Section CONTENTS Page

1 Introduction to Compression Assembler (Compass)

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

Compass is a full Z80 assembler which uses advanced compression techniques to reduce the amount of memory needed to hold your assembler source etc. It reduces the source size of an average program to about half of that needed by other assemblers, allowing bigger programs to fit in memory and speeding up loading/saving to tape. It is also very fast: assembling around 3000 lines per minute (over 10 times the rate of ZEAP, for example).

Compass is 6K and runs under Nas-Sys 1 or 3 on a standard Nascom. It is supplied with a utility to convert source from ZEAP.

This manual describes how to use Compass, what it does, and includes some technical details to help you get the best from it.

2 How to use Compass

Compass is supplied on cassette, packaged within a relocator program. A utility to convert ZEAP source files follows it on the cassette, which is described in Appendix 1.

When you first use Compass from cassette, it is necessary to load the relocator program and execute it to place the assembler at a suitable place in memory. This process is described in Appendix 2. Then you could save the generated Compass on one of your own cassettes so that on subsequent occasions you can load it directly from there.

Starting Compass is described fully in Appendix 3. You execute it at its first address, giving the start and end addresses of the area used for the source as arguments. This is known as a 'cold start' and initialises Compass for use. I.e:

```
E start-address start-of-source-area end-of-source-area
```

You can subsequently use the assembler to enter and modify source etc, but before assembling a program you should use the '0' and 'S' commands to define the areas used for the object code and symbol table.

To restart Compass (e.g after returning to Nas-Sys to test the program that you are developing), execute it at its first address + 3. This is known as a 'warm start' and leaves all parameters unchanged (source/object/symbol table areas etc are the same). I.e:

```
E start-address+3
```

When you use the M or N command to return to Nas-Sys, Compass calculates a check-sum for the program source. On restarting, this is checked and a warning message is output if the program that you are testing has corrupted its source due to a bug. Naturally you can ignore the warning if you like. Once started, Compass can be used to read or write source to cassette (or disk), to edit source and to assemble source into machine code.

3 Commands

3.1 Summary

Compass provides the following commands, usable when it has been started. The commands are summarised below and described later on the pages referenced. Additionally the standard Z80 pseudo-ops (e.g ORG) are provided: see Appendix 4 for details.

Page	Command	Parameters	Description
6	A	options (L, N and S)	Assemble source;
4	B		print Beginning line of program;
4	C	/string1/ /string2/	Change string1 to string2;
4	D	lnum	Delete line or range of lines;
4	E	lnum	List source line in order to Edit it.
5	F	/string/	Find line(s) containing string;
5	H	number-of-lines	set page Height for list command;
5	I	lnum	Insert lines into source (after line lnum);
5	L	lnum	List source lines (starting with lnum);
7	M or N		return to Monitor (Nas-Sys);
6	O	low-addr high-addr	set Object code limits;
5	P	lnum	Print source line;
7	R		Read source (as Nas-Sys R);
6	S	low-addr high-addr	set Symbol table limits;
7	W		Write source (c.f Nas-Sys W);
6	?		Display source, symbol table and object code addresses.

The lnum (line number) parameters default to the current line (normally the last line printed/listed/edited/found etc) to save typing.

The delimiters of strings (shown as '/' above), can be any sensible character.

Several commands, such as Change, can be repeated over a range of lines. ESC ends the command and any other character continues it. If you have Nas-Sys 3, you can use the auto-repeat feature of keys to repeat a command right through the source by leaving a key pressed. ESC also stops Assembly.

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 the program which is of interest rather than using line numbers.

3.2 Editing Commands

The commands provided for the entry and modification of programs are: Beginning, Change, Delete, Edit, Find, Height, Insert, List and Print.

3.2.1 B (BEGINNING)

Example: B

Print the first line and make this the current line (as P 1).

3.2.2 C "old string" "new string" (CHANGE)

Examples:

```
C 'LDI' 'LDIR'
C :string containing '"': :another string containing '"':
```

Find the next line containing the first string and display it on the screen: the search starting at the current line. Then pressing ENTER replaces the first string with the second string, displays the changed line and continues; pressing ESC stops the command; and pressing any other key continues without making any change.

If the command is continued (by any key except ESC), the search resumes at the next line and carries on through the source.

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

3.2.3 D line-number(s)(DELETE)

Examples:

```
D 30
D 1 1FF
```

Delete the current line (if just D is entered); a specified line (if one number is entered); or an inclusive range of lines (two numbers entered).

The current line is set following the last line deleted.

3.2.4 Edit a line

Examples:

```
0A ; This is now the tenth line.
```

Once a line has been listed using Begin, Change, Edit, Find, List or Print it can be altered using standard Nas-sys editing. Using the cursor keys move the cursor to the line to be amended, overtype the characters in error, then press ENTER to put the amended version back into the program.

Be careful about editing lines after using Insert or Delete, as the line you wish to alter may have changed its line-number. If in doubt do a Print or List to get a fresh copy of that line.

The current line is set to the one edited.

3.2.5 F "string" (FIND)

Example:

```
F /EQU/
```

Find the next line containing a string: starting the search at the current line. Then pressing ESC stops the command; or pressing any other key continues the command to find the next line containing the string.

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

3.2.6 H number-of-lines (HEIGHT)

Example:

```
H 6
```

Set the number of lines listed at a time by the 'L' command (this is initialised to 5 when the assembler is cold-started). The current line is unchanged.
If you use a printer to obtain Compass listings, use 'H 0' to turn off paging.

3.2.7 I line-number (INSERT)

Example:

```
I 0
```

Insert lines after the specified line, or the current line if 'I' is entered by itself. 'I0' inserts lines at the start and 'I 8000' (for example) inserts lines at the end.

A prompt of ':' is output and you type the first new line following it. Pressing ENTER inserts the line into the program and presents another prompt of ':' for the next line, and so on. To stop, press ENTER with the cursor on an empty line or one just containing': '.

The current line is set to the last line inserted.

3.2.8 L line-number (LIST)

Example:

```
L 0
```

List a page of lines on the screen (the actual number defaults to 5 and can be changed by the 'H' command), starting with the current line or the one specified.

Then pressing ESC stops the command; or any other key lists the next few lines, and so on.

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

3.2.9 P line-number (PRINT)

Example:

```
P 50
```

Print (list) the specified line on the screen, or print the current line if 'P' is entered by itself.
The current line is set to the line printed (the main function of 'P' is this).

3.3 Assembly commands

The commands provided to allow the assembly of source into machine code are: Assemble, Object, Symbol-Table and '?'.

3.3.1 A options (ASSEMBLE)

Examples:

```
    A
    A LN
```

Assemble the source into machine code, as specified by any options. If no options are entered a normal assembly results.

If any error is detected during assembly, the line containing it is displayed with an error message. Pressing ESC stops, any other key continues.

The symbol table is created in the area defined by the 'S' command, and code is placed as specified by the ORG pseudo-ops in the source provided this is within the area defined by the '0' command.

The options which can follow the 'A' command are:

    L produce assembly Listing;
    N No code generated;
    S produce Symbol table listing.

The current line is set to the first line of the source.

3.3.2 0 start-address end-address (OBJECT-LIMITS)

Example :

```
    0 0C80 0EFF
```

Set the address range to which code can be written by the 'A' command (this is initialised to 0 FFFF when the assembler is cold-started). If assembly would store code outside the range it is stopped with an error.

The current line is unchanged.

3.3.3 S start-address end-address (SYMBOL-TABLE-LIMITS)

Example:

```
    0 8800 8FFF
```

Set the address range used for the symbol table generated by the 'A' command (this is initialised to 1 0 - no symbol table - when the assembler is coldstarted). 4 bytes are needed for each label in the program.

The current line is unchanged.

3.3.4 ? (DISPLAY ADDRESSES)

Example:

```
    ?
```

Display the addresses for the program source, object limits and symbol table, plus the highest address actually used by the program source.

The current line is unchanged.

3.4 Load and Save etc.

The remaining Compass commands allow source to be read and written, and allow a return to Nas-Sys. They are: Monitor, Nas-Sys, Read and Write.

3.4.1 M or N (MONITOR or NAS-SYS)

Examples:
```
        M
        N
```

Return to Nas-Sys from Compass. The program source is check-summed so that on restarting the assembler a check can be made that the source has not been corrupted (e.g by a bug in a program being tested). If the source has been changed then a warning message will be output on restart.

3.4.2 R (READ SOURCE)

Example:
```
        R
```

Read source from cassette in the same way as the Nas-Sys R command does.

You should ensure that the source area is set to the same start address as when the source was saved by 'W' (from Compass or Nas-Sys) - or the start address where is was placed by the conversion utility. The end address should be not less than that used by the source read.

3.4.3 W (WRITE SOURCE)

Example:
```
        W
```

Write the source to cassette in the same way as the Nas-Sys W command, but with Compass supplying the start and end addresses automatically.

The program source and current line are unchanged.

3.4.4 V (VERIFY TAPE)

Example:
```
        V
```

Verify a tape. Does exactly the same as the Nas-sys 'V' command.

The program source and current line are unchanged.

Appendix 1: Contents of the Compass Release Cassette

The Compression Assembler release cassette contains three programs:

1) The assembler itself, packaged within a relocator to produce a 12K machine code program which loads at 1000 and upwards.

2) The ZEAP conversion utility, for producing Compass source from ZEAP source. This is a 2K program which loads at 1000 and upwards.

3) The source of the conversion utility for use as a program to try out Compass editing, and so that you could modify it to convert from other assemblers. This is about 12K and loads at 3000 and upwards (but you can copy it to other addresses using the Nas-Sys I command of course).

Each side of the cassette contains one copy of each program, in the order assembler, utility, source. Side A is recorded at 1200 baud and side B at 300 baud. The programs follow each other closely on the cassette and are recorded in standard Nas-Sys format. They should be loaded using the R command of Nas-Sys.

Before you can use Compression Assembler, it is necessary to load the first program and use the relocator to copy the assembler to a suitable position in memory. See Appendix 2.

Appendix 2: Installing Compass

Compression Assembler is supplied as the first program on either side of the release cassette, packaged within a relocator program. You should load this and execute it to place Compass at a suitable place in memory as described below.

To load Compass and the relocator, enter R and play the cassette (using the side with the baud rate that you normally use). It loads at 1000 and upwards. Then start the relocator by executing it at its first address, 1000:

```
E1000
```

It will ask you for the start address for the Compass Assembler. We suggest that you either place it at 1000 or at the very top of memory. As this version of Compass is 1980 hex (6.35K) bytes in size, suitable addresses at the top of memory are:

Computer Memory:	16K	24K	32K	40K	48K
Start Address (hex):	3680	5680	7680	9680	B680

If you have a 16K Nascom, therefore, you should enter 1000 or: 3680

The relocator will then ask you for the address to be used for its 256 (100 hex) byte workspace. By convention all Nascom assemblers use F00 to FFF as workspace, so unless you have some special reason to place the workspace elsewhere enter: F00

The Compass assembler will then be relocated, taking much less than a second.

Now you should save the generated version of Compass on cassette, so that you can avoid needing to relocate it in future. For example, if you had relocated Compass to 3680, you would start your cassette deck on record (with a blank cassette in it) and enter the following from Nas-Sys:

```
      W 3680 5000(to save 3680 to 4FFF)
```

Note that we have no objection to you making a few copies of Compass for backup purposes, but that we would consider it a breach of copyright if you made a large number of copies.

Appendix 3: <u>Starting (Initialising) Compass</u>

When you first start the assembler after loading it from cassette (or after relocating it), you should perform a cold-start by executing it at its first (start) address.

When returning to Compass after having temporarily returned to Nas-Sys (e.g to test a program that you are developing), you should perform a warm-start by executing it at its first-address+3. This leaves values such as the addresses to be used for source, code, symbol-table set as they were before. It also performs a check that the program source has not been corrupted in the time since you were last using Compass.

<u>Cold Start</u>

1) Execute Compass at its first address, giving as arguments the first and last addresses to be used for the source (in hex). You are advised to always use the same start address for all program sources for simplicity. For example, if Compass lay from 3680 to 4FFF you might enter:
```
E 3680 2000 367F
```

2) If you intend to assemble programs rather than simply modify source, use the <u>S</u>ymbol-table command to assign an area for this (allowing 4 bytes per label in your program source). For example you might enter:
```
S 1F80 1FFF
```

3) Optionally restrict the range of addresses to which code can be written using the Object command (by default code can be written to any memory address: allowing flexibility, but with some associated risk). For example you might enter:
```
0 1000 1F7F
```

<u>Warm Start</u>

Simply execute Compass at its first address + 3. Following from the example above, if you had subsequently returned to Nas-Sys for some reason, you would have restarted Compass by entering:
```
E 3683
```

This checks that the source is still OK, reporting an error if any check-sum error is detected, and returns to Compass: leaving all address settings unchanged. The Compass code is also checked. You can ignore errors if you like.

If you cold-start Compass by mistake, meaning to warm-start it, you will just get an error message due to missing source-area parameters. If, however, you entered these as well then Compass would 'delete' the existing source. It is not really deleted though and can be restored if you modify the first four bytes of the source area to spaces (£20). Thus the assembler is fairly foolproof.

Appendix 4: Syntax of Assembler Statements

The syntax of assembler statements is the same as that used by ZEAP, the de-facto standard, with the exception of a minor difference in the treatment of quotes in string constants.

Compass handles all Z80 mnemonics, but these are not described in this manual (if you've seen the size of Z80 manuals you'll appreciate why!), see the Suggested Reading Appendix 10 for references. The syntax of the assembler specific features alone is described below:

String Constants

Strings are delimited either by single or double quotes (so that, for example, a double quote can be included in a string if it is delimited by single quotes). A string can only occur as an operand of the DEFM pseudo-op, e.g:
"any characters" "the string's contents"

Constants

All numbers are assumed to be decimal unless preceeded by £ or followed by H to indicated that they are hex.

```
      £A  = 10 decimal or 0A hex
      0AH = 10 decimal or 0A hex
      11  = 11 decimal or 0B hex
```

Expressions

An expression is a sequence of one or more elements seperated by '+' or '-', where an element can be:
1) a constant (as above);
2) a label (defined by the EQU pseudo-op for example)
3) a dollar sign, representing the value of the current assemble address counter;
4) a character constant representing an ASCII value (e.g "A" or 'A').

Expressions are evaluated left-to-right and may not contain brackets.

Labels

Labels are user-defined names with associated values. A label is made up of any number of letters or numbers, although its first character must be a letter. Upper and lower case letters are permitted and 'A' and 'a' are NOT the same.

A label is defined by putting it in column 1 of a statement. For example:

```
START LD A, 'R'
....
JP END
....
JP START
....
END ....
```

A statement without a label must have a space in the first column (or have a ';' there if it is a comment).

Comments

Everything after the first ';' on a line is ignored, up to the end-of-line.

Pseudo-ops

The following pseudo-ops are recognised by Compass and have their standard meanings:

```
      ORG   expression        DEFB  expression {,expression}
label EQU   expression        DEFW  expression
      DEFS  expression        DEFM  string
```

Opcodes

Valid opcodes are as follows (assuming my typing is OK). They have their standard meanings.

```
ADC  ADD  AND  BIT  CALL CCF  CPDR CPD  CPL  CPIR
CPI  CP   DAA  DEC  DI   DJNZ EI   EXX  EX   HALT
IM   INC  INDR IND  INIR INI  IN   JP   JR   LDDR
LDD  LDIR LDI  LD   NEG  NOP  OR   OTDR OTIR OUTD
OUTI OUT  POP  PUSH RES  RETI RETN RET  RLA  RLCA
RLD  RRA  RRCA RRD  RRC  RR   RLC  RL   RST  SBC
SCF  SET  SLA  SRA  SRL  SUB  XOR
```

Character Constant

A character constant is a single-character string. It evaluates to the ASCII code of the character and can be used in expressions.

Appendix 5: Error Messages

All error messages start with the letters 'CA' and a two digit 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 summarised below with brief explanations.

CA01 Bad arguments

A) If you are cold-starting Compass, this message indicates that the addresses given for the start and end of the source area were invalid. See Appendix 3.
B) Otherwise, you used a Compass command with too many, too few or the wrong arguments. See Section 3.

CA02 Bad command

The command entered was not a valid Compass command. See Section 3.1.

CA03 Break

No error has occurred, this merely indicates that ESC was pressed during an Assemble, Change, Find or List command. The current line is set as follows:
A) If you were assembling, it is set to the first line;
B) if Compass had displayed a line and was awaiting input, it is that line;
C) otherwise it is whichever line was being processed at the time.
Enter 'P' to print the current line if you are unsure which this is.

CA04 Not found

The current line is off the end of the source because the preceding Change, Find or List command reached the end. You tried to Print this line off the end.

CA05 No memory

A Change, Edit or Input command caused the program to grow and there was no available memory for this. In the case of Change or Edit the old version of the line may have been lost, so use Print to check for this and if necessary re-Input the,line (it will have been listed previously on the screen). If possible, cold-start Compass again with a higher upper limit for the source area. In future use the '?' command regularly to keep track of program size.

CA06 Corrupted

The code of Compass has been overwritten. Press RESET and reload Compass from tape.

CA07 Bad source

Occurs after a warm-start or Read command. The source contains an ercor:
A) A line is longer than 48 characters in length;
B) The source is too large for the available source area;
C) The source checksum does not fit the source, which may be corrupt.
Continuing to run Compass may cause it to crash, but if you wish you could use commands such as ?, List etc. to pin down the problem.

CA08 Too Long

A Change command would have resulted in a line that was too long (over 48 characters), so it has not been altered. The Change command continues.

CA09 End = hhhh

Assembly has finished. The address hhhh is that of the byte following the generated code (in hex naturally).

CA10 Bad opcode

The opcode or pseudo-op in the line above is not valid.

CA11 Bad operands
The operand(s) in the line listed above are not valid.

CA12 Brackets

A closing bracket seems to be missing in the line listed above. Alternatively, some other error is in the expression ahead of the bracket.

CA 1 3 Overflow

During the evaluation of an expression in the line listed above, a result was calculated that was too big to be held in 16 bits (over 65535).

CA14 Bad char

A character constant had no closing quote. Remember that the same type of quote must start and end a character constant.

CA15 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 65535).

CA16 Undefine symbol

A label used as an operand in the line above is not defined anywhere (i.e it does not appear in the first column of any line). Have you mis-spelled it? Labels following pseudo-ops must be defined before they are used: forward references are not allowed in this case for efficiency reasons.

CA17 Bad instruction

The operands used in the statement listed above do not match the opcode.

CA18 Code range

A byte of object code would have been placed outside the limits previously set by the Object command. Use ? to display these limits, and Find to locate all ORG pseudo-ops. Maybe your program has grown bigger than you expected.

CA19 Missing symbol

No label occurs at the start of the EQU statement above.

CA20 Bad string

A string constant has no closing quote. Remember that the same type of quote must start and end a string.

CA21 Too many operands

Characters have been found after the end of the statement above and these are not part of any comment. Perhaps the semi-colon which should start the comment following the statement has been omitted.

CA22 Value

An expression that should produce an 8 bit result is too large (over 255), or if negative is less than -128.

CA23 Too far

A relative jump on the line above (JR or DJNR), would have to jump more than -126 to +129 from the current position. This is not allowed.

CA24 Too many symbols

Too little symbol table space was reserved for the program: 4 bytes are needed per label. Use ? to find how much symbol table space is reserved and Symbol to allocate more.

CA25 Duplicate symbol

The label in the line above had previously been declared earlier in the program. Labels can not be redefined. Use Find to locate the earlier declaration and decide which you want. Note that Change can be readily used to change some occurrences of a label, leaving others.

Appendix 6: Conversion to Compass from Other Assemblers

The conversion utility is the second program on each side of the cassette. It converts source files from ZEAP format so that they can be used with Compass.

To use the conversion utility, load it into memory using the Nas-Sys R command. It will load from 1000 upwards. Note that it does NOT require Compass to be in memory.

Then load your ZEAP source file into addresses from 2000 upwards. (If your version of ZEAP uses source from 1000 up: load this first, move it to 2000 using the Nas-Sys I command, and then load the conversion utility).

Now execute the conversion utility:

```
      E 1000
```

This will convert the ZEAP source file to a Compass source file starting at the same address (2000) and ends by displaying the top address of the new compressed source file. You can load Compass to work on the file, or save it to cassette etc.

Note that the conversion utility is not particularly clever and Compass may detect errors in the converted source (e.g due to the difference in string format). No problem: use the Compass 'A' command to assemble the program (with option N for No code if you like), continue the assembly until you have half-a-dozen errors on the screen and then change them and repeat the process. Because Compass assembly is so fast compared to ZEAP you will find this a simple process. Remember also that the Change command can be used to change all occurrences of a string just keeppressing ENTER each time the string is found. You will soon end up with a valid Compass program and remember that this is MUCH easier than typing the program in again from scratch!

Appendix 7: Workspace Structure

Compass uses about 256 bytes for its stack and workspace. Normally this will occupy addresses F00 to FFF as shown below, but the relocator can be used to place the workspace at any convenient address of course. The workspace is used as follows:

Address	Length	Name	Use
F00	3	BADCMD	Contains a jump instruction to a routine which prints the error message "CA02 Bad command". To add your own commands, simply replace this routine with one of your own. See Appendix 9.
F03	3	CODE	Contains a jump instruction to a routine which saves one byte of generated code. See Appendix 9.
F06	3	LINE	Contains a RET instruction. See Appendix 9.
F09	1	NULL1	Zero.
F0A	2F	buffer	Holds one source line, either packed or unpacked, during Change, Find or Insert commands.
F39	1	NULL2	Zero.
F3A	2F	FBUF	String to be found by Change or Find.
F69	2	STARTP	Address of start of source area.
F6B	2	ENDP	Address of last byte of source area.
F6D	2	OBJP	Address of start of object area (set by Object).
F6F	2	OBJE	Address of last byte of object area.
F71	2	SYMP	Address of start of symbol table (set by Symbol).
F73	2	SYME	Address of last byte of object area.
F75	1	TOKEN	Work value used during assembly.
F76	1	TEMP	Work value used during assembly.
F71	1	TEMP2	Work value used during assembly.
F78	2	NXTTOK	Work value used during assembly.
F7A	2	PAGSIZE	Value of current Height setting.
F7C	2	CURPAG	Number of lines left in current screen page.
F7E	2	BUFPOS	Address within FBUF during Change.
F80	2	NXTADR	Address of following line, usually.

Address	Length	Name	Use
F82	2	CURLIN	Line number of 'current line'.
F84	2	CURADR	Address of start of 'current line'.
F86	2F	NBUF	Replacement string used by Change.
F86	2	LINNUM	Number of line currently being assembled.
F88	2	LINADR	Address of start of line currently being assembled.
F8A	1	PASNUM	Holds 0 during the first assembly pass and £FF during the second pass (except when ORG, EQU and DEFS statements are being processed).
F8B	2	CURLOC	Address where next object byte will be stored.
F8D	2	ASMSTK	Saved value of stack pointer at the start of assembling the current line.
F8F	1	OPTLST	'Y' if listing produced, otherwise 'N'.
F90	1	OPTCOD	'Y' if code is to be stored, otherwise 'N'.
F91	1	OPTSYM	'Y' if symbol table is to be listed, otherwise 'N'.
F92	1	OPCODE	'compressed' value of opcode for line currently being assembled.
F93	1	OPRAND	Operand value for the line currently being assembled.
F94	1	OPINDX	Work value.
F95	1	PREFIX	£DD if the current instruction uses IX; £FD if it uses IY; or 0 if it does not use indexing.
F96	1	OPVAL	Value related to current opcode: for a one-byte instruction this is the code byte for example.
F97	2	POSOPR	Address of first character of operands.
F99	2	VALUE	Value of any 16-bit expression during its evalu-ation by the second pass of assembly.
F9B	1	REG8	Work value.
F9C	1	REG16	Work value.
F9D	1	CC	Value related to the condition used in any conditional jump or call statement.
F9E	1	DD	Twos-complement value used in indexed instructions, e.g 0 for JP (IX) or JP (IY).

Address	Length	Name	Use
F9F	2	LABEL	Address of symbol table entry for the label at the start of the line currently being assembled, or 0 if there is no label.
FA1	1	CODCNT	Number of bytes generated by the current instruction (unless this is DEFM).
FA2	2	CODPTR	Address within CODBUF where the next byte of code will be temporarily stored before being copied to the destination are if appropriate (see CURLOC).
FA4	4	CODBUF	Buffer for the code generated from the current instruction (unless this is DEFM etc).
FA8	58	STACK	Processor stack used by Compass.

Appendix 8: Compression Method and Token Values

Compression Assembler achieves its source compression by replacing sequences of characters by single bytes. This is much the same method as is used by ROM BASIC. As the sequences chosen relate naturally to the structure of valid assembler statements, this also allows faster assembly (and this is reason why some sequences are of length 1).

The sequences are as follows:

Code	Characters	Code	Characters	Code	Characters	Code	Characters
80	A	A0	IX,	C0	BIT	E0	NOP
81	B	A1	IY,	C1	CALL	E1	OR
82	C	A2	AF′	C2	CCF	E2	OTDR
83	D	A3	(C),	C3	CPDR	E3	OTIR
84	E	A4	(SP),	C4	CPD	E4	OUTD
85	H	A5	(HL),	C5	CPL	E5	OUTI
86	L	A6	(BC),	C6	CPIR	E6	OUT
87	I	A7	(DE),	C7	CPI	E7	POP
88	Z	A8	A,	C8	CP	E8	PUSH
89	P	A9	I,	C9	DAA	E9	RES
8A	M	AA	(C)	CA	DEC	EA	RETI
8B	AF	AB	R	CB	DI	EB	RETN
8C	HL	AC		CC	DJNZ	EC	RET
8D	BC	AD		CD	EI	ED	RLA
8E	DE	AE		CE	EXX	BE	RLCA
8F	SP	AF		CF	EX	EF	RLD
90	IX	B0		D0	HALT	F0	RRA
91	IY	B1		D1	IM	F1	RRCA
92	NZ	B2		D2	INC	F2	RRD
93	NC	B3		D3	INDR	F3	RRC
94	PE	B4		D4	IND	F4	RR
95	PO	B5		D5	INIR	F5	RLC
96	(HL)	B6		D6	INI	F6	RL
97	(BC)	B7	ORG	D7	IN	F7	RST
98	(DE)	B8	EQU	D8	JP	F8	SBC
99	(IX	B9	DEFB	D9	JR	F9	SCF
9A	(IY	BA	DEFW	DA	LDDR	FA	SET
9B	AF,	BB	DEFS	DB	LDD	FB	SLA
9C	SP,	BC	DEFM	DC	LDIR	FE	SRA
9D	HL,	BD	ADC	DD	LDI	FD	SRL
9E	BC,	BE	ADD	DE	LD	FE	SUB
9F	DE,	BF	AND	DF	NEG	FF	XOR

Spaces surrounding opcodes are automatically removed (and added back when the line is displayed on the screen) for further compression.

Compass compresses source lines very efficiently, and labels are usually the most significant contributers to the size of a compressed program because of this (together with comments). To get the best from Compass, therefore, keep labels and comments short.

Appendix 9: Hooks (Reflections)

Compass does not alter the Nas-Sys workspace, so it can probably be used with other utility programs if this is desired.

The Nas-Sys 'X' command (e.g X0) can be used to produce a hard copy listing of the source, assembly output or symbol table by copying these to a printer as they are displayed on the screen. The following sequence would do all three (assuming X0 had been entered and Compass restarted).

```
H 10000
L
A LS
```

When Compass is cold-started, three "jumps" are set up within its workspace (see Appendix 7). These are used when Compass is running and by changing them you could add code to handle:
- additional commands;
- the automatic copying of code elsewhere, or extra range checking;
- additional instructions (opcodes)

The jumps provided are:

Address	Name	Use
F00	BADCMD	This is called when Compass cannot recognise the entered command. Normally it holds a jump to the routine which prints "CA02 Bad command". DE holds the address of the start of the command on the screen. Any replacement routine should save and restore all registers and end with a RET.
F03	CODE	This is called during the second pass of assembly whenever Compass puts a byte of code into memory. A holds the byte and the destination address is held in CURLOC. Any replacement routine should save and restore all registers and end with a RET.
F06	BADOPC	This is called when Compass con not recognize an opcode during Assembly. Normally it holds a jump to the routine which prints "CA10 Bad opcode" and aborts assembly of the current line. Any additional opcodes will have to be recognised as a sequence of ascii characters, as additional opcodes will not be compressed by Compass.

Appendix 10: Suggested Reading

Many books are available about Z80 programming (and if you look at the size of them you will see why this manual had to be short on details of Z80 assembler language). Two of the best are:

Z80 Assembly Language Programming Manual

Published in the USA by Zilog, this is excellent for nitty-gritty technical details but is 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 Z80 programming. It is probably the best book to buy (from your local computer book shop, NOT from Level 9 please!).