ZEN DEBUG
OPERATING INSTRUCTIONS

SHARPSOFT Ltd.

ZEN DEBUG is an add-on program module for the ZEN Z80 assembler, and offers a very flexible and easy to use Z80 machine code development system. The combination of ZEN and ZEN DEBUG also provides an ideal machine code teaching aid.

LOADING

After loading ZEN, return to the monitor by jumping to Ø with the Goto command. After loading ZEN DEBUG in the normal way, control will be passed back to ZEN.

USE

After loading ZEN DEBUG, a new command 'B' will be available from ZEN. Using this command in the usual way will result in a display similar to that shown in figure 1.

COMMANDS

All commands are used in the same way as in ZEN, ie. the appropriate letter followed by any number that may be required followed by 'CR'.

The commands available (explained later) are:

Z ... Zen	Tn .. Test	A ... Absolute	K ... Klear
F ... Fill	W ... Work out	' ' .. centre	S ... Set flags
C ... Copy	Gn .. Goto	R ... Registers	+n .. add
Ln .. Locate	Jn .. Jump	Mn .. Memory	-n .. subtract
Nn .. Next	P ... Print	Un .. Unassemble	
Vn .. Verify	D ... Displacement	X ... Xchange	

NUMBERS

Numbers in ZEN DEBUG are entered in exactly the same way as in ZEN, ie. followed by 'H' for hex. numbers, 'O' for octal numbers, and nothing for decimal numbers. Additionally, binary numbers may be used by following the number with a 'B'. ZEN may also now use binary numbers in the same way (both in and out of programs).

THE DISPLAY

The display is shown at all times when ZEN DEBUG is in use. A small amount of screen 'snow' should be expected due to the speed at which the display is created. A typical example of the display is shown in figure 1.

There are two modes of operation: the mode dealing with the Z80 registers (top half of the display; input prompt is 'REG>') and the mode dealing with the memory (lower half of the display; input prompt is 'MEM>'). The middle four lines are used as a scrolling text area.

The arrows are labelled in figure 1 for reference later.

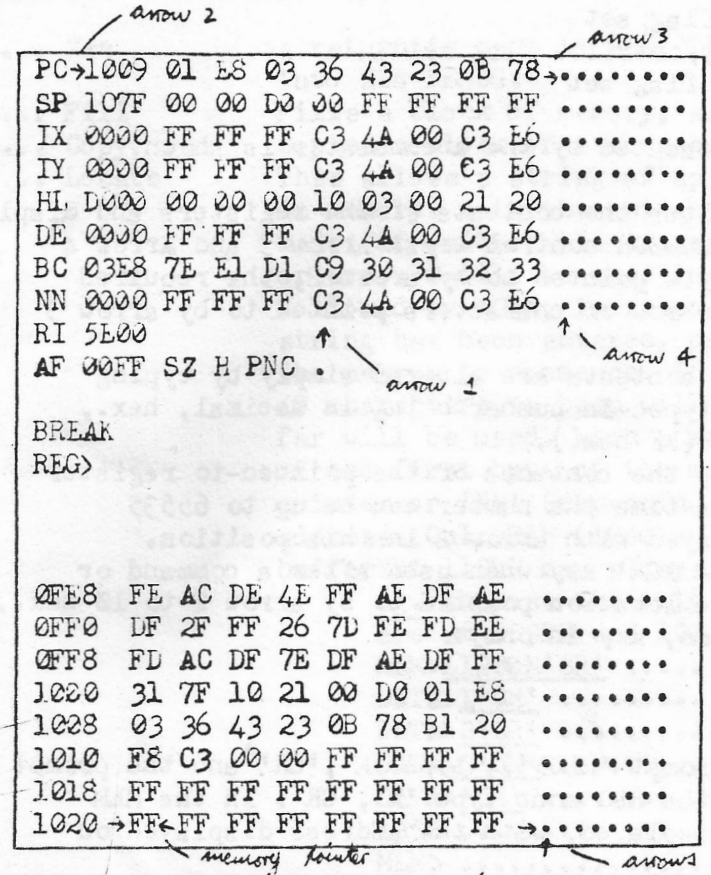
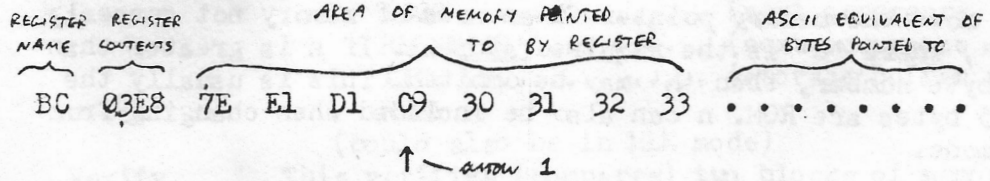


FIGURE 1 : A TYPICAL DISPLAY (actual numbers may differ from those shown)

REGISTER DISPLAY

The letters down the left-hand edge represent the Z80 registers (PC=program counter, SP=stack pointer etc.). The meaning of the various other characters on the line is best described by an example. Taking the 'BC' row as the example:



NOTE: The exact byte pointed to by any register is always displayed in the column pointed to by Arrow 1 (see figure 1).
The 'NN' row corresponds to a 'pseudo-register' NN. Its function is simply to allow any area of memory to be examined, and has nothind to do with the Z80 registers.

On the 'RI' row, the contents of the refresh ('R') register and the interrupt ('I') register are shown: these do not of course act as a combined 16-bit register pair, so no memory contents are shown on the rest of this row.

On the 'AF' row, the contents of the accumulator and flag registers are shown. These are followed by the flag display, using the standard letters to indicate any flags set, as shown below:

S: Sign flag set
H: Half-carry flag set
N: subtract flag set
Z: Zero flag set
P: Parity/overflow flag set
C: Carry flag set

After the flags, the ASCII character represented by the accumulator is shown.

Arrow 2 (see figure 1) is used to alter the contents of the registers and displayed memory. It can be moved using the normal CURSOR control keys. Arrow 3 and Arrow 4 together point to the equivalent of the byte pointed to by arrow 2, the required character being at the intersection of the row of characters pointed to by arrow 3 and the column pointed to by arrow 4.

Assuming arrow 2 points to RAM, the contents are altered simply by typing in the new contents and typing 'CR', the typed-in number being in decimal, hex., octal or binary, and not greater than 255 (FF hex.).

If arrow 2 is moved to the far left, the contents of the pointed-to register can be changed in a similar way, only this time the number can be up to 65535 (FFFF hex.). Arrows 3 and 4 are not displayed with arrow 2 in this position.

Note that the CURSOR keys act as the 'CR' key when used after a command or number. Thus to change the contents of the location pointed to by arrow 2 to 10 hex., and to move arrow 2 on to the next location, key in only:

'1','0','H','CURSOR'→

MEMORY DISPLAY

To change from the register mode (prompt 'REG>'), type 'M','CR' and the prompt will change to 'MEM>'. To change back to the REG mode type 'R','CR'. In the MEM mode, a continuous block of 64 bytes is displayed, with the address displayed on the left-hand edge every 8 bytes.

The MEM mode allows more substantial memory changes to be made than is easily possible in the REG mode. The 'memory pointer' arrows (see figure 1) can be controlled with the CURSOR keys, and arrows 5 and 6 will follow similarly to arrows 3 and 4 in the REG mode.

The contents of memory are changed as in the REG mode, but this time the memory pointer automatically moves on to the next location, except when a CURSOR key is used instead of 'CR'. To move the memory pointer to an area of memory not currenly displayed, type 'Mn','CR', where 'n' is the required address. If n is greater than 255 (FF hex.) ie. a two-byte number, then 'M' may be ommited. This is usually the case, since the first 255 bytes are ROM. n can also be included when changing from the REG mode to the MEM mode.

COMMAND USE

Having covered the meaning and use of the display, the commands available can be dealt with in more detail.

Most of the commands can be used in both the REG and MEM modes; some however apply to only one mode, and these are left until last. In the examples given, underlined characters are typical user responses, and the prompts should not be typed.

COMMANDS USEABLE IN BOTH MODES

Z ..... Zen — A return is made to ZEN. The new ZEN command 'B' will get back into ZEN DEBUG.

F ..... Fill — Fills a block of memory, as in ZEN.

C ..... Copy — Copies a block of memory as in ZEN.

L(n) .. Locate — This allows a string of up to 40 characters to be searched for in memory. n is the optional address from which the search is made: if omitted the address pointed to by the memory pointer or arrow 2 will be used instead. Each number in turn is entered followed by 'CR', in respose to the 'DATA ' prompt. When the string has been entered, pressing 'CR' with no number will cause the search to commence. If an attempt is made to enter more than 40 characters, a beep will be produced and the fourty enered so far will be used. When a matching string is found, the address of the first byte of the match will be printed and the memory pointer or 'NN' (depending on mode) will be loaded with this address. Only RAM (1000H-CFFFH) will be searched, and the storage buffer at 11CBH may always be found.

eg. suppose the only two occurences of the string 01,02,03 are at 153BH and 1C00H, and a search is required from 1200H.

```
MEM>L1200H'CR' ..... (search from 1200H)
DATA>1'CR' ......... (01=first byte to be found)
DATA>2'CR' ......... (02=second byte to be found)
DATA>3'CR' ......... (03=third byte to be found)
DATA>'CR' .......... (end of data)
135B ............... (first occurence of string)
MEM> ............... (memory pointer now at 153BH)
```

(could also be in REG mode)

N(n) .. Next — This repeats the last Locate performed, using the same set of data. n is as in the Locate command.

eg. suppose that in the above Locate example, the other occurances of the string were required.

```
REG>N'CR' .......... (find next occurence, n not given)
1C00 ............... (next occurence, NN loaded with this)
REG>N'CR' .......... (next occurence again)
11CB ............... (storage buffer found)
REG>
```

(could also be in MEM mode)

V ..... Verify — This verifies (compares) two blocks of memory. START>,STOP>and WITH> are prompted for, and if the two blocks are equal, 'OK' is printed.Otherwise, a beep is produced and the memory pointer or NN is loaded with the first byte address in the second block which differs from the corresponding byte in the first block, ie. the address of the first difference between the two blocks is used. The differing addresses of the two blocks are also printed on the screen.

eg. The block of memory from 1200H is to be compared with the block starting at 4000H.

```
MEM>V'CR' .......... (verify)
START>1200H'CR' .... (first address)
STOP>2000H'CR' ..... (end of first block)
WITH>4000H'CR' ..... (beginning of second block)
OK ................. (if blocks are equal)
```

If the blocks differ at 4200H:

```
MEM>V'CR'
START>1200H'CR'
STOP>2000H'CR'
WITH>4000H'CR'
1400,4200 ..... (addresses of different byte,memory
                 pointer loaded with 4200H)
```

(could also be in REG mode)

T ..... Test

Tests a block of memory. START> and STOP> are prompted for, and a counter is displayed on the screen to indicate the address of the location currently being tested. 'OK' is printed if the memory is good, otherwise the memory pointer or MN is loaded with the faulty address. This address is also printed on the screen, together with the number written into the location and the faulty number read out. For each location in turn, every number from 0 to 255 is written in and read out. If each number is read out correctly, the location is good, the original contents are restored and the next location is tested.

eg. Suppose the block of RAM from 4000 to5000H is to be tested.

```
REG>T'CR' ..... (test)
START>4000H'CR' ... (start of block)
STOP>5000H'CR' .... (end of block)
OK ..... (memory good)
```

If there was a fault at 4601H:

```
REG>T'CR'
START>4000H
STOP>5000H
4601,00,C3 ..... (fault at 4601, 00 written, C3 read)
```

(could also be in MEM mode)

W ..... Work out

This allows arithmetical expressions to be evaluated. DATA> is prompted for, and the calculation entered. Calculations are performed as in ZEN, ie. strictly from left to right and using 16-bit integer numbers. Permitted operators are the usual +,-,*, /,.(logical OR), and &(logical AND). After entering the expression press the '=' key followed by the letter of the base the answer is required in (H for Hex., O for Octal, B for binary or nothing for decimal). After pressing 'CR', the answer will be printed, replacing the letter, in the required base.

eg. To calculate D3C hex.*173 octal + 73 ORed with 110110 binary, with the answer in hex.:

```
REG>W'CR' ..... (work out)
DATA>D3CH*1730+73.110110B=H'CR' ... (expression)
     The 'H' at the end will be replaced with
       '...=5C1FH'..... (answer)
REG>
```

To convert 3C0 hex. to decimal:

```
REG>W'CR'
DATA>3C0H='CR'
     '960' will be printed after the '='
REG>
```

(could also be in MEM mode)

U(n) .. Unassemble

This disassembles from address 'n'. For each instruction, the address of the first byte, the actual bytes constituting the instruction, the ASCII equivalent of the bytes and the standard mnemonics are printed on the screen. The output is in pages on the video screen, and to pause at the end of a page, press 'BREAK'. Then pressing 'Q' or 'SHIFT BREAK' will end disassembly, and any other key will continue.

eg. To disassemble from 132BH:

```
REG>U132BH'CR'..... (disassemble from 132BH)
                        ASCII characters
132B 31 F0 10    ...  LD SP,10F0H
132E DD 21 C7 12 .... LD IX,12C7H
1332 CD CA 13    ...  CALL 13CAH
1335 CD E5 02    ...  CALL 02E5H
   etc.
```

G(n) .. Goto

This jumps to address n as in ZEN, but after a breakpoint is reached, 'BREAK' is printed and control is passed back to ZEN DEBUG.

J(n) .. Jump

This executes a program as in the Goto command, but allows it to be stepped through one instruction at a time. After pressing 'CR' there will be no prompt, just a cursor and a small disassembled listing. Several 'sub-commands' are now available:

V .. Video — This has a 'toggle' action. Normally, ZEN DEBUG updates the display after each step, but pressing 'V' will allow the program being stepped through to use the display. A 'V' will be shown to the right of the flag display, and a second press will return to the previous state.

D .. Disassemble — This also has a toggle action. Normally, the previos one, and the next three, instructions are disassembled and displayed in the same format as in the 'U' command, but pressing 'D' will supress disassembly. A 'D' is displayed to the right of the flag display when the disassembler is activated, and the current instruction has an arrow to the left of the mnemonics.

C .. Call — When the next instruction is a call, this sub-command will cause the subroutine to be CALLed without stepping through it. After RETurning, however, single stepping will resume. Thus a subroutine which is known to be working need not be repeatedly and laboriously stepped through.

J .. Jump — This instruction causes the next program instruction to be skipped, and is useful for exiting a loop before the usual condition has been reached.

R .. Run — Whilst the 'R' key is held down, the program is stepped through quickly, updating the display (depending on the 'V' and 'D' commands).

X .. Xchange This displays the alternate Z80 register set if the 'V' and 'D' commands are set up appropriately as above.

U .. Undo Pressing 'U' will 'undo' the previous instruction, ie. step backwards. Memory contents will not be restored, however, and RETurn instructions cannot be 'undone' because of the way the ZEN 'TRAP' routine works. The main use of the 'U' sub-command is in correcting mistakes, eg. if it was intended to CALL a subroutine using the 'C' sub-command, but instead the instruction was stepped, then pressing 'U' will return to a position where 'C' can be used. Note that only one instruction can be 'undone' at a time.

SHIFT BREAK A return to the usual ZEN DEBUG command loop is made, with 'BREAK' being printed.

ANY OTHER KEY causes the next program instruction to be stepped.

Note that monitor routines will not be stepped through, but stepping will resume afterwards (as in the 'C' sub-command above). The stepping works by setting a breakpoint after each instruction, so the stack pointer must be set before-hand to an area of RAM. (as in the ZEN Goto command). Its value may, of course, be changed during the course of a program. Under exceptional circumstances (eg. when monitor routines do not RETurn), the breakpoint may remain in memory, and will show up as a RST 38H instruction (FF hex.). Should this occure, the correct byte can be found at 12F3H and the correct byte should replace the FF.
When single stepping, it is useful to observe the REG display, specially when any of the registers are being used as pointers to tables etc. The top two items on the stack can also be seen. Note that two areas of memory can be monitored simultaneously when stepping: one 64 byte area using the MEM display, and one 8 byte area using the NN pointer.

eg. The following program will 'white out' the screen. When assembled, it can be viewed through ZEN DEBUG and will look similar to figure 1.

```
        ORG 1000H     ; start of program
        LOAD 1020H
        LD SP,107FH   ; spare RAM
        LD HL,0D000H  ; screen beginning
        LD BC,1000    ; number of screen locations
LOOP:LD (HL),67       ; 67=display code for white square
        INC HL        ; next screen location
        DEC BC        ; decrement count
        LD A,B        ; =0 ?
        OR C
        JR NZ,LOOP    ; continue if not
        JP 0          ; else go to monitor
```

To single step the above example, first ensure that SP is pointing to unused RAM.
Then:
REG>J1000H'CR' .....(step from 1000 hex.)

Pressing 'V' will allow the program's effects to be seen. Holding down the 'R' key will Run the program. Pressing 'V' again will cause the display to be updated after each step, and only the 'gaps' will be filled in. Holding down 'R' will again run the program faster. The 'D','U', and 'X' sub-commands can also be tested at this point. When PC points to the '20' byte (JR NZ,n at 100FH),pressing 'J' will skip the conditional jump, and the next step will go to the monitor. (In the case of a jump to 0 the break-point will not be left in memory).
If the program is stopped by pressing SHIFT BREAK, it can be continued just by pressing 'J','CR', ie. the address need not be specified if it already in PC.
(could also be in MEM mode)

D ..... Displacement The byte pointed to by the memory pointer or NN will be taken and treated as the displacement part of a relative jump (JR n). The memory pointer or NN will then be loaded with the address of the jump destination. The command is particularly useful for finding the destination of relative jumps.

eg. After the example program above (see the Jump command) has been assembled, typing M1010H will move the memory pointer to the displacement byte of the relative jump. To see where the jump would be to, type:
MEM>D'CR' .....(find destination of jump)
MEM> .....(pointer will point to 1009H, ie.LOOP)
(could also be in REG mode)

A ..... Absolute This is similar to the 'D' command above, but two bytes are taken as being an absolute address for a jump or call. The two bytes must be in the usual lower-byte-first order. The command saves having to type 'M' followed by the address in higher-byte-first order.

P ..... Print This will print out the address pointed to b y NN or the memory pointer, and saves counting bytes on the screen to work out the address. If the command is used in the REG mode when arrow 2 is at the far left of the display, then 1200H (beginning of user RAM) will be printed. This also applies to the optional 'n' in some of the above commands.

'SPACE' The function of pressing 'SPACE','CR' depends on the mode.

REG MODE
'SPACE' moves arrow 2 to the byte pointed to b y NN

MEM MODE
'SPACE' moves the position of the byte pointed to by the memory pointer to the centre row of the MEM display,and moves it onto the next location, but unlike the CURSOR key, does not 'stop' at the end of the row.

REG MODE ONLY COMMANDS

X ..... Xchange — As when single stepping, the alternate set of Z80 registers is displayed (ie. HL',DE',BC' and AF'). A second 'X' will display the original set again.
(cannot be used in MEM mode)

K ..... Klear — The contents of all the Z80 registers are cleared (ie.set to 0) except SP, which is set to the 24 byte User Stack area in ZEN (USTK in the ZEN listing). Thus before using the Jump or Goto commands, it is a good idea to clear the registers first.

S ..... Set — This allows the flags to be set by name. DATA> is prompted for, and the flag display is changed accordingly.

eg. To set the carry and zero flags:

```
REG>S'CR'  ..........( set flags)
DATA>CZ'CR'  ......... (Carry and Zero flags now set)
REG
```

The letters used are those used to display the flags (see figure 1).
(cannot be used in MEM mode)

MEM MODE ONLY COMMANDS

+(n) .. add — The memory pointer is moved on to the next 64 byte block of memory, ie. 64 is added to the address of the memory pointer. If n is included, the pointer is moved n blocks of 64 bytes further on.

-(n) .. subtract — Similar to the '+' command above, but subtracts 64 from the memory pointer instead of adding.

EFFECTS OF ZEN DEBUG ON ZEN

As well as adding the extra command 'B' to ZEN, and allowing ZEN to use binary numbers, ZEN DEBUG also allows greater control over the ZEN Print command. Pressing 'BREAK' during printing will pause the listing until another key is pressed. When paused, 'Q' will jump back to ZEN. 'SHIFT BREAK' will stop the listing and exit back to ZEN whether paused or not.

The only other effect of ZEN DEBUG is to move the symbol table and source code higher in memory, to make room for ZEN DEBUG,which resides between the end of ZEN at 2332H and the new beginning of the symbol table at 3731H. ZEN DEBUG is 5K long.

END