This is a list of all the commands available in the interim version of the ZEN assembler on the Enterprise. This is version 1.1, which can be checked by typing ":HELP" while in ZEN. Many of the commands are identical to the Lynx version of ZEN and are marked "as documented". Care must be taken with upper and lower case letters in commands since in many cases they are different.

ONCE LOADED, ZEN CAN BE INITIALISED BY TYPING : ZEN (ENTER)

A a Assemble followed by prompt

Source file > <Enter> for in memory file, else any filespec for assembly direct from cassette (or other device).

Option > <Enter> for internal assembly (no list output)
V for listing to video
E for listing to parallel printer
F for listing file to cassette
X "XABS" file must ORG at 0C00Ah
A "APP" file must ORG at 100h

LENGTH is missing

Note: Zen generally looks at the ESCAPE key during assembly listings etc. Also the default number base is set to decimal by the assembler.

B Byte search. As documented.

C Copy. As documented.

c Call addresses for execution in real time, used by single step debugger.
c displays all addresses
c0 Cancels all addresses
cn Address "n" added to list of real time calls.

D Down. As documented

E e Editor. As documented.

F File. As documented.

G Global replace. As documented.

g Global move. As documented.

Hn How - quick disassembly to 8 instructions from address "n". If "n" ommitted then next 8 instructions.

In Input from I/O port "n". If "n" ommitted then use remembered port number.

J Jump. As documented.

j Jump with return. As documented.

K Kill. As documented.

L l Locate. As documented.

M Modify in Hex. As documented.

m Modify in ASCII. As documented.

N Not used.

On Output to I/O port "n". If "n" ommited then use remembered port number.

P Print to video. As documented.

p Print to printer. As documented.

Q Query to video. As documented.

q Query to printer. As documented.

R Read source (or RS). As documented (but not in DEBUG mode).

Rn Run the single-stepper from address "n" (in DEBUG mode only). If "n" is ommitted then use the current PC. Prompts for a breakpoint address (ENTER for none). Pressing "X" will stop it.

r ss. Will always be 0000Dh.

S Sorted symbol table. As documented (but not in DEBUG mode).

Sn Single step from address "n" (in DEBUG mode only). If "n" is ommitted then use current PC. Promots for breakpoint address (ENTER for none). Pressing any key will step the next instruction, except for "X" which exits.

T Target. As documented.

U Up. As documented.

V Verify source file for CRC error check. NOT comparison with memory.

NOT SUPPORTED (V) Change video mode - but not enough RAM in cassette version.

W Write source file (or WS). As documented (not in DEBUG).

Wn Sets window to "n" (in DEBUG mode only). If "n" is ommitted then switches window mode off.

X Examine registers. As documented.

Xn Examine registers with parameter exchange. As documented.

Z Zap lines as documented.

z Zero. fills the whole of free memory from the end of source to himem with 0.

:text Pass "text" around system extensions (:HELP, :BASIC etc).

Single key commands (if 1st Keystroke)

Up - up 1 line.

Down - down 1 line.

Left - back 1 page.

Right - forward 1 page.

Shift Left - Start of text.

Shift Right - End of text.

- - back 64 bytes of memory dump.

; - forward 64 bytes of memory dump.

, - edit current line.

Erase - Clear screen.

Multiple Letter Commands

DU Duplicate section of code.

DA Disassemble.
Output option V, E or F or <enter> for text buffer.

DE toDisables single step debugger ON/OFF sets default number base

RI Read input source file from cassette without upsetting text buffer.

RL Read library file from cassette. After file opened, prompts for text. File searched sequentially for left string match with text and subsequent code inserted into source file.
Terminated by - ESC key
STOP key
X in page pause
"; END" in text file.

REF Cross reference listing. As documented.

WA Write block of memory as "APP" file.

WX Write block of memory as "XABS" file.

PSEUDO OPS

LISTON - Listing is on by default.
LISTOFF

END - Must be present.
ORG

DS or DEFS - Define un-initialised storage.
DD or DEFD - Define initialised storage.
  DD 100,0 generates 100 zero bytes.
  DD 50,"." generates 50 full stops.

DW or DEFW - Define word.
DB or DEFB - Define bytes.
DM or DEFM - Define string.

SHORT - Only first 4 bytes of DM listed.
LONG - Full listing of DM strings.

EQU - Symbol equate.

LOAD - Loads source file.

IF - zero is FALSE, non-zero is TRUE.
IF NOT
IFEQ value, value - each value is one byte number or label.
ENDIF

++++++++++- END OF DOCUMENT +++++++++-+

TO COPY THE ASSEMBLER FROM CASSETTE

In order to write the Assembler to a cassette recorder or other storage device it is necessary to reduce the demands on the memory in a 64K machine. The procedure outlined below achieves this using the COPY command.

a) Push pause on recording machine

b) OPEN £200: "TAPE: ZEN" ← NO SPACE REQUIRED

c) Release pause on recording machine

d) OPEN £201: "TAPE: ZEN" ACCESS OUTPUT ← NO SPACE REQUIRED

e) COPY FROM £200 TO £201

f) CLOSE £201

g) CLOSE £200

Explanation;

b) Opens the cassette file for reading.

d) Writes out the header chunk.

e) Reads in 1st 4K of file and then writes it out.
Then reads in remainder of file but does not write it out.

f) Writes out the remainder of the file.

g) Closes input channel.

NOTE:

You must use the correct remote socket for the tape you are loading from. If you use the wrong one then step d) will make the tape you are reading from turn on, instead of the one you are writing to.

The other remote can be used to control the tape you are writing to BUT note that step b) will cause both remotes to start.

The remote for reading is the one next to the cassette IN socket on the Enterprise.

You must connect 2 cassette recorders, one to read, one to write.

ENTERPRISE DEVELOPMENT

ZEN ASSEMBLER

TABLE of CONTENTS

	page
Introduction	1.1
Preliminaries	
Loading	1.1
Keyboard Input	1.2
Memory Usage	1.2
Command Summary	1.3
Monitor Commands	
Byte Search	2.1
Copy Memory	2.1
Colour	2.1
Fill	2.2
Ink	2.2
Modify Numeric	2.2
Modify Ascii	2.2
Query and Query-to-printer	2.2
Zero free memory	2.2
Editor commands	
Text Entry	3.1
Text Editing	3.1
Text Manipulation	3.2
Down	3.2
Global replace	3.2
Global move	3.2
Howbig	3.2
Kill	3.3
Locate	3.3
Print and Line-Print	3.3
Target	3.3
Up	3.3
Zap	3.3
Page and Scroll	3.3
Source Syntax	3.3
Assembler Commands	
Assemble	4.1
Output options	4.1
Increase symbol table	4.1
Reference listing	4.2
Sorted symbol listing	4.2
Comments	4.3
Operators	4.3
Operands	4.3
Labels & Symbols	4.4
Pseudo-ops	4.5
Assembler output	4.7
Example Source File	4.8
Error messages	4.9

Debugging commands
Debugging 5.1
Re-entry address 5.1
Jump 5.1
Jump with return 5.2
Operating system 5.2
Examine registers 5.2
Exchange registers 5.2

Cassette Input / Output
Write source 6.1
Verify source 6.1
Read source 6.1
Read object 6.1
Verify object 6.1
Write object 6.1

Miscellaneous Section
Rom routines 7.1
Printer driver and Printer commands 7.1
Zen pointers 7.2
Re-entering Zen from RESET 7.2

: string Sends string to operating system for extension scan

For I/O redirection, use device names.

Z E N Editor / Assembler

Introduction

Zen is a complete package for the creation of Z80 machine code programs and combines a text editor, fast Z80 assembler, machine code monitor and de-bugger. A comprehensive range of commands is available for loading and saving source and object files to and from cassette.

The original 4K Zen was created by John Hawthorne and this extended Lynx version has been produced by Laurie Shields. The very first Zen was hand assembled by John shortly after the Z80 chip was invented and all subsequent improvements and adaptions to other micros, (Nascom, Sharp, T S80, Lynx, Colour Genie, etc), have been by Zen re-assembling improved versions of the source code.

The structure of this manual will follow the logical division of Zen's functions and their associated commands into seven main categories, namely, a Preliminary section on loading, keyboard input and memory usage, Machine code monitor, Text editor, Assembler, Debugger, Cassette I/O, Miscellaneous for the bits and pieces.

Preliminaries

Loading

Zen is supplied on cassette in Lynx 500 baud machine code format, the filename is "ZEN". To load use the MLOAD command as follows:

```
MLOAD "ZEN" <cr>    Where <cr> stands for the RETURN key.
                    If a bad read is detected then the Lynx displays
                    an appropriate message and jumps to its Monitor.
                    To return to Basic key: J <cr>,
                    else to re-read the tape key   "ZEN" <cr>.
```

Once successfully loaded Zen will take over control.

After initialising the screen will clear, the copyright notice is shown and following any keystroke the command loop prompt is displayed.

```
Z>
```

Zen now wants a command.

Keyboard Input

Whilst awaiting a command, Zen is constantly scanning the keyboard testing for one of a number of special keys, but only when pressed as the first keystroke following the Z> prompt. These keys are decoded as immediate commands to give greater ease of use without always waiting for the <cr> key.

The up and down arrows move up and down one line at a time through the source file; if no file exists or an attempt is made to go beyond the end of the file then an EOF error message is displayed. The shift-up and shift-down arrow keys move to the start or end, and the right and left arrows paging forwards or backwards. The ',' key means edit the current line and Return by itself clears the screen. The use of these commands is explained fully in the text entry section.

If none of the foregoing keys are pressed then Zen stores the inputted keystrokes in a buffer, no action being taken until the <cr> key is pressed. Zen's commands generally take the form of a command letter followed by in some cases an optional numeric or character parameter. Spaces before the command letter, or between the letter and parameter are not permitted. Numeric parameters can be postfixed with H or h (hexadecimal), O or o (octal), d (decimal) or b (binary).

Typical single letter commands without parameters are A for assemble and X for examine user registers. Examples of commands with parameters are Q7A00H for query memory block starting at 7A00 Hex, and LCassette for locate the text string "Cassette" in the source file. Commands which expect a parameter where none is given default to either the number 1 or the last used parameter as explained in detail later on.

Memory Usage

Zen is a low memory machine code program that starts at 6C00H and occupies memory up to about 8400H depending on version. The next 512 bytes following the end of Zen are reserved for the Symbol Table (explained later) and immediately following the symbol table will be the source file for the assembler. The assembler source code is stored in memory exactly as keyed in without line numbers and without tabs between fields. The memory size limit set prior to loading Zen is respected at all times and the size of the source file and the Himem value can be displayed by the command H, for Howbig.

Command Summary
==========

A	Assemble	a	~~Not used~~ assemble
B	Byte search	b	Not used
C	Copy memory block	c	~~Colour (paper)~~
D	Down	d	Not used
E	Enter editor	e	Not used
F	Fill	f	Not used
G	Global replace	g	Global move
H	Howbig	h	Not used
I	Increase Symbol Table	i	~~Ink~~
J	Jump	j	Jump with return
K	Kill	k	Not used
L	Locate	l	Not used
M	Modify (Numeric)	m	Modify (Ascii)
N	New line (edit)	n	Not used
O	~~Operating system~~	o	Not used
P	Print	p	Line-print
Q	Query	q	Query to line-printer
R	Read	r	Not used
S	Sorted Symbol Table	s	Not used
T	Target	t	Not used
U	Up	u	Not used
V	~~Verify~~	v	Not used
W	Write	w	Not used
X	Examine registers	x	Exchange registers
Y	Not used	y	Not used
Z	Zap text	z	Zero free memory.
REF	Cross reference listing		

Monitor Commands

These monitor commands are provided for direct inter-reaction with t Lynx's memory:

B	Byte search	numeric parameter
C	Copy	
c	Colour (paper)	numeric parameter
F	Fill	
i	Ink	numeric parameter
M	Modify in hexadecimal	" "
m	Modify in Ascii	" "
Q	Query memory	" "
q	Query memory to printer	" "
z	Zero free memory	

B Byte search, initialises a search for the parameter value. Where the parameter is greater than 255, Zen will search for the 2 byte value stored in the Lsb/Msb Z80 format, otherwise the search will be for the single byte.
Zen will then prompt for the Start and Stop addresses.
Memory will be searched accordingly and on locating the first corresponding byte, its address will be displayed.
To continue the search just key B.

Example ... Z>B27 <cr>
Start>1000H <cr>
Stop>1500H <cr>
On the early Lynxs Zen will respond with:
Address 1158
The search can be continued with B <cr> until the end of search address is reached whence Zen will display the message: Stop>.

Note: Modify, Byte search and Query all use and maintain the s current object pointer. Having found a particular byte by the sea routine then just keying M or m <cr> will enter the appropriate mod mode at that address, or keying Q <cr> will display the memory bl starting at that address.

C Copy, copies a block of memory from Start Address to Stop Address inclusive to a Destination Address.
The copy is intelligent and overlapping areas of memory can be moved with complete security. If the addresses do not overlap then the source data remains intact.

c Colour, the supplied parameter is used as the colour for the background in subsequent printing.

F Fill, will fill a block of memory from Start Address to Stop Address inclusive with a Data constant.
During Copy and Fill no check is made on whether the memory receiving area is correctly storing the data.

I Ink, the suplied parameter is used as the ink colour for all subsequent printing.

M Modify, lets you examine and modify memory contents. If you supply an address parameter then display commences at that address. Modify takes a default of the Current Object Pointer which is an object pointer similar to the source current line pointer.

The address and the byte at that address will be displayed in hex followed by a prompt for input data. The default response will cause a step onto the next address. Entering an actual value causes that byte to be replaced with your parameter. Your parameter is erased on the video display, the new value is displayed and Zen steps to the next address.

Entering the character "-" as a parameter means back-step one byte and to exit simply key a full stop "." as your input.
The current object pointer is left pointing at the last byte displayed.
Zen checks after each modification and will exit to command mode on an error, eg trying to modify ROM.

m Modify-in-Ascii, similar to modify but only single character accepted as input, except for exit ".." and backstep "--".

Q Query, displays a block of memory in hex and as literal Ascii. This command also takes the current object pointer as the default parameter.
Memory is displayed as eight lines of eight bytes, each line comprises three fields:

1 Address
2 Eight bytes in hex
3 Eight bytes literal

Control characters (less than 20H and greater than 0BFH) are converted to full stops to prevent the display reacting.
The current object pointer is left pointing at one greater than the last byte displayed.

q Lineprint Query, as above but with output to the printer.

z Zero, fills the whole of free memory from the End-of-Source to Himem with binary 0.

Editor Commands

These commands are provided for the creation and editing of the assembler source file. The text is stored in the computer's memory and when the text entry or editing is complete the whole file can be written to cassette.

D	Down through file also down-arrow	numeric	parameter
E	Enter text		
G	Global replacement		
g	Global move		
H	Howbig source file		
K	Kill in-memory file		
L	Locate text string	character string	parameter
N	New, (edit) current line also comma		
P	Print lines to screen	numeric	parameter
p	Print to lineprinter	"	"
T	Target to line	numeric	parameter
U	Up through file also up-arrow	numeric	parameter
Z	Zap (delete) lines	numeric	parameter
→	Page forward		
←	Page backward		

Shift up-arrow moves to start of file
Shift down-arrow moves to end of file

Text Entry

Taking the 'E' and 'N' commands first of all, as these are concerned with the process of text creation.

E <cr> causes Zen to enter the text entry mode by displaying the current line number and awaiting keyboard input, similar in fashion to the auto command in Basic. If no source exists then the line numbering starts at 1, otherwise the editor enters the file at current line number, pushing that line and all after it down to make room for the new ones. Keying 'N' followed by <cr>, or just comma if first keystroke, engages the edit mode for the current line.

For text entry and editing Zen uses the Lynx's Rom routines for character input. During assembly Zen formats the output to the video and printer so there is no need to set out the source code in tabular or field format. A completely free format is accepted, the only requirement being that a colon ':' is placed immediately after a label and that a space is generally required between the Opcode and the Operand.

When displaying lines with the various editor commands Zen examines the line first to determine whether or not it contains a label. If a label is present then Zen inserts four spaces into the display after the line counter so that the labels stand out, otherwise the line is displayed after the counter exactly as it is stored in memory.

To terminate the editor mode enter a period '.' as the first character of the next line.

Text Manipulation

As the Z80 Assembly Language is entirely line orientated the Editor in Zen is line rather than character orientated. Zen always maintains an internal pointer to the current line in the source file; the current line always being the last one displayed.

D Down, pointer moves down towards the end of file (parameter) lines.
Example ... D37 <cr> moves the pointer down 37 lines.
D <cr> is the same as down-arrow, moving down one line.

G Global, replaces a search text string with a replacement one.
Without a parameter, eg just G <cr> the whole of the file is searched. Where a numeric parameter is supplied, eg G3 <cr> only the parameter number of lines are searched starting at the current one.
Following the command you are prompted for search and replacement text strings. A default entry to either prompt causes an abort back to the command loop prompt.

```
Example ... Z>G <cr>
            Change>CASSETTE <cr>
            To>Tape <cr>
```

Would result in every occurrence of 'CASSETTE' in the whole of the source file to be changed to 'Tape', with a reduction in file size of four bytes for each change.

```
Example ... Z>G1 <cr>
            Change>HL <cr>
            To>BC <cr>
```

Would result in the substitution of BC for HL in the current line only (as the parameter 1 was supplied). So if the current line was LD HL,1234 then it would become LD BC,1234

g Global move, is a block move of any number of lines of text from one part of the source file to another. To execute the command Zen needs the start and end line numbers of the block, the line number in front of which the block is to be placed and sufficient free memory equal to the size of the block, otherwise the command will be aborted with a Memory Full error message. Zen also checks that the destination line number lies outside the range of the block. To move a block to the end of the file a destination number one greater than the ending line is accepted. On completion the first line of the file is displayed.

```
Example ... Z>g <cr>
            Start>10 <cr>
            Stop>20 <cr>
            Dest>1 <cr>
```

This sequence of commands will move lines 10 to 20 inclusive from their original position to be in front of line 1, ie the start of the file. Lines 10 to 20 become lines 1 to 11, the old lines 1 to 9 become 12 to 20 and lines 21 onwards are the same.

H Howbig, displays the start and end addresses of the in memory source file and also the highest available byte in Ram.

K Kill, erases the source file from memory by making the end of file pointer equal to the start, this being the state of the pointers when Zen is initially entered. Note that after killing a file it's contents are still in memory but not accessible as they are beyond the new Eof. New text will overwrite the old file.

L Locate, will find an arbitrary target string in the source file.
Example ... LBIT 7,(HL)
Moves the pointer to the first line containing the string 'BIT 7,(HL)' makes it the current line and displays it. The file is searched downwards from the current line. If the string is not located the pointer is at EOF.
There are no restrictions on the string content.
L without a parameter defaults to the previous search string.

P Print, displays (parameter) lines on the video display.
The last line displayed is the new current line.
Example ... P10 <cr> will print 10 lines to the video in scrolling mode without clearing the video first.

p Line Print, as for Print but output to the printer.

T Target, pointer moves to the parameter line number.
Example ... T25 <cr> makes line 25 the current line.

U Up, pointer moves up (parameter) lines.
Example ... U37 <cr> moves the pointer up 37 lines.
The up and down arrows also scroll one line at a time.

Z Zap, erases (parameter) lines from the file, starting with the current one.
Example ... Z37 <cr> will delete the current and next thirty six lines from the text.

→ Page forwards, the video screen is cleared and the next fifteen lines of text, starting at the current line, are displayed.

← Page backwards, Zen assumes that the current line is at the bottom of the screen and after clearing the video, 23 lines, terminating with the one 22 before the current one, are displayed.

Note Zen uses the following syntax in the source coding:

(1) EX AF,AF rather than EX AF,AF'

(2) The abbreviated format of just (IX) or (IY) instead of (IX+0) and (IY+0) is not permitted.
So that LD A,(IX) shown in some listings must be coded LD A,(IX+0)

(3) Negative offsets to the index registers must be shown as such rather than as a positive offset of a negative value, eg:
LD A,(IX-10H) rather than LD A,(IX+0F0H)

(4) When assembling to cassette the program execution address must be indicated to the assembler with the EXEC pseudo-op.

Assembler Commands

The following commands are provided for assembling the source file and production of the resulting object code or list files.

A	Assemble.	
I	Increase Symbol Table	numeric parameter.
REF	Cross reference listing	character parameter.
S	Sorted symbol table listing	character parameter.

A Assemble, the source file from start of file to the END pseudo-op. Zen will then prompt for an output option, which are:

V	Video	(List to video).
E	External	(List to external printer and video).
C	Cassette	(Object code to tape in MLOAD format).

The default option, ie just <cr>, generates no output (except if a LOAD pseudo-op is included within the file then object code generated will be loaded into memory). This is the fastest assembly mode and should be used until all source errors are eliminated. If you select the cassette option then you will be further prompted for a filename, which must be in double quotes.

I Increase, will increase the amount of memory allocated for the symbol table by the numeric parameter, provided there is sufficient free memory available, otherwise an error message is displayed showing where the Eof would have resulted.

Example ... I300 <cr> gives an increase of 300 bytes to the symbol table and a corresponding reduction in the text buffer.

Initially Zen reserves 512 bytes for creating, during the first pass, a look-up table of label names and their associated values. With large programs this may not be sufficient and assembly would terminate with the error message 'Full'.
The increase in symbol table size is achieved by moving the whole of the source file towards Himem.

REF Reference, produces a listing of all the labels stored in the symbol table, their hexadecimal values, the line number of their occurrence, and all references to them in the in-memory source file. If a selector parameter letter is provided, eg REFX <cr>, then the listing would be restricted to labels beginning with that letter.

After the command you will be prompted for an output option, similar to those for the assembler list file.

S Sort, alphabetically sorts the symbol table (first letter only) built during the previous assembly. Adding a selector letter to the command will restrict the listing to symbols beginning with that letter. Output options as for REF.

Example ... SG <cr> will give all the labels beginning with G.

Zen expects source statements to be constructed according to the syntax defined in the Zilog Z80 Assembly Language Programming Manual. Each line of the source file is a statement divided, conceptually, into at most four fields:

```
MESSAGE: LD HL, GREETING; Say hello

Label
. . . . Operator
. . . . . . . . Operand(s)
. . . . . . . . . . . . . . . Comment
```

We say conceptually divided because the components of a statement don't have to be positioned into fields. As long as you use the correct separators (spaces, commas, etc.) ZEN accepts statements in free format.

COMMENTS Comments are ignored by the assembler. They are preceded by a semi-colon ";" and are terminated by end of line.

OPERATORS There are 74 generic operators (CALL,LD,JP,etc.). In addition there are the PSEUDO-OPS described later.

OPERANDS The number of operands in a statement depends upon the operator. Examples:

```
NOP     No operands
JP      One operand
BIT     Two operands
JR      One or two (JR SYMBOL or JR Z,SYMBOL)
```

Operands may be:
Register names (A,B,DE,IX, etc.)
Condition codes (Z,NZ,C,M,etc.)
Numbers

The number group is the most complex. All the following are accepted as numbers:

ASCII LITERALS
The assembler will generate the ordinal value of any character enclosed in single or double quotes.

NUMBERS
Decimal, hex, octal and binary bases are accepted with decimal the default. Hex numbers are "H" postfixed, octal numbers are "O" postfixed and binary are "B" postfixed. Numbers must begin with a digit, leading zero is sufficient.

SYMBOLS
These are explained in detail later on.

PROGRAM COUNTER
This is an internal variable which simulates the runtime PC. It is accessed by using $ (dollar).

OPERANDS In addition all of the preceding data types may be elements of an expression formed using the infix mathematical operators:

+	Addition	/	Division
-	Subtraction	&	Logical AND
*	Multiplication	.	Logical OR

An expression can be used anywhere that a simple number can be used. The following are all valid:

```
LD DE, START*2-782
LD A,"P".8OH
JR NC,$-5
```

Expressions are evaluated strictly left to right with no precedence ordering. Arithmetic is unsigned 16 bit integer and overflow will be ignored.
Elements in an expression need not be delimited by separators as the math operators are implied separators.

LABELS A label is a way of marking a statement. Each time you use an operator like JP, CALL, etc. you need a way of specifying the destination as an operand. Assembly language allows you to use a symbolic name as a label. BASIC is an example of a language without this facility, the line numbers act as labels. Symbols will be explained in greater detail.

SYMBOLS

A symbol is a name with an associated value, the name is used rather than explicitly stating the value. A symbol's value is declared to the assembler in one of two ways:

1 By placing it at the start of a statement. The assembler assigns the value of the program counter to it.

2 By using the EQU pseudo-op.
This allows you to assign your own value to a symbol.
Example BACKSPACE: EQU 8

Whichever method is used a symbol must be postfixed with a colon ":" when declared. A symbol must begin with a letter but may contain letters or numbers after that. Letters may be upper or lower case. Zen allows symbols of any length although symbols longer than seven characters will affect the listing.

There are certain reserved keywords which cannot be used as symbols. These are:

Operator names, register names and condition code names.

Note that all keywords are uppercase, using the same name in lowercase would be perfectly acceptable as a label.

PSEUDO-OPS

These are additional operators which have no equivalent in the Z80 instruction set but are understood by the assembler. They are used in the same way as the normal operators.

LISTOFF	Stops listing	(No operand)
LISTON	Cancels Listoff	(No operand)
END	End assembly	(No operand)
DS or DEFS	Define Storage	(One operand)
DW or DEFW	Define Word	(One operand)
EQU	Equate	(One operand)
ORG	Origin	(One operand)
EXEC	Execution address	(One Operand)
LOAD	Load memory	(One operand)
OFFSET	Offset object code	(One operand)
DB or DEFB or DEFM	Define Byte(s)	(Multiple operands)

END This operator MUST be used to terminate assembly. Failure do do so will result in an error message and an incomplete assembly.

DS or DEFS Skips a number of object bytes. Commonly used to reserve space for a text buffer, stack, etc., where the object code doesn't need to be defined.

DW DEFW Generates a word (two bytes) in the object file in reversed order as required by the Z80 sixteen bit instructions.

Example ... BUFFER: DW VALUE
Would make the contents of location BUFFER equal to VALUE.

DB DEFB DEFM Generates the value of the operand(s) in the object file, takes as many operands as desired, separated by commas.

Example ... DB 6, 93H,"T".80H, NEWLINE

Each operand may be an expression but obviously no expression can have a value greater than two hundred and fifty-five decimal. The program counter will be incremented after every operand as if each were on a separate line.
In addition to the usual data types any operand may be of the type ASCII literal string.

Example ... MESSAGE: DB"Ready Cassette", NEWLINE

Strings may be of any length but, unlike single character ASCII literals, may not form part of an expression. A string is formed in the same way as a single character literal, by enclosing in matching quotes. Note that single and double quotes are implied separators like the infix math operators.

You may use a quote, of either type, as a literal by using the OPPOSITE type of quote as the delimiters.

EQU Assigns a value to a symbol.

Example ... NEWLINE: EQU 13

The operand may, as usual, be an expression but there is a restriction on the symbols you may use in the expression. This is because the operand must be capable of immediate resolution. The value of any symbols used in the expression must already be known to the assembler, forward referenced symbols will result in the UNDEFINED error flag.

Example ... NEWLINE: EQU BACKSPACE+5
BACKSPACE: EQU NEWLINE-5

This sequence is ILLEGAL because each symbol is defined in terms of the other. The "no forward reference" rule is designed to prevent you making such a mistake inadvertently.

In practice you will probably never encounter such a situation as most EQUATES have simple operands.

ORG Defines the origin of the object file. This operator may be used as often as desired throughout an assembly to produce sections of code at different locations.
The operand must conform to the "no forward reference rule for obvious reasons.

EXEC Tells the assembler the execution address of the program if being output to cassette.

LOAD Lets you load the object code into memory, at any address, as it is produced. Note that the use of a subsequent ORG operator turns off the loading process; each time you use a new origin you must specifically re-establish the LOAD command.

The loading process is entirely independent of the output option specified on entry to the assembler. If the operator is not used then no memory location outside Zen will be altered.

Note: The LOAD instruction must not precede the ORG pseudo-op. As the LOAD is independent of the ORG address it is possible for code to be generated for one location in memory and loaded into another. After assembly the 'C' command can be used to copy the code to its correct location.

OFFSET Tells the assembler to add the value of the operand to all load addresses of object code whether to cassette, disc or listing file. Assembly defaults to an offset of zero and this pseudo-op may be ignored unless it is required to produce code to load at one address but to execute following a move to another.

LISTOFF Tells the assembler to stop generating any output to the video or printer.

LISTON Re-establishes the listing process. Assembly always starts in this mode.

Assembler Output

Zen supports two list devices, the 40 character x 24 line video and an external printer.

Zen generates completely formatted output to all the devices.
Listings are output a page at a time with a delay between pages.
The pages are set at 23 lines to the video and 60 lines to the printer.

Pause Control: Pressing any key during the delay between pages will hold the listing until another key is depressed. If the key pressed is 'X', then the current operation is aborted and Zen returns to the command mode.

Sort and Cross Reference outputs are similar in principle to assembly listings with pause control at the end of each page.

The following iistings of a short machine code program to print a message on the video and await the <cr> key input is shown in three forms:

a) Text as keyed in with Zen's auto line number prompt,
b) Source file as displayed on the video with the page command, and
c) the totally formatted assembler output.

a) Text as keyed in, the shaded characters are Zen's prompts:

```
Z>E
    1 ORG 8A00H
    2 LOAD 8A00H
    3 START:EXEC START
    4 LD HL, MESSAGE
    5 LOOP:LD A,(HL)
    6 INC HL
    7 OR A
    8 JR Z,KEYIN
    9 RST 8H
   10 JR LOOP
   11 MESSAGE:DB'HIT RETURN',0
   12 KEYIN:CALL 202FH
   13 CP 13
   14 JR NZ,KEYIN
   15 RET
   16 END
```

b) Page display of file:

```
    1      ORG 8A00H
    2      LOAD 8A00H
    3 START:EXEC START
    4      LD HL,MESSAGE
    5 LOOP:LD A,(HL)
    6      INC HL
    7      OR A
    8      JR Z,KEYIN
    9      RST 8H
   10      JR LOOP
   11 MESSAGE:DB'HIT RETURN',0
   12 KEYIN:CALL 202FH
   13      CP 13
   14      JR NZ,KEYIN
   15      RET
   16      END
```

(c) Assembler Output

This is the final output format from the assembler using the 'V' or 'E' output option. The page numbering is omitted due to limited screen size when outputting to the video.

```
Page  1

    1                           ORG  8A00H
    2                           LOAD 8A00H
    3                 START:    EXEC START
    4 8A00 210B8A               LD   HL,MESSAGE
    5 8A03 7E         LOOP:     LD   A,(HL)
    6 8A04 23                   INC  HL
    7 8A05 B7                   OR   A
    8 8A06 280E                 JR   Z,KEYIN
    9 8A08 CF                   RST  8H
   10 8A09 18F8                 JR   LOOP
   11 8A0B 48495420 MESSAGE:    DB   'HIT RETURN',0
   11 8A0F 454E5445
   11 8A13 524E00
   12 8A16 CD2F20   KEYIN:      CALL 202FH
   13 8A19 FE0D                 CP   13
   14 8A1B 20F9                 JR   NZ,KEYIN
   15 8A1D C9                   RET
   16                           END
Exec Addr 7A00
```

Points worth noting:

(1) Leading zero suppression on all decimal numbers.
(2) The fully formatted output.
(3) The LOAD pseudo-op in line 2 following the ORG in line 1.
(4) Line 11 has a seven character label 'MESSAGE'.
(5) In line 11 the zero byte is included after a comma.
(6) All the bytes of the message are expanded and their hex values given against their memory address.

Assembler Error Handling

If the assembler finds an error in the source code the following will happen:

(1) The ROM cassette 'Off' routine is called.
(2) Assembly terminates.
(3) An ERROR message is displayed.
(4) The incorrect line becomes the editor current line.
(5) The line is displayed.
(6) The command loop is re-entered.

You can now correct the error and assemble again.

Error Messages

```
Double Symbol . . You have declared the same symbol more than once.
Undefined . . . . You have used an undefined symbol.
Reserved  . . . . You have used a reserved keyword for a symbol.
Symbol  . . . . . An obligatory symbol is missing (eg with EQU).
Full  . . . . . . The symbol table is full  (Use I command).
EOF . . . . . . . You have forgotten END and have hit EOF.
ORG ! . . . . . . No origin specified.
Huh?  . . . . . . The line doesn't make sense.
!   . . . . . . . Unexpected condition such as <ESC> key.
Operand . . . . . Something is wrong with the operand......
                  Examples ..... LD A,256    (too big)
                                 BIT 9,B     (no bit 9)
                                 LD(DE),C    (illegal instruction)
                                 LOAD over Zen's code or source file
                  Also any attempts to index or jump relative
                  out of range.
```

The assembler will catch all incorrect statements.

Debugging Commands

The following commands are provided for use with Zen's inbuilt de-bugging feature. They are intended primarily as an aid during the process of creating a machine code program and consequently a disassembler facility is not included. Where the user needs to monitor the workings of machine code programs other than his own, then a more dedicated de-bugger package is recommended. A number of Zen's commands, such as Bytefind, although documented under the Monitor section, could also have applicability during de-bugging.

J	Jump	numeric parameter
j	Jump with return	numeric parameter
O	Operating system return	
X	Examine user registers	
x	Exchange user registers	

Debugging

Should you wish to test a piece of machine code it is necessary to instruct the Z80 to jump out of Zen's control to the memory address where the start of the new routine is situated. Usually once the test program has executed we wish control to be returned to Zen for further development. This requires a special command to the Z80 after the last instruction we want executing, so that the Z80 knows where to find Zen.

There are three ways of doing it. Firstly Zen has a debugging re-entry address three bytes after the normal start and we can include in the test program's source file CALL REENTRY wherever we want control to return to Zen. Naturally we must also include REENTRY EQU 6C03H, this being the value of Re-entry for the Lynx tape version of Zen. Secondly, the lower case 'j' Jump command puts Zen's re-entry address onto the user stack prior to jumping so that provided the Stack Pointer isn't moved and the routine ends with a return (C9H), then control will pass back to Zen. The third method employs the 'breakpoint' technique, where at the terminating address of the program under test the actual coding is altered to an instruction to return to Zen. Once Zen is re-entered the contents of the breakpoint address are restored to their original value. This is the method used by Zen following a 'J' command, where the next response from Zen is a prompt for the address of the breakpoint. Naturally this technique is only applicable to Ram based programs as it is impossible to set a breakpoint in an unalterable part of Rom.

J Jump, tranfers control to a user program at the parameter address supplied. You will then be prompted for a breakpoint address. Both of these input parameters have default values.

By keying just J <cr>, execution will commence at the existing User Program Counter (UPC), which is displayed. The default response to the breakpoint request means that no breakpoint will be set. If you supply a breakpoint then the byte at that address is saved and a RST 30H instruction is inserted there. When the breakpoint is encountered control is transferred back to Zen via the REENTRY vector to the TRAP handler where the original code is replaced. The entire Z80 machine state is saved by TRAP and completely restored to the machine at JUMP.

j Jump-with-Return, is similar to Jump but the user program is treated as a subroutine which has been called. The machine state, with the exception of the SP and PC, is restored, the address of the TRAP handler is put onto the stack and control is transferred to the parameter value. If no parameter is supplied then the default of the last previously supplied value is used. Control will eventually return to Zen via a RET instruction and consequently it is impossible to trap the program counter prior to the return to Zen, but otherwise the whole of the machine state is recorded and available for inspection.

If you have keyed in the little sample program given in the previous section then after assembling it you can use the above de-bugging commands. Since the coding is written as a sub-routine use the Jump-with-return command by keying:

```
Z>j8A00H <cr>
```

The message should be displayed and until you press the Return key your Lynx will be completely un-responsive as the routine ignores all other keystrokes. On return to Zen the usual command loop prompt is shown.

Alternatively you could use the normal Jump command and set a breakpoint at the end of the program to re-enter Zen. The breakpoint would be set at the address of the RET command, ie 8A1DH and the command sequence, including Zen's prompts is as follows:

```
Z>J8A00H <cr>
Brkpnt>8A1DH <cr>
```

The result will be the same as for Jump-with-return and when back at command level you can use the examine registers facility to see the complete machine state of the Z80 at the breakpoint.

O Operating system return. On receiving this command, which does not take any parameters, Zen jumps to 38E8H, the re-entry address for the Lynx Basic.

X Examine user registers. The machine state recorded by the TRAP handler, or if prior to any de-bugging the initialising register values, is displayed. All the Z80 registers are labelled and displayed with the main or prime registers on the top line and the alternate or non-prime set on the second line. Both the flag registers are fully decoded in the conventional fashion. The value shown for the PC register is the default address for the U used by Jump (not jump-with-return), and the SP register shows the User Stack location that will be used during debugging.

x Exchange user registers. The user registers are displayed in turn with prompts for their new values. The default entry leaves the values unchanged and a '.' will exit the routine.
If the AF register pair are altered then the flags are decoded.