LERM Z80 TOOLKIT v2.

CONTENTS

LERM Software

11, Beaconsfield Close,
Whitley Bay, Tel: 091 2533615.
Tyne and Wear NE25 9UW.

SIDE 2:CTOS code is at 27000, length 5479
TLKT HIGH is at 57200, length 8238
TLKT LOW is at 27000, length 8238

Should any user require a copy of the TOOLKIT that loads to any other address in the range 24000 to 57200, then just send £2-00 plus your original tape plus a SAE and the details of the load address that you require. The Assembler cannot be supplied at any address other than the one that it is already at.

LOADING THE TAPE

Rewind the tape to the start of the side of your LERM tape.
SIDE ONE: contains FULL program
Type LOAD "" start the tape.
This loads in the ASSEMBLER and TOOLKIT. After loading, you are asked to STOP THE TAPE. If you want to load in CTOS as well you just press the "y" key and start playing the tape again. You will then have all 3 programs in memory. If you press "n" for no, then you will have the ASSEMBLER and TOOLKIT In memory. To enter the ASSEMBLER press "A", the TOOLKIT press "T", for CTOS if loaded press "C", or press the "B" key to return to BASIC.

SIDE TWO: CTOS on its own, plus TLKT high and low versions.
To load in CTOS on its own, rewind the tape to its start, then enter LOAD "", and wait until loading is complete. Now follow the instructions, given in the section "CTOS MANUAL".

To load the high version of the Toolkit type LOAD "TLKT HIGH", then ENTER, and play the tape.

To load the, low version, type LOAD "TLKT LOW" then start the tape.

PART A - USING THE ASSEMBLER

The assembler is a two pass Assembler, and it works on the principle of a word processor. Full cursor control is available to edit or modify the text as you write it. A summary of the key controls are at the end of this manual. It also accepts multi-line statements like BASIC, separated by the ":" sign.

e.g. 00010 LD A,0:LD B,5:LD HL,30000

It automatically has CAPS LOCK ON. To type the lower case HOLD DOWN the CAPS-SHIFT KEY - i.e the total reverse of normal.

NOTE for those with the ZEUS Assembler:
Any user with ZEUS Assembler source files will be happy to note that this Assembler will accept those files, with just a small modification. The only thing to be modified once the source has been loaded are the labels and any DISP (displacement value) that has been used. All labels must be shortened to 6 bytes only, plus any DISP must refer to the exact address that the user wishes their code to be assembled at.
e.g DISP 40000 in ZEUS meant that the assembled code would be placed at the ORG address with 40000 added on. In Z80A it would be placed at 40000.

DEFINITIONS.

SOURCE FILE. All typed text including opcodes labels and any of your own comments.

ORJECT CODE. This is the machine code produced by the assembler

LABEL. A pointer to an entry point in the source code.

DIRECTIVE. These are special instructions to the assembler and only tell the assembler to do some particular job, they will not be assembled as machine code.

SYMBOLS. A label that has had a numeric value given to it at assembly time.

Don't worry too much about these terms for the time being. We hope that they will become obvious through our examples.

ASSEMBLER INSTRUCTIONS.

The below are all DIRECTIVES as they tell the Assembler what to do, and are not part of the assembled code.

ORG address This will tell the assembler to assemble the code for a particular address, and unless a DISP is used the object code will be assembled to this address. eg: ORG 40000 means assemble the code to run from 40000, and place it at 40000.

DISP address This tells the assembler to put the code at the DISP address but to assemble it for the ORG address, (code must be saved from the DISP address and loaded back to the correct ORG address). A DISP of zero will default to the ORG address. (e.a ORG 60000, DISP 0 means code is assembled at 60000). This is a useful command if the code you are writing needs to be assembled into any address that is currently occupied, ie: over the top of the assembler.

ENT Used if you should want to run the assembled code from within the assembler. It should be put in a line on its own in front of where you want to run from. Once the code has been assembled typing X plus ENTER, will run the code. All source code should be saved to tape before using this option in case of a crash.

EQU Means Equate or equals. A label may have a value assigned to it. Useful If you want to refer to a point outside of the source code. This takes the form LABEL EQU VALUE or PRINT EQU 00016

DEFB nn,nn DEFINE BYTES, Inserts 8 bit data at the current address separated by a (,).
eg: DEFB 22,10,10

DEFW. DEFINE WORDS, Inserts 16 bit data (addressees) at the current address separated by a (,).
eg: DEFW 16384,6912

DEFM. DEFINE MESSAGE. Insert the string at the current address enclosed in quotes.
eg: DEFM "example"

$ This will tell the Assembler to use the current assembly address. eg: LD B,20
DJNZ $
In other words loop back to the instruction DJNZ.
This could have been written as LD B,20
LOOP DJNZ LOOP

To learn how to use this program we will use an example. If you haven't already done so, then LOAD the program Z80A. Once loaded you will be presented with a menu, pressing key "A" will enter the assembler.

LINE NUMBERS.....
Like any basic programming, line numbers are needed. This can be done manually by typing in the line number then a space. The program will produce them for you - to do this type the letter "I" plus ENTER. A 00010 will appear on the screen, if you now press ENTER again the cursor will drop to the next line and a 00020 will appear, this is your auto line number mode. So the "I" key is a DIRECTIVE and tells the assembler to automatically give line numbers starting at ten, in steps of ten.

CLEAR SCREEN.....
OK now press caps shift and the "9" key, this is the clearscreen key. Type "I" once again and you will be in auto line mode again.

TABULATE.....
Now press capshift and the number key "2". The cursor will move across the screen, this is the tabulate key and saves you typing spaces. Now type in the example from listing one, not forgetting the tabulate key. (do not type in the line numbers). ENTER must be pressed after the entry of every line, and no line must be allowed to continue onto the next line. If this does happen then the line or lines will not be entered into the file.

LISTING 1.

```
00010         ORG 40000
00020 START   LD B,245
00030         LD HL,22528
00040         LD A,0
00050 LOOP    LD (HL),A
00060         INC HL
00070         INCA
00080         DJNZ LOOP
00090         RET
00100 END     EQU $
00110 LENGTH  EQU END-START
00120 _
```

LISTING 2.

```
00010         ORG 40000
00015         ENT
00020 START   LD B,255
00030         LD HL,22528
00040         LD A,0
00050 LOOP    LD (HL),A
00060         INC HL
00070         INC A
00080         DJNZ LOOP
00090         RET
00100 END     EQU $
00110 LENGTH  EQU END-START
```

CLEARLINE.....
Now press capshift and the number key "1", the line containing the cursor will disappear (00120 >), and so CLEAR the line. Press ENTER now and you will be back into command mode. So to CLEARLINE simply place the flashing cursor somewhere on the line and use CAPSHIFT "1". This does NOT delete the line.

CURSORS.....
If you now use your cursor keys you will be able to move the cursor to any position in the text that you wish, so move the cursor into line 00020 and alter the number to 255 eg:

```
00020 START   LD B,255
```

Once done press ENTER until the cursor is below the text.

ASSEMBLING.....
Now press key "A". This will attempt to assemble the code to

address 40000. If you have typed exactly as the text including the error on line 00070, then the message "Error 0" will appear, and the line that the error is on will appear below.

```
What you will see is :          00070    INCA
This is wrong and should read : 00070    INC A.
```

INSERT/DELETE.....

To rectify this, place the cursor on the A and press capshift key "4". A space will appear between the C and the A. If you now press capshift key "3", then the space will disappear again. So every time you press CAPSHIFT "4" a space is inserted, and CAPSHIFT "3" a character is deleted at the cursor position. The normal DELETE key works as in ZX BASIC. (e.g type 20 and ENTER to delete line 20). Put a space back into the line end just press ENTER until the cursor is below the text again.

LIST.....

Clear the screen (capshift 9) or (GRAPH on the plus 2). Now press the letter key "L" and ENTER. This will LIST the first part of the text again. Pressing ENTER will LIST the next page if present. Pressing the "A" key should now result in the cursor dropping below the "A" in the listing. If this is so then the code has been assembled with no errors. The OBJECT CODE is now at 40000, the address where you assembled it. What this piece of code will do is to poke the addresses from 22528 (the start of the attributes area) to 22784, with the value held in the A register. Later we will singlestep this piece of code and it will become a bit clearer, but for now it will colour the first third of the screen. Now press capshift "9" to clearscreen, then press "L," and ENTER to LIST again. Now type in at the cursor position `15 ENT`
and press ENTER. Clear screen and list again. You will see that line 15 has been inserted at the correct place, as in listing two.

RUNNING THE CODE.....

Assemble the code again by pressing the "A" key, and to execute the assembled machine code press "X" and ENTER. A pretty pattern will have appeared at the top of the screen. Good, this is as it should be. If not then clearscreen and LIST. Check that your text is the same as listing two. If not then edit the text using the cursors, pressing ENTER once each line has been corrected. You should be getting used to using the assembler by now so we will move on to some more useful commands. First clear the screen and LIST.

RENUMBER.....

There is a full renumber option within the assembler. If CTOS is present then RENUMBER is automatically DISABLED to stop you from accidentally RENUMBERING. To ENABLE the RENUMBER function from the ASSEMBLER press key "M", ENTER, then "R". To DISABLE simply repeat the instruction. Pressing "R" in the MONITOR of the ASSEMBLER (see later for explanation of monitor) switches the RENUMBER function off/on. If you try to RENUMBER with this function DISABLED you will get an ERROR message.

If the RENUMBER function is switched ON, just press "R" plus ENTER, clear the screen again, and LIST. Now you will see that the source file has been renumbered from line 10 in steps of 10,

line 15 is now line 20. This is the default, line 10 and steps of 10, you can give different values if you wish, try R 10 100 (plus ENTER). You should have line 10 as a start but the next should be 110. The first number after the R gives the first line number to be used, then, after a space, the second gives the STEP size. Try a few others and see for yourself.

SYMBOL TABLE.....
OK so now another function and this one will explain why we used labels START END and LENGTH. When writing machine code it is sometimes useful to know the addresses of certain pieces of code. It Is also nice to know how long a piece of code is, so pressing the "S" key plus ENTER, will produce what is known as the SYHMOL TABLE. The labels that you used in your source code, will be produced with the addresses that have been assigned to them on assembly. If you press "S" plus ENTER, you will see the symbol table appear starting from the last one entered. You will also see the label called LENGTH. It has a value inserted against it and this is the length of the object code (your machine code). Now clearscreen again and re-list, the list command can be given parameters, for instance L 10 50 (plus ENTER) will list lines from line 10 to line 50.

SAVING AND LOADING SOURCE OR OBJECT CODE..
Now to saving the source file and the object code. To save the source file you need to know where the source starts and how long it is. (Default start = 27000 or 33000 if CTOS is present). To find this out just press the letter "T" plus ENTER. The start of the source file and the length will be displayed. These numbers should be written down plus the start of the object code, and its length should also be noted. (from the symbol table).

EXIT TO BASIC.....
Once this is done then an exit to basic should be made. To do this key "Q" and press ENTER. This will take you back to the MENU. Now pressing key "U" will return you to basic. To SAVE the source you should type SAVE "name" CODE START,LENGTH. (e.g SAVE "source" CODE 27000,1000).
Start and length being the figures that you wrote down for your source. You should do the same for the object code. Always verify after saving. To LOAD any previous SOURCE, exit to basic as just described and just LOAD "" CODE and start the tape. Naturally you can redirect the destination of the source code - eg. LOAD "source" CODE 33000.

RE-ENTRY TO THE PROGRAM...
To re-enter the program just type RUN and enter.

RETRIEVE SOURCE.....
On entry to the assembler you will need to retrieve your source file and to do this you should press key "0" plus ENTER, and key "L" plus ENTER to list again. If you are just entering the assembler for the FIRST TIME then there is no need to type "O" or "L". In fact if you do the program may crash and you would need to reload it again.

Under normal circumstances the source code defaults to address

27000 but you can put it where you want as long as it is between Z7000 and 50000, although 50000 would be a bit high, as the symbol table grows downwards from 51000.
e.g. LOAD source code into 33000 from BASIC. ENTER the usual "run" command, and press key "A" to access the Assembler mode. Now press key "0", then a space, followed by 33000, then ENTER. You have now told the Assembler that the start of the source is at 33000 rather than the default value of 27000. If you re-enter BASIC, then go back to the assembler you must again begin by entering O 33000. Alternatively (see explanation below) you can alter the default to 33000 by pressing key "M", ENTER, key "O", and ENTER. This will continue to be the new default until the Z80A program is reloaded, or it is re-set.

CREATING A NEW SOURCE FILE.
To start a new source file just press key "N", then a space, then the address, and press ENTER. e.g. N 33000. Simply pressing "N" and ENTER starts the new source at the default value. Pressing "T" plus ENTER, will now give the start of the source code. If an exit to basic is made for any reason and a subsequent re-entry to the assembler, then to retrieve the source file you will have to press "O", then a space, then the address plus ENTER, before you try to list. Pressing "I" and ENTER gives the current default value.

As briefly described above, there is a way to change the default address, which remains changed unless the process is repeated or the Assembler reloaded. This entails going into the MONITOR mode.

ENTERING THE MONITOR..
To enter the monitor press key "M" and ENTER. If this is done accidentally then pressing "Z" and ENTER will return you to the assembler. Once in the monitor press "O" plus ENTER, and you will be informed of the default address on the screen, and once again returned to the assembler. By entering the monitor however the default value of the "O" key in the Assembler is changed to the default value for the start of the source code. You can change this by using say O 33000 (or N 33000), but the default remains at 27000 unless you enter the monitor ("M" and ENTER), then press "O" and ENTER in the monitor mode. Having used the monitor to redefine the result, any exit and re-entry to the assembler will now only need you to press "O" and ENTER to retrieve your source. There are other useful functions from within the monitor but we will come to those later.

PRINTER.....
To make a permanent record of the source file you would use a printer. To turn on the printer option press key "P", then the number key "1", then ENTER. i.e P1 ENTER
The printer is switched off using key "P", the number key "0", and ENTER. With the printer ON, all outputs to the screen and printer.
e.g switch printer on, then press "L" and ENTER. The first page of your source file will be sent to the printer. Pressing ENTER will print the next page, on so on.

If you ENTER the command L 200 900 10 plus ENTER, it will send from line 200 to 900 in steps of 10 to your printer without stopping.

FIND.....

Another useful function is the find option. If your source file is quite large, then knowing what line or lines hold a particular label is sometimes handy, so pressing F "label", and ENTER will find every occurrence of that label.
e.g to find all lines containing the statement "LD A,3" press
F "LD A, 3" then ENTER.
It will not give you the address of the label, but there is a monitor function that will give you that option. (after assembly only).

REMOVING LINE/LINES...

Sometimes a line or a block of lines need to be removed. To remove a single line just type the line number and press ENTER. To remove a block of lines then press "D", then a space, then the first line number, a space, then the last line number to be deleted, then ENTER.
eg. D 20 100 deletes lines 20 to 100 inclusive. Pressing "D" on its own will have no effect.

REMARKS OR COMMENTS...

A comment can be put on a line on its own or it can be put after the opcode, to do this use a ";".

```
00010 ; this is a remark
00020 START LD A,0: LD HL,START+10 ; this is the start
00030 ;
```

Line 10 contains just a remark.
Line 20 a label, plus an mnemonic (e.g the LD A,0),and a remark.
Line 30 creates a blank line useful for breaking up the text.
Remarks must not be allowed to overflow one line to the next.

LABELS.....

Labels can be any combination of uppercase, lowercase or numerals, but spaces may not be used between a label. The same name may be used but only if it uses upper and lower case characters differently. eg START,start,STARt,STArt,STart,Start,StArT,sTaRt. All these labels can be used in the same file although it could be a bit confusing. Labels can have an offset assigned to them.
e.g

```
 00010       LD (HL),255
 00020 LOOP  RET
 00030       LD A,0 ; this is a two byte instruction.
 00040       INC A
 00050       CP 255
 00060       JR NZ,LOOP+3 ; jump back to line 00030
 00070       LD (LOOP+2),A; modify the second byte in line 30
 00080       RET
```

In the above, line 60 has an offset of the number 3, and line 70 one of the number 2. Also the above example has used a label from a previous subroutine, so saving the need to use another label. Normally we would not do it exactly as shown but it does give you the general idea.

HEX OR DECIMAL INPUT..

The assembler automatically defaults to decimal numbers, but if you wish you can use HEX. HEX numbers must be preceded by a hash sign "#", the assembler will then interpret them.

NOTES ON ASSEMBLY AND THE SYMBOL TABLE...

Using the full Z80 toolkit minus CTOS then 23.5k of source can be typed in. Using the assembler on its own, this increases to 32k. If any text tries to overwrite other parts of the program then on pressing ENTER the line will not be accepted and you will be told "Out of memory". Similarly if the symbol table is about to overwrite the top of the source code on assembly, then you will be told "Out of symbol space". There are ways round both of these problems see the monitor section).

A USEFUL TIP:

We sometimes assemble our code into the ROM the first time round. To do this use your normal ORG but on the next line use a DISP statement and use any address ABOVE 0 (we use 1000).

```
 e.g   10 ORG 40000
       20 DISP 1000
       ....etc.....
```

This will assemble with no problems as you cannot overwrite the ROM, so no code will have been produced, but any errors in the source code will have been picked up and displayed. Also if the assembly was successful then a symbol table will have been produced, starting from just below where the Assembler lies growing down the memory (e.g 51000,50999,50998... to say 49010). This keeps the RAM free from any code until you actually want it to be put into RAM.

When assembling any errors that are displayed can be corrected on the actual display. You don't have to LIST the offending lines to do the corrections, but if more than one error is shown then after correcting the line and pressing the ENTER key, you should use the cursor key to move to the next line to be edited. Do remember that if you use a DISP in the ROM then there is no object code to be run, so DON'T press "X" if you have used an ENT in the source. The symbol table can be interrogated from within the monitor, but only after assembly. When all is well, save the source code, and alter the DISP to a place in RAM (say 40000 - must be somewhere unused by our Z80 program or by the source code!). You can then save the object code.

THE MONITOR

The Assembler has its own MONITOR. It consists of various routines to make programming a bit easier for you. To enter the monitor press "M" and ENTER, and the message Monitor will appear. All the routines will exit back to the Assembler automatically, except for the FIND routine. To exit the Monitor in case of accidental entry, then press "Z" and ENTER, and the message "Z80 Assembler" will appear. To obtain any of the options below press the key and NOT the brackets. e.g. To convert the number 145 into hex press key A then the number, then ENTER.

KEYS..... (complete commands by pressing ENTER).

(A) plus a number in the range 0 to 65535 will give you the HEX equivalent.

(A) plus hash and then a HEX number will give you the decimal equivalent.

(B) plus a number in the range 0 to 65535 will give you the low byte plus the high byte.

(B) plus hash and then a HEX number will also give you the low byte plus the high byte.

(C) Will toggle the screen colours from white screen black text to blue screen and white text. Using the "C" key again will restore the white screen.

(O) If after setting a NEW source position from within the Assembler, you wish to make this the default source position than pressing the "O" key will do this for you. You will be informed of the new default and returned to the Assembler.

(F) This is a search facility, but it works on the symbol table. It can only be used after the source code has been assembled. What happens is that the Assembler reads through your source text, and creates a table called the "symbol table" of all the values it calculates. This table grows downwards from just below the Assembler. It will produce the value that has been assigned to a specific label on assembly, and is useful if you wish to find the entry point to a particular routine in the object code. To use it type F" label ENTER. DO NOT put quote marks after the label, just press ENTER.

e.g To find the value given to a label called "FRED" press F "FRED then ENTER.
You will be given the address of the label in HEX and DECIMAL, but this routine will not return to the Assembler, so when you have finished just press "Z" and ENTER to return to the Assembler.

(L) Find and display the very last line number in the source file. Useful if you have just loaded the source and wish to continue writing your program from the very bottom.

(S) FOR LARGE SOURCE FILES ONLY: This routine will toggle the symbol table into the BOTTOM OF THE SCREEN or back to JUST BELOW THE ASSEMBLER. This is useful in the event that on assembly you are told "Out of symbol space". This will happen if the source code is about to be overwritten by the symbol table. We repeat again that the symbol table grows downwards towards the top of the source code, which grows upwards. It will only happen when the source code is quite large. Toggling the symbol table into the screen can be very helpful, but it means that when you press "S" plus ENTER, for the symbol table, you won't get any result.

BACK IN THE ASSEMBLER mode, you can, by pressing key "S",

space, then a number (say 3), get a listing just 3 values from the symbol table. So what we sometimes do is to press "S", space, then "3" and ENTER, as this is usually enough to give us the first 3 symbols. (e.g START END and LENGTH). If we need to know any more of the symbols, then we clear the top of the screen using the clearline key, enter the monitor and once again use the clearllne key to keep the top of the screen clear. Now we use the FIND option in the monitor, and if there is more than one label whose value we wish to know then we keep editing the label name and pressing enter until we have finished. All the time we are doing this the symbol table must be on the screen and we must not allow anything to corrupt it. It looks like a load of garbage, but it is data. Once we have finished, pressing "Z" plus ENTER, will take us back to the assembler. We can clearscreen and list, any code that we assembled is in the memory, but the symbol table is missing. If we exit to basic and then re-enter, the symbol table will default to the normal position just below the assembler. There is no message to tell you where you have put the symbol table.

(T) Plus an address will display the address and the value that is in that address plus any ASCII character. e.g press "T", space, then the number zero, plus ENTER so: T 0
This will clear the screen and display the following:

```
0 : 243       :
1 : 175       :
2 : 17        :
3 : 255       :
4 : 255       :
5 : 195       :
6 : 203       :
7 : 17        :
8 : 42        : *
```

and so on until the screen is full. The program will wait for you to press the SPACE BAR before displaying the next screenfull. To exit from this option press the SYMBOL SHIFT and the "A" key. The screen will clear and you will be back in the assembler.

(Z) Exit back to the assembler.

One extra special function of the monitor is the ability to reclaim the memory that the TOOLKIT (disassembler + single stepper) occupies. This is in case you should want only the assembler in memory so that you can write a large program without the toolkit being in the way. The symbol table pointer will be moved up in memory and the top of memory pointer will also be moved up. To use this option you will have to hold down the SYMBOL SHIFT key and press key "X", a "£" will be printed. Press ENTER to return to basic. What you should now do is to edit line 10 of the basic loader to read RANDOMIZE USR 59400. All that you need to do to enter the assembler again is to type RUN. Pressing the shifted "X" key once the TOOLKIT has been removed will he ignored. Removing the TOOLKIT will give you 32000 bytes of memory for source code providing CTOS is not in memory.

The TOOLKIT section of this program is approximately 8400 bytes long and was written with the above assembler. To do this the symbol table had to be toggled into the screen, There was nowhere to assemble the code because the source very nearly filled the machines memory. We assembled the code over the top of the beginning of the source code. To do this we had to use the correct ORG but then DISP of 24000 was where we had the source code defaulting to. If you use a source file below 27000 then don't forget to use CLEAR from BASIC. We used CLEAR 23999, as this is about as low as you can go. We had to have a few lines at the top of the source code that just had remarks in them, we also saved the source code before assembly. Once assembled we could not LIST as the object code was now where source had been. We had to exit to BASIC and save the object code from 24000 and then after NEWING the machine reload the code into its proper ORG address to test it. Of course if the ORG had been made 24000 and no DISP had been used then we could have tested it straight away. We only tell you this so as to give you an idea of how to overcome problems when writing large programs.

SAVING THE ASSEMBLER PROGRAM ONLY .....

A copy of the assembler without the toolkit can be made if you want one. To do this LOAD the full program and then use the Monitor function to remove the TOOLKIT, as stated above (use SYMBOL SHIFT and "X" in monitor mode, then press ENTER).

Once in BASIC write the following:

```
10 PRINT USR 59400:CLS:STOP
20 CLEAR 26999 : LOAD "ASSEMcode" CODE: GOTO 10
30 SAVE "ASSEMBLER" LINE 20 : SAVE "ASSEMcode" CODE 59400,6026
```

(alter as appropriate if you have a disc or other drive system)

Put a fresh tape into your tape recorder, type GOTO 30 and press ENTER, start recording and press ENTER when prompted. A copy will be made for you. Before making a copy, the source code default can be changed, the colour of the screen can be changed as well and these will both become the default values of your copy. You have thus saved a CUSTOMISED version of the Assembler.

SAVING THE ASSEMBLER AND CTOS.....

Load in from TAPE the full Z80 Toolkit, including CTOS by pressing "y" when prompted (see LOADING THE TAPE instructions). Enter the ASSEMBLER by pressing the "A" key, then the MONITOR by pressing the "M" key, and then press SYMBOL SHIFT and the "X" key TOGETHER. You will now be back in BASIC, so EDIT LINE 10, and INSERT LINE 15 so:

```
       10 CLS: POKE 27161,0: RANDOMIZE USR 27000
       15 PRINT USR 59400:CLS:STOP
```

Change LINE 120 to

```
       118 GOTO VAL"15"
```

You can now save the BASIC and CODE so:

```
       SAVE "ASS+CTOS" LINE 20
       SAVE "CTOS CODE" CODE 27000,5479
       SAVE "ASSEM CODE" CODE 59400,6026
```

(alter LINE 20 as appropriate for your disc or other systems)

PART B - THE TOOLKIT

The TOOLKIT is entered by pressing the "T" key from the MENU once program has been loaded. The two copies of the TOOLKIT on SIDE 2 (high and low versions) on their own, don't have a menu. You will be presented with a screen display that shows all of the registers. At this point the program is waiting for you to give it an address, typing ENTER will default to zero. If you still have the previous example from the assembler in memory then type 40000 and press ENTER. The input line will disappear and the opcode line will display

```
40000 : ld b,245
```

If you want the display in upper case then just press the "U" key to toggle upper or lower case opcodes. You can also change the decimal numbers to hex by pressing the "H" key, both of these keys will toggle the display. No harm will come to the program being single stepped. If you choose to have the display in HEX then all subsequent input values will also have to be input in HEX unless you change back to decimal first. If you now press the "Z" key the display will change to the disassembler mode, the hex or decimal rule still applies, the same toggle keys are available from this mode.

Pressing the "Z" key again will take you into the loader mode (see later).

Entry to the disassembler will take the value from the opcode line in the singlestep mode, and start disassembly from that address. The same goes for the loader mode, the value is the address of the first instruction on the screen. While in either singlestep or disassembler mode, pressing "D" will display from the current address at the top of the screen, an ASCII dump. Exit from all of these modes is via the "Q" key.

Below are listed all the key presses and their uses from the various modes.

SINGLESTEP KEYS .....

(A) Pressing this key will prompt you for a new address, ENTER will default to address zero.

(B) Exit back to basic, this is prompted with a Y or N, (only the two toolkit programs on their own) - see (M)

(C) The space bar is normally used to singlestep, but pressing "C" will auto step until you press the space bar again to stop - useful if you have the printer turned on.

(D) This will produce an ASCII dump of the memory from the current address. Exit is via the "Q" key. Any value outside of the ASCII characters is displayed as a space character.

(E) This will toggle the alternate registers, a message is displayed at the bottom of the screen to remind you. The values can be changed by toggling the alternate registers to the top section of the screen, and once satisfied then toggle them back again.

(H) Toggle the display into hex or decimal, all input modes will default to the current base. The default will carryover to the other modes.

(L) Remove the LOGO from the top of the screen. Sometimes it is useful to have the top line free of anything, so pressing the "L" key will remove the logo, pressing the same key again will restore it.

(M) Exit back to the main menu (only the full version Z80A). You will be prompted with a Yes or No.

(O) This retrieves the last address that you entered in the input mode. It saves going into the input mode, all of the registers other than IY and SP are zeroed.

(N) This is the same as the above, but it leaves all of the registers intact.

(P) This turns the printer ON or OFF, a message is displayed at the bottom of the screen to remind you that its on. If the printer is on when you enter the disassembler then it will automatically be turned off. The same is true if you exit the disassembly mode while the printer is on. This program will output to a ZX type printer or a full size printer.
If your printer requires setting up by loading in some machine code, or using some BASIC commands, then this should be done before entering the MAIN MENU.

(Q) Pressing this key will allow you to change the values in any of the registers. Input must be in the base chosen (hex or decimal). To change any register just press A,B,C,D,E,H,L,X, or Y. The appropriate register will flash and you can type in your new value. Pressing the delete key (capshift zero) will erase the complete input, just press enter when you are satisfied.

To change a 16 bit register. i.e BC,DE,HL,SP, then press capshift plus one of the following :
B = BC, D = DE, H = HL, S = SP. The 16 bit register on the screen will flash, in all cases pressing enter will default to zero. Pressing "P" will allow you to change the value held at the current stack position (this is not a push or pop). Once again press enter when you are satisfied. To change a flag value (either a 1 or 0), press "F". You will be prompted for a particular flag. Typing "A" will reset all flags back to zero, otherwise the chosen flag will just toggle and you will be taken straight back to singlestep. To change any value in the alternate register set, then pressing "E" and toggling the flags before pressing "Q" will allow you to do this. Pressing enter before a register or a flag has been selected will exit back to singlestep leaving the values intact. The values of the separate I,X And I,Y are shown on the screen, these can only be changed as a 16 bit register. The separate registers are shown because the instructions that Zilog did not document work on the high and low bytes of both IX and IY.

(R) This will reset all registers back to zero except IY and SP. These last two will be reset to their start up values. You will be prompted with a Y or N.

(T) This is the trap mode. Any instruction that is about to modify (POKE) a memory location will be trapped. You will be asked "OPERATE THIS INSTRUCTION Y or N". You will be able to decide for yourself by looking at the screen whether the instruction would corrupt snything, and take the appropriate action. If you press "N" the instruction will be ignored. A message will remind you the trap is on. This is useful if you are in autostep mode, or are stepping someone elses code and are not quite sure what is happening.

(U) This will toggle the opcode line or the full disassembly into upper or lower case, depending on your preference.

(V) This will clear the attributes area of the screen only.

(Z) This will enter the disassembler. Exit from the disassembler is via "Q". The start of disassembly will be the address of the current stepping instruction.

USING SINGLESTEP MODE

Single stepping is useful for debugging your code, and is a helpful utility when developing a piece of software. Certain instructions would cause problems if they were allowed to operate. i.e. DI : EI : LD R,A : LD I,A : HALT: IM0 : IM1 : IM2 These instructions will be shown but will not operate. Any real time code such as loading or saving routines, or any interrupt routines will not work.

Apart from these restrictions you should find singlestep a useful utility. For instance if there is an instruction that you find difficult to understand. (e.g. how are the flags affected? or what happens to the register involved?). Using the assembler to input the opcode, and then using singlestep can make things a lot clearer.

Using the example that we used in the assembler mode, we will now singlestep the code from address 40000. If you have been reading the instructions and trying various keys, and the example from listing two is still in the memory, then press "A" and enter 40000 when prompted, then press ENTER. You should have

```
40000 : ld b,255
```

If not then press "M" and exit to the menu. Re-enter the assembler and type in and assemble the code as in the example. Once that is done exit the assembler with the "Q" key and re-enter the TOOLKIT.

Ok, that done enter 40000 and you should have

```
40000 : ld b,255
```

If you now press the "Z" key the display will change to the disassembler mode and you will be able to check that your code is as it should be. Pressing the "Q" key will exit back to the

singlestep mode. If you look at the B register on the screen you will see it holds zero, pressing the space bar will operate the instruction and you will see the value 255 appear in the B register. At this point you can change the B register value if you wish. This will only change the value on the screen, it will not change the value in the memory location. To do this you will have to go into the disassembler mode and use the "K" key. Continue to press the space bar and you will see the code starting to work. After a few moments press the "T" key, the message "TRAP ON" will appear at the bottom of the screen. Now press "C" this will start to autostep but every time the instruction: ld (hl),a appears the program will stop and ask you "OPERATE THIS INSTRUCTION Y or N". Typing "N" will ignore the instruction, and you can verify this by looking at the place on the screen where the (POKE) would have occurred, it will have been left alone.

To turn the trap off again press the "T" key, now press "C" again, and leave the program running for a few minutes until it finishes. You will notice that it came to a stop once it encountered the RET instruction. This will only happen if in the normal course of events the RETurn would have been to BASIC. As we have not used any calls pushes or pops, then the stack has only a zero on it. If the stack had any value other than zero then a jump to the location that the stack pointed to would have been made. A program that works properly will always stop when it comes across a RET and the return is a zero. A CALL or a POP will take the value zero from the stack. This function only works in autostep mode.

Ok now that we have worked this example you will have some pretty flashing colours on the screen. If you now press "V" the attributes area that has been poked will be cleared. Now we will have a look at a ROM routine. Press "R" and answer Y to the prompt. Press "A" and enter the value 16 decimal or 10 hex. Now press "Q" and then "A" to change the value in the "A" register, type in 65 and press ENTER. What this ROM routine will do is to print the ASCII character that is held in the "A" register, in this case 65 decimal = the character "A". Press "L" to remove the logo from the top of the screen, and now press "C" for autostep and just sit back and watch what happens. You will see the character "A" slowly building up. Once the routine has finished it will come to a stop. If you now press "O" to retrieve address 16, and then "q" and enter a new value into the "A" register, then you can have another go. Experiment with different values in the range 33 to 255.

DISASSEMBLER MODE

The disassembler is entered from singlestep via the "Z" key.

CONTROL KEYS.....

(A) Change the address to disassemble from.
(C) Auto disassemble, useful if using the printer.
(D) Enter the ASCII dump. "Q" to exit.

(H) Toggle hex to decimal or vice versa.
(M) Modify a memory location.
(P) Printer on off.
(Q) Exit back to singlestep mode.
(U) Toggle display between upper or lower case.
(Z) Enter the loader mode.

LOADER MODE
Pressing "Z" will enter the loader mode. This mode is useful for entering those hex or decimal magazine listings, and once entered, exit to the disassembler will enable you to look at the code, exit from the disassembler and you can now singlestep the code. Entry will be in either hex or decimal depending on the base that has been selected. On the left hand side of the display you are shown what is already in the particular location, the right side of the display will be waiting for you to enter your new value. Pressing ENTER will skip over the location leaving it intact. Delete will erase the complete input. and wait for another. The ASCII character associated with the location is shown, but only if the code falls in the range 32 to 127. In the event that you wish to clear some of the memory, then selecting the address before entering from the disassembler, and then once the loader mode has been entered, pressing symbol shift and "B" will enter zeros.

ASCII DUMP
There are only two keys in this mode:
SPACE KEY - moves to the next page.
"Q" KEY - exits this mode back to either the singlestep mode or the disassembler mode.

Any code inside the range 0 to 31 or 128 to 255 will be shown as a space character. Instructions for this mode are on the bottom of the screen.

MODIFY MEMORY LOCATION
This is used to effect a POKE into memory. To use it press the "M" key, and follow the simple instructions. You are asked to enter the address to poke, then the value to be poked into the address. Pressing ENTER defaults to zero.

PART C - MANUAL FOR SOURCE = CREATOR (CTOS)

SOURCE CREATOR has been written to allow you to create a source file for the LERM ASSEMBLER from any piece of machine code (or "object code") up to 6000 bytes in length. i.e An ASSEMBLER creates machine code from a source file, this program does the reverse. Allowance is also made for bytes which represent blocks of data (e.g messages to be printed, etc).

Options are available to :-
(i) Create data blocks, up to a maximum of 100, These will appear in the final source as DEFB's.
(ii) Create labels within the source file.
(iii) Create EQU's to enable easier alteration of any absolute addresses within the object code.
(iv) Create a source file directly from any block of ROM.

The main uses for SOURCE CREATOR would be:
(a) Moving machine code to a higher or lower address, i.e. to enable it to be used with micro, wafa or disc drives, which may use the RAM that the original object code occupied.
(b) As a tutorial, in conjunction with Z80 TOOLKIT.
(c) As a tool to enable users to alter or modify pieces of code from ROM or other sources.
(d) As a way to transfer source files from any other assembler.

1. LOADING

You are provided with 2 versions of CTOS on tape:
(a) Is a stand alone program (SIDE 2).
(b) Within the complete Z80 TOOLKIT PROGRAM (SIDE 2).

When using any of the options, before loading, first NEW the Spectrum (using RANDOMIZE USR 0). All memory has to be cleared so that CTOS can locate Z80 TOOLKIT if it is resident, and make appropriate adjustments.

To LOAD follow the instructions given under the LOADING THE TAPE section of this manual.

The length of source created by CTOS will vary according to which version is used, but ALL source files are created from address 33000.

(a) The stand alone version from SIDE 2 of the tape.
This option allows 32k of source to be created. When loading this source back into the ASSEMBLER use LOAD "source" CODE,27000 after entering the ASSEMBLER press 0 27000
This tells the assembler that the source file starts at 27000. The default can be changed to 27000 by pressing "M", ENTER, "O", and ENTER. (letter "O" not the number zero).

(b) The version in the FULL TOOLKIT from SIDE 1 of the tape. This option allows 17k of source to be created, which may be listed immediately. Also all functions of Z80 TOOLKIT may be utilised.

(c) Version when ASSEMBLER and CTOS present. (see appropriate section of the manual on how to create, this version).
This is the same as (b) except that 26k of source may be created.

The resulting source file MUST be saved when using version (a) but it is recommended that this also be done as good practice when using the other versions.

2. GETTING STARTED

Before any work should be done on creating a source file, a few words of advice. Any piece of object code you wish lo create a source file from should at least be fairly well known to you. There is little point in creating a file if you don't know what it does! The correct way to go about it is to first make sure it will fit into SOURCE CREATOR when it is converted.

First have your section of object code saved on a separate tape. This must be no longer than 60100 bytes in length. To ensure that this is the case you can first load it and resave using SAVE "code" CODE,address,6000

NOTE CTOS uses the screen as a buffer. The machine code which CTOS works on must be placed into the screen, starting at 16384. No more than 6000 bytes of object code should ever be loaded into the screen at anyone lime. Doing so will only corrupt CTOS and the Spectrum may crash.

There are ways that any length of object code may be processed, but this must be done in small sections and the various pieces of object code joined after assembly. This process is only recommended to those who are very familiar with machine code and would involve many hours of work.

Assuming your piece of object code is less than 6000 bytes, we would advise using a small block of code first. Under 1k for a start, then follow the instructions on screen.

3. CREATING THE SOURCE

When loaded you will see the prompt INPUT ORIGINAL START

This must, be the start address of the object code you are to load into the program. i.e. the address it usually sits at. Do NOT assume that this is the address in any RND USR you may have seen that calls the object code. Not all object code IS called from the address at which it sits.

When entered you will then be asked to INPUT ORIGINAL START

Likewise this must be the last address at which any of the object code appears. Do NOT assume this to be the address of the start of the last instruction, it may be a four byte instruction and three, of these will be lost. If in doubt add 4 to the final address.

After this you will be prompted HOW MANY DATA BLOCKS

If this is the first time you have used SOURCE CREATOR and you are not sure of the number of blocks of data in the object code you wish to create the source from, then type 0 and enter. The code will then be transferred or you will be prompted to load.

It is recommended that any blocks of data are found before a source file creation is attempted. A source file containing DEFB's will be both shorter and will contain fewer meaningless labels. The reason for this is that most JR condition instructions ore within the range of the ASCII code and thus would generate labels in the source file which need not he there. See HINTS AND TIPS for a fuller explanation.

If you have prepared notes about the object code then enter the number of data blocks. The screen will now clear. At this point a prompt BLOCK No. n will appear at the foot of the screen.

You will now be asked .. INPUT START OF BLOCK.
Then when entered ..... INPUT END OF BLOCK ...

Enter the start and end addresses of each block as prompted.

If the end address of the object code entered is less than 16384 i.e. within the ROM, then it will automatically be transferred into the processing buffer - screen memory. This is to enable faster access to ROM based routines. If however you wish to look at areas of other paged ROMs or EPROMs i.e. disc, wafa or microdrive, which occupy addresses 0 to 16384 then see notes in HINTS AND TIPS.

If the last address was above 16384 i.e. in RAM then you will be prompted < Start tape and press any key >

The program defaults to loading from tape but this may be altered to any other medium by changing the basic loader to suit your own choice. See HINTS AND TIPS for further details.

Set your tape at the point where the piece of object code you wish to create source from starts and press play on the tape.

The code will be loaded into the screen and in a few seconds the source file will have been created.

As the screen and attribute memory are used as buffers throughout the process of source creation, please do not be alarmed when you see strange things occurring. It is all quite normal and not a crash.

At any of the next stages you may get a message reading....
NO MORE ROOM FOR FILE
In the event of this happening then please see notes at the end of this section.

You will now be prompted :- DO YOU WANT LABELS Y/N

If YES the program will search for and store in the screen buffer all possible occurrences of labels. This process will inevitably find some labels which are both inside and outside the code. As all 16 bit numbers look the same to the Spectrum, there will also be some that do not have any validity. This however is of no harm but cannot easily be overcome.

If NO then see notes at end or this section.

The screen will clear and three messages appear.

```
(I)   LABELS FOUND      nnn
(II)  LABELS INSERTED   nnn
(III) LABELS OUTSIDE    nnn
```

(I) Is the number of possible labels within the source file.
(III) Is the number of possible labels pointing outside the file.
(II) Will count up as each label is inserted.

Labels found are the actual number or labels that need to be

inserted. Some will not be found because they will be pointing into the middle of an instruction. If any labels are found outside the code, you will then be asked DO YOU WANT EQU's Y/N

Pressing Y will add EQU's to the beginning or end of the source file depending on where the original object code was situated. If at this stage you enter N then the L prefix on all unfound labels must be edited out later.

The process of creating the source file will now be completed and you will be asked to:

save the resulting file if using version (a)
OR enter the ASSEMBLER for versions (b) or (c)

NO MORE ROOM FOR FILE message

If the amount of object code is too long for CTOS to create a full source file then a "NO ROOM FOR FILE" message will appear. The best course of action in the event of this happening would be to start again with at least 100 bytes less object code. This of course depends on exactly how much of the process was completed before the NO MORE ROOM message appeared. If a few EQU's are all that is missing it may be wiser to spend time sorting them out in the assembler rather than losing 100 bytes from the end of the code. See HINTS AND TIPS for further explanations and examples.

4. WORKING ON THE SOURCE

When the assembler is entered you should be looking at a source listing with line numbers corresponding to the relevant addresses of the original piece of object code. If EQU's have been entered then they will appear either at the start or end of the file. Depending where in memory the original code was situated.

DO NOT RENUMBER the file as these line numbers are the only way you have of knowing where you are. The renumber function in LERM ASSEMBLER will be inactive on entry. To enable it if renumbering is desired then do:- M to enter monitor and press R. This will then return you to the assembler with renumber enabled. Then use renumber as normal. (pressing R in the monitor mode of the assembler switches the renumber off and on)

You could look upon it at the moment as a living disassembly, one that you can alter at will. In fact CTOS can act as a relocatable disassembler. Code is always seen in its original location. Unlike most disassemblers which at some point necessitate the relocation of code to observe it in it's actual location.

You will also see that all labels agree with line numbers and 16 bit numbers all have an L prefix. These will enable you to find your way around the source file with the help of the line editor in the LERM ASSEMBLER. The reason the file is created in this manner is to allow large pieces of object code to be processed into source files in small manageable blocks, then after assembly, rejoined as the final object code.

You should now be able to assemble the object code. First enter a line number lower than the first line in the source file thus

nnnnn ORG xxxxx : DISP 100.

(where nnnnn is a suitable line number near the top of the source file, and xxxxx a suitable ORG address)

In the event that you are creating a source file from address 0 then a multistatement line must be used.
e.g. 00000 ORG nnnnn : DISP 100 : DI
in the case of the Spectrum's ROM.

Now do "A" to assemble the object code. This will go through the two passes needed for full assembly, but not create any object code as the DISP points into ROM. However if by any chance there are any labels not defined this pseudo-assembly will catch them as errors but not allow the source file to be overwritten or come to any other harm. See HINTS ANID TIPS for reasons for missing labels.

From this point on use the file created as any other source file. Adding meaningful labels and altering it at will. The rest is up to you.

CTOS - HINTS AND TIPS

1. EMBEDDED DATA.

In all machine code there is always the possibility that there will be some data embedded within actual code. The problem to the user is how to recognise what is and is not data. Usually this is achieved by a process of trial and error, but there are ways to make more intelligent guesses. Below are listed a few of the methods we use. There may well be more.

(a) When disassembling the code you are to work on, look out for instructions which don't seem to make sense. This would include a register being loaded many times over in succession e.g.

LD B,C : LD B,L : LD C,C : LD C,A : LD D,L

These are in fact the way that, all the lower case vowels would appear to any disassembler. All ASCII codes from 64 to 127 are also LD reg 1,reg 2 instructions. So could well he an alphabetic character.
Also look out for JR displacement instructions clustered together, especially one after another, If no data blocks have been defined at the outset then these will all be tagged with a remark thus:- JR NZ,45I23;*45. The 45 in this case is the actual byte as found in the object code before the two's compliment has been worked out to find the actual displacement. Therefore the two bytes at that point in the code would be:- 32 and 45. Displacements and two's compliment are too complicated a subject to discuss in this short handbook. We leave it to you to discover the mysteries of the Z80.

(b) A better, less hit and miss, way to find data is to use the ASCII dump method. For this you can use Z80 TOOLKIT or the MONITOR in LERM ASSEMBLER. See manual for full instructions on use of either. When using the dump you will see the ASCII characters as they would appear normally e.g.

47725 PRESS Q TO QUIT!PRESS

In this instance the P is the start of a piece of data. Its address is at 47729. This would then be the start of a data block.

(c) This last method would only be of use once the source file has been created. This will also solve another small problem that you may encounter. After trying to assemble the file you may get some ERROR 9 reports, Undefined label. The label cannot be found because it should appear in the middle of what is at the time, an instruction e.g.

```
39999        LD HL,(L0)
40002        ETC.
```

A three byte instruction, could be the end of one block of data and the start of another. This could read:-

```
39999        DEFB 42
40000 L40000 DEFB 0,0
```

In this example a label pointing to 40000 would exist, but as no line number 40000 can be found then an error occurs. All this may only be a short block of data it could have been over looked. To solve it either enter the data as DEFB's at the same time adding line 40000 or amend the label which points to it to L39999+1 And insert a label L39999 at line 39999. The lines to be altered will have been listed at the time of the error.

2. THE NO MORE ROOM FOR FILE report

This report may occur at any time during the source creation process. All it means is that the space in RAM allocated to the source file is full. It can never overflow, so there is no chance of a crash. When this happens you will be left with three choices.

(a) Save the source already created.
(b) Try again with less code at the start.
(c) If using the copy including Z80 TOOLKIT change to one which gives you more source area.

There is no right or wrong course of action to take. It depends on exactly how much source and what state of the full process you required has been completed. If in doubt take a look at the resulting source file first. Then decide yourself if it is near enough to completion for you to complete by using the editor. There are times of course when reducing the amount of code fed in would be best. If you asked for labels and the NO MORE ROOM report occurred when only a small percentage had been entered, then a shorter piece of code would be better than a file not labelled. Trial and error are the only sure way to get to know SOURCE CREATORS capabilities.

3. LOADING FROM OTHER MEDIUMS.

CTOS will default to loading object code from tape, but this can be changed by altering the basic loader. To do this first MERGE the basic loader. When it's in the machine change the LOAD "" CODE,16384 to your own requirements. Then resave to tape or whatever other storage you wish to use. CTOS starts at 27000 to allow conversion to any other known storage device used with the Spectrum.

4. ROM's, EPROM's ETC.

It may be the case that you wish to look at a section of a paged device which uses the same addresses as the Spectrum ROM. As CTOS copies directly any code below 16384 automatically into the screen, it would appear that you are unable to load from tape. This is NOT the case however, as all that needs to be done is to POKE 27397 with 201 (normally it is 208) to stop the automatic copying, and force CTOS to LINE 100 to load the CODE into 16384. If you return to BASIC and enter GOTO 120 this POKE is done for you. To return to normal enter GOTO 130.

5. EXAMPLES OF OUTPUT.

Finally here are a few examples of source files using SOURCE CREATOR.

(A) THE ORIGINAL SOURCE LISTING.

```
       00010         ORG 50000           This is a source file
       00020 START   LD A,2              as it would look if
       00030         CALL 5633           it was written using
       00040         LD BC,19            the LERM ASSEMBLER.
       00050         LD DE,STRING
       00060         CALL 8252
       00070         RET
       00080 STRING  DEFB 22,10,10
       00090         DEFM /THIS IS /
       00100         DEFM /A STRING/
```

(B) NO DATA BLOCKS OR LABELS OR EQU'S

```
       50000         LD A,2
       50002         CALL L5633
       50005         LD BC,L19
       50008         LD DE,L50015        In this example you
       50011         CALL L8252          will notice that, the
       50014         RET                 data is seen as
       50015         LD D,10             instructions.
       50017         LD A,(BC)
       50018         LD D,H
       50019         LD C,B              From line 50015 they
       50020         LD C,C              seem to make little
       50021         LD D,E              sense. Why LD D,10
       50022         JR NZ,L50097;*73    when three bytes
       50024         LD D,E              later we get LD D,H.
       50025         JR NZ,L50092;*65
       50027         JR NZ,L50112;*83    Also note the JR
       50029         LD D,H              instructions grouped
       50030         LD H,D              together.
       50031         LD C,C
       50032         LD C, (HL)          We could guess that
       50033         LD B,C              all lines to 50033 were data.
```

(C) ADDING LABEL's and EQU's

```
       49994 L50112  EQU 50112           Some of these are
       49995 L50092  EQU 50092           meaningless to the
       49996 L50097  EQU 50097           code. They are
       49997 L8252   EQU 8252            created because of
       49998 L19     EQU 19              the JR's.
       49999 L5633   EQU 5633
```

```
       50000         LD A,2
       50002         CALL L5633
       50005         LD BC,L19
       50008         LD DE,L50015        The only label we get
       50011         CALL L8252          is the one at 50015
       50014         RET
       50015 L50015  LD D,10
       50017         LD A,(BC)           This does at least
       50018         LD D,H              point us to the data
       50019         LD C,B
       50020         LD C,C
       50021         LD D,E              We guessed right when
       50022         JR NZ,L50097;*73    we said it looked
       50024         LD D,E              like it started here
       50025         JR NZ,L50092;*65
       50027         JR NZ,L50112;*83
       50029         LD D,H              Note the *73 *65 and
       50030         LD H,D              *83, these are the
       50031         LD C,C              data bytes used to
       50032         LD C,(HL)           calculate the
       50033         LD B,C              displacement in the JR's.
```

(D) ADD DATA BLOCKS.

```
       49997 L8252   EQU 8252            Now we have a piece
       49998 L19     EQU 19              of source listing
       49999 L5633   EQU 5633            which looks nearly
       5OOOO         LD A,2              like the original.
       50002         CALL L5633
       50005         LD BC,L19           The EQU's are
       50008         LD DE,L50015        meaningful and the
       50011         CALL L8252          data is inserted.
       50014         RET
       50015 L50015  DEFB 22,10,10       All that is needed is
       50018         DEFB 84,72,73       an ORG where ever we
       50021         DEFB 83,32,73       wish, then assemble
       50024         DEFB 83,32,65       it.
       50027         DEFB 32,83,84
       50030         DEFB 82,73,78
       50033         DEFB 71
```

The above source file could be edited and START labelled at line 50000. It could be renumbered and even DEFM's inserted to replace the DEFB's. Looking at it we see that the majority of DEFB's are within the ASCII set. This means they are alphabetic characters. We could also label 50015 as MESSG or STRING to make it more meaningful to us.

Of course all conversions will not be this easy, but all the tips in this section should help you to compile a source file in a short lime.

In tests 3000 bytes of code, known to and well used by the operator, have been converted to a source file and assembled at another address in less than 5 hours. The same process look 2 weeks of hard typing and working out to achieve by hand.

ASSEMBLER ERROR CODES

ERROR 0 = Illegal character or an incomplete statement.
ERROR 1 = Label too long, six characters only.
ERROR 2 = closed bracket expected. eg)
ERROR 3 = Truncation error or jump out of range.
ERROR 4 = , expected.
ERROR 5 = Context error.
ERROR 6 = Redefined label. (probably already used that name)
ERROR 7 = Open bracket expected. eg (
ERROR 8 = Illegal mnemonic.
ERROR 9 = Undefined label.

ASSEMBLER KEY SUMMARY

FUNCTION	KEY	PARAMETERS	EXPLANATION
ASSEMBLE	A	x	Assemble and print x errors at a time. e.g. press keys "A", "1" would assemble and print only maximum of ONE error. Default = 10.
DELETE	D	x y	Delete from line x to line y inclusive. e.g. D 200 300 should appear on the screen to delete lines 200 to 300. Press ENTER to execute.
FIND	F	F"Ring"x y z	Find "Ring" between line x and line y, and print z occurrences at a time.
AUTO LINE NUMBERS	]	x y	Start auto line numbering from line x with increments of y.
LIST	L	x y z	List from line x to line y in steps of line z.
MONITOR	M		Enter the MONITOR mode.
NEW SOURCE FILE START	N	x	Create a new source file at address x.
RETRIEVE OLD SOURCE FILE.	O	x	Retrieve old source file at address x.
PRINTER	P	x	x = zero for OFF, one for ON
RETURN TO BASIC	Q		Exit the assembler.
RENUMBER	R	x y z	Renumber the source file from line z, to start with line x and increments of y.

SYMBOL TABLE	S	x	Print the symbol table x lines at a time.
SOURCE FILE	T		Print the start and length of the source file.
RUN CODE	X		Run code from ENT statement.

ASSEMBLER SCREEN EDITOR

KEY	FUNCTION
CAPSHIFTED 1 or EDIT	Clear the line containing the cursor.
CAPSHIFTED 2 or CAPSLOCK	Move to the next tab stop.
CAPSHIFTED 3 or TRUE VIDEO	Delete the character under the cursor.
CAPSHIFTED 4 or INV VIDEO	Insert a space character at the cursor position.
CAPSHIFTED 5,6,7,8 CURSORS	Cursor control keys.
CAPSHIFTED 9 or GRAPHICS	Clear the screen and home the cursor.
CAPSHIFTED 0 or DELETE	Normal delete key.
CAPSHIFT held down + a letter	Gives LOWER case. e.g CAPSHIFT "A" gives "a".

MONITOR KEY FUNCTIONS

FUNCTION	KEY	PARAMETER	EXPLANATION
HEX/DEC	A	x	Convert hex number to decimal. e.g press "A",space,"#","f" to convert hex "f" to dec 15.
DEC/HEX	A	x	Convert decimal number to hex. e.g press "A" "1" "5" to convert 15 dec into hex "f"
LOW BYTE HIGH BYTE	B	xx OR #xx	Convert a 16 bit number to its high byte and low byte.
CHANGE COLOUR OF SCREEN		C	Toggle screen colours from white screen, black ink, to blue screen and white ink.
DEFAULT SOURCE CODE	O (letter)		Default the source code after using N in assembler mode.
SEARCH SYMBOL TABLE	F	"string	Find and display the value associated with the label.

			only after assembly. e.g F"fred finds value or label "fred"
LAST LINE	L		Print the last line in the source file.
RENUMBER TOGGLE	R		Switches the RENUMBER facility OFF/ON in the Assembler.
TOGGLE SYMBOL TABLE	S		Toggle the symbol table into the screen or back below the assembler.
TABULATE MEMORY	T	xx	Displays the memory contents from x address. SYMBOL SHIFT "A" to exit back to the assembler.
EXIT TO ASSEMBLER	Z		Exit to the assembler in case of accidental entry.
REMOVE THE TOOLKIT	SYMBOL SHIFT X		Removes the toolkit from memory and reclaims the memory leaving just the assembler. Only works with the full program Z80A. CTOS also remains if in memory.

UNDOCUMENTED CODES

There are quite a few instructions in the Z80 processor that Zilog did not document which are reproduced below. The singlestep and disassembler will handle them all with no trouble, a beep will sound to warn you if one is encountered. The aasembler will not accept them as mnemonics but they can be entered into a source listing by using the following method. As an example one such instruction is

LD A,IX'I

This means load the 'A' register with the IX's I value. This is why the I and the X is shown separately on the screen. The decimal numbers for the above opcode are 221,124. Theme can be inserted into the assembler as follows :

```
00100 DEFB 221,124 ; this is LD A,IX'I
```

As you can see we have used a comment, so that you can see what the instruction is supposed to be. The above example is the format that the singlestep and disassembler will display them in. The codes are printed below.

NOTE that to use IY instead or IX you should substitute DD for FD or 221 for 253 other than that the codes are the same.

UNDOCUMENTED CODES...

MNEMONIC	HEX	DECIMAL	MACHINE CYCLES
SLS A	CB 37	203 55	8
SLS B	CB 30	203 48	8
SLS C	CB 31	203 49	8
SLS D	CB 32	203 50	8
SLS E	CB 33	203 51	8
SLS H	CB 34	203 52	8
SLS L	CB 35	203 53	8
SLS (HL)	CB 36	203 54	15
SLS (IX+DIS)	DD CB XX 36	221 203 DIS 54	23
SLS (IY+DIS)	FD CB XX 36	253 203 DIS 54	23

There are more undocumented shift instructions than this, but the disassembler and singlestep does not include them, so we haven't either. Below are the rest of the undocumented instructions that will be handled by this program.

Note that these are not official mnemonics, but just a form that we feel can be understood.
The first instruction here for example means :
ADD A INCLUDING THE CARRY FLAG, TO THE HIGH BYTE OF IX, or ADC A WITH IX's I.
The next instruction means ADD A INCLUDING THE CARRY FLAG, TO THE LOW BYTE OF IX, or ADC A WITH IX' s X, and so on for the rest.

MNEMONIC	HEX	DECIMAL	MACHINE CYCLES
ADC A,IX'I	DD 8C	221 140	8
ADC A,IX'X	DD 8D	221 141	8
ADD A,IX'I	DD 84	221 132	8
AND A,IX'X	DD 85	221 133	8
AND IX'I	DD A4	221 164	8
AND IX'X	DD A5	221 165	8
CP IX'I	DD BC	221 188	8
CP IX'X	DD BD	221 189	8
DEC IX'I	DD 25	221 37	8
DEC IX'X	DD 2D	221 45	8
INC IX'I	DD 24	221 36	8
INC IX'X	DD 2C	221 44	8
LD A,IX'I	DD 7C	221 124	8
LD A,IX'X	DD 7D	221 125	8
LD B,IX'I	DD 44	221 68	8
LD B,IX'X	DD 45	221 69	8
LD C,IX'I	DD 4C	221 76	8
LD C,IX'X	DD 4D	221 77	8
LD D,IX'I	DD 54	221 84	8
LD D,IX'X	DD 55	221 85	8
LD E,IX'I	DD 5C	221 92	8
LD E,IX'X	DD 5D	221 93	8
LD IX'I,A	DD 67	221 103	8
LD IX'X,A	DD 6F	221 111	8
LD IX'I,B	DD 60	221 96	8
LD IX'X,B	DD 68	221 104	8
LD IX'I,C	DD 61	221 97	8

MNEMONIC	HEX	DECIMAL	MACHINE CYCLES
LD IX'X,C	DD 69	221 105	8
LD IX'I,D	DD 62	221 98	8
LD IX'X,D	DD 6A	221 106	8
LD IX'I,E	DD 63	221 99	8
LD IX'X,E	DD 68	221 107	8
LD IX'I,IX'I	DD 64	221 100	8
LD IX'I,IX'X	DD 65	221 101	8
LD IX'X,IX'I	DD 6C	221 108	8
LD IX'X,IX'X	DD 6D	221 109	8
LD IX'I,XX	DD 26 XX	221 38 XX	11
LD IX'X,XX	DD 2E XX	221 46 XX	11
OR IX'I	DD B4	221 180	8
OR IX'X	DD B5	221 181	8
SBC A,IX'I	DD 9C	221 156	8
SBC A,IX'X	DD 9D	221 157	8
SUB IX'I	DD 94	221 148	8
SUB IX'X	DD 95	221 149	8
XOR IX'I	DD AC	221 172	8
XOR IX'X	DD AD	221 173	8

SOME USEFUL ROM ROUTINES

To understand these routines better then we would suggest the book THE COMPLETE ROM DISASSEMBLY printed by MELBOURNE HOUSE, written by Dr IAN LOGAN and Dr FRANK O'HARA.

PRINTING CHARACTERS.

PRINT-A-1 Use by RST 16 (in hex 10).
Called with the "A" register holding the character to be printed
This will print control characters as well as ASCII.
e.g.

```
00010 ORG 40000
00020 LD A,65;      this will print the character "A" -
00030 RST 16;       could replace with LD A,"A as Assembler
00040 RET;          replaces "A with the CODE of the string.
```

e.g.

```
00010 ORG 40000
00015 AT EQU 22;          must put this before line 130 !
00020 START  LD A,2
00030        CALL 1601;   SETS UP A CHANNEL NEEDED FOR PRINTING
00040        LD HL,MESS;  HL= MESSAGE START
00050 LOOP   LD A, (HL)
00860        CP 255
00070        RET Z;       STOP PRINTING WHEN REACH CHAR 255.
00080 ;
00090        RST 16;      PRINT CHARACTER
00100        INC HL;      SET UP NEXT CHARACTER
00110        JR LOOP;     GOTO LABEL LOOP TO PRINT NEXT CHARACTER.
00120 ;
00130 MESS   DEFB 17,0,16,7,AT,10,0; prints PAPER O,INK 7,AT
00140 ;                                                  10,0
00150        DEFM "This is a test":DEFB 255; 255 gives end.
00160 END EQU $
00170        LENGTH EQU END-START
```

PRINT THE NUMBER IN THE BC REGISTER.

```
LOAD BC WITH NUMBER TO BE PRINTED.
CALL 2D2B HEX or 11563 DECIMAL - called "STACK BC".
CALL 2DE3 HEX or 11747 DECIMAL - called "PRINT FP".
RET

e.g
00010 ORG 40100
00020 LD BC,65534   this will print the number 65534.
00030 CALL 11563
00040 CALL 11747
00050 RET
```

MAKE A BEEP.

```
CALL 03B5 HEX 949 DECIMAL ROUTINE NAME BEEPER.
e.g.
00010 ORG 40200
00020 LD DE,100;    try a few values of your own, but don't
00030 LD HL,100;    make them too large.
00040 CALL 949;     This is a real time piece of code, so it
00050 RET    ;      cannot be properly singlesteped.
```

THIS IS ANOTHER ROUTINE THAT WILL PRINT A STRING.

```
00010         ORG 41300
00020 START   LD A,2
00030         CALL 5633
00040         LD BC,19      ; length of string to be printed.
00050         LD DE,STRING  ; address of the string.
00060         CALL 8252
00070         RET
00080 STRING  DEFB 22,10,10
00090         DEFM "THIS IS A STRING"
00100 END     EQU $
00110 LENGTH  EQU END-START
```

This example will print at position 10 to on the screen,"THIS IS A STRING". Change the 10 10 to 0,10 and singlestep it.

OTHER ROUTINES

AN 8 BIT PAUSE LOOP.

```
00010         ORG 41400
00020         LD B,255      ; loop round 255 times doing nothing
00030 LOOP    NOP           ; until B = O. then return.
00040         DJNZ LOOP
00050         RET
```

A 16 BIT PAUSE LOOP.

```
00010         ORG 41500
00020         LD BC,12345   ; loop round 12345 times doing
00030 LOOP    DEC BC        ; nothing until A = 0. then return.
00040         LD A,B
00050         OR C
00060         JR NZ, LOOP
00070         RET
```

WAIT FOR A KEY PRESS.

```
00010         ORG 41600
00020 KEYLP   LD A,(23560) ; 23560 is a system variable,
00030         CP "a        ; wait until the "a" key is pressed
00040         JR NZ,KEYLP  ; and then just return.
00050         RET
```

NOTE: Before using m/code to print to the screen, it is necessary to set the print position to the TOP LEFT (i.e at 0,0). This can be done, with a line, so:

```
       00010  XOR A;       makes A=0, uses less memory than LD A,0
       00020  LD (23612),A
```

These routines are not original, but they may help the beginner, to start to understand m/code. There are a few good books on the market written for the SPECTRUM owner wanting to learn m/code.

EVERY EFFORT HAS BEEN MADE TO ENSURE THE ACCURACY OF THESE PROGRAMS AND THE CONTENTS OF THIS MANUAL.

ASSEMBLER - EXTRA INFORMATION

1. Finding the start address and length of source code

This can be done from BASIC rather than using the "T" command from within the ASSEMBLER. So you can easily modify the BASIC to save the source code. The following could be used.

```
1000 DEF FN f(x)= PEEK x + 256*PEEK (x+1)
1010 LET start= FN f(65459):LET end= FN f(65461)
1020 LET length= 1 + end - start
1030 INPUT "Name of source file ";LINE d$
1040 SAVE d$ CODE start,length
```

start= start address of source, and end= last address used, so that length is the difference between them plus 1. The last 2 bytes of the source are always 255 - that's how the assembler knows where the file ends.

2. Merging source files

If you have already got a source file in memory and want to add another on the end of it, then all you have to do is to load your second file into ONE LESS than the value of "end" given in the example above OR just enter

```
PRINT PEEK 65461 + 256* PEEK 65462 - 1
```

If the value was say 40300, then load your second code into 40300. e.g LOAD "newcode" CODE 40300

3. Altering the default start of source from BASIC

When you enter the assembler the default start of the source is 27000. By POKING the low and high bytes into the addresses 59482, 63007, 63012, and 65459, you can change the default.

e.g To change the default to 40000 (which in low byte, high byte order is 64 and 156 as 64+156*256 = 40000) do the following

POKES:	Address	Poke
	59482	64
	59483	156
	63007	64
	63008	156
	63012	64
	63013	156
	65459	64
	65460	156

4. Expressions

It is possible to use the assembler to calculate labels by evaluating an expression with a "plus" or "minus" in it.

```
e.g    10 FRED EQU 23000
       20 ALAN EQU FRED+10
       30 JOHN EQU FRED-20
```