MicroWorld Z80

Editor/Assembler

Instruction Manual

Machine Code Programming For Your Microbee.

CONTENTS/INDEX

INTRODUCTION

This package consists of a "line oriented" text editor with automatic line numbering and an assembler which will generate Z80 machine code from standard Zilog Mnemonics as specified in the "Zilog Z80-Assembly Language Programming Manual". Macro operators and certain arithmetic operators are not supported.

The Editor files may be saved to, or loaded from, cassette using the Save and Get commands.

Multiple files may be present in memory at the same time. Lines from the primary file may be copied and appended to the specified secondary file, or the entire contents of a specified secondary file may be merged back into the primary file with automatic renumbering of the primary file if required.

Specific lines in the currently "open" file may be accessed by their line number of by cursor control. Once accessed the lines may be altered, appended to, replaced, or deleted. Lines may be inserted into the file by simply assigning a line number appropriate to the location you wish it to take up in the file. Likewise a line, or block of lines, may be manipulated or killed by specifying the block of lines involved.

Global search and replace functions are supported by the editor, each occurrence in the run is reported as it is found.

Up to 14 lines on either side of the "current line" may be inspected, without moving the line pointer, by using the "View" feature.

Files may be typed to a printer with the line numbers stripped off, allowing the Editor to be used for letter writing.

The assembler produces its object code directly into memory ready for immediate execution. An offset feature allows the object to be located in a location other than where it would normally reside, to prevent overlaying important memory areas during assembly. Since a three pass technique is used the object code may even overlay the source file if desired.

Assembler labels may be up to 6 characters in length and provided that they start with an ascii character may have almost any other character imbedded within them. Source listings generated during assembly may be suppressed or directed to either VDU or printer.

Full error reporting is provided even when source listing is suppressed, a "wait on error" function is allowed, with conditional return to the Editor automatically set up on the line containing the error.

Print directives are provided to allow the listing to printer to be turned on or off under software control, this allows the printing of partial source listings. Listing of the object code for long strings or data statements is automatically suppressed to conserve paper, a "switch" is provided to allow listing in full if desired.

GETTING IT RUNNING

There are three resident software packages supplied with the Microbee. If you have battery backup, any one of the three may be running when you turn your Microbee on. Microbees without battery backup will always enter BASIC when they are turned on.

These three software facilities are described briefly here.

EDITOR/ASSEMBLER	The editor and Z-80 assembler described in this manual. When the Editor/Assembler is operating, the user is prompted with '*' and an underline cursor '_'.
BASIC	Microworld 16K Basic. A superhuman basic interpreter, often considered by most Microbee owners to be more powerful than a locomotive (until they discover the joys of programming in machine language with the Editor/Assembler). The user can tell when the Basic interpreter is running by noting that the prompt is '>' and the cursor in an underline '_'.
MONITOR	This is the software facility to use when you want to know all about the nitty-gritty details of the stuff inside the computer's memory. The main details of the monitor are laid out in Appendix-E. The monitor gives the same prompt as Basic '>' but with a white blob cursor (so you can tell them apart).

Getting into the Editor/Assembler is a simple matter. From basic, typing EDASM will get things started for you. (Note that in reading this manula,if you are instructed to type anything, you are expected to terminate the line by hitting <return> unless you are specifically told not to). If you haven't used the Editor/Assembler recently, it will have forgotten how much memory your Microbee has. If so, you will be asked the question 'Memory Size?'. If you have a 16K Microbee, reply by typing 4000. This is the memory size of your machine expressed in hexadecimal (4000-hex = 16384-dec = 16K). 32K Microbee owners can reply with 8000.

If you wish to enter the Editor/Assembler from the monitor, simply type X . This command means 'exchange' and can also be used to get from Editor/Assembler back into the monitor.The name of the command comes from the fact that many users spend a lot of time 'swapping' between the Editor/Assembler and the monitor.

There are many ways to get into the monitor. As already noted, X gets into the monitor from the Editor/Assembler. Typing <reset>M will get into the monitor from anywhere. To do this, just hold down <reset> for a second and then hold down the M key and release <reset> while keeping the M key held down. This is not a command, so it is not necessary to hit <return>. You can do a <reset>M absolutely anytime (even in the middle of a program).

Getting into basic is a bit more complicated. Typing <reset><esc> (in the same ways as <reset>M) will do a 'cold start' in basic, which will remove any basic programs or Editor/Assembler files. <reset><esc> is a last resort to start from scratch when everything goes wrong. Like <reset>M , this does not require a <return> and can be done anytime. Appendix-G contains a bit of imformation about file and program retrieval if you wish to restore files or programs after an accidental or unavoidable cold start.

A more dignified way of 'cold start'ing basic is the B command (from the monitor or Editor/Assembler). This still wipes all files and programs, being identical to <reset><esc>, but is more often used out of choice rather than desperation. If you wish to get into basic without losing your programs and files simply type G 8021 (from the monitor) or X 8021 (from the Editor/Assembler). Either of these commands will run the machine code program at 8021-hex which happens to be the basic 'warm start' vector.

Hitting <reset> (on its own, held down for at least a second) will return the user to the most recently used system. For example, if <reset> is hit in the middle of a program executing (called from the monitor), control will be returned to the monitor. If the program had been run from the Editor/Assembler, then <reset> would have returned us to the Editor/Assembler. The Microbee remembers which system was running by updating the relevant 'where am I'-information every time one of the commands mentioned above is used to change from one system to another. This includes using <reset>M and <reset><esc> but does not work with G 8021 or X 8021, because a warm start in basic does not update this 'where am I'-information. This means that if basic is entered by a warm start (by G 8021 or X 8021), then any subsequent <reset> will cause the system to revert back to the monitor or Editor/Assembler, depending on which one basic was called from.

The way to fix this problem is by manually changing the warm-start jump vector in the scratch locations at 00A2-hex. Do this by getting into the monitor and typing E A2 which will display data in and around location 00A2. Hit M (without <return>) to modify the memory and then type 2180 (without <return>). This enters the Basic-warm-start vector into the locations 00A2 and 00A3 so hitting <reset> now will enter Basic and cause Basic to be warm-started every time <reset> is hit from now on (until an entry into the Monitor or Editor/Assembler changes the vector).

BRIEF DESCRIPTION OF EDITOR

As you read this manual, you will notice that whenever a new term is introduced into the text it is shown in CAPITAL LETTERS. If after reading the paragraph, you are still not sure of the meaning of the word, try looking it up in the glossary of term towards the end of this manual. Readers who have never met an editor are advised to read Appendix-A "What is and Editor" before proceeding further into this section. Anything that is inadequately explained will probably be covered in more detail in the Appendix.

This editor is a line editor, enabling users to create files in a manner similar to basic programs. Each line has a number and the lines reside in the file in numerical order. The editor can operate on two files, designated PRIMARY FILE and SECONDARY FILE. This facility enables the user to use bits and pieces of old files to create new files by moving text from one file to another. In practice, many files may reside in memory simultaneously, with only two of them being active at any one time. To make a file active (thus deactivating a currently active file), you only need to open a file (primary or secondary) at the place in memory where the file to be activated resides. This means that the user is responsible for keeping track of the location of of all the inactive file since the editor is ignorant of their existance (until they are made active).

The editor commands allow you to insert, delete or replace lines. The full dictionary of editor commands is detailed in the next section. One special command is E (edit), which enables the user to work on one line of a file in a more detailed way, inserting, deleting and replacing characters in the line. A whole new set of commands (subedit instructions) become available when the E command is excecuted. The subedit commands are listed in Section 5.

EDITOR INSTRUCTIONS

We will now deal with the editor commands in detail. Anyone who is unsure of exactly what is going on is advised to look at Appendix-C for a tutorial example of the use of the editor (and the assembler). The commands described here are listed in some sort of order of maximum comprehensibility, so anyone requiring an alphabetical listing is requested to look in the command index in Appendix-D.

Note that in every command that can specify a line number as a parameter, # can be used to refer to the first line of the file, * can be used to reference the last line in the file and . refers to the current line. These are known as WILD CARDS.

Z — This command creates a new file, if an address is given (e.g. Z 3000) the file is to be created at the specified address. If no address is specified, the file will be created at 1000-hex, by default. The file is automatically opened as the primary file.

ZS — This creates a file as above and makes it the current secondary file. If an address is given (e.g. ZS 3800) then the file will be created at that address. When no address is specififed, the file is created 1K above the end of the current primary file. Note that the editor does not do any checks to ensure that it does not destroy the secondary file by enlarging the primary file to wipe over it. You may have a number of secondary files in memory at one time but only one can be open (active) for use in operations with the primary file.

O — Once we have created a few files here and there in memory, they are all the same format, so files that were created as secondary files will be no different to files created as primary files. So, we can deactiveate our current primary file and open any other inactive file by the O command. For example O 2000 will make the file at 2000-hex the current primary file (if there is a file at 2000-hex). Once again, "no address" defaults to 1000-hex. If no file is resident at that address, a ‘No File Here’ error will occur because the Editor/Assembler is smart enough to know what a file looks like in most cases. However, it is always good practice to query the state of the file (with the Q command) since it is possible that some rubbish in memory may look like a file.

OS — This attempts to open a secondary file at the address specified (e.g. OS 3800), thus deactivating the previously open secondary file.

I The insert command allows you to add new lines to the primary file. For example I180,10 will cause everything subsequently typed to be inserted into the file at lines 180,190,200,... as specified by the starting number (180) and the step size (10). If the step size is ommited, the editor will remember the step size used last time an insert was performed. So I180 will insert lines from line 180 in the same size steps as used in the last Insert command. If the start line is ommited, the lines are inserted after he current line in the file (The current line is the line most recently looked at). However, if an I command is used before any other instructions have been done, then the default action is to insert at line 100 in step sizes of 10. If the linenumber steps over an existing line, the insert is completed and terminates with a 'No Room Between Lines' error message. For example, if a file already has lines numbered 230 and 250 an I240,5 command will insert lines at 240 and 245 and then terminate because of lack of room between lines. The I instruction can also be aborted by typing <ctrl>C or <ctrl>A (neither of which require a <return>).

D The delete command has two formats, D245 or D240:250. In the first case, only line 245 is deleted. In the second case all lines between 240 and 250 (inclusive) are deleted. As with all commands that allow a line number to be specified, if no line number is given, the current line is assumed. It is not recommended to use WILD CARDS with the delete command, D#,* is not the best way to delete the entire file, use the Z command instead.

R Replace the specified line in the file and then go into insert mode. The command R240,10 is equivalent to D240 followed by I240,10. Once again, if no parameters are given, default step size and line number are as for the insert command.

N This command will renumber the complete file. For example N300,10 will renumber the file so that the first lin is numbered 300 and subsequent lines are numbered in steps of 20 (320,340,360,...).

Q Query the status of the currently open primary file. Four addresses are printed out: START or file, address of CURRENT LINE in file, address of END of file and UPPER LIMIT of memory.

P Print lines onto the video display. So P200:450 will print all lines from 200 to 450 inclusive. The wildcards '#', '*' and '.' can all be used with the print command. In all instances of the print command, the last line printed becomes the current line. If only one line number is specified (e.g. P300) then this line alone is printed and becomes the current line. A special version of the command is P with no parameters specified which prints

one screenful of lines starting at the current line. Thus a useful way to look through the entire file is to print the first line (P#) and then print the rest of the file one screenful at a time (P).

^ The circumfles causes the current line pointer to be moved backwards by one line (towards the start of the file). The new current line is printed. This command does not require a <return> which makes it fast for looking through the file backwards.

L/F The line feed is the same as ^ except that the current line pointer steps forward through the file. No <return> is required.

V View the 14 lines around the current line without changing the current line pointer. If the command contains a number (e.g. V9) then the current line will appear at this line on the screen (in this case the current line will be the ninth line in the fourteen printed on the screen).

L Same as P command but with output directed to the printer port instead of the video display.

T This is another version of the P command but with the output sent to the printer and the line numbers stripped from the file (useful for letter writing).

E Edit a line using the subedit command listed in Section 5 of this manual. E100 will edit line 100. E will edit the current line.

F The Find command will search through the file looking for a given string, starting from the line AFTER the current line. The command F/elephant/ will look for the next occurence of the word elephant and print the line (making it the new current line). The command F will search using the string used for the last search (useful for searching for the same string in several places in the file). If a C command is executed, the last string used in an F command is forgotten (because the same storage is used for the C command).

C The command C/mouse/elephant/ will search for the next occurance of the string 'mouse' and change it to elephant. The command C/mouse/elephant/* will continue repeating the change until the end of the file is reached. This will only do the change to the first occurance of 'mouse' on each line, and will not do anything to the lines before the current line.

CO The copy command will take a block of lines from the current primary file and append them to the end of the currently open secondary file. The primary file is not

altered by this command. So the command CO100:300 will copy lines 100 to 300 (inclusive) onto the end of the secondary file. After each copy, the status of the secondary file is printed as a reminder to the user that the editor does not check or protect the secondary file.

M The merge command will copy the entire contents of the currently open secondary file into the primary file (insertine after the current line). The lines of the secondary file are inserted with line number steps of 2. The lines following the inserted text are also renumbered in steps of 2 for as many lines as necesary to put the line numbers back into increasing order. The command could be M300 to merge the secondary file in after line 300 in the primary file or just M to do the merge after the current line.

S Saves the current primary file to cassette. The command must be in the form S "NAME" with the quotes and file name (up to six characters) being compulsory.

G This command will load a file from tape. The command must be in one of two forms: G* will load a file with any name and G:"NAME" will not load the file unless it was saved with the name label "NAME".

B The Bye command does a cold start of basic (quitting the Editor/ Assembler).

X This command will execute a machine code program. The command X with no parameters will execute the monitor (so this is the command to use to get into the monitor). The command X3500 will execute the machine code program starting at 350-hex.

A This is the Assemble command. Sections 6 and 7 of this manual are devoted to describing the workings of this facility of the Editor/ Assembler.

When inserting lines into a program, remember the following keys have special meanings:

<tab> leaves blanks up to the next character position which is a multiple of 8. Useful for putting stuff into columns (such as assembly language programs).

; recognised by the assembler, meaning that everything following the ‘;’ until the end of the line is to be ignored as comments.

<B/S> backspaces over characters, deleting them from the input line buffer (but doesn’t delete them from the screen).

SUB EDIT INSTRUCTIONS

Whilst in EDIT mode, the normal editor command set is not available, and a separate set of commands are used. Note that noe of these commands require a <return>, since the <return> key has a special meaning of its own.

L — This command lists the entire line to the video display so you can see what the line looks like so far. This is useful at the start of an edit to see what the line looks like and during an edit to see how the edited line looks so far. Note that the line as it appears when the L command is executed does not have to be included in the file since the q command will quit the edit without the changes being put into the file.

Q — Quit the edit without changing the file. In other words pretend the line was never edited.

<space> — The space key will move the cursor one position forward along the line and reveal the characters in the line as it passes over them. There is no equivalent key for moving backwards since the back-space is destructive.

<B/S> — The backspace key is destructive as all characters backspaced over will be removed from the line.

A — Ignore all changes made so far. Restart the edit with the original line unchanged.

I — Insert all subsequent characters into the line. This sequence is terminated by a <return> to finish the edit. If further changes are required, you must re-edit the file.

X — This is the append command. The pointer is moved to the end of the line and the insert command mode is entered.

nC — Change the next ‘n’ characters in the line to whatever you type next. You must now type the specified number of characters. For example, if you are in the middle of an edit and you are at the start of the word ELEPHANT and you type 5CMOUSE, then the first five letters of the word will be changed so the word will now be MOUSEANT. If no value of n is given, it is assumed to be 1.

nD — Delete the next ‘n’ characters in the edit line. If no value for ‘n’ is given it is assumed to be 1.

H — Delete all characters after this point to the end of the line, and then go into insert mode to add more text to the end of the line. If you only want to delete the remainder of the line without adding more stuff to the end, just type H<return> (since the <return> will stop the insert and exit from the edit.

nSx This command moves the pointer to the ‘n’th occurance of the character x. For example, 5Sq will move the pointer to the 5th occurance of the letter ‘q’ in the line.

nKx This will delete all characters from the current position to the ‘n’th occurance of the letter x. So, 5Kq will delete all characters from the current position to the 5th occurance of the letter ‘q’.

<ret> End the edit and put the edited line into the file (so don’t throw the edited line away as for the Q command). Control is returned to the normal command mode.

E The same as <ret>, not usually used byt provided for compatibility with other editors. This does not work from inside the insert sub command which can only be terminated by <return>.

DESCRIPTION OF ASSEMBLER

Readers who have never met an assembler before are advised to read Appendix-B, "What is an assembler", before proceeding further into this section.

The Assembler is a three PASS device, this means that your source file is read from beginning to end three times by the assembler during the assembly process. On the first pass the LABELS are recorded in a special list, called a SYMBOL TABLE, and addresses or values are assigned to them. This list starts at the TOP OF MEMORY address, given on entry to the editor, and grows down through memory as each new label and value are added. Checks are made to ensure that the symbol table does not "crash" into the end of your source file. If this is about to occur, an error message "Symbol table OVF" is generated, and the assembly is aborted. At the end of pass 1 the assembler knows the location and value of every SYMBOLIC REFERENCE in your source file. Pass two is used to interpret the NMEMONICS and assign the values to all symbolic references in the argument field. It is during this pass that most errors will be detected, the source listing and printouts are also generated at this time. The third pass is used, if required, to generate the OBJECT program into memory.

To commence assembly you issue an A command from the editor. The format of this command is fairly exacting as an OFFSET may be specified, and a number of SWITCHES may be included to direct the assembler to perform specific tasks during assembly. NOTE, if no switches are specified, the command must be typed as A <ret> with the space after the A being compulsory.

The offset allows the assembler to locate the output program code at a different address in memory to the address that it is intended to operate at. This feature will not normally be required by Microbee users since a special area at 400hex has been allocated for them to generate and run their program in.

The offset value is simply added to the adress that each byte will be stored at. eg an offset of 1000 specified for a source ORGed at location 400hex will cause the output code to be stored starting at location 1400hex. Reverse offsets are possible due to address wrap around at FFFFhex, this means that an offset of 0F000 will cause the code to be stored at 1000hex bytes lower in memory. Note that the offset address is always in hex, no H is required after the address, and any value commencing with an alpha character must be preceded by a zero. After assembly, any code generated with an offset must be moved to its correct location before it may be run.

There are up to six SWITCHES that may be specified in the assembly command line. Each switch when used is identified by typing a slash before it. The switches are:-

WE This switch directs the assembler to stop whenever an error is detected during pass 2 of the assembly. The error is displayed to the VDU, and the assembler waits for a direction from the keyboard. If Control C is pressed, the assembly is aborted, and command is passed back to the Editor with the error line set up as the current line so that you may examince or edit the line. If C is pressed (not control C) the WE switch will be cleared and assembly continues, however the assembler will not stop on further errors. If any other key is pressed the assembly continues with the WE switch still active.

NO This directs the assembler NOT to output any object code during assembly. The NO switch should always be used for trial assemblies until you are sure that no errors exist.

NL This suppresses listing to the VDU, errors if encountered will however still be displayed. The listing may alternatively be turned on and off by special print directives in the file. *L ON turns on the listing and *L OFF turns the listing off. These comands are useful for printing part listings from the assembler.

NS Do not list or print the symbol table at the end of the assembly.

LP Direct all listings to the line printer device instead of to the VDU.

PT When listing strings (DEFM pseudo) the object field only lists the first byte of the string. This is done to conserve paper when printing. If however you wish the object field for the string to be listed in full, use the PT switch.

The precise formats of the A instruction are shown in the command index.

PSEUDO MNEMONICS and ARITHMETIC OPERATORS

As stated earlier the assembler broadly complies with the format as defined in the ZILOG assembler, several variables are allowed to assist those familiar with 8080 assemblers. The PSEUDO operators supported are:-

ORG	nnnn	Set or redefine object address counter.
DEFB	n	Define byte to be value n.
DB	n	Same as DEFB.
DEFL	nnnn	Temporarily equate label value, may be re defined later.
DEFM	'ssss'	Define contents of an ascii string.
DEFR	n	Set default radix value. 16-hex, 8=octal, 10=decimal (10 normal) if not defined, defaults to decimal values.
DEFS	nn	Reserve nn memory locations.
DEFW	nnnn	Define value of 16bit 'word' to be nn.
DW	nnnn	Same as DEFW.
EQU	nnnn	Permanent equate label value, cannot be re defined.
END		End of source listing. Stop assembly.

As well as the pseudo operators certain arithmetic operators are available for use in the operand field. These are:-

D	Consider value decimal regardless of default radix.
H	Consider value hex regardless of default radix.
O	Consider value octal regardless of default radix.

The 4 remaining operators may be used in conjunction with each other on a line, they have no assumed Heirarchy they are executed in strict left to right sequence.

+	Add the two values or labels.
-	Has two functions. When used between two labels or values it produced the difference value. When used on its own before a label it negates the value (2's complement).
&	Produces the Logical AND value of two labels or values.
<	This is the logical rotate left or right operator. The form <4 will shift the bits of the value or label left by 4 bits. The form <-5 will shift the value of the operand by 5 bits in a right direction.

EDIT ERROR MESSAGES

The following messages may occur whilst using the editor.

"String not found"

The Find or Change command could not locate the string requested. If you are sure that it should have been there, you may have started the search from after the line with the string in it, in which case go to the top of the file and try again. Or you may have spelled the word incorrectly or used the wrong alpha case.

"Command format error"
The arguments you have provided in the command line are incorrect. You may have used an illegal wild card, or forgotten to include a comma or colon etc. See the section on editor instructions for the command you wish to use.

"No such line"
The line number you have requrested does not exist in the currently open file. You have probably done a file renumber and the line number no longer exists. If you know any reasonably unique words or labels that exist on the line, try to locate it with the Find command. If not you will just have to step through the file with the print command till you locate the area you require.

"File full"
The end of the file has reached the upper limit of memory allocated by the answer to "Memory size?" when you entered the editor. If this is really the top of your available memory, you will have to buy more memory before you can continue, or delete some comments from the file to make it smaller. Since there is no way of re-defining the memory size from within the editor, reallocating extra memory (if available) is a little messy. You may either save the file to cassette, reboot the editor to get the "memory size?" message and reload the cassette, or exit back to a MONITOR level and adjust the memory size bute directly. A list of the location of the main scratch areas is provided in the appendices.

"Illegal command"
The command letter used at the start of the line was not recognised by the editor, or the argument to the command was in an incorrect format. See the chapter on editor instructions for the command you wish to use.

"Line number too large"
Line numbers greater than 65534 are not permitted.

"No text in file"
You have issued a command that cannot be used on any empty file. eg tried to use Replace or Edit to start inserting into the empty file. In this instance use Insert mode.

"No room between lines"
In INSERT or REPLACE modes if the next line to be inserted (after step size added) will not fit below the next existing line in the file the editor will abort with this message. To continue the insert you may either reduce the step size by I.,1<ret> or renumber the complete file by N100,10<ret>

"No file here"
You have given an instruction to re-open an OLD file at a location where the editor can’t find a valid file. If you are sure that there should be a file here it is possible

that some location has been corrupted (possibly bad ram or a glitch on the mains etc). If you feel competent to try to find the bug, Read the section on the layout of a file and using your monitor try to find and fix the error. You can always reset to the file with an O command on re-entry.

"No Secondary file"

You have attempted to use the COpy or Merge commands when the Editor does not have an "open" secondary file. You have either forgotten to declare a secondary file (use ZS command) or have done an interim assembly or rebooted into the editor, your old secondary file will probably still be intact and may be reopened with the OS command.

"No room for merge"

When doing a merge from secondary to primary file, the Editor must first move the end of the primary file upwards in memory to provide a 'hole' into which the secondary file would fit. If this error message is displayed, the 'gap' between the two files is smaller than the length of the secondary file, and it would have been damamaged during the merge. You must therefore move the secondary file higher in memory. There are several ways of doing this, which one you should use depends on the particular situation you have. We suggest the following technique be used. (for example primary file at 2000, sec file at 3000) Query and record the status of the primary file. eg

Q<ret>
2000 2000 2F80 3FFF

Set up the secondary file, renumber and query its status. eg

Q3000<ret>
N100,10<ret>
Q<ret>
3000 3000 33FD 3FFF

(In all cases you MUST ensure that the secondary file is 'normalised' by renumbering it before proceeding after setting to a secondary file.) Notice that the gap between the file is only 80hex butes, and the secondary is nearly 400hex in length However there is more than its own length above itself. In this case we can create a copy of the secondary file higher in memory. eg ZS3000<ret> CO#:*<ret> O2000<ret> OS3800<ret> We are now back in the original primary file with the secondary now at 3800 and plenty of room for the merge. In some cases the size of the secondary file (or its location) may not allow us enough room to make a copy above itself. If the length of the secondary file is more than twice the gap from the end of the primary file to the end of memory we cannot do the merge in one operation anyway, however all is not lost. Remember that what we used to consider the secondary file is now our primary file and may be saved to cassette. After saving a copy of this file, we may be able to open a new file (with the Z command) at a location where when the file is reloaded

we will be able to do our merge. If not we can proceed to delete some of the end of it till it is small enough for the merge, then reload the saved copy of the old secondary file (at a suitable location), delete what we previously merged and remerge the remainder. There will be a lot of swapping between files, this is messy, but in an emergency justified.

ASSEMBLER ERROR MESSAGES

The following messages may occur whilst attempting to assemble a file.

"Bad label"
The "word" encoutered in the label field (extreme left) does not satisfy the requirements of a label. It must not be more than six characters long and must start with an upper case alpha character. No spaces or question marks may be imbedded within the label, and it must be separated from the mnemonic field by a space or tab.

"Branch out of range"
You have used a "relative" instruction (eg JR or DJNZ) to branch to a location in your program that is more than 128 bytes away. Either rearange your program to bring the destination closer, or use an "abolute" branch instruction (eg JP).

"Illegal format"
Your line of source is not laid out out in accordance with, or contains characters not supported by, the standard ZILOG requirements. Refer to " Z80-Assembly Language Programming Manual."

"Missing information"
The end of line was encountered beforee all information required had been read. You may have imbedded a semicolon in the line, or simply left out an argument.

"END missing"
The assembler found an end of file marker before the END statement, you have probably forgotten to insert one, or may have put it in the label field by mistake. Not a fatal error but will inhibit the generation of object code.

"Duplicate label"
This label has been previously defined, or may be one on the assemblers pre-defined list. eg use of HL or AF etc as labels will invoke this error message.

"Field OVF"
Whilst resolving arithmetic arguments in the operand field of a line, a value was produced that is greater than 65535 decimal (FFFFhex).

"Ref duplicate label"
This message will be invoked on all lines containing reference to duplicated labels.

"Symbol table OVF"
The symbol table being produced in pass 1 of the assembly has grown down to the point where, if assembly continued, the source code would be damaged. This is a fatal error and assembly is immediately aborted.

"Label not known"
You have attempted to reference a label which has not been defined in the label field. Usually invoked by spelling mistakes, or forgetting to assign scratch locations.

"Expression error"
The operand (address or value) field could not be resolved. It may contain a value or character that is not supported in the current radix default (eg alpha character in decimal expression) or applying a 16 bit mask to an 8 bit value.

THE MICROBEE ENVIRONMENT

Perhaps the most difficult part of any machine code program is the interface to outside world. This chapter will hopefully give the user an idea of how the MicroBee input and output facilities are used.

Fortunately, due to the well planned structuring of most of the software on the MicroBee, the input and output routines used by Basic are also easily used by any other program. For example, the program below is a simple demonstration of these routines.

```
        ORG     400H    ;Start code generation at 400-hex
INPUT   EQU     8006H   ;Location of input routine
OUTPT   EQU     800CH   ;Location of output routine
START   CALL    INPUT   ;Get one character from the keyboard
        LD      B,A     ;Transfer this character into B reg
        CALL    OUTPT   ;Send this character to the VDU
        JR      START   ;Go back and do it again
        END
```

The routine at 8006 will wait for a key to be hit and return the ASCII value of that key in the A register. For a full list of the ASCII character set of the MicroBee, see AppendiX-I. The routine at 800C will take a character in the B register and print it onto the VOU screen.

This program can be assembled with the A command and then excecuted by typing X 400. Anyone who has had previous experience with assembly language programming will appreciate the value of these general subroutines provided on the MicroBee, as tools for taking some of the hard work out of the task of interfacing their programs to the outside world. Some of the more useful routines availiable on the MicroBee are described below.

Note that some of these "routines" never return you to your program because they jump to such places as Basic. These are labeled with an *.

Decimal address	Hex address	Function
32768	8000	* Same effect as hitting reset
32774	8006	Wait for keyboard (in A)
32798	801E	* Execute Basic program (RUN)
32801	8021	* Warm start into basic
32804	8024	Initialise Hires graphics
32807	8027	Initialise Lores graphics
32810	202A	Initialise Inverse in PCG
32813	802D	Initialise Underline in PCG
32816	8030	Set dot X=HL Y=DE
32819	8033	Reset dot X=HL Y=DE
32822	8036	Invert dot X=HL Y=DE All these graphics routines return with flag set to NZ if coordinates out of range.
32825	8039	Test a dot X=HL Y=DE Returns NZ if dot set or error
32828	803C	plot a line X-start coordinate is in 80FD Y-start coordinate is in 80FF X-end coordinate is in 80F9 Y-end coordinate is in 80FB
32831	803F	Redirected Input (in A)
32834	8042	Redirected Output (from A)
32837	8045	Redirected print out (from A)

The redirected inputs, outputs and printouts correspond to the INPUT, PRINT and LPRINT basic commands respectively. The devices can be selected (as with IN#, OUT# or OUTL# in basic) by setting the bit maps in locations 00E4-hex (228-dec), 00E2-hex (226-dec) or 00E3-hex (227-dec) respectively. For example, if location E2-hex contains the number 21-hex (00100001-binary) then devices 5 and 0 are selected so any calls to the routine at 8042-hex will send output to the video screen (device 0) and also to the RS-232 port at 1200 baud (device 5).

The addresses of the routines selected by the bit map are kept in tables in scratch. The output devices are pointed to by a table at 00B2-hex, and the input device table is at 00C2-hex. This means that the user can create his (or her) own routines pointed to by the bit map by simply altering the table used to point to the routines. For example, you can write your own output

routine if you desire, and access it by putting the address of the routine into locations 00BE and 00BF-hex and setting bit six of the output device bit map (as long as output device 6 is not already used).

The tables below list the input and output device options selectable by the bit maps. The pointer addresses given are the addresses of the table entries that contain the addresses of the routines. For example, locations 00B4 and 00B5-hex contain the sixteen bit address of the routine to output a character to the parallel port.

OUTPUT DEVICE NUMBERS:		ADDRESS OF POINTER:
0	VDU output device (normal)	00B2-hex (178-dec)
1	MicroBee parallel port output	00B4-hex (180-dec)
2	300-baud cassette output	00B6-hex (182-dec)
3	1200-baud cassette output	00B8-hex (184-dec)
4	RS232 at 300 baud	00BA-hex (186-dec)
5	RS232 at 1200 baud	00BC-hex (188-dec)
6	Null routine	00BE-hex (190-dec)
7	Null routine	00C0-hex (192-dec)

INPUT DEVICE NUMBERS:		
0	Normal MicroBee keyboard	00C2-hex (194-dec)
1	Parallel port (for external keyboard)	00C4-hex (196-dec)
2	300 baud cassette	00C6-hex (198-dec)
3	1200 baud cassette	00C8-hex (200-dec)
4	RS232 at 300 baud	00CA-hex (202-dec)
5	RS232 at 1200 baud	00CC-hex (204-dec)
6	Null routine	00CE-hex (206-dec)
7	Null routine	00D0-hex (208-dec)

PROGRAMMABLE GRAPHICS CHARACTERS

The video screen of the MicroBee is memory mapped in locations F000-hex (61440-dec) to F7FF-hex (63487-dec). Any ASCII character stored in location F000-hex will appear at the top left corner of the screen, and F3FF-hex maps to the bottom right of the screen on a 16x64 screen format (this means that on a MicroBee with a 16x64 VDU format, the screen memory map only uses half of the availiable space for screen memory). Because each screen location contains one byte, there are 256 possible characters that can go in each character position the screen. If the top bit of the byte in a screen memory location is zero (i.e. the number is in the range 0..127) then the character will appear on the screen as a normal ASCII character.

The actual shape of these characters is defined in a Read Only Memory. This ROM is accessible to the programmer by a devious trick shown below. In addition, if the top bit of the byte in a screen location is one (i.e. the number is in the range

128..255) then the character that appears on the screen will be defined by the programmable character generator. This is a character generator contained in RAM for the user to define his/her own characters. The memory map of the video memory is shown below.

Decimal address	Hex address	Description
61440	F000	Screen map (top left corner)
62463	F3FF	Bottom right corner (16x64)
63359	F77F	Bottom right corner (24x80)
63487	F7FF	End of screen map
63488	F800	Start of programmable character generator
63503	F80F	End of first character (code 80-hex)
63504	F810	Start of second character (code 81-hex)
61440+(16*X)	FZZ0	Start of character code ZZ-hex (X-dec)
65535	FFFF	End of last character (code FF-hex)

The Read Only Memory for the character generator for the 128 ASCII characters is accessed by writing a 1 to output port 0B-hex (11-dec). When this is done, the screen memory map is removed and replaced by the character generator ROM. So, instead of having screen RAM for the first 2K and PCG RAM for the other 2K of the video memory, the video memory will contain all of the character generator for the VDU, with the first 2K being the permanent read only memory.

The VDU (for a l6x64 display), displays the text on the screen in 256 lines with 512 dots on each line. This is precisely the resolution of the hires graphics. A bit of quick arithmetic will reveal that each character must be composed of 8 dots per line and 16 lines. The creation of a programmable character is best described by example :-

CREATING A PCG CHARACTER

Maniputating PCG characters manually is most easily done in the Monitor, so get into the Monitor (by anyone of the methods list in chapter 2). Type E F300 to examine memory location F300-hex. It should contain 20-hex which is the ASCII code for the blank character. Hit M (for Modify) and type 80 (don’t hit return). This will put the character on the screen (at the left edge about 3/4 down the screen). The character that appears is the PCG character defined in memory locations F800..F80F. We can now change this character to one of our own.

Type <esc> (no carriage return) to get back into the Monitor command mode and type E F800. Hit M (for Modify) and then type 3C4242... as shown below.

```
F7F0 XX XX XX XX  XX XX XX XX  XX XX XX XX  XX XX XX XX
F800 9C B2 B2 67  25 25 3D 19  18 66 66 42  81 E7 A5 A5
F810 XX XX XX XX  XX XX XX XX  XX XX XX XX  XX XX XX XX
F820 XX XX XX XX  XX XX XX XX  XX XX XX XX  XX XX XX XX
F830 XX XX XX XX  XX XX XX XX  XX XX XX XX  XX XX XX XX
F840 XX XX XX XX  XX XX XX XX  XX XX XX XX  XX XX XX XX
```

These 16 bytes define the character that will appear on the screen whenever the byte 80-hex appears in the screen memory map (e.g. at F300-hex in this case). The way the character is defined is illustrated below.

```
Byte      Binary         Character shape
                         ++++++++++
9C        10011100       +X  XXX  +
B2        10110010       +X XX  X +
B2        10110010       +X XX  X +
67        01100111       + XX  XXX+
25        00100101       +  X  X X+
25        00100101       +  X  X X+
3D        00111101       +  XXXX X+
19        00011001       +   XX  X+
18        00011000       +   XX   +
66        01100110       + XX  XX +
66        01100110       + XX  XX +
42        01000010       + X    X +
81        10000001       +X      X+
E7        11100111       +XXX  XXX+
A5        10100101       +X X  X X+
A5        10100101       +X X  X X+
                         ++++++++++
```

This character will appear on the screen whenever a 80-hex byte appears in the memory location coresponding to that screen position. If we had put the 16 bytes into memory at FD50-hex (for example), then the character would have been defined as character number D5-hex instead of 80-hex.

SCRATCH LOCATIONS

Some of the more usable scratch locations have been described here as they were relevant to the routines that used them. A more comprehensive list of the MicroBee scratch locations is given in Appendix-G.

SOME HELPFUL HINTS

Probably the most helpful hint that can ever be given is that regularly creating backup copies is the best means of preventing major disasters. It has been stated that while ever mans fingers are pointed towards a keyboard they will occasionally get in front of his brain. Imagine wishing to print your latest hours of typing with an T#:* and accidentally pressing D#:* thereby deleting the entire file. It has happened!!! It may be the last backup was made a half an hour ago, but it is always easier to redo the last half hours work than to start from the beginning.

Get into the habit of always starting your source files with a comment line which has the file name, and or description, AND THE DATE. I t often happens that when you wish to reload from an old file there are several backup copies of it in existence. The date, and if desired the time, of the save will help to identify the most recent copy.

Always do your trial assemblies with the WE and NO switches specified. This will ensure that no crashes occur, and that you must attend to each error as it is reported.

Sometimes an error will be reported on a line, and yet you cant see anything wrong. It is possible that a non printable character may have been inserted into the line by either an editing error, or a power glitch. The best solution here is to delete the line and type it in again. If the error is still there, re read the manuals for the type of line you are inserting. The error message will usually help to pin it down.

If you wish to delete a line from the file, use the D command, don't try to edit it back to nothing with the backspace key, this can cause some unpredictable crashes under certain circumstances.

The two most common problems with assembled programs that don't appear to work, is forgetting to append a H to hex values, particularly when EQUating monitor calls, and not keeping the number of PUSH's and POP's balanced in sub routines. If your programs seem to crash in a great heap, try looking at these two problem areas first.

The line numbers inserted onto each line by the editor are not recognised by the assembler as labels, therefore you cannot use them as symbolic references for CALLS or JUMPS etc as you would in BASIC.

If whilst trying to list a file to the VDU the P command refuses to go past a certain point in the file, or appears to be looping back on itself, try renumbering the file, this will usually fix the problem. What has happened is that a faulty memory location (or a glitch) has caused a line number in the file to appear to be lower than the one before it. and the editor

has become confused.

The "end of file marker" is two bytes containing FF FF, this represents line number 65534 and all other lines will be placed before it. If for any reason one of these two bytes gets damaged, the file will appear to continue on into whatever rubbish happens to be in memory. The fix here is to step forward gradually through the file till you are sitting on the line before the crash, then use the Q command to locate your position in memory, and then use the system monitor to replace the two FF's at the end of the file. See appendix B for the exact layout of a file to get a better understanding of the problem. Remember that most crashes can be recovered from if you use a little thought before proceeding. Rushing into the "fix" will often only compound the problem.

GLOSSARY OF TERMS

ADDRESS Describes an actual memory location in the computer. with assemblers it is normal to refer to addresses in HEXADECIMAL notation. If the address starts with a letter it is correct procedure to prefix the address with a zero to avoid confusion with a word or label. eg 0BAD is an address.

APPEND Add to the end of. eg append a comment to a line.

ARGUMENT The value, address, or name that we wish the assembler to assign to the particular field. The argument may be a complex statement comprising arithmetic or logical operators.

ASSEMBLER See the chapter "what is an assembler".

BASIC An "english word" oriented interpreter, used to allow people not experienced in advanced programming techniques to create and run their own computer programs.

BLOCK MOVE A command that allows you to physically move the location of a file or collection of bytes.

BUFFER An area of memory set aside for storage and processing of commands, editing of lines, resolving argument fields etc.

BYTE An 8 bit hexadecimal number (00 to FF) which represents the 256 different values that can be stored in one location of computer memory.

COMPILER Any program which can produce a machine or object code output from a mnemonic source file.

COPY A command which allows you to duplicate a line, or lines, into another part of memory.

CP/M A disk based operating system for 8080 and Z80 processors. (copyright by Digital Research U.S.A.).

CURSOR A pointer to your current location in the line or file, usually shown as a flashing square on the display.

CONTROL CODE These are special key codes typed on the keyboard to instruct the program to perform a certain task. Sometimes a special key is provided, other times you must simultaneously press the CONTROL key and a letter key. the TAB key is the same as control I. abbreviated as CntrlI or ^I.

DEFAULT The condition or value that the program assumes in the absence of a specific value being given by the user.

DELETE The act of removing a line or group of lines from the file.

DELIMITER A character (usually <ret> or /) used to signify the end of the line, or argument within the line.

DGOS A machine language operating system sold by Applied Technology for use with their S100 processor series. (DGOS is copyright by Mr D. Griffiths).

EDIT The method of altering, or correcting a line in the file.

EDITOR See the chapter "what is an editor."

EQUATE To assign a value to. To define the meaning of.

FIELD Refers to a sub section of a line. eg label field, mnemonic field, operand field, comment field.

FILE Any collection of words, letters, characters, numbers or data stored on cassette, disk, or in the computer’s memory.

FREE FORM Means that the program is not affected by the precise layout of your entry within certain constraints, eg you may use single or multiple spaces instead of tabs etc.

GLOBAL All inclusive, not just limited to the area you are working in.

REX or HEXADECIMAL A system of counting with a base of 16 rather than the more familiar base of 10, comprises the digits 0 to 9 followed by A to F.

INSERT The act of providing a new line, or lines into the file.

LABEL A "word" or group of characters, starting with a letter, which defines a particular location or value.

LINE The collection of words starting with the LINE number and ended by pressing the the <ret> key that comprise an instruction, or command, for the assembler to process.

MACHINE CODE The collection of hexidecimal numbers read directly by the processor that comprise a program. Also referred to as OBJECT CODE.

MERGE The act of bringing back into the file a line, or group of lines, from elsewhere in memory. The converse of COPY.

MICROBEE A small, self contained, z80 based microprocessor, sold by Applied Technology pty Ltd.

MNEMONIC A "word" or symbol, often heavily abbreviated which refers to a specific task you wish the computer to perform.

MONITOR A program (usually in EPROM) used to perform the essential tasks of loading tapes, printing to the VDU, reading the keyboard, etc.

NULL LINE An empty line, used to visually break up the program into modules. This is created by pressing <ret> only as a line entry. Null lines are ignored by the assembler.

OBJECT CODE The collection of hex numbers that comprise a computer program, the output from the assembler is object code, also referred to as MACHINE CODE.

OFFSET A constant value added to the address when the assembler is outputing the object code, this causes it to be stored in a different memory location from the one where it would normally be run.

OPERAND The value field. It is in this field that the value to be assigned, or the address to be used is determined.

ORG The origin address specified in your source file that directs the assembler as to where the program is required to operate in memory.

PATCH An alteration to the program that is done outside the main body of the program. Usually placed at a location where it is convenient for the user to be able to modify, or customise the program for his own needs.

PERIPHERALS Additional devices connected to the computer to perform specific tasks. PRINTERS, CASSETTE recorders, etc are examples of peripheral devices.

PROGRAM A group of commands, or instructions, that direct a processor to perform a specific task.

PRINTER The computer "Hard copy" device, an electronic typewriter of some form, connected to the computer.

PSEUDO Literally ’false’. Pseudo mnemonics, although not really a defined mnemonic, are used in the mnemonic field to specify certain tasks to be performed.

REPLACE This command deletes one line from the file, and allows you to insert a new line, or lines in its place.

RENUMBER The automatic process of re-adjusting all the line numbers so that they are in ascending sequence, with equally spaced steps.

SCREEN This is an alternate word used to describe the VDU or T.V.

SOURCE FILE The list of instructions, created with the editor, and read by the assembler, to create your machine language program.

SOURCE LISTING The printed listing from the assembler that shows the source file with the addresses and object code produced for each line.

STATUS A display of the START, CURRENT POINTER, and END addresses of the file, plus the currently set upper limit of memory.

STRING A group of characters, similar to a sentence in normal speech.

SWITCHES Two letter groups, used to direct the assembler to perform particular tasks during assembly, if not specified, each switch is considered to be in a OFF state.

SUB EDIT Whilst in EDIT mode, an alternate set of command letters are used to direct the processor, these are called SUB EDIT instructions.

SYSTEM A collective term refering to the group of components which make up your "computer".

VDU The computer display device, usually a modified TV receiver.

VIEW The ability to look around the area you are currently working in without altering the current pointers.

WILD CARDS Characters used to perform a task within certain broadly specified restraints, eg P.:* means display from current location to end of the file, or G* means load in any program from tape regardless of name or type.

ZILOG The American company who developed and produced the Z80 processor system, and laid down the preferred MNEMONICS to be used to refer to its many operation codes.

z80 A CPU chip which is an advanced, upwards compatable, version of the 8080 series of processors. As well as its instructions, a Z80 can also execute own all of the 8080 instructions.

8080 A CPU chip developed by INTEL U.S.A. The predecessor of the Z80 processor that you are currently running.

Appendix-A WHAT IS AN "EDITOR" ?

Essentially an EDITOR is a program which allows you to create in memory a "file" containing text. You may use the editor to enter, correct, or rearrange your file and to save and retrieve it from some storage medium, usually (disk or cassette).

There are two types of editors in common usage, these are known as "SCREEN" editors and "LINE" editors. The screen editor is the more sophisticated of the two, it uses the T.V. screen (VDU) as a "window" into the file. To visualise this consider a piece of card with a cutout that is the width of the printing on this page and about 20 lines high, you can slide the card up and down the page and see the window effect in action. In the screen editor you have a flashing point of light (cursor) that can be moved anywhere on the display. If you try to move the cursor above or below the screen, the window is automatically moved up or down the file so that the cursor is always on the screen. As you insert or delete letters or words, the entire screen display is redrawn to show how that part of the file is layed out.

The really sophisticated screen editors provide features like justifying the left and right edges of the lines so that they are both straight as in this manual. These editors are usually referred to as "word processors", they are very expensive programs to purchase, use up a large amount of processor memory, and require a "DISK" system for file storage. This is because the editor itself is so large that there is not much room left in memory for the file, therefore it must be edited directly from the disk. Enough of this dreaming of what might be, you are stuck with a "line" editor.

The line editor is quite different in concept, it considers your file to be a collection of LINES, each line has a maximum length, usually the width of your TV screen; these lines are considered to be in sequential order usually with a LINE NUMBER attached to them. The line numbers are usually arranged to increment by 10 as each line is inserted into the file. This is done to allow extra lines to be inserted between existing ones. The main disadvantage here is that if you need to insert more than 9 lines between any pair of existing lines you must renumber the file and then any printout you may be using to edit your file from will disagree in numbering. In practice this is not a major problem.

To use the editor to create a file, you just type the required lines whilst in INSERT mode, remembering to press the return key at the end of each line. Up to this point it is nearly as easy to use as a screen editor, however here the similarity ends. If you wish to change something in the file, you may direct the editor to go to a line number that is near where you wish to make the change, move the cursor (now called a CURRENT LINE POINTER) up or down the file till it points to the line you wish to change, and give an appropriate command to DELETE, EDIT, INSERT, REPLACE, etc. If your command was Edit, this version of

the Editor is provided with a 'SUBEDIT' package which gives you some of the advantages of a screen editor, however you are confined within the bounds of the line you are currently editing.

To make life a little bit easier (who said it wasn't meant to be), we have provided commands to allow you to search your file for particular words or phrases, and if required replace them with alternate ones.

Although line editors usually have line numbers associated with them, the editor does all the allocation and distribution of them, Since over 65000 are allowed you will always run out of memory long before you run out of numbers, unless you number your file in increments of a hundred or so. The file may also be typed with the line numbers automatically suppressed for letter writing etc.

Appendix-B WHAT IS AN "ASSEMBLER" ?

Although a computer may appear to be an extremely complex and intelligent piece of equipment, it is in reality a number of PERIPHERALS (VDU, keyboard, memory, printer, cassettes, etc) all connected to the CPU chip (Central Processor Unit). This tiny silicon wafer in a 40pin I.C. package has one claim to fame; it can do a small number (a couple of hundred) simple tasks, very quickly, and very reliably. It is instructed to do each of these tasks by a sequence of numbers which when strung together as a series of tasks produce a PROGRAM. It is the program, not the computer which produces the illusion of intelligence. Whilst the CPU unit in your MicroBee knows that the sequence C30080 means to go to memory address 8000hex and start "RUNNING" the BASIC interpreter we have installed there, you as the user cannot be expected to know what all the hundreds of combinations of numbers will mean.

It will be obvious by now that if you wish your computer to do any useful work for you it must be given programs to do do these tasks, so how do you provide these? You could go out and purchase all your programs. That is assuming they are available, and are within your financial resources. An alternate approach is to purchase a "HIGH LEVEL LANGUAGE" such as PILOT, BASIC, PASCAL, FORTRAN, COBOL, etc. These programs allow you to list your program requirements using a group of "english" like words sometimes referred to as MNEMONICS. When run, these programs "interpret" your words and provide the functions required.

The main disadvantages with this approach are the cost of the interpreters, the amount of memory they require to operate, restrictions in the tasks that can be performed, and the speed of execution is fairly slow. The speed factor is particularly important if you wish to play real time games such as "SPACE INVADERS". Whilst COMPILER versions of most of the above programs are available they are not suitable for small systems.

Having reached the subject of compilers we can now discuss the simplest compiler of them all, the ASSEMBLER. I use the word simplest only in the sense that you can't just say PRINT and have the program go away and produce a complete SUB PROGRAM to do a 'print message' function as you would with an interpreter; however in this simplicity lives its versatility, you have complete control over each and every instruction in the program.

As with an interpreter you provide a SOURCE FILE consisting of mnemonics, in this case each LINE of the source describes ONE instruction that you wish the computer to perform, you may also provide LABELS on any line so that you may later instruct the computer to go to this point in the program. Comments may also be inserted on the line so that you or your friends may later be able to work out what was intended when you wrote it. As well as referring to a location within a program, labels may also be assigned absolute values. This has the advantage of allowing you to alter this value everywhere in the program by just altering

the value at the location it is declared. Another reason for assigning values to labels is that it is often simpler to remember a name than the value assigned to it, eg it is easier and clearer to say TAB or SPACE than to remember that they are 09 and 32 respectively.

In case you haven't already guessed, the assembler interprets your source code and produces, directly in memory, the sequence of numbers referred to as MACHINE CODE or OBJECT CODE that make up the program ready for your processor to execute at full machine speed. It is good practice to save your source file on cassette in case you wish to make alterations in the future, or wish to assemble the program to operate at a different location in memory. The latter can be done by simply changing the origin (ORG) address and re-assembling the file.

Appendix-C SOME "HANDS ON" EXPERIENCE

By far the best way to come to grips with any new processor operating system is to have someone demonstrate the program in actual use, and then try it out by yourself under supervision. Unfortunately we can't quite do this here but we can guide you through a tutorial with the sample program listed at the end of this Appendix. Hopefully this will give you a better understanding of the commands and their use when we describe them in more detail in the main sections of the manual.

First you must get into the program. From Basic, simply type EDASM. More details on getting into the Editor/Assembler are given in chapter 2. When you are asked Memory size?, reply with 4000 (if you have a l6K MicroBee) or 8000 (if you have a 32K MicroBee). After you hit <return>, you should get the usual Editor/Assembler prompt '>'.

Type Q<ret>
The program will respond 1000 1000 1000 3FFF

If you have a 32K MicroBee, the last number will be 7FFF. The first address indicates that the file will start at 1000-hex. The second address is the location of the CURRENT LINE pointer and is now pointing to the start of the file. The third address indicates that the file ends at 1000-hex (which is logical cause there ain't no file there yet) the last address is one byte lower than the upper limit of memory that you set on entry, the editor will not attempt to use memory above this point.

Now to enter our sample file:- After reading this paragraph, type I<ret> The program responds 00100 _
(_ represents the cursor)

You may then type in the file called "TEST" in appendix A. The program as listed contains some deliberate errors to demonstrate editing and error handling. If you make a couple more errors typing it in don't worry, we will show you how to correct them at the end. Don't forget to press <ret> at the end of each line. You will notice that each line is tabulated into columns, this tabulation is normally produced by pressing the TAB key (or Control and I simultaneously) at each break instead of the SPACE key. This neat layout format is to make the code more easily read by you and is not essential for the assembler, it is quite happy with just single spaces between each FIELD.

If you make a complete bollox of it first time just reset the processor and start again. Notice that the editor inserted the line numbers for you. N.B. WHEN YOU HAVE FINISHED THE LAST LINE TYPE ^A or ^C. THIS MEANS PRESS THE CONTROL KEY AND THE A or C KEYS SIMULTANEOUSLY, this will get you out of the INSERT mode and back to the editor.

You should now be back in the editor with the "*" prompt showing and the test file typed in, with or without a few more

typing errors of your own. It is suggested that you save the file on cassette now to save having to retype it all back later. Do this by starting the recorder and typing S "TEST"<ret> The editor will reprompt with a * when it has completed the save.

To examine the file you have typed in, type P#:*<ret> and the entire program should be listed to the VDU (a bit too quickly to read). Now type P#<ret> and only the top line will be printed, each time you press the Line Feed key one more line will be displayed, by this means you can step through your file line by line. press the cicumflex key '^' and you will see that you will step backwards through the file one line each time you press the key. Back to the top of the file again by P#<ret>, now press P<ret> and you can step through the file a screen full at a time.

Lets assume you really mucked up line 200, the easiest way is to REPLACE it by typing R200<ret> and type the line in again, end the line with a return as normal. Notice that you press return twice, first after the command and again at the end of the line. The program should return to the editor with the message "no room between lines". Normally the replace mode would allow you to replace the. line with more than one line, but here after replacing line 200 the counter was incremented to 210 and the program would have destroyed an existing line if you had continued. If you wish to put in more than 1 line use the command R200,2<ret> this would allow you up to 5 lines numbered 200, 202, 204, 206, 208 before it aborted out, you can of course finish earlier by the pressing Control and C (or A).

There are two important points to note, here firstly you must not press the Control C at the end of the last line before you exit or it will not be inserted into the file, end the line with a <ret> and type the AC when it prompts with the next line number (which you do not wish to enter). Secondly if you used the format R200,2 your line number increment counter is now set up for steps of 2, and all future changes will be in steps of 2 until it is reset to some other value. The step size may be altered by appending a comma and new step value to the REPLACE, INSERT, or RENUMBER commands. You will have noticed that the editor starts up with the first line number as 100 and steps of 10. These are the DEFAULT values.

If you made any typing errors you may either fix them now with the replace command, or wait till after we have examined the EDIT function and edit out your errors.

Lets EDIT line 100, type E100<ret> the editor prints 00100 with the cursor after the number, ready for a sub edit command. Press the space bar a few times, you will notice that the characters are gradually revealed on the line, the space bar is doing a "non destructive" forward space function. Now type L and the entire line is displayed and the cursor is redisplayed at the start of the line ready for another edit of the line. Now type X and the line is displayed with the cursor sitting at the end of the line ready for you to append a comment. Now type FRED LIKES

NUTS<ret> and edit the line again, you will notice that your comment has been added to the end of the line. Use the X sub command to go to the end of the line again and use the backspace key to step back over your comment then press return, your comment has been removed from the line. This demonstrates that backspace is "destructive". If the error you wish to correct is near the end of a line it is often faster to go to the end of the line, backspace over the error and retype the end of the line.

The line should be back to something like its original condition. Now type E100<ret> again, press the space bar to reveal the ";" as before and type 4DIANOTHER<ret> the 4D instructed the editor to delete the next 4 characters, then you told it to insert the following characters, and the "<ret>" caused it to finish the edit of the line. Your line now reads 00100 ;ANOTHER program for "skywriter". If what you wished to replace was the same length as what you were replacing you could have typed 4CBILL<ret> which would have CHANGED the next 4 characters to BILL. There are many other commands in the sub edit mode but what you know now should allow you to fix up any mistakes you may have made whilst typing in the test file, we will deal with the rest in detail later.

One final hint at this stage, if you really make a muck up of your file and wish to start again from the version saved on cassette type Z<ret> to erase the file in memory, then type G "TEST"<ret> and play the tape you recorded before. After the tape has loaded you should be back at the Editor-Assembler message with the file in memory ready to edit again.

Assuming that all has gone well and your file looks exactly like the one in the appendix we are ready to try to assemble it into a working program. If the version you previously saved to cassette is not correct, it would be advisable to resave the file now.

To do a trial assembly type A/NO/WE<ret>. The 2 letter groups following each slash are termed SWITCHES and these direct the assembler to do specific functions during the assembly. The 2 switches used here are:-

NO means No Object code is to be output into memory.
WE means Wait if you find an Error. (and report it)

If all has gone according to plan you should have seen several screen fulls of SOURCE LISTING scroll up the VDU, and when line 400 was reached, the assembly was stopped and an error reported. If you now type Control and C simultaneously you will find yourself back in the editor with the CURRENT LINE POINTER at the start of the line with the error in. Now type E<ret>L and the line will be displayed. What the assembler was complaining about was an unknown label called STRNG, you will notice that when we defined it in line 480 it was called STRING. To fix this up type X then two backspaces and ING<ret> which should fix this error. Now try the assembly again.

Again the assembler should stop at an error, this time on line 490, now the error is reported as an "argument error". What we are being told is that the value 0A is incorrect, remember that in-its present setup the assembler is expecting all values to be decimal and 0A is a hex number. The fault can be fixed by changing the value to 10 (the decimal equivalent of 0Ahex) or by simply adding a H to the end of the line to tell the assembler that we require a hex value. This time the file should assemble without any errors.

If any other errors are reported you should be able to work them out by reference to the chapter on "assemb1er errors" and rechecking your file against the listing in the appendix.

I cannot stress the concept of backup too strongly, now that you have a working file, save it to cassette.

Note that a list of all the labels, with their values was printed at the end of the assembly, this is called a SYMBOL TABLE. As a final check before we generate the object program check that VDU is shown as F000 in the symbol table and START is shown as 3000. It is so easy to forget to put the H after hex values and unless the value contains an ascii character the assembler will happily accept it as decimal.

The final step now is to assemble the file and generate the program in memory. To do this type A/WE<ret> it is always good practice to use the WE switch since it is better to stop the assembly if an error occurs than to miss seeing the error reported, and try to run a faulty program.

When the assembler has finished and returned to the editor prompt you may attempt to run your program with the eXecute command X3000<ret> the top 4 lines of the screen should now be cleared and a small rocket should slowly circulate round the top of the screen producing a banner with the alphabet on it. In this simple program I have not attempted to allow any exit, it will continue ad nauseam till you reset the processor. Resetting the processor may with some systems destroy the source file, and you may need to reload from cassette to continue experimenting with the source file.

TEST SOURCE LISTING

This is the test program you are required to enter for the tutorial. Note that you enter exactly what is shown here with the exception of the numbers shown in the right hand column; these are shown for your convenience and should match the number automatically inserted by the editor at the start of each line. If you make a mistake whilst typing a line you may use the backspace key to fix it, If you do not notice it till after you finish the line, wait till we have shown you how to EDIT lines.

```
;Test program for "skywriter"                                  00100
VDU     EQU     0F000H                                        00110
SPACE   EQU     20H                                           00120
        ORG     3000H                                         00130
START   LD      HL,VDY                                        00140
        LD      A,'A'                                         00150
        LS      (CHAR),A                                      00160
CLEAR   LD      (HL),SPACE      ;Clear top of VDU             00170
        INC     HL                                            00180
        LD      A,L                                           00190
        OR      A                                             00200
        JR      NZ,CLEAR                                      00210
        LD      HL,VDU                                        00220
FLY     LD      DE,0            ;Speed value                  00230
WAIT    DEC     DE              ;Slow down display            00240
        LD      A,D                                           00250
        OR      E                                             00260
        JR      NZ,WAIT                                       00270
        LD      A,(CHAR)                                      00280
        LD      (HL),A          ;Put char to VDU              00290
        INC     A                                             00300
        CP      'Z'+1                                         00310
        JR      C,STORE                                       00320
        LD      A,'A'                                         00330
STORE   LD      (CHAR),A                                      00340
        INC     L                                             00350
        PUSH    HL                                            00360
        CALL    PLANE                                         00370
        POP     HL                                            00380
        JR      FLY                                           00390
PLANE   LD      DE,STRNG        ;There is an error here       00400
PRINT   LD      A,(DE)                                        00410
        OR      A                                             00420
        RET     Z                                             00430
        LD      (HL),A                                        00440
        INC     L                                             00450
        INC     DE                                            00460
        JR      PRINT                                         00470
STRING  DB      '>'                                           00480
        DB      0A              ;Another error                00490
        DB      9                                             00500
        DB      0                                             00510
CHAR    DB      'A'                                           00520
        END                                                   00530
```

Editor.

B	Exit to 'monitor'	Q	Query file status
C	Change /stringl/string2/	R	Replace lines
CO	Copy to secondary file	S	Save file called "NAME"
D	Delete lines from primary file	T	Type, no line numbers
E	Enter edit sub mode	V	View lines around current
F	Find/ string/	X	Exit from editor to address
G	Get from tape "NAME" or *	Z	Create new primary file
I	Enter insert mode	ZS	Create new secondary file
L	List to printer	^	Step back one line
M	Merge from secondary file	'	Freeze VDU during scroll
N	Renumber priamary file	B/S	Destructive backspace
O	Open old primary file	L/F	Step forward one line
OS	Open old secondary file	CtlC	Exit insert mode
P	Print file to VDU		
C/R	End of line character		

Sub edit

A	Ignore changes, restart edit	L	List full line, restart edit
nC	Change next n characters	Q	Quit do not alter original line
nD	Delete next n characters	X	Insert from end of line
E	End edit include changes	SPACE	Non destructive move right
H	Delete rest of line, & insert	C/R	End edit return to editor
I	Insert string into line		
nKx	Kill all chars to the n'th x		
B/S	Destructive move left		

Assembler

A <ret>	Normal assembley with object code produced (space between A and <ret>)
Annnn <ret>	Assemble with object offset in memory by nnnn
A/Sl/S2/S3/S? <ret>	Assemble with switch control
Annnn /Sl/S2/S? <ret>	Assemble with offset under switch control

' Freeze display during listing, any other key continues listing

Switches

WE	Wait if error found	NS	No symbol table displayed
NO	No object code produced	LP	Produce full listing to printer
NL	No source listing		
PT	Print strings in full		

After error encountered during assembly

Control C	Return to editor with error line as current editor line
C	Clear down error switch (no wait if extra errors)
Any other key	Continue assembly with error switch still on

Appendix-E Microbee MONITOR

Since Microbee, in its standard form is essentially a "BASIC" only microprocessor, with the ability to run pre-recorded machine code programs, it was decided to provide some of the functions found in a conventional MONITOR based system. These should be considered as helpful tools to create, modify, and run your own machine level programs rather than a complete operating environment. The functions provided have been limited to those that can be most profitably fitted in the space at the end of assembler EPROMs. Since the start location will vary as changes or updates are made to the Editot/Assembler it is entered by typing X <CR> form the Editor.

MONITOR COMMANDS

A nnnn <CR>
ALTER MEMORY, nnnn is memory address to be altered. (same as the Examine memory, with no need to type M to alter memory)

B <CR>
Return to basic.(cold starts back to basic)

C xxxx yyyy nnnn
Compare two blocks of memory. xxxx is start address of the first block, yyyy is the start address of the second block and nnnn is the number of bytes to be compared. Any differences between the first block and the second block will be displayed on the bottom half of the screen in the following format: aaaa ff ss
aaaa is the address of the difference in the first block.
ff is the contents of the first block.
ss is the contents of the second block.
The Compare command will be terminated when the total number of bytes (nnnnH) has been compared or when the screen is filled up. If there are too many differences to fit on the screen, compare smaller blocks.

D "FILE" M xxxx yyyy nnnn <CR>
Dump file at 1200baud. File name no longer than six letters and enclosed in double quotes. M is a single character filetype. This should be a letter to indicate the type of the file: e.g.
B for BASIC files
M for MACHINE language files
S for SOURCE files
D for DATA files.
Note that BASIC will only load file types "B" and "M".
xxxx is the start address of the data to be written to cassette.

yyyy is the end address of the data to be written to cassette (inclusive).
nnnn is an optional auto execute address. This auto execute address is stored in the header and defaults to the same as the start address if no addressis given. BASIC uses this to run "M" type files, but the machine language monitor does not use it when a file is Read.

E xxxx <CR> Examine memory location xxxx. This will produce a hexadecimal core dump on the screen with a cursor indicating the byte addressed by xxxx. The cursor can be moved left, right, up or down by depressing the CTRL key and the letters A, S, W or Z respectively (^A,^S,^W,^Z). This interactive core dump is particularly versatile entry and examination mode whose merits will soon become apparent to you. To change the contents of any location, when the cursor is pointing to the desired location press the "M" key to authorize the modify mode and then enter the desired hexadecimal value(s). Entry, can be continued without haveing to rekey "M" until the cursor is moved by cursor control or the Examine command is terminated. The examine command can be terminated at any time by pressing the "ESC" key. This will return control to the command mode but will leave the last core dump on the screen.

v <CR> This command simply clears the screen and allows the user to use the Microbee as a "T.V. or Glass typewriter". The text is of no use as it cannot be printed or saved, but the user can get an idea of what control characters are supported by the VDU driver etc.

W "FILE" M xxxx yyyy nnnn <CR>
Write a file to tape at 300 baud. Same format as the Dump command.

X <CR> This command will jump to the Editor / Assembler (if the X command is used in the Editor/Assembler it will jump you back to the Monitor).

F xxxx yyyy zz <CR>
This command will FILL memory from the start address xxxx to the end address yyyy with the value zz Hex (if no value is given

it will default to 00 Hex).

G xxxx <CR> This command premits program execution to commence from any address above 01FF Hex. The warm start address of the machine language monitor is pushed onto the stack before jumping to the user's address, so that the user program can return control to the monitor by executing a "RET" (e.g. C9Hex) instruction.

M xxxx yyyy llll <eR>
Move block of memory. Move the block of memory at location xxxx to location yyyy. llll is the number bytes to be moved, it is a Hex value(llll is the LENGTH of the block)

P <CR> This command clears the screen (page clear) and returns to the command mode.

R "FILE" xxxx <CR> This command Reads tape files. The file name is optional but can be used to load a particular file off a tape. xxxx is also an option to load the file to address xxxx in memory. The machine language monitor will initialise the tape routines and start looking for a valid header (produced by the "D" or "W" commands). If no filename is given, the first file found will be loaded. If a filename is given the monitor waits for the correct file before loading; however, each header will be displayed as it is encountered. This allows you to observe what files are on the tape.
If a CRC error is detected during a read, the read will be terminated and a "C" put at the end of the command line. To verify a machine language file use: R "FILE" 8000 <CR> as this will read the file into ROM space and hence nothing will be destroyed in memory in case of a bad Dump or Write.

DGOS TAPE FORMAT:

-NULLS	16	At least 16 null characters (00H).
-SOH	1	Start of header character (01H).
-NAME	6	Filename. Nulls in unused positions.
-TYPE	1	Filetype. single ASCII character.
-LENGTH	2	Length of file.
-LOAD ADDR	2	Load addresses.
-AUTO ADDR	2	Execute addresses. (see text)
-SPEED	1	Speed: 01H for 1200 Bd & 00H for 300 Bd.

-EXEC	1	00H for no auto execute, FFH for auto execute.
-SPARE	1	Spare byte (not used).
-CRC	1	CRC byte for header.
-DATA	256	Byte data block. (see note)
-CRC	1	CRC for data block.

NOTE: All data blocks are 256 bytes long, except for the last one which be from 1 to 256 bytes long to make up the total length of the file. The last block will be followed by a CTRL character.
Loading can be terminated at any time by pressing the RESET key.

S xxxx yyyy aa (bb) (cc) (dd) (ee) (ff) <CR>
Search from start address xxxx to end address yyyy. aa is the byte to be searched for. bb to ff are (optional) bytes to searched for. Any address at which the byte or bytes are found will be displayed on the lower half of the screen.

NOTE: If you enter a character that is not 'understood' by the monitor a flashing question mark will appear. Use the 'BACKSPACE' or 'ESC' key to cancel the error and return the monitor to the command mode.

Appendix-F LAYOUT OF A FILE

For those who wish to attempt to recover a "crashed" file, or convert a differently formatted file for use with this editor, the precise layout is defined here.

There is no start of character, nor is there any end of line character stored on the line, the lines are stored sequentia1y, in ascending order of line numbers. Each line consists of a 16 bit value representing the line number, normal 8080/Z80 address format is used, that is the low byte first. eg line number 100 will appear as 64 00 (100 decimal is 0064 hex). Following the line number is a single hex byte representing the number of characters (not counting the line number or itself) that are in the line. The actual characters (in hex) of the line follow the length byte. After the last line of the file is the "end of file marker" which consists of 2 bytes, doth FF, the end of file marker serves two. purposes, FF FF represents line number 65535, and ensures that all other lines will be stored below itself. It also represents the end of the file. Note that the end of file pointer displayed by the Q command shows the location of the first of these two bytes. And any attempt to save the file from outside the environment of the editor must include both FF's.

To satisfy the editor that a valid file exists, It must be capable of stepping through the file by means of the length bytes, until it finds the end of the file marker below the end of memory address. Whilst stepping through it must not find a length byte longer than 7F hex, nor must there be any characters in the line that have the sign bit set (reverse video). If any of these criteria are not met, the message "No file here" will be displayed.

The following is a sample file, at 2000hex, and the memory image produced.

After printing the file, the status displayed by the Q command was:

```
Q <CR>
2000  2026  202D    3FFF


00100:TEST LINE
00110       ORG   100H
00120       JP    0D000H
00130       END



2000    64 00 0A   3B 54 45 53 54 20 4C 49 4E 45
200D    6E 00 09   09 4F 52 47 09 31 30 30 48
2019    78 00 0A   09 4A 50 09 30 44 30 30 30 48
2026    82 00 04   09 45 4E 44
202D    FF FF
```

Appendix-G EDITOR SCRATCH LOCATIONS

This list is not comprehensive, but mearly covers those locations you are most likely to wish access to. When shown as xxxx/y it means that 16 bit value is used.

0200/1	Pointer to start of currently open primary file.
0228	Secondary file open flag.
0229/A	Start of secondary file pointer.
022B/C	End of secondary file pointer.
022D/E	Current line pointer.
022F/0	End of memory pointer. Set up on entry to editor.
0231/2	End of file pointer (points to first FF).
0233	step size between line numbers.
0243/C3	128 character general purpose buffer.
02C4/5	Pointer to current character being accessed in buffer.
02C6	Size of line currently in buffer.
02D4/5	Error count. Keeps count of number of errors during assembly.
02D6	I/O suppress flag. Non zero suppresses output to both VDU and line printer. Used to suppress listings under NL switch etc.
02D8	Printer flag. Non zero causes printer to be used instead of VDU.
02DF/0	Pointer to bottom of symbol table. Only valid after compleation of an assembly.
02E1	Suppress line numbers on printout if non zero.
02E2	File initialized flag. Editor will ask for "Memory size", and create a null file at the default file address if this location is not set to 55hex on entry to the Editor.
02E9	Current value default radix.
02EA	Reserved for line number of printouts.
02EB	Reserved for page number of printouts.
02F0/5	Assembler switch storage scratch.
02F7/06	Cassette name compare buffer.
0207/16	Cassette header and address buffer.
0217/97	Return address stack.

```
BFF5    :;---------------------------------
BFF5    :;      BASIC SCRATCH AREA
BFF5    :;---------------------------------
0000    :               org     srtch
0000    :; 100H boundary ..
0000    :hash_table     equ     .
0000    :               defs    128
0080    :;Need {CTC} vectors to 10H boundary ..
0080    :move_here      equ     .       ;NZ scratch start
0080    :ctcv1  defs    4*2
0088    :piov1  defs    2
008A    :piov2  defs    2
008C    :break_disab    defs    1       ;disables break key if nz here
008D    :save_disab     defs    1       ;disables SAVE etc if NZ
008E    :               defs    '20'-(6*2)-2-7  ;reserved
0099    :col_flag       defs    1       ;=255 if colour BEE, else 0
009A    :col_mrsl       defs    1       ;color mode byte result
009B    :col_rslt       defs    1       ;normal byte result
009C    :col_fore       defs    1       ;foreground color
009D    :col_back       defs    1       ;background color
009E    :col_mode       defs    1       ;color mode
009F    :ucl_flag       defs    1       ;control UC mode of LIST
                                        ; nz=list in lower
00A0    :;
00A0    :stack  defs    2       ;top of memory=stack
00A2    :rst_jump       defs    2       ;warm-start jump address
00A4    :chk_byte       defs    2       ;55AA if initialized
00A5    :save_exec      defs    2       ;machine language exec address
00A8    :jp_mem defs    3       ;modifiable jump vectors for keywords
00AB    :jp_net defs    3
00AE    :jp_edasm       defs    3
00B1    :               defs    1       ;reserved
00B2    :;
00B2    :; this is the input and output vector tables
00B2    :out_tab        defs    2*8
00C2    :in_tab defs    2*8
00D2    :;
00D2    :; this is the crtc table, copied to ram for esc a,s,w,z
00D2    :crtc_tab       defs    2
00D4    :crtc_hor       defs    5
00D9    :crtc_ver       defs    3
00DC    :crtc_curs      defs    6       ;cursor control byte
00E2    :;
00E2    :out_dev        defs    1       ;select output from above table
00E3    :outl_dev       defs    1       ;select lprint output
                                         from out_tab
00E4    :in_def defs    1       ;selects input from in_tab
00E5    :;
00E5    :vdmode defs    1       ;video mode control byte
00E6    :speed  defs    1       ;vdu delay period
00E7    :chars_used     defs    1       ;my addition for USED, etc.
00E8    :plot_type      defs    1       ;has "R","I" or space
00E9    :lo_cycle       defs    1       ;controls tape routine speed
```

```
00EA    :RS_baud        defs    1       ;controls RS232 speed
00EB    :; this is a shared scratch : graphics plot / tape routines
00EB    :gr_pseudo      equ     .
00EB    :required       defs    6       ;the load name to match
00F1    :header defs    6       ;dgos type header block
00F7    :fltype defs    1       ;file type character
00F8    :flleng defs    2       ;length of file
00FA    :flstrt defs    2       ;normal starting address
00FC    :flauto defs    2       ;jp to this address if flexec nz
00FE    :flspeed        defs    1       ;0 -> 300bd, nz -> 1200 bd
00FF    :flexec defs    1       ;is nz if this is auto execute
0100    :flprotect      defs    1
0101    :;
0101    :; From here on the scratchpad may change between versions
0101    :alpha_rev      defs    1       ;non z -> alpha reversed
0102    :key_down       defs    2       ;this key-code is down
0103    :rept_count     defs    2       ;controls speed of repeat key
0105    :rept_flip      defs    1       ;defines if retrace or not
                                         at last look
0106    :last_ascii     defs    1       ;keeps last char send
                                         by scn_in for repeat
0107    :keep_char      defs    1       ;keeps an ASCII code
                                         for inkey$'s use
0108    :kbds           defs    1       ;for port a keyboard
0109    :port_ack       defs    1       ;zero if parallel port out
                                         usable
010A    :buff_count     defs    1       ;no. bytes in cassette
                                         redirect buffer
010B    :cursor defs    2       ;cursor address 'F000'+
010D    :sec_flag       defs    1       ;nz if last vdu_out char
                                         was excape
010E    :holdde defs    2       ;save de during ops
0110    :showcr defs    1       ;let list-line do inverse
0111    :trwd           defs    1       ;ignore o/p ovf STR
                                         if -1 else max/line
0112    :svelps defs    2
0114    :sveaut defs    2       ;save de during ops
0116    :uplink defs    2       ;scratch for renum
0118    :sram           defs    2
011A    :rnd            defs    '10'    ;random number seed
012A    :load_opt       defs    1       ;holds load options
012B    :;
012B    :free           defs    1       ;pcg storage ..
012C    :vdu_addr       defs    2
012E    :new_char       defs    pcg_charz
013E    :contpos        defs    2       ;continue address (de)
0140    :contstart      defs    2       ;start of currently
                                        executing statement
0142    :err_trap       defs    2       ;=line num if ON ERROR active
0144    :err_code       defs    1
0145    :err_line       defs    2       ;status of last error
0147    :;
0200    :               org     (.>8+1)<8
0200    :; 100H bounary ..
```

```
0200    :;*** from here may be overloaded by machine language progs. ***
0200    :;* this should be at scratch+'200'
0200    :ref_counts     defs    n_pcg_chars*2
0300    :free_recs      defs    n_pcg_chars*2
0400    :;
0400    :; Need 100H boundary           org     (.>8+1)<8
0400    :astck  defs    '100'
0500    :avar           defs    '1B0'
06B0    :ivar           defs    26*2
06E4    :fnstor defs    '10'
0700    :               orf     (.>8+1)<8
0700    :;      Fore to 100H boundary
0700    :fpbuff defs    '28'
0728    :ibufs  defs    '100'-'28'
0800    :ovufs  defs    'C0'
08C0    :
08C0    :sd             defs    1
08C1    :fw             defs    1
08C2    :dp             defs    1
08C3    :tmp1           defs    2
08C5    :tmp2           defs    2
08C7    :count  defs    1
08C8    :               defs    1
08C9    :auto           defs    1
08CA    :crlb1  defs    2
08CC    :tmp3           defs    2
08CE    :tmp4           defs    2
08D0    :pbgn           defs    2
08D2    :pend           defs    2
08D4    :stlv1  defs    2
08D6    :optr           defs    2
08D8    :vstrt  defs    2
08DA    :tmpa           defs    2
08DC    :sstrt  defs    2
08DE    :xswe           defs    1
08DF    :xsw1           defs    1
08E0    :prmt           defs    1
08E1    :sprmt  defs    1
08E2    :tmpc           defs    1
08E3    :zone           defs    1
08E4    :mode           defs    1
08E5    :tmpd           defs    2
08E7    :tmp5           defs    2
08E9    :dloc           defs    2
08EB    :flags  defs    1
08EC    :lims           defs    2
08EE    :autstp defs    1
08EF    :autln  defs    2
08F1    :tmp8           defs    2
08F3    :tmp9           defs    2
08F5    :ptrps  defs    2
08F7    :;stach moved (shouldn't be in table C), replaced by ..
08F7    :autdef defs    2       ;default for edit
08F9    :tmpf           defs    2
08FB    :ablv1  defs    1
08FC    :odvce  defs    1
08FD    :contln defs    2       ;CONT line No
08FF    :               defs    1       ;space filler
0900    :pstrt  equ     .
0900    :
0900    :end            equ     .
0900    :               end
```

Appendix-H ZILOG MNEMONICS

Alphabetical
Assembly Mnemonic Operation

ADC	HL,ss	Add with carry Reg. pair ss to HL
ADC	A,S	Add with carry operand s to Acc.
ADD	A,n	Add value n to Acc.
ADD	A,r	Add Reg. r to Acc.
ADD	A,(HL)	Add location (HL) to Acc.
ADD	A,(IX+d)	Add location (IX+d) to Acc.
ADD	A,(IY+d)	Add location (IY+d) to Acc.
ADD	HL,ss	Add Reg. pair ss to HL
ADD	IX,pp	Add Reg. pair pp to IX
ADD	IY,rr	Add Reg. pair rr to IY
AND	s	Logical 'AND' of operand s and Acc.
BIT	b,(HL)	Test BIT b of location (HL)
BIT	b,(IX+d)	Test BIT b of location (IX+d)
BIT	b,(IY+d)	Test BIT b of location (IY+d)
BIT	b,r	Test BIT b of Reg. r
CALL	cc,nn	Call subroutine at location nn if condition cc is true
CALL	nn	Unconditional call subroutine at location nn
CCF		Complement carry flag
CP	s	Compare operand. with Acc.
CPD		Compare location (HL) and Acc. decrement HL and BC
CPDR		Compare location (HL) and Acc. decrement HL and BC repeat until BC=0
CPI		Compare location (HL) and Acc. increment HL and decrement BC
CPIR		Compare location (HL) and Acc. increment HL, and decrement BC repeat until BC=0
CPL		Complement Acc. (1's complement)
DAA		Decimal adjust Acc.
DEC	m	Decrement operand m
DEC	IX	Decrement IX
DEC	IY	Decrement IY
DEC	ss	Decrement Reg. pair ss
DI		Disable interrupts
DJNZ	e	Decrement B and Jump relative if B=0
EI		Enable interrupts
EX	(SP),HL	Exchange the location (SP) and HL
EX	(SP),IX	Exchange the location (SP) and IX
EX	(SP),IY	Exchange the location (SP) and IY
EX	AF,AF'	Exchange the contents of AF and AF'
EX	DE,HL	Exchange the contents of DE and HL
EXX		Exchange the contents of BC,DE,HL with the
HALT		HALT (wait for interrupt or reset)
IM	0	set interrupt mode 0
IM	1	Set interrupt mode 1
IM	2	Set interrupt mode 2
IN	A,(n)	Load the Acc. with input from device n
IN	r,(C)	Load the Reg. r with input from device (C)
INC	(HL)	Increment location (HL)
INC	IX	Increment IX

```
INC  (IX+d)     Increment location (IX+d)
INC  IY         Increment IY
INC  (lY+d)     Increment location (lY+d)
INC  r          Increment Reg. r
INC  ss         Increment Reg. pair ss
IND             Load location (HL) with input from port (C),
                decrement HL and B
INDR            Load location (HL) with input from port (C),
                decrement HL and decrement B, repeat until B=0
INI             Load location (HL) with input from port (C); and
                increment HL and decrement B.
INIR            Load location (HL) with input from port (C),
                increment HL and decrement B, repeat until B=0
JP   (HL)       Unconditional Jump to (HL)
JP   (IX)       Unconditional Jump to (IX)
JP   IY         Unconditional Jump to (IY)
JP   cc.nn      Jump to location nn if condition cc is true
JP   nn         Unconditional jump to location nn
JR   C,e        Jump relative to PC+e if carry=l
JR   e          Unconditional Jump relative to PC+e
JR   NC,e       Jump relitive to PC+e if carry=0
JR   NZ,e       Jump relitive to PC+e if zero (z=1)
JR   z,e        Jump relative to PC+e if zero (Z=l)
LD   A,(BC)     Load Acc. with location (DC)
LD   A,(DE)     Load Acc. with location (DE)
LD   A,I        Load Acc. with I
LD   A,(nn)     Load Acc. with location nn
LD   A,R        Load Acc. with Reg. R
LD   (BC),A     Load location (BC) with Acc.
LD   (DE),A     Load location (DE) with Acc.
LD   (HL),n     Load location (HL) with value n
LD   dd,nn      Load Reg. pair dd with value nn
LD   dd,(nn)    Load Reg. pair dd with location (nn)
LD   HL,(nnfrom port (Celative to PC+e if non zero (Z=0)
LD   A,(BC)     Load Acc. with location (BC)
LD   A,(DE)     Load Acc. with location (DE)
LD   A,I        Load Acc. with I
LD   A,(nn)     Load Ace. with location nn
LD   A.R        Load A
LD   IY,(nn)    Load IY with location (nn)
LD   (IY+d),n   Load location (IY+d) with value n
LD   (IY+d),r   Load location (IY+d) with Reg. r
LD   (nn),A     Load location (nn) with Acc.
LD   (nn),dd    Load location (nn) with Reg. pair dd
LD   (nn),HL    Load location (nn) with HL
LD   (nn),IX    Load location (nn) with IX
LD   (nn),IY    Load lOcation (nn) with IY
LD   R,A        Load R with Acc.
LD   r,(HL)     Load Reg. r with location (HL)
LD   r,(IX+d)   Load Reg. r with location (IX+d)
LD   r,(IY+d)   Load Reg. r with location (IY+d)
LD   r,n        Load Reg. r with value n
LD   r,r'       Load Reg. r with Reg. r'
LD   SP,HL      Load SP with HL
LD   SP,IX      Load SP with IX
```

LD	SP,IY	Load SP with IY
LDD		Load location (DE) with location (HL), decrement DE,HL and BC
LDDR		Load location (DE) with location (HL), decrement DE,HL and BC; repeat until BC=0
LDI		Load location (DE) with location (HL), increment DE,HL, decrement BC
LDIR		Load location (DE) with location (HL), increment DE,HL, decrement BC and repeat until BC=0
NEG		Negate Acc. (2's complement)
NOP		No operation
OR	s	Logical 'OR' of operand s and Acc.
OTDR		Load output port (C) with location (HL), decrement HL and B, repeat until B=0
OTIR		Load output port (C) with location (HL), increment HL and decrement B, repeat until B=0
OUT	(C),r	Load output port (C) with Reg. r
OUT	(n),A	Load output port (n) with Acc.
OUTD		Load output port (C) with location HL, decrement HL and B
OUTI		Load output port (C) with location HL, increment HL and decrement B
POP	IX	Load IX with top of stack
POP	IY	Load IY with top of stack
POP	qq	Load Reg. pair qq with top of stack
PUSH	IX	Load IX onto stack
PUSH	IY	Load IY onto stack
PUSH	qq	Load Reg. pair qq onto stack
RES	b,m	Reset Bit b of operand m
RET		Return from subroutine
RET	cc	Return from subroutine if condition cc is true
RETI		Return from interrupt
RETN		Return from non maskable interrupt
RL	m	Rotate left through carry operand m
RLA		Rotate left Acc. through carry
RLC	(HL)	Rotate location (HL) left circular
RLC	(IX+d)	Rotate location (IX+d) left circular
RLC	(IY+d)	Rotate location (IY+d) left circular
RLC	r	Rotate Reg. r left circular
RLCA		Rotate left circular Acc.
RLD		Rotate digit left and right between Acc. and location (HL)
RR	m	Rotate right through carry operand m
RRA		Rotate right Acc. through carry
RRC	m	Rotate operand m right circular
RRCA		Rotate right circular Acc.
RRD		Rotate digit right and left between Acc. and location (HL)
RST	p	Restart to location p
SBC	A,s	Subtract operand s from Acc. with carry
SBC	HL,ss	Subtract Reg. pair ss from HL with carry
SCF		Set carry flag (C=l)
SET	b,(HL)	set Bit b of location (HL)
SET	b,(IX+d)	Set Bit b of location (IX+d)
SET	b,(IY+d)	Set Bit b of location (IY+d)
SET	b,r	Set Bit b of Reg. r
SLA	m	Shift operand m left arithmetic
SRA	m	Shift operand m right arithmetic
SRL	m	Shift operand m right logical
SUB	s	Subtract operand s from Acc.
XOR	s	Exclusive 'OR' operand s and Acc.