Simple Demonstration Progams Using NAS-SYS 1.

These programs set out to demonstrate some of the features of NAS-SYS 1. Initially the machine code instructions are explained in detail (with the actual instruction mnemonics bracketed), later the instruction mnemonics only will be used. These simple demonstrations do not try to cover the subject thoroughly but are intended to give some indication as to the use of machine code assembly.

The first program is given as an example of how to overcome one of the slight disadvantages of NAS-SYS. There are two ways of printing a text string on the monitor screen:

1) The normal way using the CRT routines; 'PRS' called using restart 28H which puts the string which follows it on the screen until 'PRS' sees '00' which terminates it. Or 'ROUT', called using restart 30H, which prints the contents of the A register on the screen (in ASCII) each time it is called.
2) Copying characters directly into the video RAM.

In NAS-SYS, the extensive screen editing commands do not allow direct access to line 16 (the top line of the monitor screen) using the normal CRT routines. As the top line is an ideal location for titles etc, addressing the top line must be achieved in some other fashion.

Program 1

In this program a title is copied directly into the video RAM.

```
0C80  3E 0C          Load the accumulator with
                     the code to clear the
                     screen. (LD A, 0CH)

0C82  F7             Call the routine at 30H
                     labelled 'ROUT'. (RST 30H)

0C83  21 8F 0C       Load the HL register pair
                     with the start address of
                     the title. (LD HL, 0C8FH)

0C86  11 D6 0B       Load the DE register pair
                     with the start location on
                     the screen. (LD DE, 0BD6H)

0C89  01 11 00       Load the BC register pair
                     with the length of the
                     title. (LD BC, 0011H)

0C8C  ED B0          Copy the title using a copy
                     instruction. (LDIR)

0C8E  76             Stop the Nascom. (HALT)

0C8F  54 48 49 53    Title as an ASCII string.
0C93  20 49 53 20
0C97  54 48 45 20
0C9B  54 49 54 4C
0C9F  45
```

The next five programs are designed to be built up into one continuous program. Having entered the first program, and learned how it works, the second program is added to it, likewise with the third, etc.

Program 2

Program 2 clears the screen and loads the title in similar way to program 1. There is a difference, as this program does not copy the title directly, instead, each character is copied as before, but a delay (called as a subroutine) is inserted between each character. As the copy routine used is not automatic, checks have to be made to determine when the title is fully loaded.

```
0C80  3E 0C          LD A, 0CH   Load A with a
                     clear screen symbol.

0C82  F7             RST 30H   Print it using
                     restart labelled 'ROUT'.

0C83  21 EE 0C       LD HL, 0CEEH  Point HL to
                     the start of the title.

0C86  11 DE 0B       LD DE, 0BDEH  Point DE to
                     the screen location.

0C89  01 05 00       LD BC, 0005H  Load BC with
                     the length of the title.

0C8C  ED A0          LDI  Copy one character.

0C8E  CD E4 0C       CALL 0CE4H  Call the delay
                     subroutine.

0C91  AF             XOR A  Exclusive OR A to
                     clear it, making it 00.

0C92  B1             OR C  OR C with A. If C was
                     00, then the Z flag is set.

0C93  20 F7          JR NZ, -7  If the Z flag
                     was not set, jump back to
                     0C8CH.

0C95  76             HALT  Stop the Nascom.
```

The next part of program 2 is the delay subroutine, which makes use of the delay in NAS-SYS, labelled 'RDEL', called by restart 38H, followed by the title.

Note that 'RDEL' is 2.5mS when using a 4MHz clock, and 5mS when using a 2MHz clock. When using a 2MHz clock (Nascom 1), the B register should be loaded with 10H (at 0CE7H) to halve the length of the delay loop.

Note that this next part does not follow directly after the above, but must be typed in before program 2 is used.

```
0CE4  F5             PUSH AF  Save the contents
                     of the AF register pair.

0CE5  C5             PUSH BC  Save the contents
                     of the BC register pair.
```

The contents of these registers must be saved, as they contain information to be used later, which would otherwise be destroyed by the subroutine.

```
0CE6  06 20          LD B, 20H  32 times the
                     delay is required, so as
                     a DJNZ loop is to be used
                     B is loaded with 32.
```

```
0CE8  FF            RST 38H  Call the delay
                    routine labelled 'RDEL'.

0CE9  10 FD         DJNZ -1  Decrement B by 1.
                    If B not zero, jump to
                    0CE8H.

0CEB  C1            POP BC  Restore the BC
                    register pair.

0CEC  F1            POP AF  Restore the AF
                    register pair.

0CED  C9            RET  Return from the
                    subroutine.

0CEE  41 20 42 4F   The title as an ASCII
0CF2  58            string.
```

Program 3

In this, the next part of the program we propose to draw a vertical column of X's from a location near the bottom of the screen up towards the top. To do this the 'ROUT' routine will be used, having first located the cursor at the desired position. A 'DJNZ' loop is set up which sequentially prints an X, moves the cursor up to the next line, then prints a backspace.

Note that having printed a character, the cursor is automatically moved on to the next position. Hence the backspace.

This is not the most economic way to construct this routine, but serves by way of demonstration.

```
0C95  21 50 0B      LD HL, 0B50H  Point HL
                    to the cursor position
                    required on the screen.

0C98  22 29 0C      LD (0C29H), HL  Load HL
                    into the cursor store, thus
                    altering the cursor on the
                    screen.
```

This simple little routine may be used at any time to locate the cursor at a desired position on the screen.

```
0C9B  06 0D         LD B, 0DH  Load B with 14
                    as 14 X's are required.

0C9D  3E 58         LD A, 58H  Load A with the
                    code for an X.

0C9F  F7            RST 30H  Call the CRT
                    routine labelled 'ROUT' to
                    print the character.

0CA0  3E 13         LD A, 13H  Load A with the
                    code for a cursor 'up move'.

0CA2  F7            RST 30H  Print it.

0CA3  3E 08         LD A, 08H  Load A with the
                    code for a backspace.

0CA5  F7            RST 30H  Print it.

0CA6  CD E4 0C      CALL 0CE4H  Call the delay
                    subroutine.

0CA9  10 F2         DJNZ -12  Decrement B, if
                    not zero, jump to 0C9DH.

0CAB  76            HALT  Stop the Nascom.
```

Program 4

Program 4 uses the string print routine called by restart 28H, this restart is labelled 'PRS'. The string to be printed is a space followed by an X. As the string is enclosed within a DJNZ loop, and the cursor is not manipulated by an 'up move' command as in the last routine, a horizontal row of X's is printed. Note that as the last screen commands in the previous program were an 'up move' and a backspace, it is appropriate to print one more X before entering the loop. Although this program has much the same effect as the previous program, it is much shorter because of the use of the 'PRS' routine.

```
0CAB  3E 58         LD A, 58H  Load A with the
                    code for an X.

0CAD  F7            RST 30H  Print it.

0CAE  06 10         LD B, 10H  Load B with 16
                    as 16 X's are required.

0CB0  EF            RST 28H  Call the CRT
                    routine labelled 'PRS'.

0CB1  20 58 00      ASCII codes for a space and
                    an X. The 00 tells 'PRS'
                    that this is the end of the
                    string.

0CB4  CD E4 0C      CALL 0CE4H  Call the delay
                    subroutine.

0CB7  10 F7         DJNZ -7  Decrement B. If B
                    not zero jump to 0CB0H.

0CB9  76            HALT  Stop the Nascom.
```

Program 5

The next program is similar to program 3. In fact this prints a second column of X's at the end of the horizontal row of X's. No explanation will be given as this is so similar to the other program.

```
0CB9  06 0D         LD B, 06H
0CBB  3E 14         LD A, 14H
0CBD  F7            RST 30H
0CBE  3E 08         LD A, 08H
0CC0  F7            RST 30H
0CC1  3E 58         LD A, 58H
0CC3  F7            RST 30H
0CC4  CD E4 0C      CALL 0CE4H
0CC7  10 F2         DJNZ -12
0CC9  76            HALT
```

Program 6

The last program in this group gives a good demonstration of the use of the 'PRS' routine. In many ways this is similar to program 4. However, here, the line is printed backwards, using the cursor 'left move' command. Remember that in printing a character the cursor moves one space to the right, hence the three 'left moves' to reach the correct position for the next X.

The small routine at the end is a loop calling the delay subroutine. Note that the delay routine is also a loop. As this is a loop with a lesser loop inside it, it is known as a 'nested loop'. After the delay, the program returns to the start and repeats itself.

```
0CC9  EF            RST 28H  Call the routine
                    labelled 'PRS'.

0CCA  11 11 11 00   ASCII string moving the
                    cursor back three places.

0CCE  06 10         LD B, 10H  Load B with 16
                    as 16 X's are required.

0CD0  EF            RST 28H

0CD1  58 11 11 11   ASCII string of an X then
0CD5  00            three cursor 'left moves'.

0CD6  CD E4 0C      CALL 0CE4H  Call delay.

0CD9  10 F5         DJNZ -9

0CDB  06 10         LD B, 10H  Load B with 16
                    to loop the delay 16 times.

0CDD  CD E4 0C      CALL 0CE4H  Call the delay.

0CE0  10 FB         DJNZ -3

0CE2  18 9C         JR -115  Jump back to start
                    of program, 0C80H.
```

The last four programs give a simple demonstration of the use of NAS-SYS internal subroutines, which are accessed from a table of numbers called by the restart labelled 'SCAL'. To use an internal subroutine the appropriate restart code (in this case 'DF') is followed by the table number. It will be noticed that some of the table numbers are marked 'not normally used', this is because it is usually easier to use the 'Input/Output' restarts (RIN and ROUT).

In the next two programs the Input Output routines are not used (except to print on the monitor screen in one instance), and the functions of 'RIN' and 'ROUT' are replaced by internal subroutine calls from the table.

From now on, the operands of the instruction mnemonics will be replaced by the labels assigned to the operands; thus, RST 30H will be refered to by its label and will be written RST ROUT, likewise defined bytes (DEFB) will be refered to by label, DEFB SRLX means the byte in the table which points to the subroutine labeled SRLX.

Program 7

This little program outputs the characters typed on the keyboard to the monitor screen and the tape recorder. In this way a tape record of what was typed is preserved.

The first thing the program does is to output a string of characters, which when replayed put the Nascom in the 'H' mode. This can be done, as, when the Nascom is waiting for a key press, it is in fact scanning for an input from either the keyboard or the tape recorder. Refer to the descriptions of the subroutines used.

```
0C80  21 8E 0C                LD HL, TABLE  Point
                              HL at the table of
                              characters to be sent
                              out.

0C83  06 06                   LD B, TABLE LENGTH

0C85  DF 6D                   RST SCAL, DEFB SOUT
                              Call SOUT and send
                              the characters.

0C87  DF 7B          LOOP     RST SCAL, DEFB BLINK
                              Call BLINK routine
                              to get a character.

0C89  DF 65                   RST SCAL, DEFB CRT
                              Call CRT to print it.

0C8B  DF 6F                   RST SCAL, DEFB SRLX
                              Call SRLX to send it
                              to the tape recorder.

0C8D  18 F8                   JR LOOP  Jump back to
                              LOOP.

0C8F  0C 45 30 0D    TABLE    Table of characters
0C93  48 0D                   to be sent.
```

Now this routine is very inefficient, as the tape recorder is running all the time, and as minimum speed on the Nascom is about 30 characters a second, a significant improvement in tape economy could be achieved if the message were first stored in the memory then sent to the tape recorder all at once.

Program 8

This program sets B to account for the characters to be sent before the start of the text (the prefix), then points HL at the location where the text is to start. It then enters a loop, first saving HL (as this is lost when getting a character), then checking if the character is an '@'. If an '@' is found then the program branches to 'END'. If the character is not an '@' then the character is printed on the screen. Next HL is restored, and the character saved in memory at the location 'pointed to' by HL (labelled 'BUFFER'). HL is incremented by one, and B is incremented by one. The program then loops back for another character.

When an '@' is encountered, the program branches to 'END'. At this instant, HL is pointing to the the location of the '@' on the screen (a function of 'BLINK'), and B contains a count of the characters (plus 6 for the prefix).

First HL is 'POPped' to 'throw away' the PUSH at 0CE5H. In this program there is no real nesseccity for this, as HL is not required, but as this would leave the stack two down, it is both untidy, and, in a different program, could lead to serious problems. Therefore, the rule; if the stack has been 'PUSHed', and this is later not required, 'throw away' the stack.

The program then outputs a message to the screen reminding you to turn on the tape recorder then waits for a key press before continuing. Routine 'KBD' was chosen for the wait, as using 'RIN' may cause a false start because 'RIN' scans the tape input as well as the keyboard, and the tape recorder may well output a few false characters as it starts up.

Having seen a key press, the routine outputs the characters to the tape recorder using 'SOUT'. When the output is complete, the program outputs a newline to the screen, and returns to the monitor using subroutine 'MRET'.

Note that this routine does not contain any checks as to the quantity of characters stored, and as the program uses 'SOUT' which can only count up to 256, the number of characters should not exceed this ammount (minus 6 for the prefix).

```
0C80  06 06                LD B, TABLE LENGTH

0C82  21 BE 0C             LD HL, BUFFER  Point
                           HL at text space.

0C85  E5            LOOP   PUSH HL  Save HL.

0C86  DF 7B                RST SCAL, DEFB BLINK
                           Call BLINK to get a
                           character.

0C88  FE 40                CP 40H  Compare with
                           the ASCII code for an
                           '@'.

0C8A  28 07                JR Z END  If Z flag
                           set, jump to END.

0C8C  F7                   RST ROUT  Call ROUT
                           to print it.

0C8D  E1                   POP HL  Restore HL

0C8E  77                   LD (HL), A  Save the
                           character at HL.

0C8F  23                   INC HL  Increment HL

0C90  04                   INC B  Increment B

0C91  18 F2                JR LOOP  Go get
                           another character.

0C93  E1            END    POP HL  Throw away
                           stack.

0C94  EF                   RST PRS  Print the
                           following string.

0C95  0D 54 75 72          The message.
0C99  6E 20 6F 6E
0C9D  20 72 65 63
0CA1  6F 72 64 65
0CA5  72 2E 00

0CA8  C5                   PUSH BC  Save BC.

0CA9  DF 61         LOOP1  RST SCAL, DEFB KBD
                           Scan the keyboard for
                           a key press.

0CAB  30 FC                JR NC LOOP1  If no
                           key down, jump back
                           to LOOP1.

0CAD  C1                   POP BC  Restore BC

0CAE  21 B7 0C             LD HL, TABLE  Point
                           HL at the prefix.

0CB1  DF 6D                RST SCAL, DEFB SOUT
                           Send to tape.

0CB3  EF                   RST PRS  Print the
                           following string.

0CB4  0D 00                'newline'

0CB6  DF 5B                RST SCAL, DEFB MRET
                           Call MRET to return
                           to NAS-SYS.

0CB8  0C 45 30 0D   TABLE    The prefix.
0CBC  48 0D

0CBE                BUFFER   Text space.
```

Program 9
=========

This next program gives an insight into decimal to binary conversions, and also demonstrates the more normal use of the 'PRS', 'BLINK' and 'ROUT' routines. The program also uses another routine, 'B2HEX' (refer to the description of this routine).

The program first puts out a message and then scans the keyboard for an input. Having received an input, the character is displayed on the monitor, then converted from ASCII to decimal by the simple expedient of subtracting 30H from it.

In this instance no checks are made to test the validity of the character, which must be a decimal number. In practise this sort of programming is very bad, as invalid inputs should be trapped, and a backspace allowed for correcting the input in the event of a mistake.

Having converted the number from ASCII to decimal, the number is saved in B. The routine is then repeated to get another number which is saved in C. The A register is then cleared, and the B register multiplied by 2 which is done by shifting the binary number left by one bit. The new number formed is added to A. The B register is now multiplied by 4 (two left shifts), and the result again added to A. C is then added to A. The number has now been converted to pure binary.

The contents of the A register are saved whilst a further message is put out, then restored, and A printed using 'B2HEX', followed by a further message. The routine then jumps back to be repeated until terminated by a RESET.

From now on the programs will be printed in a more compact form known as assembley listing.

```
0C80  EF                         RST PRS
0C81  0C 00                      Clear the screen.
0C83  EF                  LOOP   RST PRS
0C84  57 68 61 74 20 69          Message.
0C8A  73 20 74 68 65 20
0C90  6E 75 6D 62 65 72
0C96  20 3F 20 00
0C9A  DF 7B                      RST SCAL, DEFB BLINK
0C9C  F7                         RST ROUT  Print it.
0C9D  D6 30                      SUB 30H  Convert.
0C9F  47                         LD B, A  Save in B.
```

```
0CA0  DF 7B                    RST SCAL, DEFB BLINK
0CA2  F7                       RST ROUT  Print it.
0CA3  D6 30                    SUB 30H  Convert.
0CA5  4F                       LD C, A  Save in C.
0CA6  AF                       XOR A  Clear A.
0CA7  CB 20                    SLA B  Shift left.
0CA9  80                       ADD A, B  Add to A.
0CAA  CB 20                    SLA B  Shift left.
0CAC  CB 20                    SLA B  Shift left.
0CAE  80                       ADD A, B  Add to A.
0CAF  81                       ADD A, C  Add to A.
0CB0  F5                       PUSH AF  Save AF.
0CB1  EF                       RST PRS
0CB2  0D 49 73 20 00           Message.
0CB7  F1                       POP AF  Restore AF.
0CB8  DF 68                    RST SCAL, DEFB B2HEX
0CBA  EF                       RST PRS
0CBB  20 69 6E 20 48 45        Message.
0CC1  58 2E 0D 0D 00
0CC6  18 BB                    JR LOOP
```

Program 10

This program is given as another easy example of arithmetic routines. In this case the two numbers are input to the B and C registers in much the same way as program 9. Hence little explanation is given.

A is then cleared and by manipulation of the bits in the B and C registers, a crude form of binary multiplication is carried out, the answer being accumulated in A. Further manipulation is carried out on the result to convert it back into a decimal number. The result is then printed using 'B2HEX' and the program loops back to the beginning.

```
0C80  EF                       RST PRS
0C82  0C 00                    Clear the screen.
0C83  EF               LOOP    RST PRS
0C84  31 73 74 20 6E 75        Message.
0C8A  6D 62 65 72 20 00
0C90  DF 7B                    RST SCAL, DEFB BLINK
0C92  F7                       RST ROUT
0C93  D6 30                    SUB #30
0C95  47                       LD B, A
0C96  EF                       RST PRS
0C97  20 74 69 6D 65 73        Message.
0C9D  20 32 6E 64 20 6E
0CA3  75 6D 62 65 72 20
0CA9  00
0CAA  DF 7B                    RST SCAL, DEFB BLINK
0CAC  F7                       RST ROUT
0CAD  D6 30                    SUB #30
0CAF  4F                       LD C, A
```

The numbers are now saved in B and C. A is then cleared, and bit 0 in C tested. If the reslt is a 0, the proram jumps forward and shifts B one to the left. If the result was not zero, B is put into A before the left shift. The shifting to the left is equivalent to adding 0's to the partial products in ordinary decimal long multiplication.

```
0CB0  AF                       XOR A  To clear it.
0CB1  CB 41                    BIT 0, C  Test bit.
0CB3  28 01                    JR Z L1  If 0, jump.
0CB5  78                       LD A, B
0CB6  CB 20            L1      SLA B
```

Then the next bit in C is tested, and depending on the result, the partial product is added to A, the process is repeated until all the bits in C (4 bits) are accounted for.

```
0CB8  CB 49                    BIT 1, C  Test bit.
0CBA  28 01                    JR Z L2  Jump if 0.
0CBC  80                       ADD A, B  Add to A.
0CBD  CB 20            L2      SLA B  Shift left.
0CBF  CB 51                    BIT 2, C  Test bit.
0CC1  28 01                    JR Z L3  Jump if 0.
0CC3  80                       ADD A, B  Add to A.
0CC4  CB 20            L3      SLA B  Shift left.
0CC6  CB 59                    BIT 3, C  Test bit
0CC8  28 01                    JR Z L4  Jump if 0.
0CCA  80                       ADD A, B  Add to A.
```

The A register now contains the binary result of the product of the B and C registers. This now has to be converted back into decimal. This is done by successively subtracting 10 (in decimal) from the number and counting the number of 10's until a carry is found. This partial result, which represents the number of '10's' in the number, then has 10 added back to it (as one too many 10's were subtracted in producing a negative result) and is then shifted four places to the left, and the remainder added to it, giving the decimal representation in A. 'B2HEX' is then used to print it.

```
0CCB  06 00            L4      LD B, 0  To clear it.
0CCD  04               L5      INC B  Increase count.
0CCE  DE 0A                    SBC #0A  Subtract 10.
0CD0  30 FB                    JR C L5  If no C,jump.
0CD2  05                       DEC B  Adjust count.
0CD3  CE 0A                    ADC #0A  Add 10.
0CD5  CB 20                    SLA B  Shift left.
0CD7  CB 20                    SLA B  Shift left.
0CD9  CB 20                    SLA B  Shift left.
0CDB  CB 20                    SLA B  Shift left.
0CDD  80                       ADD A, B  Add to A.
0CDE  3D                       DEC A  Adjust count.
```

The A register now contains the number in decimal, which is output to the screen by using 'B2HEX'.

```
0CDF  F5                       PUSH AF  Save AF.
0CE0  EF                       RST PRS
0CE1  20 69 73 20 00           Message.
0CE6  F1                       POP AF  Restore AF.
0CE7  DF 68                    RST SCAL, DEFB B2HEX
0CE9  EF                       RST PRS
0CEA  0D 0D 00                 Message.
0CED  C3 83 0C                 JP LOOP  Jump to
                               start.
```

Remember, care must be taken, as some of the routines modify the registers, and if these are still required, the registers should be saved.

It is hoped that the above examples give some assistance in the use of NAS-SYS internal routines to simplify machine code (or assembly) programming. Almost all the routines accessible from the table may be treated as modules, and the descriptions given elsewhere should be adequate for an understanding of the use of each module.