CONTENTS

SECTION 0 PRELIMINARIES.

0.1 Introduction.

NASMON is a powerful 4K operating system suitable for use with both NASCOM 1 and NASCOM 2 microcomputers.

NASMON incorporates an extended screen editor with features such as block move, long lines, append from tape and find and substitute commands - see Section 2 for full details.

Also included within NASMON is a Front Panel mode which displays all the registers and flags and allows the programmer to access both the memory and the registers directly. In addition many useful and powerful commands are provided to give the user remarkable debugging facilities. See Section 1 for further details.

NASMON features a repeat keyboard with both the initial delay and the repeat rate changeable.

All the major i/o routines within the monitor are vectored through locations in RAM so that it is possible for the user to write his or her own routines to interface with, say, half-duplex terminals or parallel printers or whatever and then interface these routines with NASMON through SYS ('F7') calls - see Sections 3 and 4.

NASMON uses some of the Z80 restart (RST) instructions to give two byte system calls (SYS), relative subroutine calls (RCAL) and breakpoints (BRK) - see Section 3. The other restart instructions are vectored through the workspace RAM - see Section 4.

0.2 Action on RESET.

On RESET or power-up the screen will be cleared and you will be prompted, via '_' in the top left of the screen, to enter the required start address of text terminated by any non-hex character. This address should be greater then 3FFFH.

After you have entered the required start of text NASMON will set various pointers and storage locations and then display a page of text. This will contain only one character of text since an empty file is assumed; the character will be '←', the end-of-line character.

The top line will contain information on the text - see Section 2.1. In this case the first 4 columns will contain the address of the end of text which will be one greater than the specified start of text since the file is empty. The position of the cursor within the text will be shown in columns 7-9 and will be 1; we are at the first character in the line. Finally columns 12,13 and 15-17 will show the relative position of the various Find and Move pointers, see Section 2.3. In this case this part of the display will be '-- ---' since all the pointers are set to the beginning of text initially.

Now refer to Sections 1 and 2 to understand how to use the Screen Editor and Front Panel commands.

Note: if RESET is pressed while you have text in the machine then, instead of entering an address in response to '_', use CTRLC. This action will recover the text exactly as it was previously except that the Find and Substitute strings will be lost. The RESET will have initialised all the workspace except TEND , the Find and Move pointers, BUFPOS and ENDRAM.

0.3 The Keyboard.

NASMON supports both types of NASCOM keyboard.

Letters are normally upper case while the use of the SHIFT key reaches the lower case characters.

The action of the SHIFT key may be inverted using CTRLK whenever characters are being read by SYS 11 (see Section 3) i.e. whenever you are typing in response to a blinking cursor.

Entry to the Front Panel always resets the action of the SHIFT key as it was on RESET i.e. its use reaches lower case.

Both the @ key and the CTRL key (on NASCOM 2 type keyboards) function as a control key allowing you to reach codes 01H to 1AH (@A to @Z) and codes 60H (@space bar) and 78H to 7FH (use @ with codes 38H to 3FH i.e. @; to @?). Each occurrence of @ in the above can of course be replaced by CTRL on NASCOM 2 type keyboards.

The '@' sign is reached via CTRL- or @- and not via SHIFT@ as in some other monitors.

'J' is available directly on NASCOM 2 type keyboards and may be reached via @0 (@zero) on NASCOM 1 type keyboards.

The GRAPH key on the extended keyboard set bit 7 of the character while held down enabling the graphics ROM if this is fitted and the graphics switched on.

Note: by a combination of keys all ASCII codes are available through the extended keyboard. The older keyboard can reach all codes except:

00H, 1CH, 1DH, 1FH, 5CH and 5FH.

0.4 Entering Hexadecimal Numbers.

Many commands within NASMON require you to enter a hexadecimal number e.g. the Editor 'T' and 'U' commands and the Front Panel 'J' and 'M' commands. All these commands use the same routine HEXNUM (see F713 in Section 3) to read the hex number. Since this routine is used so often within the monitor some comments on its function are given below:

a. HEXNUM accepts any number of hex digits with the following provisos: if less than 4 digits are entered then the number is padded to the left with zeros giving a total of 4 digits returned if more than 4 digits are entered then HEXNUM remembers and displays only the last 4 entered.

b. Any non-hex character, except '-', acts as a terminator.

c. If a minus sign '-' is entered at any time then the number is negated e.g. both -20 and 2-0 return FFE0H.

d. On exit from HEXNUM, HL contains the number, C the number of digits entered, A the terminator and bit 7 of B is set iff a minus sign was entered at any time while bit 6 of B is set iff the number contains only denary digits.

e. HEXNUM recognises CTRLC.

0.5 Conventions used in this Manual.

CTRLA means 'hold the CTRL (or @) key down and press A'.
SHIFTA means 'hold the SHIFT key down and press A'.
If a character after a hex number is underlined then this means that the character is acting as a terminator.

SECTION 1 THE FRONT PANEL.

In this mode, which is entered via CTRLZ from SYS 11, you are presented with a display of the Z80 registers, the flags and a 20 byte portion of memory centred around the location held in MEMPOS - see Section 4. The value held by MEMPOS, the Memory pointer, is indicated in the display by two arrows: → ←.

The Front Panel display is updated after the execution of each command so that you can monitor any changes that may occur.

See Appendix 1 for an example Front Panel display.

Commands are entered from the keyboard in response to the prompt ' '. Commands take effect immediately; there is no need to terminate them with an ENTER/NEWLINE. Invalid commands are simply ignored.

The commands available are:

ENTER / NEWLINE — increment the Memory pointer by one so that the 20 byte memory display is now centred around an address one greater than it was before.

'/' — decrement the Memory pointer by one.

',' — decrement the Memory pointer by eight - used to step backwards quickly.

'.' — increment the Memory pointer by eight - used to step forwards quickly.

':' — update the Memory pointer so that it contains the address currently on the stack. This is useful when you want to look around the return address of a called routine or generally for looking at the location pointed to by the first address on the stack.

G — get a named file from tape. Under NASMON programs and data are stored on tape in blocks, with a header label preceding the blocks and a trailer label following them (see Appendix 2). The 'name' of the tapefile is contained within the header label and may be up to 6 characters in length.

The 'G' command first prompts '$' and you should respond to this with the name of the file that you are searching for followed by ENTER/NEWLINE.

Having obtained the name of the required file the command scrolls the display up, turns the motor drive LED on and begins to read through SYS 4.

When a header label is encountered the name within it is displayed on the screen and if this name corresponds to the name of the requested file then the file is loaded, otherwise 'G' searches for another header label.

As the file is loaded the start address of the block currently being loaded is displayed and, to the left, the actual byte.

Each block is 128 bytes long and includes a checksum. Data is first loaded into a buffer (BUFFER - see Section 4) so that if a checksum error occurs then the relevant memory will not be corrupted. If no checksum error is

detected then the block will be transferred from the buffer to the relevant memory addresses. However, if an error is detected then the buffer is not cleared, the message 'error' is displayed and you are given the option of either abandoning the load via CTRLC or attempting to reload the block (after rewinding the tape) by typing any other character.

The last block in the file should be a trailer label. This will contain the autostart address of the file i.e. the address to which execution is to be transferred at the end of the load.

Having encountered the trailer label 'G' will turn off the motor drive LED and transfer execution to the autostart address.

Note: at the start of each block a check of the keyboard is made and if CTRLC is down then the loading will be aborted and control passed back to the editor command loop. This enables you to escape from 'G'.

H — set a breakpoint at the location addressed by the Memory pointer.

This command places FFH at the location of the breakpoint, stores the original contents of this address in BRKVAL and stores the address itself in BRKADR (see Section 4 for the addresses of these locations).

Only one breakpoint at a time may be set using 'H' but you may set any location in RAM to FFH to force a break at that location (see Section 3.3); in that case, though, the original contents will not be restored.

Note: RESET initialises BRKADR to zero and so, if a break occurs after a RESET then the original value at the location will <u>not</u> be restored.

I — intelligent copy. This is used to copy a block of memory from one location to another. It is intelligent in that the block of memory may be copied to locations where it would overlap its previous locations.

'I' prompts for the inclusive start and end locations of the block to be copied ('First:', 'Last:') and then for the start address to which the block is to be moved ('To:').

If the start address of the block is greater than the end address then the message 'error' will be displayed and control passed back to the front panel command loop.

J — execute code from specified address. This command first prompts on the same line with ':' and then waits for you to enter a hexadecimal address (see Section 0.4).

On termination of the hex address the stack will be reset to 1000H and the Program Counter (PC) will be loaded with the specified address.

Example:

J:5000CR will begin executing code at address 5000H.

K

continue execution from the address held in the Program Counter (PC). This command will probably be used most frequently in conjunction with the 'H' command. An example should help to clarify this usage:

Say you are single-stepping (using 'Z') the code given below and you have reached address 5920H. You are now not interested in stepping through the subroutine at 6000H but wish to see how the flags are set at the end of the call to the subroutine at 5800H.

```
5920 CD0060          CALL £6000
5923 2A0050          LD   HL,(£5000)
5926 7E              LD   A,(HL)
5927 111458          LD   DE,£5814
592A CD0058          CALL £5800
592D 2003            JR   NZ,lab1
592F 320250          LD   (£5002),a
5932 211458 lab1     LD   HL,£5814
```

Proceed as follows: set a breakpoint, either using 'H' or directly with FFH, at location 592DH and issue a 'K' command. Execution continues from the address held in the PC which, in this case, is 5920H. A 'break' will then occur at location 592DH and the registers and flags will be displayed as they are after the call to the subroutine at 5800H - you will then be able to inspect the changes that have occurred and continue to single-step through the code.

So 'K' is useful for executing blocks of code without resetting the stack as 'J' does.

L

tabulate, or list, a block of memory starting from the Memory pointer.

'L' clears the screen and displays the hexadecimal representation and ASCII equivalents (or graphics) of the 112 bytes of memory starting from the current position of the Memory pointer.

The display consists of 14 rows with 8 bytes per row, the ASCII being displayed at the end of each row.

At the end of a page of the display you have the option of either returning to the Front Panel display by pressing BACKspace or continuing to display the next page (112 bytes) by pressing any other key (other than CTRLC).

Note: in the ASCII display characters below 20H are displayed as '.'.

M

set the Memory pointer to a specified address.

This command prompts with ':' and waits for you to enter a hexadecimal address. The Memory pointer is then updated with this address and the memory display changed accordingly.

Useful as a prelude to entering code, tabulating memory etc.

N — find the next occurrence of the hex string previously defined by 'S' (see below).

'S' allows you to define a string and then searches for the first occurrence of it; if you want further occurrences of the string then use 'N'.

'N' begins searching from the Memory pointer and updates the memory display when the next occurrence of the string is found. Note that an occurrence of the string will always be found at location 0007H since this is where NASMON stores the master copy of of the pattern.

O — jump through RVECT (see Section 4).

This command has exactly the same effect as J:C16CR; it transfers execution to 0C16H after resetting the stack. After RESET or power-up 0C16H, 0C17H and 0C18H contain a JP £1800 instruction so that execution will be further transferred to location 1800H. This will enter NASNEM, our 2½K disassembler, if this has been supplied in EPROM.

If you desire to change the effect of 'O' then simply change locations 0C17H and 0C18H so that they contain the address (low-order byte first) that you wish 'O' to jump to.

P — put a block of memory to tape.

'P' prompts for the inclusive start and end addresses of the block of code that you wish to dump to tape after first having prompted for the name under which the block is to be saved - this must be up to 6 characters in length and terminated with ENTER/NEWLINE. Finally this command prompts for an autostart address which is the address to which control will be transferred when the block is loaded from tape using 'G'. If this is defaulted then the autostart will have the same effect as RESET.

Example:

```
P
$TESTCR
First:4000CR
Last:5000CR
Start:4000CR
```

will dump the block of memory between 4000H and 5000H inclusive to tape under the name 'TEST' with an autostart address of 4000H.

If 'First' is greater than 'Last' then the message 'error' will be displayed and the command aborted.

Note: the dump may be aborted at the beginning of each block by holding CTRLC down.

Q — flip register sets.

After RESET the Front Panel displays the standard register set HL, DE, BC and AF. The use of 'Q' will display the alternate register set HL', DE', BC' and AF' which may be distinguished from the standard set by the single quote "'" after the register name.

If 'Q' is used when the alternate register set is displayed then the standard set will be displayed.

R — this is used to investigate the destination of relative jumps or calls. It takes the byte currently addressed by the Memory pointer, treats this byte as a relative displacement and updates the memory display accordingly.

Example:

Say the Memory pointer is addressing location 6800H and that the contents of locations 67FFH and 6800H are 20H and 16H respectively which could be interpreted as a JR NZ, +24 instruction. To find out where this branch would reach on a NZ condition simply press 'R' when the Memory pointer is addressing the displacement byte 16H. The display will then update to centre around 6817H, the destination of the branch.

Remember that relative displacements greater than 7FH are treated as negative; 'R' takes this into account.

See also the 'U' command in connection with 'R'.

S — search memory for a specified string.

The command scrolls up the command line and waits for you to enter the required string. This is done by entering one hex byte at a time, terminated by any non-hex character. The string itself is terminated by entering a non-hex character only. On termination of the definition of the string 'S' searches memory, starting from the Memory pointer, for the first occurrence of the string and then updates the memory display to point to the first character of the found occurrence.

Example:

Say that you wish to search memory, starting from 5000H, for occurrences of the pattern 3EFFH (two bytes). Proceed as follows:

```
M:5000CR    set Memory pointer to 5000H.
S
3ECR        define the first byte of the string.
FFCR        define the second byte of the string.
CR          terminate the definition.
```

After the final ENTER/NEWLINE (this could be any non-hex character as explained above) 'S' proceeds to search memory from 5000H for occurrences of the string 3EFFH. When the string is found (there will always be one occurrence at 0D07H - see 'N') the memory display is updated to point to the first character of the string.

Further occurrences of the string may be found using 'N'.

The string may be up to 256 characters in length and is stored from 0D07H onwards.

T — fill memory between specified limits with a specified character.

'T' prompts for 'First:', 'Last:' and 'With:'. These should be the start and end addresses (inclusive) of the block that you wish to fill and the byte that you wish to fill the block with.

'error' will be displayed if the start address is greater than the end address.

U used in conjunction with the 'R' command.

Remember that 'R' updates the memory display according to a relative displacement i.e. it emulates a JR or RCAL (Relative CALl) instruction.

'U' is used to update the memory display to where the <u>last</u> 'R' was issued.

Example:

5200 47	51F3 77
5201 20	51F4 C9
→ 5202 F2 ←	→ 51F5 F5 ←
5203 06	51F6 C5
display 1	display 2

You are on display 1 and wish to know to where the relative jump 20F2H branches. So you press 'R' and the memory display updates to display 2. Now you investigate the code following 51F5H for a while and then wish to return to the code after the original relative jump to see what happens if the Z flag <u>is</u> set. So press 'U' and the memory display will return to display 1.

Note that you can only use 'U' to return to the <u>last</u> occurrence of 'R', all previous addresses are lost.

V used in conjunction with the 'X' command.

'V' is similar to the 'U' command in effect except that it updates the memory display to where it was before the last 'X' command was issued.

Example:

4702 AF	442D 18
4703 CD	442E A2
→ 4704 2F ←	→ 442F E5 ←
4705 44	4430 21
display 1	display 2

You are on display 1 and wish to look at the subroutine at 442FH. So you press 'X' with the display centred as shown; the memory display then updates to display 2. You step through this routine for a while and then wish to return to the code after the original call to this subroutine. So press 'V' and display 1 will reappear.

As with 'U' you can use this command only to reach the address at which the <u>last</u> 'X' command was issued; all previous addresses at which 'X' was used are lost.

W used to initialise NASNEM, our 2½K disassembler, if supplied in EPROM.

'W' transfers execution to location 1C06H which is the address of the routine in NASNEM that initialises the interface of the disassembler with the monitor. Put simply, the use of 'W' turns on the disassembly of the instruction at the Memory pointer to the top line of the Front Panel display - as long as NASNEM is supplied in EPROM.

To switch off the disassembly enter 02H at location 0C12H - this works for NASNEM supplied in EPROM or on tape.

X used to investigate the code at the destination of extended jumps and calls.

'X' takes the address specified by the byte at the Memory pointer and the byte at the Memory pointer + 1, and then updates the memory display so that it is centred around that address. Remember that the low order half of the address is specified by the first byte and the high order half of the address is given by the second byte.

Example:

Say you wish to look at the routine that the code C30563 (three bytes) jumps to. Set the Memory pointer so that it addresses the 05H within the JP and then press 'X'. The memory display will be updated so that it is centred around location 6305H.

See also the 'V' command in connection with 'X'.

Y enter ASCII from the Memory pointer.

'Y' prompts with ':' for you to enter a string of ASCII characters terminated by BACKspace. The characters entered will be echoed after the colon and their hex equivalents entered immediately into memory starting from the Memory pointer.

Note that 'control' characters (00H - 1FH) are not echoed; this includes ENTER/NEWLINE.

Z single-step.

After 'Z' the instruction that is pointed to by the Program Counter (PC) is executed and, immediately after the execution, the Front Panel is displayed with the registers, flags and memory display updated so that they reflect the state of the machine after the execution of the instruction given its state before the execution. The PC will now contain the address of the instruction immediately following the instruction just obeyed (or the destination address if a jump or call instruction was executed) and the Memory pointer will also be set to this address.

Two ways in which this command might be used follow:

- you have written a piece of code starting at 5000H which you wish to test out, instruction by instruction. So load the PC with 5000H using the ';' command (see below) and then use 'Z' continually, noting the effect of each instruction on the registers etc.

- you wish to investigate the effect of a small body

of code within a large program. Set a breakpoint (using 'H') at a point just prior to the code in which you are interested and execute the program. A 'break' will occur at the address where you set the breakpoint and you can now step through the relevant small piece of code using 'Z'. When finished you can then continue the rest of the program by using 'K'.

'Z' can be used to step through any piece of code including NASMON itself.

'Z' utilises the NMI feature of the NASCOM/Z80 by setting bit 3 of port 0. If an NMI is invoked directly (i.e. not through port 0) then control is passed to location 0C43H.

< single-step the next 2 bytes.

This is useful when you come across a SYStem call or Relative CALl while single-stepping which you do not wish to investigate in detail; it is effectively a double-step command. The operation of '<' is as follows:

A breakpoint is set at Memory pointer + 2 and a 'K' is issued; this has the effect of treating the instruction at the Memory pointer as if it was an ordinary 2 byte Z80 instruction.

Example:

```
          5010 3E          5014 60
          5011 4F          5015 AF
        → 5012 F7 ←        5016 69
          5013 01          5017 01
```

You are single-stepping the above code and you come to the F701 i.e. SYS 1; output A to the video. You obviously do not wish to go through the laborious process of single-stepping through the SYS 1 routine and equally it is tedious to have to set a breakpoint at 5014H and issue a 'K'. So simply press '<' and the SYS 1 will be executed and a 'break' will occur at 5014H automatically.

> single-step the next 3 bytes.

Similar to '<' above this command effectively triple-steps from the Memory pointer; it is useful for executing CALL instructions in one go.

'>' sets a breakpoint at Memory pointer + 3 and issues a 'K'; this has the effect of treating the instruction at the Memory pointer as if it was a standard 3 byte Z80 instruction.

Example:

```
          7130 06          7134 98
          7131 FE          7135 21
        → 7132 CD ←        7136 00
          7133 80          7137 00
```

You are single-stepping the above code and you come to the CD8098 (CALL £9808). You know that the subroutine at 9808H works and exactly what effects it has on the registers etc. so you do not wish to single-step it; simply press '>' and the subroutine will be called and then a 'break' will occur at 7135H automatically.

';' modify registers - see below.

Modifying Memory.

The contents of the address given by the Memory pointer may be modified by typing in between 1 and 4 hex digits followed by any non-hex character, apart from CTRLC and ';' (see below). The two least significant (one digit is padded to the left with a zero) hex digits are entered into the location addressed by the Memory pointer and then the command specified by the terminator is obeyed. If the terminator is not a valid command then no action is taken.

Examples:

F2CR	F2H is entered and the Memory pointer advanced by 1.
123.	23H is entered and the Memory pointer advanced by 8.
EM:E00_	0EH is entered at the current Memory pointer and then the Memory pointer is updated to E00H. Entering a space also acts as a terminator and is shown as a terminator to the argument of the 'M' command.
8CR	8CH is entered and then the Memory pointer is updated to the destination of the relative displacement 8CH i.e. to Memory pointer - 115.
ZASD_	5DH is entered and the Memory pointer is left as it is since the terminator was a space, not a command.

Modifying Registers.

If a string of hex digits is entered as above but the terminator is ';' then the string of digits will be entered into the register currently addressed by '→'.

On RESET '→' points to the Program Counter (PC) and so using ';' as a terminator to a hex number initially will modify the Program Counter. Should you wish to modify any other register then use ';' by itself (not as a terminator) and the pointer, '→', will cycle round the registers PC through AF. Note that it is not possible to modify the Stack Pointer, SP.

Examples:

Say that the Register pointer is initially addressing the PC.

;	point to IX.
;	point to IY.
0;	set IY to 0000H.
;	point to HL.
123;	set HL to 0123H.
;	point to DE.
;	point to BC.
-1D59;	set BC to E2A7H.
;	point to AF.
FF00;	set A to FFH and reset all the flags.
;	point to PC.
5000;	set the PC to 5000H.

Note that ';' can also be used to modify the alternate register set if this is displayed. Use 'Q' to flip the display of the register sets.

SECTION 2 EXTENDED SCREEN EDITOR.

Entry into NASMON's extended screen editor is accomplished via CTRLC from any routine that scans the keyboard using SYS 2 or SYS 11 (see Section 3.2.2).

Each line of text may be up to 256 characters in length and lines are separated by the end-of-line character 0DH; the ASCII CR character.

Pointers to the start and end of text are held; TSTRT and TEND (see Section 4). On RESET the user is prompted to enter a start address or default to 4000H. This action will set TSTRT to the value entered (or 4000H) and set TEND to TSTRT+1. This has the effect of creating a new file of text at TSTRT which contains no text, except an end-of-line character at TSTRT. New files may then be created elsewhere in memory or old files opened for use by employing the 'T' and 'U' commands; see below.

Five other pointers are kept; two 'find' pointers (FARG1 and FARG2) and three 'move' pointers (MARG1, MARG2 and MARG3). The use of these pointers is explained below (see the 'F' and 'M' commands, Section 2.3).

The editor has three distinct modes of operation; the 'command' mode, the 'change' mode and the 'insert' mode. Each mode shares a core of cursor control commands; these are detailed below.

The 'command' mode is used to make block moves, block deletes, substitutes, dumps to tape or printer etc.

The 'change' mode allows text to be overwritten directly in a similar way to the simple editing used by NASMON 3 and NAS-SYS.

The 'insert' mode is employed to insert text at the current cursor position and recognises BACKspace and CTRLI (tabulate).

2.1 Screen format.

Text is held on the lower 15 lines of the video screen while the top line is reserved for display purposes and is also used as the buffer for find and substitute commands.

Each of the lower 15 lines holds up to 48 characters of text from the corresponding text line. If the text line contains more than 48 characters (including the end-of-line character) then the rest of the line may be accessed by stepping the cursor right, beyond the 42nd column position; when this is done the rest of the line is gradually brought on screen from the right and the whole screen is shifted left. When the cursor is stepped right past an end-of-line character then the cursor is set to the beginning of the next line or, if already on the last line, to the beginning of the present line.

End-of-line characters are displayed on the screen as '←'; this enables the user to be clear as to where each line ends.

No character (except CR) has any special significance within the text as far as the editor is concerned and all characters, including graphics, are displayed directly on the screen. However certain CTRL characters are recognised by HISOFT's text processor and the Programmer's Manual for this software should be consulted if you desire to insert any CTRL characters in the text.

The columns on the top line of the display have the following significance:

Column	Contents
1	for certain commands this column is used to prompt (via '$') for a name or a hex. address.
1-4	display the current end of text address.
1-6	used to hold the filename being dumped to or retrieved from tape.
3-6	address of the start of the block currently being dumped to or retrieved from tape.
7-9	position within the current text line.
12,13	pointers to the current positions of FARG1 and FARG2.
14	blank if in Command mode, 'C' if in Change mode or 'I' if in Insert mode.
15,16,17	pointers to the current positions of MARG1, MARG2 and MARG3.
19	prompt ('$') for the 'find' string.
20-33	the current 'find' string.
34	prompt ('$') for the 'substitute' string.
35-48	the current 'substitute' string.

For further details of the Find and Substitute commands see Section

2.2 Functions available in all modes.

The following are available in the Command, Change and Insert modes and have the same function in each mode.

↑ CTRLS	cursor up. If the cursor is in the top line on screen and not in the first line of text then the screen will be scrolled down and a new line of text brought in from the top.
↓ CTRLT	cursor down. If the cursor is in the bottom line on screen and not in the last line of text then the screen will be scrolled up and a new line of text brought in from the bottom.
→ CTRLR	cursor right. If the cursor is moved beyond an end-of-line character then the cursor will be moved to the start of the next line or, if currently in the last line of text, then the cursor will be moved to the start of the current line. If → is entered when the cursor is at column 42 and not on an end-of-line character then the lower 15 lines of the screen will be shifted one character left and a new character of

the current line will be brought in from the right.

← CTRLO	cursor left. If the cursor is moved beyond the start of the line then the cursor will be placed at the end of the previous line or, if currently in the first line of text, then the cursor will be moved to the start of that line. If necessary (i.e. if beyond column 42 in the text) the bottom 15 lines of the screen will be shifted one character right and new text brought in from the left.
SHIFT↑ CTRLG	displays the previous page of text. The cursor will be positioned in the line which is 15 lines less than the current line or in line 1 if there are fewer than 15 lines prior to the current line.
SHIFT↓ CTRLY	displays the next page of text. The cursor position within the page is maintained.
CTRLA	single step substitute. For details see the find command ('F') below.
CTRLB	this control character is detected within SYS 11 and advances the rate at which a character repeats when its key is held down. Use repeatedly if necessary.
CTRLC	monitor warm start. Detected within SYS 2 and SYS 11, this causes execution to be transferred to 0003H which resets the cursor to the start of text and clears the find and substitute buffers.
CTRLD	detected within SYS 11, this decreases the rate at which a character is repeated if its key is held down. Use repeatedly if desired.
CTRLF	single step find. For details see the find command ('F') below.
CTRLI	tabulate; advances the position of the cursor to the next 8 character boundary. Ignores line boundaries.
CTRLK	shift invert. CTRLK is detected within SYS 11 and inverts the action of the SHIFT key on alphabetics only. On RESET, the use of the SHIFT key will reach the lower case letters - capitals are reached directly.
CTRLP	move the cursor to the last line of this page, maintaining the cursor position within the line.
CTRLZ	enter the front panel display mode. Detected within SYS 11. CTRLZ should NOT be used to exit from Insert mode.
CH CTRLW	move the cursor to the first line of this page, maintaining the cursor position within the line.
LF CTRLJ	move the cursor to the beginning of the current line.

CS SHIFTBACK move the cursor to the end of the current line.

2.3 The Command mode.

This is the mode in which the majority of commands are entered.

Commands are single alphabetic characters and may be entered in upper or lower case; they are not echoed and, in general, they take effect immediately.

Certain commands prompt for a name (via '$') or a hexadecimal address (via '_') in the top left of the screen. A name may be up to 14 characters in length and may contain any character except CTRLC, BS (backspace, which deletes the previous character) or CR which terminates the name. A hexadecimal address is entered according to the format detailed in Section 0.4.

Commands that may have a drastic effect on the text (such as block delete) prompt, via '_', for a character; if the character subsequently entered is any character other than a 'Y' - it must be a capital - then the command is aborted.

The following is a detailed list of the commands available:

A - append text from tape. Prompts for the six character name of the textfile to be loaded and proceeds to search the tape for a header label (see Appendix 2) containing this name. When found, the textfile is appended to the current text even if the current text contains only an end-of-line character i.e. is empty.

N.B. If a checksum error occurs during the load then the message 'error' will be displayed; you then have the option of aborting the load via CTRLC or rewinding the tape and continuing the load by pressing any key other than CTRLC. However, since an append is taking place, any text that is read from the tape that has already been successfully loaded will be appended again; this can obviously lead to duplication of text within the file. If the checksum error occurs in only one or two blocks then the text can be recovered by deleting the excess text - if, however, there are many errors on the tape then you are advised to load the text via the front panel 'G' command (see Section 1) which, although it does not allow you to append, will not duplicate blocks read in twice since they are loaded at the same address.

On a successful termination of the append control is returned to the editor, setting the cursor to the start of text and maintaining the find and substitute strings.

B - set the cursor to the beginning of the first line in the current text.

C - enter the Change mode; see below.

D - delete the current line from the text. If the last line of text is deleted then the cursor is moved up to the previous line, otherwise the cursor is moved to the following line.

E - set the cursor to the last character of the last line of text.

F - set up the Find and Substitute strings. You are prompted with '$' at column 19 on the top line to enter a string of up to 14 characters terminated by ENTER/NEWLINE. This is the Find string. You are then prompted at column 34 on the top line to enter a second string of up to 14 characters terminated by ENTER/NEWLINE; this is the Substitute string. Both strings may be null.

After you have entered the Substitute string 'F' will then search for the first occurrence of the Find string starting from the current text position+1 and within the text range previously specified by the 'G' and 'H' commands - see below.

If an occurrence of the Find string is found within this range then the cursor will be positioned on the first character of the string.

Further occurrences of the Find string within the range of the Find pointers may be found by using CTRLF repeatedly.

Use of CTRLA will replace the current occurrence of the Find string with the Substitute string (even if the Substitute string is null), expanding or contracting the text as necessary. After the substitution CTRLA then does a CTRLF. Note that, if the substitution has expanded the line in such a manner that the Find string is again present after the current text position (e.g. replacing 'Pascal' with 'Hisoft Pascal'), then it will be necessary to do a further CTRLF in order to point to the next 'true' occurrence of the Find string.

If the Find string cannot be found within the range of the Find pointers defined by 'G' and 'H' then the cursor position is not changed.

G - set the first Find pointer (held in FARG1 - see Section 4) to the current text position. Both 'F' and CTRLF will only treat a found occurrence of the Find string as significant if it is present at or after the first Find pointer and before the second Find pointer.

H - set the second Find pointer (held in FARG2 - see Section 4) to the current text position.

I - enter the Insert mode; see below.

J - jump through location 0C13H. This location is initialised on RESET to JP £2000 which will enter Hisoft's 3K assembler, NASGEN, if present in EPROM. NASPAS 3, our 12K Pascal compiler, sets this location so that using 'J' will invoke compilation of the current text.

K - kill (or delete) the character at the current cursor position. It is possible to kill end-of-line characters but it is not possible to kill the last end-of-line character in the text with this command.

L - set the end of text pointer, TEND, to the start of the current line + 1 and insert an end-of-line character at the start of the current line.
This command would be most often used in conjunction with the 'U' command to open a previously created body of text by stepping one line past the required last line of text and then using 'L' - see the 'U' command below for an example.

'L' first prompts you to enter 'Y' to ensure that you want the command to take effect.

M - block move.The block of text delimited by the first two Move pointers (set by the 'N' and 'O' commands - see below) is moved before the position specified by the third Move pointer (set by the 'P' command).

The move is inclusive and the original block of text is not deleted. Since all pointers maintain their positions within the text, the block delete command may be used subsequently to delete the original block.

Text cannot be moved within itself i.e. the third Move pointer, the destination pointer, cannot lie between the first two Move pointers although it may be set to be coincident with the first Move pointer.

The 'M' command first prompts you to enter 'Y' to ensure that you are positive that the block move is to take place.

In conjunction with 'A'ppend the 'M'ove command effectively provides a Merge command.

N - set the first Move pointer (held in MARG1 - see Section 4) to the current text position.

O - set the second Move pointer (held in MARG2 - see Section 4) to the current text position.

P - set the third Move pointer (held in MARG3 - see Section 4) to the current text position.

Q - transfer execution to location 2C00H. This is the location of Hisoft's text processor when supplied in EPROM so that this command enables direct entry into that software.

R - jump through location 0C16H. This location is initialised on RESET to JP £1800 and so 'R' will cause execution to be transferred to Hisoft's 2½K disassembler, NASNEM, if this has been supplied in EPROM.

NASPAS 3, our 12K Pascal compiler, changes locations 0C17H and 0C18H so that the use of 'R' when NASPAS is loaded and RESET has not been pressed will cause the current object program (compiled by NASPAS) to be executed.

S - save the current text to tape. Prompts for a name of up to 6 characters and then dumps the text within the inclusive range specified by the first two Move pointers (set by 'N' and 'O') to tape in the standard blocked format - see Appendix 2.

When the dump is finished control is returned to the editor with the cursor set to the first character in the text and the Find and Substitute strings preserved.

T - open a new text file at the address specified. The command prompts you (with '_' in column 1 of the top line) to enter a hexadecimal address. After you have entered an address, which must be above 3FFFH, the start of text pointer is set to the address, a CR character is placed at the address and the end of text pointer is set to the address entered + 1. This has the effect of opening a new file of text at the address specified.

U - open an existing text file at the address specified. Like 'T' this command prompts you to enter a hexadecimal address which must be greater than 3FFFH. Once you have entered an address

the start of text pointer (held at TSTRT) is set to the address and the end of text pointer (held in TEND) is set to FFFFH.

This has the effect of opening a text file which lies between the address entered and FFFFH. Obviously you will not usually have this much text in memory but perhaps only the first 2K above the specified start address will contain useful text. So you use ↓ to step down the text until the cursor is positioned on the line <u>past</u> the last line of required text and then use the 'L' command to set the end of text at the beginning of this line. You will then have re-opened the text file present at the address specified.

Note that stepping down the text to reach the required last line may take some time because of the length of the non-text lines at the end of the file. You can use SHIFT↓ but this will need to be done carefully since you may position yourself in a page that contains nothing but junk.

N.B. There is no need to use 'U' if you have pressed RESET while editing; you can recover the text by simply typing CTRLC, although you will lose the Find and Substitue strings.

V - transfer execution to location 0C34H. This location is initialised to FFH i.e. 'V' will normally cause a breakpoint to occur.

However NASPAS 3 sets this location so that use of 'V' when the compiler is loaded and RESET has not been pressed will enable you to re-initialise the compiler - see the NASPAS 3 Implementation Note for NASMON Users.

W - write a block of text to printer. The block of text specified by the first two Move pointers (set by 'N' and 'O') is output via SYS 5 which, if this vector has not been changed, will cause the text within these limits to be output to the serial port and hence to a serial printer.

The SYS 5 vector may be changed (see Section 3.2.1) if it is desired to output to a parallel printer or to format the text in some way - say to add a LF after each CR.

X - expunge (delete) a block of text. This command deletes the block of text defined by the first two Move pointers (set by 'N' and 'O') and is inclusive. It will frequently be used after a block move command to delete the original block of text.

'X' first prompts you to enter 'Y' to ensure that you are positive that you want the delete to go ahead.

Y - delete the text from the current cursor position to the start of the current line. The delete will include the character at the cursor position unless it is an end-of-line character.

Z - delete the text from the current cursor position to the end of the current line, not including the end-of-line character.

2.4 The Change mode.

In this mode text may be directly overwritten. Any character except those given in Section 2.2 and those listed below may replace any character within the text, except an end-of -line character.

While in Change mode, all the functions given in Section 2.2 may be used together with the additional functions given below:

SHIFT → CTRLV	open up the line from the current position. Starting from the current text position the line is shifted one character to the right and a space inserted at the current text position. This is useful for inserting small quantities of text without having to enter the Insert mode.
SHIFT ← CTRLU	close up the current line from the current text position. The character at the current text position is deleted and the rest of the line to the right of this position is moved one character to the left.
BACK	simply moves the cursor one character to the left. This is identical to ← and is only provided for the convenience of the user.
CTRLE	enter the Insert mode - see below.
ESC SHIFTENTER	return to the Command mode.

While in Change mode the character 'C' will be displayed at column 14 on the top line as a reminder.

N.B. No action will be taken if an attempt is made to change an end-of-line character or a character beyond an end-of-line character.

2.5 The Insert mode.

In this mode text may be inserted at the current text position.

All the functions in Section 2.2 may be used together with the following:

BACK	backspace over the last character inserted. The backspace will take effect on any character currently in the insert buffer but not otherwise i.e. it is not possible to use BACK within the Insert mode to delete characters that have not just been inserted.
CTRLE	enter the Change mode - see above.
ESC SHIFTENTER	return to the Command mode.
CTRLX	causes the input buffer to be reset without moving the contents into the textfile i.e. CTRLX allows you to abandon the insertion on the current line.

The insert buffer can hold up to 128 characters and this limits the number of characters that may be inserted without using CTRLX or one of the functions given in Section 2.2 which clear the buffer. Use of these functions output the text in the insert buffer to the textfile and resets the buffer; ENTER/NEWLINE also has this effect except that the CR is also inserted into the text.

If you attempt to insert characters when the cursor is beyond an end-of-line character then the insert will take place as if the cursor had been positioned on the end-of-line character.

N.B. Do NOT use CTRLC and CTRLZ to exit from Insert mode since the buffer pointers will not be reset.

While in Insert mode the character 'I' will be displayed in column 14 on the top line of the display to remind you that you are inserting text.

2.6 Display of Find and Move pointers.

As described in Section 2.1 certain column positions on the top line of the display are used to point to the current positions of the various Find and Move pointers.

Columns 12 and 13 point to the Find pointers.

Columns 15, 16 and 17 point to the Move pointers.

The display used is as follows:

↑ means that the relevant pointer is above the current text position.

\- means that the relevant pointer lies at the current text position.

↓ means that the relevant pointer is below the current text position.

On RESET all the pointers are set to the beginning of text as they are after the use of the 'T' and 'U' commands.

SECTION 3 SYSTEM CALLS AND RESTARTS.

Certain of the Z80 restart instructions are used by NASMON, the others being vectored through locations in the workspace area (see Section 4). The restarts which are used are: RST 28H ('EF'), RST 30H ('F7') and RST 38H ('FF'). Their functions and use are detailed below.

3.1 RST 28H - 'EF'.

This restart is used to provide the programmer with a 'relative call' instruction. The Z80 language provides relative jumps of course (18H, 20H, 28H etc.) but it is quite often the case that you will want to call a subroutine which is within range of a relative displacement - this is not supported in Z80 code and you are forced to use a CDH instruction or similar which takes 3 bytes instead of the 2 bytes that a relative call would occupy.

So NASMON gives you the chance of employing relative calls by using 'EF' followed by the relative displacement of the subroutine from the address of the instruction after the relative call. This displacement is calculated in exactly the same way as for relative jumps and may take any value between -128 and +127 inclusive.

Example:

Given below is a routine to read a character from the keyboard and then output its ASCII number (0 - 255), padded with zeroes to the left. Study it carefully and note, in particular, the way in which relative calls (RCAL) are employed.

```
F708    ASCOUT  SYS   11              read keyboard.
0E64            LD    C,100           output the hundreds
EF0C            RCAL  DIGOUT          digit.
0E0A            LD    C,10            output the tens
EF08            RCAL  DIGOUT          digit.
F630            OR    £30             convert units to
F701            SYS   1               ASCII and output.
F71A            SYS   26              output CR.
18EE            JR    ASCOUT
06FF    DIGOUT  LD    B,-1            B contains count of
04      LOOP    INC   B               how many times C
91              SUB   C               divides into A at the
30FC            JR    NC,LOOP         end of LOOP.
81              ADD   A,C             make A positive.
F5              PUSH  AF
78              LD    A,B             put count in A.
F630            OR    £30             convert to ASCII
F701            SYS   1               and output.
F1              POP   AF
C9              RET                   return.
```

For details of the system routines (SYS) used in the above routine see Section 3.2 below.

3.2 RST 30H - 'F7'.

'F7' is used to gain access to certain routines within NASMON which are frequently used. The byte which follows the 'F7' specifies the routine number (see below) thus enabling system routines to be called using only two bytes instead of the usual three. This can save a substantial amount of space in a large program although this method of calling a NASMON routine takes longer than a direct CALL. 'F7' calls are SYS calls.

The byte following the 'F7' restart may take any value from 0 to 127 decimal (inclusive) i.e. 00H - 7FH. Any value outside this range will result in 'error' being displayed and control will be passed back to the 'high' level mode. Within the range 00H - 7FH certain values and ranges of values have special significance:

00H produces a monitor warm start (like CTRLC).
01H - 0AH jump through the transfer vectors in the workspace.
0BH - 18H call selected system routines within NASMON.
1CH - 7FH jump through location F7EXT (0C46H).

The ranges given above are now considered separately and in detail:

3.2.1 Routines 01H - 0AH.

These routine numbers cause execution to be transferred to locations 0C19H, 0C1CH, ..., 0C34H depending on the number i.e. F701 transfers execution to address 0C19H, F702 to 0C1CH etc.

At these locations within NASMON's workspace are jump ('C3') instructions to various input/output routines within NASMON. On initialisation these locations are set up as follows:

0C19	C36206	JP	OUTCH	output a character.
0C1C	C32810	JP	KSCAN	scan the keyboard.
0C1F	C3E207	JP	TPOUT	output to tape.
0C22	C30010	JP	TPIN	read from tape.
0C25	C3E007	JP	PRINCH	output to printer.

locations 0C28H - 0C36H are set to FFH.

From inspection of the above it may be seen that these 'F7' calls have the following effects:

F701 output the character in A to the video screen.

F702 scan the keyboard - returns with the Carry flag set and the character in A if a key is pressed. Otherwise the Carry flag is <u>not</u> set. Recognises CTRLC.

F703 put the character in A to the serial port (port 1) and wait until sent.

F704 read the serial port (port 1) and wait until a character is read - returns with the character in A.

F705 strip the top (7th) bit off the character in A and output this to the serial port (port 1), expecting a serial printer to be attached.

F706 - F70A cause a 'break' to occur between 0C28H and 0C34H.

Note that, since these input/output calls reach their destinations through vectored jumps within the workspace (i.e. RAM), then the vectors (locations 0C19H - 0C27H) may be changed if you want to write a new routine for 'print a character' or 'read from tape' etc. So if you decided to write a new print routine (and the one in NASMON is certainly rather simple, but deliberately so) at address 5000H then simply load this address into locations 0C26H and 0C27H (low order half first) and then, whenever F705 is executed, execution will be routed to 5000H, your print routine. End your routine with RET ('C9').

I/O calls F706 to F70A inclusive are reserved for your future use, maybe for paper tape routines or disc handling routines. These locations may be set to JUMP ('C3') to addresses of your choice but remember that you will have to set the JUMP ('C3') as well as the address since all the memory between 0C28H and 0C36H is initialised to FFH.

3.2.2 Routines 08H - 18H.

These 17 'F7' calls transfer execution to routines whose addresses are permanently fixed within NASMON - they are not vectored and these calls will always have the same effect - you cannot change this. Note that 'F708' is equivalent to SYS 11 etc.

The routines that these calls reach are as follows:

Call	Name	Address	Effect
F708	INCHB	1047H	waits for a character to be input from the keyboard while blinking the cursor. Character returned in A. Recognises CTRLC and CTRLZ.
F70C	LSTART	06C7H	sets HL to the start of the current video line (i.e. the line in which the cursor is). On entry assumes that HL contains the current cursor position. Corrupts A.
F70D	CLRPAN	0331H	clears the first 12 lines of the video and sets the scrolling to the bottom 3 lines of the screen. Corrupts all registers except A, IX and IY.
F70E	SCLRST	0359H	resets scrolling to 16 lines from top of the screen. Corrupts HL, DE, BC.
F70F	CLEAR	061CH	clears 'n' lines of the video screen from address 'X'. 'n' is picked up from location 0C00H and 'X' from 0C01H and 0C02H.

Call	Name	Address	Effect
F710	SEC1	06F0H	gives a delay of one second at 4MHz or two seconds at 2MHz.
F711	SPACE	0657H	outputs a space (20H) through SYS 1. Corrupts A.
F712	OUTPO	1008H	outputs to Port 0 the mask of this port held at location 0C03H.
F713	HEXNUM	10C1H	reads a hexadecimal number from the keyboard. Digits cycle round if more than 4 digits are entered. Exits if a non-hex character is pressed*. On return HL contains the hex number (padded to the left with zeroes), C contains the number of digits entered, A contains the terminator and bit 7 on B is set if a minus sign was entered within the number. Bit 6 on B is set if the number contains only denary digits. *Except a minus sign '-'.
F714	DEOUTV	1115H	outputs the hex value contained in DE through SYS 1. Corrupts A.
F715	AOUTV	1122H	outputs the hex value held in A through SYS 1. Corrupts A.
F716	GNAME	10D6H	prompts via '$' and then reads up to 14 characters from the keyboard (via SYS 2) entering them into 0C77H onwards. Only ENTER/NEWLINE acts as a terminator. Corrupts HL, DE, BC and A.
F717	MGFAST	108AH	fast message output. Takes the characters from the address in HL onwards and 'pokes' them into the video memory (does not use SYS 1). Any character in the range 81H to 00H inclusive acts as a terminator of the string.
F718	STEND	109DH	prompts for 'First:' and then 'Last:' and expects hex addresses to be input, via SYS 2, after the prompts. On exit DE contains the 'First' address, BC the 'Last' and HL the difference. If 'First' is greater than 'Last' then the message 'error' is displayed. Corrupts A.
F719	MGSLOW	107DH	'slow' message output. Takes the characters from the address held in HL and outputs them through SYS 1. Only 00H acts as a

Call	Name	Address	Effect
			terminator. Used to output graphics and interpreted CTRL characters. Corrupts A and HL.
F71A	CROUT	065CH	outputs a carriage return character (0DH) through SYS 1. Corrupts A.
F71B	DISPG	12AAH	outputs a page of text to the video. On entry HL should point to the start of a line of text, B should contain the number of lines above the line in HL that you wish displayed (0-14) and C should contain the column position of the cursor within the line, starting from 1. On exit DE contains the cursor position in the video RAM and A is corrupted.

3.2.3 Routines 1CH to 7FH.

These routine numbers are available for the programmer's own use. On SYS ('F7') calls with any of these arguments control will be passed to location F7EXT (0C46H) with the routine number in A. Locations 0C46H, 0C47H and 0C48H contain FFH on initialisation so that a 'Break' will occur if these routines are used 'cold'.

However the three locations 0C46H - 0C48H may be changed by the programmer to cause a jump to a user routine which will handle these SYS calls. So you have the possibility of calling frequently used routines within your own program by using two byte 'F7' calls.

Remember to use a JP instruction ('C3') at 0C46H, not a CALL instruction. At the end of the routine called to handle these calls issue a RET ('C9') instruction and control will be passed back to the instruction after the original F7XX call.

3.3 RST 38H - 'FF'.

When this restart is encountered control is passed directly to the code which displays the Front Panel after displaying the message 'Break'.

If the Memory pointer has the same value as the address held in BRKADR (0C55H) then the FFH is replaced with the character held in BRKVAL (0C74H).

The effect of the above is to provide a breakpoint in a particular piece of code by setting FFH at the relevant point. When the FFH is executed the Front Panel display will appear showing the registers, flags and memory as they were prior to the execution of the restart. This is obviously very useful for debugging assembly programs. Note that if the breakpoint was set using the Front Panel 'H' command then the original byte which the FFH replaces is remembered and is replaced just before the Front Panel display appears.

For further details see the Front Panel 'H' and 'K' commands in Section 1.

3.4 RSTs 00H to 20H - 'C7' to 'E7'.

RST 00H ('C7') restarts the monitor by transferring control to address 0000H i.e. it performs a cold start.

The other 4 restarts ('CF', 'D7', 'DF', and 'E7') restart to locations 0008H, 0010H, 0018H and 0020H respectively. These latter locations contain JP instructions which further transfer control to 0C37H, 0C3AH, 0C3DH and 0C40H respectively i.e. these restarts are vectored through locations in the workspace RAM.

On initialisation (RESET or a cold start) all these locations within the workspace contain FFH so that invoking any of the restarts 08H to 20H will cause a 'Break' to occur at the relevant address in the workspace.

However the programmer can amend these locations to contain JP ('C3') instructions to transfer control to various user routines which respond to the various restarts. These routines should issue a RET ('C9') instruction to return control to the instruction immediately following the restart.

The above handling of the Z80 restart instructions gives the programmer a great deal of flexibility and power within his/her program.

3.5 Self Locator.

There is a very small routine within NASMON at location 0032H. It consists of the two one byte instructions: EX (SP),HL and JP (HL).

The effect of a CALL to this location is that, on return from the call, HL is set up with the address of the instruction after the CALL and the original value of HL is on the stack.

This is very useful for writing Position Independent Code since it enables a program to find out where it is located.

Remember that this routine effectively decrements the Stack Pointer by 2 - do not forget to POP HL back off the stack before proceeding.

SECTION 4 WORKSPACE.

NASMON's workspace extends from 0C00H to 0D08H although memory above 0D08H is used dynamically by the Front Panel 'S' command, see Section 1.

On a RESET or a cold start locations 0C00H to 0C27H are initialised from locations 1104H to 11F8H within the monitor, 0C28H through 0C48H are set to FFH and 0C49H through 0C58H are set to 00H. The user is then prompted to enter the address of the required start of text, or default it to 4000H. When this has been done FARG1 (0C59H), FARG2 (0C5BH), MARG1 (0C5DH), MARG2 (0C5FH) and MARG3 (0C61H) are all set to this text-start address. Finally ENDRAM(0C63H) is set to contain the address of the last byte of RAM, assuming that RAM starts at 4000H and is continuous.

The remainder of the workspace is not initialised.

On a warm start (SYS 0 or JP 3) only locations 0C00H to 0C06H inclusive are initialised and they are set from locations 1104H to 110AH within the monitor.

FARG1 through MARG3 are set to the required start of text whenever the editor 'T' and 'U' commands are used.

Details of the contents and use of NASMON's workspace now follow:

Address	Name	Initialised value	Function
0C00H	VLINES	0FH	number of lines on the video screen that will be scrolled. Note that, initially, the top line is not scrolled.
0C01H	VSTART	080AH	address of the 'top' of the video RAM i.e. the address from where scrolling will start - low half of address in 0C01H, high half in 0C02H.
0C03H	PORT0	00H	mask of Port 0. See NASCOM documentation for details of how Port 0 is used. Use SYS 18 to update the port - see Section 3.
0C04H	FLAG	00H	the system flag used by NASMON. The following bits are used: 0 - used by DEOUT and AOUT to determine whether output should go through SYS 1 or be 'poked' to (HL). 2 - 3 used by tape routines to decide whether data should be loaded into its associated address or appended to (TEND). 4 - used in the keyboard routine to determine whether the function of the SHIFT key is inverted (1) or not (0).

Address	Name	Initialised value	Function
0C05H	PAT1L	00H	the length of the last string of characters that were defined by the Front Panel 'S' command, or the length of the current editor Find string - see Section 2.3.
0C06H	PAT2L	00H	the length of the current editor Substitute string - see Section 2.3.
0C07H	CURPOS	080AH	contains the current address of the video cursor.
0C09H	MEMPOS	4000H	contains the address around which the memory display in the Front Panel mode is centred - the Memory pointer.
0C0BH	REGPT	10H	determines which register (PC IY, IX, HL, DE, BC, AF) is addressed by ' ' in the Front Panel display (10H to 02H in twos).
0C0CH	KDELS	00A0H	controls the delay between repeated characters; the lower the value the faster a given character repeats if held down. Modified by CTRLB and CTRLD.
0C0EH	KDELL	0200H	controls the initial delay between pressing a key on the keyboard and having the character repeat - the lower the value the shorter the delay.
0C10H	NEMVEC	C30902H	this location is called every time the Front Panel display is updated. When initialised it simply causes a further jump to location 0209H which contains a RET instruction and so does nothing. However, if NASNEM, our 2½K disassembler, is loaded and has been initialised to interface with NASMON (either via 'W' or by a direct jump) then NEMVEC will route control through NASNEM so that a disassembly of the current instruction appears on the top line of the Front Panel display. The disassembly may be turned off by entering 02H at location 0C12H.

Address	Name	Initialised value	Function
0C13H	CVECT	C30020H	a general purpose vectored jump. Originally set up to enter NASGEN, our 3K assembler. Modified by NASPAS to enter the compiler and compile the text. Accessed via the editor 'J' command.
0C16H	RVECT	C30018H	a general purpose vectored jump. Originally set up to enter NASNEM, Hisoft's 2½K disassembler. Modified by NASPAS so that its use will run the previously compiled object program. Accessed via the editor 'R' command and the Front Panel 'O' command.
0C19H	VIDVEC	C3620DH	the SYS 1 vector. Set to enter OUTCH.
0C1CH	KEYVEC	C32810H	the SYS 2 vector. Set to enter KSCAN.
0C1FH	TPOVEC	C3E207H	the SYS 3 vector. Set to enter TPOUT.
0C22H	TPIVEC	C30010H	the SYS 4 vector. Set to enter TPIN.
0C25H	PRNVEC	C3E007H	the SYS 5 vector. Set to enter PRINCH.
0C28H	IO1VEC	FFFFFFH	the SYS 6 vector.
0C2BH	IO2VEC	FFFFFFH	the SYS 7 vector.
0C2EH	IO3VEC	FFFFFFH	the SYS 8 vector.
0C31H	IO4VEC	FFFFFFH	the SYS 9 vector.
0C34H	IO5VEC	FFFFFFH	the SYS 10 vector. Also used by NASPAS.
0C37H	RSTCF	FFFFFFH	the RST 8 vector.
0C3AH	RSTD7	FFFFFFH	the RST 16 vector.
0C3DH	RSTDF	FFFFFFH	the RST 24 vector.
0C40H	RSTE7	FFFFFFH	the RST 32 vector.
0C43H	NMIEXT	FFFFFFH	the NMI extension vector. Control will be passed here if an NMI is invoked directly i.e. not through bit 3 of Port 0.

Address	Name	Initialised value	Function
0C46H	F7EXT	FFFFFF	the F7 extension vector. Control will be passed here on any SYS (F7) call with an argument in the range 1CH to 7FH inclusive.
0C49H	REGALT	00H	determines whether the standard (if 0) register set is displayed in the Front Panel or the alternate register set (non 0).
0C4AH	KEYMAP	0000H 0000H 0000H 0000H 00H	the keyboard 'map' used to store the current state of the keyboard, row by row.
0C53H	KCOUNT	0000H	used in the keyboard routine to decide when to repeat a key that is held down - a temporary store.
0C55H	BRKADR	0000H	the address at which the last breakpoint was set using the Front Panel 'H' command.
0C57H	INSPOS	0000H	the address within the current textfile at which an 'I'nsert is taking place. Also acts as a flag showing (when non-zero) that Insert mode has been entered.
0C59H	FARG1	4000H	contains the address of the first Find pointer - see 'F' in Section 2.3.
0C5BH	FARG2	4000H	contains the address of the second Find pointer - see 'F' in Section 2.3.
0C5DH	MARG1	4000H	contains the address of the first Move pointer - see 'M' in Section 2.3.
0C5FH	MARG2	4000H	contains the address of the second Move pointer - see 'M' in Section 2.3.
0C61H	MARG3	4000H	contains the address of the third Move pointer - see 'M' in Section 2.3.
0C63H	ENDRAM	end of RAM	contains the address of the last byte of RAM in the system. Assumes that RAM starts at 4000H and is continuous.

Address	Name	Initialised value	Function
0C65H	LASTX		contains the value of the Memory pointer at which the last Front Panel 'X' command was issued.
0C67H	LASTR		contains the value of the Memory pointer at which the last Front Panel 'R' command was issued.
0C69H	TXTPOS		the current text position in the Editor mode i.e. the address of the character that the cursor is currently addressing in the Editor mode.
0C6BH	BUFPOS		the buffer position. When using the Insert mode within the text editor characters are entered into BUFFER and BUFPOS indicates the position within BUFFER. When the text has been emptied from BUFFER into the main textfile then BUFPOS is set to BUFFER + 1.
0C6DH	TEMP		a temporary store used by the string search commands.
0C6FH	TEND		contains the address of the current end of text which is one greater than the address of the last end-of-line character.
0C71H	CTAB		contains the address of the current command table being used by the Editor.
0C73H	NCMD		the number of commands present in the current command table being used by the Editor.
0C74H	BRKVAL		the value that was replaced with FFH when the last Front Panel 'H' command was issued.
0C75H	TSTRT		contains the address of the start of the current textfile.
0C77H	NAME		a 14 character buffer used to hold any string (padded to the right with spaces) entered via the routine TPNAME or the routine GPNAME. Used by tape routines within NASMON.

Address	Name	Initialised value	Function
0C85H	BUFFER		a 130 byte general purpose buffer used by, inter alia, NASMON tape load routine, NASMON Insert text routine and NASPAS.
0D07H	PATERN		a buffer used to store the last string of characters defined by the Front Panel 'S' command. Also used as a temporary store by the tape load routine.
0BDDH	FINDPT		a buffer in the top line of the video RAM used to hold the current Find string - see the 'F' command in Section 2.3.
0BECH	SUBSPT		a buffer in the top line of the video RAM used to hold the current Substitute string - see the 'F' command in Section 2.3.

APPENDIX 1 AN EXAMPLE FRONT PANEL DISPLAY.

```
   → PC  5000   21 00 00 ED 58 00 58 B7
     SP  0FFE   20 47 DB D2 17 30 FB D8
     IX  0000   C3 65 03 C3 94 03 54 6F
     IY  0034   57 69 74 68 E3 28 22 09
     HL  FFFF   7F C3 65 03 C3 94 03 54
     DE  21A5   FF FF FF FF FF FF FF FF
     BC  52CD   18 CA 9C 03 FE 0D 28 04
     AF  3CFF SZ H VNC
4FF6 7E   4FFA ED   4FFE E1   5002 00   5006 58
4FF7 FE   4FFB F1   4FFF C9   5003 ED   5007 B7
4FF8 20   4FFC C1 →5000 21←  5004 58   5008 ED
4FF9 38   4FFD D1   5001 00   5005 00   5009 52
_
_
_
```

Shown above is a fairly typical Front Panel display.

The first 8 lines of the display contain the registers, the name first (PC to AF), then the value presently held by the register and finally the contents of the 8 locations starting from the address held in the register.
The Flag register is decoded to show the flags currently set in the bit order that they are set within the register.

A Register pointer '→' points to the register currently addressed. See Section 1 for details of modifying registers.

The 20 byte memory display below the registers is organised as the address (2 bytes) followed by the contents (1 byte) of the memory at that address. The display is centred around the current Memory pointer, indicated by →←.

Commands (see Section 1) are entered on the bottom 3 lines of the screen in response to the prompt '_'. The display is updated after each command is obeyed.

APPENDIX 2 FORMAT OF TAPE FILES.

Files are output to and read from tape in 128 byte blocks with a header label preceding the data and a trailer label following it. Details are given below:

1. Header label.

4D06OC77FFNNNNNNNNNNNNCC

Key:

4D	start of label.
06	length of 6 character name.
OC77	address of NAME (see Section 4).
FF	signifies header label.
NN..NN	up to 6 character filename, padded with spaces.
CC	checksum.

2. Data block.

4DLLAAAA00DD.....DDCC

Key:

4D	start of block.
LL	length of block, normally 7AH.
AAAA	address in memory of start of data.
00	signifies data block.
DD..DD	up to 122 (7AH) bytes of data.
CC	checksum.

3. Trailer label.

4D00SSSS01CC

Key:

4D	start of label.
00	zero length, no data.
SSSS	the automatic start address, jumped to at end of load.
01	signifies trailer label.
CC	checksum.

APPENDIX 3 - A TEXT CONVERTER FOR NASMON 3 TEXTFILES.

```
Pass 1 errors: 00

                0001 ;Routine to convert a NASMON 3 textfile to a NASMON 4 textfile.
                0002
                0003 ;Execute at 'start' and, when prompted with '$', enter the name
                0004 ; of the old text file and start the tape.
                0005 ;The old file will be loaded and you will then be asked to enter
                0006 ;the start address of the old file. The file will then be converted.
                0007
                0008 ;Equates
                0009
4000            0010 texnew EQU  #4000
000D            0011 endlin EQU  13
00C9            0012 ret    EQU  #C9
0C75            0013 TSTRT  EQU  #C75
0C6F            0014 TEND   EQU  #C6F
04B7            0015 LOAD   EQU  #4B7
0589            0016 SAVE   EQU  #589
                0017
0E00            0018        ORG  #E00
0E00 F70E       0019 start  SYS  14             ;reset scrolling to full screen
0E02 F70F       0020        SYS  15             ;clear screen
0E[illegible]8704   0021        LD   HL,LOAD        ;move the NASMON load routine
0E0[illegible]590E  0022        LD   DE,end         ;to the end of this routine
0E0A 01D200     0023        LD   BC,SAVE-LOAD   ;at 'end'.
0E0D EDB0       0024        LDIR
0E0F 21BC0E     0025        LD   HL,end+#63     ;replace the JP (HL) here
0E12 36C9       0026        LD   (HL),ret       ;with a RET so that the load
0E14 CD590E     0027        CALL end            ;routine may be called.
0E17 21480E     0028        LD   HL,texmss      ;get old text start
0E1A F719       0029        SYS  25
0E1C F713       0030        SYS  19             ;in HL.
0E1E 110040     0031        LD   DE,texnew      ;new text start in DE.
0E21 ED53758C   0032        LD   (TSTRT),DE     ;and in TSTRT.
0E25 23         0033        INC  HL             ;increment over line delimiter
0E26 23         0034 loop   INC  HL             ;increment over
0E27 23         0035        INC  HL             ;the line number.
0E28 010000     0036        LD   BC,0           ;set up BC for the CPIR.
0E2B 3EFF       0037        LD   A,-1           ;search for #FF
0E2D E5         0038        PUSH HL             ;starting at the current text line.
0E2E EDB1       0039        CPIR
0E[illegible]       0040        DEC  HL             ;point to the #FF and
0E31 360D       0041        LD   (HL),endlin    ;replace it with a #0D.
0E33 210000     0042        LD   HL,0           ;make BC positive
0E36 B7         0043        OR   A              ;by subtracting from 0.
0E37 ED42       0044        SBC  HL,BC
0E39 44         0045        LD   B,H            ;BC gives length of pure
0E3A 4D         0046        LD   C,L            ;text + line delimiter.
0E3B E1         0047        POP  HL             ;regain the beginning of the line.
0E3C EDB0       0048        LDIR                ;move the line up
0E3E 7E         0049        LD   A,(HL)         ;and check for the
0E3F 3C         0050        INC  A              ;end of text.
0E40 20E4       0051        JR   NZ,loop        ;no  - do another line...
0E42 ED536F0C   0052        LD   (TEND),DE      ;yes - update TEND
0E46 F700       0053        SYS  0              ;and exit.
                0054
0E48 0D         0055 texmss DEFB 13
0E49 4F         0056        DEFM "Old text start:"
0E58 00         0057        DEFB 0
                0058
0E59            0059 end    EQU  $

Pass 2 errors: 00
```