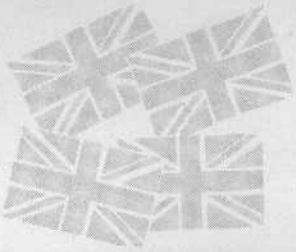
HAND BOOK

British Home



MICROCOMPUTER

NASCOM 1

SOFTWARE NOTES

NASCO
SALES LTD

LYNX Electronics
(LONDON) LTD.

92 BROAD STREET
CHESHAM, BUCKS.

Tel: Chesham (02405) 75151
Telex: 837571

THE OPERATING SYSTEM FOR NASCOM 1

-----NASBUG-----

1.

The personality of any computer system is governed by its operating system. For the Z-80 based NASCOM 1 the Operating System is called Nasbug and has been designed to assist the user to write and De-bug programmes written into the computer's memory in OBJECT CODE. The fundamental base of all Nasbug's operations is HEXADECIMAL. All DATA or CODE is entered in this form and this description assumes that all variables are in HEX.

NASBUG commences action immediately the CPU RESET KEY is pressed. Its initialisation can be read in the listing which should be read in conjunction with this description. The apparent functions to the user are that the screen is cleared and the PROMPT CHARACTER appears in the first location of the lowest line of the SCREEN. The appearance of the PROMPT means that NASBUG is waiting for your command. It also means that the hardware is fully operational and that the REFLEXIVE DATA has been loaded into RAM (See section 11). After the PROMPT the user may enter any of the following NASBUG COMMANDS:

L	Load from Serial Interface
Baaaa	Set Breakpoint
Maaaa	Modify/Inspect Memory
Taaaa bbbb	Tabulate Memory
Daaaa bbbb	Dump Memory to Serial Interface
Caaaa bbbb dddd	Copy Memory from a block starting at (a) to a block starting at (b) The length of each block is (d).
Eaaaa	Commence Execution at a (a can be implied).
Saaaa	Single Step Execution at a (a can be implied).

NOTE that there should be no space between the COMMAND and the first ADDRESS. All COMMANDS can be EDITED.

/ 1. (cont'd)

NOTE that the lower case letters above are the ARGUMENTS of the COMMAND and are always ADDRESSES. The COMMAND is taken from the screen and can be EDITED by use of the BACK-SPACE KEY. (No action is taken until the NEW LINE KEY is pressed - this applies to ALL commands).

NOTE that LEADING ADDRESSES ZEROS are NOT required. Leading DATA zeros are NOT required. In all following sections, the characters written by the USER are UNDERLINED.

2.

Maaaa

To either examine or change the contents of READ-WRITE MEMORY, the M command is used followed immediately by an address.

The user can then enter a series of data bytes which will be written into successive locations. The data must not flow into the next line i.e. before it does so a NEW LINE must be entered. Each data byte must be separated from the next by a space.

Characters entered by the USER are underlined.

The letters NL serve to represent the pressing of the NEWLINE KEY by the USER. The following example shows a programme being entered by the user. The places at which the user has pressed NEWLINE are entirely by personal preference, as long as the data string does not ever flow into the next line since NASBUG uses only those characters which appear to the right of the PROMPT on the bottom line of the screen.

```
>MCFA NL
0CFA 00>3A 00 0E NL
0CFD 00>3C NL
0CFE 00>32 00 0E NL
0D01 00>CD 3B 01 CD 35 00 NL
0D07 00>C3 FA 0C NL
0D0A 00>. NL
>
```

The M command will continue until aborted by use of the . (full stop) key plus NL.

3.

Taaaa bbbb

The Tabulate Command is to display a block of locations onto the screen. There is little point in calling for a block longer than 68H. (NOTE this represents 68 as a HEXADECIMAL VALUE and can either by 68H OR sometimes H'68'. The DECIMAL equivalent is 6 x 16 + 8 = 104D = D'104' = H '68' = 68H = 16'68. If a longer block is called for it will overwrite the screen and only the last H'68' locations will be visible at the end.

```
>TCFA D0A NL
0CFA 3A 00 0E 3C 32 00 0E CD
0D02 3E 01 CD 35 00 C3 FA 0C
.
>
```

4.

Daaaa bbbb

THE DUMP COMMAND uses the same code in the operating system as TABULATE thus its data format is identical. What is not apparent in the T command is that each line of text on the screen is not as it was originally written. The line is initially generated as below which is the format for recording on to cassette or TTY. The D command causes the same format line as for T to be transmitted not to the screen but to the TTY (or SERIAL OUT) only and precedes the start of transmission with the operation of the 'START PERIPHERAL DRIVE' Light Emitting Diode (LED) which the user should take as their cue to start the cassette motor running. In practice the motor can be operated a while before since no false characters are transmitted ahead of the main data. The end of the block is identical to that for T except that the System turns the LED off just prior to presenting the PROMPT for the next command.

The format below is broken after the user's NEW LINE to show the cassette characters which do not appear on the screen. The duration of the DUMP command is a function of the UART CLOCK and is about 25 characters/second for cassette.

```
>DCFA D0A NL
>

0CFA 3A 00 0E 3C 32 00 0E CD | 97 BS BS NL
0D02 3B 01 CD 35 00 C3 FA 0C | 16 BS BS NL
.
>
```

5.

Caaaa bbbb dddd

The COPY COMMAND copies the contents of a block of memory starting at location a and continuing until location a + d into the block starting at b, thus the last byte to be copied will be from a + d to b + d.

NOTE that the COPY COMMAND MUST BE USED WITH GREAT CARE. Since the data is transferred immediately, data in the overlapping regions of blocks which overlap is destroyed. Thus to copy successfully the user must observe:-

b must be greater than a + d

DOWN COPYING can be effected by using an intermediate data area to transfer the data to in order to copy down. As long as this block is clear of a + d the command can be used again to transfer back to the overlap region. This procedure is demonstrated in the following example to 'open up' a block of code to insert a byte. The user has omitted to put in a byte at address H'D05 and the last address of the data is at H'D37'. The COPY command is therefore used to transfer the block H'D05' to H'D37' inclusive to an unused area at H'F00'. This block is then transferred back to a block starting at H'D06'. The C command can be used to set whole areas of memory to any code by executing the copy command with the address of the first block being one address less than the address of the second. The third argument is the number of bytes to be set to the value of the contents of the first byte.

```
>TE00 E0F NL
0E00 01 02 03 04 05 06 07 08
0E08 09 0A 0B 0C 0D 0E 0F 10
>

>CE05 F00 32 NL
>CF00 E06 32 NL
>

>TE00 E0F NL
0E00 01 02 03 04 05 06 06 07
0E08 08 09 0A 0B 0C 0D 0E 0F
>
```

6.

E aaaa

The EXECUTE COMMAND transfers the contents of the REGISTER SAVE AREA into the internal registers of the Z-80 and finally forces PC to aaaa. EXCEPT that after encountering a BREAKPOINT (see section 8) 'a' need not be specified since, if it is not, the last executed value for PC will be put into PC by the E Command. This also applies to the S COMMAND (see Section 7).

```
>ME00 NL
0E00 00>20 NL
0E01 00>. NL
>ECFA NL
!"#$%&'()*+,- (ETC.)
```

NOTE that the first instruction initiates the SINGLE STEP procedure which is then ignored and execution continues normally. Thus, the first instruction to be executed will receive a NON-MASKABLE INTERRUPT during its execution. All subsequent instructions will not. This means that the HALT instruction will not be obeyed if it is at the address of the current E command where specified or implied.

This shows the Z-80 executing the HALT (at H'0F03') but being interrupted out of it by the NMI trap laid for it by NASBUG. The next sequence shows the Z-80 halted and this state is indicated by the HALT LED on the NASCOM I card.

```
>MF00 NL
0F00 00>0 0 0 76 0 C3 86 2 NL
0F08 00>. NL
>EF03 NL
>                (BACK IN MONITOR LOOP)
>EF02 NL
_                (HALTED).
```

7.

S aaaa

As with the EXECUTE command the 'a' may be implied, BUT SO MAY 'S'. This command is one of the most useful commands available for checking correct program execution. It is so useful that the NEW LINE KEY has been programmed to extend its function into not only permitting the SINGLE STEP ADDRESS to be implied (as being simply the next instruction), but the S CHARACTER ITSELF is implied, thus the executions of the SINGLE STEP COMMAND can be continued once initiated by means of only the NEW LINE character. Thus since the operation of the SINGLE-STEP proceeds by means of the NMI input (i.e. single-stepping is partially hardware driven) the S command can be used to examine ROM code i.e. NASBUG will step through ROM code.

The use of the implied address of both E and S is illustrated in the following example:-

```
>B0 NL
>E0 NL
>BD04 NL
>E0 NL
>SCFA NL
1000 0CFD 2042 FFCF FF00 0600
>S NL
1000 0CFE 2120 FFCF FF00 0600
> NL
1000 0D01 2120 FFCF FF00 0600
>ME00 NL
0E00 21>. NL
>S NL
0FFE 013B 2120 FFCF FF00 0600
>E NL
!1000 0D04 2120 FFCF FF00 0600
>E NL
"1000 0D04 2220 FFCF FF00 0600
>E NL
#1000 0D04 2320 FFCF FF00 0600
>B0 NL
>E NL
$%&'()*+,-
                (ETC)
```

S (cont'd)

At every execution of SINGLE STEP, the following registers internal to the Z-80 are put up onto the SCREEN:-

SP	PC	AF	HL	DE	BC
Stack Pointer	Program Counter	Accumulator and Flags	Register Pair HL	Register Pair DE	Register Pair BC

8.

B aaaa

The BREAKPOINT COMMAND should be entered at location 0 whenever NASCOM I is first switched on since the BREAKPOINT ADDRESS is saved in memory. The RESET button does not clear this location otherwise the place would be lost if the RESET was used to return to NASBUG. Since the processor has no other means than RESET to know if it is first powered up or not, this setting of a breakpoint at 0 should become a user habit on first turning on the machine.

The use of the BREAKPOINT is illustrated in Section 7. The purpose of the BREAKPOINT is to insert a trap sequence into the code in RAM at the address specified after the B. This puts H'E7' into aaaa and removes the byte which existed there into address 0C17. The program is then executed until it finds the BREAKPOINT and then execution is transferred to NASBUG at location 0020. This causes the current registers to be put up onto the screen in the same way as for SINGLE STEP. The original code is restored by entering BO i.e. by setting a new breakpoint at location 0.

After setting a breakpoint the next instruction MUST be an Execute instruction - always follow "B" with "E". (See previous Example.)

9. L

The LOAD command executes a routine which interprets data on the last line of the SCREEN as data in the format of the DUMP command but to be loaded into the memory at the address specified at the beginning of the line. The process is the exact reverse of the DUMP (or TABULATE) code but with the following exception: If the check-sum byte is not true, the line is not loaded into the memory but is SCROLLED on to the next line on the SCREEN. Thus at the end of the LOAD (which occurs when the end-of-DUMP characters are recognised) the SCREEN contains only invalid lines which have not been loaded due to tape or recorder errors. By re-winding the tape, the errors can usually be corrected by re-loading If a note is taken of faulty lines and the recurrance of the same line is observed only then will it be necessary either to load again or to correct the line using the M command.

10.

Software implications of NASBUG.

The single step feature occupies the NMI line the processor so that this input and the associated instructions are not available to the user, without modifying the hardware. NASBUG has been written to be maximally usable by the user. All commands are in the form of sub-routines which can be called by the user software.

The COMMAND TABLE is arrived at reflexively so that its table of commands can be extended i.e. users can write their own commands and use them in conjunction with NASBUG.

The KEYBOARD TABLE is called reflexively so that it can be re-written in RAM to re-assign values to each key, or to increase the number of keys used.

(NOTE the LICON KBD uses a special circuit for key operation and so additional keys need additional hardware to access them).

11.

Sub-routines in NASBUG which can be called by the user are:-

KDEL	at	0035	Gives a 6ms delay
MOTFLP		0051	Turns cassette motor lamp/drive on/off (alternate calls)
SRLOUT		005D	Puts A into UART
KBD		0069	Returns with Carry SET if a character was found from the Keyboard. Character in A.

Using reflections:-

$KTAB This address in RAM has the address of the new keytable.

(NOTE that both CRT and KBD are entered by NASBUG via a JUMP in RAM. This means that the user can alter the locations at $CRT and $KBD to cause NASBUG to look elsewhere (in user RAM) for both these subroutines).

CRT	013B	A character in A is put onto the SCREEN or CURSOR moved (BS or CR) or SCREEN is CLEARED (FF).

There is a reflexive jump at $CTAB which gives NASBUG the start address of its own COMMAND TABLE. This means that the user can add COMMANDS by changing this jump. The existing commands can all be incorporated into this table.

CHIN	003E	This routine calls $KBD and also looks at the UART to see if any SERIAL DATA is available. If either occurs, the routine returns with Carry set and the character in A.

$NMI is the address which the NMI uses reflexively. It can be changed.

12.

The example program illustrated in Section 7 shows the use of some of the sub-routines available in NASBUG. A listing of the program would be (in SOURCE STATEMENTS):

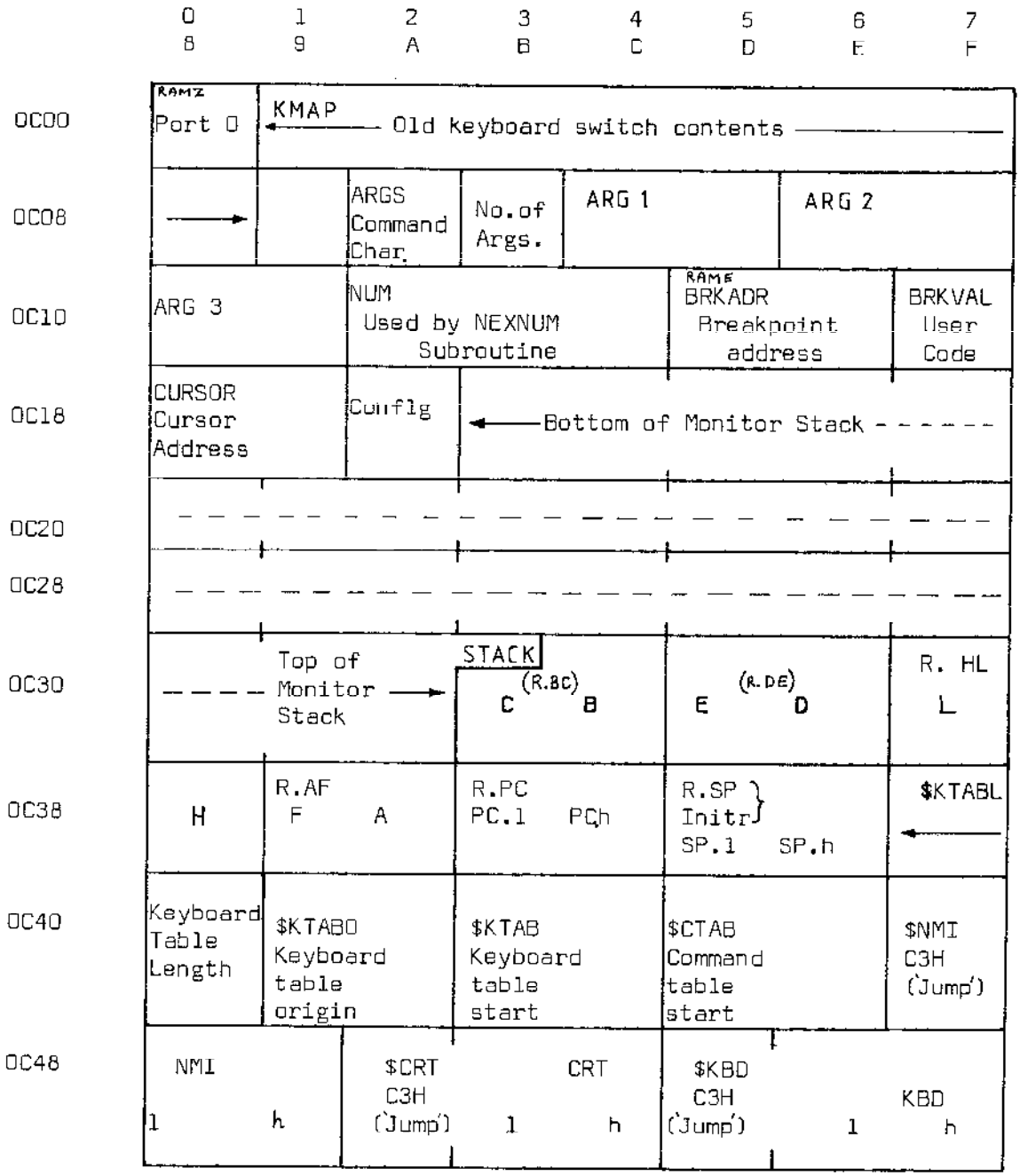
```
            ORG   H'OCFA'      Start code at CFAH
CRT         EQU   H'013B'
KDEL        EQU   H'0035'
REG         EQU   H'0E00'
LOOP        LD    A,REG
            INC   A
            LD    REG,A
            CALL  CRT          Go to the subroutine at 13BH
            CALL  KDEL         Go to the subroutine at 35H
            JP    LOOP         Return to the first instruction
```

Only the statements from LOOP onwards are actually coded, the code being as entered in the example of section 2.

13.

The memory locations available for user programming in the unexpanded Nascom 1 extend from ØC5Ø to approximately ØFEØ (depending on the use made of the user's stack).

14: DETAILS OF MEMORY USED BY NASBUG



MEMORY USED BY NASBUG

(TO GENERATE THE CORRESPONDING MAP ON-SCREEN TYPE:-

T0D0 0C4F NL .)

(THE FIRST LOCATION AVAILABLE FOR USER PROGRAMMING IS 0C50.)

PART 17: NASBUG MONITOR PROGRAMME

SECTION A: NOTES ON Z-80 MONITOR

000 START: initialise SP to C33
clear RAMZ to RAME-1(C00 to C14)
initialise relfections (C3D to C4F)
clear CRT screen
go to:

359 STRTO: remove breakpoint
go to:
PARSE

---

286 PARSE: read a line
(2nd last line on screen is now
the line)
If blank, then if last instruction
was 'S' then insert 'S' (single key single step)
Save command char at ARGS (C0A)

29E PLOOP: Call NEXNUM to get argument,
if there is one, put at ARG1...,
go to PLOOP:
Set number of arguments in ARGS+1.
Search CTAB for Command character.
If not found go to PARSE.
Else push PARSE on stack, Jump
to subroutine. (Faking a CALL)

2D0 EXEC: OR 2FF STEP:
↓ ↓
Set CONFLG=-1 Set CONFLG=0
↘ ↙

EXEC1: Re-initialise NMI reflection
(in case user has changed it)
Throw away return to PARSE

If argument supplied, put in R.PC

Restore BC, DE, AF (First AF is old HL)

Restore user SP

Push user PC on stack

Restore HL

Save AF while activating NMI

RETN to user's instruction

User instruction
↓
NMI
↓
TRAP

305 TRAP: Add 1 to users PC on stack

(Decremented later in code common

to TRAP & Breakpoint)

Save AF, HL;

Clear NMI flag in port 0

If CONFLG nonzero (i.e. was an 'E')

Save users instruction at breakpoint,

Insert RST 4 there.

Restore HL, AF, decrement PC,

RETN to user's code.

Otherwise

Save DE and go to

BPTI (326)

Breakpoint entry 020:

Save AF, HL, DE

then:

326 BPTI: Save BC; HL← SP;

Copy registers from User stack

to register save area in Monitor's RAM

```
                        Decrement user's PC so it points to

                        Breakpoint address.

                        print out users registers

347  REGSI:

359  STRTU:             restore instruction at breakpoint

                        and go to
                        PARSE

069   KBD:              Save regs.
(via $KBD)
                        Clear counter

                        Initialise pointer to map

                        Read Row 0  (shift)

                        into KMAP

     KSC1:              Increment counter.

                        Increment KMAP pointer, check for change.

                        If different --> KSC2

     KSCIA:             Repeat 8 times

     KSC 8:             Clear carry (no key found)

     KSC 9:             Restore regs - return

     KSC 2:             delay.  (KDEL - 035)

                        read again.

                        Calculate column no. of changed bit, (C)

                        bit mask  (D).

                        Check whether change really occurred

                        if not go to KSC 1A

                        Update map

                        If a release go to KSC 1A

                        Manufacture magic number from

                        shift key, loop counter, bit number

                        Search KTAB for this number. (via $KTAB)
```

```
                        (If not found, clear shift bit & try again)
                        If still not found go to KSC8
                        Calculate ASCII code from address within
                        KTAB  (0EA)
0E0   KSC3:             set carry (character found), return.

-------------------------------------------------------------------

03E   CHIN:             check keyboard & UART until
                        character received.

-------------------------------------------------------------------

05D   SRLOUT:           put character in UART
                        wait until sent.

-------------------------------------------------------------------

053   FLIP:             flip a bit in Port 0

-------------------------------------------------------------------

04A   FLPFLP            flip & flop a bit in Port 0

-------------------------------------------------------------------

035   KDEL:             delay;  PUSH & POPs  lengthen the loop

-------------------------------------------------------------------

13B   CRT:              ignore character 0.
                        Save regs.
                        FF? yes.  put -1 in top left,
                                then 48 spaces
                                then 16 zeros   }
                                                } repeated 15 times
                                then 48 spaces  }
                                then -1 in bottom right
      CRT0:             set HL to bottom left
      CRT1:             put cursor on screen, save cursor
      CRT2:             restore regs, return
                        Replace cursor with blank.
                        BS? ---> decrement cursor, skip
                                 over margins.
                                 If -1 reached, increment cursor
                                 again;  go to CRT1 to put
```

```
                                cursor on screen & return.
                        CR? → CRT3  (scroll)
                        Ordinary character :  Put it on screen.
                             Skip cursor over margins
                             -1 reached?  no → return via CRT1
195     CRT3:           Scroll;  clear bottom line and go to
                        CRT0 to reset cursor.
1DB     INLINE:         print prompt
1DE     INL0:           get character;  BS? → INL2;
                        CR? → return via CRLF
1E9     INL1:           display character;  → INL0
1EE     INL2:           BS?
                        Check for prompt, if so go to INL0
                        otherwise go to INL1 to do backspace.
-------------------------------------------------------------------
                        (RST 5)
028     PRS:            Pick up 'return address'.
                        Send characters until 0 to CRT
                        Return to instruction after 0.
-------------------------------------------------------------------
224     B2HEX:          save number, shift top hex to bottom.
                        Call B2HEX1 (to print it)
                        restore number
24D     B2HEX1:         print bottom hex digit, return.
-------------------------------------------------------------------
25A     NEXNUM          DE points to line on screen.
                        Skip blanks
                        NUM = number of digits in number.
                        NUM+1, NUM+2 ← number.
-------------------------------------------------------------------
1AD     MODIFY          Pick up address from ARG1
1B0     MOD1:           Print address,
                        Print contents.
```

```
                        Call inline to read line.
1BF   MOD2:             Number there?→ modify memory,
                                       increment address, → MOD2
ICF   MOD3:             '.'  ? go to return.
                        Any numbers on line?  no,→ increment address then
                        go to MOD1
```

```
IFC   TABCDE:           address = (ARG1)
      TBCD 1:           address = ARG2?
                                  Yes ⟶ print '.'; return.
                        Clear checksum.
                        Print address (TBCD3)
216   TBCD1A:           Print memory (TBCD2, accumulates checksum)
                        Increment address
                        Repeat 8 times from 1A:
                        Print checksum.
                        Backspace over checksum then go to
                        TBCD 1 :
22B   TBCD2:           Accumulate checksum, print in hex.
232   TBCD3:           Print address using TBCD2.
```

```
3D1   DUMP:             Turn on Motor then call
                        Delay.
                        Set $CRT to SRLOUT
                        Call TABCDE
                        Restore $CRT to CRT
                        Return via routine to turn motor off.
```

37C	LOAD:	turn on motor
	LOD 1:	set to beginning of line
	LOD 1B:	get character, ignore BS
		Not CR? go to display, then back to LOD1B
	LOD 1A:	Pointer to beginning of line
		'.' ? to MOTFLP to turn off & return
		Read numbers into top left screen margin.
		Check checksum. If incorrect, scroll, then go to LOD1
		then copy into memory ⟶ and return to LOD1.

3EF	COPY	Block transfer from the block starting
		ARG1 to the block starting ARG2 the
		number of bytes in ARG3.

HEXASM NOTES

Numbers are normally in decimal.
Numbers in other bases are specified by:
   base'number e.g. 16'FF = 2'11111111 = 255

'*' is the 'bit' operator, eg *7 = 16'80, *0 = 1

'A gives the ASCII code for A, i.e. 65

Angle brackets < > are used as brackets within expressions

the statement FRED=3; equates the symbol 'FRED' with 3

'.' is the location counter, e.g.
   .=3 sets the location counter to 3 (ORG 3)

RAM. specifies the start of RAM
.ROM; .RAM; switch between ROM and RAM

';' is used to separate statements

```
0000 010203          .WORD  1,2,3;   [ assembles 3 words containing 1, 2 & 3]
0003 FFFF0100        .ADDR  -1,1;    [ assembles as 2 double-words (low order first)
0007                 .BLKW  3;       [ reserves 3 words as RAM
000A                 .BLKA  2;       [ reserves 2 double words

000E 78              A~B;            [ is equivalent to  LD A,B
000F 3E03            A~#3;           [ is equivalent to  LD A,3
0011 3A0700          A~FRED;         [ is equivalent to  LD A,(FRED)
```

load's which have no direct Z-80 equivalent are treated as two instructions;
the first loads the accumulator, the second storin the value. e.g.

```
0014 3E053207003E0532 FRED~#5;       [=]          A~#5; FRED~A;
```

other shorthand's are:

```
001E B7B7            TSTA;  [=]     OR A;
0020 AFAF            CLA;   [=]     XOR A;
```

the following opcodes have different names in the Z-80 manual:

```
CMA;      =     CPL
DBNZ LABEL; =   DJNZ LABEL;
J LABEL;  =     JP LABEL;
BR LABEL; =     JR LABEL;
```

```
                                operations which normally take one argument can be given
                                more than one, with the expected result.    e.g.:

0022 F5C5D5F5C5D5       PUSH AF,BC,DE;      [=]  PUSH AF; PUSH BC; PUSH DE;


                                the code:

0028 20033A0700                 IF Z; A~FRED; FI;

                                is equivalent to:

002D 20033A0700                 BR NZ lab1; A~FRED;
0032                    lab1:



                                the code:

0032 2005                       IF Z;
0034 3A0700                      A~FRED;
0037 1803                       ELSE;
0039 3A0600                      A~JIM;
003C                            FI;

                                is equivalent to

003C 2005                       BR NZ lab2;
003E 3A0700                      A~FRED;
0041 1803                        BR lab3;
0043 3A0600             lab2:    A~JIM;
0046                    lab3:


                                in  IF FRED Z; the code A~FRED; TSTA; is inserted before the first branch.
```

```
HEXASM V005 ASSEMBLY ON 15-FEB-78 AT 21:33.      PAGE 3        ,LP:<CSDOC
DK:CSDOC.SRC     HEXASM NOTES

                         JIM=6
                         FRED=7
  0046                   .END
```

```
HEXASM V005 ASSEMBLY ON 15-FEB-78 AT 21:33.      PAGE 4        ,LP:<CSDOC
SYMBOL TABLE

FRED   =0007
JIM    =0006
LAB1   :0032
LAB2   :0043
LAB3   :0046
RAMTOP:0000
RAM.   =0000
ROMTOP:0046
ROM.   =0000
       =0046
```

```
                        RAM.=16'C00

                          initialise stack pointer and RAM

0000 31330C             START:  SP_#STACK
0003 21000C0615                 HL_#RAMZ; B_#RAME-RAMZ;
0008 36002310FB                 (HL)_#0; INC HL; DBNZ .;
                                set reflections
000D 212801113D0C               HL_#INITT; DE_#INITR;
0013 011300EDB0                 BC_#INITE-INITT; LDIR;
                                initialise crt
0018 3E1ECD3B01                 A_#FF; CALL CRT;

001D C35903                     J STRT0;

                         BREAKPOINT RESTART
0020 F5E5D5C32603               PUSH AF,HL,DE; J BPT1;
0026 00                         NOP;
0027 00                         NOP;

                         RST 5 = PRINT FOLLOWING STRING, TERMINATED BY 00

0028 E3                 PRS:    EX (SP),HL;
0029 7E23B7             PRS1:   A_(HL); INC HL; TSTA;
002C 2805CD4A0C18F6             IF NZ; CALL $CRT; BR PRS1; FI;
0033 E3C9                       EX (SP),HL; RET;

                            keyboard debounce delay routine
0035 AF                 KDEL:   CLA;
0036 F5F1F5F13D20F9C9           PUSH AF; POP AF; PUSH AF; POP AF; DEC A; BR NZ .; RET;

                             read a char from keyboard or uart (first come first served)
003E CD4D0CD8           CHIN:   CALL $KBD; RET CS;
0042 DB0217DB01D8               IN A,2; RLA; IN A,1; RET CS;
0048 18F4                       BR CHIN;

                                set & reset a bit in I/O port 0
004A F5CD5300F11802     FLPFLP: PUSH AF; CALL FLIP; POP AF; BR FLIP;

                                start or stop motor
0051 3E10               MOTFLP: A_#*4;
                                flip a bit in port 0
0053 E521000CAE         FLIP:   PUSH HL; HL_#PORT0; XOR (HL);
0058 D30077E1C9                 OUT 0,A; (HL)_A; POP HL; RET;

                            put character out thru UART, and wait till sent
005D D301               SRLOUT: OUT 1,A;
005F DB0287F818FA               IN A,2; ADD A; RET M; BR .;

0065 00                         NOP; [padding]

                          NMI VECTOR
0066 C3470C                     J $NMI;
```

```
                        routine to read from keyboard
                              carry is set if a char. is available
                              the standard ASCII code for the char is returned in A
                              EXCEPT FOR the following chars
                                      BS=16'1D   backspace
                                      CR=16'1E   carriage return (=newline)
                                      FF=16'1F   form feed =clear screen

0069 C5D5E5             KBD:    PUSH BC,DE,HL;
006C 3E02CD4A00                 A_#*1; CALL FLPFLP;
0071 21010CDB002F77             HL_#KMAP; IN A,0; CMA; (HL)_A;
0078 0608                       B_#8;
007A 3E01CD4A00         KSC1:   A_#*0; CALL FLPFLP;
007F 23DB002F57AE2007           INC HL; IN A,0; CMA; D_A; XOR (HL); BR NZ KSC2;
0087 10F1               KSC1A:  DBNZ KSC1;
0089 B7                 KSC8:   TSTA;
008A E1D1C1C9           KSC9:   POP HL,DE,BC; RET;
008E CD3500             KSC2:   CALL KDEL;
0091 DB002F5F7AAE               IN A,0; CMA; E_A; A_D; XOR (HL);
0097 0EFF160037                 C_#-1; D_#0; STC;
009C CB120C1F30FA               RL D; INC C; RRA; BR NC .;
00A2 7AA35F                     A_D; AND E; E_A;
00A5 7EA2BB28DD                 A_(HL); AND D; CMP E; BR Z KSC1A;
00AA 7EAA77                     A_(HL); XOR D; (HL)_A;
00AD 7BB728D6                   A_E; TSTA; BR Z KSC1A;
00B1 3A010CE610B0               A_KMAP; AND #*4; OR B;
00B7 878787B1                   ADD A; ADD A; ADD A; OR C;
00BB ED4B3F0C2A430CEDB1         BC_$KTABL; HL_$KTAB; CPIR;
                            check again for unshifted character
00C4 280B                       IF NZ;
00C6 2A430CED4B3F0C              HL_$KTAB; BC_$KTABL;
00CD E67FEDB1                    AND #16'7F; CPIR;
00D1                            FI;
00D1 20B6                       BR NZ KSC8;
00D3 ED4B430C37ED42             BC_$KTAB; STC; SBC HL,BC;
00DA ED4B410C097D               BC_$KTABO; ADD HL,BC; A_L;
00E0 3718A7             KSC3:   STC; BR KSC9;

                            set breakpoint address
00E3 2A0C0C22150CC9     BREAK:  HL_ARG1; BRKADR_HL; RET;
```

```
                        table entries represent key number for each ASCII code
                            appearing in ASCII order starting at code 16'1D
                            Each entry is in the format SRRRRCCC
                            where S=1 implies that shift key must be down
                            RRRR=8-row number (number in counter)
                            CCC=column number (bit number)
                        Setting all ones (16'FF) implies that there is no key
                            for this code
                        If the shift key is down and no code is found,
                            then the table is searched again as if
                            the shift key were up.

00EA 088809             KTAB:   .WORD 16'08,16'88,16'09;                        [   BS,  FF,  CR
00ED 149C9BA392C2BAB2           .WORD 16'14,16'9C,16'9B,16'A3,16'92,16'C2,16'BA,16'B2; [ SPACE -   '
00F5 AAA298A0290A2119           .WORD 16'AA,16'A2,16'98,16'A0,16'29,16'0A,16'21,16'19; [   (   -   /
00FD 1A1C1B2312423A32           .WORD 16'1A,16'1C,16'1B,16'23,16'12,16'42,16'3A,16'32; [   0   -   7
0105 2A221820B18AB999           .WORD 16'2A,16'22,16'18,16'20,16'B1,16'8A,16'B9,16'99; [   8   -   ?
010D 0D2C41133B334310           .WORD 16'0D,16'2C,16'41,16'13,16'3B,16'33,16'43,16'10; [   @   -   G
0115 402D38302831 3925          .WORD 16'40,16'2D,16'38,16'30,16'28,16'31,16'39,16'25; [   H   -   O
011D 1D2415344535112B           .WORD 16'1D,16'24,16'15,16'34,16'45,16'35,16'11,16'2B; [   P   -   W
0125 443D3C                     .WORD 16'44,16'3D,16'3C;                        [   X,  Y,  Z
```

```
                        RAM workspace area

0128                    .RAM
0C00                    RAMZ:       [ this part cleared on RESET ]
0C00                    PORT0:  .BLKW;  [ copy of output port 0 ]
0C01                    KMAP: .BLKW 9; [ keyboard switch state table ]
                          argument list set up by PARSE ]
0C0A                    ARGS:   .BLKW 2;
0C0C                    ARG1: .BLKA;
0C0E                    ARG2: .BLKA;
0C10                    ARG3: .BLKA;
0C12                    NUM: .BLKW 3,

0C15                    RAME: [ end of cleared RAM ]
0C15                    BRKADR: .BLKA; BRKVAL: .BLKW;
0C18                    CURSOR: .BLKA;  [CRT CURSOR ADDRESS
0C1A                    CONFLG: .BLKW;
0C1B                            .BLKA 12; [Monitor's stack space]
0C33                    STACK: .BLKA;
0C35                            .BLKA;
0C37                    R.HL:   .BLKA;
0C39                    R.AF:   .BLKA;
0C3B                    R.PC:   .BLKA;
0C3D                    INITR:
0C3D                    R.SP:   .BLKA;
                          reflections
0C3F                    $KTABL: .BLKA; [ reflected length of KTAB]
0C41                    $KTABO: .BLKA; [ reflected offset to first character in KTAB]
0C43                    $KTAB: .BLKA;
0C45                    $CTAB: .BLKA;
0C47                    $NMI: .BLKW 3
0C4A                    $CRT: .BLKW 3;
0C4D                    $KBD: .BLKW 3;

0C50                    .ROM;

                          reflection initialisation table

0128 0010               INITT:  .ADDR 16'1000; [END OF RAM
012A 3E00                       .ADDR 64+3-5; [$KTABL]
012C 1D00                       .ADDR 32-3; [$KTABO]
012E EA006303                   .ADDR KTAB,CTAB;
0132 C30503                     J TRAP;
0135 C33B01                     J CRT
0138 C36900                     J KBD
013B                    INITE:
```

```
                        CRTRAM=16'800; [ CRT ram addr ]
                        CUR='_; [cursor character = underline]
                        BL=32; [space]
                        CR=31; [NEWLINE]
                        FF=30; [SHIFT+8$= CLEAR SCREEN]
                        BS=29; [BACKSPACE ]

                        CURLIN=CRTRAM+10+(14*64); [current CRT line]
                        LINE=CURLIN-64; [start of previous line]
                            Routine puts a char on screen
                                margins of screen contain zeroes except for top left
                                and bottom right which contain -1.
                                FF  initialses screen and puts cursor on bottom line
                                BS  backspaces
                                CR  carriage returns and line feeds

 013B B7C8              CRT:    TSTA; RET Z; [ignore nulls]
 013D F5C5D5E5                  PUSH AF,BC,DE,HL;
 0141 FE1E                      CMP #FF;
 0143 202F                      IF Z;
                                     [initalise screen]
 0145 21090836FF                 HL_#CRTRAM+9; (HL)_#-1;
 014A 230630                     INC HL; B_#48;
 014D 36202310FB                 (HL)_#BL; INC HL; DBNZ .;
 0152 0610                       B_#16;
 0154 36002310FB                 (HL)_#0; INC HL; DBNZ .;
 0159 EB210A08                   EX DE,HL; HL_#CRTRAM+10;
 015D 01B003EDB0                 BC_#15*64-16; LDIR;
 0162 3EFF32BA0B                CRTRAM+(14*64)+58_#-1
 0167 218A0B            CRT0:   HL_#CURLIN;
 016A 365F22180C        CRT1:   (HL)_#CUR; CURSOR_HL;
 016F E1D1C1F1C9        CRT2:   POP HL,DE,BC,AF; RET;
 0174                           FI;
                                [remove cursor]
 0174 2A180C3620                HL_CURSOR; (HL)_#' ' ;
 0179 FE1D                      CMP #BS;
 017B 200B                      IF Z;
                                  [ backspace (thru margins if necessary)]
 017D 2B7EB728FB                 DEC HL; A_(HL); TSTA; BR Z .;
 0182 3C20E5                     INC A; BR NZ CRT1;
 0185 2318E2                     INC HL; BR CRT1;
 0188                           FI;
 0188 FE1F2809                  CMP #CR; BR Z CRT3;
                                   [ put char on screen, scroll if necessary]
 018C 77                        (HL)_A;
 018D 237EB728FB                INC HL; A_(HL); TSTA; BR Z .;
 0192 3C20D5                    INC A; BR NZ CRT1;
                                   [ scroll]
 0195 110A08214A08      CRT3:   DE_#CRTRAM+10; HL_#CRTRAM+10+64;
 019B 017003EDB0                BC_#14*64-16; LDIR;
 01A0 211000                    HL_#16;
 01A3 190630                    ADD HL,DE; B_#48;
 01A6 36202310FB                (HL)_#BL; INC HL; DBNZ .;
 01AB 18BA                      BR CRT0;
```

```
                        memory modify, arg1=address

01AD 2A0C0C         MODIFY: HL_ARG1;
01B0 CD3202         MOD1:   CALL TBCD3,
01B3 7ECD4402               A_(HL); CALL B2HEX;
01B7 CDDB0111520B0600       CALL INLINE; DE_#LINE+8; B_#0;
                              note that line starts at LINE+8
01BF E5CD5A02       MOD2:   PUSH HL; CALL NEXNUM;
01C3 7EB72808               A_(HL); TSTA; BR Z MOD3;
01C7 237EE17704             INC HL; A_(HL); POP HL; (HL)_A; INC B;
01CC 2318F0                 INC HL; BR MOD2;
01CF E11AFE2EC8     MOD3:   POP HL; A_(DE); CMP #'.; RET Z;
01D4 78B72001 2318D5        IF B Z; INC HL; FI; BR MOD1;

                        print system prompt and read a line

01DB EF3E00         INLINE: RST 5; ">";
01DE CD3E00FE1D2809 INL0:   CALL CHIN; CMP #BS; BR Z INL2;
                              return on CR
01E5 FE1F2857               CMP #CR; BR Z CRLF;
                              put out char and continue
01E9 CD4A0C18F0     INL1:   CALL $CRT; BR INL0;
                              handle backspace; dont allow backspace over prompt
01EE ED5B180C1B1AFE3E INL2: DE_CURSOR; DEC DE; A_(DE); CMP #'>;
01F6 28E63E1D18ED           BR Z INL0; A_#BS; BR INL1;

                        tabulate code. ARG1=start addr, ARG2=end
                            routine is used by Dump command

01FC 2A0C0C         TABCDE: HL_ARG1;
01FF ED5B0E0CE5B7ED52 TBCD1: DE_ARG2; PUSH HL; TSTA; SBC HL,DE;
0207 E13805EF2E1F00C9       POP HL; IF CC; RST 5; ".",CR; RET; FI;
020F 0E00CD3202 0608        C_#0; CALL TBCD3; B_#8;
0216 7ECD2B0223     TBCD1A: A_(HL); CALL TBCD2; INC HL;
021B CD3C0210F6             CALL SPACE; DBNZ TBCD1A;
                        put out checksum and backspace over it so it doesnt show
0220 79CD4402               A_C; CALL B2HEX;
0224 EF1D1D1F00             RST 5; .WORD BS,BS,CR,0;
0229 18D4                   BR TBCD1;
022B 57814F7AC34402 TBCD2:  D_A; ADD C; C_A; A_D; J B2HEX;
0232 7CCD2B02       TBCD3:  A_H; CALL TBCD2;
0236 7DCD2B021800           A_L; CALL TBCD2; BR SPACE;

023C 3E201817       SPACE:  A_#' ; BR JCRT;
0240 3E1F1813       CRLF:   A_#CR; BR JCRT;

                        print A in hex

0244 F51F1F1F1F     B2HEX:  PUSH AF; RRA; RRA; RRA; RRA;
0249 CD4D02F1               CALL B2HEX1; POP AF;
024D E60FC630       B2HEX1: AND #16'F; ADD #'0;
0251 FE3A3802C607           CMP #'9+1; IF CC; ADD #'A-'0-10; FI;
0257 C34A0C         JCRT:   J $CRT;
```

```
                        read in a hex number, DE being used as pointer to line
                           NUM+1, NUM+2 contain the number
                           NUM set non zero if there is a number there at all

025A 1AFE201328FA1B   NEXNUM: A_(DE); CMP #' ' ; INC DE; BR Z .; DEC DE;
0261 AF21120C                 CLA; HL_#NUM;
0265 7723772377               (HL)_A; INC HL; (HL)_A; INC HL; (HL)_A;
026A 1A2B2B           NN1:    A_(DE); DEC HL,HL;
026D D630F8                   SUB #'0; RET M;
0270 FE0A3808                 CMP #10; BR CS NN2;
0274 D607                     SUB #'A-'0-10;
0276 FE0AF8                   CMP #10; RET M;
0279 FE10F0                   CMP #16; RET P;
027C 133423ED6F       NN2:    INC DE,(HL),HL; RLD;
0281 23ED6F                   INC HL; RLD;
0284 18E4                     BR NN1;

                        main monitor loop; read a line and obey it

0286 CD0B01           PARSE:  CALL INLINE;
0289 114B0B010A0C1A           DE_#L NE+1; BC_#ARGS; A_(DE);
0290 FE20                     CMP #' ' ; [ CHECK FOR STEP REPEAT]
0292 20050AFE5320ED           IF Z; A_(BC); CMP #'S; BR NZ PARSE; FI;
0299 020313AF02               (BC)_A; INC BC,DE; CLA; (BC)_A;
                           get the arguments
029E 03CD5A02         PLOOP:  INC BC; CALL NEXNUM;
02A2 7EB7280D                 A_(HL); TSTA; BR Z PEND;
02A6 237E022303               INC HL; A_(HL); (BC)_A; INC HL,BC;
02AB 7E02                     A_(HL); (BC)_A;
02AD 210B0C3418EB             HL_#ARGS+1; INC (HL); BR PLOOP;
02B3 ED4B0A0C2A450C   PEND:   BC_ARGS; HL_@CTAB;
02BA 7EB728C8         PEND1:  A_(HL); BR A Z PARSE; [no such command]
02BE 23                       INC HL;
02BF B9                       CMP C;
02C0 2805                     IF NZ;
02C2 00                        NOP; [patch]
02C3 232318F3                  INC HL,HL; BR PEND1;
02C7                          FI;
02C7 5E2356                   E_(HL); INC HL; D_(HL);
02CA 218602E5EBE9             HL_#PARSE; PUSH HL; EX DE,HL; J (HL);
```

```
                              execute command, if arg supplied then this is start address

' 02D0 3EFF321A0C        EXEC:   CONFLG_#-1;
                               common to E and S, conflg tells which
                                  set NMI for end of instr
  02D5 21050322480C      EXEC1:  HL_#TRAP; $NMI+1_HL;
  02DB E1                        POP HL; [RUBBISH]
' 02DC 3A0B0CB72806              IF ARGS+1 NZ;
  02E2 2A0C0C223B0C               HL_ARG1; R.PC_HL;
  02E8                           FI;
  02E8 C1D1F1F1                  POP BC,DE,AF,AF;
  02EC 2A3D0CF9                  HL_R.SP; SP_HL;
  02F0 2A3B0CE52A370C            HL_R.PC; PUSH HL; HL_R.HL;
' 02F7 F53E08D300                PUSH AF; OUT 0,#*3;
  02FC F1ED45                    POP AF; RETN;

                              step, if arg supplied then this is address

  02FF AF321A0C18D0      STEP:   CLA; CONFLG_A; BR EXEC1;

  0305 E323E3            TRAP:   EX (SP),HL; INC HL; EX (SP),HL;
  0308 F5E5                      PUSH AF,HL;
' 030A 3A000CD300                OUT 0,PORT0;
' 030F 3A1A0CB72810              IF CONFLG NZ;
  0315 2A150C7E32170C             HL_BRKADR; A_(HL); BRKVAL_A;
  031C 36E7                       (HL)_#8'347; [RST 4]
  031E E1F1E32BE3                 POP HL,AF; EX (SP),HL; DEC HL; EX (SP),HL;
  0323 ED45                       RETN; FI;
  0325 D5                        PUSH DE;
  0326 C521000039        BPT1:   PUSH BC; HL_#0; ADD HL,SP;
  032B 11330C                    DE_#STACK;
  032E 31330C010800EDB0          SP_#STACK; BC_#8; LDIR;
  0336 5E23                      E_(HL); INC HL;
  0338 56231B                    D_(HL); INC HL; DEC DE;
  033B ED533B0C223D0C            R.PC_DE; R.SP_HL;

                                 print out regs  SP PC AF HL DE BC

  0342 213F0C0606                HL_#R.SP+2; B_#6;
  0347 2B7ECD4402        REGS1:  DEC HL; A_(HL); CALL B2HEX;
  034C 2B7ECD4402                DEC HL; A_(HL); CALL B2HEX;
  0351 CD3C02                    CALL SPACE;
  0354 10F1                      DBNZ REGS1;
  0356 CD4002                    CALL CRLF;
  0359 2A150C3A170C77    STRT0:  HL_BRKADR; A_BRKVAL; (HL)_A; [RESTORE BREAKPOINT]
  0360 C38602                    J PARSE;
```

```
                        command table
                            format:  character, address of subroutine

0363 4DAD01         CTAB:   .WORD 'M; .ADDR MODIFY;
0366 43EF03                 .WORD 'C; .ADDR COPY;
0369 45D002                 .WORD 'E; .ADDR EXEC;
036C 53FF02                 .WORD 'S; .ADDR STEP;
036F 54FC01                 .WORD 'T; .ADDR TABCDE;
0372 42E300                 .WORD 'B; .ADDR BREAK;
0375 4C7C03                 .WORD 'L; .ADDR LOAD;
0378 44D103                 .WORD 'D; .ADDR DUMP;
037B 00             NOP;

                        load command

037C CD5100         LOAD:   CALL MOTFLP; [start motor]
037F 218A0B22180C   LOD1:   HL_#CURLIN; CURSOR_HL;
0385 CD3E00FE1D28F9 LOD1B:  CALL CHIN; CMP #BS; BR Z .;
038C FE1F2805               CMP #CR; BR Z LOD1A;
0390 CD4A0C20F0             CALL $CRT; BR NZ LOD1B;
0395 118A0B0608     LOD1A:  DE_#CURLIN; B_#8;
039A 1AFE2ECA5100           A_(DE); CMP #'.; J Z MOTFLP;
03A0 CD5A022A130C           CALL NEXNUM; HL_NUM+1;
03A6 7D844F                 A_L; ADD H; C_A;
03A9 E5210008E5             PUSH HL; HL_#CRTRAM [TEMP BUFFER IN FIRST 8 WORDS]; PUSH HL;
03AE E5CD5A02237E   LOD2:   PUSH HL; CALL NEXNUM; INC HL; A_(HL);
03B4 E17723814F10F3         POP HL; (HL)_A; INC HL; ADD C; C_A; DBNZ LOD2;
03BB CD5A02237EB9           CALL NEXNUM; INC HL; A_(HL); CMP C;
03C1 E1D1                   POP HL,DE;
03C3 2007                   IF Z;
03C5 010800EDB018B3  BC_#8; LDIR; BR LOD1;
03CC                        FI;
03CC CD400218AE             CALL CRLF; BR LOD1;

                      DUMP,  uses same code as TABULATE

03D1 CD5100         DUMP:   CALL MOTFLP;
03D4 0600                   B_#0;
03D6 CD350010FB             CALL KDEL; DBNZ .;
03DB 2A4B0CE5               HL_$CRT+1; PUSH HL;
03DF 215D00224B0C           HL_#SRLOUT; $CRT+1_HL;
03E5 CDFC01                 CALL TABCDE;
03E8 E1224B0C               POP HL; $CRT+1_HL;
03EC C35100                 J MOTFLP;

                       copy,  arguments:from, to, length

03EF 2A0C0CED5B0E0C COPY:   HL_ARG1; DE_ARG2;
03F6 ED4B100CEDB0           BC_ARG3; LDIR;
03FC C9                     RET;

03FD 000000         .BLKZ 16'400-.; [PAD OUT TO END OF ROM]
0400                .END START
```

SYMBOL TABLE

```
ARGS   :0C0A      LOD1B :0385
ARG1   :0C0C      LOD2  :03AE
ARG2   :0C0E      MODIFY:01AD
ARG3   :0C10      MOD1  :01B0
BL     =0020      MOD2  :01BF
BPT1   :0326      MOD3  :01CF
BREAK  :00E3      MOTFLP:0051
BRKADR:0C15       NEXNUM:025A
BRKVAL:0C17       NN1   :026A
BS     =001D      NN2   :027C
B2HEX  :0244      NUM   :0C12
B2HEX1:024D       PARSE :0286
CHIN   :003E      PEND  :02B3
CONFLG:0C1A       PEND1 :02BA
COPY   :03EF      PLOOP :029E
CR     =001F      PORT0 :0C00
CRLF   :0240      PRS   :0028
CRT    :013B      PRS1  :0029
CRTRAM=0800       RAME  :0C15
CRT0   :0167      RAMTOP:0C50
CRT1   :016A      RAMZ  :0C00
CRT2   :016F      RAM.  =0C50
CRT3   :0195      REGS1 :0347
CTAB   :0363      ROMTOP:0400
CUR    =005F      ROM.  =0128
CURLIN=0B8A       R.AF  :0C39
CURSOR:0C18       R.HL  :0C37
DUMP   :03D1      R.PC  :0C3B
EXEC   :02D0      R.SP  :0C3D
EXEC1  :02D5      SPACE :023C
FF     =001E      SRLOUT:005D
FLIP   :0053      STACK :0C33
FLPFLP:004A       START :0000
INITE  :013B      STEP  :02FF
INITR  :0C3D      STRT0 :0359
INITT  :0128      TABCDE:01FC
INLINE:01DB       TBCD1 :01FF
INL0   :01DE      TBCD1A:0216
INL1   :01E9      TBCD2 :022B
INL2   :01EE      TBCD3 :0232
JCRT   :0257      TRAP  :0305
KBD    :0069      $CRT  :0C4A
KDEL   :0035      $CTAB :0C45
KMAP   :0C01      $KBD  :0C4D
KSC1   :007A      $KTAB :0C43
KSC1A  :0087      $KTABL:0C3F
KSC2   :008E      $KTABO:0C41
KSC3   :00E0      $NMI  :0C47
KSC8   :0089      .     =0400
KSC9   :008A
KTAB   :00EA
LINE   =0B4A
LOAD   :037C
LOD1   :037E
```

PART 18. HOW TO USE AND PROGRAMME YOUR NASCOM 1

SECTION A: HOW TO USE THE NASCOM 1

1. See page 3 of the handbook: 'Important notes for all NASCOM users'.

2. Ensure power supply voltages are correct before connecting to NASCOM 1. (Borrow a meter if necessary).

3. It is not advisable to switch the power supplies at low voltage as some devices (e.g. character generator and EPROM) can be destroyed by a momentary absence of the -5V supply. If it is necessary to switch the low voltage supplies they must be sequenced so that the -5V supply is always switched on first (before either of the positive supplies) and switched off last. This problem is avoided with mains supply switching due to the gradual rise and fall of the low voltages ensured by the large electrolytic capacitors incorporated in the smoothing circuits.

4. When the system is operating and the TV set tuned in, run through the NASBUG command examples in part 16 and in Section E of this part of the handbook. You can then try your own hand at programming or enter programmes obtained from any source.

5. If you wish to add further peripherals to your system see the following Section B and all relevant technical handbooks.

6. If you are new to computers see Section C and any good textbooks for further fundamentals.

7. If you wish in future to use a high level language or an assembler with your NASCOM 1 see Section D and the latest NASCOM product announcements.

8. Please join the International Nascom Users' Club and let us know any problems, constructive criticisms, hardware and software ideas you may have, any requests for future products and any interesting uses you have found for your NASCOM 1.

SECTION B: PERIPHERAL HARDWARE

Developments both in the semiconductor industry and in home computing are occurring at such a rate that these notes can only act as a pointer to the possibilities that exist for the experimenter. (See our seminar notes for some ideas).

However powerful a computer may be its usefulness is governed vitally by the interfaces through which it communicates with the user and with the real world. Without further specialised equipment the basic NASCOM 1 provides visual display of 768 characters on a video monitor or domestic TV set, a serial interface for programme or data storage on a domestic tape recorder, 16 parallel input/output lines (at 5V, TTL level) as well as a solid-state alphanumeric keyboard.

Although no specific tape recorder is recommended some useful points can be made. The NASCOM 1's simple 'cassette' interface relies on the presence or absence of a signal and can therefore be misled by tape dropouts or spurious interference. We therefore advise the use of the thickest tape (C60) from a reliable manufacturer. (A C60 tape can store some 85K Bytes of data or 20K bytes using the standard dump format). There is no need for ultra-low noise or CrO2 hi-fi tape. In addition great care should be taken to keep the tape heads and drive capstan clean and clear of oxide particles. All hi-fi stores sell tape head maintenance kits.

Any tape recorder, whether cassette or reel to reel, may be used. However a machine with a tape position counter is strongly advised as an aid to locating recorded data. The only argument in favour of a more expensive hi-fi deck is that the mechanism is less likely to wrinkle the tape and better speed control is probable. A stereo cassette deck should be operated in the mono mode, as there is too little separation between tracks to make separate recordings on each channel.

As an alternative, the serial I/O circuitry may be adjusted (using links LK 2, 3 & 4) to provide the standard 20mA loop interface with a Teletype or the standard RS232 interface with an external VDU, serial keyboard, etc. all via the 16 pin SK2. By altering connections to the UART control inputs (See Data Sheet provided) the serial interface may be used with a Baudot 5 bit code teleprinter. If an external clock signal for the UART (at 16 times the bit rate) is fed in via LK4, then data transfers may occur at any rate up to 200K baud. Any of these alternatives will need short driving programmes in order to perform transmission and interpretation of control signals as may be desired. If cassette operation is also wanted on the same unit, then either a multipole switch, relay or digital switch (e.g. CMOS 4016) could be wired in place of the links.

On a more mundane level there are few alternatives to be considered for the TV interface. The figure of 48 characters per line was considered the maximum legible on a domestic receiver. Experience has shown some difference between individual receivers, due to their various video IF

(Intermediate Frequency) responses. A colour TV set is not advised as the colour dot or stripe pattern combined with any convergence errors will render the characters less distinct. The NASCOM 1 may however be used with receivers built for virtually any colour or transmission standard in the world (except the old 405 line system) providing the tuner will receive UHF bands 4 and 5. A VHF only receiver could probably be used if the modulator fundamental frequency were reduced, e.g. by adding a turn or two to coil L1. Although intended for use with the 625 line 50Hz system, the NASCOM 1 could be used with a 525 line 60Hz receiver in North/South America or Japan if the picture height were reduced as the line frequencies are virtually identical.

If a receiver is purchased specially with the NASCOM 1 in mind, a small modern black and white portable with rectangular screen is the best answer. However, if perfect picture quality is necessary a video monitor is the only ideal solution. We do not advise anyone to attempt to convert a TV receiver to accept a video input unless they are fully aware of the dangers involved and are experienced in repairing TV sets. It is to be hoped that a video input will soon become a common feature of new receivers.

The 16 parallel input/output lines (and 4 associated control lines) available can be connected via suitable buffering circuitry to any device the user may provide. As inputs they can accept any data in digital form such as time of day/date from a clock system, ASCII code from a separate parallel connected keyboard, analogue data (such as temperature) after A/D conversion or simple connection to an on/off switch or sensor for each line. As outputs they can for example feed a parallel connected printer for hard copy output, control light levels via D/A converters and thyristor dimmers or provide information on 7-segment LED displays. The PIO Controller can be operated in various different modes as may be most convenient. For example, data can be read by the CPU from the parallel ports either by regular scanning of the port status or by the automatic generation of an interrupt when data is available. See the PIO technical manual for further details. Note also that two bits of Port 0 are available to the user in both directions.

In future, through the standard bus interface, the user will be able to plug into his system memory and I/O expansion cards (which we will be making available) or standard prototyping cards containing any circuitry he may desire. In this manner, virtually any number of additional serial or parallel interfaces and special purpose devices, such as the Z-80 CTC (Counter/Timer Controller), can easily be accommodated.

SECTION C: COMMUNICATING WITH THE COMPUTER

By itself the microprocessor or its equivalent at the heart of a large computer is not a useful device. If an array of switches and indicators is connected to all the address, data and control lines then the internal read/write memory (the Registers) and the specified set of instructions

for the device can be tested and confirmed to work (assuming it can be made to operate with a clock period of several seconds.)

At the next level, a real system dedicated to one particular purpose (such as the recently available single chip microcomputers) must contain some Read Only Memory (ROM) for programme storage and I/O lines to interface with the outside world (and may also need a small RAM scratchpad memory if the internal registers provide insufficient temporary storage). Such a device may be used to control a washing machine, a burglar alarm, an electronic calculator or a peripheral device in a larger computer system. It is, however, totally inflexible, as its internal programme will perform that one task only.

A microcomputer such as the NASCOM 1 gains its flexibility from the provision of a larger amount of Random Access or Read/Write Memory (RAM), to store any programme or data, convenient peripheral interfaces for the user, facilities to expand memory or peripherals virtually without limit and a monitor programme (or 'BUG') in ROM to tie the whole into a system. At its simplest, the latter may comprise a 'bootstrap loader' that merely enables data fed into a particular peripheral to be loaded into sequential locations in RAM. In our case, however, the NASBUG monitor in EPROM provides a comprehensive operating system that controls the keyboard, serial interface, video/TV display, programme entry, modification, execution and de-bugging. Thus we have a computer capable not only of performing any one or more of the functions mentioned above but also useful as a development aid in the design of any system dedicated to a particular purpose.

So far however all these units are controlled or programmed by the user in the only language that they are designed to obey: this is called machine code and comprises digital 'words' of 8 bits (1 byte) for the Z80 (although computers may use any word length e.g. 4, 8, 12, 16, 24, 32 or more bits). For our own convenience we represent these 8 binary digits in written form using 2 digits of hexadecimal code. Thus a word of 8 bits is split into 2 groups of 4 bits which are then each replaced by a single digit representing a number in the range of 0 to 15. A mixture of letters and numbers are used in hexadecimal code as shown below:-

BINARY (Number Base two)	DECIMAL (Base Ten)	HEXADECIMAL (Base Sixteen)
0000	0	0
0001	1 (= 2^0)	1
0010	2 (= 2^1)	2
0011	3	3
0100	4 (= 2^2)	4
0101	5	5
0110	6	6
0111	7	7
1000	8 (= 2^4) [2^3]	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

BINARY (Number Base Two)			DECIMAL (Base Ten)	HEXADECIMAL (Base Sixteen)
Further examples:				
		0001 0000 (1 byte)	16 ($=2^4$)	10
		0001 0001	17	11
		0001 1000	24	18
		0001 1111	31	1F
		0010 0000	32 ($=2^5$)	20
		0011 0000	48	30
		0100 0000	64 ($=2^6$)	40
		0110 0100	100	64
		1000 0000	128 ($=2^7$)	80
		1100 1000	200	C8
		1111 1111	255	FF
0000	0001	0000 0000 (2 bytes)	256 ($=2^8$)	100
0000	0001	1111 0100	500	1F4
0000	0010	0000 0000	512 ($=2^9$)	200
0000	0011	1110 1000	1000	3E8
0000	0011	1111 1111	1023	3FF
0000	0100	0000 0000	1024 ($=2^{10}$ or 1K)	400
0000	0111	1101 0000	2000	7D0
0000	1000	0000 0000	2048 ($=2^{11}$ or 2K)	800
0001	0000	0000 0000	4096 ($=12^{12}$ or4K)	1000
0010	0000	0000 0000	8192 ($=2^{13}$ or8K)	2000
0100	0000	0000 0000	16384 ($=2^{14}$ or16K)	4000
1000	0000	0000 0000	32768 ($=2^{15}$or 32K)	8000
1111	1111	1111 1111	65535	FFFF
ditto plus 1:		(3 bytes)	65536 ($=2^{16}$ or64K)	10000

(Any Hexadecimal number may appear in documents with a suffix 'H' in order to avoid confusion. E.g. 16 = 10H. Note also that computer terminals, printers, keyboards and display devices usually distinguish between the letter 'O' (which has no numerical meaning) and the digit zero by adding a stroke through the middle of the digit thus: 'Ø'. Typewriters do not usually have this facility, but no confusion is normally likely to arise. In manuscript, however, many programmers prefer to add the stroke for clarity).

One of the most vital functions provided by the NASBUG operating system therefore is to convert Hexadecimal code pairs typed on the 16 relevant keyboard keys into 8 bit bytes and enter these into memory. We have, therefore, a sy stem which we can programme (and into which we can load data) in the hexadecimal representation of machine code. We can call this Object Code (or, more loosely, machine language).

SECTION D: ALTERNATIVE PROGRAMMING LANGUAGES

Although at the deepest level every computer can only operate in its own machine language (and every different CPU has its own language) there are numerous advantages to be gained from programming the computer (in machine code) to communicate with the user in a language more nearly compatible with everyday human speech or specialist mathematical functions (with numbers in the decimal system). There are many such so called 'High Level' languages in use, each with its own adherents and each with its advantages and disadvantages. Among the widespread high-level languages are BASIC,

FORTRAN, COBOL (specially intended for business use), APL (Scientific), PL/1 and many more. Their most notorious feature is the way that each user and each computer manufacturer tend to diverge slightly from the standard (if any is laid down) so that a programme in, say, BASIC written for one machine may require subtle editing before it will work on another, due to a small difference in dialects. In general, however, they are a boon as they enable a programmer to write his programmes in a language that can be run (with luck first time) on <u>any</u> computer fitted with a compiler or interpreter to convert that high level language across into that particular machine's low-level language.

The NASCOM 1 memory expansion board has facilities provided to accept a 2K 'TINY BASIC' Interpreter in EPROM which will enable users to become acquainted with the essentials of BASIC programming. Plans have already been announced for a powerful 16K BASIC to be made available later.

Intermediate, between the low level machine language and the various high level languages, is the "ASSEMBLY LANGUAGE" or Source Code specific to each machine. For the convenience of programmers the manufacturer specifies for each machine language instruction (or Opcode) a more easily memorable 'mnemonic' code. For example for the Z80 we have:-

MNEMONIC	MEANING
LD	Load
EX	Exchange
HALT	Halt awaiting interruptor RESET.
IN	Accept data from peripheral port.
JP	Jump to another part of the programme.

(etc.)

Following the mnemonic we may have an OPERAND such as a register, a number, a memory location etc. For full details of the mnemonic codes, operands and addressing modes available see the Z80 Technical and Programming manuals.

If we do not have a high level language facility available or do not wish to use a high level language in order to economise on memory usage (or because the application is more suited to low level language programming) then we will be working in Assembly Language.

We therefore list the mnemonics and operands and then (using the tables in the Z80 handbooks) convert them into a parallel list of Hexadecimal Object Code. For lengthy programmes this procedure can become tedious, and so a programme called an Assembler can be used to perform the conversion for us. There are various types; from a simple interpreter that converts mnemonic code into object code directly; to multipass compilers that make sophisticated use of labels for subroutines etc. For those who require such facilities we will be making a powerful assembler available shortly.

The remainder of this guide shows how to programme the basic NASCOM 1 in Z80 machine language, using the peripherals provided.

SECTION E: PROGRAMMING THE NASCOM 1

1: Preliminary

Upon switch-on the NASCOM 1 enters a random state similar to the result of a programme that has run wild. The screen will be full of arbitrary characters, one or two of the LED indicators may be lit and there will be no response to the keyboard (except for the RESET key). In order to bring the monitor programme into operation it is necessary to operate the RESET key (which causes the CPU to Jump to the instruction at 0000; i.e. the first byte stored in the NASBUG monitor EPROM).

ACTION	RESULT
RESET	LED's turned off. Screen Cleared (except perhaps for one arbitrary character placed by breakpoint routine). PROMPT (>) and CURSOR (_) printed in bottom left hand corner.

The system is now executing a programme loop that scans the keyboard (providing software debounce and Hex code allocation via a look-up table) and also the UART for any serial input data. Any such data from either source is displayed in the next available screen position.

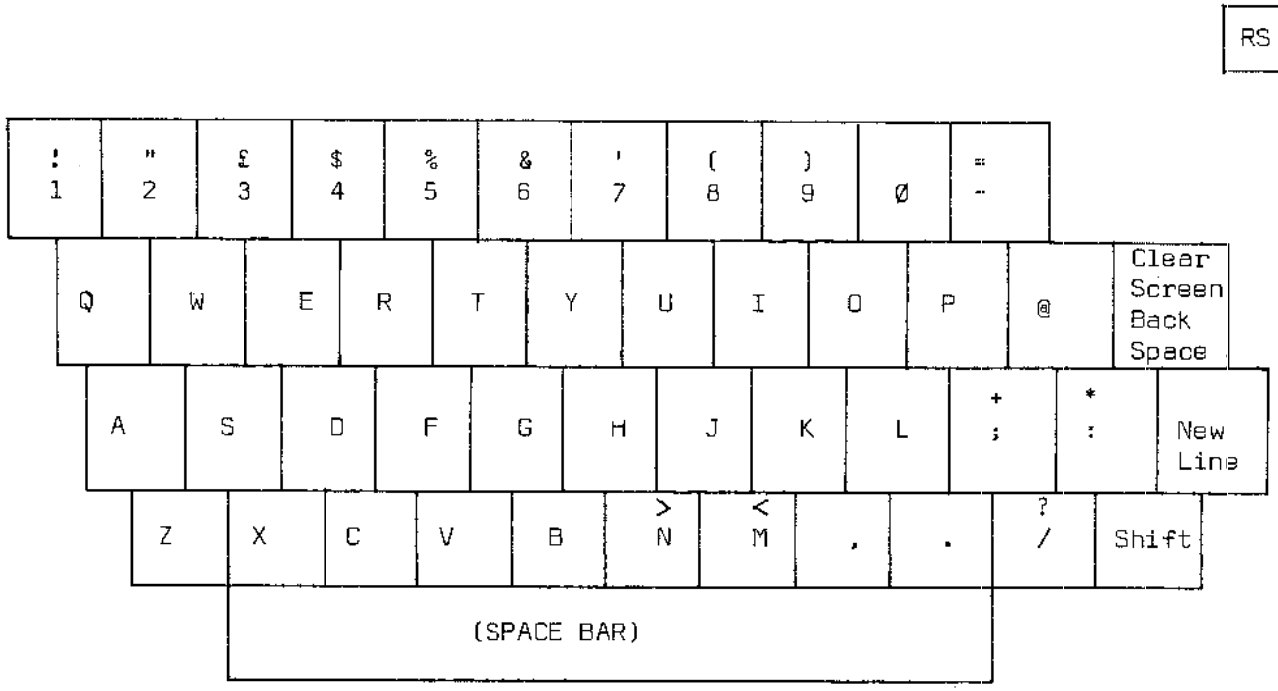
Before proceeding further, however, it is necessary to complete system initialisation by placing the breakpoint location out of harm's way (i.e. out of RAM). The standard procedure is to place the breakpoint at location 0000 as follows:-

ACTION
BO NL
ED NL

The user may now proceed to type any of the available keyboard characters on the screen and use any of the 8 monitor commands.

2. Keyboard Characters

The standard keyboard and monitor look-up table provide the following characters and control functions:



NASCOM 1 KEYBOARD LAYOUT

(SHOWING SHIFTED CHARACTER ALLOCATIONS)

If the SHIFT key is depressed together with a key for which no shifted character is allocated then the unshifted character will be displayed. If several keys are depressed together each of the characters will be displayed in an arbitrary order. If an attempt is made to place 48 or more characters in a line an upward scroll is automatically performed and the cursor moved to the bottom left hand corner.

One or more characters may be deleted and the cursor moved left by use of the BACKSPACE key. Backspacing is inhibited by the PROMPT (>) symbol however if this does not appear at the start of a line then BACKSPACE will move the cursor to the right hand end of the line above. The ultimate limit is the left hand end of the fifteenth line up.

The screen may be cleared (including the unscrolled top line) by use of the SHIFT and BACKSPACE keys together. This places the CURSOR in the bottom left hand corner but does not issue a PROMPT symbol, so the NEWLINE key must be pressed before issuing a monitor command.

The NEWLINE (NL) key causes the contents of each of the bottom 15 lines to be scrolled up by one line (the tope line of the 15 being lost) and the PROMPT and CURSOR symbols to be issued at the start of the cleared bottom line. This key is the most important of all, as it also performs the 'Enter Command' function, providing one of the 8 command characters is next to the PROMPT symbol in the bottom left hand corner.

If for any reason (e.g. use as a video typewriter for message display) it is desired to inhibit the monitor commands, one or more spaces should always be inserted between the PROMPT symbol and the first character. (Alternatively, a simple programme can have the same effect).

3. Monitor Commands

(For full details and examples see part 16 of this handbook).

M	Inspect/Modify memory contents. (Used for entering user programmes). - Terminated by a full stop.
T	Tabulate memory contents.
D	Dump from memory to serial interface (for cassette, etc.)
L	Load from serial interface to memory. (May be terminated by "NL full stop NL NL" if no full stop read off tape).
C	Copy memory contents from one section of memory to another.
B	Set breakpoint in user programme.
E	Execute programme.
S	Single step through programme. (Terminated by a different command or a full stop).

When a breakpoint is met (or when single stepping) six register pairs are displayed on the screen as follows:-

SP PC AF HL DE BC

To display register contents at any time the command S3FF NL (single step through a 'NOP' instruction) may be used.

The S command may be used to single step through programmes either in ROM (EPROM) or RAM and so can be used to demonstrate monitor routines (except for part of the keyboard scanning routine where lock-out will occur).

Initial Data or Address zeroes may be omitted so that the following commands are identical in effect:-

TO 5F = T0000 005F

When single stepping from a location other than the start of a programme beware of making illegal demands on the Stack Pointer. Never single step through a 'RET' or 'POP' instruction unless the corresponding 'CALL' and 'PUSH' instructions have previously been obeyed.

If in any doubt where the breakpoint is located check the contents of locations 0C15 and 0C16 (the least significant byte is first). Normally these will both contain zeroes if the breakpoint was last set at location 0000 (as during system initialisation).

If in <u>any</u> difficulty use the RESET key to regain control.

If a programme has gone wrong you may either single step from the start to see the point at which any register contents differ from those expected or move a breakpoint progressively through the programme until the trouble spot is isolated.

4. Programme Development Example

<u>Aim</u>: To display on the screen in table form (as on page 33) the full character set available from the character generator.

<u>Method</u>: We will dive straight into the problem. (Those who find flowcharts clarify the reasoning will find it instructive to draw their own. Consult textbooks/magazine articles if further information is required).

<u>Stage</u>	<u>Comments</u>
(a)	We must initialise any registers that are to act as pointers to Memory (or the screen) or to contain a constant for loop counting, etc. We will not know what initialisation is needed or the order in which it must appear until we have considered the next two stages:

Stage	Comments

(b) The programme will contain one basic loop in which VDU RAM location has a character code written into it and then the pointer is moved on to the next location.

(c) We must then test the pointer location (or some other parameter) to see if we have completed a line. Instead of looping back in the programme to write the next character we must first shift the pointer to the start of the next line and then loop back to a point in the initialisation routine that will cause a new line to be written. If we have reached the end of the table we must decide how to end the programme.

(d) Consider item (b). The obvious 'write' instruction (if we choose not to use subroutines in the monitor programme) is "LD (HL), A". We therefore use the HL register pair to point to VDU RAM memory locations and the accumulator (the A register) to contain the character code. If we space out the table roughly to fill the top half of the screen we must move the pointer right 3 spaces for the next position.

Thus the core of the programme is:-

```
LD (HL), A        Write character on screen.
INC A             Code for next character.
INC HL  }
INC HL  }         Move pointer to next location.
INC HL  }
```

(e) Consider item (c). DJNZ is a convenient 'test and loop' instruction and it uses the B register. We must therefore precede the main loop with an initialisation instruction to set the number of characters per line:

```
LINE:  LD  B,  10H   (10H = 16 Decimal).
```

After the main loop we will have:

```
DJNZ - 05H        If B not zero decrement B and jump
                  back 5 places to the location labelled
                  "WRITE" (We know the previous 5
                  instructions each comprise a single byte).
```

A convenient method of shifting the pointer to the start of the next line (rather than e.g. using INC HL sixteen times) is to store 16 (or 10H) in the DE register (as it is not needed elsewhere) and add it to HL thus:-

```
ADD  HL,DE
```

We must now detect the end of the table. We can do this either by testing the character code in A or the pointer location in HL. If we use the former method:-

```
CP 80H            Detect 129 th. character (1st was 00H).
```

We must then loop back to location "LINE" if A does not contain 80H thus:-

```
JRNZ "LINE"        Jump back to line if zeroflag
                   not raised (we can insert the
                   actual jump offset when converting
                   to object code subsequently).
```

We can then end the programme in one of several ways. We could use:

```
HALT
```

to light the 'HALT' LED and leave the CPU in effect executing a continuous series of NOP's. We could only exit from this however using RESET (or a peripheral interrupt if so programmed). We could call the KBD routine and test for particular key pressings to branch to any other programmes. The most useful ending however is jump back into the main monitor loop ("PARSE") using:-

```
JP  0286 H
```

We can then use any of the 8 monitor commands as was the case before the programme was executed.

(f) We can now list the complete programme with object code and initialisation routines. (The labels and comments are provided for guidance only). The locations shown for entering the programme are quite arbitrary as it contains no absolute jumps or calls and may be placed anywhere in User RAM.

TO DISPLAY CHARACTER SET ON SCREEN

LOCATION	OBJECT CODE	LABEL	SOURCE CODE	COMMENT
0CE0	11 10 00	START	LD DE,0010H	Width of margin.
0CE3	21 0B 08		LD HL, 080BH	Near top left of screen.
0CE6	3E 00		LD A, 00H	First character code.
0CE8	06 10	LINE	LD B, 10H	Characters per line.
0CEA	77	WRITE	LD (HL), A	Write character.
0CEB	3C		INC A	Code for next character.
0CEC	23		INC HL	Move pointer
0CED	23		INC HL	right 3 places.
0CEE	23		INC HL	
0CEF	10F9		DJNZ-5H	Jump to WRITE if line not ended.
0CF1	19		ADD HL,DE	Start of next line.
0CF2	FE 80		CP 80H	Test for end of table.
0CF4	20F2		JRNZ -CH	Jump to LINE if not.
0CF6	C3 86 02		JP 0286H	Jump to PARSE in NASBUG.

(g) The object code programme is entered using the M instruction as shown in part 16 of this handbook.

(h) It may then be tabulated using the T command thus:-

```
TCE0  CFF
        NL
```

and checked for any obvious mistakes.

(i) The S command may then be used to single step through the start of the programme to check that the registers are being manipulated correctly.

(j) The B instruction may be used to set a breakpoint in the main loop. For example:-

BCEB NL

(k) The programme can be executed up to the breakpoint (A'B' instruction must be followed by an 'E' instruction in any case). The starting location is 0CE0 so we type:-

ECE0 NL (or E0CE0 NL)

(l) We can then either use S to single step to check that the relative jump offsets are correct or type simply:-

↵NL

to pass round the loop once and return to the breakpoint. As stated earlier 6 major register pairs are displayed after each breakpoint or single step in order that any problem may be analysed.

(m) We can move the breakpoint elsewhere if desired (for example to 0CF1 at the end of one character row).

(n) We can remove the breakpoint:-

B0NL

(o) And then execute the programme normally:-

ECE0NL

(p) Note that a routine to clear the screen could be added at the start (See Section F).

(q) Due to the monitor's scrolling action, the character table will not remain on the screen while single-stepping or using the breakpoint.

(r) In order to become familiar with programming the user may next wish to modify or 'improve' the above programme. (For example to display a narrower table, to display only part of it or to re-locate it on the screen).

5. A Further Example Programme

Aim: To display the full character set as before but with a variable delay routine between 'write' instructions.

Execute Codes { $ED00_{NL}$ to cycle every minute or so (Exit via RESET)
$ED08_{NL}$ to write once and then return to monitor.

TO WRITE CHARACTER SET SLOWLY

LOCATION	OBJECT CODE	LABEL	SOURCE CODE	COMMENTS
0D00	CD 10 0D	START A	CALL 0D10H	Call subroutine.
0D03	CD 12 0D		CALL 0D12H	Call Sub.2 (Clear).
0D06	18 F8		JR -6H	Back to Start.
0D08	CD 10 0D	START B	CALL 0D10H	Call subroutine.
0D0B	C3 86 02		JP 0286H	Jump to monitor.
0D0E	00 00		NOP, NOP	(Padding).
0D10	0E 80	SUB ROUTINE	LD C,80H	1st character (plus 80H.)
0D12	11 10 00	SUB.2	LD DE,0010H	Width of margin.
0D15	21 0B 0B		LD HL, 0B0BH	Near top left of screen.
0D18	06 10	LINE	LD B, 10H	Characters per line.
0D1A	71	WRITE	LD (HL),C	WRITE character.
0D1B	79		LD A,C	Test for space code.
0D1C	FE 20		CP 20H	
0D1E	28 01		JR Z,+ 3H	Choose next character
0D20	0C		INC C	if not clearing screen.
0D21	23		INC HL	Move right
0D22	23		INC HL	3 places.
0D23	23		INC HL	
0D24	79		LD A,C	Variable delay depending on Register C, i.e. character being written.
0D25	08		EX AF,AF'	
0D26	AF		XOR A	
0D27	3D		DEC A	
0D28	20FD		JRNZ-1H	
0D2A	08		EX AF,AF'	
0D2B	3D		DEC A	
0D2C	20F7		JRNZ-7H	
0D2E	10EA		DJNZ-14H	Jump to WRITE unless end of line.
0D30	19		ADD HL,DE	Move to next line.
0D31	7C		LD A,H.	Detect end of
0D32	FE 0A		CP 0AH.	table.
0D34	20 E2		JRNZ - 1CH	Jump to LINE unless table ended.
0D36	0E 20		LD C, 20H	Set space code for next pass.
0D38	C9		RET	Return to calling routine above.

Rather than giving a full description of this programme, it is left to those who wish to analyse or modify it to learn from so doing. Note that a different register has been chosen to store the character code so as to leave the accumulator free. There is nothing ideal about the methods used in this (or any other programme). The essential criterion is simply "DOES IT PERFORM THE ALLOTTED TASK AND NOT HAVE ANY HARMFUL SIDE EFFECTS?"(such as overwriting a section of memory reserved for another purpose by the programmer).

SECTION F: PROGRAMMING HINTS

You should soon be able to write any programme your time, growing skill, peripherals and RAM capacity will allow (be it a computer game, a personal diary, a business accounting system or just a moving 'video wallpaper' pattern). Of course the longer programmes are much easier and quicker to write with the aid of an Assembler or High Level Language facilities but in the interim here are some essentials and ideas to enable you to use your NASCOM 1 to the full. More information may be gleaned by studying the various magazines and the NASBUG listing.

Some NASBUG Subroutines, Etc. (See also page 47)

1. CD 6900 = CALL 0069H (CALL KBD). This scans the keyboard and if a key is pressed returns the corresponding Hex code in A and sets the carry flag.

2. CD 3B 01 =CALL 013BH (CALL CRT)This takes a code previously placed in A and displays the character corresponding to the 7 least significant bits in the present cursor position on the screen (and moves the cursor on). If the code in A was 1EH then the screen will be cleared.

3. CD 3C 02 = CALL 023CH (CALL SPACE). Moves cursor on one position.

4. CD 40 02 = CALL 0240H (CALL CRLF). Scrolls display up one line.

5. CD 44 02 = CALL 0244 H (CALL B2 HEX). Takes 8 bit byte in A and displays the two digit hexadecimal representation of it on screen.

6. CD 35 00 = CALL 0035H (CALL KDEL). Returns after 7½ ms delay (and clears A).

7. C3 86 02 = JP 0286H (JP PARSE). Jump into main monitor loop scanning UART and keyboard for data/commands.

8. To re-write CRT (Screen display) routine or amend it:- insert address of new routine in 0C4B and 4CH.

9. Ditto, for keyboard scanning routine: 0C4E and 4FH.

10. To redefine Command table insert starting address of new table at 0C45 and 46H.

11. Ditto, for keyboard table: 0C43 and 44H.

12. Insert length of new keyboard table in 0C3F and 40H.

13. Insert origin (if different) of new keyboard table in 0C41 and 42. (In each case the most significant byte is placed last).

14. The Command table may be disabled by typing:-
MC45$_{NL}$ then FF.$_{NL}$

15. The spare output bits from port zero can be controlled by setting or resetting the corresponding bits in location 0C00H while the monitor remains in use.

16. The first address av ailable for user programming is 0C50H (Not 0C60H as indicated elsewhere).

Z80 Programming Hints

17. To clear the accumulator y ou may use either:-

```
    3E 00        LD A, 00H.
or: AF           XOR A.
```

The only advantage of the former is that it does not affect the flags.

18. If you require to set flags (e.g. after loading A) then you may use:-

```
B7     OR    A
```

without changing the contents of A.

19. If an arbitrary no. in the range 00H to 7FH would be useful try:-

```
ED  5F       LD A, R.
```

Although this is a random number the first time it is used, it will be predictably different a fixed number of M1 states later.

20. To copy the carry flag to all bits in A use:-

```
      9F  SBC  A, A.
```

21. To copy the carry flag to all bits in H and L use:-

```
      ED 62  SBC  HL, HL.
```

22. To initialise PIO ports as outputs from the NASCOM 1 use:-

```
      3E  0F  LD A, 0FH (0F = 'MODE 0')
      D3  06  OUT (06), A. (for port 4).
      D3  07  OUT (07), A  (for port 5).
```

23. To initialise PIO ports as inputs to the NASCOM 1 you may use:-

```
      3E  4F  LD A, 4FH  (4F = 'MODE 1')
      D3  06  OUT (06),A (for port 4).
      D3  07  OUT (07),A (for port 5).
```

(For details see PIO Handbook).

24. To assist in the calculation of relative jumps here is a brief table:-

Jump (Decimal)	Jump (HEX)	Object Code (2nd Byte)
-126 (max.)	-7EH	80
- 62	-3EH	C0
- 32	-20H	DE
- 30	-1EH	E0
- 16	-10H	EE
- 14	-0EH	F0
- 12	-0CH	F2
- 10	-0AH	F4
- 9	-09H	F5
- 8	-08H	F6
- 7	-07H	F7
- 6	-06H	F8
- 5	05H	F0
- 4	-04H	FA
- 3	-03H	FB
- 2	-02H	FC
- 1	-01H	FD
+ 3	+03H	01
+ 4	+04H	02
+ 5	+05H	03
+ 6	+06H	04
+ 7	+07H	05
+ 8	+08H	06
+ 16	+10H	0E
+ 18	+12H	10
+ 32	+20H	1E
+ 64	+40H	3E
+128	+80H	7E
+129 (max).	+81H	7F

SECTION G: Conversion Between 8080 and Z80 Programmes

The Z80 machine code instruction set is an expanded version (or superset) of that for the earlier 8080 microprocessor. It includes 2 byte opcodes and 4 byte instructions (max. 3 for the 8080). An 8080 machine code programme will therefore run on the Z80 but not vice-versa unless any specific Z80 codes used are re-written in a longer form. The additional Z80 instructions (e.g. to use relative jumps or the index registers) all begin with:-

08, 10, 18, 20, 28, 30, 38, CB, D9, DD, ED or FD.

(Note that 20 and 30 are used by the 8085 for special instructions - read/set interrupt mask - not compatible with the Z80).

The mnemonics specified in the two microprocessors' assembly languages are however very different and so the NASCOM 1 user who wishes to use and understand any of the many published 8080 programme listings may find the following conversion table useful:

MNEMONIC CONVERSION TABLE

8080		Z80	
ACI	n	ADC	A,n.
ADC	r	ADC	A,r.
ADC	M	ADC	A,(HL).
ADD	r	ADD	A,r
ADD	M	ADD	A,(HL).
ADI	n	ADD	A,n.
ANA	r	AND	r
ANA	M	AND	(HL)
ANI	n	AND	n
CALL	nn	CALL	nn
CC	nn	CALL	C,nn.
CM	nn	CALL	M,nn.
CMA		CPL	
CMC		CCF	
CMP	r	CP	r
CMP	M	CP	(HL)
CNC	nn	CALL	NC, nn
CNZ	nn	CALL	NZ,nn
CP	nn	CALL	P,nn.
CPI	n	CP	n
CPE	nn	CALL	PE,nn
CPO	nn	CALL	PO,nn
CZ	nn	CALL	Z,nn
DAA		DAA	
DAD	B	ADD	HL,BC.
DAD	D	ADD	HL,DE.
DAD	H	ADD	HL,HL.
DAD	SP	ADD	HL,SP

8080		Z80	
DCR	r	DEC	r
DCR	M	DEC	(HL).
DCX	B	DEC	BC
DCX	D	DEC	DE
DCX	H	DEC	HL
DCX	SP	DEC	SP
DI		DI	
EI		EI	
HLT		HALT	
IN	n	IN	A, (n)
INR	r	INC	r
INR	M	INC	(HL)
INX	B	INC	BC
INX	D	INC	DE
INX	H	INC	HL
INX	SP	INC	SP
JC	nn	JP	C,nn
JM	nn	JP	M,nn
JMP	nn	JP	nn
JNC	nn	JP	NC,nn
JNZ	nn	JP	NZ,nn
JP	nn	JP	P,nn
JPE	nn	JP	PE,nn
JPO	nn	JP	PO,nn
JZ	nn	JP	Z,nn.
LDA	nn	LD	A,(nn).
LDAX	B	LD	A,(BC).
LDAX	D	LD	A,(DE).
LHLD	nn	LD	HL,(nn).
LXI	B,nn	LD	BC,nn.

8080		Z80	
LXI	D,nn	LD	DE,nn.
LXI	H,nn	LD	HL,nn.
LXI	SP,nn	LD	SP,nn.
MOV	r,r'	LD	r,r'
MOV	M,r	LD	(HL), r.
MOV	r,M	LD	r, (HL)
MVI	r,n	LD	r,n
MVI	M,n	LD	(HL),n
NOP		NOP	
ORA	r	OR	r
ORA	M	OR	(HL)
ORI	n	OR	n
OUT	n	OUT	(n),A
PCHL		JP	(HL)
POP	B	POP	BC
POP	D	POP	DE
POP	H	POP	HL
POP	PSW	POP	AF
PUSH	B	PUSH	BC
PUSH	D	PUSH	DE
PUSH	H	PUSH	HL
PUSH	PSW	PUSH	AF
RAL		RLA	
RAR		RRA	
RC		RET	C
RET		RET	
RIM	(8085)	-	
RLC		RLCA	
RM		RET	M
RNC		RET	NC
RNZ		RET	NZ

8080		Z80	
RP		RET	P
RPE		RET	PE
RPO		RET	PO
RRC		RRCA	
RST	0	RST	00H
RST	1	RST	08H
RST	2	RST	10H
RST	3	RST	18H
RST	4	RST	20H
RST	5	RST	28H
RST	6	RST	30H
RST	7	RST	38H
RZ		RET	Z
SBB	r	SBC	A,r
SBB	M	SBC	A,(HL)
SBI	n	SBC	A,n
SHLD	nn	LD	(nn),HL.
SIM	(8085)	-	
SPHL		LD	SP,HL.
STA	nn	LD	(nn),A.
STAX	B	LD	(BC), A.
STAX	D	LD	(DE),A.
STC		SCF	
SUB	r	SUB	r
SUB	M	SUB	(HL)
SUI	n	SUB	n
XCHG		EX	DE,HL
XRA	r	XOR	r
XRA	M	XOR	(HL)
XRI	n	XOR	n
XTHL		EX	(SP),HL.

KEY

r or r' = Register A, B, C, D, E, H or L
n = 8 bit number or port address.
nn = 16 bit number or memory address.

Note in particular the following possible sources of confusion:-

8080	Z80
CP	= CALL P
CMP or CPI	= CP
-	= CPI
JP	= JP P
JMP	= JP

In order to keep 8080 compatibility together with an expanded instruction set a few different Z80 opcodes happen to have the same functional effect.

e.g. (one byte):	0F	RRCA
=(two byte):	CB 0F	RRC A

SECTION H: LAST WORD

Please keep in touch with us via the NASCOM USERS CLUB, and let us know your problems and successes. We also welcome lists of any corrections or suggested additions to our or the relevant manufacturers' data in order to keep everyone equally informed (via the club newsletter).

We look forward to sharing with you a continuing role in the microprocessor revolution.