generated listing will be similar to the NUMERIC LIST OF INSTRUCTION SET located at the rear of your Editor Assembler manual.

GETTING STARTED WITH EDAS 4.0

It is strongly recommended that before using your new Editor Assembler, you should make a BACKUP copy to use in a working environment and retain the EDAS diskette as your MASTER copy. The BACKUP utility procedures are found in your "TRS-80 Model II Owner's Manual" in the section entitled "UTILITY PROGRAMS". After creating a BACKUP copy of the EDAS diskette, store the MASTER diskette in a safe place. Use only your "working" copy for production.

EDAS 4.0 is a directly executable program file. It is accessed simply by entering:

```
EDAS40
```

in response to the TRSDOS query

```
TRSDOS READY
```

....

EDAS 4.0 will load, execute, and display the following:

```
GALACTIC SOFTWARE LTD.  MODEL II EDITOR ASSEMBLER 4.0
 By Roy Soltoff, MISOSYS and Bill Schroeder, Galactic
      Copyright 1980 by Galactic Software ltd.
 >
```

The right carat ">" which appears in reverse video, is the prompting character displayed by EDAS whenever it is ready to accept a command. If you would like a memory jogger as to what commands are acceptable to EDAS, just depress <ENTER>. The entire command repertoire will be instantly displayed.

WELCOME TO THE WORLD OF SOPHISTICATED BUSINESS SOFTWARE

ASSEMBLY LANGUAGE

Syntax

The basic format of an assembly language statement is:

(LABEL)   OPCODE   (OPERAND(S))   (COMMENT)

LABELS

A label is a symbolic name of a line of 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-Z). A label may not contain the dollar sign ($) character. The dollar sign ($) 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)

OPCODES

The OPCODES for the Galactic Software Model II Editor Assembler correspond to those in the Z-80-ASSEMBLY LANGUAGE PROGRAMMING MANUAL, 3.0 D.S., REL 2.1, FEB 1977.

OPERANDS

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

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 - hexadecimal
- D - decimal
- O - octal

A constant not followed by one of these letters is assumed to be decimal. A constant must begin with a digit. Thus "FFH" is not permitted, while "0FFH" is valid.

Expressions using the "+", "-", "&", and "<" operators are described in the 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 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.

Examples:

```
001E     CON30    EQU    30

0010     CON16    EQU    10H

0003     CON3     EQU    3

002E     A2       EQU    CON30 + CON16
```

Subtraction (-)

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

Examples:

```
     000E    A2        EQU   CON30 - CON16

     FFF2    A4        EQU   -A2
```

Logical AND (&)

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

Examples:

```
     3C00      A1      EQU     3C00H & 0FFH

     0000      A2      EQU     0 & 15

     0000      A3      EQU     0AAAAH & 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. The magnitude of the shift is determined from the numeric value of AMOUNT

Examples:

```
     C000      A1      EQU     3C00H < 4

     03C0      A2      EQU     3C00H < -4

     BBFF      A3      EQU     3CBBH < 8 + 255

     03C0      A3      EQU     15 + 3C00H < -4
```

Z-80 STATUS INDICATORS (FLAGS)

The flag registers (F and F') supply information to the user regarding the status of the Z-80 at any given time. The bit positions for each flag are shown below:

7	6	5	4	3	2	1	0
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 contain 6 bits of status information which are set or reset by CPU operations. Four of these bits are testable (C, P/V, Z, and S) for use with conditional jump, call, or return instructions. Two flags (H, N) are not testable and are used for BCD arithmetic. Two flag register bits (3, 5) are not used by the Z-80.

CARRY FLAG (C)

The carry flag is set or reset depending on the operation being performed. For "ADD" instructions that generate a carry and "SUBTRACT" instructions that generate a 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 no 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 0 of any register or memory location.

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

The carry flag can also be set (SCF) or 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 "zero". For all "SUBTRACT" instuctions, N will be set to a "one".

PARITY/OVERFLOW FLAG (P/O)

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  =  0111 1111     MINUEND
   (-)-64  =  1100 0000     SUBTRAHEND
   ------------------------------------
     +191  =  1011 1111     DIFFERENCE
```

The minuend sign has changed from a positive to a negative giving an incorrect difference. The overflow flag 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 "one" bits in a byte are counted. If the total is odd, "ODD" parity (P=0) is flagged. If the total is even, "EVEN" parity is flagged (P=1).

During search instructions (CPI, CPIR, CPD, and CPDR) and block transfer instructions (LDI, LDIR, LDD, and 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 zero, otherwise the flag is a one.

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 (0) 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
0	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 a certain instruction is a zero.

For 8-bit arithmetic and logical operations, the Z flag will be set to a "one" if the resulting byte in the Accumulator is zero.

For compare (search) instructions, the Z flag will be set to a "one" 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 state of the indicated bit.

When inputting or outputting a byte between a memory location and an I/O device (INI, IND, OUTI, or 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 Z-80 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 "zero" in bit 7. A negative number is identified by a "one". The binary equivalent of the magnitude of a positive number is stored in bits 0 to 6 for a total range of from 0 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 an I/O device to a register, "IN r,(C)", the S flag will indicate either positive (S=0) or negative (S=1) data.

PSEUDO-OPS

There are nine pseudo-ops (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 Z-80 processor. These assembler pseudo-ops are:

ORG nn	Sets address reference counter to the value nn.
EQU nn	Sets the value of a label to nn in the program: can occur only once for any label.
DEFL nn	Sets the 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 transfer address (i.e. END LABEL or END 6000H). The transfer address is used by the TRSDOS program execution to transfer control to the address specified in the END statement.
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 2-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 contents 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 0-63.

ASSEMBLER COMMANDS

The Galactic Software 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.
*LIST ON	Causes assembler listing to resume, starting with this line.

COMMANDS:

The GALACTIC Model II Editor Assembler can perform the following commands. These commands may be typed after the prompt symbol ">" which is displayed in reverse video for clarity. The prompt symbol appearance indicates the "command mode" of the Editor Assembler. The following list contains all command mode instructions recognized by the Editor Assembler with a brief description of each.

A	Assemble source currently in the text buffer.
C	Display the FILESPEC of the last source text file accessed either by Load or Write.
D	Delete specified line(s).
E	Edit a specified line of text.
F	Find a specified string of characters.
G	Globally change a string of characters (STRING1) to another string of characters (STRING2) throughout the text buffer.
H	Provide hard copy output (line printer) of a specified range of text buffer lines.
I	Insert source text line(s) at a specified line with a specified line number increment.
J	Jump to a specified address.
L	Load a source text file from disk.
M	Move a block of text from one location to another.
N	Renumber source text lines in the text buffer.
P	Print a specified range of source text code currently in the text buffer.
Q	Quit the Editor Assembler and return to TRSDOS.
R	Replace lines currently in the text buffer.
S	System command to execute any TRSDOS command from within the environment of the Editor Assembler.
T	Type source text lines without line numbers to a line printer.

U	Display the memory utilization - bytes used by the text, bytes available, and the first free address.
W	Write the current text buffer to disk.
F1	Clear the CRT screen.
F2	Page forward the display 23 lines.
↑	Scroll up one source text line.
↓	Scroll down one source text line.
HOLD	Performs a functional pause of any operation.

GALACTIC EDITOR ASSEMBLER COMMAND DETAILS

1.> ASSEMBLE (A)

Syntax: A (SWITCH(/SWITCH)...)

SWITCH may be any of the following five options:

NL	No assembler listing is written to the screen.
NO	No assembled object code is generated to disk or memory.
NS	No symbol table is printed either to the screen or the line printer (if enabled).
LP	Send assembler listing, error messages, and symbol table (if enabled) to a line printer.
WE	Cause the assembly to wait when an error occurs. Depressing any key will continue assembly until another error is found. If you want to continue the assembly without stopping for additional errors, enter the character <C>

If you want to assemble an object program to either disk or memory, do not enter the switch parameter, "NO". The prompt

```
Object code to disk or memory (D,M)?
```

will be displayed. A response of "M" will assemble the object code to memory. You will not be permitted to overwrite any region below the end of the text buffer nor will you be permitted to overwrite the symbol table stored in high memory. The error message,

```
     Attempt to overwrite protected region - job aborted
```

will be displayed if your assembled program will violate these restrictions. Upon successful completion of the assembly to memory, the message,

```
          Memory region loaded
          XXXXX is the transfer address
```

will be displayed.

A response of 'D' assumes a disk object code file. The 'D' response will issue the query,

```
          Enter filespec
```

Respond with the filespec that you want to use to save the assembled program file. The Editor Assembler will open the file if existing and output the message,

```
          Replaced
```

or create the file if non-existant and output the message,

```
          New file
```

Assembly will start and the program file will be written to disk.

2.> DISPLAY CURRENT SOURCE FILESPEC (C)

Syntax: C

This command will display the filespec used for the most recent Load or Write command. If neither Load nor Write were utilized, or the text buffer region was cleared, the message,

```
     Filespec unknown
```

is displayed.

3.> DELETE (D)

Syntax: D (line1 (,line2))

This command is used to delete the line or lines specified from the source text buffer. The character <T> is used to indicate the top of the text buffer and the character <B> is used to indicate the bottom of the text buffer.

Examples:

D 100,500	Delete lines 100 through 500 (inclusive) from the text buffer.
D T,B	Delete the entire text buffer.
D	Delete the current source text line. A period (.) may also be used to indicate the current line.
D 105	Delete the single line 105.

4.> EDIT (E)

Syntax: E (line)

This command permits the user to edit or modify any source text line. While in the edit mode, the line being editted is displayed in reverse video. The syntax and function of all edit subcommands are identical to those implemented in the DISK BASIC editor.

Edit Subcommands:

A	Abort and restart the line edit.
nC	Change n characters.
nD	Delete n characters.
E	End editting and enter the changes.
H	Delete the remainder of the line and insert the following string. The "H" command should not be used to delete an entire line of text. There M U S T 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 the starting position of the line.
Q	Quit and ignore all editting.
nSx	Search for the nth occurrence of x.

BACKSPACE	Move edit pointer back one space.
ESCAPE	Escape from any edit mode subcommand.
ENTER	Enter the line in its presently editted form and exit the edit mode.

5.> FIND (F)

Syntax: F (string)

The edit buffer is searched starting at the current line+1 for the first occurrence of "string". If no string is specified, the search is the same as that of the last Find command in which a string was specified (provided a Global command was not previously specified). If the search string is found, the line containing it is displayed and period (.) is updated to the displayed line. If the string is not found, the message,

```
String not found
```

is displayed and period (.) remains unchanged. A "PT" command can be used to position the line pointer to the top of the text buffer prior to use of the Find command.

6.> GLOBAL (G)

Syntax: G /string1/string2/

A string of characters can be changed throughout the text buffer by one easy command. The GLOBAL CHANGE command will change the appearances of STRING1 to the sequence STRING2. No changes will be performed on the first line of the text buffer. Also, only the first appearance of STRING1 in each line that STRING1 appears will be altered.

The first non-blank character becomes the string delimiter (the slash character is shown above; any character is permitted). Null strings are not permitted (i.e. the string must contain at least one character).

It is not necessary that STRING2 be the same length as STRING1. It can be of lesser, equal, or greater length; however, no string can exceed 16 characters in length. If a change would result in a line exceeding the maximum line length (128), the change will not be performed on that line and the message,

```
FIELD OVERFLOW
```

will be issued. The search for STRING1 continues for the remaining lines.

A line which contains STRING1 will be displayed as it exists both before and after the change. The <HOLD> key may be used to pause the output. Use of the <BREAK> key will stop further changing.

Example:

```
      G /MODIFY/ALTER/
```

7.> HARDCOPY (H)

Syntax: H (line1 (,line2))

This command will print a line or a group of lines to a line printer. If a properly paged display is desired, it is suggested that you set the forms control by issuing the Editor Assembler's "System" command as in:

```
     S FORMS (P=xx,L=xx,)
```

Examples:

H T,B	Print the entire text buffer.
H 100,500	Print lines 100 through 500 inclusive.
H.	Print the single line pointed to by period (.).
H	Print the 23 lines starting with the current line.

8.> INSERT (I)

Syntax: I line# (,inc)

The Insert command is used to insert or add text lines in the buffer. All lines of source text are entered with the use of the Insert command. After using the Insert command to specify were you wish to place new lines, the EDITOR will generate the designated line number and allow the inserting of that numbered text line. After entering the first text line the editor will generate the next line number higher, as specified by your increment selection. Incremental line numbers will continue to be generated as long as there is room between lines or room left in the text buffer.

The <BREAK> key will allow you to leave the insert mode at any time.
If a desired increment is not specified the last specified increment is assumed. Period (.) may be used for "line#" to indicate the current line.

9.> LOAD (L)

Syntax: L (filespec)

The Load command will read the file denoted by the FILESPEC into the text buffer. The text file will be concatenated to any text already in the text buffer. FILESPEC is explained in your TRSDOS (tm) user manual under the "TRSDOS" section entitled "file specification". It is composed of a FILENAME, optional EXTension, optional PASSWORD, optional DRIVE reference, and optional diskette name as in

```
     FILENAME/EXT.PASSWORD:D(DISKETTE NAME)
```

(ex. YOURPROG/ASM:1). If you do not enter the FILESPEC, Editor Assembler will use the filespec entered for the last Load or Write command provided there is text already in the text buffer. If the text buffer is empty and you do not enter a filespec with the Load command, Editor Assembler will prompt you for the filespec.

10.> MOVE (M)

Syntax: M line1, line2, line3

This command is used to move a block of lines from one location in the text buffer to another. A large quantity of text lines can be moved to a different position in one easy operation. In the command syntax, "line1" and "line2" are the beginning and ending line numbers of the text block to be moved. "Line1" and "line2" are permitted to reference the same line number if only one line is to be moved. "Line3" is the line number of the line that the text block will follow after the move. The line number references must be offset by commas (,). If any of the entered line numbers are non-existant, the message,

```
     No such line
```

will be issued.

"Line3" is not permitted to equal "line1" or "line2". "Line3" is not permitted to be a line interior to the range "line1" through "line2". The message,

```
     Command parameter(s) incorrect
```

will be issued if your input violates any of these conditions.

The text to be moved is stored temporarily in the spare text region. If this region is not large enough to store the block, the message,

```
     Text buffer full
```

will be issued. Try moving the block in segments.

Upon completion of the move, all lines in the text buffer will be renumbered starting from ten (10) and using the line increment currently in effect. Renumbering is absolutely essential to perform proper operation of Editor Assembler commands.

Example:

You desire to move the block of text starting at line 500 and ending at line 900 to follow line 1510. Issue the command,

```
     M 500,900,1510.
```

11.> RENUMBER (N)

Syntax: N (line(,inc))

The "N" command is used to renumber the lines in the text buffer. The first line in the buffer is assigned the number specified as "line". If "line" is not specified, it defaults to 00100. The remaining lines in the buffer are renumbered according to the increment (inc) or the previous increment in a RENUMBER, REPLACE, or INSERT command if the increment was not specified. Period (.) points to the same line as it did before the NUMBER command was used, but the actual number of this line may be changed.

Examples:

N	Renumbers from 100 with the previous increment.
N5	Renumbers from 5 with the previous increment.
N10,5	Renumbers from 10 in steps of 5.

12.> PRINT (P)

Syntax: P (line1(,line2))

The PRINT command will display a line or a group of lines on the monitor screen. Period (.) is updated to point to the last line printed.

Examples:

P T,B	Displays all lines in the text buffer.
P 100,500	Displays lines 100 through 500 inclusive.
P .	Displays the current line only.
P	Displays 23 lines starting with the current line. The PRINT command operates in a screen scroll mode. If you want to "page" the screen, use the "F2" command.

13.> QUIT (Q)

Syntax: Q

The QUIT command is used to exit the Editor Assembler and perform a proper return to TRSDOS. By using command "Q", the <BREAK> key will be restored to TRSDOS.

14.> REPLACE (R)

Syntax: R (line<inc))

The REPLACE command only replaces one line and enters INSERT mode. If "line" exists, it is deleted then inserted. If line doesn't exist, it is inserted as with the INSERT command. If "inc" is not specified, the last increment specified by an INSERT, REPLACE, or RENUMBER command is used. Period (.) is always updated to the current line.

Examples:

R	Replace the current line.
R 100,10	Start replacing lines beginning at line 100 and incrementing with 10.
R 100	Start replacing at line 100 using the last specified increment.

15.> SYSTEM (S)

Syntax: S ANY-TRSDOS-COMMAND (PARAMETERS)

The SYSTEM command is used to interface with TRSDOS while in the environment of the Editor Assembler. Any TRSDOS command can be accessed. It is recommended that you not attempt to access the TRSDOS "COPY" nor "BACKUP" commands due to the possibility of overwriting the Editor Assembler. IT IS

IMPORTANT TO NEVER DEPRESS THE <BREAK> KEY DURING A SYSTEM TRSDOS COMMAND. To break any TRSDOS command, use the <ESCAPE> key.

Examples:

S DIR	List the diskette directory.
S FORMS (P=51,L=42)	Set printer parameters.
S LIST filespec	List the contents of a file.
S PURGE :d	Delete files from drive "d".

16.> TYPE (T)

Syntax: T (line1(,line2))

The TYPE command prints a line or group of lines onto the Line Printer. Period (.) is updated to point to the last line printed. This command is much like the HARD COPY command, only no line numbers are printed. Only the source text is printed.

17.> MEMORY USAGE (U)

Syntax: U

This command will display the number of bytes of text buffer in use, the number of bytes spare and the first address available for assembly to memory.

This command is useful to ascertain requirements for storing the text buffer to disk. Note that a disk file, which is written in ASCII, will contain an additional four (4) bytes per text line. The 4 bytes arise from the difference in storage formats of text in memory versus text in an ASCII file.

It also is useful when assembling to memory. Since the Assembler will not permit you to overwrite it or the text buffer, you will have to "ORG" your program in the free text buffer area. The first available address is output by this command.

An example of output is:

```
12288 bytes in use
27934 bytes spare
37292 (91AC) is the first free address
```

18.> WRITE (W)

Syntax: W (filespec)

This command will write the text buffer to the file denoted by FILESPEC. If no FILESPEC is entered, the filespec referenced by the previous Load or Write command will be used unless the text buffer is empty. If a FILESPEC is unavailable for use, you will be prompted for it.

If the file denoted by FILESPEC is non-existant, a file will be created and the message,

```
      New File
```

will be issued.

If the file denoted by FILESPEC is an existing file, it will be replaced by the write operation and the message,

```
      Replaced
```

will be issued. YOU WILL NOT BE GIVEN AN OPPORTUNITY TO CANCEL A WRITE REQUEST ON AN EXISTING FILE. Know what you are doing.

19.> SCROLL UP (↑)

The SCROLL UP command displays the line preceding the current line and updates period (.) to point to the line displayed. If the current line is the first line in the text buffer, it is displayed and period (.) remains unchanged. SCROLL UP is an immediate command and must be the first character of a command line in order to be interpreted.

20.> SCROLL DOWN (↓)

The SCROLL DOWN command displays the line following the current line and updates period (.) to point to the line displayed. If the current line is the last line in the text buffer, the last line is displayed and period (.) remains unchanged. SCROLL DOWN is an immediate command and must be the first character of a command line to be interpreted.

21.> CLEAR SCREEN (F1)

The <F1> key is used to perform a functional clear screen (similar to "S CLS").

22.> PAGE FORWARD (F2)

The <F2> key is used to advance the display by 23 lines. This command is similar to the PRINT command except that the monitor screen is cleared before displaying the 23 lines.

23.> PAUSE (HOLD)

The <HOLD> key is used to pause the computer during a display during any assembly or Editor Assembler printing. When a pause is sensed, depression of any key except <HOLD>, <SHIFT>, or <CTRL> will continue the operation paused.

Error Messages

The Galactic Software Model II Editor Assembler recognizes three types of errors. These are:

1.> Command errors - The error message is displayed and control is returned to command mode.

2.> TRSDOS errors - The error message (or error number) is displayed and control is returned to command mode.

3.> Assembler errors - These three types of errors may occur while executing an Assemble command.

   a. Terminal - Assembly is terminated and control is returned to command mode.

   b. Fatal - Processing of the line containing the error is immediately stopped and no object code is generated for that line. Assembly proceeds with the next line.

   c. Warning - The error message is displayed and assembly of the line containing the warning continues. The resulting object code may not be what the programmer intended.

Following is a list of all errors and an explanation of each.

COMMAND ERRORS

1.> Buffer full

There is no more room in the text buffer for adding text.

2.> Command parameter(s) incorrect

Any command line not entered according to the syntax appropriate for that command will generate this error message.

3.> Illegal command

The first character of the command line entered does not specify a valid Editor Assembler command.

4.> Invalid source file

A Load filespec command was issued where the file identified by filespec is not a valid Editor Assembler source file.

5.> Line number too large

Renumbering with the specified starting line number and increment would cause line(s) to be assigned numbers greater than 65529. The renumbering is not performed. This message would also be displayed if you attempted to INSERT a line with a line number exceeding 65529.

6.> No room between lines

The next line number to be generated by INSERT or REPLACE would be greater than or equal to the line number of the next line of text in the edit buffer. The increment must be decreased or the lines in the buffer renumbered.

7. No such line

A line specified by a command does not exist. The command is not performed.

8. No text in buffer

A command requiring text in the buffer was issued when the text buffer was empty. The commands Load, Insert, Quit, System, Jump, <F1>, and Display current filespec can be executed when the text buffer is empty. All other commands require at least one line of text to be in the buffer.

9.> String not found

The string being searched for by the Find command could not be found between the current line and the end of the text buffer. This message will also be displayed at the completion of a Global command.

TRSDOS ERRORS

1.> Disk drive not ready

This message will be displayed after a Load, Write, or Assemble to disk command is executed if either the specified drive is not ready (no diskette, diskette in backwards, drive door not closed, etc.) or the specified drive does not exist.

2. Disk is write protected

A Write command was issued with a filespec designating a drive loaded with a diskette that is protected from a write operation.

3.> Unusable file specification

A Load, Write, or Assemble to disk command was executed with a filespec that did not conform to TRSDOS specifications. It is also possible that the drive specified was not in the range 0-3.

4.> Filespec not in directory

The filespec entered for execution of a Load, Write, or Assemble to disk command could not be located in the drive directory.

5.> Access denied (password incorrect or missing)

An attempt was made to access a TRSDOS file. Either the password entered was incorrect or no password was entered for a password protected file.

6.> Too many files in the directory

The directory space is full on the designated diskette.

7.> No disk space available

A Write or Assemble to disk command was executed which resulted in a file using the available disk space prior to completion. The operation terminates and the file is closed.

NOTE: Under TRSDOS 1.2, a TRSDOS system error causes unpredictable behavior of the system when a diskette becomes full. It is strongly recommended that you pay close attention to the amount of available space on a diskette by issuing the System commands DIR or FREE. As a diskette's available free space diminishes, you may want to avoid creating any new files on it and continue your operation with a diskette that has sufficient free space. File storage requirements for the text buffer may be ascertained using the Editor Assembler's USAGE command.

8.> Hardware failure during I/O

This message is displayed when a disk operation is unsuccessful and TRSDOS returns error code 41 or 49.

9.> Printer is not ready for use

Any output sent to the line printer when the printer is unavailable will generate this error. The printer may be turned off, out of paper, in trouble, or not plugged into the system.

10.> TRSDOS error code # <xxxxx>

Any other TRSDOS error not specifically identified above will be displayed in this form. If you want the full TRSDOS explanation, issue the command:

```
S ERROR xxxxx
```

TERMINAL ERRORS

1. Attempt to overwrite protected region (job aborted)

During an assembly to memory, a block of code was assembled that would load into a memory region other than the spare text buffer area. Your program will not be permitted to load to an address below the end of the text buffer or above the symbol table. Use the Usage command to locate the first available memory address.

2.> Symbol table overflow

There is not enough memory for the assembler's symbol table. You have three options:

a. Remove comment lines and/or comments following Z-80 code operands. This may free up enough space to perform the assembly.

b. TRSDOS locks out space above X'F300' for user use. This space is utilized by the DEBUG program and Serial port drivers. If your operation will not use the either serial port and DEBUG is to remain OFF, then this space can be recaptured. Do the following:

   1. Save your current text buffer.

   2. Return to TRSDOS via the Quit command.

   3. Enter the TRSDOS command, "DEBUG ON"

   4. Load the Editor Assembler program using the TRSDOS "LOAD" command.

   5. Enter the TRSDOS command, DEBUG

   6. Using the DEBUG command "R", change register pair "DE" to X'FFFF'.

   7. Using DEBUG's Jump command, jump to X'3403'. You will enter the Editor Assembler with its top-of-memory now set to X'FFFF'.

8. Enter the Editor Assembler command, S DEBUG OFF

9. Load your previously saved text buffer and attempt to assemble it.

c. Split your source program into two or more programs that can be assembled separately.

FATAL ERRORS

1.> Bad label

The character string found in the label field of the source statement does not match the criteria specified under ASSEMBLY LANGUAGE - LABELS.

2.> Expression error

The operand field contains an ill-formed expression.

3.> Illegal addressing mode

The operand field does not specify an addressing mode which is legal with the specified opcode.

4.> Illegal opcode

The character string found in the opcode field of the source statement is not a recognized instruction mnemonic or assembler pseudo-op.

5.> Missing information

Information vital to the correct assembly of the source line was not provided. The opcode is missing or the operands are not completely specified.

WARNINGS

1.> Branch out of range

The destination of a relative jump instruction (JR or DJNZ) is not within the proper range for that instruction. The instruction is assembled as a branch to itself by forcing the offset to hex X'FE'.

2.> Field overflow

A number or expression result specified in the operand field is too large for the specified instruction operand. The result is truncated to the largest allowable number of bits.

3.> Multiply defined symbol

The operand field contains a reference to the symbol which has been multiply defined. The first definition of the symbol is used to assemble the line.

4.> Multiple definition

The source line is attempting to illegally redefine a symbol. The original definition of the symbol is retained. Symbols may only be redefined by the DEFL pseudo-op and only if they were originally defined by DEFL.

5.> No end statement

The program END statement is missing

6.> Undefined symbol

The operand field contains a reference to a symbol which has not been defined. A value of zero is used for the undefined symbol.

TECHNICAL SPECIFICATIONS

Object file format

The disk file object code format consists of the following structure:

1.> A file header string consisting of:

a. The first byte in the file is a hex byte X'05' which indicates the header field of an object file.

b. The second byte is the header length byte and indicates the length of the header following.

c. The length byte is followed by the FILENAME and EXTENSION that was specified when the file was last written to.

d. The filename field is immediately followed by the entire DATE string as recovered by the TRSDOS SVC DATE - Function Code 45. By LISTing the first sector of the file, you can determine when the file was last written by examining this header. A TRSDOS RENAME command will not change the filename stored in the header.

2.> Multiple blocks of object code depending on the length of your assembled program are placed next. The object code blocks have the following code format:

a. A beginning byte of X'01' which indicates the start-of-block

b. A 1-byte length indicating the length of the code block following, including the block load address (block length of 256 will show X'02'). The Editor Assembler writes 128-byte blocks (length = X'82').

c. The block length byte is followed by the 2-byte block load address which is the address that will be loaded with the first byte of the block.

d. Finally the block immediately follows for as many bytes as two less than the block length.

3. Steps 2a, 2b, 2c, and 2d are repeated for as many blocks as are in the file. An X'02' is then written to indicate the end of the program code and the start of the entry point or

transfer address. An X'02' is written to indicate the length of the entry point address. This is then followed by the 2-byte entry point or transfer address generated from the label or constant in the operand field of the assembler source END statement.

Source file format

The source code file format is as follows:

1.> A header record as described under "Object file format" is also used for source files with the exception that the first byte is a hex X'53' to identify the file as source.

2.> Text lines are written in ASCII each composed of a 5-character line number (bit 7 is not set), a space, the text line, ending with an <ENTER> (X'0D').

3.> The file end is indicated by an end-of-file mark of X'1A' which would be in the first character position of a text line.

4.> Model I source text files follow a different format (header start byte of X'D3', followed by a 6-character filename with text line numbers having bit 7 set). In spite of this difference, source files generated on a Model I machine using the MISOSYS DISK*MODified EDTASM and uploaded to a Model II machine can be loaded into this Editor Assembler.

LINKAGE TO DEBUGGING

In order to facilitate the debugging of user generated programs, a number of features have been built into this Editor Assembler. It provides the option of assembling source code directly to memory. It provides a command to transfer control to a user-supplied address (via the JUMP command). It provides for the access of DEBUG through the System command. Other subtle enhancements have been implemented.

A re-entry address to the Editor Assembler has been provided. If at any time during the debugging phase, you want to return to the Editor Assembler without reinitializing it (which would have deleted the entire text buffer), and are under the control of DEBUG, issue a DEBUG Jump command to X'3400'. A return to the Editor Assembler will be performed and it will take over the supervision of the <BREAK> key without reinitializing the pointers to the text buffer.

When you exit from the Editor Assembler by means of the Jump command, address X'3400' is pushed onto the stack just prior to executing the jump. If your program maintains stack integrity, an easy return to the Editor Assembler is achieved by means of a "RET" instruction. An example of this procedure is as follows:

```
BEGIN     LD     (SPSAV),SP     ;SAVE THE POINTER
          .
          .
          .
          USER PROGRAM
          .
          .
          .
EXIT      LD      SP,(SPSAV)    ;RESTORE STACK
          RET                   ;& RETURN TO EDAS
```

Z-80 INDEX TO INSTRUCTION SET

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 two machine cycles. The first cycle contains four clock periods (T states). The second cycle contains three clock periods for a total of seven 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.

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 0 to 255. "nn" specifies a two-byte expression in the range 0 to 65535.

4. "d" specifies a one byte expression in the range -128 to +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 0 to 7.

7. "e" specifies a one-byte expression in the range -126 to 129.

8. "cc" specifies the state of the Flags for conditional JR and JP instructions.

9. "qq" specifies any one of the following register pairs:

   BC, DE, HL, & 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. "'" specifies any one of the following:

   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 the following:

   r, (HL), (IX+d), & (IY+d)

```
**************************************************************
**************************************************************
**************************************************************
****************                            *****************
****************        I M P O R T A N T   *****************
****************                            *****************
**************************************************************
**************************************************************
**************************************************************
```

GALACTIC SOFTWARE LTD.   11520 N. PORT WASHINGTON RD.
MEQUON, WISCONSIN 53092

GALACTIC SOFTWARE WOULD LIKE TO PROVIDE YOU WITH THE BEST TECHNICAL SUPPORT POSSIBLE. TO PROVIDE THIS SUPPORT WE NEED TO KNOW WHO OUR CUSTOMERS ARE. SO PLEASE FILL OUT THE CARD BELOW AND RETURN IT TO US PROMPTLY. LEAVE THE REMAINDER OF THIS PAGE IN THIS MANUAL!! THE REGISTRATION NUMBER MUST BE MENTIONED ON ALL CORRESPONDENCE WITH US OR WHEN PHONING OUR CUSTOMER SERVICE DEPARTMENT, SO DON'T LOSE IT!!

E D I T O R / A S S E M B L E R

SERIAL # 80402056 VER # ....
(C) 1980 BY GALACTIC SOFTWARE LTD.

E D I T O R / A S S E M B L E R

SERIAL # 80402056 VER # ....
(C) 1980 BY GALACTIC SOFTWARE LTD.

COMPANY NAME ______________________________
NAME ______________________________
ADDRESS ______________________________
CITY ______________ STATE ______ ZIP ______
AREA CODE ______ PHONE ____________ EXT ______

WHEN DID YOU PURCHASE THIS PROGRAM? MO. ____ YR. ____
WHERE WAS IT PURCHASED?
DEALER NAME ______________________________
ADDRESS ______________________________
CITY ______________ STATE ______ ZIP ______

11520 N. Port Washington Rd.
Mequon, Wisconsin 53092

Put Stamp Here.
The Post Office
will not deliver
mail without
postage.

galactic software ltd.

11520 North Port Washington Road
Mequon, Wisconsin 53092