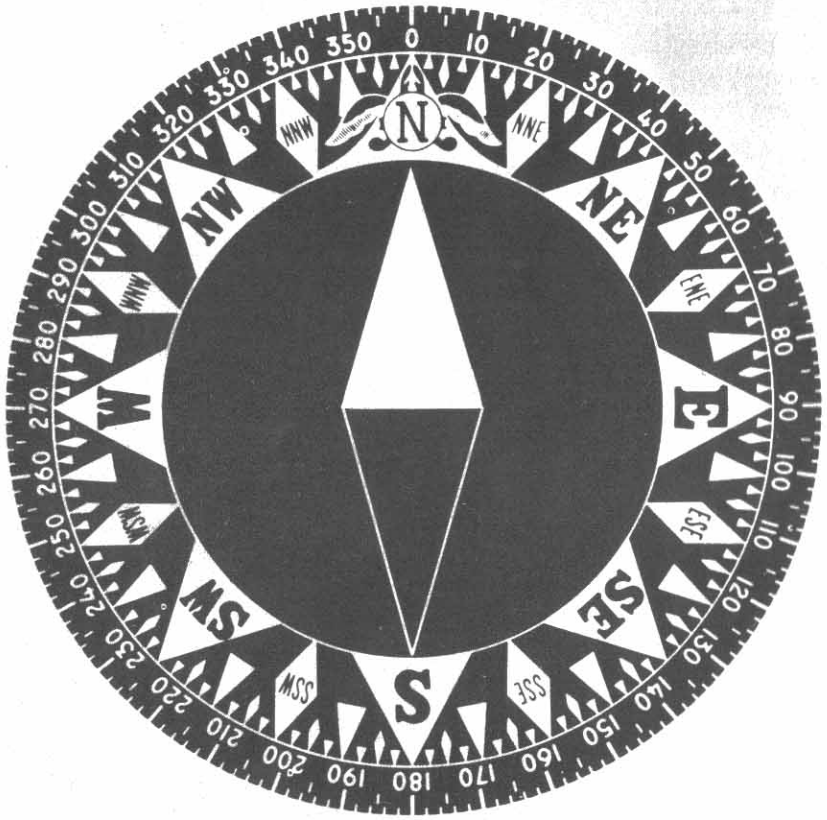
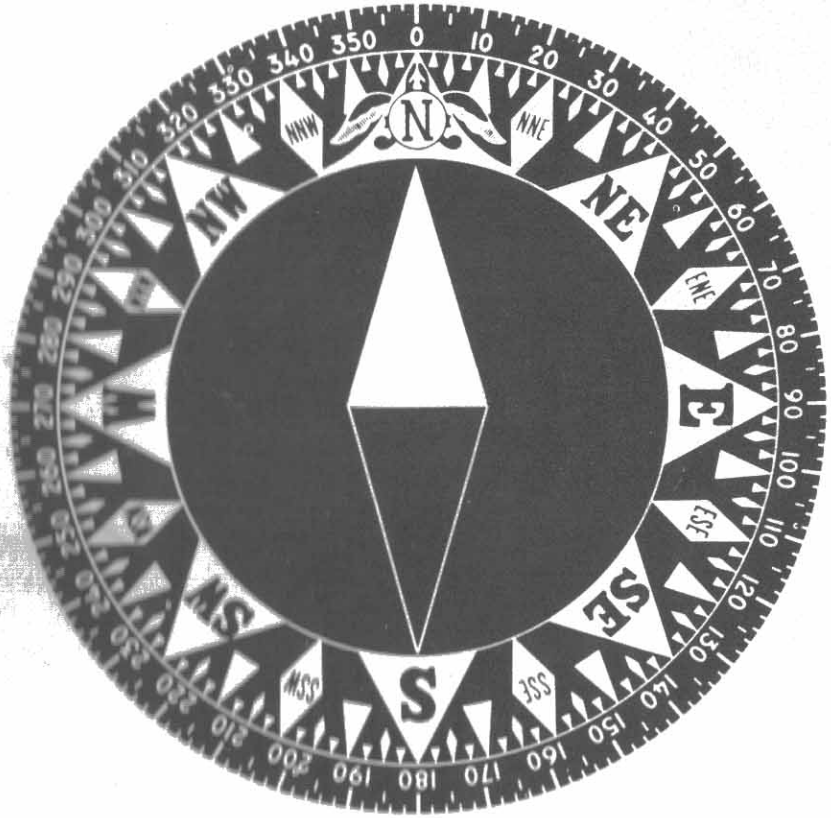
COMPASS

Points the RIGHT way to machine code



COMPASS

Points the RIGHT way to machine code



COMPASS

by Level 9 Computing

Contents | Page

Introduction

Compass (Compression Assembler) was produced by, and is copyright (C) of Level 9 Computing, 229 Hughenden Road, High Wycombe, Bucks.

Compass is a full 8K assembler for the Z80 assembly language. It uses compression techniques to reduce the amount of valuable RAM needed for source and symbol space - to about half that needed by other assemblers. This allows bigger programs to fit in memory and speeds up loading and saving to cassette.

Two versions of Compass are supplied on the cassette. The first uses 8K bytes of user RAM and assembles at about 3000 lines per minute. The second version uses 256 bytes of user RAM (the other 8K fits in the unused 8K of alt. green screen memory) and assembles at about 500 lines per minute.

This manual describes how to use Compass and gives technical details to help you get the best from it. If you have never used Z80 assembler before you are advised to obtain one of the books suggested near the end of this manual.

Installation

Each side of the cassette has two files recorded on it. The two sides are the same but recorded in a slightly different format to suit different cassette recorders. A good cassette recorder will load both sides of the cassette, however the majority may only reliably load on one side of the tape. Once you have identified the side that your system will load you might like to make a backup copy of compass on one of your own tapes, this make it easier to load the next time you want to use Compass.

The two files are both copies of Compass; they are recorded closely following each other on the tape. The first copy occupies 8K bytes of memory from &8ØØØ to &9F2Ø. The second copy occupies 231 bytes from &9FØØ to &9FE7 plus the 8K bytes of alt. green screen memory (see section on MEMORY MAP for details.)

Loading

The first copy is loaded by the monitor 'R' command. From Basic enter:

```
MON
ØD44 3B64 63ØE 24BE
4D2A Ø18C FFFF 2ØEF 63ØD FFFF 9FFF 24BD
.Z...P..
*R "COMPASS"
```

Start the tape on PLAYBACK. When loading is finished the cursor reappears. You could now either save a backup copy of Compass:

```
*D 8ØØØ 9F2Ø Ø "COMPASS"
```

or start Compass immediately:

```
*G 9FØØ
```

The second copy is similar, although it contains a relocator to move Compass into the alt. green memory. From Basic enter:

```
MON
ØD44 3B64 63ØE 24BE
4D2A Ø18C FFFF 2ØEF 63ØD FFFF 9FFF 24BD
.Z...P..
*R "COMPASS"
```

Once loaded the relocator must be run. However if you want to save a backup copy this must be done before doing the relocation. This is because the alt. green memory bank cannot be loaded or saved to tape.

To save a backup of the second Compass file enter:

*D 7EØØ 9FE7 Ø "COMPASS"

To relocate compass enter:

*G 7EØØ

Relocation takes about five seconds, when complete the cursor reappears.
Now continue the same as for the first copy; to start enter:

*G 9FØØ

If you Quit Compass it can be restarted by the command:

*G 9FØ3

This works for either version.

With compass installed you can now enter a new source with the Input command or return to the monitor with Quit to load a previously saved source.

Before using Compass it helps to have a working knowledge of using the Lynx monitor. This is described in the Lynx User Manual.

Overview

In Compass a Z80 assembler program is a number of source lines, but unlike Basic these lines are not given line numbers. However the List and Assemble commands put a line number at the beginning of each source line displayed and these can be used to find individual lines using the Position command. In effect you can pretend that the numbers are there, but that the source is being continually re-numbered.

Anything may be put in a source line (except control-codes such as ESC or RETURN), but Compass will not Assemble any line that it does not understand. A valid source line has the general format:

symbol opcode operands ;comment

All these four fields are optional, but an opcode will be present if operands are also there, and if the opcode is EQU then symbol must be present.

symbol, if used must occur at the start of the line. On entering the line no spaces must be put after the colon (":") prompt. When the line is listed one space is put between the line number and the symbol.

opcode must have at least one space entered before it, so that when listed two spaces occur between the line number and the opcode. More than one space will still be valid, but obviously this will use up more memory.

operands are determined by the opcode for this line.

;comment may occur on any source line. The first semi-colon (";") on a line that is not part of a string or character constant causes itself and all following characters up to the end of the line to be ignored by the Assemble command. This is the equivalent to the Basic REM statement.

The total length (excluding line number) of any source line may not exceed 80 characters. It will be truncated (without any error message) if it's too long. If you use Change to make a line longer than 80 characters this produces the error "8, Too long" and the Change terminates

Command Summary

Compass provides the following commands, usable when it has been started. The commands are summarised below and described later on the pages referenced.

Page	Command	Parameters	Description
11	A	options	Assemble source
8	B		Move to Beginning of source
8	C	string1 string2	Find and optionally change string1 to string2 throughout source
8	D	lnum lnum	D elete source lines
9	F	string	Find line(s) containing string
17	G		Execute assembled object code
9	I	lnum	Insert lines into source after line lnum
10	L	lnum	List source lines.
13	N	low-addr high-addr	News (deletes) program source
12	O	low-addr high-addr	Set Object code limits
10	P	lnum	Position a source line for editing
17	Q		Quits Compass, returning to the monitor
13	R	address	Relocate object code
18	S		Clear screen
12	T	low-addr high-addr	Set symbol Table limits
14	W		Write source to tape
17	?		Display source, object and table limits
10	(Control)Q		Recall last command or edit source line

The lnum parameter (line number) defaults to the current line (normally the last line printed/listed/edited etc.) to save typing.

Strings in commands can be delimited by any sensible character (i.e. not space, return or ESC). The start and end delimiter of each string must be the same.

The Assemble, Change, Find and List commands repeat over a range of lines. ESC ends the command and holding down a shift key pauses it. Change prints each source line it finds then waits for ESC, return or space to be pressed. Input only ends when an empty line (i.e. nothing after the ":" prompt) is entered.

Note that line numbers are in hex, and that the program is automatically renumbered following any change so you will normally use Find to locate a section of source which is of interest rather than using these numbers.

Editing Commands

The commands for the entry and modification of source programs are Begin, Change, Delete, Find, Input, List, Position and (Control)Q.

BEGIN

B

The first line of the source becomes the current line. If the source is empty then error "4, Not found" occurs. Begin is normally used before Change or Find to ensure that all source lines are found or changed.

CHANGE

C string1 string2

Example: C ":" ";"

Starting from the current line find the next line containing string1 (excluding delimiters) and display it on the screen. Pressing return replaces the first occurrance of string1 in that line with string2 and displays the replaced line. Pressing space leaves the line unaltered and continues to find more lines containing string1. Pressing ESC at any time stops the change.

The current line is set to the line at which ESC was pressed to stop the Change or to the end of the source if Change didn't find any (more) lines.

DELETE

D lnum lnum

Example: D 1A 21

Deletes the current line if just D is entered; A specified line if one lnum is given; or an inclusive range of lines if start and finish line numbers are given.

If just D was entered the text part of the deleted line is listed, so you can re-input the line if you typed D by accident.

The current line is set to the line following the last line deleted.

FIND

F string

Example: F '(HL)'

Starting with the current line find the next line containing string (excluding delimiters) and print it. If one of the shift keys is held down find will pause after each line is displayed until ESC is pressed or the shift is released.

ESC will stop find at any time.

The current line is set to the line at which ESC was pressed to stop the command or to the end of the source if Find didn't find any (more) lines.

INPUT

I lnum

Example: I Ø

Insert lines after the specified line, or the current line if I is entered by itself. If lnum is zero then lines are inserted before the first source line and if lnum is a very large number (e.g. 8ØØØ) lines are inserted after the end of the source.

Input gives a prompt of ":" following which you can type in the new source line. Pressing return inserts the line into memory and gives another prompt. To stop Input press return after the prompt without entering a line (i.e. enter a blank line).

The current line is set to the last line inserted.

LIST

L lnum

Example: L 1A

Lists source lines starting with the specified line or the current line if just L is entered.

Pressing ESC stops List and pressing one of the shift keys pauses list until ESC is pressed or the shift key is released.

The current line is set to the line at which ESC was pressed to stop the command or to the end of the source if the last source line was reached.

POSITION

P lnum

Example: P

If P is entered on its own the current line is displayed.

If a line number parameter is given P sets the current line to that line and lists it. If the requested line does not exists or the current line is the end of the source then error "4, Not found" is displayed.

Use P to set the current line to a specified line before using Find, Change or Delete; or to put a source line into the edit buffer before using (Control)Q.

(Control)Q

To use (Control)Q Press 'Q' while the control key is held down. This must be typed as the first character on a line otherwise (Control)Q has no effect. This is similar to Lynx Basic. (Control)Q re-displays either the last command typed (if the command was illegal and gave an error message) or allows a source line to be edited following a Position command.

Warning: (Control)Q only works if entered as the first character of each line. Pressing ESC, delete or any other such key disables (Control)Q for the rest of the line. Do NOT attempt to use (Control)E - this only works for lynx Basic and may crash the Lynx if used from the monitor or from within Compass.

Assembly Commands

The five commands Assemble, New, Object, Relocate and Table are all used to control the assembly of source into Z80 machine code.

ASSEMBLE

A options

Example: A LS

Assembles (translates) the source into machine code using options to control the assembly. If no options are given then a normal assembly results.

If any error is detected during the assembly the line in error is displayed followed by an error message. The assembler is two-pass in operation so some errors (for example Duplicate symbol) are given before others (for example Undefined symbol). Other errors (for example Bad opcode) are given twice, once on each pass.

The symbol table required for an assembly is created in the area allocated by the Table command and object code is placed under control of ORG psuedo-ops within the source and by the Object and relocate commands.

The options which can follow the 'A' on the same line are:

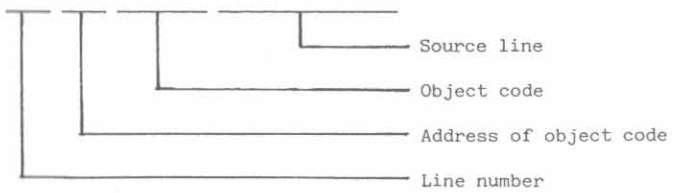
- L produce an assembly listing
- N do not store the generated object code
- S produce a symbol table

After an assembly the current line is always the beginning of the source.

Pressing ESC at any time during an assembly stops the command.

If the 'L' option is specified an assembly listing is produced in the following format:

```
001A 8800 ED7B     LL01 LDIR ; Copy
```



OBJECT

O low-addr high-addr

Example: O 88ØØ 8FFF

Sets the address range to which object code can be Assembled. If the assembly would generate code outside these limits the assembly is aborted with the error "18, Code range"

The address at which the generated object code is stored will be the low-addr given in the Object command unless an ORG pseudo-op is used of Relocation is in effect.

TABLE

T low-addr high-addr

Example: T 78ØØ 79ØØ

Sets the address range used for storing the symbol table generated by an Assembly command. Each symbol defined in the source will require four bytes of Table space. If the table if too small then the assembly is stopped with the error "24, Too many symbols".

Each group of four bytes in the table has the format:

The first two bytes form the 16-bit address (low byte first) of the start of the source line which defined the symbol. The third and fourth bytes store the 16-bit (low byte first) value of the symbol. If the symbol represents an 8-bit value or result then the fourth byte will be zero, or &FF if the 8-bit value is negative.

The highest address actually used during an Assembly can be displayed by the ? command.

NEW

N low-addr high-addr

Example: N 7ØØØ 9EFF

Similar to the NEW command of Lynx Basic but also sets the address range used for program source. The main use for this is to increase the amount of memory available for source programs.

To delete a source from memory it is often easier to use "D Ø FFFF"

If you change the low-addr of the source area then any files you subsequently load from tape will have to be moved by the monitor 'I' command to its new address.

RELOCATE

R address

Example: R 64Ø7

Sets the address at which object code is stored during an assembly. "R Ø" is the default and turns off the relocation so that object code is stored at the address to which it is assembled (within the limits set by the Object command).

e.g. R 64Ø7
O 9FØØ 9FFF

then assembling the source:

```
ORG &9F1Ø
DEFB &76
```

will store the value &76 at address &6417

Loading and Saving

Compass provides no direct commands for saving or loading source or object files to tape. However saving a source program is assisted by the Compass Write command, and loading any file is easy using the monitor.

SAVE

W

Example: W(return)(Control)Q(return)

Write quits Compass and returns to the monitor. It also fills the edit buffer with the monitor command that dumps the source to cassette, so that once in the monitor all you do is press (Control)Q followed by return. (Control)Q recalls a command of the form:

```
*D 64Ø7 6438 Ø ""
```

After pressing (Control)Q but before (return) you can edit this command (using the cursor keys, as usual) to put in your own file-name, if required.

After saving return to Compass using the command:

```
*G 9FØ3
```

LOAD

To load a previously saved source you use the monitor 'R' command. Return to the monitor using Quit, read the source then return to Compass:

```
Q
*R ""
*G 9FØ3
```

APPEND

Since Compass relies on the monitor commands for saving and loading tape files this makes it difficult to append source files - the monitor will only load a saved file back to the address originally used for the save.

To append two files requires the use of several monitor commands and just a touch of hex arithmetic.

Suggested Method

If the two files are already saved on tape called "A" and "B" then this sequence will append "B" to the end of "A":

1) Load "A" into memory and use the Compass ? command to find its 'low' and 'top' addresses.

   Example:
```
*R "A"
 "A"
*G 9FØ3

?
Source 64Ø7 77FF Top= 642C
Object 78ØØ 7DFF Top= 78Ø9
Reloc  ØØØØ      Ent= ØØØØ
Table  7EØØ 7FFF Top= 7EØC
```

   This gives 'low' as &6407 and 'top' as &642C.

2) Load "B" and also use ? but only note its size ('top'-'size')

   Example:
```
Q
*R "B"
 "B"
*G 9FØ3

?
Source 64Ø7 77FF Top= 642F
Object 67ØØ 7DFF Top= 78Ø9
Reloc  ØØØØ      Ent= ØØØØ
Table  7EØØ 7FFF Top= 7EØC
Q

*A 642F 64Ø7
C836 ØØ28 D6
```

   This gives the length as &642F - &64Ø7 = &ØØ28

3) File "B" is now copied up in memory to make room for "A" which will be loaded below it. Quit compass and use the monitor 'I' command:

   I low-address-of-B top-address-of-A size-of-B

   Example: *I 64Ø7 642C ØØ28

4) Stay in the monitor and load file "A". This will exactly fit the gap below "B" created by step (3) above.

```
Example: *R "A"
          "A"
         *
```

5) Using the monitor 'H' and 'M' commands examine the area around the end of file "A" - where it meets the start of "B". The bytes of interest are the last three of "A" and the first byte of "A", so using 'H':

```
      H  top-of-A-3

Example: *A 642C 3
         642F 6429 ??
         *H 6429
         6429  00 C8 0A 03 50 52 53 E7
```

These four bytes should be modified to be spaces (hex 2Ø)

```
Example: *M 6429
         6429 (ØØ) 2Ø
         642A (C8) 2Ø
         642B (ØA) 2Ø
         642C (Ø3) 2Ø
         642D (5Ø) .
```

6) Now warm-start compass. This will give a "Bad source" error since you have modified bytes within the source area. Using List and Find locate the first line of file "B" and remove the extra four spaces which have been added.

```
Example: *G 9FØ3

         7, Bad source
         P 6
         ØØØ6     PRS POP HL
         C "    " ""
         ØØØ6 PRS POP HLreturn
```

7) The append is complete. With pratice this process will become almost automatic. You can now continue to use Compass; it might be a good idead to save the appended files, lest some accident should occur.

General Purpose Commands

The other remaining commands include Go, Quit and ?.

GO

```
      G
```

Go executes the object code last assembled from the point in the code defined by the ENT pseudo-op. If the last assembly did not define an ENT address then Go has no effect (no error is displayed, Compass continues awaiting another command).

Warning: Do not use Go when Relocation is in effect.

QUIT

```
      Q
```

Quit exits Compass and returns to the monitor. The edit buffer is filled with the command "G 9FØ3" so if you Quit by accident then (Control)Q(return) will return you to Compass.

?

```
      ?
```

Displays the 'low', 'high' and current 'top' addresses for the source, object and table areas, plus the current values for Relocation and ENT.

The display is as follows:

```
      Source aaaa bbbb Top= cccc
      Object mmmm nnnn Top= oooo
      Reloc  pppp      Ent= qqqq
      Table  ssss tttt Top= uuuu
```

'aaaa' and 'bbbb' are the low and high addresses of the current source area as defined by the New command. 'cccc' is the current maximum address used by the source.

'mmmm' and 'nnnn' are the low and high addresses of the current object area as defined by the Object command. 'oooo' is the address of the last byte of generated object code produced by the last Assembly. If the last Assembly did not generate any code then this value is ØØØØ.

'pppp' is the current value of the Relocate command. If relocation is not in effect then this value is ØØØØ.

'qqqq' is the address last assigned to an ENT pseudo-op. If the last assembly did not define an ENT then this value is ØØØØ and if two ENTs were defined the value given is the address of the last one in the source.

'ssss' and 'tttt' are the low and high addresses of the current symbol table area as defined by the Table command. 'uuuu' is the maximum+1 address used for symbol space during the last Assembly. If no assembly has taken place since Compass was cold-started then this value is ØØØØ.

Memory Map

The powerful method of 'bank-switching' used by the Lynx is beyond the scope of this manual. More comprehensive information is available in Camputers newsletter: "LYNX USER" (June 83 issue) or in the Lynx Techincal Manual.

The 48K LYNX uses 24K RAM for the screen display, 8K is unused and hidden away in Bank-3 (this is the alt. green memory) and 16K is directly available to the user (although 2¼K is already used by the monitor and Basic.)

The 96K LYNX is similar with an additional 24K directly accessible and the other 24K located at addresses normally occupied by the Operating System ROM. This is summarised below:

<table>
<tr><th>Bank</th><th>ØØØØ
1FFF</th><th>2ØØØ
3FFF</th><th>4ØØØ
5FFF</th><th>6ØØØ
7FFF</th><th>8ØØØ
9FFF</th><th>AØØØ
BFFF</th><th>CØØØ
DFFF</th><th>EØØØ
FFFF</th></tr>
<tr><td>1</td><td rowspan="3">Operating System</td><td rowspan="3"></td><td colspan="2">User RAM</td><td rowspan="3"></td><td></td><td rowspan="3">Plug in ROM</td></tr>
<tr><td>2</td><td rowspan="2">Red</td><td rowspan="2">Blue</td></tr>
<tr><td>3</td><td rowspan="2">alt. green</td><td rowspan="2">Green</td></tr>
</table>

48K LYNX

<table>
<tr><th>Bank</th><th>ØØØØ
1FFF</th><th>2ØØØ
3FFF</th><th>4ØØØ
5FFF</th><th>6ØØØ
7FFF</th><th>8ØØØ
9FFF</th><th>AØØØ
BFFF</th><th>CØØØ
DFFF</th><th>EØØØ
FFFF</th></tr>
<tr><td>1</td><td rowspan="3" colspan="3">Operating System</td><td colspan="5">User RAM</td></tr>
<tr><td>2</td><td rowspan="2"></td><td>Red</td><td rowspan="2"></td><td>Blue</td><td rowspan="2"></td></tr>
<tr><td>3</td><td>Alt. green</td><td>Green</td></tr>
</table>

96K LYNX

The Lynx monitor uses memory between &6ØØØ and &64Ø6 as its stack and workspace. Basic uses the monitor and also uses memory between &9FAC and &9FFØ as its stack.

Compass occupies &8000 to &9F2Ø or &9FØØ to &9FE8 if you use the second copy on the tape. The second copy, whilst making an extra 7¾K available clashes with Basic - so you can't use these two at the same time without taking extra care - see below.

Compass initialises the source, object and table areas on a cold-start using values read from a table stored at addresses &8ØØØ-&8ØØB. If you have saved your own copy of Compass (see INSTALLATION) you could personalise this by altering these values to reflect your own usage.

&8ØØ1,&8ØØØ	Low address for source
&8ØØ3,&8ØØ2	High address for source
&8ØØ5,&8ØØ4	Low address for object code
&8ØØ7,&8ØØ6	High address for object code
&8ØØ9,&8ØØ8	Low address for symbol table
&8ØØB,&8ØØA	High address for symbol table

Using Basic with Compass

Basic requires the vector at address &9FF5 and &9FF4 in order to initialise by the monitor 'J' command. Neither version of Compass uses this address so you can always use the 'J' command to return to Basic after using Compass. Then do 'NEW' to clear any garbage put by Compass into the Basic program area.

If you use the second copy of Compass then running Basic will overwrite part of Compass. This will crash if you subsequently try to re-start Compass. The solution is to save the overlapping memory area (&9FØØ-&9FE8) after Compass has been installed, then only re-load this area when moving from Basic to Compass.

Error Messages

All error messages start with a number to identify the message uniquely. Additionally, where an error occurs during Assembly the source line containing the error is listed. The error messages are listed below in numerical order, together with an explanation of each message.

1, Bad arguments

A Compass command was followed by too few, too many, or the wrong type of argument(s).

2, Bad command

The first non-space character of the command was not a valid Compass command.

3, Escape

The ESC key was pressed during an Assemble, Change, Find or List command. The current line is set as follows:
If you were assembling it is set to the first line.
During List it is the last line displayed.
During Change or Find it will be the last line processed.
If in doubt Position will display the current line.

4, Not found

Occurs in response to a Position command. If just P is entered This indicates that the previous Change, Find or List command reached the end of the source (i.e. was not stopped by ESC). If P was followed by a line number then no line with this number exists. The current line is set to the end of the source.

5, No memory

A Change or Input command caused the source to grow and there was not enough memory for this. This can also occur when using (Control)Q to edit a line. During Input the last line entered will have been lost and Compass will terminate the Input command. During Change or edit the affected source line will have been lost. In future use the ? command regularly to keep track of program size.

6, Corrupted

The code of Compass itself has been overwritten. The message only occurs once as Compass then remembers the new checksum value. If the error occurs again then this is because Compass Has been overwritten again. The safest way to recover is to switch off and start again, reloading Compass and the latest version of the source from tape. Running a corrupted version of Compass could cause anything to happen.

7, Bad source

The source file has changed since Compass last examined it. The message only occurs once as Compass then changes the checksum back to its correct value. Continuing to use Compass may cause it to crash, but if you wish you could use commands such as ? or List to pin down the problem.

8, Too long

A Change command would have resulted in a line that was too long (over 80 characters) so the line has not been changed. The command is terminated.

9, Duplicate symbol

Occurs during an Assembly. The symbol defined in the line listed above has already been declared earlier in the source: Labels cannot be redefined. Use Find to locate the earlier declartion. Note that Change can be readily used to change some occurrances of a symbol, leaving others.

10, Bad opcode

Occurs during an Assembly. The opcode or pseudo-op in the line listed above is not valid. There must be at least two spaces in the source line between the line number and the opcode, otherwise Compass will take the opcode as being a symbol instead. This error will be given twice for each source line with this error during one Assembly command.

11, Bad operands

Occurs during an Assembly. The operands in the line listed above are not valid. This error will be given twice for each source line with this error during one Assembly command.

12, Brackets

Occurs during an Assembly command. A closing bracket is missing, or there is an error ahead of the bracket. This error will be given twice for each source line with this error during one Assembly command.

13, Overflow

Occurs during an Assembly command. During the evaluation of an expression in the line listed a result was calculated which was too big to be held in 16 bits (i.e. over 65535 or under -32768). This error only occurs once for each source line with this error except for ORG, DEFS, IF and EQU psuedo-ops.

14, Bad char

Occurs during an Assembly command. A character constant has no closing quote. The same type of quote must start and end a character constant. This error will be given twice for each source line with this error during one Assembly command.

15, Bad constant

A '&' was not followed by a hex digit or a hex/decimal number became too big to be held in 16 bits (over 65536). This error will be given twice for each source line with this error during one Assembly command.

16, Undefined symbol

A label used as an operand in the line listed above is not defined anywhere (i.e. does not appear in the first column of any line). Symbols used as operands in ORG, DEFS and IF psuedo-ops must be defined before they are used: forward references are not allowed in this case for efficiency reasons.

17, Bad instruction

The operands used in the line listed above do not match the opcode. This error will be given twice for each source line with this error during one Assembly.

18, Code range

Occurs during an Assembly command. A byte of object code would have been placed outside the limits previously set by the Object command. Use ? to display these limits. This error will never be generated for an ORG or DEFS. The assembly is aborted.

19, Missing symbol

Occurs during an Assembly command. No label occurs at the start of the EQU psuedo-op in the line listed above.

21, Too many operands

Occurs during an Assembly command. Characters have been found after the end of the statement listed above which were not part of a comment. Also occurs if DEFB is given more than four operands. Perhaps the semi-colon (";") which should start the comment has been omitted? This error will be given twice for each source line with this error during one Assembly command.

22, Value

Occurs during an Assembly command. An expression in the line listed above that should produce an 8-bit result is too large (over 255) or if negative is under -128.

23, Too far

Occurs during an Assembly for JR and DJNZ opcodes. The relative jump in the line listed above would jump more than -126 to +129 from the current address.

24, Too many symbols

Occurs during an Assembly command. Too little symbol space was allowed for the source being assembled. The remainder of the assembly is aborted. Four bytes are required for every symbol defined. Use the ? command to find out much symbol table space is currently reserved and Table to allocate more. After a successful Assembly ? will tell you how much space was actually used.

25, Press Control-Q RETURN to re-start
Command is: G 9FØ3

Occurs in response to the Quit command. You are returned to the monitor. You may then use 'J' to return to Basic or use any other monitor command.
If you did Quit by accident then (Control)Q(return) will return you to Compass. Alternatively "G 9FØ3" has the same effect.

26, Press Control-Q RETURN to save.
Command is: D xxxx yyyy 0 ""

Occurs in response to the Write command. You are returned to the monitor. If you then type (Control)Q this recalls the command 'D xxxx yyyy 0 ""' where xxxx and yyyy are automatically the current 'low' and 'top' addresses for the source. You may edit this command (e.g. to put in a file-name) then press RETURN to save the source. Once saved you can return to Compass by the monitor command "G 9FØ3".

Syntax

The syntax of Z80 assembler is reasonably standard. This section lists some of the differences between Compass and the versions described by the books mentioned at the end of this manual

Expressions

An expression is a sequence of one or more elements separated by '+' or '-'. where an element may be a constant, a symbol, a character constant or '$' representing the value of the current assembly address counter. Expressions are evaluated left to right and may not contain brackets.

Constants

All numbers are assumed to be in decimal unless preceded by '&' or followed by 'H'.

```
Examples:  &A    = 1Ø decimal or ØA hex
           ØAH   = 1Ø decimal or ØA hex
           11    = 11 decimal or ØB hex
```

The first character of a constant must either be '&' or a digit 'Ø' through '9'.

Character Constants

A character constant may be used anywhere in place of a decimal or hex constant. It's value will be the ascii code of the character enclosed between the two quotes

```
Examples:  'A'   = 65 decimal or 41 hex
           "a"   = 97 decimal or 61 hex.
```

String Constants

A string can only occur as an operand to the DEFM psuedo-op. A string may be delimited by any sensible character (i.e. not ESC, return or semi-colon). The start and end delimiter must be the same character.

```
Examples:  /"name"/
           "the string's contents'
```

Symbols

A symbol (label) can be any length up to the maximum length of a source line. It must start with a letter ('A' through 'Z' or 'a' through 'z') but can contain both letters and digits. Lower case is not the same as upper case.

Psuedo-Ops

In addition to the standard Zilog opcode nmemonics Compass recognises the following psuedo-opcodes:

DEFB DEFM DEFS DEFW ELSE END ENT EQU IF LIST

These are described in detail below.

DEFB

```
DEFB expression  ,expression
```

Example: DEFB &D,Ø

DEFB accepts between one and four operands. each operand is an expression which evaluates to an 8-bit result in the range -128 to +255 inclusive. For each operand one byte of object code is generated with the same value as the result of that expression.

DEFW

```
DEFW expression
```

Example: DEFW start+3

DEFW always take one operand which must be an expression which evaluates to a result in the range -32768 to +65535 inclusive. The 16-bit result is divided into two 8-bit values, and each is used to generate two bytes of object code. The first object code byte is the low 8 bits of the 16-bit result and the second bytes is the top 8-bits.

DEFS

```
DEFS expression
```

Example: DEFS 2

DEFS does not generate any object code. It takes one operand which is an expression. The assembly counter is incremented by the value of the result of that expression, thus leaving a 'hole' in the object code. This is useful for defining data areas (workspace, variables, stack space etc.)
The expression must not forward-reference symbols not yet defined in the assembly.

DEFM

DEFM string

Example: DEFM "Please press RETURN:"

DEFM takes one operand which must be a string of zero or more characters with start and end delimiters. The two delimiters must be the same but can be any sensible character except a semi-colon.
For each character except the delimiters one byte of object code is generated which will have the value of the ascii code for that character. There is no limit on the length of the string, except that it must all be on one 80-character source line.

EQU

symbol EQU expression

Example: INK EQU &625B

EQU defines a symbol to have a constant value. Like comments EQU makes a program easier to understand and debug at the expense of making the program bigger.

e.g. The program:

```
 LD A,1
 LD (&625B),A
```

could be re-written as:

```
GREEN EQU 1
INK EQU &625B
...
 LD A,GREEN
 LD (INK),A
```

This makes it clearer that this is equivalent to Basic's:

INK GREEN.

Conditional Assembly

Three psuedo-ops are used to provide conditional assembly:

IF ELSE END

END

END

This always signals the end of a conditionl section of a program. The line following an END is always assembled.

ELSE

ELSE

If encountered on its own ELSE causes assembly to be suspended until the next END is found or the end of the source is reached. This is not intended to be its normal usage, however.

IF

IF expression

Example: IF debug

IF takes one operand which must be a expression which results in a 16-bit result in the range -32768 to +65535 inclusive. The expression must not forward-reference any symbol not yet defined by the assembly.

If the expression has a result of zero (Ø) then assembly is suspended until the next ELSE or END pseudo-op, or the end of the source is reached. Assembly continues with the statement following the next ELSE or END.

If the expression has a non-zero result then the IF has no effect

Summary

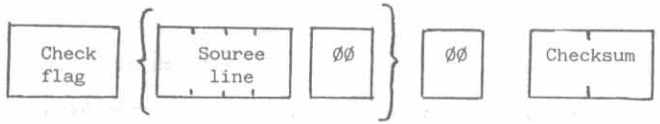
IF and END should occur in pairs. Optionally an ELSE may be placed between the IF and before the END. Source between the ELSE (if present) and the END is only assembled if the expression is zero. Otherwise the source between the IF and ELSE is assembled.

Source Format

This section is intended for the more experienced user. It describes the format used to store the source - in sufficient detail to allow you to examine the source using the monitor 'H' and 'M' commands. Additionally, if you get the error "7, Bad source" you could use the monitor to find and perhaps correct the error(s).

The source is completely position-independant, and may by copied around memory using the monitor 'I' command.

The general format for a source file is:



the brackets { } enclose a sequence which (if no source is present) may not exist or may be repeated once for each source line present.

Each source line may be 1 to eighty bytes terminated by a ØØ byte.

With no source in memory four bytes of source memory are used to store the following:

This sequence is stored by a cold-start (G 9FØØ) or by the New command.

The first byte "Ø1" is the check-flag. A value of ØØ supresses the error "7, Bad source". Any non-zero allows this message to appear.

The second byte "ØØ" is the first byte allocated to the source program. The value "ØØ" is used to mark the end of a source line. An empty line marks the end of the program.

The third and fourth bytes are a checksum, calculated on the values of all the bytes following the 'check flag' and before the checksum.

Example:

The following is a short Compass source program:

```
ØØØ1  ENT
ØØØ2 LOOP DI
ØØØ3  JP LOOP
```

This is stored in memory as:

```
Ø1 B3 ØØ 4C 4F 4F 5Ø CB ØØ D8 4C 4F 45 5Ø ØØ ØØ C9 Ø4
```

Compression

Compass 'compresses' source files in much the same way as Lynx Basic - by replacing sequence of characters within program lines by codes that cannot be generated by the keyboard. As the sequence chosen relate naturally to the structure of valid assembler statements this allows faster assembly (and this is why some of the sequences are of length 1).

The sequences are as follows:

Code	Characters	Code	Characters	Code	Characters	Code	Characters
8Ø	A	9F	DE,	C3	CPDR	E2	OTDR
81	B	AØ	IX,	C4	CPD	E3	OTIR
82	C	A1	IY,	C5	CPL	E4	OUTD
83	D	A2	AF'	C6	CPIR	E5	OUTI
84	E	A3	(C),	C7	CPI	E6	OUT
85	H	A4	(SP),	C8	CP	E7	POP
86	L	A5	(HL),	C9	DAA	E8	PUSH
87	I	A6	(BC),	CA	DEC	E9	RES
88	Z	A7	(DE),	CB	DI	EA	RETI
89	P	A8	A,	CC	DJNZ	EB	RETN
8A	M	A9	I,	CD	EI	EC	RET
8B	AF	AA	(C)	CE	EXX	ED	RLA
8C	HL	AB	R	CF	EX	EE	RLCA
8D	BC			DØ	HALT	EF	RLD
8E	DE	B2	LIST	D1	IM	FØ	RRA
8F	SP	B3	ENT	D2	INC	F1	RRCA
9Ø	IX	B4	IF	D3	INDR	F2	RRD
91	IY	B5	ELSE	D4	IND	F3	RRC
92	NZ	B6	END	D5	INIR	F4	RR
93	NC	B7	ORG	D6	INI	F5	RLC
94	PE	B8	EQU	D7	IN	F6	RL
95	PO	B9	DEFB	D8	JP	F7	RST
96	(HL)	BA	DEFW	D9	JR	F8	SBC
97	(BC)	BB	DEFS	DA	LDDR	F9	SCF
98	(DE)	BC	DEFM	DB	LDD	FA	SET
99	(IX	BD	ADC	DC	LDIR	FB	SLA
9A	(IY	BE	ADD	DD	LDI	FC	SRA
9B	AF,	BF	AND	DE	LD	FD	SRL
9C	SP,	CØ	BIT	DF	NEG	FE	SUB
9D	HL,	C1	CALL	EØ	NOP	FF	XOR
9E	BC,	C2	CCF	E1	OR		

Spaces surrounding opcodes are automatically removed for further compression. The spaces are added back for Change, Find and List.

Compass compresses source lines very efficiently. However symbols and comments are usally the most significant contributors to the size of the compressed source program. To get the best from Compass therefore, keep symbols and comments short.

Suggested Reading

Many books are available about Z80 programming. They are all very large books - which explains why this manual had to be short on details of the Z80 assembler language. Some books you might like to look at are:

Z80 Assembly Language Programming Manual

Published in the USA by Zilog, this is excellent for nitty-gritty technical details, but is definitely not a "teach yourself" book. As few places stock it in this country you would probably need to order it via a good book shop.

Programming the Z80

Written by R. Zaks and published by SYBEX Inc.
This is an excellent introduction to Z8Ø programming. It is probably the best book for a beginner to buy (from your local computer book shop, NOT from Level 9 please!).

Index