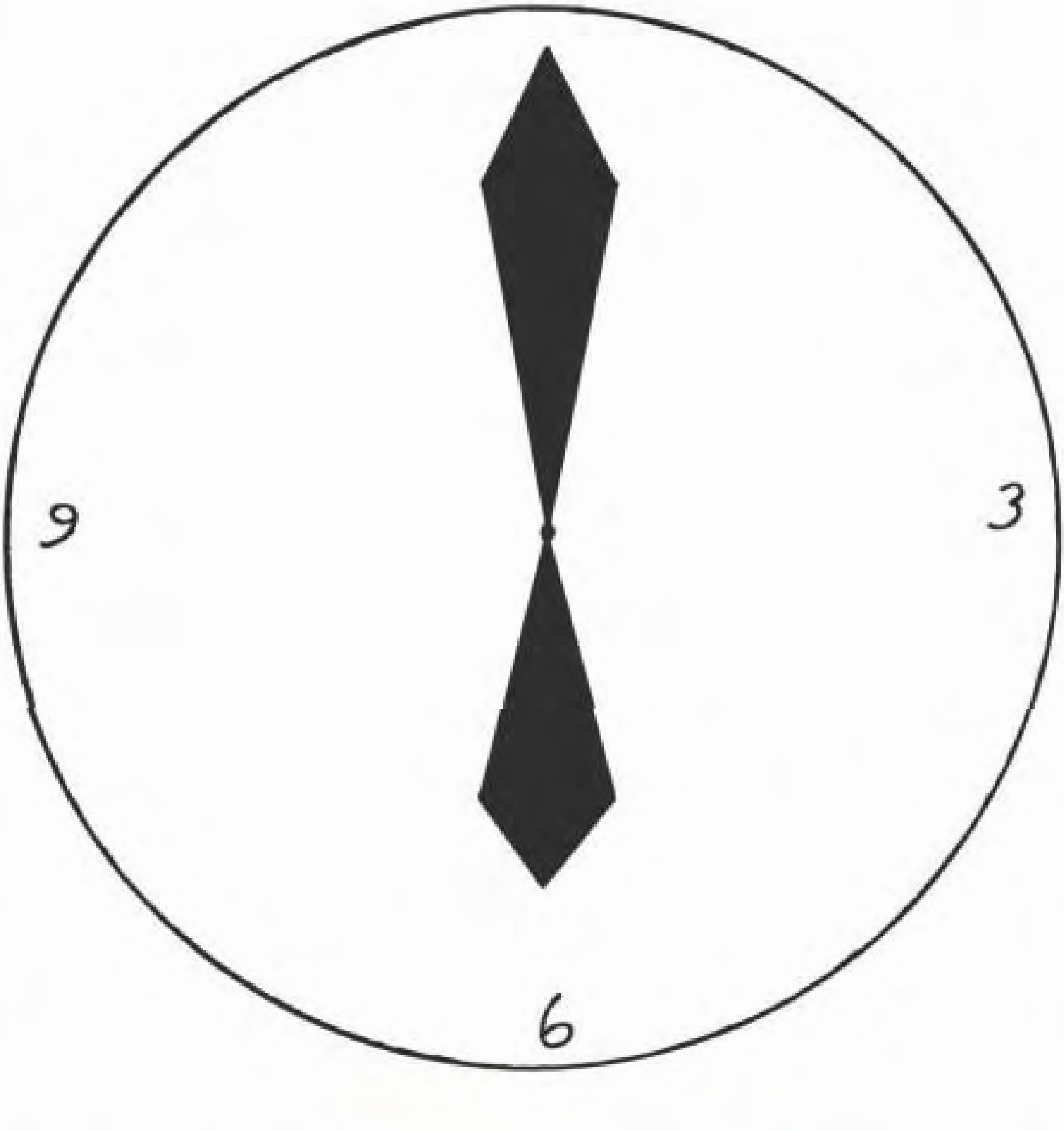
nascom NEWSLETTER

Volume 3, No. 2
May 1983
£1.25



Lucas Nascom

NASCOM NEWSLETTER Vol 3, No 2 May, 1983

Contents

Editor - Ian J Clemmett

Published by - Micro Power Ltd., 8/8a Regent Street,
Chapel Allerton, Leeds LS7 4PE

Printed by - Dataform Press

Editorial

Two issues in pretty quick succession and with a bit of luck this should get me back in step with when the magazines should be getting sent out. I should be able to get another issue out sometime in June and from then on every two months. What an optimist!!!

My heaps of potential articles for the magazine are starting to wane a bit. If anybody out there has got anything that he/she thinks the world should know about, now is the time to send it in.

There are some interesting things in the 'News for Nascom' section of the magazine and I can recommend the keyboard buffer/clock mod. It seems to work well on my system.

Well, until the next issue. ....

IJC

SYS-EX - A 1K extension program for NAS-SYS monitors - Part 2

by David G. Johnson

In the last part we described in outline, most of the features of a 1K extension to NAS-SYS. The monitor extension itself will be printed in full in a later issue. For now, we will continue with a detailed description of the additional keyboard commands provided by SYS-EX.

a
Display arguments

Display the contents of the arguments (ARG1 through ARG10) in the NAS-SYS workspace.

Each argument is displayed as a four digit hexadecimal number. Arguments 1 through 8 are displayed on the first line following the command letter and arguments 9 and 10 are displayed on the second line following the command letter.

b AAAA BB
Tabulate memory in double byte format

Tabulate the contents of memory in double byte, four digit values.

The contents of any specified portion of memory are displayed. Each pair of consecutive bytes is treated as representing a four digit hexadecimal number. The format of the display is similar to that used in the 'a' command, with 8 four digit numbers displayed on each line.

AAAA - memory address at which the tabulation is to start.

BB - length of required tabulation. (i.e. the number of double byte values). If more than two digits are entered, only the final two are used.

The 'a' and the 'b' commands share common processing which relies heavily on the NAS-SYS routine TBCD3.

c AAAA BBBB CCCC
Swap the contents of two specified areas of memory

The values which are present in two specified portions of memory are exchanged.

This command is similar to the NAS-SYS 'C' (Copy) command in that the arguments used define the same values. However, instead of the 'source' area just overwriting the 'target' area, the source is moved to the target and the target is simultaneously moved to the source.

The start address of each portion of memory and the length of memory to be exchanged are supplied with the command. Providing that the two specified portions of memory do not overlap, the operation of the command is straightforward.

Should the two specified portions of memory overlap, the effect of the command is to 'rotate' the contents of a section of memory. The memory section which is rotated is determined by: (i) the lower of the two start addresses, (ii) the higher of the two start addresses, plus the specified length, minus one.

The memory addresses at the two limits are included in the rotation. Rotation occurs in a downwards direction with the overflow from the lowest address being wrapped around to the highest address.

AAAA - start address of the first portion of memory.
BBBB - start address of the second portion of memory.
CCCC - length of memory (bytes) to be exchanged.

d -AAAAA

Decimal to hexadecimal conversion

Converts a decimal integer in the range minus 32768 to plus 65535 to hexadecimal.

If the number to be converted is negative, a minus sign should be typed immediately before the first digit of the decimal number. The number itself may only contain the digits 0 through 9.

The result of the conversion to hexadecimal is displayed on the first screen line following the command letter. The result is always displayed as a four digit hexadecimal number. In addition, if the value can be represented by a single byte, a two digit hexadecimal number is also displayed on the same line. Note: If a positive value is entered, it should be remembered that hexadecimal equivalents which start with a digit greater than 7, may also be interpreted as negative values.

The decimal number to be converted is supplied on the same line as the 'd' command. The value may be preceded by spaces and is terminated either by the first following space or by the end of the line, whichever occurs first.

Values from the NAS-SYS arguments are not used by this command. Also, values which are entered on the command line are not placed into the NAS-SYS arguments (ARG1 through ARG10). However, NUMV is set to the hexadecimal value and NUMN is set to the number of decimaligits entered.

-AAAAA - decimal number for conversion in the range -32768 to 65535

The actual conversion is quite straightforward. The routine examines the decimal digits from left to right. The result register is initially set to zero. The next digit is then added to the 'result' and, unless it is the rightmost digit, a copy of the 'result' is added to the 'result' a further nine times (i.e. multiplication by ten). This process is repeated until all of the input digits have been processed. As the arithmetic is all done in hexadecimal, the result is in hexadecimal.

e AAAA BBBB
Erase a byte

Delete a single byte at a specified memory location and move the contents of higher memory locations down to fill the gap.

The section of memory delimited by: (i) address for deletion plus one, (ii) top limit for move minus one, is moved to the memory locations delimited by (i) address for deletion, (ii) top limit for move minus two. The 'top limit for move' parameter is the first memory address from and above which memory is neither read from nor written to by this command. The top limit for move must be greater than the address for deletion by at least two bytes.

After deletion of the specified byte, the command responds by displaying on the following screen line the hexadecimal value of the deleted byte.

AAAA - memory address at which the byte is to be deleted.
BBBB - top limit for move. Memory addresses from this address upwards are not accessed.

The command uses the Z80 block transfer instruction LDIR to shift the contents of memory downwards.

f AAAA
---- find mask ----
Find

Find a specified character string, or a sequence of bytes specified by hexadecimal values, and display the memory address(es) at which it occurs.

The memory address at which the search is to start is taken from the first argument (ARG1).

After entry of the 'f' command, the cursor is positioned at the start of the next line and the find mask may then be entered in one of the following two formats.

1. Character string

Theirst position in the mask line must contain the double-quote character ("). The required character string must start in the second position in the line and may contain any characters (including spaces) except the double-quote character. The string is terminated by a further double-quote character or, if this is not present, the end of the line.

2. Sequence of bytes specified by hexadecimal values.

The mask is entered as a sequence of single byte hexadecimal numbers (0 to FF), each one separated from the next by at least one space. The first value may start in the first position on the line or may be preceded by as many spaces as are required. The entire find mask must be contained within a single line.

After entry of the mask, memory is searched from the specified start address up to the top end of memory (FFFFH). The response is always displayed on the screen line following the entered mask. If no match is found, the response is XXXX.

If between one and eight matches are found, the memory address of the start of each match is displayed in the response line. After the final match, XXXX is displayed to signify that the search continued to the top end of memory. If nine matches are found, all nine memory addresses are displayed in the response line. After nine matches the command terminates without searching any further.

AAAA - memory address at which the search is to start.

The logic of this command is really quite involved. A full description would require an article on its own. Basically, the command obtains the first byte in the required mask and searches through memory until it finds a match. The second byte of the mask is then compared with the next location in memory; if that matches, the third byte of the mask is compared and so on. If the 'n'th byte fails to match, the command continues with searching for the first byte at the memory location one beyond the previous first byte match. Clearly, the command will for most of it's time be searching for and failing to find a match on the first byte in the mask. This matching exercise on the first byte in the mask was therefore made as quick as possible, in order to minimise the execution time of the command. However, this approach does mean that if the first byte of the mask occurs frequently in memory, the command will take a little longer. On average, the command will search 64K in a couple of seconds. However, if the first two bytes of the mask are -say- 00 00 (00 usually seems to occur frequently) it may take up to ten seconds to search 64K. Times are for a processor running at 4MHz with wait states.

h AAAA

Hexadecimal to decimal conversion

Converts a hexadecimal integer value in the range 0 to FFFF, to decimal.

The result of the conversion to decimal is displayed on the first screen line following the command letter. The hexadecimal number is first displayed as it's signed decimal equivalent with a preceding minus sign if the value is negative. If the number is negative, a further value is displayed to show the positive decimal equivalent of the unsigned format hexadecimal number.

In order to determine the sign of the input argument, the command uses the number of input digits typed to determine whether the argument represents a single or a double byte value. If the number of digits typed is greater than two, a double byte value is assumed, otherwise a single byte value is assumed. If required, leading zeroes may be typed in order to force a double byte conversion.

AAAA - hexadecimal value for conversion

Rather than just describe the operation, a commented assembler listing of the conversion follows. The actual SYS-EX code is slightly modified to cater for negative numbers.

```
         LD   HL,(NUMV)   value to be converted
         LD   A,20H       space character in A
         OR A             carry flag off
         PUSH  AF         space character on stack
         LD   DE,000AH    ten in DE (and E)
      L1 LD BC,FFFFH      minus one in BC
      L2 INC  BC          ) keep subtracting ten from
         SBC  HL,DE       ) HL until value becomes
         JR NC,L2         ) less than 0.    BC=HL/ten

         LD    A,E        ) remainder of HL/ten in A
         ADD   L          )      ADD  30H turn into ASCII code
         PUSH  AF         character (A) on stack
         LD   H,B         ) divide by ten figure
         LD   L,C         ) in HL

         LD   A,H         ) loop round again until
         OR   L           ) HL is 0000H. (OR sets
         JR   NZ,L1       ) the carry flag off)

      L3  POP  AF         character to print in A
         .ROUT            print character from A CP 20H Z flag if
                          final space
         JR NZ,L3         loop until all printed
```

i AAAA BBBB CC
Insert a byte

Insert a single specified byte at a specified memory location and move the original contents of that address and above, upwards in memory to make space for the insertion.

The section of memory delimited by: (i) address for insertion (ii) top limit for move minus two, is moved to the memory locations delimited by (i) address for insertion plus one, (ii) top limit for move minus one. The specified byte for insertion is then moved to the insertion address. The 'top limit for move' parameter is the same as defined for the 'e' command.

AAAA - memory address at which the byte is to be inserted.

BBBB - top limit for move. Memory addresses from this address upwards are not accessed.

CC - hexadecimal representation of the byte to be inserted.

The command uses the Z80 block transfer instruction LDDR to shift the contents of memory upwards.

k
Accept input from keyboard only

Alter the address of the NAS-SYS input table such that only keyboard input is permitted. That is, input from the serial input port is not accepted after entry of this command.

The command eliminates the spurious input of random characters which is often associated with the switch on and switch off of attatched tape recorders. Both the NAS-SYS and the SYS-EX read and verify commands (and also NAS-SYS 1 Load command) are unaffected in their operation by the action of the 'k' command. If required, the normal NAS-SYS tables may be restored by using the NAS-SYS 'N' command.

The command adds 4 to the 'normal' input table value and then stores this value using the NAS-SYS NIM function. This has the desired effect both in NAS-SYS 1 and NAS-SYS 3.

l

--- label name to be written ---

Label a cassette tape

Writes a tape label with a specified name. Once written a tape label will be recognised and displayed by the SYS-EX commands 'q', 'r', 's' and 'v'.

After entry of the 'l' command, the prompt 'lName:' appears on the next screen line and the required label name is entered on that line. The tape drive LED is switched on and after about two seconds (4MHz), the tape label is sent to the serial output port for writing to tape. Finally, the tape drive LED is switched off.

The format of the tape label is as follows;

4 x D3H
0 to 42 characters of label name
1 space (00H if name length is 42)
00H.

File and label names are normally entered on the same line as the prompt '.Name:' which immediately follows the entered command line. However, it is quite possible to enter a name from a different line after first moving the cursor to the line required. The name must begin in column 7 and must be immediately preceded by the : character (7CH = CONTROL/SHIFT/,). Any characters (including spaces) may be included in file and label names. All space characters which appear to the right of the final non-space character are ignored.

If the CLOAD command in the 8K Basic causes a tape label to be read, Basic will interpret the label as a program file with the name D3H (a block graphics character). This is unlikely to cause any problems.

n

Return to NAS-SYS

Returns exclusively to NAS-SYS using a NAS-SYS MRET instruction.

This command will seldom be required as all of the NAS-SYS keyboard commands are available in the normal way from within SYS-EX.

Another Clock

By Rory O'Farrell

The article by Mr. G. Kirby in Micropower Vol 2 No 3 on a Real Time Clock was most interesting. As readers buses are getting more and more populated, it might be advisable to consider adding an output buffer to his circuit, as the 58174 chip does not have the high output of a bus driver chip. Those who are daunted by the prospect of wiring up a board to plug into the bus might like to try another approach.

This approach is to hang the 58174 on the PIO, in a very similar circuit. Due to the slow speed of the RTC chip, it is possible to drive it from the port lines, at the expense of rather complicated software, although it is now possible to use the clock on an unexpanded Nascom. I have been using this circuit for nearly a year and have modified much of my software to read the clock.

The circuit diagram is simple enough, but a few points might bear comment. The CS (Chip Select) line is inverted using a CMOS inverter, which is also driven from the standby battery in power down mode. The effect of power down on the chip (without the inverter) seems to be to place a low level on CS, NRDS and NWDS. For some reason, this results frequently in loss of time. Adding the CMOS inverter means that the external low level of the PIO on power down is transmitted to the 58174 chip as a high level, not enabling the chip, thereby preventing time loss. In addition to the three control lines I have detailed, there are four address lines and five data lines. These are connected up to the two ports of the PIO.

To Port A are connected lines as follows:
- Bit A0 - DB0 pin 7
- Bit A1 - DB1 pin 6
- Bit A2 - DB2 pin 5
- Bit A3 - DB3 pin 4
- Bit A4 - pin 13 (interrupt line)

To Port B are connected
- Bit B0 - AD0 pin 12
- Bit B1 - AD1 pin 11
- Bit B2 - AD2 pin 10
- Bit B3 - AD3 pin 9
- Bit B4 - NWDS pin 3 (Write Strobe)
- Bit B5 - NRDS pin 2 (Read Strobe)
- Bit B6 - CS pin 1 (inverted Chip Select)

I do not give the connections for the Nascom, as they differ between N1 and N2s. They can easily be read off the circuit diagrams.

CMOS is sensitive to certain powerdown conditions, which give rise to the need for the transistor regulator drawn by Mr. Kirby. It is out of specification to connect voltage levels to inputs of CMOS chips which are more than 0.5 volts above the power supply levels of the chip. Doing this can cause the chip to latch up, with consequent data loss. That is why the need for the transistor circuit arises. A simpler method is shown in my circuit. The Diode D1 can be a Germanium OA45 or similar, which can offer a Vf(forward voltage drop) of approx 0.5 volt. Unfortunately, such a diode will have an Ir(reverse leakage) of typically 50uA. This leakage current is some five to ten times the power down supply current of the 58174. In consequence, the battery only lasts 1/6th as long as it should (approx three weeks opposed to 4 - 6 months). Replacing the germanium diode with a 1N4148 or similar may give the latch up problem, as Vf over such a diode is usually 0.7volts. The reverse leakage is only 0.025uA, so we don't affect our power consumption dramatically. Electrovalue list a Silicon Schottky Barrier Diode BAS 70-03 with Vf = 0.410 volts and Ir < 200 nA (0.0002uA). This diode costs a matter of pence, and cures the problem of latch up, providing a very simple circuit.

It is possible to obtain a very neat Nicad battery, listed in several suppliers catalogues as PCB Battery, which makes a very compact power supply for battery back up. The whole unit can be made on a 2" square Veroboard, and works most reliably. The variable capacitor is in theory capable of adjusting the timekeeping of the clock to allow for variations in crystal and temperature, but can be omitted, if you don't mind resetting the clock every few months to adjust the seconds. I must admit that inspite of having the variable capacitor, I've not been able to adjust my clock to keep pace with the pips on the BBC. It runs either two secs a day fast or slow, and I can't get any adjustment between those limits. If rebuilding the unit, I'd be very tempted to omit the variable capacitor in the interests of constructional simplification. One very important point - the power supplies for the 4049 go to pin 1 (+5v)[pin one] and pin 8 (0v)!

The software consists of setting up address and control lines on one port, reading the data in on the other, and resetting the control lines. As I list it, the software is rudimentary and extremely inelegant, but it will serve to show that your board works. It sets the clock with the time stored in memory at 0D80H, and using the routine at 0D00H causes the clock to be read and the time printed on the screen. This routine is no practical use, as it prints the information obtained from the clock as a string of figures. If you E0D00 twice in succession, you will see the time change on the righthand side of the strings. I leave the organisation of your own reading routines to you. In assembler listings, I print date and time at the head of each page. The time is relevant there, as one may have a few printouts in the course of an evening, and the time serves to distinguish between them. On the other hand, my word processor requires only the date and perhaps day of week. I intend having only one routine to read the clock, which will form part of a set of extended monitor routines. Any program requiring clock data will call this routine, and from its information select the relevant data. When the chip changes (ticks?) between two reads, the register next read is set to 0FH, to indicate faulty data. In this case, it is necessary to reread the clock completely. National Semiconductor recommend that you should wait for a 0FH signal from the clock before embarking on a full read. The routines listed here print 'FAIL' whenever a read fails for this reason, and proceed to try again. The decision on how best to cope with this I leave to you! Setting up a version of the same routines to read the clock and print the time on the screen indicates that a read takes approx 1 mS.

In using this chip, a useful adjunct is to acquire a copy of the National Semiconductor data sheet, which gives an overall view of the intricacies of the chip, in particular use of the interrupt. I've not achieved 100% reliability in my use of the interrupts and have therefore disregarded them in the interests of getting on with things, but I feel sure that this circuit will allow them to be used reliably. The software is probably capable of considerable improvement, but as listed here it works on a 4 Mhz machine. 2 Mhz may cause problems, as the width of the NRDS (Read strobe) should be limited to 15uS max according to the datasheet.

I am very happy with this chip and recommend it without reserve in the circuit attached. If you are into Clocks, the May '82 BYTE, in Ciarcia's Circuit Cellar, shows a circuit for another NS chip. This chip is the MM58167A, which is slightly more expensive than the 58174, but offers a number of extra facilities. It will time from 1/10000 secs up to months, but knows nothing about leapyears. In addition, it has a low power interrupt mode, allowing it when powered down to signal an interrupt when a specified time and date is reached. This could offer the possibility of your machine being able to turn itself on on Tuesdays at 2.30 p.m. to do whatever you wish! I intend to build a number of specialised boards for my Nascom, and am seriously considering incorporating the 58167A into one of them.

```
        TITLE'CLOCK DRIVER '
                CRT:     EQU  65H
                CRLF:    EQU  6AH
                MRET:    EQU  5BH
                B2HEX:   EQU  68H
                CLKREG:  EQU  05H  ;Clk regs lines on port B
                SPACE:   EQU  69H
                CLKDAT:  EQU  04H  ;Clk data lines on port A
                CR:      EQU  0DH
                 BITMOD: EQU  0FFH ;Token to set PIO
                 INMSK:  EQU  1FH;Bottom five lines are input
                 OUTMSK: EQU  10H ;Interrupt line is output
```

```
                              ORG   0CB0H ; Set clock with Data
                              LOAD  0CB0H ;Stored at 0D80H

0CB0 3EFF        SETWR:       LD    A,BITMOD;Set PIO to write time
0CB2 D306                     OUT   (CLKDAT+2),B
0CB4 3E10                     LD    A,OUTMSK
0CB6 D306                     OUT   (CLKDAT+2),A
0CB8 3EFF                     LD    A,BITMOD
0CBA D307                     OUT   (CLKREG+2),A
0CBC 3E00                     LD    A,0        ;Set output masks
0CBE D307                     OUT   (CLKREG+2),A
0C90 1E00                     LD    E,0
0C92 3E00                     LD    A,0 ;Initialise REG 0
0C94 CD2B0D                   CALL  WRITE
0C97 DD21800D    MAIN:        LD    IX,DATA
0C9B 3E04                     LD    A,4;Register 4 = units of minutes
0C9D 060C                     LD    B,12       ;12 more registers
0C9F DD5E00      MNLP:        LD    E,(IX+0)   ;Get time
0CA2 CD2B0D                   CALL  WRITE
0CA5 DD23                     INC   IX
0CA7 3C                       INC   A
0CA8 E60F                     AND   0FH
0CAA 10F3                     DJNZ  MNLP
                 ;Now we wait for the exact time to
                 ;start clock
0CAC EF                       PRS
0CAD 48697420                 DEFB 'Hit any key to continue',CR,0
0CB1 616E7920
0CB5 6B657920
0CB9 746F2063
0CBD 6F6E7469
0CC6 CF                       RIN
0CC7 3E0E                     LD    A,14 ;Select register 14
0CC9 1E01                     LD    E,1 ;Signal to start clock
0CCB CD2B0D                   CALL  WRITE ;Start clock
0CCE 1830                     JR    SETRD;Jump to read clock rout.

                 ;
                 ;
                              ORG   0D00H ;Pick easy add. to remember
                              LOAD  0D00H ;E 0D00 to read time

                 ;Set up ports - won't always be setting
                 ;clock before read.

0D00 3EFF        SETRD:       LD    A,BITMOD
0D02 D306                     OUT   (CLKDAT+2),A
0D04 3E1F                     LD    A,INMSK
0D06 D306                     OUT   (CLKDAT+2),A
0D08 3EFF                     LD    A,BITMOD
0D0A D307                     OUT   (CLKREG+2),A
0D0C 3E00                     LD    A,0        ;All lines outputs
```

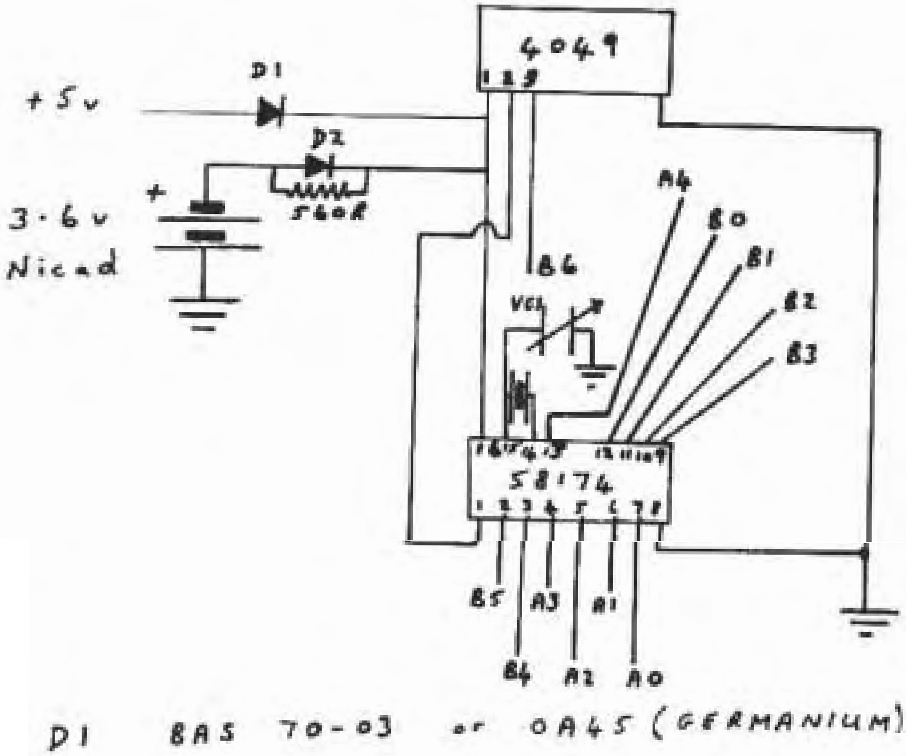
```
0D0E D307                   OUT   (CLKREG+2),A
0D10 3E01        RESTART:LD     A,1;Set for initial register
0D12 CD420D      RDLP:      CALL READ ;Value returns in E
0D15 47                     LD     B,A ;Store value
0D16 7B                     LD     A,E
0D17 E60F                   AND    0FH;Mask off strobes int. line
0D19 FE0F                   CP     0FH;If 0FH, clock has ticked
                 ;since last register read. Print
                 ;'Fail' and start again
0D1B 283E                   JR     Z,RESTART1
0D1D F630                   OR     30H ;Turn into ASCII
0D1F DF65                   SCAL CRT ;Print value
0D21 78                     LD     A,B        ;Get count back
0D22 3C                     INC    A ;Point to next register
0D23 E60F                   AND    0FH
0D25 20EB                   JR     NZ,RDLP
0D27 DF6A                   SCAL CRLF
0D29 DF5B        EXIT:      SCAL MRET

0D2B 0E04        WRITE:  LD     C,CLKDAT;Set up data on port
0D2D ED59                   OUT    (C),E
0D2F F670                   OR 40H+20H+10H;Select chip, but
                 ;turn off strobes
0D31 D305                   OUT    (CLKREG),A
0D33 CBA7                   RES    4,A ;Turn on WR
0D35 D305                   OUT    (CLKREG),A
0D37 CBE7                   SET    4,A ;Turn off WR
0D39 D305                   OUT    (CLKREG),A
0D3B CBB7                   RES    6,A ;Deselect chip
0D3D D305                   OUT    (CLKREG),A
0D3F E60F                   AND    0FH;Restore reg no, no strobes
0D41 C9                     RET

0D42 0E04        READ:      LD     C,CLKDAT;Set C for data port
0D44 F670                   OR 40H+20H+10H;Select chip, no strobe
0D46 D305                   OUT    (CLKREG),A
0D48 CBAF                   RES    5,A ;Turn on RD
0D4A D305                   OUT    (CLKREG),A;Set up register add.
0D4C ED58                   IN     E,(C)      ;Get data
0D4E CBEF                   SET    5,A ;Turn off RD
0D50 CBB7                   RES    6,A ;Turn off chip
0D52 D305                   OUT    (CLKREG),A
0D54 F5                     PUSH AF
0D55 7B                     LD     A,E;Mask off top 4 bits in E
0D56 E60F                   AND    0FH
0D58 5F                     LD     E,A
0D59 F1                     POP    AF
0D5A C9                     RET

0D5B EF          RESTART1:PRS ;Here if clock has ticked
0D5C 4641494C               DEFB 'FAIL',CR,0
0D60 0D00
0D62 C3100D                 JP     RESTART
                            ORG    0D80H
```

```
                             LOAD 0D80H
                   DATA:
0D80 0804          MINS:     DEFW 0408H      ;48 minutes
0D82 0102          HOURS:    DEFW 0201H      ;21 hours
0D84 0802          DAYS:     DEFW 0208H      ;28 th day
0D86 06            DYOFWK:   DEFB 6 ;Saturday. Monday = 1
0D87 0800          MONTH:    DEFW 0008H ;August
                   ;
                   ;Leapyear=8
                   ;Leapyear+1=4
                   ;Leapyear+2=2
                   ;Leapyear+3=1
0D89 02            YEAR:     DEFB 2
0D8A 00000000                DEFB 0,0,0,0,0,0,0,0;Just in case of
0D8E 00000000
                   ;overrun of counts
                             END
```



D1 BAS 70-03 or OA45 (GERMANIUM)

D2 1N4148

VC1 5-50pF

Xtal 32·76 KHz

COMPASS ADDITIONS

Extra Commands and Opcodes for LEVEL 9 COMPression ASSembler.

by Alan Marshall.

When I received the COMPASS Compression Assembler from Level 9 Computing of High Wycombe, I was disappointed to find that RCAL and SCAL were missing as pseudo-opcodes. Having used the assembler for a while and got used to it, I have been delving into the way it works. There are three jumps, at £0F00, £0F03 and £0F09, which can be intercepted and I have used the last two of these to add the two 'missing' pseudo-ops.

While looking for a way to use these jumps I came across the command table and, as two commands are duplicated in version 1.3 I decided to use those positions in the table to add two new commands.

As I have a Centronics printer which has a parallel interface, the print routine has to be via the NAS-SYS User routine. This means that the instructions to the assembler appear on the printout and, while not a disaster, this is something that I considered unnecessary.

The 'P' command is the first command that I added and operates by initialising the User routine and ports, setting the Height command to print the full listing and jumping to the assembler's own Assemble command. If you wish, you can set the Height command to a page length in line 21 for manual form feeds, or have them in your print routine though, I believe, the Epson printers have automatic form feeds.

The other thing that has often been a source of annoyance has been the fact that, to obtain part of a listing, it has been necessary to print the whole. This is changed in this command between lines £6B and £76, by using a tick in the comments column to control the switching on and off of the printer.

Once the listing has been completed the printer needs to be switched off and the 'N' command is used to do this, as well as resetting the Height command to 5.

These are simple changes since all that is done is the changing of the table addresses of redundant commands to the addresses of the new commands.

It is advisable to check that the table addresses in your version of the assembler are the same. If not, then delete the last ten lines of the listing as they will change the addresses to those of new commands, and change them manually. All addresses within the assembler are given relative to the start of the assembler so that setting the address of your copy of COMPASS in line 1 will ensure that all subsequent addresses are correct. The additions are £D5 bytes long so you will probably have to move your copy to fit the extra code at the end of it. This is advisable as the last two lines of the listing alter the length of code checked after a warm start, so that the new code is protected by a checksum, as well as the original.

The two new psuedo-ops use the CODE and BADOPC jumps to break into the program and use similar legal codes for the assembler to process.

When the assembler comes across a bad opcode, it jumps to £0F06 where it is redirected to NEWOPS. Here a check is made of the opcode and, if neither RCAL nor SCAL, jumps back to the BADOPC address in the assembler. If either is found then the first character is stored and the compressed code for either JR or SUB are substituted for processing. The reasons for these two being chosen are that they are single codes which are followed by appropriate operands. RCAL is similar to JR in operation and SCAL, like SUB, is followed by a single operand which must be £FF or less. Note that no check is made for illegal SCAL values.

This is all that is done on the first pass, but on the second a check has to be made of the 'N' option as the CODE jump is only checked if the code is to be stored. If the option has been taken then it is reset and a flag set so that the break can be made at the CODE jump.

The CODE jump at £0F03 is diverted to SUBSTITUTE and the first check is whether it is one of the new codes that is being loaded. If not, then the program returns to the assembler. At this point registers A and B contain the opcode to be loaded and this is changed to either £D7 or £DF as appropriate before allowing the program to continue. If the 'N' option was taken then this is reset and the return to the assembler avoids that part where the opcode is loaded.

That completes the alterations to the program operation and if the program is loaded as in the listing then only one thing remains to be done. This is the alteration of the checksum. The length of the program to be checked has already been altered but, of course, the checksum, which is stored after the last byte of the program, will be incorrect. The way to find out the new checksum is to set a breakpoint at COMPASS+£0409 and then warm start the assembler. When the breakpoint is reached HL will point to the checksum store (+£1A19 in the listing) and the accumulator will contain the checksum. As COMPASS+£0409 contained £E7 instead of £BE when it was checked, the checksum in the accumulator will be £29 greater than it should be, so enter the accumulator value minus £29 in the checksum store and save the program, not forgetting to include the checksum.

As you now have a rough idea of how COMPASS can be modified, you could add other commands, RIN, ROUT and RDEL should be simple as they are single opcodes with no operands, but PRS will need more thinking about.

```
E500            ; Extra Commands for COMPASS Assembler.
E500            ;
E500            ; 'P' turns printer on for listing.
E500            ; 'N' turns printer off.
E500            ;
E500            ; Extra Pseudo Ops.
E500            ; RCAL & SCAL as NAS-SYS
E500            ;
E500            COMPASS EQU £E500
E500            RCAL    EQU £18
E500            ZRCAL   EQU £D7 ; Z80
E500            ZSCAL   EQU £DF ; Z80
E500            JR      EQU £D9 ; COMPASS
E500            SUB     EQU £FE ; COMPASS
E500            PAGSIZE EQU £0F7A
E500            CURPAG  EQU £0F7C
E500            PAGNUM  EQU £0F8A
E500            CURLOC  EQU £0F8B
E500            OPTCOD  EQU £0F90
E500            RETURN  EQU COMPASS+£000
E500            ERROR   EQU COMPASS+£0CB0
E500            CONTINU EQU COMPASS+£0CA8
E500            ASSEMBL EQU COMPASS+£0B53
E500            OUTPUT  EQU COMPASS+£12B8
E500            ;
FE7E             ORG COMPASS+£197E
FE7E 211DFF     PCOMMAND LD HL,PRINT
FE81 22780C      LD (£0C78),HL ; set up user routine
FE84 3E0F        LD A,£0F ; initialise ports
FE86 D306        OUT (6),A
FE88 C670        ADD A,£70
FE8A D307        OUT (7),A
FE8C 210000      LD HL,0 ; set Height command to zero
FE8F 227A0F      LD (PAGSIZE),HL
FE92 227C0F      LD (CURPAG),HL ; also current age
FE95 DF          RST SCAL
FE96 55          DEFB 'U'
FE97 C353F0      JP ASSEMBL
FE9A DF         NCOMMAND RST SCAL
FE9B 4E          DEFB 'N'
FE9C 210500      LD HL,5 ; reset Height command
FE9F 227A0F      LD (PAGSIZE),HL
FEA2 C1          POP BC ; reset SP
FEA3 C30FE5      JP RETURN ; await next command
FEA6 FE52       NEWOPS CP 'R' ; from BADOPC (0F06)
FEA8 2805        JR Z,NEWOPS1 ; check for R/SCAL
FEAA FE53        CP 'S'
FEAC C2B0F1      JP NZ,ERROR ; CA10 Bad Opcode
```

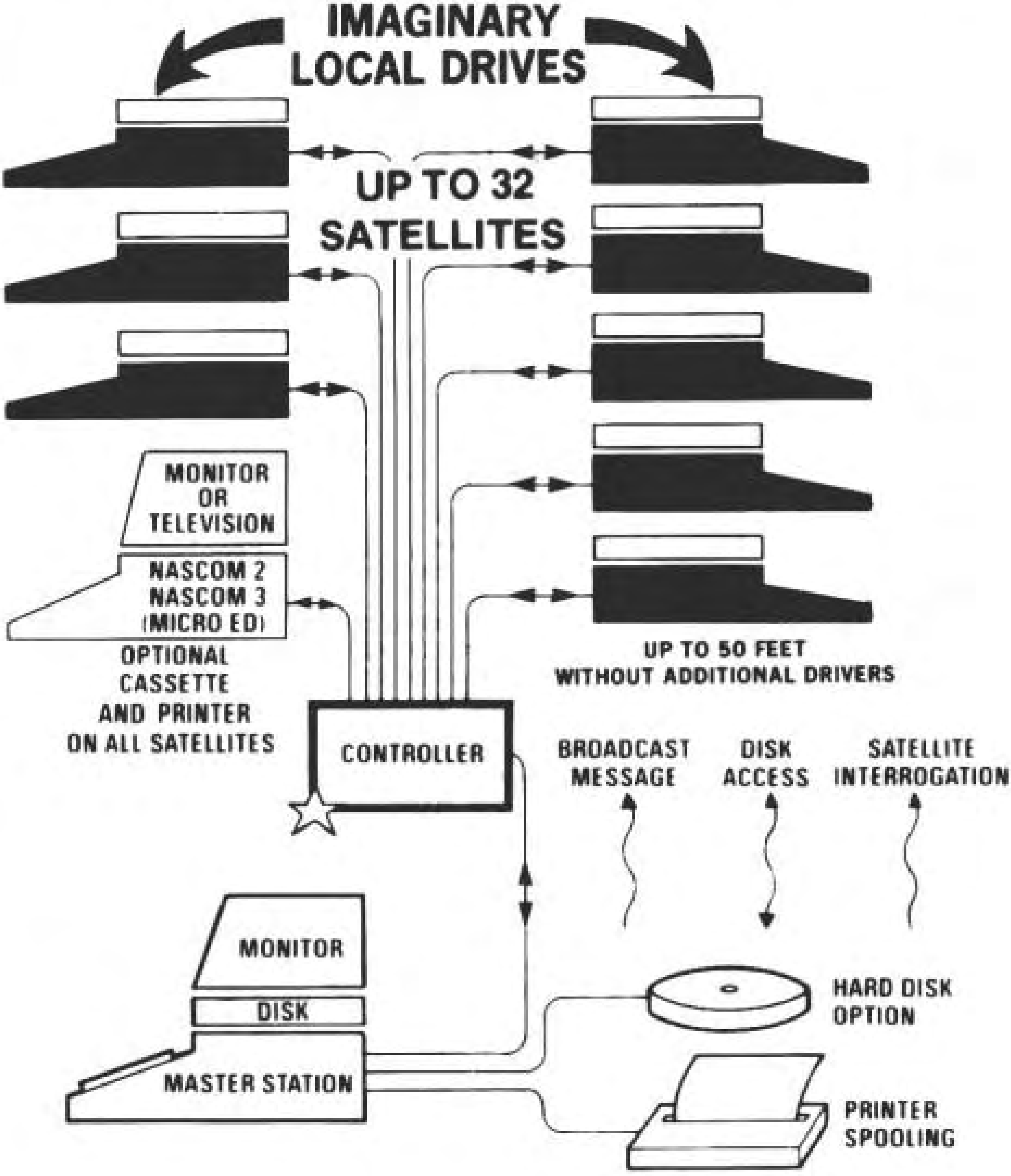
```
FEAF 324DFF   NEWOPS1 LD (STORE1),A ; save char
FEB2 0604      LD B,4 ; check rest of opcode
FEB4 114FFF    LD DE,MESSAGE
FEB7 23       NEWOPS2 INC HL
FEB8 1A        LD A,(DE)
FEB9 BE        CP (HL)
FEBA 2807      JR Z,NEWOPS3
FEBC AF        XOR
FEBD 324DFF    LD (STORE1),A ; clear store
FEC0 C3B0F1    JP ERROR
FEC3 13       NEWOPS3 INC DE ; next char
FEC4 10F1      DJNZ NEWOPS2
FEC6 E5        PUSH HL ; save pointer
FEC7 214DFF    LD HL,STORE1
FECA 3A8A0F    LD A,(PASNUM)
FECD B7        OR A
FECE 280C      JR Z,NEWOPS4 ; jump if pass 1
FED0 3A900F    LD A,(OPTCOD) ; check 'N' option
FED3 B7        OR A
FED4 2006      JR NZ,NEWOPS4 ; jump if not
FED6 CBFE      SET 7,(HL) ; set flag for ' N' option
FED8 3D        DEC A ; reset 'N' option
FED9 32900F    LD (OPTCOD),A
FEDC CB46     NEWOPS4 BIT 0,(HL) ; check first char
FEDE 2804      JR Z,NEWOPS5 ; jump if 'R'
FEE0 3EFE      LD A,SUB ; pseudo SCAL
FEE2 100B      JR NEWOPS6
FEE4 3ED9     NEWOPS5 L A,JR ; pseudo RCAL
FEE6 F5        PUSH AF ; save character
FEE7 3A8A0F    LD A,(PASNUM)
FEEA B7        OR A
FEEB 2002      JR NZ,NEWOPS6 ; jump if pass 2
FEED 3600      LD (HL), ; rest flag
FEEF F1       NEWOPS6 POP AF
FEF0 E1        POP HL ; retrieve pointer
FEF1 C3A8F1    JP CONTINU ; COMPASS has good code
FEF4 E5       SUBSITUTE PUSH HL ; from CODE
FEF5 214DFF    LD HL,STORE1
FEF8 CB4E      BIT 1,(HL) ; pseudo op ?
FEFA 2004      JR NZ,NEWOPS7
FEFC E1        POP HL ; continue if not
FEFD C3B8F7    JP OUTPUT
FF00 CB46     NEWOPS7 BIT 0,(HL) ; check character
FF02 2804      JR Z,NEWOPS8 ; jump if 'R'
FF04 3EDF      LD A,ZSCAL ; Z80 RST £18 opcode
FF06 1802      JR NEWOPS9
FF08 3ED7     NEWOPS8 LD A,RCAL ; Z80 RST £10 opcode
FF0A 47       NEWOPS9 LD B,A ; save opcode
FF0B AF        XOR A
FF0C CB7E      BIT 7,(HL) ; check 'N' ption
FF0E 2803      JR Z,NEWOPS1
FF10 32900F    LD (OPTCOD),A ; set 'N' option
FF13 78       NEWOPS10 LD A,B
FF14 CB7E      BIT 7,(HL) ; check 'N' option
```

```
FF16 3600      LD (HL),0 ; clear store
FF18 E1        POP HL ; retrieve pointer
FF19 C0        RET NZ ; return if 'N' option
FF1A C3B8F7    JP OUTPUT
FF1D E5       PRINT PUSH HL
FF1E 214EFF    LD HL,STORE2 ; check flag
FF21 CB46      BIT 0,(HL) ; set if printing
FF23 2008      JR NZ,PRINT1 ; jump if printing
FF25 FE06      CP 6 ; tick ?
FF27 2022      JR Z,PRINT4
FF29 CBC6      SET 0,(HL) ; start printing
FF2B 181E      JR PRINT4
FF2D FE06     PRINT1 CP 6 ; tick ?
FF2F 2004      JR NZ,PRINT2
FF31 CB86      RES 0,(HL) ; stop printing after
FF33 3E0D      LD A,£D ; print last line
FF35 F5       PRINT2 PUSH AF ; your own print
FF36 DB05     PRINT3 IN A,(5) ; routine to £83
FF38 CB47      BIT 0,A
FF3A 20FA      JR NZ,PRINT3
FF3C F1        POP AF
FF3D F5        PUSH AF
FF3E CBFF      SET 7,A
FF40 D304      OUT (4),A
FF42 CBBF      RES 7,A
FF44 D304      OUT (4),A
FF46 CBFF      SET 7,A
FF48 D304      OUT (4),A
FF4A F1        POP AF
FF4B E1       PRINT4 POP HL
FF4C C9        RET ;
FF4D 00       STORE1  NOP
FF4E 00       STORE2  NOP ; '1' for complete listing
FF4F 43414C20 MESSAGE DEFM 'CAL '
FF53          END EQU $
E52F           ORG COMPASS+£002F
E52F F4FE      DEFW SUBSTITUTE
E532           ORG COMPASS+£0032
E532 A6FE      DEFW NWOPS
E618           ORG COMPASS+£0118
E618 9AFE      DEFW NCOMMAND
E61E           ORG COMPASS+£011E
E61E 7EFE      DEFW PCO MAND
E8F8           ORG COMPASS+£03F8
E8F8 531A      DEFW END-COMPASS
```

N.B. As this program alters COMPASS while assembling, ensure that ALL errors are dealt with BEFORE assembling to memory.

Lucas Nascom Systems

NAS-NET 1
A Networking System for Nascom Computers



NAS-NET 1
A Networking System for Nascom Computers

NAS-NET 1 is a low-cost system which allows the sharing of resources between a network of linked Nascom microcomputers.

A system consists of a central 'master' computer linked to up to 32 satellite computers. Satlelites can be located at distances of up to at least 50 feet from the master. Data is transmitted between the master and satellites at a speed of around 64,000 baud, giving fast response in most circumstances. The master computer needs to be a Nascom 1, 2 or 3 computer equipped with disc drives and the NAS-DOS disc operating system. The satellites can be any Nascom 1, 2 or 3 computer fitted with the NAS-SYS 1 or NAS-SYS 3 operating system. The satellites are individually wired to the location of the master computer using 3 core cables connected to their standard RS 232 connections. These are connected through a multiplexer unit which is connected to the master via the standard RS 232 connection of the master station. The satellite machines require no modification other than fitting of 1K of EPROM and 3K of RAM (for a modified version of the NAS-DOS operating system), and setting of the required data transmission clock. The master station requires the fitting of a special NAS-SYS monitor EPROM, one connection from a board test point to the PL4 connector and setting of the data transmission clock. The PL4 connector can still be used for driving a parallel printer.

The satellite machines when connected to the master appear to have all the facilities normally found on a dedicated NAS-DOS machine, except for potentially dangerous commands such as Format. They are able to use the discs for both program and data storage and retrieval. The satellites will not be affected by the network until they request access to the disc. The satellites can also be disconnected from the network and used in a purely local mode with a cassette recorder. They perform identically to a standard Nascom (although the disc commands will of course be ineffective). Note that a cassette recorder cannot be used while machines are connected to the network.

The master computer retains full access to the disc operating system and all its normal functions, except that the cassette recorder cannot be used. When not connected to the network the master computer behaves exactly like an unmodified machine and retains all its disc commands.

The system allows all the machines in the network to share access to the disc drive(s) and there will also be support of spooling of data for printing by the master on a parallel printer. A serial printer can be supported by the master station by fitting the Nascom I/O card to the master. The high data rate normally gives reasonably quick response to disc access by any terminal. Note however that the satellites are handled in turn by the master, and in the extreme case of all satellites attempting to store or retrieve data simultaneously there will, inevitably, be a delay while they are all processed. As a low-cost, simple system NAS-NET 1 does nor provide any file security facilities, other than write-protection for individual files to prevent unauthorised access to files or to prevent one user writing to files while another is reading them.

This specification is provisional, and further details of the operation of the system will be released when finalised.

Lucas

Lucas Logic Limited
Welton Road Wedgnock Industrial Estate
Warwick CV34 5PZ
Tel: Warwick (0926) 497733 Telex: 312333

NEWS FROM NASCOM

This month we are using this space for the promised article on the Advanced Video Controller card (AVC) and a software/hardware modification which allows you to use a keyboard buffering technique, and generate a clock at the same time.

Those who have been waiting for their NAS-DOS version of Extended BASIC (XBASIC) should have received their copy now - apologies for the delay. We hope at some time in the future to do a version of the 'Introduction to programming the Nascom computer in (ROM) BASIC' for XBASIC, but in the meantime the existing reference manual provides all the information you need to allow you to use it.

Other documentation planned is an improved NAS-DOS manual, summary cards for various products and the final version of the SPEX manual.

Next month we plan to discuss applications of the AVC, and in particluar to introduce the first release of NAS-CAD - a computer aided design package. In addition we will discuss file access methods, with examples drawn from both ROM BASIC and XBASIC. The latter language is available under CP/M (with full graphics support), so there will be something for the CP/M user as well as the NAS-SYS/NAS-DOS user. We will also describe a method of adding soft (ie user programmable) function/control keys at a cost of £0.00, and more sophisticated means of printer control from NAS-SYS.

Mike Hessey
Technical Manager

1. EXISTING PRODUCTS

1.1 NAS-DOS/NAS-SYS utilities

It has only just been brought to my attention that the manual for these utilities states that the entry point jump table is located at nA00 etc - the table is of course at the beginning of the utilities, ie nB00 etc. Thanks to Movement Computers for pointing this out. In fact a more compact version of the utilities is now under test where the table is at nA00!!

1.2 'A picture is worth a thousand words' - the Advanced Video Controller

Computers have developed substantially in the three decades of their existence. Not only has their speed increased and their size diminished, but major strides have been made in making them easier to use. The present generation of computers is much more 'user friendly', enabling almost anyone to become a computer user. This continuing improvement in the man-machine (person-machine?) interface can only serve to widen the acceptance of computers in everyday life.

The advent of the personal computer has made it possible for problems to be solved which would previously have been prohibitively expensive to solve by means of computers. However, the cost of computing is not the only obstacle to acceptance of computers, particularly in the area of the person-machine interface.

The solutions to the person-computer interface problem are now becoming more readily available - colour graphics and speech being areas which offer the most promise. Speech simulation is already available for the Nascom range of computers, although reliable speech recognition is still some way off. The ability to communicate to the computer user complex or simple results has always been a problem in the past, but now that high-quality colour graphics displays are available this hurdle can be crossed.

Colour graphics enables the computer to present information pictorially, which greatly improves the usability of programs. Sometimes information is difficult or impossible to present using only numbers and words, but becomes easily comprehensible in graphics form. Even computer output which does not require colour graphics to be usable will usually become clearer when converted to colour. The use of high-quality colour graphics is now open to all users, enabling business, scientific, engineering and other users to gain maximum efficiency.

The addition of quality colour graphics to the Nascom range of computers is achieved by means of the Advanced Video Controller (AVC). Equally important to the effective implementation of colour graphics is comprehensive and easy to use support software, and this is provided as standard with the AVC. Access to the colour graphics is made simple from either ROM BASIC or Extended BASIC (XBASIC), the latter being available for tape, NAS-DOS and CP/M users.

The principles of displaying a graphics picture are transparent to the user, but to gain a better understanding an insight into the theory of graphics in general will be useful.

The AVC works on the bit mapped raster scan principle, which means that the computer can directly access and set the colour of any graphics point (known as pixels) on the display. This provides a much more flexible system than that allowed with a programmable character generator system. While programmable character generator (PCG) systems can offer faster graphics generation of a limted range of character shapes, they suffer severely from programming difficulties and the lack of generality of the pictures that they can represent. Normally they are limited to only 128, or possibly 256, special characters.

There are other types of graphic display systems available, the vector scan being the best known. However, vector scan devices and controllers offer less colour shades and are considerably more expensive.

The display produced by the AVC can best be considered as a two dimensional grid, the grid being divided into small squares (called pixels - picture elements) each of which can be independently controlled by the computer. When viewed from a distance these combinations of pixels merge together to form shapes. This is similar to the way that newspaper photographs are produces - if you look closely at a newspaper photograph the individual picture elements can be seen.

The AVC uses the raster scan principle. In this an electron beam scans across the face of the display screen from left to right, then moves down a line and repeats the left, right, down sequence until it reaches the bottom of the screen. This can be compared with how a person reads or writes a page of text. As the beam traverses the screen its energy is converted into light by the special coating on the inside of the screen. The intensity of the light produced can be varied by changing the energy of the beam. Colour displays are produced by having three coatings (phosphors) to give the three primary colours. The whole screen is scanned fifty times a second, which is just fast enough for the human eye to average out the light levels. Thus a stable picture is seen, and not just a moving spot of light. This is the way in which television pictures are formed.

Whereas with raster scan systems all the possible pixels are traversed by the beam, in vector scan systems the beam is moved only to the pixels which require ilumination. This provides very fast line generation, but does not give constant illumination.

The larger the number of pixels used the better will be the definition of the picture formed. The AVC has two resolution modes:

1. *SINGLE DENSITY, in which 392 horizontal and 256 vertical* pixels are used.

2. DOUBLE DENSITY, in which 784 horizontal and 256 vertical pixels are used.

The AVC uses three separate bit-mapped memory planes to store the pixels in. Bit-mapped means that each pixel on the display screen is controlled by a known bit (or bits for colour) in the memory of the AVC. Each byte of the memory controls the status of eight horizontally adjacent pixels. The use of the three planes of memory allows the status of the three primary colour planes to be controlled by the data inserted into the AVC memory. Because colour is a mixture of the three primary hues red, green and blue, any colour can be produced by controlling these three primary colours. Each of the three memory planes is used to control one of the primary colours.

These memory planes are all on the AVC board, and do not conflict with the normal program memory of the Nascom computer. They are only paged into the computer's memory map when it is necessary to read or write data to the display.

To determine which bits of which memory location of which memory plane to turn on and off in order to produce the required visual effect is quite complicated. Fortunately extensive software support is provided with the AVC which does the hard work for you. This software is accessible using ROM BASIC and Extended BASIC (XBASIC), and provides simple commands to control the graphics operations. The support software occupies around 6K of main memory. In CP/M machines it also provides an 80 column by 25 line character display as standard.

Suppose, for example, we want to plot a red circle near the top right hand corner of the display. We need only issue commands to select the colour, shape and size of the object to be drawn. In this case the command/program statements would be as follows:

```
COLOUR 1
APOLY 250,150,100
```

The instruction COLOUR indicates the colour to be used, and the subsequent number indicates which of the available colours are to be used. The colours available are as follows:

BLACK	 0	BLUE	 4
RED	 1	MAGENTA	 5
GREEN	 2	CYAN	 6
YELLOW	 3	WHITE	 7

The command APOLY specifies that we wish to draw a polygon (a circle is a polygon with an infinite number of sides!). The position to draw the circle is given by the X, Y co-ordintates (250, 150 in this case) and the size of the circle is specified by its radius, 100 in this case. The POLY command was chosen since it gives much more generality in the shapes that can be drawn than would a more specific command such as 'circle'.

Some of the commands available from XBASIC and ROM BASIC are summarised below. In ROM BASIC each command is prefixed by the word SET. Most commands are followed by one or more numeric parameters (they can be variables in BASIC). In many cases there are default values for these parameters, so that not all of them need to be specified.

MODE<a,b>
This command specifies the type of display required and which combinations of primary colours which are to be displayed.

There are five types of display available -

Text display (Nascom 2 video RAM output)

Low resolution graphics - 391 by 256 points with one of eight colours per pixel.
High resolution graphics display giving a resolution of 784 by 256 with one of two colours per pixel (on/off monochrome)
Mixed density graphics display giving resolutions of 391 by 256 and 784 by 256 combined, with each pixel having one of four colours.
Mixed text and graphics - the Nascom 2 video RAM output combined with one of the graphics displays.

COLOUR
This command specifies the colour to be used in succeeding commands. In addition to just specifying a single colour (one of the eight already mentioned) the AVC allows the use of special shading colours, which can be used to differentiate areas such as graphs or shadows in pictures. These shading colours are specified to the computer as the relative levels of the three primary components of colour (red, green and blue), allowing a selection of 4913 different colours.

BACKGND
This allows the whole of the display to be set to a specified colour ie a background colour. Again up to 4913 colours are available.

DEFAULT
This is a special command that sets up all the default values of the command parameters.

ORIGIN
This command is used to specify the centre of rotation to be used in those commands which allow a rotation to be specified.

PEN
This command has two functions - to select the combination of primary colours which are available for drawing purposes (as distinct from MODE, which determines which colours will be displayed after shapes are drawn) and to specify how the shapes are to be drawn. This last parameter is rather difficult to explain - shapes can be drawn as they would be with coloured pens on paper, but they can also be drawn in other ways. These other ways produce effects which change the shapes appearance and colour. The effects are similar to television visual effects.

WINDOW
This command instructs the computer which parts of the display can be used for the output of graphics or text. It can be used to protect areas of the screen so that they cannot be over-written.

DUMP
This command allows the picture which would normally be displayed on the screen to be output to a matrix printer with dot graphics capability.

GSAVE
This allows the user to save the current display picture on disc.

GLOAD
This command allows a previously saved picture to be recalled.

FILL
This command colours in an outlined shape.

CURSOR
This command returns the current cursor co-ordinates.

CHECK
This returns the colour of a specified pixel.

DASH
This allows the user to specify that lines and shapes are to be drawn using dotted or dashed lines.

The preceding commands are mainly concerned with the way in which pictures will be drawn. The commands which follow specify the actual drawing operations.

PLOT
This command sets the colour of a specified pixel. The position of the pixel can be expressed in absolute terms, or relative to the current position of the cursor.

LINE
This command draws a line on the display. The positions of the start and end points can be absolute or relative to the current cursor position, and lines can also be specified in terms of their length and direction.

The shapes produced can also be scaled, rotated and sheared. Shearing refers to an angular displacement of parts of the shape, so that, for example, a square becomes a diamond, or capital letters become italics.

TRI
This command is used to draw triangles, and can cope with all types of triangle.

RECT
This command is used to draw rectangles. Like other commands it provides for displacement, scaling, shear rotation and colouring.

POLY
This command draws polygons, ranging from triangles through to circles. Many parameters can be specified, such as drawing the radii or chords of circles. The command can also be used in drawing curves.

MAT
This is a special command which allows the user to define a shape and then move, colour, scale, rotate, shear etc it - a macro facility. There is no limit to the complexity of shape defined in a macro - other than the memory capacity of the computer!

LABEL
This graphics command is used to display a string of characters at a specified position, angle, slant, colour etc.

This concludes the basic description of the operation of the AVC and the software commands which are available to make it easy to use. In the next issue we will describe some applications of the AVC, and in particular we will introduce the first issue of NAS-CAD - a simple computer aided design system using the AVC.

2. TECHNICAL TIPS

In this section we are introducing the first instalment of a series of articles on modifying the NAS-SYS operating system. While the features which we describe should be useful the intention of the series is to give some of the newer Nascom computer users an insight into the operating system and its features. The routines described have generally not been optimised for efficiency in respect of either speed or memory size.

2.1 A buffered keyboard for the Nascom computer

Buffered keyboards allow the user to type in information while the computer is still performing other tasks. They need to be used with care - if you type in answers to anticipated questions, but the actual questions from the program differ from those expected, the results can be disconcerting, to say the least! Keyboard buffers usually have a limited 'type ahead' capacity - typically 20 characters or less.

The keyboard buffering system which we are going to describe can be used with any non-networked Nascom 2 or 3 computer fitted with a PIO and running NAS-SYS 3 or NAS-DOS (and Nascom 1 computers, with a few minor changes to the procedure described). All that is required to fit it is a short length of wire, a soldering iron and a bit of software, so the cost is negligible. A shortcoming, though, is that the disc user will find the buffering is inoperative while disc access is in progress. This is because disc access disables the computers interrupt system, which we use in this buffering system. You will also find that the keyboard repeat speed is less consitent, and for games use in particular a type-ahead feature can be undesirable.

The keyboard buffering can be disabled at any time by use of the RESET key or restarting NAS-SYS (EO). The length of the keyboard buffer can be changed from within a program by POKing a single location (down to 1 if required).

The description that follows is intended for those who have some knowledge of their machine, or who want to know more about it. It gives all the information you need to implement the keyboard buffer. In future issues we will give some other technical tips which will build on the ideas presented here.

1. How it works

A 50Hz clock signal derived from TP22 on the Nascom 2 board is connected to port A bit 7 of the PIO. The PIO is configured by the software to generate an interrupt when the edge of this clock pulse is received. The Z80 is set to use interrupt mode 2, in which mode an interrupt results in transfer of control to a location the address of which is located in memory at an address given by the contents of the interrupt register and data placed on the bus by the interrupting device.

At initialisation the PIO is configured for output on port B and bits 1 to 6 of port A, and input on port A bits 0 and 7. This allows a clock input to be used on port A bit 7, and also prepares the ports for parallel printer use, thus eliminating any need to configure ports in parallel printer drivers, which might otherwise upset the clock input during their configuration process. Control signals are also sent to the PIO so that it will generate an appropriate address and an interrupt when an edge occurs on the clock input of port A bit 7.

The interrupt routine and its workspace and buffer area are located at the top of memory - C800 in this case. When an interrupt occurs the interrupt routine scans the keyboard via the normal NAS-SYS routines, and places any data found in the buffer.

To insert the new keyboard input routine into NAS-SYS the entire SCAL table is copied into RAM, and the vector at 0C71 which points to the table is modified accordingly. The keyboard input entry in the table is then changed to point to the new input routine, which inspects the keyboard buffer to see if there has been any input.

Some systems programs use a similar method of moving the SCAl table to RAM and then perform their own changes to it. Such programs (eg NAS-SEMBLER) should therefore not be run with the keyboard buffering in operation.

Any program which disables interrupt will inhibit use of the keyboard, so be very careful! In particular you need to avoid cold starting BASIC, which for some incomprehensible reason disables interrupt. You can conveniently circumvent this by loading an empty BASIC program from tape or disc and then using the normal Z to warm start. This method has the added advantage that you can reserve the necessary space at the top of memory when you save this empty file, and you will then not need to remember the memory size when starting BASIC in the future. You should first J BASIC in the usual way (without loading these routines, of course), and specify 51200 as memory available. Then use the CSAVE"B" command to save the workspace, which can then be loaded via an R command in future. BASIC can the be 'cold' started by the Z command. As only one block needs to be stored on tape this is still a quick operation. Next month we will mention an even faster solution! NAS-DOS users can of course save this empty file with a command JB:BASIC, and then simply type JE:BASIC to 'cold' start BASIC.

To avoid the danger of accidentally cold starting BASIC we have eliminated the J command in the SCAL table in RAM. NAS-DOS users could instead replace the existing procedure with one to chain to the program BASIC on disc.

Interrupts have to be disabled by NAS-DOS (and CP/M) during the disc read and write operations, so the buffering function will also be momentarily disabled at these times.

The use of the interrupt routine causes a loss of around 5% in processing speed - not noticeable in most applications.

2. Modifying the hardware

This is a very simple job. Just solder a wire from Test Point TP22 on the Nascom 2 computer board to pin 24 on PL4, the PIO cable. The most tidy way of doing this would be to solder a wire on the underside of the board, and if you do not use the PIO for any other purpose, or only for a parallel printer, this is quite satisfactory. However, to allow the entire PIO to be freed for other applications I prefer to take the wire from TP22 out of the case and connect it to pin 25 of a D-type plug. This can then be plugged into the parallel interface port via the standard D-type socket (as described in Application Note AN-0005, which was reproduced in an earlier issue of Nascom News). If you are using a parallel printer this stops you using both, so hard wiring via an on/off switch may be preferable. Incidentally this pin of the PIO is not used in the standard parallel printer interfacing system, so there is no clash in hardware in this respect.

This connection provides the 50Hz clock input (from TP22) to the PIO. This same connection is made in a number of special systems - NAS-NET and DCS-MOS - and we would suggest you consider reserving this PIO line as a clock line.

3. The software

The software is all in assembly laguage, and is probably best loaded at the top of memory, as shown here. The listing which follows is fully commented, so we suggest you read through it to see how it works.

To enter the software you can just copy the machine code into memory, or better still create a source file using ZEAP or NAS-SEMBLER. This will make future additions to the keyboard routines, and other extensions which we will discuss in future issue of Nascom News, easier. Although the listing given here is for NAS-SEMBLER, we have not used any features which are not available in ZEAP.

NAS-DOS users may like to arrange the routine to be automatically loaded and enabled by the user boot (JU) command, and to change the SCAL J location to automatically execute a dummy program BASIC from disc. This will eliminate the need for any non-standard steps in using the system.

Note that we have placed certain key information in RAM at CCA0. The first three locations contain a clock, in hours, minutes and seconds, followed by a 1/50 second tick count. These are set to zero when you initialise the system. A simple BASIC program of the type listed below would let you set the clock and read it. Because of the disabling of interrupts by NAS-DOS this clock will lose time during disc access. However, for many timing applications this does not present any problems, and it is a cheap and easy way of getting a clock/timer on the computer. Naturally this type of clock will not run when the computer is turned off, and the time data is lost. The next memory location (CCA4) contains the buffer length, normally the maximum permitted value of 48 (decimal). You can reduce this if you wish - down to one to prevent typing ahead.

Note that there is a jump vector at C80C. This currently points to a RET instruction, which will be executed every 1/50 second. You could put in this vector a jump to your own routine to be executed every 1/50 of a second. You can try using this to display the time on the top line of the screen, and update it every second. Note that in adding such a routine you should be careful not to change any registers, or to make excessive use of the stack.

4. Clock read/set from BASIC

The current time can be displayed by typing RUN. To set the clock you should type RUN 100.

```
10 REM:CLOCK:
20 REM CLOCK READ AND DISPLAY
30 REM ----------------------
40 REM
50 I=12*4096+12*256+160-65536
60 PRINT
70 PRINT "Time ="PEEK(I)"."PEEK(I+1)"."PEEK(I+2)
80 END
100 I=12*4096+12*256+160-65536
110 CLS
120 PRINT "CLOCK SETTING PROGRAM"
130 PRINT "----------------------"
140 PRINT:PRINT"Type in time in the form ";
150 PRINT"Hours,minutes,seconds"
160 INPUTH,M,S
170 POKEI,H:POKEI+1,M:POKEI+2,S
180 PRINT
190 END
```

5. Source listing of keyboard buffering routines

```
                ;:TKEYINT:
                ;NASCOM BUFFERED KEYBOARD HANDLER
                ;---------------------------------
                ;COPYRIGHT (C)1983 LUCAS LOGIC LIMITED
                ;REV 2.2     10 APRIL 1983
                ;*****************************************
                ;HARDWARE REQUIREMENTS
                ;---------------------
                ;The interrupt system and PIO are used.
                ;Connect TP22 on the N2 board to the PIO
                ;Port A bit 7 (pin 24 of PL4).
                ;*****************************************
                ;DECLARATIONS
                ;------------
                BAS1    EQU    0C8     ;Page for this s/w
                BAS     EQU    0CB00   ;and start address
                INTVEC  EQU    BAS1    ;High address of vect
                KBLINK  EQU    0C32    ;Cursor blink speed
                KLONG   EQU    0C2E    ;Repeat speed delay
                PIOA    EQU    04      ;PIO data port A
                PIOAC   EQU    06      ;PIO control port A
                PIOB    EQU    05      ;PIO data port B
                PIOBC   EQU    07      ;PIO control port B
                STAB    EQU    0C71    ;Address of SCAL table
                VECLO   EQU    020     ;Low address of vect
                ZOLDKB  EQU    080     ;Old entry for k'bd
                ZOLDR   EQU    081     ;Old R routine
                ;*****************************************
                ;JUMP TABLE
                ;----------
                ;All entry points via fixed location jump
                ;table.  Any future extensions will not
                ;alter these entry points.
                        ORG    BAS
                        ENDS
C800                    ENT
C800 C3 30 C8           JP     INIT    ;Initialisation
C803 C3 8D C9           JP     PRINT   ;Reserved for printer
C806 C3 8D C9           JP     PRINT   ;and another
C809 C3 8D C9           JP     PRINT   ;!
C80C C3 8C C9   USER    JP     UROUT   ;Vector to your own
                ;                      ;clock handling extensn
C80F                    DEFS   3       ;Reserve another 4
C812                    DEFS   3       ;jump table entries
C815                    DEFS   3       ;for the future
C818                    DEFS   3
C81B                    DEFS   5
                ;Note that we have made sure the interrupt
                ;table starts at BAS+020.  This is important as
                ;the PIO interrupt vector points here.
                ;NOTE: The vector must point to an even address
                ;**********************************************
                ;INTERRUPT VECTOR TABLE
                ;----------------------
```

```
C820 00 C9          INTAB  DEFW   INTHND      ;Handler for keyboard
C822                       DEFS   14          ;Space for other vector
                    ;**************************************************
                    ;INITIALISATION OF KEYBOARD HANDLER
                    ;----------------------------------
C830 F3             INIT   DI                 ;Disable interrupts
C831 2A 71 0C              LD     HL,(STAB)   ;Get old k'bd routine
C834 11 C2 00              LD     DE,061+061;address
C837 19                    ADD    HL,DE
C838 5E                    LD     E,(HL)      ;We add this to new
C839 23                    INC    HL          ;table so that we can
C83A 56                    LD     D,(HL)      ;use this from int.
C83B ED 53 82 CD           LD     (NEXTAB),DE
C83F 2A 71 0C              LD     HL,(STAB)   ;Do the same for R
C842 11 A4 00              LD     DE,"R"+"R"
C845 19                    ADD    HL,DE
C846 5E                    LD     E,(HL)
C847 23                    INC    HL
C848 56                    LD     D,(HL)
C849 ED 53 84 CD           LD     (NEXTAB+2),DE
C84D 2A 71 0C              LD     HL,(STAB)   ;Now move old table to
C850 11 82 00              LD     DE,082      ;RAM to allow mods to
C853 19                    ADD    HL,DE       ;be made
C854 11 04 CD              LD     DE,TABLE
C857 01 7E 00              LD     BC,03F+03F
C85A ED B0                 LDIR
                    ;Put new keyboard (buffer) scan in table
C85C 21 6C C9              LD     HL,KIN
C85F 22 44 CD              LD     (TABLE+040),HL
                    ;Put in new tape read (R) (also fixes V)
C862 21 8E C9              LD     HL,READ
C865 22 26 CD              LD     (TABLE+022),HL
                    ;Eliminate 'J' call which disables interrupt
C868 2A 0E CD              LD     HL,(TABLE+0A)
C86B 22 16 CD              LD     (TABLE+012),HL
                    ;DOS users could replace the entry with a
                    ;routine which chains to a program "BASIC"
                    ;on disc.  this would be an 'empty' BASIC
                    ;program.
                    ;Put new table address in workspace
C86E 21 82 CC              LD     HL,TABLE-082
C871 22 71 0C              LD     (STAB),HL
                    ;Zero the clock
C874 AF                    XOR    A
C875 32 A0 CC              LD     (CLOCK),A ;Hours
C878 32 A1 CC              LD     (CLOCK+1),A ;Minutes
C87B 32 A2 CC              LD     (CLOCK+2),A ;Seconds
C87E 32 A3 CC              LD     (TICK),A  ;Tick counter (50Hz)
                    ;Set buffer count and pointers
C881 32 AC CC              LD     (NCHAR),A ;Count
C884 21 A4 CD              LD     HL,BUFFER
C887 22 AD CC              LD     (NEXIN),HL ;In pointer
C88A 22 AF CC              LD     (NEXOUT),HL ;and out pointer
                    ;Set up the PIO
C88D 3E FF                 LD     A,0FF       ;Both ports control
C88F D3 06                 OUT    (PIOAC),A
C891 D3 07                 OUT    (PIOBC),A
C893 3E 81                 LD     A,081       ;Port A bits 0&7 input
```

```
C895 D3 06            OUT    (PIOAC),A
C897 AF               XOR    A
C898 D3 07            OUT    (PIOBC),A ;Port B all bits out
C89A 3E 97            LD     A,097     ;Interrupt mode on
C89C D3 06            OUT    (PIOAC),A ;port A
C89E 3E 7F            LD     A,07F     ;Interrupt mask for
C8A0 D3 06            OUT    (PIOAC),A ;bit 7 only
C8A2 3E 20            LD     A,VECLO   ;Interrupt vector
C8A4 D3 06            OUT    (PIOAC),A
                ;Adjust cursor blink and keyboard speed
C8A6 21 90 01         LD     HL,400    ;Slow down cursor
C8A9 22 32 0C         LD     (KBLINK),HL
C8AC 21 1C 02         LD     HL,540    ;Speed up repeat!
C8AF 22 2E 0C         LD     (KLONG),HL
                ;Enable interrupts and return to NAS-SYS
C8B2 3E C8            LD     A,INTVEC
C8B4 ED 47            LD     I,A
C8B6 ED 5E            IM     2
C8B8 FB               EI
C8B9 DF 5B            SCAL   MRET
                ;Space for future expansion
C8BB                  DEFS   045
                ;****************************************
                ;INTERRUPT HANDLER
                ;-----------------
C900 ED 73 A8 CC INTHND LD   (CLKSP),SP ;Use special stack
C904 31 F3 CC         LD     SP,CLCKST+02F
C907 F5               PUSH   AF        ;Save registers used
C908 E5               PUSH   HL
                ;Process clock tick
C909 21 A3 CC         LD     HL,TICK
C90C 34               INC    (HL)
C90D 3E 32            LD     A,50
C90F BE               CP     (HL)
C910 20 23            JR     NZ,CONT   ;Second finished?
C912 36 00            LD     (HL),0    ;Yes - reset tick
C914 21 A2 CC         LD     HL,CLOCK+2 ;and adjust clock
C917 34               INC    (HL)
C918 7E               LD     A,(HL)
C919 FE 3C            CP     60
C91B 20 18            JR     NZ,CONT
C91D 36 00            LD     (HL),0    ;Adjust minutes
C91F 21 A1 CC         LD     HL,CLOCK+1
C922 34               INC    (HL)
C923 7E               LD     A,(HL)
C924 FE 3C            CP     60
C926 20 0D            JR     NZ,CONT
C928 36 00            LD     (HL),0    ;Adjust hours
C92A 21 A0 CC         LD     HL,CLOCK
C92D 34               INC    (HL)
C92E 7E               LD     A,(HL)
C92F FE 18            CP     24
C931 20 02            JR     NZ,CONT
C933 36 00            LD     (HL),0
                ;Now scan keyboard
C935 3A AC CC   CONT  LD     A,(NCHAR) ;Check if buffer full
C938 21 A4 CC         LD     HL,BUFLEN
C93B BE               CP     (HL)
```

```
C93C 28 22                     JR     Z,CONT2    ;Yes - so no action
C93E D5                        PUSH   DE         ;Save other registers
C93F C5                        PUSH   BC
C940 DF 80                     SCAL   ZOLDKB     ;Scan k'bd as usual
C942 30 1A                     JR     NC,EC      ;No key
C944 21 AC CC                  LD     HL,NCHAR
C947 34                        INC    (HL)
C948 2A AD CC                  LD     HL,(NEXIN) ;Put in buffer
C94B 77                        LD     (HL),A
C94C 23                        INC    HL
C94D 22 AD CC                  LD     (NEXIN),HL
C950 AF                        XOR    A
C951 11 D4 CD                  LD     DE,ENDBUF ;Check if at end
C954 ED 52                     SBC    HL,DE
C956 20 06                     JR     NZ,EC
C958 21 A4 CD                  LD     HL,BUFFER ;Back to start
C95B 22 AD CC                  LD     (NEXIN),HL
C95E C1                EC      POP    BC         ;Restore registers
C95F D1                        POP    DE
C960 CD 0C C8          CONT2   CALL   USER       ;Go to user vector
C963 E1                        POP    HL         ;Restore others
C964 F1                        POP    AF
C965 ED 7B A8 CC               LD     SP,(CLKSP) ;and stack
C969 FB                        EI                ;Enable interrupts
C96A ED 4D                     RETI
                       ;*****************************************
                       ;KEYBOARD INPUT FROM BUFFER
                       ;--------------------------
C96C 3A AC CC          KIN     LD     A,(NCHAR) ;Check if anything
C96F B7                        OR     A          ;there
C970 C8                        RET    Z          ;No - just return
C971 3D                        DEC    A          ;Yes - update count
C972 2A AF CC                  LD     HL,(NEXOUT) ;Get char.
C975 46                        LD     B,(HL)
C976 23                        INC    HL         ;Point to next
C977 22 AF CC                  LD     (NEXOUT),HL
C97A 11 D4 CD                  LD     DE,ENDBUF ;Check for end
C97D ED 52                     SBC    HL,DE
C97F 20 06                     JR     NZ,KRETP
C981 21 A4 CD                  LD     HL,BUFFER ;Set back at start
C984 22 AF CC                  LD     (NEXOUT),HL
C987 32 AC CC          KRETP   LD     (NCHAR),A ;Update count
C98A 78                        LD     A,B        ;Retrieve character
C98B 37                        SCF               ;Set carry indicator
C98C C9                URDUT   RET
C98D C9                PRINT   RET
                       ;*****************************************
                       ;TAPE READ
                       ;---------
                       ;This routine is required to disable interrupts
                       ;during tape read.  If this were not done some
                       ;tape input would be missed during processing
                       ;of interrupts.
                       ;We could just disable interrupt, but the 4
                       ;ESCAPE's would be inoperable, so instead we
                       ;restore normal keyboard scan.
C98E 2A B2 CD          READ    LD     HL,(NEXTAB) ;Restore old k'bd
C991 22 44 CD                  LD     (TABLE+040),HL
```

```
C994 F3          0273              DI                 ;Disable interrupts
C995 DF 81       0274              SCAL    081        ;Now usual tape routine
C997 FB          0275              EI                 ;Restore interrupts
C998 21 6C C9    0276              LD      HL,KIN     ;and keyboard buffering
C99B 22 44 CD    0277              LD      (TABLE+040),HL
C99E C9          0278              RET
                 0280 ;******************************************
                 0282 ;EXPANSION SPACE
                 0283 ;---------------
C99F             0285              DEFS    0301
                 0287 ;******************************************
                 0289 ;DATA AREA
                 0290 ;---------
CCA0 00 00 00    0292 CLOCK   DEFB    00,00,00   ;Hours,mins,secs
CCA3 00          0293 TICK    DEFB    00         ;Clock tick count
CCA4 30          0294 BUFLEN  DEFB    030        ;Buffer length
CCA5 00          0295 STAT1   DEFB    00         ;Spare
CCA6 00          0296 STAT2   DEFB    00         ;Spare
CCA7 00          0297 STAT3   DEFB    00         ;Spare
CCA8 00 00       0298 CLKSP   DEFB    00,00      ;Temp store for stack
CCAA 00 00       0299 OLDKBD  DEFB    00,00      ;Old k'bd address
CCAC 00          0300 NCHAR   DEFB    00         ;Character count
CCAD 00 00       0301 NEXIN   DEFB    00,00      ;Input pointer
CCAF 00 00       0302 NEXOUT  DEFB    00,00      ;Output pointer
CCB1             0303 SPARE   DEFS    013        ;Spare workspace
CCC4             0304 CLCKST  DEFS    040        ;Stack area
CD04             0305 TABLE   DEFS    03F+03F    ;SCAL table
CD82             0306 NEXTAB  DEFS    011+011    ;Extension space
CDA4             0307 BUFFER  DEFS    030        ;Buffer
                 0308 ENDBUF  EQU     $
```