ZETU

A SORCERER CASSETTE BASED ASSEMBLER PACKAGE

INSTRUCTION MANUAL

Produced under licence by SYSTEM SOFTWARE

1. LOADING ZETU

To load ZETU, enter the Monitor, type LOG and start the tape. The program will load and start execution automatically. Note that the BASIC ROM PAC need not be installed for ZETU to operate.

2. THE COMMAND MODE

When ZETU begins execution, it comes up in the Command Mode, as signified by the command prompt "*". At this point you may enter one of the following commands (the command must be all capitals).

EDIT ASMB SSRC LSRC MSRC SOBJ EXIT

Each of these commands and their related options will now be described in detail. Options related to a given command are shown in angular brackets <>.

3. THE EDITOR

Command: EDIT <C>

The command EDIT is used to enter the editing mode of ZETU. The optional "C" causes the text buffer to be cleared before editing begins (that is, C is typed when starting a new source file). When the Editor is called, the first page of text is displayed. Text may be typed in and edited using the full screen editing capabilities provided. Each line, comments included, should not exceed 64 characters (one screen line) in length. Comments should be preceded by a semicolon (;).

As mentioned, the Editor is screen oriented. Corrections may be made anywhere on the screen. They are recorded only when Carriage Return <CR> is pressed while the cursor is within the line being edited. The cursor controls (arrows and HOME key) operate in the normal fashion to position the cursor as desired. Also, the TAB key may be used to align the source text in columns. (The TABS are done using spaces so if you tab over text it will be erased. Use the cursor keys to skip over text without erasing). The text will scroll up and down should one attempt to move the cursor off the bottom or top of the screen, respectively. The cursor will not move above the top of text or below the bottom.

In addition to the cursor control keys, a series of Control Keys are available to facilitate rapid editing of the text. These keys are used by pressing the key labelled "CTRL" and the letter key at the same time. (The symbol ^ will be used to represent the CTRL key in this text). In order to make it easy to remember the control keys, a key word is associated with each function (key words are shown in capitals below). The functions available are:

^T... Go to the TOP of text and display the first page.

^B... Go to the BOTTOM of text and display the last line.

^N... Display the NEXT PAGE of text (the 30 lines following the line currently displayed at the bottom of the screen).

^I.... INSERT a line at the current cursor position.

^D... DELETE a line at the current cursor position.

∧L... Delete a character at the current cursor position. That is, move all characters past the cursor LEFT one space.

∧R... Insert a space at the current cursor position. That is, move all characters on the line from the cursor over, one space to the RIGHT. Characters at the end of the line are lost if pushed off the screen.

∧P... PRINT the source file at the printer starting from the current line pointed to by the cursor and finishing at the end of the file. The printout may be stopped before the end of pressing the Escape Key <ESC>.

∧F... Print the number of FREE BYTES remaining for use in the source file, object file and symbol table. It is up to the user to insure that the end of the text buffer is not overrun. Therefore, the free space remaining should be checked frequently when writing a large file.

∧X... This key initiates the Search and Replace (EXCHANGE) function. When ∧X is depressed, the message SEARCH FOR? will appear. You may enter, a string of up to 16 characters to be searched for. TABs are compressed to spaces. Once the search string has been typed you may press either <CR> or ∧R. If you press <CR> then only a Search will be done; ∧R allows you to enter a replacement string when the REPLACE WITH? prompt appears. Again, the string may be up to 16 characters. To abort the operation while entering strings, press ∧C.

Once the required strings have been entered, the search begins. When the search string has been located, the cursor will stop positioned on the first character. You may then choose one of the following options:

1) Press <ESC> to exit the mode whether you are in search mode (SM) or search and replace mode (SRM).
2) Press < SPACE> to skip over the string and continue searching in either SM or SRM.
3) Press <CR> to exit SM -or- to replace the string and then continue searching if you are in SRM.

∧C... Return to COMMAND mode.

If you remember the key words, you should be able to use the control codes without the need to constantly refer back to this manual.

4. THE ASSEMBLER

Command: ASMB <P> <N> <T> <L> <R> <O>

The ASMB command initiates the assembly of the source file currently stored in the text buffer. The assembler performs two passes, but since the text is in memory, the passes are performed fairly rapidly. The options (P, N, T, L, R, O) entered after the command indicate whether the output is to be printed out on a printer, whether the file is to be "linked" to a previous one, etc. Before describing the options in detail, the Assembler format, conventions and pseudo ops will be explained. Please note that this is not intended to be a course on assembly language. The reader should refer to one of the many excellent texts now available which cover this subject. The op-codes

considered valid by this Assembler may be found in the back of the "Guided Tour of Personal Computing" manual which came with your Sorcerer.

4.1 Assembler Format

The Editor provides four columns of text with the <TAB> key. The first column, beginning at the left of the screen (column /) is the Label Column. This column may contain a label of up to 6 characters. Valid label characters are upper and lower case alphabetic characters (A-Z, a-z) and any of the following 10 characters. < = > ? @ [ ] ^ ' |

»Note: The word "symbol" as used in this manual refers to any string of up to 6 characters in length used to represent a numerical constant or memory address. The word "label" refers to any string of up to 6 characters in length which appears in the label column of the source file. Thus, a "label" is a particular type of "symbol".

The next column (beginning in column 8) is the Op-Code Column. This column holds op-codes, such as RET, LD, and ADD. The next column (beginning in column 13) is the Argument Column. It holds the arguments related to the op-code. The final column (beginning in column 29) is the Comment Column. Actually, comments may begin in any column desired so long as the comment is preceded by a semicolon, as mentioned previously. The TABs between columns are compressed down to one space so that the text is stored in a compressed form in memory. For this reason, in certain cases, it is not possible to begin comments in ANY column desired. This will be discovered by experimentation.

The following diagram shows the column layout and some example text which might be placed there. The "*"s represent columns which should be left blank if they are not part of a comment:

```
LABEL  *OP * ARGUMENTS *                COMMENTS
       *CODE
VALUE1 PUSH DE                          ;SAVE IT
       XOR A
       LD D,A
       INC HL

; THIS SUBROUTINE RETURNS THE VALUE OF A MATHEMATICAL
; EXPRESSION INCLUDING HEX, BINARY AND DECIMAL NUMBERS
; AND CHARACTERS ENCLOSED IN DOUBLE QUOTES
; DETERMINE TYPE OF NUMBER AND GO TO APPROPRIATE
```

4.2 Assembler Conventions

The ZETU assembler is capable of handling hex, binary and decimal numbers as well as ASCII codes for data enclosed in double quotes. The different number bases are flagged to the assembler by placing a special character in front of the number. The default is decimal (i.e. a number with no leading character is assumed to be base 10). The special characters are:

flags a hexadecimal number of up to 4 digits. (eg/ #E003)

% flags a binary number of up to 8 digits. (eg/ %10010)

! flags a decimal number not greater than 65,535. (eg/ !25)

In addition to these flags, the dollar sign "$" may be used to represent the current value of the program counter (PC) during assembly.

A character enclosed in double quotes is evaluated as its associated ASCII value.

"A" = #41 = !65
"'" = #22 = !34
"" = #0 = !0

Note that 3 quotes in a row represent the character " and two in a row may be used to represent a null (0). Only one character may be used as an argument in quotes unless a string is being defined (see section 4.3).

The assembler is capable of performing addition and subtraction only. Division and multiplication are not accepted. The expression may contain any desired mix of hex, binary, decimal, $, ASCII and Symbols (Labels). The assembler checks for overflow when adding two numbers but not when subtracting. Thus, one must be careful not to add a number to small negative numbers since this may produce an unnecessary overflow condition. The following are examples of valid and invalid expressions.

VALID

```
LABEL +#A
#E003+40
-%111
$+#B0
```

INVALID

```
-1+#80 (this will cause an overflow)
#A- SYMBOL (don't leave spaces around "#" or "-")
```

Certain op-codes in the Z80 instruction set have arguments which consist of fixed numeric data. These include the **BIT, SET, RES, IM** and **RST** instructions. For such instructions, ZETU expects the numeric arguments to always be in a particular format and thus, a leading flag to indicate the number base (hex, binary, decimal) should NOT be used. For the **BIT, SET, RES** and **IM** instructions merely use the desired decimal digit as:

```
BIT 5, B
RES 0, A
IM 1
```

A leading "!" will produce an error message. For the **RST** instructions, use one of these hexadecimal arguments (0, 8, 10, 18, 20, 28, 30, 38) without the leading "#" as follows:

```
RST 0
RST 10
```

The interrupt vector register is represented as IV in ZETU. This applies to the two instructions:

LD IV, A and LD A, IV

The default displacement for instructions involving the index registers (IX, IY) is 0. Thus the following instructions are equivalent:

LD (IX), C = LD (IX+0), C
SET 3, (IX) = SET 3, (IX+0)

However, it is usually better for the sake of clarity to include the "+0" in these instructions.

4.3 Pseudo Operations

ZETU provides 8 pseudo-ops which allow you to do such things as reserve memory in your program or link modules together through globally defined labels. The pseudo-ops are placed in the op code column of the source file.

ORG

This pseudo-op allows the user to set the starting address (ORiGin) for an object module. Object modules are always assembled in memory directly following the source file but they are assembled so as to run properly when reloaded at the ORG address. The default is #0000 if no ORG is given or the value of the PC at the end of assembling the previous module if a LINK command is issued (see section 4.4). The argument for an ORG pseudo-op must be a constant or a symbol/label which has been previously defined. Although it is possible to use more than one ORG statement in a given module, this is not a good idea since the assembler will not keep track of where one section ends and the other begins (see SOBJ).

EQU

This pseudo-op allows the user to assign a value to a label or to EQUate two symbols. The format for the use of EQU is:

LABEL EQU ARGUMENT

where ARGUMENT is any valid arithmetic expression. For example:

HERE	EQU	THERE	assigns HERE with the value of THERE
FOUR	EQU	4	assigns the value 4 to the symbol FOUR
MIXUP	EQU	$-#10	assigns MIXUP with the value of PC-16

DEFB

This is the DEFine Byte pseudo-op. Its argument is a mathematical expression which must reduce to a number from 0 to 255 (#0 to #FF). The argument is evaluated and then stored as the next byte of the object module. Only one argument is valid per DEFB instruction. Examples:

```
        DEFB    #B0
SUFFR   DEFB    %11+200
        DEFB    "&"
```

DEFW

DEFine Word pseudo-op. This instruction is the same as DEFB except that the range of valid arguments is #0 to #FFFF (ie a two byte number). The number is stored in the next two bytes of the object module in the format Low Order Byte. High Order Byte as per the standard Z80 format.

DEFS

This pseudo-op allows the user to DEFine a String and store it as part of the object module. The ASCII codes are stored consecutively starting at the current available memory location. Examples:

```
        DEFS "This is a string"
QUOTES DEFS "Say " "hello" " loudly"
```

Quotes signs may be put in the string by using double quotes as shown.

RSRV

This instruction is used to ReSeRVe a block of memory within the object module. The argument of RSRV, when evaluated, must have a value from 0 to 255. Larger blocks may be reserved by repeated use of RSRV. The block of memory will contain the same GARBAGE that was in it at the time of assembly, so beware!

GBL

This instruction is used to define a symbol as GloBaL to all subsequent source files assembled. As each file is assembled, a symbol table is compiled. This table contains all symbols and their associated values. In addition, each symbol is followed by a flag indicating whether the symbol is global or local. At the end of assembly, all local symbols are purged from the table to make space for any files which might subsequently be linked to the current module. The GBL sets the flag to indicate that the symbol should not be purged and thus, it remains defined for use in subsequent modules (so long as the modules are LINKED using the L or R options). GBL is used just like EQU as shown in the following examples:

```
FOUR GBL 4
ERROR GBL #FF
```

If the label represents the start of a subroutine or a jump address (CALL ADDR or JP ADDR) then it should be set using the $ symbol as:

```
ADDR GBL $    ;Start of subroutine
     PUSH HL ;Remainder of subroutine
```

LCL

For very long programs, global symbols may collect in the symbol table until very little memory remains for assembly of object modules. If some global symbols are not required any longer, they may be purged from the table by

resetting their flags to indicate that they are local symbols. The LCL pseudo-op performs this function. At the end of assembly, any global symbols marked as LCL will be purged. For example, if ADDR is a global symbol, the instruction:

```
LCL ADDR
```

will cause it to be purged at the end of assembly.

4.4 Assembler Options and Operation

When the assembler is invoked with the ASMB command, the assembly process begins. On the second pass of assembly, a listing of the assembled program will be output to the screen or printer, if desired. After the listing, the symbol table is listed with the symbols sorted alphabetically by their first character only. The symbol table is always printed out; there is no option to suppress it. Should an error be found during either pass, the listing will terminate, an error message will be printed (see Appendix D), and the offending line of text will be output. Operation automatically transfers to the Editor so that you may quickly correct the error and reassemble the program. Only when the file may be properly assembled need the user save the file on tape. This saves time since the file need not be recorded and re-recorded every time an error is found.

The assembler options mentioned at the beginning of this section will now be explained. To use multiple options, one need merely type the capital letters associated with each option separated by spaces. Certain options should not be used in combination, as will become obvious.

P This initiates the PRINT option and causes the output text to be sent to the printer. The assembler is currently set up to operate with a Centronics compatible printer attached to the parallel port. See Appendix F instructions on altering the printer driver for a serial printer or for special initiating control codes.

N This option indicates that NO LISTING is to be output. The symbol table is still printed at the end of assembly, however. This option speeds up the debugging of source text by allowing the second assembler pass to be completed more rapidly. This option may be used with a new source file to quickly test if the file may be assembled without any errors. It is also useful if the user only desires an output of the symbol table.

T Normally, only the local symbols are listed in the symbol table. This saves the user from obtaining repeat listings of global symbols in hardcopy printouts of listings of linked modules. If the user wishes to have the global symbols output as well, the T (long TABLE) option should be used. Global symbols are flagged by a G besides them in the output table.

L This is the LINK option. It indicates that the current module should be "linked" to the previously assembled modules. What this means is that the assembly process takes up where it left off on the last module. The current

module will start at the address at which the last module terminated, the line numbers for the listing will continue from the last used value and the global symbols in the symbol table will continue to be defined for the current module. (NOTE: if neither the L or R options are present in the ASMB command, the symbol table will be TOTALLY PURGED before assembly. All symbols, whether global or local, will be lost from the table).

R This is the REDO LINKING option. It is used when it is desired to redo the assembly of a module which you are trying to link to previous modules. For example, if an error was found while assembling a linked module, the error could be corrected and the module reassembled using the R option rather than the L option. The L option should only be used for the first attempt at linking. Using the L option when re-assembling the module would cause the start address of the module to be the address at which the assembly process aborted on the first attempt. Also, it is necessary to purge the symbol table of symbols defined in the module before the assembly process aborted. Otherwise, the assembler will consider these symbols as ALREADY DEFINED and will print error messages. The R command performs the required purging process before re-assembling the module.

0 This is the ZERO option. When modules are linked, very often one module will contain a reference to a label in a module which has yet to be assembled. Obviously, if the module has not been assembled, the assembler will not be able to resolve the label (assign its value). In such a case, it is necessary to repeat the loading and assembling of all modules in the program a second time (see Section 10). When the first module is reloaded and re-assembled, it is necessary to use the L option so that the global symbols will not be lost. The 0 option is then used to reset the assembly address and line number to 0000 for the "second pass" of the assembler. This second pass is only necessary under the conditions stated above: a reference is made from one module to a label in a module which has yet to be assembled.

The L, R and 0 options may seem a bit confusing at this time. A run through the sample assembly in Section 10 should clear up many uncertainties and will explain the easiest method of linking together large programs.

5. SAVING A SOURCE FILE

Command: SSRC NAME (2)

The source file currently stored in the text buffer may be saved on tape at any time using the SSRC command. The file name must be specified. The default tape unit for saving is unit one, but unit two may be used if the optional 2 is added to the command. Start the tape on record before pressing CR . The required block of memory will be saved and then, when saving is complete, the command prompt will reappear. The file type of a source file is set to "S" so that it may be differentiated from an object module on reloading.

6. LOADING A SOURCE FILE

Command: LSRC <NAME>

The LSRC command is used to load a source file into the text buffer from tape. If the NAME is not specified, the first file found will be loaded. Source files may only be loaded from tape unit one. Any text currently in the buffer when a LSRC command is issued will be overwritten and lost. Should a TAPE CRC ERROR occur and cause you to exit ZETU to the monitor, ZETU may be restarted without the loss of any information by typing GO 0 <CR>.

7. MERGING SOURCE FILES

Command: MSRC NAME

The MSRC command allows a source file on tape to be merged to the end of source text currently in the text buffer. The NAME of the file must be specified and the file must be loaded from tape unit one. Care should be taken to insure that there is adequate memory remaining to store the text in before merging files since the program does not verify this. If the file is too long, it will overwrite the monitor stack and the program will crash. You should never be operating that close to the limit of the storage capacity anyway since you cannot assemble the file if there is not enough room to store the object file and symbol table.

8. SAVING OBJECT MODULES

Command: SOBJ NAME <2>

This command has the same format as that for saving a source file. The NAME must be specified and the unit may be one (default) or two if a 2 is added. This command should be used after assembling a source file. The program automatically determines which block of memory should be saved. The file type is set to "0" to indicate that the file is an object module. The GO ADDRESS in the file header is set to the ORG address of the module (so long as the ORG was at the beginning of the source module or there was no ORG in the module). Therefore, if you forget where the module should be loaded in order to run properly, you need merely look at the file header. The GO ADDRESS is then the address at which the file should be loaded. For example, the following file header:

```
NAME FI BLK ADDR GOADDR
ZETU 0 112E 457F 0500
```

Indicates that ZETU is an object module which was assembled to load at 0500 hex.

Object modules are loaded and, if necessary, linked into one unit by loading the modules into memory using the Sorcerer's Monitor and its standard LO command. Merely load the modules one by one at their proper ORG address and then resave them as one unit with the monitor SA command.

9. EXITING TO THE MONITOR

Command: EXIT

This command results in a jump to the Exidy Monitor. To restart ZETU simply type GO 0 <CR>.

APPENDIX A
Command Summary

EDIT <C>

Edit the source text
C = clear buffer

ASMB <P> <N> <T> <L> <R> <O>

Assemble a source file.
P = list at printer
N = no listing (only symbol table)
T = long symbol table (global symbols included)
L = link module to previous modules
R = redo assembly and maintain link to previous modules
O = zero line count and address

SSRC NAME <UNIT>

Save the source file

LSRC <NAME>

Load a source file

MSRC NAME

Merge a source file on tape with one in memory

SOBJ NAME <UNIT>

Save the object file

EXIT

Jump to the Monitor program

APPENDIX B
Editor Control Keys

^B Bottom of text
^C return to Command mode
^D Delete a line
^F print remaining Free space
^I Insert a line
^L delete a character
^N Next page of text
^P Print file at printer
^R insert a character
^T Top of text
^X search and replace text

APPENDIX C
Assembler Pseudo-Ops

ORG ADDR

Assemble program to load at address ADDR

LABEL EQU EXPRESSION

Assign LABEL with the value of EXPRESSION (where EXPRESSION is any valid combination of symbols, $, and hex, binary, decimal and ASCII constants).

DEFB EXPRESSION

Store one byte of data with value EXPRESSION; where EXPRESSION is as above and is a one byte value (ie/the high order byte is 00 or FF).

DEFW EXPRESSION

Store two bytes of data with value EXPRESSION; where EXPRESSION is as above but may be any two byte value.

DEFS STRING

Store a STRING in memory; where STRING is a series of characters enclosed in double quotes (").

RSRV EXPRESSION

Reserve a block of memory of size EXPRESSION; where EXPRESSION is evaluated as a number from 1 to 255.

LABEL GBL EXPRESSION

Assign LABEL with the value of EXPRESSION and mark LABEL as a global symbol, not to be purged from the symbol table.

LCL LABEL

Set the global/local flag associated with LABEL in the symbol table to local so that it will be purged from the symbol table.

APPENDIX D

Error Messages

Command Mode Errors:

INVALID COMMAND
The command is not a recognised key word. Check spelling and remember that all commands must be in capital letters.

INVALID PARAMETER
An invalid option was specified for a command. For example, EDIT G would result in this message since C is the only valid option for the EDIT command.

MISSING FILE NAME
A file name is required for saving source or object modules or for merging source files (SSRC, SOBJ and MSRC).

INVALID FILE NAME
An invalid character (invalid characters) was (were) included in the file name specified in an SSRC, LSRC, MSRC, or SOBJ command. Alphabetic characters and numbers are valid.

Assembler Errors:

INVALID LABEL
An invalid character appears in the symbol in the Label Field of the faulty line of source code (see Section 4.1).

LABEL ALREADY DEFINED
The symbol in the Label Field of the faulty line has already appeared as a label elsewhere in the source file. Use the Search and Replace function of the Editor to determine where the label appears or check to see if the label is defined as global in another module. Change the label if necessary.

LABEL NOT DEFINED
The label appearing in the line of source text has not been assigned a value by the time it is encountered on the second assembler pass.

MISSING LABEL
There is no label in a line of source code which requires one (ex/ an EQU statement). Add the required label.

SYMBOL TOO LONG
A symbol (or the label) in the line of source code consists of more than 6 characters or is interpreted as such. Check the length of all symbols and make sure that plus (+) or minus (-) signs have not been forgotten in arithmetic expressions.

INVALID SYMBOL
A symbol (NOT the label, if there is one) appearing in the line of source code contains one or more invalid characters (see Section 4).

SYMBOL NOT DEFINED
A symbol (NOT the label, if there is one) appearing in the line of source code has not been assigned a value by the time it is encountered during the

second pass of the assembler. If the symbol is defined as global (GBL) in another module, put a SYMBOL EQU $ instruction at the top to the current module to give it an artificial value until it becomes defined later.

JUMP OUT OF RANGE

The line of source code contains a relative jump to an address which is outside of the allowable range of relative jump locations. Change the jump instruction to a direct jump (JP).

ARGUMENT OVERFLOW

An argument of the instruction is out of the allowable range of values for the given instruction (ex/ the argument of a DEFB instruction is greater than 255 decimal).

INVALID ARGUMENT

An argument in the instruction is not valid for the given op code (ex/ PUSH AB).

MISSING QUOTE SIGN

Quotes are missing around the string in a DEFS instruction or closing quotes are missing around an ASCII constant.

MISSING COMMA

A comma to separate two arguments is missing (ex/ LD (IY+5) A)).

MISSING BRACKET

A bracket surrounding an argument is missing (ex/ LD (IY+5, A)).

INVALID HEX DIGIT

The assembler was expecting a hexadecimal digit but did not find one (ex/ LD A,#4G).

INVALID BINARY DIGIT

The assembler was expecting a binary digit but did not find one (ex/ LD A,%50).

INVALID DECIMAL DIGIT

The assembler was expecting a decimal digit but did not find one (ex/ LD A, 4F). Decimal is the default base. Make sure you have not accidentally forgotten to insert the proper base flag (see Section 4.2).

INVALID OP CODE

The assembler could not interpret the line of source code. Check the spelling of the op code in the line. Also, check the arguments. In some cases, unexpected arguments will produce this error message.

SOURCE FILE TOO LONG

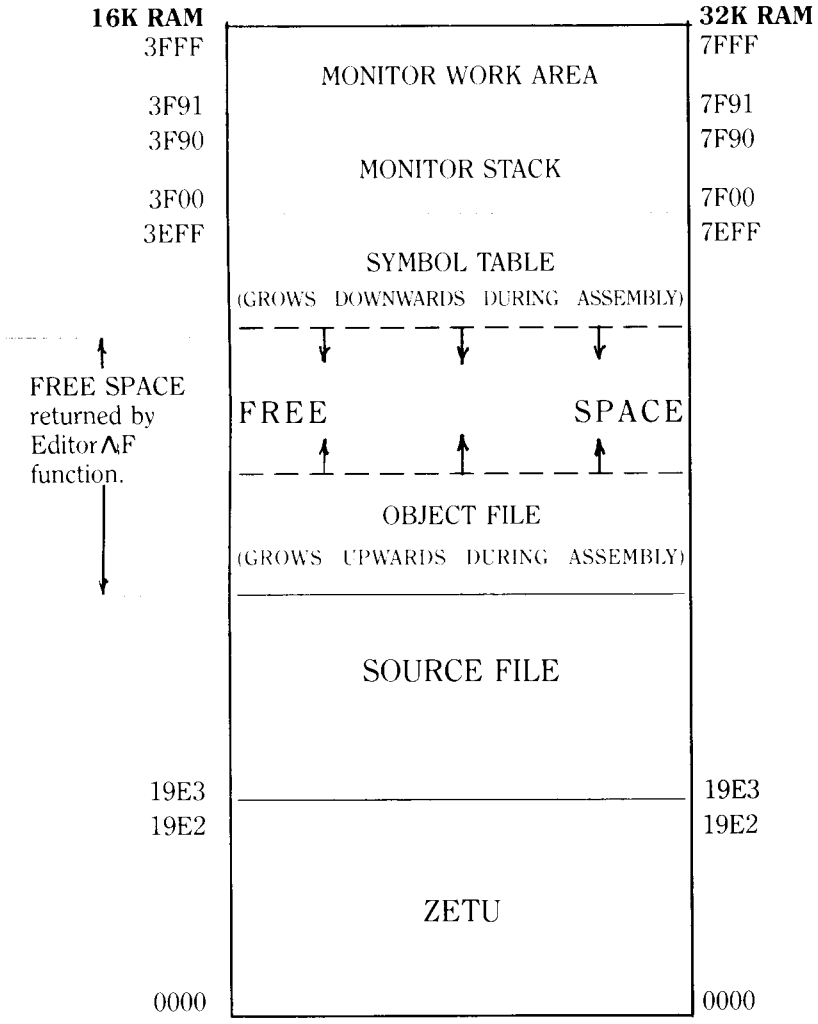
While the assembly process was taking place, the symbol table and object module overran each other (you ran out of memory). Shorten the source file or split it into two or more pieces. The line listed will give you an idea of where the assembler was when it ran out of space.

SYNTAX ERROR

All purpose error message to cover errors not dealt with by the messages given above. This may be something like a double bracket at the end of an instruction. For example, IN A, (#FF)) would produce this message.

APPENDIX E

Memory Map



APPENDIX F

Printer Driver Customization

As delivered, ZETU will operate with a Centronics compatible printer hooked to the parallel port of your Sorcerer. The printer activate/ deactivate routine is capable of sending 3 leading control characters before commencing printout and 2 before ceasing printout. This section will describe how to go about adding your own custom printer driver to ZETU and how to change the control characters.

ZETU contains a serial printer driver routine at 8BIH. You may use this routine or insert one of your own. The space allocated for your driver routine is from 8BIH to 8E0H, inclusive (a total of 48 bytes of memory).

After assembling your driver using ZETU, load it starting at 8B1H.

Using the Monitor (the ENter command), load the following hex values into the designated memory addresses.

```
06B6: 58
06B7: 11
06B8: B1
06B9: 08
```

The above procedure causes your alternate driver routine to be used for output to the printer.

The addresses where the control characters are stored are listed below. Obviously, these may be changed to suit your printer even if you do not need to change the printer driver routine as outlined above. To change them, merely use the Monitor `EN` command to load the desired values into the given addresses.

Pre-printout control code addresses:
`06BE, 06C3, 06C8`

Post-printout control code addresses:
`06D2, 06D7`

Be careful when making these changes since you are changing ZETU itself. A mistake could cause the program to crash when run.

Once you have finished making any of the above changes, record a new copy of ZETU (`SA ZETU 0 19F4`) to permanently record the changes you have made.

If you have found this piece of software valuable, you may wish to be included on our free mailing list and receive literature on all new releases. Write to:

SYSTEM SOFTWARE
1 Kent St., Bicton
Western Australia 6157