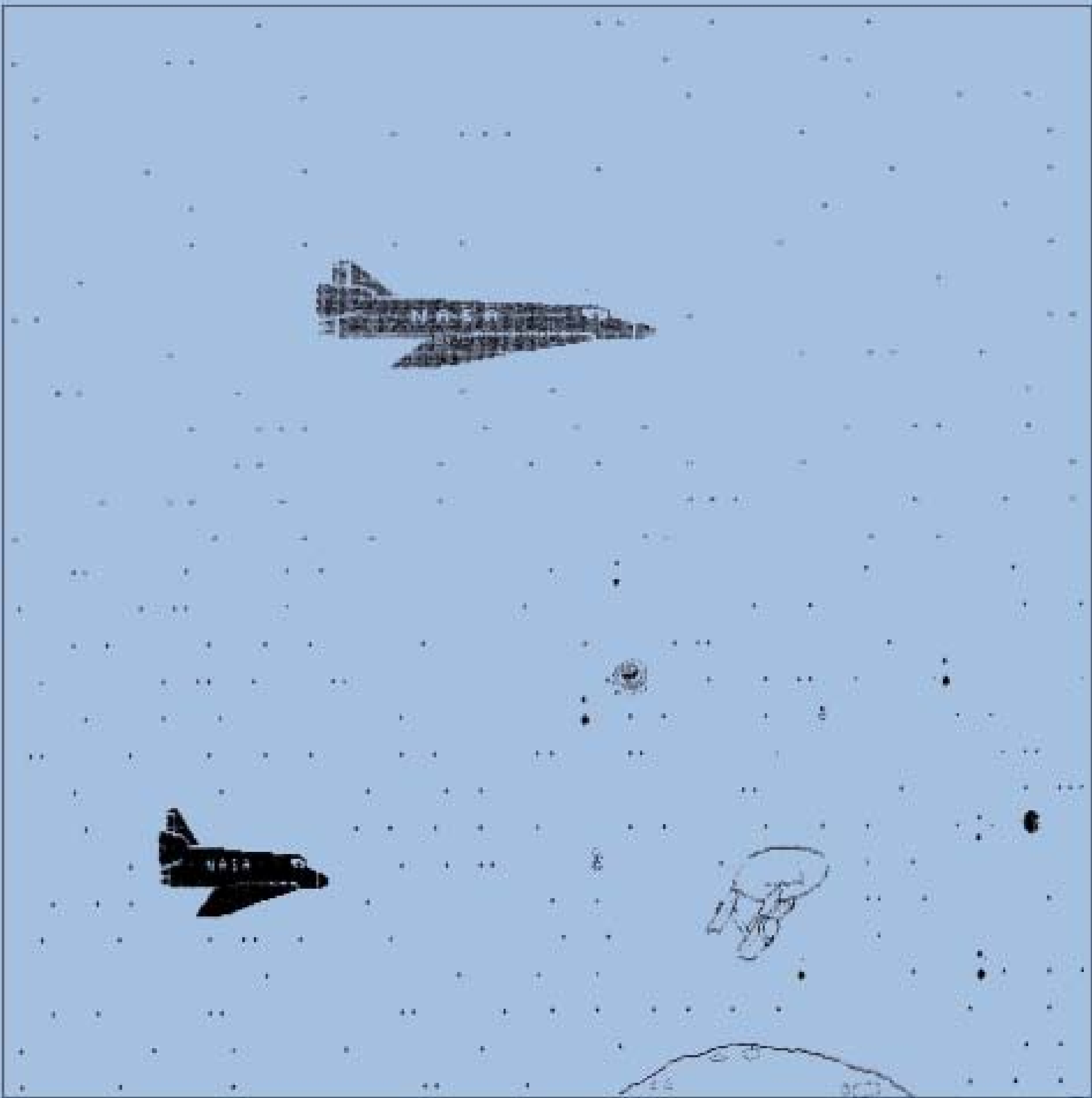
μP MICROPOWER VOLUME 1, NUMBER 3

# A MAGAZINE FOR NASCOM USERS




November, 1981 95 p

PROGRAM POWER

# NASCOM 1 & 2

## Invasion Earth (MC/G)
## (with incredible sound effects)

Our fast MC code SPACE INVADERS now has sound effects which really show what can be achieved with a programmable sound chip. The aliens fire six different missile types – intelligent homing, angled, direct, multiple warhead, exploding and radio-jamming.
Other features are:- choice of 10 speeds and 4 levels of difficulty; replace barriers if desired and advancing attackers as each wave is destroyed. More addictive than 'pot' this game may ruin your life! £10.95
N.B. If you have an earlier version of the game, we will update it for £3 – just return your original cassette.
SPECIAL OFFER – Deduct £5 if you order the Chip, Sound Board, Demo Program & Invasion Earth together.

## Jailbreak in Space (16K/MC/G)

In this exciting arcade game(similar to Cosmic Guerrilla) you defend a top security jail, holding 3 alien prisoners, against the onslaught of enemy space ships. They try to dismantle your jail 'brick-by-brick' attacking from both sides. Increasing points are awarded for aliens, brick-carrying & prisoner-freeing aliens respectively. The level of difficulty is extremely high, so do not buy this one unless you are an experienced 'arcadian'. High-score & initials of high-scorer displayed. £8.95

## Graphic Golf (16K/B/G)

Excellent us of Nascom graphics enhance this well written golf program. Play the full 18 hole course, but steer clear of the bunkers, trees and spectators. Penalty points are incurred for hitting the ball out of bounds. Random generation of wind speed & direction brings added variety. Select the right club (choice of 9 + driver) and also the direction of your shot and the force of your swing. Once on the green the screen is re-drawn to show the hole, flag and putting distance. Beat the PAR 72 £7.95

## AY-3-8910 Programmable Sound Chip

YES – This IS the amazingly powerful "Clang, Bang, Zap, Tweet" sound & music generator, with three channels which can be independently programmed for sound output and amplitude. In addition it has an 'envelope controlled' noise generator, ideal for creating explosions and firing sounds. £6.45

## Sound Chip Data Manual (60 pages)

This contains a full description of the architecture & operation of the chip, detailed advice on the interfacing to various microprocessors, and comprehensive explanations on the generation of music and sound effects. £2.25 (no vat)

## Sound Chip Interfacing Board

The board has been designed to interface between the Parallel Input/Output Port (PIO) of the Nascom and the sound chip. It is supplied ready-built and just plugs straight onto your PIO connector. Nascom 1 connectors available on request. Sound generation is illustrated in machine code & Basic, (chip not included) £13.50

## Sound Chip Demo Program (MC)

A brief summary of the main registers is given, together with a description of their functions. Thereafter, two separate modes may be selected. Direct mode allows values to be entered into the chip registers via the keyboard, making experimentation simple, thus leading to a rapid appreciation of the chip's potential. The second mode turns the keyboard into a 7 octave 'piano', displaying the notes being played as well as the values of the registers. £5.95

*** NASCOM 1 – Cottis Blandford cassette interface for N2 format, reliability & fast load £14.90
- 8K RAM required unless otherwise stated
- Please state if Nascom TAPE Basic required.

ALL PROGRAMS SUPPLIED ON CASSETTE IN CUTS/KANSAS CITY FORMAT

Please add 55p/order P & P + VAT @ 15%.
Large (15½p) Sae for FULL CATALOGUE.

PROGRAM POWER
5, Wensley Road
Leeds LS7 2LX.

**MICROPOWER** **Volume 1, No. 3** **November, 1981**

## CONTENTS



## EDITORIAL

We asked for your comments on the order forms for issues 3 and 4, and we hope to discuss some of the points raised in details in the next issue. However, the answers did show what a wide range there is in the interests of Nascom Owners, and in their levels of attainment. Many people said, in effect, don't publish articles for beginners, don't print games programs, we want articles on advanced hardware and software projects. But at least as many asked for simpler or more detailed explanations of points which they felt were skipped over because the author assumed they were too obvious to need expansion.

We are grateful to all those who took the trouble to send in their comments, and also for the letters and articles that we have received. You can be assured that all are read carefully, and that wherever possible we shall act upon the ideas you send in – so please don't stop writing.

Many of you asked for games programs, so we shall try to include more of these, although this will annoy the purists. In this issue we have included a program by D. G. Johnson, the author of 'Graphic Golf'. Although it is a simple and well known game, it demonstrates brilliant use of the Nascom 2 graphics set.

ERRATUM In the article 'Modifications to Tiny Basic' on page 15 the eighth line from the bottom should read

11DE Change to DF6421210C7E23666FC9

## LETTERS

Dear Sir,

I must congratulate Chris Blackmore on the clever software used by his Monitor.Com article. Readers may like to know that Nas-Sys 3 can equally well be adapted; however, the addresses at the top of page 13 need to be changed as follows:-

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0800 | - | 00 | 00 | 00 | 0265 | - | 11 | 0A | F8 |
| 019B | - | 21 | 0A | F8 | 0268 | - | 21 | 4A | F8 |
| 0236 | - | 11 | 0A | F8 | 0254 | - | 11 | CA | FB |
| 023E | - | 11 | BA | FB | | | | | |

Key allocations need to be changed as well, as there are only two ‘spare’ keys available.

If a disc is used, then keys D, F, J, L, Y, Z will probably be unused for there oroginal purpose and several routines may be added, e.g., Handshake, Find, Substitute, etc., as well as the extra routines in the article. At this stage I have dedicated keys as follows:

| | | | | |
|---|---|---|---|---|
| D | - | Dump to disc | 0788 | change to 3B 09 |
| F | - | Find bytes | 078C | change to 8F 0A* |
| L | - | Load from Disc | 0798 | change to 06 0A |
| Y | - | Return to CP/M | 07B2 | change to 15 09 |

* - Start address of my routine.

Incidentally, it may not be necessary to type in all of the Nas-Sys as described. If your Nascom can be switched from Nas-Sys to CP/M (see my article in INMC80-3), copy Nas-Sys or Zeap, or whatever into middle memory (4000 to 8000 somewhere) and switch to CP/M. Load in DDT or ZSID and you should still find your program in memory. It can be copied down as required using the M command. Alternatively, if you have R.COM and W.COM (Tape to disc Read and Write routines available from Nascom Dealers) you can transfer your programs that way.

Incidentally I use to think that ED was a pig. Now that I am more used to it I think it is a donkey – a bt hard to drive, but quite capable. One misses the screen editing, which is not even restored by SYS6, but the Macro commands are very useful.

C. Bowden, Stithians, Truro

P.S. I have almost got Zeap 2 (RAM Version) and Naspen VS converted to work with V.D.U. RAM at £F800, but there are a few queer effects. Can anyone, without infringing any copyrights, list the changes needed for 100% success.

Dear Sir

As I am at present working in Germany with the Forces I wonder if anybody else out here has a Nascom; if so perhaps a mini Nascom Club could be started. If so please contact me at the address below.

I have just acquired a Data Dynamics 390 printer, which appears to be normal ASR33 teletype in fancy clothing. I have no circuit diagrams for this printer and do not know how to interface it to my nascom. Can anyone help (Somebody must be using an ASR33 with a Nascom).

A. M. Morfee, Officers Mess
RAF Wildenrath, BFPO 42

## HANDS-ON

### by Viktor . . . . . Part, the third

### PRINT

Since most of us cut our programming teeth on the PRINT statement, there is not much point in covering acres of paper in explanations and illustrations – however for the benefit of those just beginning I will summarise the ground rules. [INPUT "Enter SKILL LEVEL"; X$:IF X$= "ADVANCED" THEN GOTO(next section)]

The PRINT statement outputs to a terminal, usually the screen or a printer(or both). It operates from the current position of the cursor. If used on its own, a carriage return/line feed(CR/LF) is output, thus placing the cursor at the start of the next line.

You can also print numbers [PRINT 5] , the answers to calculations [PRINT 5*6 (=30)], the value of variables [PRINT A (=10 where A=10)], string literals [PRINT "Fred"], strings [PRINT X$ (Nascom-where X$ has been defined as "Nascom"), plus a whole series of intrinsic functions which are listed in the manual. Useful examples are:-

PRINT CHR$(X) – where X has any ASCII value from 0 to 255. For example, CHR$(65) prints the letter A, CHR$(181) prints a small human figure, if your system is equipped with the Nascom 2 graphics ROM.
PRINT FRE(0) – gives you the remaining memory space available for your BASIC program.
PRINT SQR(X) – outputs the square-root of X.

In formatting text it is obviously important to know where the cursor will be positioned after the computer has carried out a PRINT instruction. If a PRINT statement is terminated with a colon, or if the line ends without a special terminator, the next PRINT statement will begin at the start of the next line, the Nascom having output a CR/LF. However, if you end the statement with a semi-colon, the next PRINT statement will start at the next available space; e.g.

10 FOR A=1 to 5:PRINT A;:NEXT will give you:-

1 2 3 4 5

Note that a space is output in front of each number – if you PRINT a negative number this space will be occupied by a minus sign.

A further modification, available within the PRINT statement itself, is the use of commas to divide the output into zones. PRINT 1,2,3 puts 1 in position 0,2 in position 14 and 3 in position 28. If you not redefined WIDTH then the third zone is 20 characters wide. If a zone is completely filled or exceeded the instruction will still be carried out but the cursor will then be moved to the start of the next free zone.

In setting out text on the screen or for printing there are a number of other useful functions in BASIC which you can use with the PRINT statement, viz.

SCREEN X,Y . . . places the cursor on the Xth position across and Yth line down.

PRINT TAB(I) . . . moves the cursor to the Ith horizontal tab position.

PRINT SPC(X) . . . moves the cursor X spaces along from the current position.

POS(I) . . . reurns the current position of the cursor. This is useful where the cursor might finish up in one of a number of different tab postions after following alternative routes to reach the current program line. In order to obtain a satisfactory print layout you could insert a line as follows:-

```
20 R=POS(I):IF R>25 THEN PRINT
```

Note (I) is only a dummy argument in this instruction, that is, because POS is a function, not a command, it has the format of a function, but what you put within the brackets is immaterial.

**SPACE SAVING IN PRINT STATEMENTS**

Since extra memory chips are so cheap these days saving space by shortening Basic lines has few advantages. However, it is quite often annoying to find that you are one or two characters in excess of 48 and therefore must have yet another line. You will no doubt have discovered by now that in many situations it is not necessary to use the second quotation mark when printing or defining strings. (e.g. [X$="FRED] is acceptable as a statement on its own or at the end of a multiple-statement line.)

Now consider the following lines:

```
20 V=99:X$="15th":Y$="JAN":Z$="1981
30 PRINT"Amount";V;TAB(25);"Period ";X$;" ";Y$;" ";Z$
```

T12'15th' and 'JAN' require closing quotes but '1981 ' does not. Line 30 will not fit onto one screen line, and could only be entered by using direct entry of the single-byte reserved words from the keyboard, using the 'GRAPHICS' key. But several characters can in fact be left out since the second quotation marks can double for the statement separators. i.e.

```
30 PRINT "Amount"V;TAB(25)"Period "X$" "Y$" "Z$
```

**DEEK & DOKE**

Like PEEK & POKE , DEEK & DOKE allow you direct access to the data held in memory, so that you can modify or read its contents. However, because PEEK & POKE operate only on single bytes the highest value that you can POKE into a memory is 255; similarly, PEEK(X) returns the value held in the single byte whose address is X.

When using DEEK & DOKE two consecutive bytes are accessed. This means that there are sixteen bits available for arithmetic, giving a theoretical maximum of 65535 (that is two to the sixteenth power, less one). When you DOKE a value to a specific address, the least significant byte is stored at the next byte above. Why are they stored in that order? Simply because that is the normal order in which the Z80 microprocessor stores sixteen-bit values. In order to cope with negative numbers the most significant bit of the high order byte is used to indicate the sign; if this bit is set, then the number is to be read as negative. The range of available values is thus -32768 to + 32767.

Let’s try an example:-

Enter DOKE 3200,32767. If you now return to NAS-SYS and tabulate from 0C80, you will find 0C80 contans FF (low-order byte with all bits set) and 0C81 contains 7F (high-order byte having all bits except the most significant one set).

Now try DOKE 3200,-1:DOKE 3202,-2:DOKE 3204,-32767

The result should be as follows:-

| | | |
|---|---|---|
| 0C80 | (low byte) | FF |
| 0C81 | (high byte) | FF |

Note computer holds this as 65536-1 i.e. (15x4096) + (15x256) + (15x16) + (15x1)

| | | |
|---|---|---|
| 0C82 | (low byte) | FE |
| 0C83 | (high byte) | FF |

The computer has 65536-2, i.e. (15x4096) + (15x2 56) + (15x16) + (14x1)

| | |
|---|---|
| 0C84 (low byte) | 01 |
| 0C85 (high byte) | 80 |

The computer stores 65536-32767, i.e. (8 x 409 6) + (1 x1)

Why, though, do we have to bother with all this complicated nonsense? Let us have a look at a few examples. Firstly, a series of DOKES is very useful for setting up short machine code subroutines, which are to be called from a BASIC program. Here the values DOKE’d have no meaning in the BASIC program – they are just the decimal values which when converted to hexadecimal can be interpreted by the computer as machine code instructions. The USR(0) routine (given in the manual) to scan the keyboard for a key depression is a very good example of this.

Secondly, POKE & DOKE are often used to change options in the monitor. You will know that under NAS-SYS the command K1 gives you lower case as the standard print output. This can be achieved from BASIC directly, or in the course of a program by the statement POKE 3111,1. Similarly, if you want to turn on output to your printer in the middle of a program, you can use the various DOKE instructions outlined in the manual. N.B. Different values apply in NAS-SYS 3 from those in NAS-SYS 1.

A third use might be to store a value which is to be picked up later by a machine code subroutine. The example which follows allows you to generate sounds from a BASIC program by flipping Bit 5 of the keyboard port.

Sub-routine:

```
6000  DOKE   3200,23533: DOKE 3202,3330: DOKE 3204, 19437
6010  DOKE   3206,3328:DOKE3208,8254:DOKE3210,211
6020  DOKE   3212,30731:DOKE3214,8369:DOKE3216,-4613
6030  DOKE   3218,75:DOKE3220,-20723:DOKE3222,211
6040  DOKE   3224,30731:DOKE 3226,8369:DOKE 3228,7163
6050  DOKE   3230,-19590:DOKE 3232,-7648:DOKE 3234,201
6060  RETURN
```

Main routine:

```
1000    GOSUB 6000:DOKE 4100,3200 (Dec. Addr.of M/C routine)
1010    DOKE 3330,X (where X=no. of complete loops)
1020    FOR I=A TO B, STEP C (where different values of A,B & C
give different sounds)
1030    DOKE 3328,I:U=USR(0): NEXT I:
```

Memory location 3330 (0D02 hex) is DOKE'd with the value which will control the length of the sounds generated, while 3328 (0D00 hex) is DOKE'd with the frequency parameters.

Another use to which I have put these instructions was in the creation of an array in a large program when I was short of memory. The original array was something like A(8,8), i.e. 64 variables, requiring 6 bytes each. By using DOKE to put values in memory and DEEK to retrieve them it was possible to use only two bytes for each variable, e.g. if the start of the array is 3584, to put the value X into what was previously A(I,J), you enter DOKE 3584+[16*(I-1)]+[2*(J-1)],X. The instruction is a lot more cumbersome but there is still a significant saving in memory usage.

**RANDOM NUMBERS**

Random numbers are very useful in games programming, both in games of chance like Pontoon or Fruit Machine and also in more complex programs like Star Trek, where you wish to vary the results of selecting a particular option in the course of the game.

The RND function does not, in fact, generate random numbers at all; it merely starts with a pre-determined value and then produces new numbers according to a formula. These values are always in the range 0 to 0.999999. If the argument used with the RND function is negative, a new sequence of numbers will be started. While different negative numbers produce different sequences, any particular negative number will always produce the the same sequence. If the argument is greater than zero, the function returns the next number in the current sequence, while RND(0) reproduces the previous number output in the current series.

To vary the start of a game you need a random start somewhere in the list, so to speak, of the numbers being generated. One method would be as follows:

```
10 INPUT"Enter a no.";N:IF N > 0 THEN N= -N 20 A = RND(N)
```

When the player enters a number he selects a new sequence for the game. However, this allows players to cheat – they can affect the random sequence, and hence the course of the game, by there initial selection. A better method is to use the keyboard scan user routine referred to above (see the manual for the m/c code and the decimal equivalents):

```
10    DOKE 4100, 3200:REM Tell BASIC where routine is located
20    PRINT "When you are ready, press any key"
30    A=RND(1):B=USR(0):IF B=0 THEN 30
```

Here, each time you start, the generation of numbers will stop at a different point due to the varying time taken to react to the message.

Another helpful routine which uses the RND function is the generation of integers within a given range e.g. a number between 1 and 12. Some BASICS already have this as a built – in function. On the Nascom you need to adopt a formula similar to the following:

```
10 A=INT((RND(1)*T)+B)
```

where T equals the top of the range and B equals the bottom. If we substitute 12 and 1 for T and B we get 12 x ( no. between 0 and 0.999999) which should be less than 12. If we then add 1 and then reduce the answer to integer format the result in most cases will be in range 1 to 12. However, due to the rounding system in the BASIC a number larger than 0.999995 is treated as being equal to 1, and the computer will then, in this instance produce answers in the range 2 to 13. So for a perfect program you will need to test for data which is out of range.

Next issue will probably see the last look at various statements and functions within BASIC. After that we can perhaps dissect a few programs or interesting parts of programs, and in doing so cover various points not looked at so far.

HAPPY KEYBOARD BASHING !!

## READING TRS-80 PROGRAM TAPES

**by Mike Fox**

There is a great deal of software available for micro-computers, but it is generally not possible to exchange programs between systems because the data is stored on tape in different formats. This article describes a method for reading and converting TRS-80 tapes for the Nascom. The project needs both hardware and software, and is for TRS-80 Level 2 Basic (also Video Genie in the U.K., P.M.C-80 in U.S.A, and System 80 in Australia and N.Z.), but it could be modified for other machines.

The TRS-80 writes tapes at 500 Baud. An 80 microsecond clock pulse is sent to the tape every 2 milliseconds. The data bits to be stored are represented by inserting an extra 80 microsecond pulse between two clock pulses for a 1, and leaving the gap empty for a 0. This of course is incompatible with the CUTS standard used in the Nascom II. Therefore a small circuit consisting of one LM3900 (an IC containing four operational amplifiers) and a couple of dozen discrete components is used to input the signal from the cassette via the Nascom PIO. Figure 1 shows the circuit diagram of the interface, while a suggested Vero layout is shown in figure 2. Make sure that pin 11 of the Nascom 2 PIO plug is connected to 0 volts on pin 16.

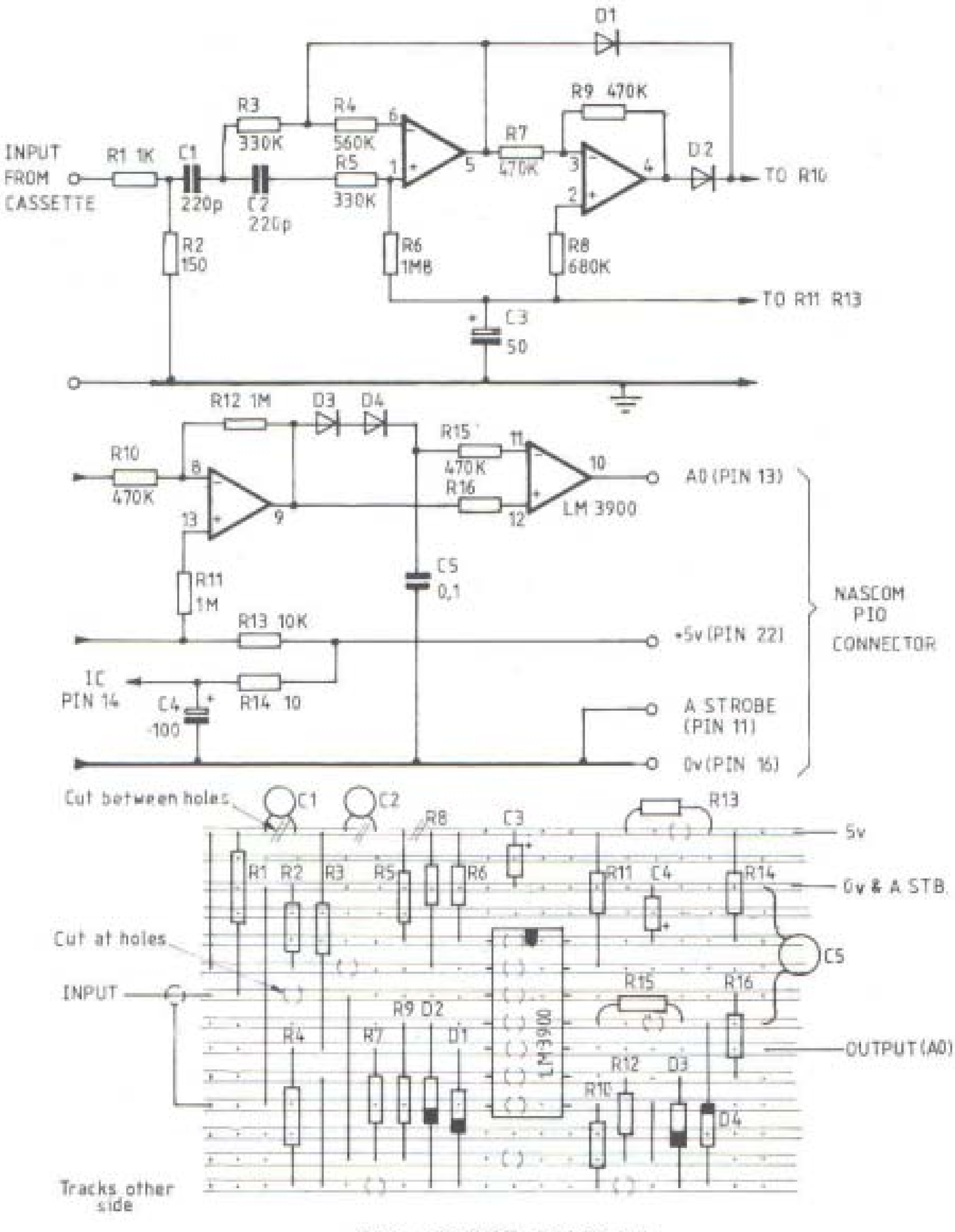
### COMPONENTS REQUIRED

| Resistors | | | | Capacitors | | Semiconductors | |
|---|---|---|---|---|---|---|---|
| R1 | 1 kohm | R9 | 470 kohm | C1 | 220 pf | LM3900 | Quad. Op-Amp. |
| R2 | 150 kohm | R10 | 470 kohm | C2 | 220 pf | | |
| R3 | 330 kohm | R11 | 1 megohm | C3 | 50 µF | D1-D4 | Small-signal silicon diode |
| R4 | 560 kohm | R12 | 1 megohm | C4 | 100 µF | | |
| R5 | 330 kohm | R13 | 10 kohm | C5 | 0.1 µF | | |
| R6 | 1.8 megohm | R14 | 10 ohms | | | | |
| R7 | 470 kohm | R15 | 470 kohm | | | | |
| R8 | 680 kohm | R16 | 470 kohm | | | | |

### SOFTWARE FOR MICROSOFT BASIC

The first part of the program reads the tape and loads it into the correct memory location for Nascom 2 Basic. As the reading is done by software timing, the delay values in the program will vary for machines running at 2 Mhz and 4Mhz. At the start of the tape there is about 4 seconds of nulls (00), followed by a sync character of A5 hex. When this character is detected the program starts to load the data from the tape starting at address £10F6; as it is stored, the data is also displayed on line one of the screen. The first four characters are SSSn, where n is the program identification. These are not used, and the actual Basic program starts at £10FA. The end of the program is indicated by three nulls, which cause a jump to part two of the tape reading routine.

In this second section, the token values used in TRS-80 Basic are converted to




TOP: CIRCUIT DIAGRAM

BOTT: 0·1in. VERO BOARD LAYOUT

the values used by Nascom's Basic (Tokens are the single byte codes used for reserved words, e.g., PRIN is stored as £9E in the Nascom, as £B2 in the TRS-80). Any token which cannot be converted is changed to a REM, and its memory location, line number (in decimal) and TRS-80 value are displayed on the screen. The first two bytes of each line point to the start address of the next line. As the TRS-80 Basic starts at £42E9 and Nascom 2 Basic at £10FA, £31EF must be subtracted from the value read from the tape. Finally location £10F9 is set to zero, and the address of the end of the Basic program is stored at £10D6, 10D8 and £10DA. The conversion routine then returns to the monitor.

To use the program, first enter Nascom Basic with a cold start by command J, then exit by entering MONITOR or by pressing RESET. Next load the tape reading/conversion program, with the correct delay values for your clock speed. Run the program by entering E0C80, connect your cassette recorder to the input port via the given circuit, and start the TRS-80 program tape. After loading is completed and control has returned to Nas-Sys, re-enter Basic with a warm start. The TRS-80 program can now be listed, and may even run.

There are several factors which can prevent the Basic program running. Some of the commands used in TRS-80 Basic are not available in the Nascom Basic, and the program may have to be modified to carry out these instructions in some other way. From the listed program and the table of token values given at the end of this article it should be possible to find out what the program is doing. Two useful articles are "Whose Basic Does What", BYTE, January 1981, page 318, and "TRS-80 Program Recovery", INTERFACE AGE, December 1980, page 100. "The Basic Handbook", by David A. Lien, published in the U.S.A. by Compusoft, is an invaluable book.

Peek and poke addresses may also vary. In particular, the TRS-80 screen consists of 16 lines of 64 characters, and is located from £3C00 to £3FFF. The decimal values of the screen locations run from 15360 at the top left, 15423 top right, to 16320 bottom left, 16383 bottom right. Remember that the Nascom top line is not scrolled, and that it is located in memory after the bottom line.

You will find that the volume and tone settings of the cassette recorder are very critical. If nothing appears on the screen, re-run the program with £008E set to 00 to stop sync checking, and £0C80 set to 00 to prevent the program ending on reading the first null characters, (the program will have to be terminated by pressing the RESET button, and the second part of the program, token conversion, will not have been executed.) Adjust the volume and tone settings until a display appears. Probably the display will not make sense, because the characters will be out of sync. Replace the original values at £0C8E and £0CC0, and try again.

If you load a program and find that when you LIST it starts correctly but then produces rubbish, the line address pointers have probably been misread. Try adjusting the volume or tone slightly and reloading.

It is possible to have a Basic cassette tape from a TRS-80 disc system, where the starting address is £6A46. The result will be that only the first line will LIST correctly, and the rest will be rubbish. Replace the subtraction values at £0D16 with £4C and at £0D1B with £59.

Don't forget to use the correct delay settings for your clock speed. For operation at 2Mhz the values required are: £0CC8=£26, £0CDA=£3C, £0CE1= £1D. At 4 Mhz the corresponding values are £53, £81, and £3D. The clock frequency of a standard TRS-80 is 1.7 Mhz, so the programs should run more quickly on a Nascom.

The conversion program halts when the screen has filled with tokens that can not be converted. You should make a note of the details, and then press any key to continue

## USING THE PROGRAM WITH CRYSTAL BASIC

The table of equivalent tokens at the end of the article can be used to modify the program so that it will read and convert TRS-80 programs for Crystal Basic. Replace the Nascom tokens in the table starting at £0DD9 with the equivalent Crystal Basic token. You will also have to change the addresses at £0C86, £0C87 and at £0D12, £0D13 to suit the start of text in your version of Crystal Basic; for example, if the program text starts at £2D00, £0C87, £0C87 must be changed to £FC2C and £0D12, £0D13 to £002D. As the first four bytes from the tape overwrite the last four bytes of the interpreter, these will have to be restored before running the program. The line pointer offsets at £0D16 and £0D1B must be also be changed; for text starting at £2D00 the values should be £E9 and £15 respectively. The end of program address should be stored at £0C87, £0C88. The program should then be listable after a warm start.

## READING MACHINE CODE TAPES

The second listing is a routine to read TRS-80 machine code programs. Such a program is first read into a block of memory starting at £1000. The routine then scans the loaded program, testing the checksums and removing the loading addresses, checksum bytes, sync bytes etc. If a checksum error is detected the start address of the block containing it is displayed; try to reload the program at a slightly different volume or tone setting. If all the checksum are correct the routine displays the executions address and returns to Nas-Sys.

You can now use Nas-Dis to disassemble the program. Of course, you will still have a lot of work to do to produce a running program. All calls to the monitor will have to be identified and replaced by their Nascom equivalent. The screen addresses and format will have to be modified. Any program using pixels will have to be changed to take account of the different codes used on the two machines (add £40 to the TRS-80 character and change bits 1,2,3,4 to 2,4,1,3).

```
                      0010 ;  PROGRAM TO READ TRS-80 LEVEL 2
                      0020 ;  BASIC PROGRAM TAPES INTO NASCOM
                      0030 ;  AND THEN CONVERT TO RUN UNDER
                      0040 ;  NASCOM MICROSOFT BASIC
                      0050 ;
                      0060 ;  BY MIKE FOX
                      0070 ;  AUCKLAND, NEW ZEALAND
                      0080 ;
                      0090 ;  14th JULY, 1981
                      0100 ;
0C80                  0110            ORG £0C80
0C80  0E00            0120            LD C, 0          ; RESET COUNTER
0C82  210A08          0130            LD HL,  £080A    ; VDU LINE 1
0C85  11F610          0140            LD DE, £10F6     ; BASIC START
0C88  CDC50C          0150 NSYSC      CALL SUB         ; GET A BIT
0C8B  FEA5            0160            CP £A5           ; SYNC BYTE A5?
0C8D  20F9            0170            JR, NZ, NSYSC    ; IF NOT, LOOP
0C8F  CDC50C          0180 NEXT       CALL SUB         ; NOW GET 8 BITS
0C92  CDC50C          0190            CALL SUB
0C95  CDC50C          0200            CALL SUB
0C98  CDC50C          0210            CALL SUB
0C9B  CDC50C          0220            CALL SUB
0C9E  CDC50C          0230            CALL SUB
0CA1  CDC50C          0240            CALL SUB
0CA4  CDC50C          0250            CALL SUB
0CA7  77              0260            LD (HL), A       ; LOAD TO SCREEN
0CA8  12              0270            LD (DE) A        ; LOAD TO MEMORY
0CA9  7D              0280            LD A, L
0CAA  FE39            0290            CP £39           ; END OF LINE?
0CAC  2803            0300            JR Z, EOL
0CAE  2C              0310            INC L            ; CONTINUE ON THIS LINE
0CAF  1802            0320            JR INLINE
0CB1  2E0A            0330 EOL        LD L, 10         ; BACK TO LINE START
0CB3  1A              0340 INLINE     LD A (DE)        ; RECALL BYTE
0CB4  13              0350            INC DE
0CB5  FE00            0360            CP 0             ; IS IT ZERO?
0CB7  2008            0370            JR NZ, NOZERO    ; CHECK FOR END
0CB9  0C              0380            INC C            ; INCREMENT COUNTER
0CBA  79              0390            LD A, C
0CBB  FE03            0400            CP 3             ; 3 IN A ROW?
0CBD  2004            0410            JR NZ, NOTEND
0CBF  184F            0420            JR TOKEN         ; TO TOKEN PROGRAM
0CC1  0E00            0430 NOZERO     LD C, 0          ; RESET COUNTER
0CC3  18CA            0440 NOTEND     JR NEXT          ; LOPP FOR NEXT BYTE
                      0450 ;
                      0460 ; SUBROUTINE TO GET A BIT
                      0470 ;
0CC5  C5              0480 SUB        PUSH BC
0CC6  F5              0490            PUSH AF
                      0500 ; ******************************************************
0CC7  0653            0510            LD B, £53        ; 4 Mhz, £26 FOR 2 Mhz
                      0520 ; ******************************************************
0CC9  10FE            0530 LP1        DJNZ LP1         ; DELAY LOOP
0CCB  DB04            0540 LP1A       IN A, (04)       ; READ PORT A0
0CCD  EEFF            0550            XOR £FF          ; INVERT IT
0CCF  1F              0560            RRA              ; ROTATE TO CARRY
```

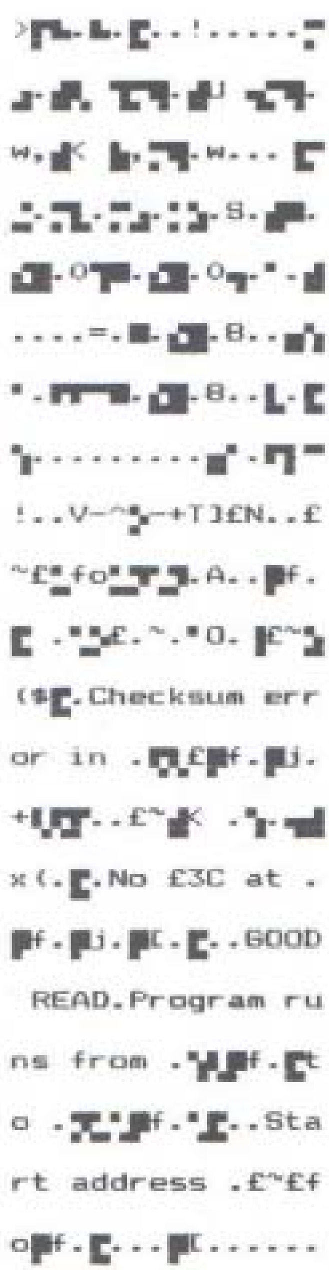
```
0CD0  30F9     0570        JR NC, LP1A      ; LOOP IF NO CLOCK
0CD2  DB04     0580        IN A, (04)       ; JUST CHECKING
0CD4  EEFF     0590        XOR £FF
0CD6  1F       0600        RRA
0CD7  30F2     0610        JR NC, LP1A      ; TRANSIENT, LOOK AGAIN
               0620 ; ************************************************************
0CD9  0681     0630        LD B, £81        ; USE £3C FOR 2 Mhz
               0640 ; ************************************************************
0CDB  10FE     0650 LP2    DJNZ LP2         ; DELAY AND WAIT
0CDD  00       0660        NOP              ; BEFORE LOOKING
0CDE  00       0670        NOP              FOR DATA PULSE
0CDF  00       0680
               0690 ; *************************************************************
0CE0  063D     0700        LD B, £3D        ; USE £1D FOR 2 mHZ
               0710 ; *************************************************************
0CE2  00       0720 LP3    NOP              ; IN SEARCH WINDOW
0CE3  DB04     0730        IN A, (04)       ; READ PORT
0CE5  EEFF     0740        XOR £FF          ; INVERT IT
0CE7  1F       0750        RRA              ; ROTATE TO CARRY
0CE8  3804     0760        JR C, GOT        ; PULSE FOUND – CHECK
0CEA  10F6     0770        DJNZ LP3         ; NO PULSE, LOOK AGAIN
0CEC  181C     0780        JR NOGOT         ; WINDOW EXPIRED
0CEE  DB04     0790 GOT    IN A, (4)        ; CHECK AGAIN
0CF0  EEFF     0800        XOR £FF
0CF2  1F       0810        RRA              ; ROTATE TO CARRY
0CF3  3804     0820        JR C, GOTONE     ; DEFINATELY A 1
0CF5  10EB     0830        DJNZ LP3         ; TRANSIENT – TRY AGAIN
0CF7  1811     0840        JR NOGOT         ; STILL IN WINDOW?
0CF9  F1       0850 GOTONE POP AF           ; RETORE REGS
0CFA  00       0860 END    NOP              ; TO END WINDOW
0CFB  00       0870        NOP              ; USING NOP AS DELAY
0CFC  00       0880        NOP
0CFD  00       0890        NOP
0CFE  00       0900        NOP
0CFF  00       0910        NOP
0D00  00       0920        NOP
0D01  00       0930        NOP
0D02  10F6     0940        DJNZ END         ; STILL IN WINDOW?
0D04  C1       0950        POP BC           ; END – RESTORE REGS
0D05  07       0960        RLCA             ; SHIFT ACCUMULATOR
0D06  CB87     0970        RES 0, A         ; ZERO AND
0D08  3C       0980        INC A            ; SET BIT 0 TO 1
0D09  C9       0990        RET              ; RETURN
0D0A  F1       1000 NOGOT  POP AF           ; NO DATA PULSE
0D0B  C1       1010        POP BC           ; RESTORE REGS.
0D0C  07       1020        RLCA
0D0D  CB87     1030        RES 0, A         ; SET BIT 0 TO 0
0D0F  C9       1040        RET              ; RETURN
               1050 ;
               1060 ; PROGRAM TO COMVERT TOKENS FROM TRS-80
               1070 ; TO  NASCOM.  ALSO  SETS  UP  POINTERS  AND
               1080 ; CORRECT LINE ADDRESS CODES  BY SUB-
               1090 ; TRACTING   £31EF  (OR  £594C  FOR  DISC)
               1100 ;
0D10  E5       1110 TOKEN  PUSH HL
0D11  21FA10   1120        LD HL, £10FA     THROW AWAY SSSn
```

```
0D14   7E            1130 NEXTL   LD A, (HL)        ; LOW BYTE OF POINTER
0D15   D6EF          1140         SUB £EF           ; SUBTRACT £EF
0D17   77            1150         LD (HL), A        ; RE-WRITE
0D18   23            1160         INC HL
0D19   7E            1170         LD A, (HL)        ; HICH BYTE OF POINTER
0D1A   DE31          1180 SBC A,  SUBTRACT £31
0D1C   77            1190         LD (HL), A        ; RE-WRITE
0D1D   23            1200         INC HL
0D1E   5E            1210         LD E, (HL)        ; SAVE LINE NUMBER
0D1F   23            1220         INC HL
0D20   56            1230         LD D, (HL)        ; SAVE LINE N UMBER
0D21   23            1240 LOOP    INC HL
0D22   7E            1250         LD A, (HL)        ; GET BYTE OF BASIC
0D23   D600          1260         SUB 0
0D25   2805          1270         JR Z, CHECKE      ; IF EOL, CHECK FOR
PROG. END
0D27   FC480D        1280         CALL M, SUBT      ; IF GREATER THEN £80
                     1290                           ; TRANSLATE TOKEN
0D2A   18F5          1300         JR LOOPO;LOOP AGAIN
0D2C   23            1310 CHECKE  INC HL
0D2D   23            1320         INC HL
0D2E   7E            1330         LD A, (HL)        ; LOAD BYTE
0D2F   D600          1340         SUB 0
0D31   2803          1350         JR Z ENDT         ; IF ZERO, PROG. END
0D33   2B            1360         DEC HL
0D34   18DE          1370         JR NEXTL          ; NOT ZERO, CONTINUE
0D36   23            1380 ENDT    INC HL
0D37   22D610        1390         LD (£10D6), HL    ; SAVE END ADDRESS
A 0D3A 22D810        1400         LD (£10D8), HL
0D3D   22DA10        1410         LD (£10DA), HL
0D40   21F910        1420         LD HL, £10F9 RESTORE ZERO @ £10F9
0D43   3600          1430         LD (HL), 0
0D45   E1            1440         POP HL
0D46   DF5B          1450         SCAL MRET         ; RETURN TO NAS-SYS
                     1460 ;
                     1470 ;
                     1480 ; SUBROUTINE TO CONVERT TOKENS
                     1490 ;
0D48   E5            1500 SUBT    PUSH HL
0D49   21D90D        1510         LD HL, TABLE      ; LOAD TBLE ADDRESS
0D4C   F5            1520 AGAIN   PUSH AF           ; SAVE DATA BYTE
0D4D   7E            1530         LD A, (HL)        ; LOAD BYTE FROM TABLE
0D4E   D600          1540         SUB 0
0D50   280D          1550         JR Z, NOTM        ; END OF TABLE?
0D52   F1            1560         POP AF            ; NO, SO RESTORE DATA
0D53   BE            1570         CP (HL)           ; COMPARE WITH TABLE
0D54   2804          1580         JR Z EQUAL        ; EQUAL?
0D56   23            1590         INC HL            ; NO – GO TO NEXT
0D57   23            1600         INC HL
0D58   18F2          1610         JR AGAIN          ; GO AROUND AGAIN
0D5A   23            1620 EQUAL   INC HL            ; MATCH FOUND
0D5B   7E            1630         LD A, (HL)        ; GET NEW TOKEN
0D5C   E1            1640         POP HL            ; RESTORE BASIC ADDRESS
0D5D   77            1650         LD (HL), A        ; STORE NEW TOKEN
0D5E   C9            1660         RET               ; RETURN
                     1670 ;
```

```
                  1680 ; NO MATCH, SO CONVERT TO REM AND DISPLAY
                  1690 ;ADDRESS, LINE NUMBER (DECIMAL) AND CODE
                  1700 ;
0D5F  F1          1710 NOTM    POP AF
0D60  E1          1720         POP HL          ; DISPLAY ON SCREEN
0D61  E5          1730         PUSH HL         ; DETAILS OF EACH
0D62  DF66        1740         SCAL £66        ; DISPLAY HL IN HEX
0D64  62          1750         LD H, D
0D65  6B          1760         LD H, E
0D66  CD890D      1770         CALL HEXDEC     ; CONVERT HEX. TO DEC.
0D69  DF69        1780         SCAL £69        ; OUTPUT ONE SPACE
0D6B  E1          1810         POP HL
0D6C  7E          1820         LD A, (HL)
0D6D  DF68        1830         SCAL £68        ; PRINT ACC. IN HEX
0D6F  DF7E        1840         SCAL £7E        ; OUTPUT TWO SPACES
0D71  DF69        1850         SCAL £69        ; OUTPUT ONE SPACE
0D73  3E8E        1890         LD A, £8E       ; LOAD CODE FOR REM
0D75  773 1900                 LD (HL), A      ; REWRITE AS REM
0D76  3A2A0C      1910         LD A, (£0C2A)   ; SCREEN FULL?
0D79  FE0B        1920         CP £0B          ; CURSOR POSITION
0D7B  200B        1930         JR NZ, NOTFUL   ; PASS UNLESS
0D7D  3A290C      1940         LD A, (C29)     ; NEXT OUTPUT SCROLLS
0D80  FEAA        1950         CP £AA
0D82  2004        1960         JR NZ, NOTFUL
0D84  CF          1970         RST 8           ; WAIT FOR KEY PRESS
0D85  3E0C        1980         LD A, £0C       ; CLEAR SCREEN CODE
0D87  F7          1990         RST £30         ; NAS-SYS OUTPUT
0D88  C9          2000 NOTFUL  RET             ; RETURN
                  2010 ;
                  2020 ; SUBROUTINE TO CONVERT HEX LINE NUMBER
                  2030 ; TO DECIMAL (NUMBER IN HL REGS.)
                  2040 ;
0D89  0E04        2050 HEXDEC  LD C, A         ; MAX. LEADING SPACES
0D8B  0600        2060         LD B, 0
0D8D  D5          2070         PUSH DE
0D8E  110A00      2080         LD DE, 10
0D91  D5          2090         PUSH DE
0D92  C5          2100         PUSH BC
0D93  CDBC0D      2110 DODIV   CALL DIVIDE     ; DIVIDE SUB ROUTINE
0D96  78          2120         LD A, B
0D97  B1          2130         OR C            ; IF ZERO, FINISHED
0D98  CAA30D      2140         JP Z, DIVEND
0D9B  E3          2150         EX (SP), HL     ; ANOTHER DIV LOOP
0D9C  2D          2160         DEC L
0D9D  E5          2170         PUSH HL
0D9E  60          2180         LD H, B         ; PUT VALUE IN HL
0D9F  69          2190         LD L, C
0DA0  C3930D      2200         JP DODIV
0DA3  C1          2210 DIVEND  POP BC
0DA4  0D          2220 LEADSP  DEC C           ; DEC. LEADING SPACES
0DA5  79          2230         LD A, C
0DA6  B7          2240         OR A            ; CHECK SPACES LEFT
0DA7  FAB00D      2250         JP M, DONESP    ; IF MINUS, DONE
0DAA  3E20        2260         LD A, £20       ; ASCII SPACE CODE
0DAC  F7          2270         RST £30         OUPUT ROUTINE
0DAD  C3A40D      2280         JP LEADSP
```

```
0DB0  5D                2290 DONESP LD E, L       ; FIRST DIGIT
0DB1  7B                2300 OUTPUT LD A, E       ; LOAD EACH DIGIT
0DB2  FE0A              2310        CP £0A        ; COMPARE WITH TEN
0DB4  D1                2320        POP DE        ; FOLLOWING DIGIT
0DB5  C8                2330        RET Z         RETURN TO MAIN PROG.
0DB6  C630              2340        ADD A, £30    ; DECIMAL TO ASCII
0DB8  F7                2350        RST £30       ; OUTPUT DIGIT
0DB9  C3B10D            2360        JP OUTPUT
                        2370 ; DIVIDE SUBROUTINE
0DBC  E5                2380 DIVIDE PUSH HL       ; DIVIDE HL BY DE
0DBD  6C                2390        LD L, H       ; PUTTING RESULT N BC
0DBE  2600              2400        LD H, 0       ; AND REMAINDER IN HL
0DC0  CDC70D            2410        CALL DIVLOP
0DC3  41                2420        LD B, C
0DC4  7D                2430        LD A, L
0DC5  E1                2440        POP HL
0DC6  67                2450        LD H, A
0DC7  0EFF              2460 DIVLOP LD C, £FF
0DC9  0C                2470 DIVADD INC C
0DCA CDD20D             2480        CALL DIVSUB
0DCD  D2C90D            2490        JP NC, DIVADD
0DD0  19                2500        ADD HL, DE    ; ADD TEN IF CARRY
0DD1  C9                2510        RET
0DD2  7D                2520 DIVSUB LD A, L       ; LEAST SIG. BYTE
0DD3  93                2530        SUB E         ; SUBTRACT TEN
0DD4  6F                2540        LD L, A
0DD5  7C                2550        LD A, H       ; MOST SIG. BYTE
0DD6  9A                2560        SBC A, D      ; SUB 0 PLUS CARRY
0DD7  67                2570        LD H, A
0DD8  C9                2580        RET
                        2590 ;
                        2600 ; CONVERSION TABLE TRS-80 THEN NASCOM
                        2610 ;
0DD9                    2620 TABLE EQU £0DD9
                        2630 ;
                        2640 ; HEX DUMP OF TABLE

0DD9  80 80 81 81 82 9D 83 9C 84 99 87 82 88 83 89 84

0DE9  8A 85 8B 86 8C 87 8D 88 8E 89 8F 8A 90 8B 91 8C

0DF9  92 8D 93 8E 94 8F A0 90 A1 91 B0 94 B1 95 B2 9E

0E09  B3 9F B4 A0 B8 A1 B9 A2 BA A3 BB A4 BC A5 BD A6

0E19  BE A7 C1 B9 C6 C7 CA A9 CB AA CC AB CD AC CE AD

0E29  CF AE D0 AF D1 B0 D2 B1 D3 B2 D4 B3 D5 B4 D6 B5

0E39  D7 B6 D8 B7 D9 B8 DA BA DB BB DC BC DD BD DE BE

0E49  DF BF E0 C0 E1 C1 E2 C2 E3 C3 E4 C4 E5 C5 F3 C8

0E59  F4 C9 F5 CA F6 CB F7 CC F8 CD F9 CE FA CF 00 00
```

```
0C80  3E CF D3 06 D3 06 EF 0C 00 21 0A 08 11 00 10 CD
0C90  B8 0C FE A5 20 F9 CD AF 0C FE 55 20 F2 CD AF 0C
0CA0  77 2C FE 3C 20 F7 2C CD AF 0C 77 12 13 18 F8 CD
0CB0  B2 0C CD B5 0C CD B8 0C C5 F5 06 53 10 FE DB 04
0CC0  EE FF 1F 30 F9 DB 04 EE FF 1F 30 F2 06 81 10 FE
0CD0  00 00 00 06 3D 00 DB 04 EE FF 1F 38 08 10 F6 F1
0CE0  C1 07 CB 87 C9 DB 04 EE FF 1F 38 04 10 E7 18 EF
0CF0  F1 00 00 00 00 00 00 00 00 10 F6 C1 07 CB C7 C9
0D00  21 02 10 56 2D 5E D5 2D 2B 54 5D 23 4E 06 00 23
0D10  7E 23 E5 66 6F E5 DD E1 DD 09 41 0E 00 DF 66 00
0D20  EF 20 00 E1 D5 23 13 7E 12 81 4F 10 F8 23 7E B9
0D30  28 24 EF 0D 43 68 65 63 6B 73 75 6D 20 65 72 72
0D40  6F 72 20 69 6E 20 00 EB E3 23 DF 66 00 DF 6A 00
0D50  2B E3 EB CF 00 00 23 7E FE 3C 20 03 F1 18 AC FE
0D60  78 28 16 EF 0D 4E 6F 20 23 33 43 20 61 74 20 00
0D70  DF 66 00 DF 6A 00 DF 5B 00 EF 0D 0D 47 4F 4F 44
0D80  20 52 45 41 44 0D 50 72 6F 67 72 61 6D 20 72 75
0D90  6E 73 20 66 72 6F 6D 20 00 F1 E3 DF 66 00 EF 74
0DA0  6F 20 00 DD E5 E1 DF 66 00 E1 EF 2E 0D 53 74 61
0DB0  72 74 20 61 64 64 72 65 73 73 20 00 23 7E 23 66
0DC0  6F DF 66 00 EF 2E 0D 00 DF 5B 00 00 00 00 00 00
```




**READING MACHINE CODE TAPES**

Enter E 0C80 and start the tape. The routine displays the name of the program on the tapefollowed by a ‘prompt’, >. As each byte is received from the tape it is displayed on the screen (as in a Nascom tape read). When no further data is received turn off the recorder and press RESET. All the data on the tape, including the block lengths, addresses, and sync bytes, is now in memory from £1000 upwards. Enter E 0D00; if the program has loaded correctly, i.e., if all the checksums are correct, the routine will list the address where the program should be located and the ‘start’ address. If there are any checksum errors, the addresses of these blocks are listed – try reading the tape again.

Please note (i) The routine does not transfer the program to its correct address
(ii) The data blocks are 128 characters long.

## TOKEN VALUES FOR TRS-80, MICROSOFT, AND CRYSTAL BASIC

| COMMAND | TRS | NAS | XTL | COMMAND | TRS | NAS | XTL | COMMAND | TRS | NAS | XTL |
|---|---|---|---|---|---|---|---|---|---|---|---|
| END | 80 | 80 | 80 | LSET | AB | | | < | D6 | B5 | B1 |
| FOR | 81 | 81 | 81 | RSET | AC | | | SGN | D7 | B6 | B2 |
| RESET | 82 | 9D | | SAVE | AD | | | INT | D8 | B7 | B3 |
| SET | 83 | 9C | | SYSTEM | AE | | | ABS | D9 | B8 | B4 |
| CLS | 84 | 99 | | LPRINT | AF | | | FRE | DA | BA | |
| CMD | 85 | | | DEF | B0 | 94 | 96 | INP | DB | BB | B6 |
| RANDOM | 86 | | | POKE | B1 | 95 | 97 | POS | DC | BC | B7 |
| NEXT | 87 | 82 | 82 | PRINT | B2 | 9E | 98 | SQR | DD | BD | B8 |
| DATA | 88 | 83 | 83 | CONT | B3 | 9F | 99 | RND | DE | BE | B9 |
| INPUT | 89 | 84 | 85 | LIST | B4 | A0 | 9A | LOG | DF | BF | BA |
| DIM | 8A | 85 | 86 | LLIST | B5 | | | EXP | E0 | C0 | B8 |
| READ | 8B | 86 | 87 | DELETE | B6 | | | COS | E1 | C1 | BC |
| LET | 8C | 87 | 88 | AUTO | B7 | | | SIN | E2 | C2 | BD |
| GOTO | 8D | 88 | 89 | CLEAR | B8 | A1 | 9B | TAN | E3 | C3 | BE |
| RUN | 8E | 89 | 8A | CLOAD | B9 | A2 | 9C | ATN | E4 | C4 | BF |
| IF | 8F | 8A | 8B | CSAVE | BA | A3 | 9D | PEEK | E5 | C5 | C0 |
| RESTORE | 90 | 8B | 8C | NEW | BB | A4 | 9E | CVI | E6 | | |
| GOSUB | 91 | 8C | 8D | TAB( | BC | A5 | A1 | CVS | E7 | | |
| RETURN | 92 | 8D | 8E | TO | BD | A6 | A2 | CVD | E8 | | |
| REM | 93 | 8E | 90 | FN | BE | A7 | A3 | EOF | E9 | | |
| STOP | 94 | 8F | 91 | USING | BF | | | LOC | EA | | |
| ELSE | 95 | | | VARPTR | C0 | | | LOF | EB | | |
| TRON | 96 | | | USR | C1 | B9 | | MKI$ | EC | | |
| TROFF | 97 | | | ERL | C2 | | | MKS$ | ED | | |
| DEF STR | 98 | | | ERR | C3 | | | MKD$ | EE | | |
| DEF INT | 99 | | | STRING$ | C4 | | | CINT | EF | | |
| DEF SNG | 9A | | | INSTR | C5 | | | CSNG | F0 | | |
| DEF DBL | 9B | | | POINT | C6 | C7 | | CDBL | F1 | | |
| LINE | 9C | | | TIME$ | C7 | | | FIX | F2 | | |
| EDIT | 9D | | 8F | MEM | C8 | | | LEN | F3 | C8 | C1 |
| ERROR | 9E | | | INKEY$ | C9 | | | STR$ | F4 | C9 | C2 |
| RESUME | 9F | | | THEN | CA | A9 | A5 | VAL | F5 | CA | C3 |
| OUT | A0 | 90 | 92 | NOT | CB | AA | A6 | ASC | F6 | CB | C4 |
| ON | A1 | 91 | 93 | STEP | CC | AB | A7 | CHR$ | F7 | CC | C5 |
| OPEN | A2 | | | + | CD | AC | A8 | LEFT$ | F8 | CD | C6 |
| FIELD | A3 | | | - | CE | AD | A9 | RIGHT$ | F9 | CE | C7 |
| GET | A4 | | | * | CF | AE | AB | MID$ | FA | CF | C8 |
| PUT | A5 | | | / | D0 | AF | AC | | FB | | |
| CLOSE | A6 | | | ^ | D1 | B0 | AA | | FC | | |
| LOAD | A7 | | | AND | D2 | B1 | AD | | FD | | |
| MERGE | A8 | | | OR | D3 | B2 | AE | | FE | | |
| NAME | A9 | | | > | D4 | B3 | AF | | FF | | |
| KILL | AA | | | = | D5 | B4 | B0 | | | | |

NOTE: PRINT @ will convert correctly, but it is not in the Nascom Microsoft. The Crystal Basic PRINT @ is followed by the column and row of the printing position (similar format to the SCREEN command), but the TRS-80 PRINT @ uses a single number to express the screen position; the top left of the screen is 0, top right 64, bottom left 960, bottom right 1023. Similarly, there may be variations in the operation of other commands in the three Basics.

io systems ltd.

## A/D BOARD FOR NASCOM £135 + VAT

Fast Analogue to Digital conversion on the NASCOM

- * 8 Bit resolution
- * 8 input channels
- * 30 microsecond conversion
- * Prototyping area
- * Sample and hold
- * Overvoltage protection
- * Full flag/interrupt control
- * Built and tested

## EPROM PROGRAMMER £63 + VAT

* Programs 2708/2716 – 3 rail
2508/2758 – 1 rail
2516/2716 – 1 rail
2532/2732 – 1 rail

* Zero insertion force socket

* Built and tested

## GRAPHICS BOARD FOR NASCOM £55 + VAT

Very high resolution graphics on your NASCOM

- * 384 x 256 bit mapped display
- * Mixed text and graphics
- * 4Mhz NASCOM required
- * Graphics software supplied
- * Full software control
- * Built and tested

## DUNCAN LANGUAGE FOR NASCOM £12 + VAT

* Duncan is a fast real time interpreter / control language for NASCOM and was featured in “PRACTICAL COMPUTING” May 81.

---

6 LALEHAM AVENUE, MILL HILL, LONDON, NW7 3HL. Tel. 01-959 0106

## AUTO GRAPHICS SELECTION ON A NASCOM

Most Nascom-2s are equipped with the graphics ROM, and many Nascom-1s have some form of graphics capability, either by means of the sadly-departed Econographics kit or a locally-produced or commercial system. Often there is need to switch between two sets of graphics if you use special characters - for example, to display the pieces for a chess program such as Sargon.

This can be done by switching the $\overline{CE}$ lines of the chips on and off with mechanical keys, or, even worse, by using one line from the PIO as a latch to enable the required ROM; this ties up the PIO needlessly.

The simple circuit described here uses one of the two spare output lines - from port 0, the keyboard port. The spare lines are bits 2 and 5 of this port. The status of the port is reflected at £0C00. By modifying £0C00 to set the selected bit to 1 the corresponding line is set high without affecting the other lines, and it stays that way until set back to zero by a program command, or until the RESET button is pressed. If a program uses the special graphics ROM, you merely have to include the following machine code routine at the start of the programs :-

```
3E 04      LD A, 4          ; BIT 2 - USE LD A, 32 FOR BIT 5
32 00 0C   LD (£0C00), A    ; CHANGE TO 2ND GRAPHICS ROM
```

and at the end of the programs:-

```
AF         XOR A            ; SET A TO ZERO
32 00 0C   LD (£0C00), A    ; RESTORE STANDARD GRAPHICS
```

**CONSTRUCTION**

Make up a "piggy-back" board, using a small piece of Veroboard or a small PCB, with one 24-pin wirewrap socket, one normal 24-pin socket, and one 14-pin socket. Cut pin 18 off the wirewrap socket, leaving about 1/4 inch for wiring. Connect pins 1 — 17 and 19 — 24 -from the wirewrap to the normal 24-pin socket. The 14 pin socket is wired as shown in figure 1, and the 74LS00 is plugged in. The standard graphics chip is placed in the normal socket, and the alternative ROM in the wirewrap socket. The board is then inserted into the socket vacated by the graphics chip on the main board, using the extended leads of the wirewrap socket as a plug. Connect a wire from the keyboard socket (pin 13 for bit 2 on a Nascom 1, pin 8 on a Nascom 2) to the input of the 7400 flip-flop as shown.

The circuit is shown for 2716-compatible chips, but the principle applies to almost any ROMs or EPROMs - just be sure that you wire the outputs from the flip-flops to the correct pin on the I.C.s you use. The 2716 chip can be ‘selected’ by voltages applied to pins 18 and 20. Pin 20 is the chip select line ($\overline{CS}$), while pin 18 is Power down/Program line. If EITHER line is taken to +5 volts the data lines of the switch to a high-impedance state. In the case of a 2708, only pin 20 can be used to select the chip.

The circuit can be used to switch between two sets of graphics held in a single 4K EPROM – a 2532. Only one 24-pin socket is required, and the output from the flip-flop is again connected to pin 18 of this socket – but in this case this is the top address line, switching between the two sets of 2K graphics data in the ROM. If the standard Nascom-2 graphics data is stored in the bottom 2K of the chip, pin 18 should be connected to pin 8 of the 74LS00.

COMPONENTS REQUIRED

1 wirewrap 24-pin socket
1 standard 24-pin socket
1 14-pin socket
1 10 kohm resistor
Veroboard

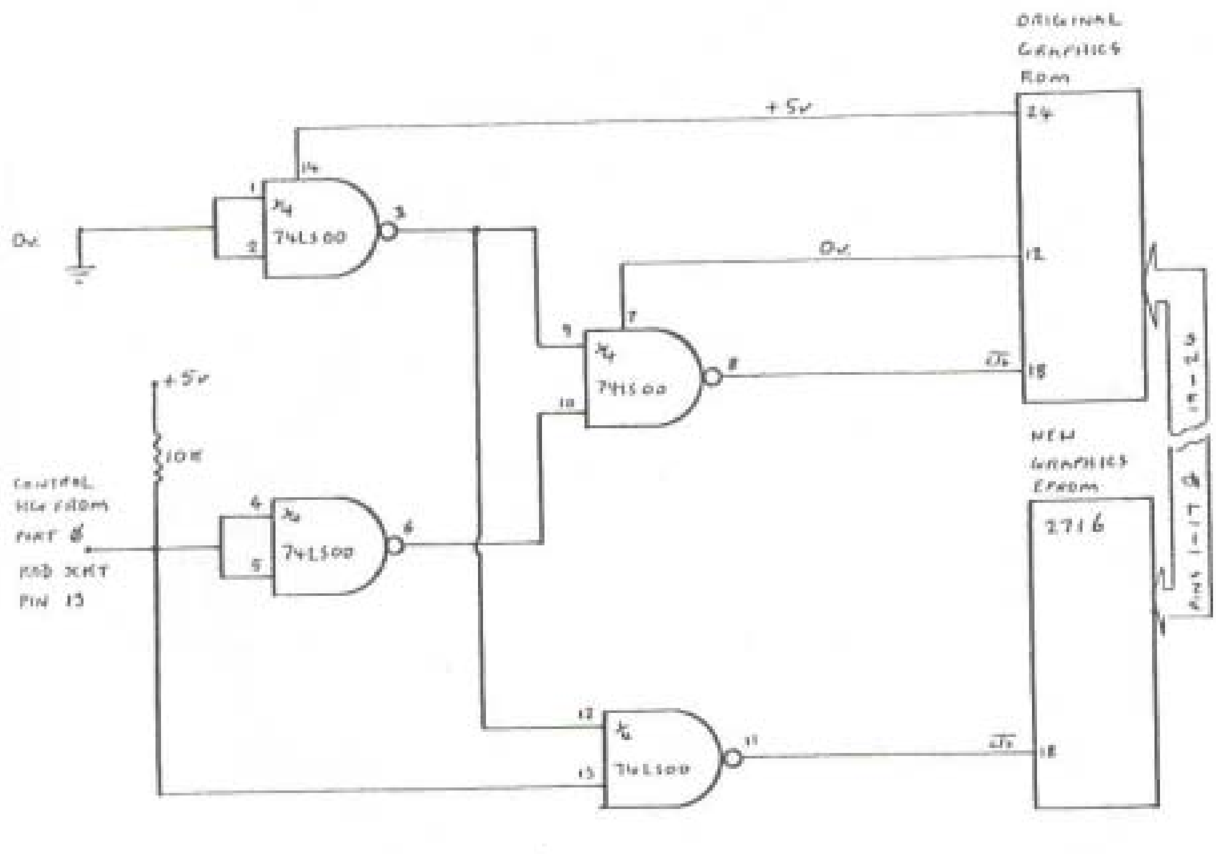



FIG. 1

```
10 REM   * HANGMAN * (C) D.G.Johnson 1981
20 REM   ~~~~~~~~~~
30 REM   SET UP M/C CODE AND PUT UP TITLE
40 REM   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
42 CLEAR 500:DIM M$(4):Z$=CHR$(0)
44 DATA27085,14336,-13564,6399,18178,10927,-817
9,233
50 DATA 31711, 1080, -53,536,-20665,3370,-5664,0
52 IF PEEK(1)=0 THEN RESTORE 50
60 DOKE 4100,3340:FOR I=3340 to 3354 STEP 2
70 READ J:DOKE I,J:NEXT
80 CLS:F=0:A$= "* HANGMAN *":FOR I=1 TO 11
90 POKE 3036+I, ASC(MID$(A$,I,1)):NEXT
100 REM SEARCH FULL LIST AND CHOOSE WORD
110 REM ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
130 RESTORE 9000:W=-1
140 READ A$:W=W+1:IF A$ <> "." THEN 140
150 RESTORE 9000
160 I=0 TO INT(RND(1)*W):READ A$:NEXT
162 FOR J=1 TO LEN(Z$)
164 IF I=ASC(MID$(Z$,J,1)) THEN F=1
166 NEXT: IF F THEN F=0:GOTO 150
168 Z$=Z$+CHR$(I)
170 B$= "":C$= "": G$= "": G=0: H=0: L=LEN(A$)
180 FOR I=1 TO L:B$=B$+ "_":NEXT
190 S=48-2*L:IF S>28 THEN S=28
200 REM       START
210 REM       ~~~~~
220 X=S:Y=5:GOSUB 900:PRINT "Your guess"
230 X=S:Y=3:GOSUB 900
240 FOR I=1 TO L
250 PRINT MID$(B$,I,1); " ";
260 NEXT
270 IF B$<>A$ THEN 350
280 M$(1)="Well done! You win!"
290 M$(2)="                    "
292 IF RND(1)>.2 THEN 300
294 M$(1)="O.K. Smartie!   But I"
296 M$(2)="will  still  hang   you!"
298 H=1:GOSUB1000:H=2:GOSUB1000:H=9:GOSUB1000
300 M$(3)="If    you    would    like"
310 M$(4)="another game press y"
312 GOSUB 920
320 C=USR(0):IF C<0 THEN 320
330 IF CHR$(C)="Y" THEN 80
340 CLS:X=20:Y=7:GOSUB 900
342 PRINT "Goodbye!":END
350 G=G+1
360 X=S:Y=6:GOSUB 900
362 PRINT "No. ";G;" please."
370 X=27+G:Y=9:GOSUB 900
380 C=USR(0): IF C<0 THEN 350
390 C$=CHR$(C): PRINT C$
400 FOR I=1 TO 4
410 M$(I)="                              "
420 NEXT
430 GOSUB 920
440 FOR I=1 TO LEN(G$)
450 IF MID$(G$,I,1)=C$ THEN F=1
460 NEXT: IF F THEN F=0:GOTO 680
462 G$=G$+C$
```

```
470 FOR I=1 TO L
480 IF MID$(A$,I,1)<>C$ THEN 510
490 B$=LEFT$(B$,I-1)+C$+RIGHT$(B$,L-I)
500 F=1
510 NEXT
520 IF F THEN F=0:GOTO 630:REM Good guess
530 M$(1)="        * WRONG  *        "
532 M$(2)="                          "
534 M$(3)="                          "
536 M$(4)="                          "
540 IF G<10ORG=11ORG=12 THEN 562
550 M$(3)="This   looks   dangerous"
560 M$(4)="You'll   be   hung   soon!"
562 GOSUB 920
570 GOSUB 1000: REM Next step in hanging
580 IF F THEN F=0:GOTO 600: REM If hung
590 GOTO 230: REM Loop back for next guess
600 M$(1)="You lose!          The word"
610 M$(2)="was   "+A$
620 GOTO 300: REM Another game?
630 M$(1)="      * SUCCESS  *       "
640 M$(2)="                          "
650 M$(3)="                          "
660 M$(4)="                          "
662 GOSUB 920
670 GOTO 230: REM Loop back for next guess
680 M$(1)="You have  already  made"
690 M$(2)="that guess.               "
700 M$(3)="I   do   not   allow   such"
710 M$(4)="duplication.              "
712 GOSUB 920
720 GOTO 570:REM Back to wrong guess loop
900 SCREEN 1,1:PRINT CHR$(23):SCREEN X,Y
910 RETURN
920 FOR I=1 TO 4
930 J=LEN(M$(I))
932 IF J>20 THEN PRINT "Message too long":STOP
940 IF J<20 THEN M$(I)=M$(I)+" ":GOTO 930
950 X=28:Y=11+I:GOSUB 900
960 PRINT M$(I);
970 NEXT
980 RETURN
1000 H=H+1:IF H=10 THEN F=1
1010 IF H>1 THEN 1110
1020 FOR I=1 TO 15:SCREEN 1,I
1030 PRINT CHR$(255);:NEXT
1040 FOR I=2 TO 16:SCREEN I,1
1050 PRINT CHR$(219):NEXT
1060 FOR I=2 TO 16
1070 SET(I,18-I):NEXT
1100 RETURN
1110 IF H>2 THEN 1190
1120 FOR I=12 TO 15
1130 SCREEN 6,I:PRINT CHR$(128);
1140 SCREEN 26,I:PRINT CHR$(128);:NEXT
1150 SCREEN 8,12:FOR I=1 TO 17
1160 PRINT CHR$(129);:NEXT
1170 POKE 2768,255:POKE 2786,255
1180 POKE 2832,133:POKE 2850,132
1182 RETURN
1190 RESTORE 8000
```

```
1200 FOR I=1 TO 4*H+4
1210 READ J,K:POKE J,K:NEXT
1220 IF F=0 THEN 1400
1230 SCREEN 8,12:PRINT CHR$(23):SCREEN 8,12
1240 PRINT SPC(17)
1320 RESTORE 8000
1330 FOR J=1 TO 44:READ K,L:POKE K,32:NEXT
1340 POKE 2137,148
1350 RESTORE 8000
1360 FOR J=1 TO 44:READ K,L:POKE K+64,L:NEXT
1370 POKE2264,185:POKE2266,185:POKE2329,0
1380 POKE2839,157:POKE2840,157
1390 POKE2842,157:POKE2843,157
1400 RETURN
8000 DATA 2137,153,2136,139,2138,138
8002 DATA 2713,148
8010 DATA 2199,131,2203,130,2263,130
8020 DATA 2267,131,2328,157,2329,086
8030 DATA 2330,157,2198,144,2200,111
8040 DATA 2202,111,2204,145,2265,095
8050 DATA 2327,131,2331,130,2390,131
8060 DATA 2396,130,2453,131,2461,130
8070 DATA 2455,148,2459,148,2460,130
8080 DATA 2454,131,2523,148,2519,148
8090 DATA 2517,079,2525,079,2583,148
8100 DATA 2587,148,2585,148,2647,148
8110 DATA 2649,148,2651,148,2710,144
8120 DATA 2711,147,2715,146
8130 DATA 2716,145,2392,094,2548,094
8140 DATA 2520,094,2586,094
9000 DATA AIREDALE , AUTONOMY , BEFUDDLED
9002 DATA BOARDINGHOUSE , CAMARADERIE, CAMBER
9004 DATA FREQUENCY, PROBATION, KNIGHTHOOD
9010 DATA WATERMELON, BREADFRUIT, SATSUMA
9020 DATA HAMMER, CHISEL, SCREWDRIVER, PLIERS
9030 DATA BRADAWL, PLANE, WORKBENCH
9040 DATA ASPIDISTRA , CLEMATIS, MARIGOLD
9050 DATA NASTURTIUM, NARCISSUS, DAFFODIL
9060 DATA JACKET, TROUSERS, RAINCOAT, SOCKS
9070 DATA CUMMERBUND, CRAVAT, JUMPER, CARDIGAN
9080 DATA TROMBONE, PIANOFORTE, BASSOON, TUBA
9090 DATA CELLO, TRUMPET, CYMBAL, GUITAR
9100 DATA CONSTANTINOPLE , COINCIDENCE
9110 DATA CIRCUMLOCUTION, PARAPSYCHOLOGY
9120 DATA CRUMBLE , CHEQUEBOOK , CASSETTE, ENVELOPE
9130 DATA COMPUTER , ELECTRICITY, CEILING, RADIATOR
9140 DATA PRINTING, SPECIALITY, FLOWERPOT
9150 DATA ESTRANGE, ZEPHYR, ZEALOUS, XYLOPHONE
9160 DATA YASHMAK, WISECRACK , WINDSHIELD, ZIP
9170 DATA WHEREWITHAL, VIBRATION, VESTIBULE
9180 DATA VERMICELLI, TRUCULENT, CHROMATIC
9190 DATA SYNONYM, SYNCHRONIZE, SCYTHE
9200 DATA PIN, SIT, SET, RACKET , RACECOURSE
9210 DATA QUOTATION, QUICKSILVER, QUARTERMASTER
9220 DATA PULP, PUFFBALL, PROXIMITY, PSYCHOTIC
9230 DATA ILLUSTRATION, PRAGMATISM, POSTMAN
9240 DATA POLYGON, ARCHITECTURE, ORTHODOX
9250 DATA ONYX , OMNIBUS , NONCONFORMIST, TRY
9290 DATA MURMUR, MYOPIC , MEADOWSWEET, MAYOR
9999 DATA .
```

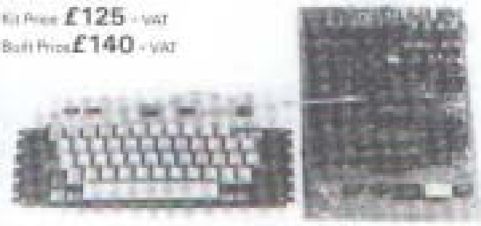
# nascom

## .... Computing for everyone

Nascom products, designed and built in Britain, are now backed by Lucas one of Britain's foremost industrial companies. This is a vindication of the innovative design of the Nascom computer and an assurance of its future.

### NASCOM-1

Kit Price **£125** + VAT
Built Price **£140** + VAT



12" x 8" PCB carrying 5L SI MOS packages, 16 1K MOS memory packages and 33 TTL packages. There is on board interface for UHF or unmodulated video and cassette or teletype. The 4K memory block is assigned to the operating system, video display and EPROM option socket, leaving a 1K user RAM complete with keyboard.

### NASCOM 2

Kit includes all parts to build CPU board which has resident 8K microsoft BASIC and 2K NAS-SYS 1 monitor for machine code programming. Included with kit is a fully assembled LICON QWERTY SOLID STATE KEYBOARD specially designed to exploit the potential of the NAS-SYS monitor. Other interfaces include video to monitor or domestic TV, Kansas City standard cassette interface (300/1200 baud) or RS232/20mA teletype interface.
In addition to full character generator graphics ROM is provided to give BASIC on board graphics capability.
System uses Z80A which gives selectability between 2 or 4 MHz.

Nascom 2 Kit Price **£225** + VAT

Power supply—3 amp. Suitable for powering of basic Nascom 1 or 2 and memory expansion.

**£32.50** + VAT

### RAMBOARD

SERIES B ram board gives user option of 16K DYNAMIC RAM. This board can be arranged in page mode to allow use of up to 4 with NASCOM 2. Boards are fully buffered but PAGE MODE facility is an optional extra. This card can be used at 4MHz without wait state.

16K **£120** + VAT

### NASCOM IMP PLAIN PAPER PRINTER


The Nascom IMP (Impact Matrix Printer) features:
● 60 lines per minute ● 80 characters per line ● Bi-directional printing ● 10 line print buffer ● Automatic CR/LF ● 96 characters ASCII set (includes upper/lower case, $, £) ● Accepts 8½" paper (pressure feed) ● Accepts 9½" paper (tractor feed) ● Tractor/pressure feed ● Baud rate from 110 to 9600 ● External signal for optional synchronisation of baud rate ● Serial RS232 interface

**£325** + VAT

### NASCOM FIRMWARE

CPU card can accommodate either 8K of static memory or 8 2708 EPROMS. This allows for inclusion of standard firmware on board.
**ASSEMBLER** Version 2.0 of ZEAP (Z80 Editor Assembler Package) offers in 4K features found normally only in far larger programs.
A comprehensive line editor is provided in addition to an assembler operating in standard Z80 mnemonics. Direct assembly to memory allows immediate program execution. ZEAP can take advantage of special features of NAS-SYS, which was itself developed on this assembler. Supplied on tape at **£30.00** plus VAT or in 4 x 2708 EPROMs at **£50.00** plus VAT.
**DISASSEMBLER** The NAS-DIS 3K disassembler reverses the effect of assemblers such as ZEAP by turning machine code into assembler program, automatically labelling and cross referencing to produce a complete program listing, saving hours of tedious hand disassembly when program analysis is required. Supplied in 3 x 2708 EPROMs at **£37.50** plus VAT.
**DIAGNOSTIC PACKAGE** NAS-DEBUG is a 1K addition to NAS-DIS which provides remarkable facilities for error elimination, including a full register display which may be edited by the cursor. An unusual feature is the provision for examination of the program in assembler as the machine single steps through it. A second video page may be assigned to allow work on programs which use the screen. A very powerful assembler-based system for program development could be realised on a NASCOM-2 with appropriate external memory by fitting the 8 ROMs containing ZEAP, NAS-DIS and NAS-DEBUG into the sockets on the computer board. This system would function immediately on switching on, without needing programs to be loaded from tape. Supplied in a 2708 EPROM at **£15.00** plus VAT and must be operated with NAS-DIS.

**NAS-SYS 3. THE NEW OPERATING SYSTEM FOR NASCOM 2.**
Supplied in 1 x 2716 EPROM.
NAS-SYS 3 is the latest in the current series of Nascom monitors and includes features such as adjustable keyboard repeat and cursor speed, full interrupt handling and a number of powerful routines and commands making this probably the most comprehensive 2K monitor ever written for a microcomputer. **£40.00** plus VAT

| TO BE INTRODUCED | Price |
|---|---|
| Programmable graphics colour board | TBA |
| Twin floppy disk unit, double sided, double density with NASDOS or CPM 2.2 | TBA |

**WATCH IT!!**
Nascom products fully assembled and packaged in the near future!

**Please send SAE for list of distributors.**

**Lucas Logic**

**Nascom Microcomputers**
Division of Lucas Logic Limited
Welton Road Wedgnock Industrial Estate Warwick CV34 5PZ
Tel: 0926 497733 Telex: 312333

## NAS-SYS MONITORS

**by J.Haigh**

### LOAD L

In Nas-Sys 1 this command is used to load data from a paper tape reader. The data must have been stored on the tape in the format used by the standard Nas-Sys 1 tabulate commands i.e., the address of the first data byte, eight data bytes, checksum, all represented in hexadecimal and separated by spaces; the line is terminated by a carriage return. As the tape is read the data is displayed on the screen; when the end of the line is reached (detected by the carriage return) the data is read into the workspace by the same routine which reads the arguments supplied with commands. Thus the loading address is read into ARG1 (£0C0C, £0C0D), the eight data bytes into ARG2 - ARG9 (£0C0E to £0C1D), and the checksum into ARG10 (£0C1E, £0C1F). The routine then totals the values stored in ARG1 - ARG9 and compares the result with the checksum. If the values are identical the eight data bytes are copied from the workspace to the appropriate memory location, the cursor is reset to the beginning of the line, and the next block is read in, overwriting the last line. If a checksum error is detected, or if invalid characters are detected by the routine which reads the data into the workspace, the faulty line is scrolled up the screen and the routine proceeds to the next block.

Data can be written to a paper tape punch in the correct format by routing the output of the Tabulate command to the serial port by means of the External (X) command. However, in Nas-Sys 3 the Load command has been dropped; the address stored in the subroutine table for command L (at £0798) is £0366, the Error subroutine. Because the Nas-Sys 3 tabulate command is not restricted to the - format of a Load command it has been made more versatile in the line lengths it can produce, and also no longer gives a checksum byte.

### MODIFY Mxxxx

This permits direct insertion of data into memory from the keyboard. When the command is entered the address xxxx is displayed, followed by the byte currently at that address, the cursor is moved left three spaces after the routine which displays the data byte; since this routine outputs the two digits which represent this byte in hexadecimal followed by a space, this places the cursor on the first character of the byte. Data can now be typed in hexadecimal format, successive bytes being separated by one or more spaces. When the newline key is pressed the monitor interprets the current line; the first number on the line is taken as the address at which data storage is to start. If the first group of characters on the line is not a valid hexadecimal number, that is, does not consist only of the ASCII characters 0 - 9 and A - F representing a hexadecimal number between 0 and FFFF the word ‘Error’ is printed, and the routine restarts at the last valid address.

If a valid address is obtained subsequent hexadecimal numbers on the line are entered into memory until the end of the line is reached (detected by means of the nulls with which the screen margins are filled) or until a non-valid entry is found. If all the numbers are valid the modify routine continues on the next line, displaying the updated address and the byte at that address, when further data can be entered.

Although the data to be entered is in bytes, the routine which evaluates the successive groups of characters is designed to handle sixteen-bit values, but only the least significant eight bits are put into memory, thus FA, 1FA and 37FA will all go into memory as FA. If the number exceeds FFFF an error message will be generated and the routine will reset to the address at the start of the line, but data will have been entered into memory up to and including the first invalid entry.

If a character is encountered which does not lie in the ranges 0 - 9 or A - F the above error process will normally occur, but here are four exceptions to this. A full stop terminates the Modify command and returns control to the monitor. An oblique stroke changes the address to be modified to the hexadecimal number following the stroke; an error message is produced if the characters following the stroke are not in the ‘hexadecimal’ set, but if no number is entered the address changes to zero, A colon causes the routine to backstep one address. Because the Modify routine leaves the current line when it encounters one of the above three characters, either to return to monitor or to start a new line, you cannot use more than one character per line; you cannot, for example, backstep three spaces be entering 0C90 **:::** N/L; only the first will be effective and address 0C8F will be displayed.

The fourth ‘special’ character is the comma; this causes the ASCII code of the following character to be entered into memory. In this case you can enter as many codes as will fit on the line, and you can mix them freely with the usual hexadecimal codes. For example

EF,H,E,L,L,O 00

will be entered as

EF 48 45 4C 4C 4F 00

Note that you do not need to enter spaces to separate the bytes in addition to a comma.

**NORMAL N**

This command resets the addresses of the output and input tables, stored at £0C73 and £0C75, so that output is routed only to the CRT and input is accepted only from the keyboard and serial input port. The U command changes these addresses so that input and output first calls user routines previously specified at £0C7B and £0C78. Thus once the address of a printer routine, for example, has been stored at £0C78 - £0C79, commands U and N can be used from the keyboard, and DF 55 and DF 4E can be used within programs, to turn the printer on and off.

**OUTPUT 0 xx yy**

This routine sends data yy to port xx. The port number, xx, is placed on the bottom eight address lines, A0 - A7, and this is decoded to determine which input/output device is activated. The data to be sent, yy, is placed on the data bus, and the activated device receives it. The main use of the output command is to communicate with external devices via the PIO chip, so perhaps a few words on the operation of this device would not be out of place here.

The MK3881 PIO chip used on the Nascom is a programmable input/output chip which the main processor sees as four ports; of these, two ports each provide eight lines, which can be programmed to be input, output or bidirectional, for external communication, while the remaining two ports are used to control the operation of the chip. On the basic Nascom port A is addressed as port 4, and it is controlled by port 6; port B is addressed as port 5 and its control port is port 7.

A convenient way to study the operation of the PIO chip is to use the 0 and Q commands to write to and read from the PIO, while monitoring the state of the lines with a logic probe, or with the Bits and P.C.s port probe.

**PREGS**

In the Nas-Sys 1 monitor command P merely produces an error message; in Nas-Sys 3 the command prints out the contents of the procesors main registers, previously stored in the workspace from £0C61 to £0C6C, together with the current contents of the I, IX and IY registers. Virtually the same code is used to display the registers in Nas-Sys 1, although the format of the display is slightly different (see the S command), but it is not written as a subroutine, so it cannot be accessed from the keyboard or from user programs.

**QUERY Q xx**

Q xx obtains data from port xx and displays it on the screen in hexadecimal format. To be able to obtain data via the PIO chip you will have to program the chip by writing to the appropriate control port using the 0 command.

**READ R xxxx (y)**

The Read command loads data from a cassette tape written in the format used by the standard Write command. After turning on the tape LED the routine sets the input/output table addresses to their ‘normal’ values, saving the addresses that were at £0C73, £0C75 on the stack, so that they can be restored at the end of the Read routine. This means that the Read routine can be called from any program, even if it uses special tables for input/output routines.

The routine then scans the keyboard and the serial input to find either the four consecutive FF's which mark the start of each block of tape data, or the four 'escapes' from the keyboard which will abort the Read routine. When the block start is found the next four bytes, which indicate the loading address, the block length and the block number, are loaded into HL and DE. They are then displayed on the screen by routine 6C, which simultaneously adds the four bytes together, returning with the value in the C register. The next byte received from the tape is compared with this checksum in C, and if the values are not identical a qestion mark is printed on the screen and subsequent data is ignored until the next 'start of block' marker is found.

If the checksum for the block header is correct Nas-Sys 3 checks to see if an argument has been entered with the Read command. If it has, this argument is added to the loading address, so that the data can be loaded to a different address from that specified in the write command. In Nas-Sys 1 this facility is not available.

The data is now read from the tape; if the command letter entered is R (i.e., we are doing a Read) the data is loaded into memory; if not (for example, if we are using the Verify command) data is not stored. In either case the bytes are summed into register C as they are received from tape. When a number of bytes equal to the block length specified in the header has been received, the total in C is compared with the next byte - the data checksum. If the two are not identical, a question mark is printed; however, the faulty data bytes or bytes which caused the checksum error have been loaded into memory.

If the checksum test is passed a full stop is printed and the routine then checks to see if the block just loaded was the last block. If not, the routine looks for the next 'start of block' marker. When the last block is detected the input/output addresses are reset and the routine terminates by jumping to the subroutine which flips the tape LED.

The Read routine has two faults. Firstly, it loads faulty data; if you are trying to load a tape which is producing a lot of read errors you cannot load a program correctly by reading the tape repeatedly, even if you have several copies of the same program on the tape, because blocks which have loaded correctly are corrupted by faulty reads in subsequent passes. Of course, you can overcome this by copying the program to a different location and then recopying blocks which initially read incorrectly as error-free reads are obtained. However, this is a fiddly task, and in any case you can't use this method if the program is longer than half the available memory. A second fault is that blocks can be missed entirely without an error message being produced if one of the FF's in the 'start of block' marker is misread.

In order to overcome these faults I use a slightly different Read routine, which puts the tape data into a buffer and only transfers it to the correct memory location if the checksum tests are passed. But where can you locate the buffer? Wherever you put it, sooner or later you will want to load a program to that location. The only solution seems to be to use the screen RAM for temporary storage. The program therefore starts by clearing the screen; as it uses the margins as well as the 'visible' screen RAM, it also clears the screen at the end of the routine to restore the zeroes which the delineate the margins. A tally is kept on the screen of blocks which have been read correctly, and when all the blocks have been obtained the routine stops.

The revised Read incorporates the 'load offset' of Nas-Sys 3. A second argument can be used to force transfer of data from the buffer to memory, even when the checksum is wrong. This ensures that if you only have one copy of a program, and a persistent error on the tape, you don' t loose the whole block.

Because bad data is not written to memory, it is not necessary to use a separate verify command. To verify a tape you have just recorded you merely read it back with the R command - if the recording was faulty it will not corrupt the stored program. Therefore the routine does not test the value stored at ARGX (£0C2B), which is how the standard Nas-Sys routine distinguishes between Read and Verify. Consequently unless you change the address for the V command this will also read a tape into memory.

```
EF0C00    READ    DEFB £EF £0C 00    ; CLEAR SCREEN
DF5F              SCAL ZMFLP         ; TURN ON TAPE LED
DF77              SCAL ZNNOM         ; RESET OUTPUT TABLE ADDRESS
E5                PUSH HL            ; SAVE OLD ADDRESS ON STACK
DF78              SCAL ZNNIM         ; RESET INPUT TABLE ADDRESS
E5                PUSH HL            ; SAVE OLD ADDRESS ON STACK
0604      R1      LD B, 4            ; LOOK FOR 4 CHARS.
CF        R2      RST RIN            ; GET CHARACTER
3C                JR NZ,   R1;T44; IF NOT, KEEP LOOKING
10FA              DJNZ R2            ; HAVE WE GOT 4 YET?
CF                RST RIN            ; NOW GET HEADER BYTES
6F                LD L, A            ; FIRST BYTE INTO L REG.
CF                RST RIN            ; SECOND BYTE
67                LD H, A            ; INTO L REGISTER
CF                RST RIN            ; THIRD BYTE
5F                LD E, A            ; INTO E REGISTER
CF                RST RIN            ; FOURTH BYTE
57                LD D, A            ; INTO D REGISTER
EF1B00            DEFB £EF £1B 00    ; PUT CURSOR BACK TO START
4F                LD C, A            ; SET C TO ZERO
DF6C              SCAL ZTX1          ; PRINT HL, DE: GET CHECKSUM
CF                RST RIN            ; GET NEXT BYTE
B9                CP C               ; COMPARE WITH CHECKSUM
20E6              JR NZ, R1          ; IF WRONG, START AGAIN
48                LD C, B            ; SET C TO ZERO
43                LD B, E            ; PUT BLOCK LENGTH INTO B
E5                PUSH HL            ; SAVE HL
21000A            LD HL, £0A00       ; SET HL TO BUFFER START
CF        R3      RST RIN            ; GET DATA BYTES
77                LD (HL) A          ; PUT INTO BUFFER
23                INC HL             ; INCREMENT BUFFER ADDRESS
81                ADD A, C           ; CHECKSUM CALCULATION
4F                LD C, A            ; SAVE IN C
```

```
10F9                DJNZ R3            ; KEEP GOING TO END OF BLOCK
CF                  RST RIN            ; GET NEXT BYTE
B9                  CP C               ; IS CHECKSUM CORRECT?
E1                  POP HL             ; RECOVER HL
3A0B0C              LD A (£0C0B)       ; LOAD NUMBER OF ARGUMENTS
2804                JR Z, R4           ; IF CHECKSUM O.K., JUMP
FE02                CP 2               ; SECOND ARGUMENT ENTERED?
20CD                JR NZ, R1          ; IF NOT, DON'T COPY
4B          R4      LD C, E            ; PUT BLOCK LENGTH INTO C
0D                  DEC C              ; IF C = 0 THE B MUST BE SET
03                  INC BC             ; TO 1 FOR COPY ROUTINE
A7                  AND A              ; ANY ARGUMENTS TO COMMAND?
7A                  LD A, D            ; SAVE BLOCK NUMBER IN A
2805                JR Z, R5           ; NO ARGS., SKIP OFFSET
ED5B0C0C            LD DE (£0C0C)      ; GET FIRST ARGUMENT
19                  ADD HL, DE         ; ADD OFFSET TO HL
11000A      R5      LD DE, £0A00       ; SET DE TO BUFFER
EB                  EX DE, HL          ; EXCHANGE REGISTERS AND COPY
EDB0                LDIR               ; FROM SCREEN TO LOAD ADDRESS
6F                  LD L, A            ; RECOVER BLOCK NO. FROM A
2609                LD H, 9            ; POSITION FOR BLOCK TALLY
74                  LD (HL), H         ; MARK POSITION
68                  LD L, B            ; SET L TO ZERO
7C                  LD A, H            ; PUT TALLY CHARACTER IN A
BE          R6      CP (HL)            ; IS TALLY CORRECT?
23                  INC HL             ; SCAN TALLY
2002                JR NZ, R7          ; IF NOT, SKIP
10FA                DJNZ R6            ; CHECK ALL 256
BE          R7      CP (HL)            ; NOW CHECK IF END OF TALLIES
23                  INC HL             ; KEEP SCANNING
28AB                JR Z, R1           ; IF NOT END, KEEP READING
10FA                DJNZ R7            ; SCAN ALL 256
E1                  POP HL             ; RECOVER ORIGINAL INPUT
TABLE
22750C              LD (£0C75), HL     ; RESTORE AT £0C75
EF0C00              DEFB £EF £0C 00    ; CLEAR SCREEN
DF5F                SCAL ZMFLP         ; TURN OFF TAPE LED
C3 3C 07            JP £073C           ; RESTORE OUTPUT TABLE
```

No assembly addresses are given in the above listing, because the program is essentially relocatable. It will fit in the space used by the standard read routine in either Nas-Sys 1 (£065E to £06CE) or Nas-Sys 3 (£065E to £06CB). The jump address with which the program ends should be £0741 for Nas-Sys 1 and £073C for Nas-Sys 3.

* - * - * - * - * - * - * - * - *

## NEWS FROM THE CLUBS

First a small success - as a result of a letter in the first issue of Micropower a new User Group has been formed the Nascom - Thames Valley User Group. Regular meetings in the Slough/Staines/Windsor area are planned, and the group hope to publish a newsletter. Further details can be obtained by contacting, after 7.30 p.m., Pat Dubock, STAINES 50341, Mike Rothery, WINDSOR 56106, or Ken Ford, STAINES 59662.

The Computer section of the Cornish Radio Amateur Club meets on the third Monday of each month in the S.W.E.B. Social Club, Pool, Redruth. The average attendance is 20 - 30, with a Nascom contingent of 10 - 15. The November meeting will present "Flowcharting"; in December the topic will be "Machine Code Continued " .

York Computer Club meets every Monday at the Holgate W.M.C., New Lane, Acomb (Near the Carriage Works). Any new Nascom-owning members would be very welcome, as the Nascom users are outnumbered by owners of plastic boxes from Japan, U.S.A., and Cambridge. Ring Rupert Brown on York (0904) 792023, evenings only, or drop in to the Club (bar prices are well subsidised!).

The Merseyside Nascom Group still meets on the first WEDNESDAY of each month, in spite of a note to the contrary in one of the glossies. The next meeting will be the Christmas Beanfeast, and it is hoped that representatives from Lucas will be present. Meetings are held in the Mona pub, near Pierhead.

---

## MICRO=MARKET

Small non-commercial advertisements, £2 per ad.

FOR SALE RAM A card with 8K; £45 o.n.o. RAM B card with 16K; £75 o.n.o. Both in full working order.
C. Bowden, Tel. 0209 860 480 evenings.

FOR SALE NASCOM IMP PRINTER, ANY SENSIBLE OFFER. NASCOM VERO FRAME, £12. NASCOM KEYBOARD COVER, £2.
Malcolm Connah, Tel. 01-500-1000, Ext. 109, Office hours.

FOR SALE NASCOM 1 plus buffer board, £95. NAS-SYS 3, £9. ZEAP on EPROMS, £25 (I'm buying a NASCOM 2).
Paul Thompson Tel. 041 339 8855 ext. 7120 (days)
041 332 3841 (evenings)

---

**LICON SWITCHES** with blank caps to update Nascom 1 keyboard, £2.15 each, or £19.50 for 10 (including V.A.T.). Please add 35p P&P per order. See Micropower, Issue 2 for connections.

CHIATRONIX LTD., 22, St. Michaels Avenue, Houghton Regis, Dunstable, Beds., LU5 5DN. Tel. 0582 61697.

---

PROGRAM POWER

# NASCOM 1 & 2

## Nasprint 80

Nasprint 80 is a 2K program which greatly extends and simplifies the operation of Nas-Pen. New functions supplied by Nasprint 80 includes:

Pagination
Output a page number of each page
Output a title on each page
Centre title
Text formatting with embedded control codes. e.g. Change line length; change line spacing; change margins; centre line between margins; new page; output control codes to printer.

The program contains a parallel printer routine for a Centronics type interface, specifically designed for the Epsom MX-80, but the program can be used with any printer, parallel or serial, as the output is routed through an address in RAM.

The program also facilitates the operation of a printer with Zeap, Nas-Dis, De-bug, Nas-Sys & ROM Basic; the software/firmware being used is selected fro a menu and Nasprint 80 then changes the necessary addresses to produce a hard copy output.

The program is supplied in 2x2708's for fitting 2716's in the RAM A card. £14.95

## New Fase (16K/MC/G)

New version of the space invaders type with each new fleet of invaders having a different shape & kind of motion.Missiles fired at you come straight down or diagonally left to right & vice versa.

Destroy one 'fase' & move onto the next. The fuel level is shown graphically and you can refuel if you obliterate four fleets. Your score is shown at the end of a game and the top ten scorers are ranked. Once again the difficulty level has been set very high. £7.95

## Starship Command (16K/B/G)

The 'real-time' Space Adventure for 'thinking' campaigners!

You command the sole fighting ship of a small league of planets who are pledged to resist the oppression of the powerful Terran Federation.

The 3-dimensional planetary system is divided into 729 sectors (9x9x9), your viewscreen revealing neighbouring sectors 5 wide by 3 high by 3 deep. It can be rotated to look up & down as well as N,S,E & W.

You will encounter friendly, neutral & hostile planets and, of course, enemy interceptors. Your long term objective is to raise the morale of the system's inhabitants so as to bring forth a spontaneous rebellion against the Federation. This can be achieved progressively by winning in combat and converting neutral planets. The opposite occurs if you flee from a fight, upset neutral planets or just skulk!

Machine-code sub-routines ensure the clashes with the enemy are exciting. There are six levels of skill and many other features. Full instructions are given in a separate program. £9.95

## Moon Raider ( MC/G)

The 'Scramble' game you have been waiting for!! Blast the asteroids, enemy missiles & ramships out of the sky as you skim over the mountains on the moon's surface. Bomb the fuel dumps and enemy defences. Higher points scored for hits closer to the ground. Maximise your total score on restricted supplies of fuel. If you survive the first part of the game you enter the 'tunnel', with rocky projections above & below you! Four skill levels, excellent graphics & excrutiating sound via the keyboard port. £8.95

*** NASCOM 1 – Cottis Blandford cassette interface for N2 format, reliability & fast load £14.90
- 8K RAM required unless otherwise stated
- Please state if Nascom TAPE Basic required.

ALL PROGRAMS SUPPLIED ON CASSETTE IN CUTS/KANSAS CITY FORMAT

BARCLAYCARD VISA

Please add 55p/order P & P + VAT @ 15%.
Large (15½p) Sae for FULL CATALOGUE.

PROGRAM POWER
5, Wensley Road
Leeds LS7 2LX. +