80-BUS NEWS

MAY – JUNE 1983 VOL.2 1SSUE 3

- DIY LIGHT PEN
- NASCOM BASIC DIS-ASSEMBLED
- EPSON FX80 REVIEW
- SERIAL INTERFACES

£1.50

The Magazine for NASCOM & GEMINI USERS

May – June 1983. **80-BUS NEWS** Volume 2. Issue 3.

CONTENTS

 Published by Interface Data Ltd and printed by Excelsior Photoprinting Ltd., High Wycombe.

SUBSCRIPTIONS

Annual Rates (6 issues)	UK	£9	Rest of World Surface	£12
	Europe	£12	Rest of World Air Mail	£20

Subscriptions to 'Subscriptions' at the address below.

EDITORIAL

Editor : Paul Greenhalgh Associate Editor : David Hunt

Material for consideration to 'The Editor' at the address below.

ADVERTISING

Rates on application to 'The Advertising Manager' at the address below.

ADDRESS: 80-BUS News,
Interface Data Ltd.,
Woodside Road,
Amersham, Bucks, HP6 5EW.

EDITORIAL

On Schedule

Well, it looks as though we've done it!! Having produced the last issue somewhat late, it looks as though we have got back on schedule with this one. Mind you, I'd better not make too much of it, or some law or other will prevail, and it will be six months to the next issue!

Some of you who have subscriptions will probably be receiving this issue in the same post as the previous one. Sorry if this comes as rather a shock, but as I doubt that we will have got all of the last issue in the post by the time we get this one back from the printers, it seems silly to double up on postage costs. If you don't want to read two issues at once, then put this one away for three or four weeks! Bye, bye for a while.....

Some time later

Hello there. And what has been happening since you received the last issue of this wonderful magazine? It seems like only yesterday! Well, for a start, I am afraid that I have to report that despite further letters to **Lucas/Nascom** and to **IO Research** we have still **NOT** heard from them about the I/O mapping of their 80-BUS/Nasbus boards. Now, the questions asked were not too tasking, and went along the lines of:

a) What boards have you/do you/will you produce that occupy Z80 I/O ports?
b) What are the recommended standard addresses for these boards, what alternatives are there, and how are these options achieved (links/PROM/etc.)?
c) Are NASIO and DBDR provided (necessary for Nascoms)?
d) Can the boards be used with Nascom 1/2/3, Gemini GM811 & GM813?
e) Are there any special restrictions that we should know about?

Answering this lot should take about 10 minutes per board, and nobody makes that many boards that this should produce a serious problem. And why am I making all this fuss? Well, there are now about 25 80-BUS compatible boards available, all fighting for 256 ports. Not a serious problem yet? Well, unfortunately there have already been clashes. Gemini brought out the GM812 IVC some time before the Lucas/Nascom AVC. And yet the IVC uses ports B1, B2 and B3, and guess what? - the AVC uses ports B0, B1 and B2. Yes, the AVC and IVC can both be located elsewhere if necessary, but the standard software support for both of them assumes these port addresses. And why should someone who may well want both boards be hampered by this restriction? Another example of this clashing is between the Climax colour board and another board being produced by a new manufacturer. The Climax uses C0-D0, and the first release of the new board (which I can say no more about - tease, tease!) uses C0-CF. It looks as though they'll have to change it before going into full production. And we all know that IO Research have already shown a prototype of their Pallette board, and have announced two other forthcoming products as well. What ports do these use, and what will they clash with? This is my reason for the third part of question (a) above. I am not expecting all existing and potential manufacturers to send me full details of their yet-to-be-released products, but it would be nice if they sent a note saying "New board, available approx 3 months, uses ports 40-9F"! In order not to prolong the agony, awaiting the reply of obviously disinterested parties, you will find in this issue a VERY brief summary of current port usage. If you know of any boards that I have missed, any mistakes that I have made, and any boards that are under development (no need to say what), then PLEASE drop me a note.

Well, that's all there is space for. Material and ads. for the next issue by July 30th please. And until then, TTFN.

A LIGHT PEN FOR THE NASCOM 2

J. AUCKLAND

D.Parkinson's article on light pens, in VOL.1 Issue 3 of 80-BUS, prompted me to finish writing this piece on the design of a light pen for use with the Nascom 2. The pen has been in use for quite some time and seems to be reasonably accurate, taking in to account the minimum resolution available with the standard Nascom Graphics chip.

The design for the light pen evolved because of my need for a device that would allow me to select routines from menus of routines. Some of these routines were to be used to input data by using the screen. I wanted a routine that would enable me to define the envelope shape of a particular sound that was to be generated in a micro-processor controlled synthesiser. Thus a peripheral was needed that would permit the definition of envelope shapes by drawing the envelope shapes directly on to the screen. It seemed that a light pen would be the most obvious device to use and so I looked to various manufacturers for a suitable light pen. Unfortunately, it transpired that the majority of pens were either out of my price range, or not quite the design that I required, and so I decided to build a pen specifically for the Nascom 2. The design works with my system, and the pen and its associated hardware is far cheaper than any other pen that I know of.

PEN DESIGN

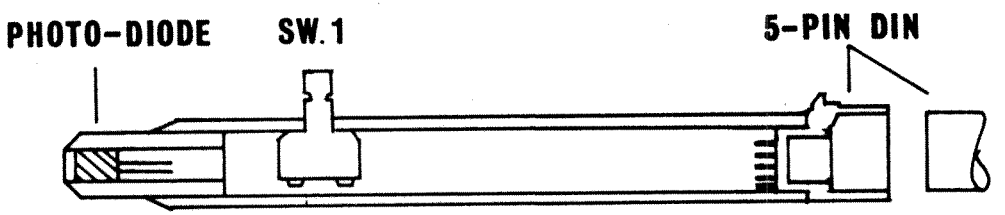
Having looked at the various photo-electric devices that could be employed in the design of the pen I decided to use a Light Activated Switch, which is an RS component type 305-434 (5v), as the photo-sensitive device. This device includes a variable-threshold switch that not only enables the sensitivity of the pen to be altered, but also reduces the component count by having the threshold switch as an integral part of the sensor.

Refering to figure 1 it can be seen that the output from pin 4 of the sensor is usually buffered by a TTL gate to bring the switched output to a clean TTL compatable level. The time-constant components, R1 and C1, set the sensitivity of the device, large values of RC give a high degree of sensitivity. It would be quite possible to use the circuit in figure 1 to directly interface with an input port. On the prototype, a pulse of 20us duration was output from the photo-sensor every time the raster struck the active area of the pen. This could be used to identify an individual pixel, or sub-pixel, on the screen area. By loading each successive VIDEO RAM location with the data FFH and by scanning the input ports for a HIGH from the pen, it is possible to identify the pen location on the video display unit. For a higher degree of accuracy, each VIDEO RAM memory location can be sequentially loaded with the data C1H, C2H, C4H, C8H, D0H, E0H. This data will then enable the pixel to be subdivided into 6 sub-pixels, and will put a small block of white at the top left-hand corner of the pixel, then at the middle left-hand side, and then at the bottom left-hand corner of the pixel. The last three bytes of data put the small block at the top right, mid right, and bottom left-hand corners of the pixel. Therefore, by successively loading each contiguous pixel with the six bytes of data, it becomes possible to identify the pen location on a screen with a resolution of

$$(48 \times 2) \times (16 \times 3) = 4608 \text{ locations}$$

If the output from the pen were connected to bit 7 of port 4, a program for identifying the pen position might look something like figure 3.

LIGHT PEN



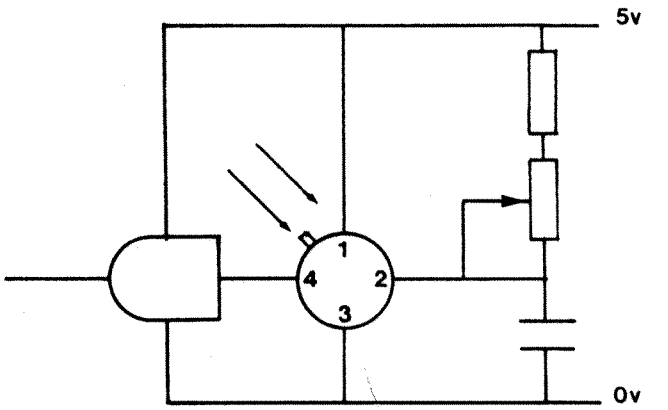
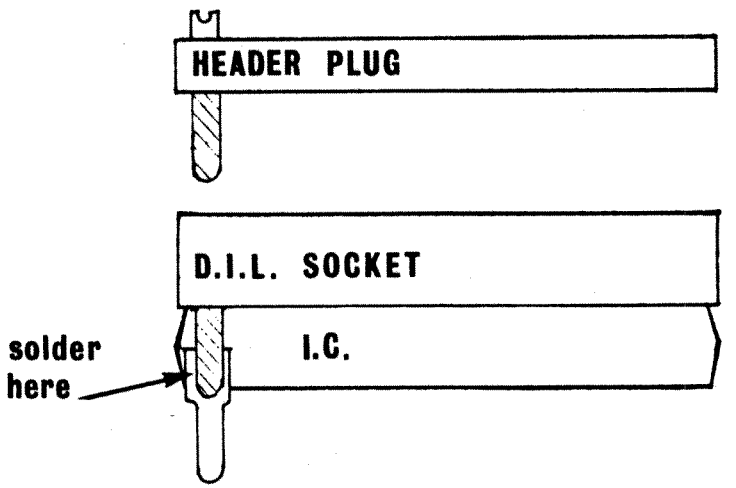


fig.1

LIGHT-ACTIVATED SWITCH



'PIGGY-BACK' PLUG

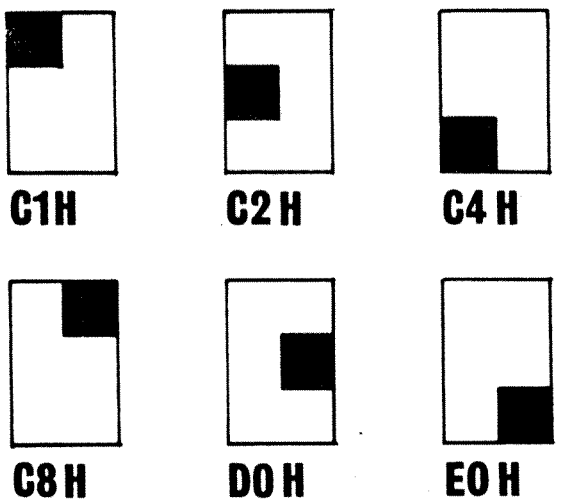


fig.2

NAS-GRAPH IMAGES

This method, crude though it may be, does also need synchronising with the BLANKING signal so that the "pen search" can commence as the screen scan starts. Otherwise, pixels may be tested after the raster scan has passed, and prevent the pixel under the pen from being tested at the same time as the raster is being displayed under the pen. This method of scanning without waiting for the blanking signal, gives rise to a screen full of random dots which, interesting though it may be for the first 10 secs., does not really solve the problem in hand. Also, if the scan program has to wait for the beginning of a new picture scan before it can load the screen with one of the six sub-pixels, it can be seen that each pixel will take at least 6 x 20ms to scan. So to scan all of the 768 locations will take

6 x 768 x 20ms = 92.16 seconds

With no stretch of the imagination can I say that this is satisfactory. A faster method would be to fill the screen with white and to scan the input ports for a HIGH from the pen. When the raster strikes the light-activated switch the other port could latch in the memory address of the VIDEO RAM that was being accessed at the time of the raster strike. As there are 1024 VIDEO RAM locations, it would be necessary to load in 8 bits to one port, and another 2 bits into the other port. This method would give the pen location on any of the 768 screen pixels. If the screen were successively filled with the six sub-pixel bytes, the resolution could be increased to the full 4608 locations.

As the VIDEO RAM is scanned under hardware control it is possible to tap off the required address lines and to feed them directly to the input ports. The program that I was using at the time of the pen's design, required the identification of the pen location as it traversed the screen in a left-right direction. This would enable me to draw a wave-shape directly on to the screen and then store it for later use. Also routines had to be identified from a selection of menus, and so the pen location was required for selection identification.

Refering to figure 4 it can be seen that the VIDEO RAM is addressed by A0 to A9, the address lines A0 to A5 sequentially address 64 screen memory locations, 48 of which are displayed, in a left-right manner. A6 to A9 select the 16 rows of 64 locations in a top-bottom manner, though it must be noted that the screen top row is in fact the last memory row to be addressed, ie. 0BCAH to 0BF9H. Even though the Video ram-scanning hardware is scanning from 0000H to 03FFH on the address lines, IC47, which is the N2MD PROM, selects the VIDEO RAM write-enable when memory locations 0800H to 0BFFH are addressed. Thus there is an offset of 0800H to consider when evaluating the pen location from the VIDEO RAM hardware address lines A0 to A9.

VIDEO ADDRESS LINES

To obtain the address lines, a small amount of soldering is required. It is possible to tap off the address lines directly from the board, though I prefer to keep all soldering and additions to the component side of the board, and it was for this reason that I decided to adopt a "piggy-back" approach to the address line location.

There are a number of ways to find the pen location by using the address lines. the approach that I took was to identify 1 sub-pixel in a column of 48 sub-pixels. As long as I knew which column I was addressing I could find the pen location on that column. It would be possible to find the pen position on the left-right axis as well as the up-down axis, as long as I looked at all the

ZEAP Z80 Assembler - Source Listing

fig. 3

```
                0040 ;*****************************************
                0050 ;THIS PROGRAM FINDS THE PEN LOCATION BY
                0060 ;SEQUENTIALLY LOADING THE SCREEN WITH
                0070 ;IMAGES AND THEN LOOKING AT THE PEN STATUS
                0080 ;TO SEE IF THE RASTER IS ILLUMINATED UNDER
                0090 ;THE PEN.VBLANKING IS TAKEN TO BIT 6 OF
                0100 ;THE INPUT PORT,THE PEN OUTPUT TO BIT 7.
                0110 ;*****************************************



                0140 ;EQUATES FOR THE PROGRAM
1AB7 080A       0150 ORIGIN EQU  080AH       ;SCREEN TOP LEFT
1AB7 0010       0160 MARGIN EQU  16          ;DON'T TEST MARGINS
1AB7 00C1       0170 IMAGE1 EQU  0C1H        ;PIXEL TOP LEFT
1AB7 00C2       0180 IMAGE2 EQU  0C2H        ;PIXEL MID LEFT
1AB7 00C4       0190 IMAGE3 EQU  0C4H        ;PIXEL BOTTOM LEFT
1AB7 00C8       0200 IMAGE4 EQU  0C8H        ;PIXEL TOP RIGHT
1AB7 00D0       0210 IMAGE5 EQU  0D0H        ;PIXEL MID RIGHT
1AB7 00E0       0220 IMAGE6 EQU  0E0H        ;PIXEL BOTTOM RIGHT
1AB7 0020       0230 SPACE1 EQU  20H         ;SPACE
1AB7 0010       0240 LINES  EQU  16          ;NO.OF LINES TO TEST
1AB7 0030       0250 COLUMS EQU  48          ;NO.OF COLS TO TEST


0C80            0280         ORG  0C80H          ;WORKSPACE RAM
0C80            0290         ENT

0C80 3E0C       0310         LD   A,0CH          ;CLEAR THE SCREEN
0C82 F7         0320         RST  30H

0C83 3E4F       0340         LD   A,4FH          ;SET PORT 4 TO BE
0C85 D306       0350         OUT  (6),A          ;IN INPUT MODE

0C87 210A08     0370 START   LD   HL,ORIGIN   ;POINT TO TOP LEFT
0C8A 0610       0380         LD   B,LINES        ;16 CHARACTER LINES
0C8C C5         0390 NEWLNE  PUSH BC             ;SAVE THIS

0C8D 0630       0410         LD   B,COLUMS       ;48 COLUMS

0C8F 36C1       0430 SCANST  LD   (HL),IMAGE1 ;LOAD FIRST IMAGE
0C91 CDBC0C     0440         CALL PENTST         ;IS THERE A HIT?

0C94 36C2       0460         LD   (HL),IMAGE2
0C96 CDBC0C     0470         CALL PENTST

0C99 36C4       0490         LD   (HL),IMAGE3
0C9B CDBC0C     0500         CALL PENTST

0C9E 36C8       0520         LD   (HL),IMAGE4
0CA0 CDBC0C     0530         CALL PENTST

0CA3 36D0       0550         LD   (HL),IMAGE5
0CA5 CDBC0C     0560         CALL PENTST

0CA8 36E0       0580         LD   (HL),IMAGE6
0CAA CDBC0C     0590         CALL PENTST

0CAD 3620       0610         LD   (HL),SPACE1    ;ERASE IMAGE5
0CAF 23         0620         INC  HL             ;NO HIT,THEN NEXT PIXEL
0CB0 10DD       0630         DJNZ SCANST         ;DO FOR 48 COLUMS

0CB2 C1         0650         POP  BC             ;RETRIEVE LINES TO DO
0CB3 111000     0660         LD   DE,MARGIN      ;MISS OUT THE MARGIN
0CB6 19         0670         ADD  HL,DE
0CB7 10D3       0680         DJNZ NEWLNE

0CB9 C3870C     0700         JP   START          ;TRY AGAIN

                0730 ;*****************************************



                0740 ;ROUTINE TO TEST FOR A BLANKING PERIOD
                0750 ;AND A RASTER SCAN HIT.
                0760 ;*****************************************

0CBC DB04       0780 PENTST  IN   A,(4)      ;TEST THE PORT!
0CBE CB77       0790         BIT  6,A            ;IS THE BLANKING LOW?
0CC0 20FA       0800         JR   NZ,PENTST      ;NO,THEN LOOK AGAIN

0CC2 DB04       0820 BLANK   IN   A,(4)      ;TEST UNTIL THE SCREEN
0CC4 CB77       0830         BIT  6,A            ;SCAN COMMENCES
0CC6 28FA       0840         JR   Z,BLANK

0CC8 DB04       0860 PENHIT  IN   A,(4)      ;NOW LOOK FOR A HIT
0CCA CB77       0870         BIT  6,A
0CCC 280B       0880         JR   Z,RTN          ;NEW BLANK PERIOD?

0CCE CB7F       0900         BIT  7,A            ;HIT? TEST UNTIL A NEW
0CD0 28F6       0910         JR   Z,PENHIT       ;BLANKING PERIOD

0CD2 D1         0930         POP  DE             ;REMOVE RETURN ADDRESS
0CD3 DF         0940         RST  18H            ;PRINT LOCATION ON
0CD4 66         0950         DEFB 66H            ;SCREEN

0CD5 DF         0970 AGAIN   RST  18H        ;PRESS ANY KEY TO
0CD6 62         0980         DEFB 62H            ;START AGAIN
0CD7 30FC       0990         JR   NC,AGAIN

0CD9 C9         1010 RTN     RET
```

address lines, but this would mean looking at more than 1 input port, and I was already using that for something else.

Address lines A6 to A9 can be used to identify which of the 16 character lines is being addressed. This brings the pen location resolution to 1 in 16. To increase the resolution we can look at either all 14 raster lines, or, as the minimum sub-pixel is only 4 raster lines, we can look at the address lines that relate to the selection of each group of 4 raster lines. The RS lines, RS0 to RS3, are used to select the appropriate raster lines for each pixel, and so by looking at RS2 and RS3 it becomes possible to identify on which of the sub-pixels the pen is resting.

A6 to A9 can, therefore, be treated as the Most Significant Bits, and RS2 and RS3 as the Least Significant Bits, of a six-bit address that can identify a location in a column of 48 locations.

To tap off these address lines, I soldered 16-pin header-plugs directly on to IC68 and IC53. This method allowed me to solder wires to the plug with ease, and gave test-points to see if I had blown the chip by soldering too close to it! The six address lines were taken along the board and put on to the bus at pins 59 to 64. On the Gemini 80-BUS, these pins are now allocated as interrupt request lines, powerfail warning, and backup power, so it might be advisable not to put the video address lines on to the bus if you can help it. The Video Blanking signal, VBLANK, was also taken to the bus so that the screen data would be updated only in the screen blanking period. this prevents screen flicker when drawing images on to the screen.

OTHER HARDWARE

For certain routines, I wanted to know if the pen was in the same place, or whether it had moved from an area of white to an area of black. Unfortunately, you can not merely test the pen status. This is because the sub-pixel consists of 4 raster lines, each being 64us in duration, and each raster-strike on the active area of the pen produces a pulse of 20us duration. As the picture frame is re-displayed every 20ms, the sub-pixel will produce a train of four 20us pulses every 20ms. It is only through the persistance of vision that the image appears to be constantly displayed.

It now becomes necessary to build a circuit that will output two pulses. One pulse will be of 20us duration, to indicate that the pen has been struck by a raster, the other pulse will go high when the pen has been struck, and remain high for period of time that is greater than 20ms and will also be kept high every time that the pen is struck. By doing this, a high state is output as long as the pen remains over the displayed area, even though the area is being refreshed every 20ms and being displayed for only 80us.

Figure 5 is the circuit diagram for the pen interface. IC1 is an NE555 timer configured as an astable multivibrator, the frequency of which is adjusted by RV1. IC2 is a 7493 Binary counter and IC3 and 4 are 7400 NAND gates and 74LS126 TRI-STATE buffers.

On power-up, the 7493 Q outputs are in a low state and so IC3(a), in conjunction with the inverted output of IC2 and the output from the clock generator, begins to clock the counter through 16 counts until Q3 goes high. At this point, one of the inputs to IC3(a) becomes low and so the clock pulses are inhibited and the 7493 ceases to count. The HIGH at Q3 is inverted by IC3(b) and buffered by the TRI-STATE buffer IC4(a), the output of which will be enabled only when SW1 is closed.

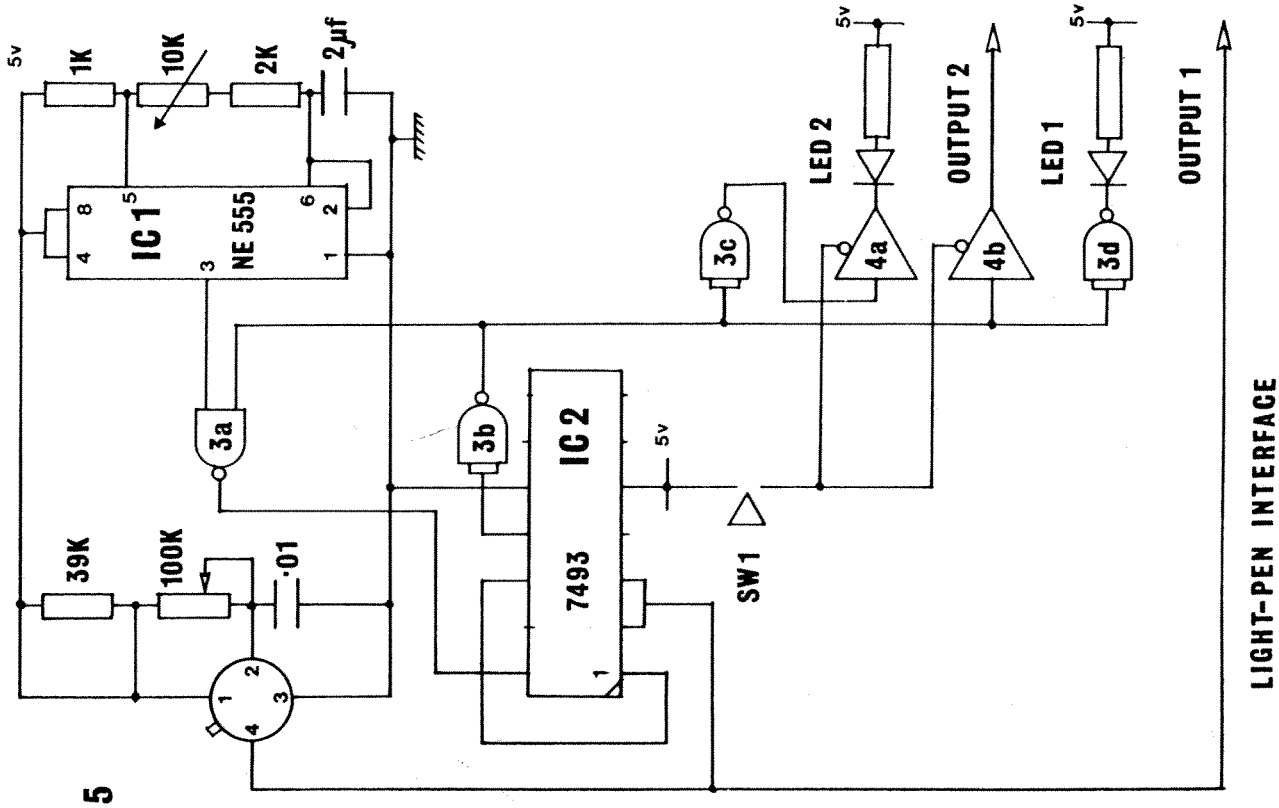


fig. 5

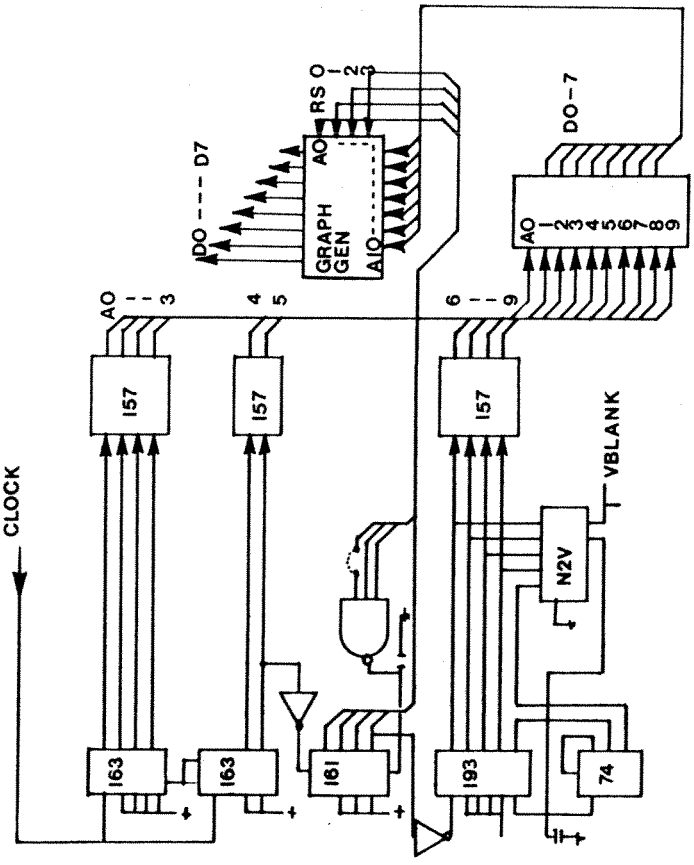


fig. 4

NASCOM VIDEO-SCAN HARDWARE

It can be seen that the output to the ports is now held at a LOW state. If the pen is now activated a 20us pulse will be sent to both the OUTPUT1 and the RESET input of the counter. Q3 now goes LOW, and thus IC3(b) goes HIGH, enabling the clock input and allowing the count to begin again. The output2 has now gone HIGH indicating a hit. As long as the frequency from the clock generator is lower than 20ms x 16 ie about 800Hz, then IC2 can not count up to 16 before the receipt of another reset pulse. Therefore, if the pen is struck by the raster every 20ms, IC2 will be reset and Q3 will remain LOW and the output2 HIGH.

Even though the pen has moved from a white area to a black area, it will not register the change until 16 clock pulses have been received, and RV1 can be used to slow down the clock and increase the time taken to register a change in state. This can be used to effectively slow down the speed of the pen. Similarly, the frequency can be increased to a point where the reset pulse has no effect. The two LEDs indicate the pen's status. LED1 indicates whether the pen has registered a hit, and LED2 shows if the data has been enabled into the ports.

The prototype was built on a small piece of veroboard and housed in a small diecast aluminium box, and seems to have tolerated an immense amount of knocking. The pen was constructed using a small length of plastic tube with a 5-pin plug and socket to allow it to be disconnected from the system. I found that it was easier to assemble the pen in three pieces than to try and poke everything down the tube. If the sensor is glued into a smaller tube, (I used a drilled, solid piece of plastic for this) and then that inserted into a larger tube, the pen can be easily separated for modificaction. Similarly, the switch can be set at the junction of two tube halves to facilitate re-wiring if necessary.

It should be possible for most people to construct a light pen using the above design,though I am quite sure that most readers will be aware that this is only one way of obtaining information as to the whereabouts of the raster.

Another method could be to count the blanking pulses, or indeed build a circuit on the lines of the NASCOM VIDEO RAM scanning circuitry, or buy a GEMINI video board and have done with the interface problems!

Now that ARFON have gone into receivership, it may be that this do-it-yourself light pen will be the cheapest on the market, and I see no reason why it can not be used for the GEMINI video board.

CLASSIFIED ADS.

PLUTO Colour Graphics Card with Extended Command ROM.
Suitable for any 80-BUS System.
Retail price including VAT over £500.
Will accept £320. Ring Bob on 01-542-0873.

WORDSTAR FOR THE GEMINI IVC

by Richard Beal

Wordstar is almost certainly the most widely used microcomputer word processing package in the world. Produced by Micropro, it is available ready configured for many computers. However, almost any microcomputer can run the package, because it comes with an excellent install program and extensive instructions on how to patch the program to specify any features not readily covered by the install program. This article describes how to patch Wordstar for the Gemini IVC and keyboard, making best use of the Gemini features, for example the cursor movement keys on the right side of the keyboard.

[Ed.'s note - Gemini have recently obtained a number of software licenses, including those for Micropro's Wordstar, Spellstar and Mailmerge. These packages are therefore now available from Gemini dealers in Gemini disk format, already installed for the IVC.]

Wordstar can be obtained in its standard form on an 8 inch diskette, and is also available on many 5 inch formats such as Superbrain. Having migrated the software from the alien disk format to your Gemini format, which may be done using a program such as COPYSB, or with the help of a friendly dealer, you are ready to install it. You must run the install program, even if you are going to enter all the hand patches for every option. This article does not cover the printer patches, which are often unnecessary as you can select the CP/M list device and a standard TTY printer.

When you run the install program, you will find that the Gemini IVC is not a menu option, so you must select the user installed patches option for each menu option which you cannot answer. Having written the installed Wordstar back to disk, you are ready to load it under the debug utility and patch it. DDT, ZSID, or GEMDEBUG are all suitable. If you use DDT or ZSID, remember to work out the number of pages for the SAVE command before you start the debug program.

I will simply give you the values which I have found useful, as a complete description of all the options can be obtained best by working it out yourself from the manual. These patches are for Wordstar Version 3.0, and will not work on earlier versions.

Address	New values	Meaning
		The following group of changes alter various initial values.
0360	02	Change initial help level to 2.
0361	00	Display helpful message at start of first edit.
0392	FF	Enter Wordstar in non-document mode instead of document mode. Useful if you have a version just for program text editing.
03E7	Name	Name of Wordstar, change if not WS.
		The next change alters the keyboard to be consistent with CP/M etc.
049B	C3 67	Make Backspace delete a character, instead of just moving the cursor.

The following group of changes allow use of the cursor keys on the Gemini keyboard.

052D	7F	Deactivate Ctrl-Del (1F).
0649	1C 00	Cursor left
064B	65 63	
064D	1D 00	Cursor right
064F	5B 63	
0651	1F 00	Cursor down
0653	24 64	
0655	1E 00	Cursor up
0657	3E 64	

The following group of changes alter the initial messages displayed on the screen.

0181	Date	Inspect and change as you wish.
0190	Terminal	Suggest "Gemini IVC".
01B4	Printer	

The following changes are for the IVC display.

0248	19 50	Rows and columns on screen.
024A	02 1B 3D	Position cursor.
0253-025D		Must all be zero.
025E	20 20 00 00 00 00 00 00 C9	
026D	02 1B 2A	Erase to end of line.
0274	01 0B	Delete line.
027B	01 0E	Insert line.
0284	02 1B 41	Inverse video on.
028B	02 1B 4E	Inverse video off.
0292-02A3		Must all be zero.
02A4	00 00 C9 00 00 C9 00 00 00 00	
02AE	01 01	Delays - not needed and 01 is the minimum value.
02B0	00 00 00	
02D2	10	Reduce "long delay" to sensible length of time.

The next value applies ONLY to users of SYS with virtual disk and the VFLIP option selected.

02DC	10 (normally 01)	Default disk drive to test for the Wordstar overlay file.

When you have installed these changes, remember to save the new version of Wordstar on disk. Then run it and it should work perfectly.

AN INTRODUCTION TO BASIC ON DISK – THE MICROSOFT MBASIC INTERPRETER

P. D. Coker

[Editors' note:
When Dr. Coker's article was first submitted I pointed out to him that this represented an article on 'How to use a ripped off MBASIC', a practice 80-BUS News could not condone. Dr. Coker agreed that in that light, I could edit it in any way I wanted. However, on re-reading the article, I gleaned so much from it that I consider it a very good commentary on the MBASIC manual (which I thought I had read thoroughly). In the end, I find I have done very little of the scissors job on the article that I threatened. I am unable to tell which version of MBASIC Dr. Coker is using, my guess is version 4.x, I have noted several descrepancies between this and mine which is version 5.21, dated mid 1981. D.R.H.]

Microsoft BASIC is available in a number of versions which differ mainly in the number of features which they have available and the medium in which they are supplied. Perhaps the commonest form is the 8k ROM BASIC which is supplied as standard on the Nascom 2 and several other machines, but increasing numbers of 80-BUS users have the disk-based version which offers many additional facilities. The purpose of this note is to provide a simple guide to some of the more useful features which may be of interest. In this article, MBASIC refers to the disk version of Microsoft BASIC; BASIC refers to other versions.

MBASIC has been available for several years and is regularly updated. Users who have experience with BASIC on mainframe computers will find that most of the statements, commands and functions with which they are familiar are identical in use, or at worst only marginally different. There is also a wide range of facilities not usually found in mainframe BASIC implementations. The 'SAVE' command is a little more awkward to use and there are no matrix-handling (MAT) statements. Alcock (1977) in his book 'Illustrating BASIC' shows how these can be programmed in the absence of MAT instructions.

Having loaded MBASIC and achieved the 'sign-on' message, MBASIC will display the amount of free memory you have at your disposal: MBASIC occupies about 28kbytes and a 48k or 64k system is recommended. Following the 'sign-on', the prompt, 'Ok' is displayed.

In common with most computer software documentation, the MBASIC manual is not particularly 'user friendly' on first reading. Quite full explanations of the commands, functions and statements are given, but it is not designed for the beginner who can easily become confused by the intricacies of, for example, file handling using random-access files. Some users will have lost, or never had a manual (whoops!).

There are, in general, no hard and fast distinctions between statements and commands in MBASIC and in many cases, commands can be executed as part of a statement and vice versa - experience will show when you can't!

The following is a resume of some of the commands and statements first time users or users converting from Nascom BASIC (or other BASICs) will not necessarily be familiar with. The more usual commands and statements such as LET, FOR -- NEXT, etc, I take as read and well understood. In the lists of commands and statements which follow, square brackets enclose optional parts and < and > signs enclose parts which the user has to supply.

AUTO:

This is a remarkably useful command for automatically generating line numbers; to use it, type:
AUTO [<initial line>[,[<increment]]:
e.g. AUTO 100,10 will generate line numbers from 100 in increments of 10.

AUTO will provide a line number after every <cr> until Control-C (^C) is typed. If the initial line is omitted from the command, it is assumed to be 10 and this is the default value for the increment. If AUTO generates a line number that is already in use, the line number is followed by an asterisk to warn you that any input will replace the existing line. This can happen after an editing session - with unfortunate consequences!

RENUM

This is an extremely handy command and allows program lines to be spread or packed, particularly after editing; new lines can be inserted between existing lines. To use RENUM, type:

RENUM [<A>[<B>[,<I>]]]

where A is the new line number of the first line to be renumbered, B, the line number, numbers below which will NOT be renumbered. I is the increment. Default values for A and I are 10, and if B is omitted, the whole program will be renumbered in sequence. For example:

RENUM - renumbers the whole program in increments of 10, starting with line 10.

RENUM 150,,20 - renumbers the whole program which will then start at 150 and line numbers will be incremented by 20.

RENUM 1000,100,50 - renumbers lines from 100 so that they start at 1000, incrementing by 50.

Note that RENUM alters correctly all line numbers appearing after a GOTO, GOSUB, IF....THEN, ON....GOTO or ON....GOSUB. RENUM cannot be used to change the order of program lines nor can it (in common with AUTO) create line numbers greater than 65529; an error message will result!

REM:

Apart from the normal use of 'REM' after the line number for program comments, it is possible to use a single quote (') which may be placed anywhere on the line, after the line number. Everything on the same line which follows the " ' " will be ignored; this can be particularly useful for detailed comments - for example:

```
100 FOR I=1 TO 255 'GO THROUGH THE ASCII CHARACTER SET
110 PRINT I,CHR$(I) 'PRINT THE APPROPRIATE CHARACTER ON THE SCREEN
120 FOR T=1 TO 500 'WAIT A BIT SO THAT YOU CAN SEE IT
130 NEXT T
140 NEXT I 'GO ROUND AGAIN! (AND AGAIN!)
```

EDIT:

The EDIT command in MBASIC is quite sophisticated and only a few of its features will be covered here. Editing can be carried out both while the program is being typed in and on the finished program. Single characters can be changed either by typing <backspace> or <delete>. In the latter case, the deleted letter is enclosed by backslashes (\) and a carriage return occurs if there are no preceding characters. In this version of MBASIC, the typing of <^A> instead of the <cr> at the end of the line puts you into EDIT mode and allows the use of almost all the features of the EDIT command. If, having typed a <cr> you wish to edit the line just typed, use 'EDIT .' - the full-stop indicates to the EDIT command that the current line is to be edited.

EDIT Commands:

I Inserts new characters into the line which is being edited (to a maximum of 255 characters including the original text). Insertion is stopped by typing <esc> or <altmode>, depending on your keyboard.

<del> The delete key typed during an Insert command will delete the character to the left of the cursor.

<cr> If a carriage return is typed during an Insert command, the effect is to terminate the Insertion and print the rest of the line; the edited version will then replace the original.

X The action of the X command is similar to the I command, but it causes the printing of all characters to the right of the cursor, then it enters the 'I' mode. This is a useful facility when new statements are to be added to the end of an existing line (but don't forget the colon which is used as a statement separator).

H The H command has a similar action to the X command but it deletes, rather than prints all characters to the right of the cursor, before entering the I mode.

D Characters may be deleted by the use of this command. nD deletes n characters to the right of the cursor where n is an integer in the range 0 to 255 (0 defaults to 1 and if n is omitted, a default of 1 character is assumed). The deleted character(s) are enclosed in backslashes and the cursor will be positioned just to the right of the last character which has been deleted.

S Searching for a character is carried out by this command which has the general form nSx where n is the nth occurrence of the character 'x'. The default value for n is 1 and the search begins with the second character to the right of the cursor. All characters passed over by the cursor in its search are printed and if the sought-for character is not found, the cursor will stop at the end of the line; if the search was successful, the cursor will print the character and stop to its right.

K This has a similar function to the S command but instead of printing all characters as it passes over them, the command deletes them. The command has the general form nKx where n is the nth occurrence of character 'x'; a default of 1 is assumed.

C The C command is used for character replacement in a line. It has the general form Cx and changes the character on the right of the cursor to the character 'x'.

E The E command terminates editing and outputs a carriage return; unlike the <cr> command, the remainder of the line is not printed.

Q Typing of Q restores the original line and control is passed to command level.

L If the L command is used, the remainder of the line is printed , a <cr> is output and then the line number is printed; editing is restarted at the beginning of the line.

NOTE - If, during the running of a program, a SYNTAX ERROR is found, MBASIC will automatically enter the EDIT mode, printing the message 'Syntax error in nn', followed by the line number of the offending line. Editing can then start.

The best way to find out how the Editor works is to practice before you write your Pan-galactic Black Hole Simulation program. [Ed. - This is all superfluous if you use the Gemini on screen edit mode or Richards' SYS, although under these circumstances the line length must not exceed the width of the screen.]

SYSTEM:

The SYSTEM command exits from MBASIC and returns to the disk operating system.

LOAD:

The LOAD command has the format 'LOAD "<filename>"' and locates and loads the file into memory ready for use. On completion of a successful load, the 'Ok' prompt is given. The first double quote (") is mandatory but the second can be omitted. [Ed. - For drive selection see the note under SAVE.]

SAVE:

The SAVE command is used, as its name suggests, to save a program from memory to disk. It has the following format:

SAVE "<filename>",[<disk drive no.>]

The effect of the SAVE command is to leave the current version of the program in the computer's RAM and to copy an exact replica to disk where it is stored under the <filename>. A frequent cause of program losses is when one inadvertently uses a previously used filename (of a different program) as the <filename>. In this case, the previous program is deleted from disk and replaced with a copy of the current program under the old name - which can be frustrating! The moral is to remember your filenames or to backup your program disks. SAVEing an ASCII file is done by specifying the disk no. (usually 0 or 1), followed by an A - for example, SAVE "TEXT.BAS",0,A saves TEXT.BAS as an ASCII file on drive 0. ASCII files are larger than the normally saved files and are both slower to load and more extravagant of disk space; their chief advantage is that they can be used as direct input to other programs if required.

[Ed. - My version of MBASIC works differently, in the form of:
SAVE "<[disk drive name:]filename>",[protect or ASCII option]
So SAVE "B:FRED",A means save file FRED to drive B: in ASCII format. This format also holds true for LOAD, KILL, OPEN, NAME - AS, MERGE and CHAIN.]

KILL:

This command erases a specified file from disk; if the file does not exist, an error message is output as is also the case if an attempt is made to erase a file which is in use. The format of the KILL command is:
KILL "<filename>".

NAME:

Renaming of files is achieved by the use of the NAME command which has the following form:
NAME "<old filename>" AS "<new filename>"
If the proposed new file name already exists, an appropriate error message is printed.

MERGE:

The MERGE command is used when it is desired to put part or all of two programs together to form a new program. It should be used with care since the results of merging two programs can be unpredictable (or just plain messy!). It has the following form:
MERGE "<filename>"
The effect of this command is to load <filename> into the program already in RAM. Where the loaded program and the original program have identical line numbers, the loaded line numbers replace the existing line numbers. The file name to be loaded must specify an ASCII format saved program, otherwise a 'BAD FILE MODE ERROR' will occur.

FILES:

This command is used to print out the names of ALL files, regardless of type, on the current disk. It is equivalent to the 'CAT' or 'CATALOG' commands in mainframe BASICS, and to the 'DIR' command when at disk operating system level.

Naming of files:

In MBASIC, all program and data files have a filename. This is a 'string' expression between 1 and 8 characters in length; the first character must not be a null (0) or a byte of 255 (dec.). The format of a filename permits the addition of an extension, which can be used to convey further information about the file. The usual way of naming files is to give the filename, followed by a full stop and a 3 character extension. Thus FUNNYONE is a valid filename and so are FUNNYONE.BAS, FUNNYONE.DAT or FUNNYONE.123 but OFUNNY is invalid because it has a '0' as its first character. As a matter of sensible programming technique, it is wise to name BASIC program files with <filename.ext>, e.g. COUNTER.BAS for the program or COUNTER.DAT if it is a data file. Note that, then loading a BASIC program file with 'LOAD', inclusion of the .BAS extension is optional, although it might be wise to include it if you have, for example, a data file as well as a program file under the same filename. The same holds true when SAVEing a program. File names can consist totally of numbers, or have some non-alphanumeric characters - 13579081.BAS is allowed, as is FILE$$1 or PROG£2. Filenames in lower case are the same as their equivalents in upper case - SHEAR and shear , for example.
[Ed. - Careful!!! If given a lower case file name, MBASIC will save and load it as such. However, if say an ASCII file were saved with a lower case file name for the purpose of loading into an editor for instance, then CP/M would not be able to find it as all CP/M command line input is converted to upper case automatically.]

File Handling:

MBASIC has excellent disk file handling capabilities; the statements 'OPEN' and 'CLOSE' do just that. Files may be sequential or random as far as input and output are concerned. The 'OPEN' statement has the following format:
OPEN "<mode>",<file number>,"<filename>",[<drive no.>]
<mode> is a single letter string (O,I, or R), which indicates the type of input or output to or from the file. O implies that the file is opened for results to be written INTO it (i.e., an output file); I shows that the file is opened so that data can be read FROM it (i.e., an input file). In both cases, the data movement is sequential (in order). The R mode is used for random input/output in groups of up to 128 character records, and any record of a file can be read or written to at any time; unlike the sequential file, a random file can be written to or read from at any time.
<file number> is the number of the file as used in the program and is an integer from 1 to 15; it refers to the file in later I/O operations.
<filename> is the name of the file, and must be enclosed in double quotes.
<drive no.> is the number, usually either 0 or 1, of the disk on which the file resides, or will be written to. [See Editors' note under SAVE.]
For example:
OPEN "I",1,"INPUT.DAT" - will open file INPUT.DAT so that data from it can be read by the program.
OPEN "O",2,"RESULTS" - opens file RESULTS so that data from the program can be written into it.
OPEN M$,N,F$ - opens the file whose name is in F$ (supplied earlier in the program, together with M$ and N) as file number N in mode M$.
[Ed. - Mine is different again, in the form:
OPEN <mode>,<file number>,"<[drive name:]filename>",[record length]
All is as above, except that the random access file mode may vary the length of the records saved between 1 and 128 thus saving disk space.]

Input and Output to sequential files:

This involves the use of the INPUT and PRINT statements: INPUT is used to read data from a file as follows:
INPUT £<file number>,<variable list>
where <file number> is the number of the file OPENed for INPUT and <variable list> is the list of variables which are to be read in as would be the case with a normal INPUT statement, although no '?' prompt is printed. When numeric values are input, leading spaces are ignored but everything else is assumed to be part of the number, which terminates when a space, <cr+lf> or comma is read. When string items are input, the first character which is not a leading space or line feed is assumed to be the start of a string item; if this first character is found to be a ' " ', the item is taken to be a quoted string and all characters between the first quote and a matching second quote are returned as characters in the string value. The string may not be more than 255 characters long and will be terminated before this if a comma, <cr> or <lf> are encountered.

PRINT £<file number>,<variable list> is used to save results to a file.

CLOSE <file number> terminates input or output to that file; CLOSE without a file number, terminates I/O to all files previously opened.

The following example of sequential file I/O for numeric items may be useful:

```
1000 OPEN "O",1,"RESULTS"
1010 PRINT £1,A,S,D        'Write the contents of "RESULTS" to the file
1020 CLOSE 1               'Close it for writing, set pointer to start of file
1030 OPEN "I",1,"RESULTS"  'Open "RESULTS" for reading in data
1040 INPUT £1,A,S,D        'Read in the data
```

Files are also closed when the 'END' statement or 'NEW' command are executed; the 'STOP' command does not close files.

LINE INPUT may be used with disk files and has the same format as INPUT; it is used for reading in an entire file line without using quotes, commas etc. and is particularly useful if a SAVEd (ASCII) BASIC program is to be read as data by another program.

EOF (end of file) detection. This function is used to detect when there is no more data in the input file; it takes the form: X=EOF(<file number>). When there is no more data, EOF is TRUE (-1), otherwise it is FALSE (0); an INPUT PAST END error will occur if an attempt is made to read past the end of a data file. For example:

```
100 OPEN "I",1,"RESULTS"
110 I=0
120 IF EOF(1) THEN 160
130 INPUT £1,X(I)
140 I=I+1
150 GOTO 120
160 .... next part of program
```

Random file I/O:

So far, all the examples of file handling which have been discussed here have concerned sequential files, but there are a number of applications where the time taken to access a sequential file will be excessively long since in order to pick out a data item near the end, all the previous data records have to be read! This is particularly awkward where, for example, a selective search of a mailing list or inventory is required. The rate of data transfer in random I/O files is also faster and the process altogether more efficient than is the case with sequential files, but the technique is rather more complex, and you are advised to try the sequential method to gain experience.

OPENing random files is done in exactly the same way as sequential files, except that the mode is set to "R"; CLOSEing is done in an identical fashion.

Reading and writing data to random I/O files:

This is accomplished by using the PUT and GET commands. Each random file has a buffer up to 128 byte long associated with it and data is moved into or out of this buffer from or to the data file. PUT takes data and writes it into the buffer and GET reads data from the data file into the buffer; the format for these commands is:

PUT <file number>,[<record number>]
GET <file number>,[<record number>]

If the <record number> is omitted from either statement, then that record number is incremented by 1 from the previous GET or PUT, and the appropriate record is read or written into the random buffer. The highest record number which is allowed is 2046 and an initial GET or PUT without a record number will read or write the first record. If an attempt is made to GET a record which has not been PUT, that record is returned as a series of zeroes and no error message is displayed. Further information on random I/O files should be obtained from an MBASIC manual.

[Ed. - Mine is capable of 32767 records.]

DELETE:

Three possible versions of this command exist:

DELETE <line number> deletes that line only;

DELETE-<line number> deletes every line of the current program up to and including the current line;

DELETE <line number>-<line number> deletes inclusive line numbers from the first to the last.

LLIST:
LPRINT:

These have the same functions as LIST and PRINT in 8k ROM BASIC, but output is directed to the printer. If a program halts for no obvious reason and LPRINT has been used, check that the printer is switched on and 'on-line'!

DEFUSR:

This has the function of defining the starting addresses of assembly language subroutines; up to 10 such subroutines can be used. The format is: DEFUSR <digit>=<X> where X is the starting address.

ERASE:

The arrays or vectors specified are deleted and may be redimensioned if required. The format is: ERASE <array name 1>, ... <array name n>

PRINT USING:
LPRINT USING:
The format of these statements is:
(L)PRINT USING <string>;<list>
These print (on screen or printer) the values of the expressions in the <list> in a manner dictated by the <string>; the <list> contains constants, variables or expressions to be printed, separated by punctuation as in the PRINT statement.

ON ERROR GOTO:
The format for this statement is:
ON ERROR GOTO <line number>
When an error occurs, it diverts to the specified line; sets the variable ERR to the error code and ERL to the offending line number. ON ERROR GOTO 0 is used to disable error trapping.

RESUME:

This is used to restart program execution after error trapping. Normally, a line number is specified but if it is omitted or set to zero, then the program resumes at the statement where the error occurred. RESUME NEXT causes program execution to resume at the statement after the one where the error occurred.

SWAP:

A facility which allows the values of two variables to be exchanged. They must be of the same type - e.g. strings, integers or reals. The format is: SWAP <variable 1>,<variable 2>.

TRON:

Enables the trace flag; this prints the line numbers of statements as they are executed and is useful for debugging. It can be used both as a command and as a statement.

TROFF:
Disables TRON; TRON is also automatically disabled by the 'NEW' command.

Operators:

Operators are used to add, subtract, multiply etc. in statements; there is an established order of precedence as follows: parentheses - 'brackets' have the highest precedence and the rest follow in decreasing order; exponentiation (^), negation (-), multiplication (*) and division (/), integer division (\), modulo (MOD), addition (+) and subtraction (-) (subtraction is not the same as negation!), then the relational operators = , <> , < , > , <= , =< , >= , =>. The logical operators which follow are also in decreasing order of precedence: NOT, AND, OR, XOR, EQV and IMP. (XOR is exclusive OR, EQV is Equivalence and IMP is Implication.) All the logical operators are 'bitwise'.

Numbers and other numerical expressions:

MBASIC permits several ways of processing or printing numbers. 8K ROM BASIC recognises integers and real numbers (integers are numbers without a decimal part - e.g. 10, 3, 1001 or -54); real numbers are those with numbers following the decimal point and are of single precision accuracy (the first seven digits of the number are regarded as significant and the seventh is rounded up). Scientific notation is also allowed - for example, 1,000,000 is 1E6). In addition to these, MBASIC permits the use of double precision (14 significant figures) arithmetic. Numerical constants with more than 7 digits, or a D in place of the E in the exponent are double precision, as are constants followed by a £ sign, as in 1.2534789£. If a constant is followed by an exclamation mark, it is considered as a single precision constant, even if it has more than 7 digits; for example 4.5555671! is interpreted as 4.55556.

MBASIC also allows hexadecimal and octal constants and these must be prefixed by &H or &O respectively; the constant must not contain digits other than 0 - 7 for octal constants, and 0 - 9, with the letters A - F for hexadecimal.

Variables may be declared with single or double precision, or as integers or strings. They may be explicitly declared by 'type' as follows:

Type	Symbol
String 0 to 255 characters	$
Integer from -32768 to 32767	%
Single precision (up to 7 digits, exponent between -38 and +38)	!
Double precision (up to 16 digits, exponent between -38 and +38)	£

Types may be changed by explicit definition, another version of DEF; if no type is specified, MBASIC assumes that all variables are single precision. The format of the DEF statement is as follows:

DEFINT a .....defines a as an integer variable
DEFSTR a .....defines a as a string variable
DEFSNG a .....defines a as a single precision real variable
DEFDBL a .....defines a as a double precision real variable

One or more letters may be defined in this way - thus DEFINT I - N defines variable names beginning with the letters I to N as integers and DEFDBL D indicates that variables beginning with the letter D are of double precision type.

Variables in MBASIC can be of any length (within reason) but only the first two alphanumeric characters are used and the first character must be a letter. [Ed. - Different again. Variables may be up to 42 letters or numbers, the first must be a letter and the first letters must not form a reserved word.]

Intrinsic functions in MBASIC:

Most users of ROM BASIC will know that a range of mathematical, algebraic and string functions are available. MBASIC has more!

CINT (X) converts X to integer
CSNG (X) converts X to single precision
CDBL (X) converts X to double precision
(the range of the argument for CINT(X) must be between -32768 and 32767, otherwise an arithmetic overflow occurs)

ERR and ERL respectively give the error code and line number of the last error which has occurred.

FIX (X) returns the truncated integer part of X. It is equivalent to SGN(X) * INT(ABS(X)) and does not return the next lower number for a negative value of x, unlike INT(X).

HEX$(X) returns a string in which the value represents the hexadecimal equivalent of the decimal (X). OCT$(X) does the same in octal.

INSTR([I,]X$,Y$) looks for the first occurrence of the string Y$ in the string X$ and indicates the position. I is an optional offset in the range 0 - 255.

SPACE$ (X) returns a string of X spaces; SPC (X) prints X blanks on the terminal.

STRING$ (I,J) returns a string whose length is I and is composed of ASCII code J (J is in the range 0 to 255 but not all the codes are printable).

Other items of possible use....

The statement (or command) RANDOMIZE allows you to 'seed' (or set up) a random number generator.

The largest number that MBASIC will allow is 1.70141E38 and the smallest positive number is 2.9387E-38 .

There is probably no better way of finding out what MBASIC will do than trying it out on short programs - you will probably find out more this way than by studying the manual - although the manual is an invaluable source of reference if you get stuck!

One problem which you may come across is that of portability - not everybody has MBASIC on disk and most of the commands and statements mentioned in this article are only found in extended or disk versions. MBASIC is interpreted on a line by line basis and a program in MBASIC is slower in execution than would be the case with the same program which has been written and compiled using Digital Research's CBASIC which is designed for use with CP/M. MBASIC is usable on systems with all sorts of DOS and appears to have more functions and facilities and, unless speed of execution is important, would be preferable if you were considering the purchase of a disk BASIC.

I would like to thank my colleague, Laurie Hodges for the opportunity to use the MBASIC facility on his machine; thanks are also due to Microsoft for producing a manual which provided the inspiration for this article, and to Dave Hunt for some constructive 'hacking'.

CAN ANYONE HELP?

The following books are out of print, and Rory O'Farrell would like to hear from anyone who can help him obtain copies - all reasonable expenses paid!

A.V. Hershey, "Calligraphy for Computers". NWL Report No. 2101 (Dalgren, Va: U.S. Naval Weapons Laborarory, Aug 1967) NTIS number AD662398.

Wolcott, N.M. and J. Hilsenrath. A Contribution to Computer Typesetting Techniques: Tables of Coordinates from Hershey's Reportory of Occidental Type Fonts and Graphic Symbols. Washington D.C.: National Bureau of Standards Special Publication No 424, U.S. Dept of Commerce, U.S. Government Printing Office, 1976.

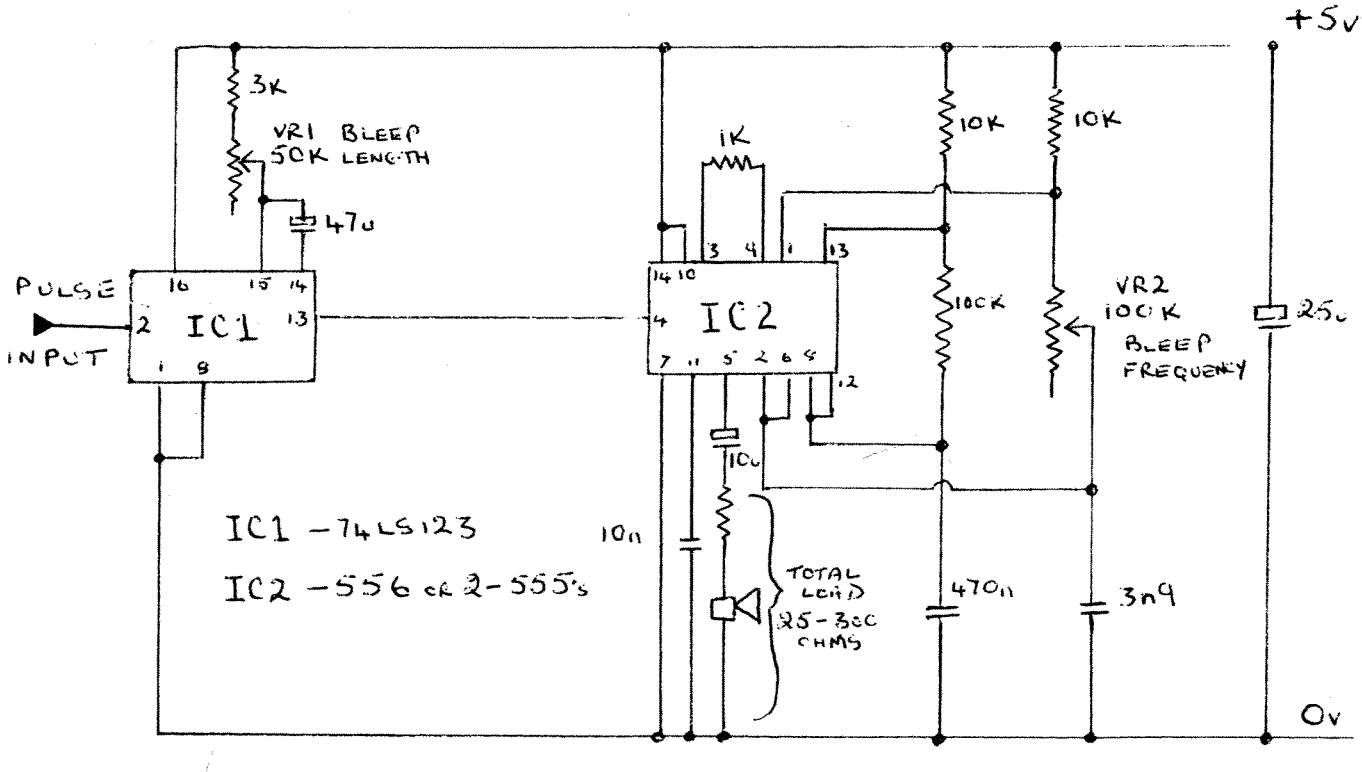
ECONOMY BLEEPER

By R. A. E. Milton

After reading the review of the E.V. Bleeper in Vol 2, Iss 1, I realised that this could be a useful addition to my Nascom, however at £12+ I thought this was a bit pricey. So I decided to take the cheap way out and design my own, it turned out to be so succesful that I thought other readers may find it useful.

As I have not got an IVC I'm not sure what sort of signal is available to trigger the bleeper, but the circuit below is quite simple and could be easily adapted to work from any input. The rising edge of a pulse will trigger the monostable IC1 and this will make the output go high for a period determined by the setting of VR1. Whilst the output of IC1 (and pin 4 of IC2) is high then IC2 will oscillate and produce a two tone warble from the speaker, the tone of the warble can be altered by VR2. The volume is controlled mainly by the type of speaker used. I found an old telephone insert with a DC resistance of 80 ohms which works well, but a conventional speaker can be used. I've got my bleeper hanging on bit 2 of Port 0 where it is activated by sending this bit high, this is simple in machine code or BASIC.

Whilst on the subject of hardware, many moons ago I built the graphics mod (INMC80 issue 5) and found that it worked very well but gave very small vertical gaps between continuous characters. On my Nascom 1 this was overcome by changing IC18 from 74LS123 to an ordinary 74123 and replacing R6 with a 10K ohm preset, this can then be adjusted so that it just removes the gap without the characters breaking up. It's probably a bit naughty but its been working for a year with no trouble.



WARNING BLEEPER.

THE EPSON FX80 PRINTER.

A review by **Rory O'Farrell**

If there is a standard printer in the microcomputer world, it is probably the EPSON MX80. While considering the purchase of one of these, I learned of the imminent arrival of a new model, the FX80. Its preliminary specification and the reputation of EPSON decided me to wait a few weeks and get one of these. À la David Hunt in INMC No. 7, I propose to review this printer, using it to print out its own review, which will be sent to the Editor as camera-ready copy. This will explain any departure from the high standard of typesetting in the 80BUS News.

The FX80 is approximately twice the size of the IMP, measuring 17"w x 14"d x 4"h. As standard, it comes with a Centronics interface, but optional interface boards are readily available to give RS232 and IEEE 488; these may also provide bigger buffers. I purchased my printer with an RS232 8k buffer. The printer was delivered in a sturdy card carrying box, with foam inserts. To insert the 8k RS232 interface buffer, it was necessary to open the case by removing four Phillips-type screws. The 8k buffer plugged into a prefitted socket within, sitting on four plastic supports, for which screws were provided by the buffer manufacturers. Being designed for the MX80, the buffer was a tight squeeze, as a 40 way umbilical cheated it of about $^{1}/_{16}$". By loosening the adjoining board, enough room was created. The buffer board is selectable for the usual parity and stop bit options. These selections are made by sliding little (supplied) connectors down onto prepositioned lines of contacts. These are known as Berg connectors. No soldering is required. This complete, it only remained to set the power-on options of the FX80 and the Microbuffer, then refit the lid. A little panel over the Centronics connector removes to allow access to the DB25 connector for the RS232 interface. The options I set as follows:

Microbuffer Baud rate 1200

Microbuffer Link block S1

Pin B	Open
Pin B	Jumped (Inverted busy)
Pin X	Open
Pin W	Jumped (Hrdwre H/shk)

Microbuffer Link block S2

Pin SB	Jumped (1 stop bit)
Pin WL	Open (8 bit word)
Pin EP	Open (Not selected)
Pin PI	Open (Parity disabled)

FX80 DIP switch 1

SW1.1	Off	Selects 80 column length on power up
SW1.2	On	" 0 font
SW1.3	Off	" paper end detector
SW1.4	Off	" buffer for prog. chars
SW1.5	Off	" Normal print
SW1.6	On	")
SW1.7	Off	) Type font with £ char
SW1.8	Off	)

FX80 DIP switch 2

SW2.1	On	Printer permanently selected
SW2.2	On	Buzzer on
SW2.3	Off	1" perf. skip off
SW2.4	Off	Auto LF off

As supplied, the printer comes with both friction and pin feed. The friction feed can handle roll paper and cut sheet, the pin feed can handle paper from 9½"to 10". I don't think that it will handle the 9.25" paper width of 'A4'. Optional tractors are available to handle from 9" down to 4". There is a little more movement in the pinfeeds, but I'm not sure if it will take that intermediate size. If you are into roll paper feed (jumbo teletype rolls cost very much less than pin feed paper), then order the machine with the roll paper holder.

In its facilities, this printer offers a great deal. It prints using descenders, which greatly improves the legibility of lines. Normally, it prints in Pica, with 80 chars per line. It can provide **Enlarged with 40 cpl**, or Condensed at 137 cpl. Elite mode at 96 cpl, **Enlarged Elite at 48 cpi and Enlarged Condensed are all readily available.** It is also possible to print all characters in **Proportional spacing, where each letter occupies only its correct width.** Besides the above options, the printer can also print in *Italics, which font can be printed in all the above spacings.* But the good news I have kept until last - the printer can have up to 256 programmable characters! These need not all be programmed - it is possible to copy the standard character sets over to the 2k buffer used for the programable characters, and modify only those characters you wish. In this mode, one can access the characters in proportional or constant width modes, and print in the various weights of type illustrated in this review.

It provides 9 different alphabet options for the accent symbols required in other languages. Which one of these is set on powerup is determined by option switch selection, but they can all be accessed under software control. You will have noticed my use of à. This was achieved by switching to the French character set, where it replaces the @ sign. The only character set in which it cannot power up seems to be the version intended for Japan. This uses the standard ASCII set with \ replaced by ¥. Accompanying the FX80 is a substantial manual, spiral bound and nearly 200 pages thick. This explains, using a version of Microsoft BASIC, how to access all the different facilities of the printer. These programs will be easily adapted to the Nascom. Using XBASIC, it is only necessary to substitute PRINT £1; for the LPRINT commands. While printing plain text, the head whizzes along at 160 cps. Fancy printing slows it down somewhat, as does selection of the Quiet mode. This turns the printing speed down to 80cps, with a reduction in noise - but it is not terribly noisy anyway!

Print can be executed in a number of type weights. **This example is in Emphasised, and this in Double strike.** Not all styles are available in all faces, but there are sufficient for most purposes. Line feeds are software programable in a number of options, and form lengths can also be software selected. Both Horizontal and Vertical Tabbing are fully supported, the vertical option supporting up to eight different forms, each with different vertical tabbing positions. Reverse linefeeding

can be achieved with software, which may greatly simplify plotting applications. The various software selectable options are invoked by sending an ESCape code followed by specific byte(s) to the printer. Underlining can be turned on or off at will. The printer can be instructed to set the left margin of the print area, and also the right margin, forcing the type to wrap around when this position is exceeded.

For graphics, this printer can support a number of dot densities. These are

Mode	Dots/8"	Head Speed (inch/sec)
Normal density	480	16
Dual density	960	8
Dual den, dual speed	960	16
Quad density	1920	8
CRT graphics 1	640	8
Plotter	576	12
CRT graphics 2	720	8

These can all be accessed by using the appropriate ESC code, followed by bytes indicating the length of the graphic code following, then by the bytes of graphic information. As the graphics are potentially very powerful, a cursory demonstration of them would not do them justice, so I won't attempt one. I am looking forward to linking the FX80 to my still to be delivered Climax Colour Graphics board to draw (via suitable software!) pictures in black and white of the screen contents.

The table of options printed above configure the FX80 so that it may be plugged in as a replacement for the IMP, with the proviso that the EPSON provides its handshake at RS232 levels on Pin 20 of the Microbuffer RS232 connector, while the IMP supplies it at TTL levels on pin 19. The level shifting circuit is dealt with in detail by David in 80BUS News Vol 1, No 3, p.34.

As I am sure is obvious, I am very pleased with the printer. It costs approx £425 plus VAT, and the 8k serial buffer costs approx £125. A roll paper holder is strongly recommended, and tractor feed would be necessary for label or narrow form handling. Its speed of 160 cps is very impressive, though it does slow down for twiddly bits. I suggest that many of its users will opt to drive it at 9600 or 19.2k baud. I look forward to many years of service from it, when I have finished customising my text editor to allow for the differing spacings and type densities. This is being written on a quickly cobbled up set of patches to an existing text editor, but the printer demands better than that!

80–BUS PORT MAP

Owing to lack of supply of information from two companies (see Editorial) there seems little point taking up pages detailing information that may be in error. However, there is a definite need for some published information on who is using what ports, so the following page gives brief details of the port arrangements recommended by each manufacturer for their own boards. If you know of any additional boards that should be listed, please supply the information ASAP. In the next issue we hope to publish this information in much greater detail.

Arfon Microelectronics

AM819 - Speech board F6

Climax Computers

CC837 - Colour Graphics board C0-D0

EV Computing

EV814 - IEEE 488 board 34-3F

Gemini Microcomputers

GM802 - 64K RAM board . . . . . FF

GM803 - EPROM/ROM board . . . . . FF

GM809 - FDC board E0-E4

GM811 - CPU-I/O board . . . . . B0 & B4-BF

GM812 - Intelligent Video Controller (IVC) board . B1-B3

GM813 - CPU-64K-I/O board . . . . . B4-BF, FE & FF

GM816 - Multi-I/O board 10-2F

GM829 - FDC/SASI board E0-E7

GM833 - 512K RAM-DISK board . . . . . FB-FD

IO Research

IO824 - A/D Convertor board 30-33

IO828 - PLUTO Colour Graphics board A0-A1

MAP 80 Systems

MAP256 - 256K RAM board . . . . . FE

MAPVFC - Video/Floppy Controller board E0-EF

Microcode (Control)

MP826 - 32K CMOS RAM board . . . . . FF

Nascom Microcomputers

Nascom 1/2/3 - CPU-RAM-I/O-Video boards 00-07

RAM B - 48K RAM board . . . . . FF

AVC - Colour Graphics board B0-B2

FDC - FDC board E0-E4

I/O - PIO/CTC/UART board . . . . . 08-0B, 11-12, & 14-1F

NASCOM
ROM
BASIC
DIS–ASSEMBLED

PART I

BY CARL LLOYD–PARKER

A description of BASIC's usage of the memory

*** The work space RAM from 1000 to 10F8 ***

Name	Addr	What it is used for
WRKSPC	1000	Jump to warm start BASIC
USR	1003	Jump for user defined function "USR(X)". This is initialised to give "?FC Error".
OUTSUB	1006	Skeleton for output to port "n" as the 8080 does not have the "OUT (C),r" instruction. The port "n" is loaded into 1007.
DIVSUP	1009	Skeleton subtraction routine for division. The dividend, divisor and quotient cannot all be held in the registers therefore the divisor is loaded into this routine so that there are sufficient registers for the dividend and quotient.
SEED	1017	3 byte seed for random number generator.
	101A	Table of floating point values used by RND. The seed is used to find which value of the eight to multiply the last random number by.
LSTRND	103A	Where the last random number "RND(0)" is kept.
INPSUB	103E	Skeleton for input from port "n" as the 8080 does not have the "IN r,(C)" instruction. The port "n" is loaded into 103F.
NULLS	1041	Number of nulls to output after carriage return. This value is set by the "NULLS n" command.
LWIDTH	1042	Width of terminal. This is set by "WIDTH n" command.
COMMAN	1043	Width of terminal for printing with commas. Why this has to be a seperate byte I don't know, however, the "WIDTH n" command sets LWIDTH but does NOT set this value, this has an irritating result when using BASIC with a printer and trying to have more than three columns using commas in "PRINT" statements, what ever you set "WIDTH" to you don't get more than three columns! This can be overcome in a simple way: "POKE 4163,252" - This makes "WIDTH" work correctly.
NULFLG	1044	Nulls after input byte flag. This is a flag that is examined and then zeroed by the teletype line input routine (TTYLIN). If the character input routine sets this flag before returning the input character then a null is output before the character. The only use I can think of for this is for VERY slow terminals which need to be "woken up" before a character is sent to them!

CTLOFG 1045 Control "O" (Disable output) flag.
If this flag is set then no output will not appear at the terminal. This flag is flipped every time control "O" is typed from the keyboard.

LINESC 1046 "LINES" counter.
This is initially set to the value in LINESN and decremented after every line. If this value reaches zero then it is loaded with the value in LINESN and BASIC waits for a character from the keyboard.

LINESN 1048 "LINES" number.
This is set by the "LINES n" command for the number of lines to be LISTed at a time.

CHKSUM 104A Check sum for array loading and saving.
This accumulates the value of all the bytes in the array so that errors can be detected during "CLOAD*"

NMIFLG 104C Non-maskable interrupt flag.
When an NMI is received this flag is set to let BASIC know that the break was caused by an NMI.

BRKFLG 104D Break flag.
This flag is set by the input routine to let BASIC know that the break key was pressed.

RINPUT 104E Reflection for "INPUT" routine.
When an "INPUT" instruction is encountered BASIC jumps to the input routine via this jump. This value may be changed by the user as in the NASCOM BASIC manual. They go to the following:-

"DOKE 4175,-25"	Output CR,LF and get a line	(CRLIN)
"DOKE 4175,-6670"	No CRLF, get a line	(GETLIN)
"DOKE 4175,-6649"	Get a line by character	(TTYLIN)

POINT 1051 Reflection for "POINT (X,Y)" function (Unused!)
This contains a jump to the "POINT (X,Y)" routine so that the user SHOULD be able to change it to be some other function. However, due to a "mis-firing" in someone's brain when the "POINT" routine was added, the function driver (FNOFST) tests for "POINT" and jumps DIRECTLY to the "POINT (X,Y)" routine!

PSET 1054 Reflection for "SET (X,Y)" routine.
This contains a jump to the "SET (X,Y)" routine in the same fashion as in "POINT". The "SET (X,Y)" routine is called via this reflection so that the user CAN change it to add more routines to BASIC.

RESET 1057 Reflection for "RESET (X,Y)" routine.
This contains a jump to the "RESET (X,Y)" as in PSET.

STRSPC 105A Start of string space pointer.
This contains a pointer to the start of string space. It is initially set to 50 bytes below the end of memory but can be changed by the "CLEAR n" statement.

LINEAT 105C Current line number.
This contains the value of the current line being executed. If a direct statement is being executed then this is -1 and if "Memory size?" is being asked it contains -2. This value is examined by the ERROR routine to see if a line number has to be printed or not. If this value is -2 then the error must have occurred in reply to "Memory size?" so BASIC is cold started again.

BASTXT 105E BASIC program text origin.
This is a pointer to where the BASIC program is stored in memory. It usually points to 10FA which is the usual start of a BASIC program. This can be changed if a program is located somewhere else in memory such as in ROM.

BUFFER 1061 Terminal input buffer.
This is a 72 character buffer where all input from the keyboard is to be stored such as commands, BASIC lines and "INPUT".

CURPOS 10AB Cursor position.
This contains the current value returned by "POS (X)" and it is incremented every time a character is printed. When a new line is started this value is zeroed, however, when a back space is printed it is NOT decremented and when "SCREEN X,Y" is done it is not set to the new "X". This can cause spurious CRLFs to be output when "SCREEN" and back spaces are used. A cure for this is to set "WIDTH 255".

LCRFLG 10AC Locate / Create variable flag.
This is used by the variable search routine to tell if it is in a "DIM" statement or not so that it knows if it has to locate or create the specified array.

TYPE 10AD Data type flag.
This flag contains the "type" of the current expression.
That is:- zero = Numeric, non-zero = string.

DATFLG 10AE Literal statement flag.
This flag is used to tell BASIC that it is pointing at a literal statement such as a string in quotes or a "REM" or "DATA" statement.

LSTRAM 10AF Last available RAM pointer.
This contains the address of the highest location in RAM that BASIC will use. It can be changed by the "CLEAR,n" statement.

TMSTPT 10B1 Temporary string pointer.
This contains a pointer into the temporary string pool.

TMSTPL 10B3 Temporary string pool.
This is a store of 4 temporary strings that were created by such things as "LEFT$","CHR$" and string concatenation.

TMPSTR 10BF Temporary string.
This string block for the current string being constructed. All string blocks are four bytes long. The first byte gives the length of the string. The second byte is not used and the third and fourth bytes form the address in memory where the string itself can be found.

STRBOT 10C3 Bottom of string space.
This contains a pointer to the bottom of the string area that is currently being used. Each time a new string is defined it is moved into the string area below this pointer and the pointer is adjusted to point to the new bottom of string area. If there is not enough room for the new string then a "garbage collection" is performed to remove all unused strings. If there still is not enough room then an "?OS Error" occurs.

CUROPR 10C5 Current operator address.
This contains the address of the current operator in EVAL so that the registers may hold other values without using the stack which is being used to hold sub-expressions.

LOOPST 10C7 Loop start address.
This contains the address of first statement in the FOR loop which is being constructed. This address is later moved into the FOR block on the stack.

DATLIN 10C9 DATA statement line number.
This contains the line number of the current position of the DATA statement pointer. It is used by DATSNR to tell the user in which line the bad DATA occurred.

FORFLG 10CB FOR / FN / array flag.
This contains a flag as to what GETVAR must find.
A value of 00H means find a variable or array element.
A value or 01H means find an array name.
A value of 64H means find a variable only.
A value of 80H means find an FN function.

LSTBIN 10CC This flag is set when ever any input is made into BUFFER.
RETURN first tests to see if the GOSUB was a direct statement and if so checks this flag, if this flag is set then an INPUT statement has occurred within the subroutine and so as far as RETURN is concerned, BUFFER now contains garbage so it jumps back to command mode.

READFG 10CD READ / INPUT flag.
This contains a flag to tell the READ/INPUT routine if it is processing a DATA statement or data for an INPUT statement. If this value is zero then INPUT is active else READ is.

BRKLIN 10CE Break line point.
This contains the address in the line where break occurred. It's value is saved so that CONT knows where to continue.

NXTOPR 10D0 Next operator address.
This contains a pointer into the expression being evaluated by EVAL. It is used to keep track of where it is in the string.

ERRLIN 10D2 Line number of break.
This contains the line number of the line where the break occurred. It is saved so that CONT knows what line it is in.

CONTAD 10D4 Continue address.
This contains the address of the statement where CONT will continue.

******** Values from here on are saved during CSAVE ********

PROGND 10D6 End of program.
This contains the address of the byte after the end of the BASIC program text.

VAREND 10D8 End of variables.
This contains the address of the byte after the last variable.

ARREND 10DA End of arrays.
This contains the address of the byte after the last array.

NXTDAT 10DC Next DATA item.
This contains the address of the next item of DATA to be READ.

FNRGNM 10DE FN argument name.
This contains the name of the argument for the current FN function. If an FN function calls another FN function then this name is saved on the stack.

FNARG 10E0 FN function argument.
This is the floating point value of the the current FN function's argument. If an FN function calls another FN function then this value is saved on the stack together with the name of the FN argument.

FPREG 10E4 Floating point register.
This is a floating point number for the current value. It is a four byte store using 24 bit normalised sign and magnitude representation for the mantissa and excess 128 representation for the exponent of two.
Example of the number 35.25 in floating point:-

```
35.25 in binary is       100011.01
Which is the same as     100011.01 *2^ 00000000

The binary point is moved so that it precedes the first "1"
This gives               .10001101

The point was moved left 6 times dividing the number by 2^6 so
6 must be added to the exponent to re-multiply by 2^6.
This gives             .10001101 *2^ 00000110

As the bit after the point is ALWAYS "1" this bit can be used
to store the sign of the number "0" for +ve, "1" for -ve
So +35.25 is stored as  .00001101 * 2 ^ 00000110
Which in 24 bits is     .000011010000000000000000 *2^ 00000110

128 is added to the exponent so that overflows and underflows
can be more easily detected.
So the whole number in binary is:-
                        00001101 00000000 00000000 10000110
Which is                  0  D     0  0     0  0     8  6   HEX
The bytes of the mantissa are stored in reverse order.
This gives the value for +35.25 as      00 00 0D 86
And -35.25 would be stored as           00 00 8D 86
```

SGNRES 10E8 Sign of result.
This contains the sign of the result for multiplication. Both values to be multiplied are tested and if their signs are different then the product will be negative otherwise it will be positive. The sign for the product is stored here so that it can be tested after to make the result correct.

PBUFF 10E9 Number print buffer.
When a floating point number has to be converted into ASCII for PRINT or STR$ the ASCII number is built up in this buffer by NUMASC so that it can be output or assigned to a string.

NULVAL 10F6 Multiply value.
This contains the 24 bit multiplier because there are not enough registers to hold the multiplier, multiplicand and product all at the same time.

PROGST 10F9 Program start.
This is the byte before the first line in the program. It must be zero to tell the execution driver that the next (actually the first) line is to be executed.

******** How a program is stored in memory ********

Example:- The program:-

```
  10  FOR A=1 TO 5:PRINT A,SQR(A):NEXT A
  20  END
```

is in memory, it would look like this:-

```
PROGND  111C    Pointer to byte after program

10FA    15 11   Pointer to next line (1115)
10FC    0A 00   Line number (10)
10FE    81      FOR token
10FF    20      Space
1100    41      A
1101    B4      = token
1102    31      1
1103    20      Space
1104    A6      TO token
1105    20      Space
1106    35      5
1107    3A      :
1108    9E      PRINT token
1109    20      Space
110A    41      A
110B    2C      ,
110C    BA      SQR token
110D    28      (
110E    41      A
110F    29      )
1110    3A      :
1111    82      NEXT token
1112    20      Space
1113    41      A
1114    00      End of line
1115    1A 11   Pointer to next line (111A)
1117    14 00   Line number (20)
1118    80      END token
1119    00      End of line
111A    00 00   Pointer to next line (0000 = End of program)
111C
```

******** How variables and arrays are stored ********

Variables such as AB AB$ and FN AB are all stored in the simple variable area of memory. The start address of this area is held in PROGND and the end address is held in VAREND.

If AB=10 and AB$="TEXT" and FN AB(XY) had been defined then the memory would look like this

```
42 41           Name of AB in reverse (42 41 = "B" "A")
00 00 20 84     Floating point value for 10

C2 41           Name for AB$ (C2 is "B" with bit 7 set)
04              Length of string (4 characters)
??              This byte unused for string
LL HH           Address where "TEXT" is to be found

42 C1           Name of FN AB (C1 is "A" with bit 7 set)
LL HH           Address of function (After "=")
59 58           Argument name in reverse (59 58 = "Y" "X")
```

Arrays such as AB(1,3) and AB$(3,1) are stored in the array area of memory. The start address of this area is held in VAREND and the end address is held in ARREND.

If DIM AB(1,3),AB$(3,1) had been entered then the memory would look like this:-

```
42 41           Name of array AB in reverse
25 00           Bytes used for array (0025 = 37)
02              2 dimensions
04 00           Size of second dimension including zero element
02 00           Size of first dimension including zero element
00 00 00 00     AB(0,0)
00 00 00 00     AB(1,0)
00 00 00 00     AB(0,1)
00 00 00 00     AB(1,1)
00 00 00 00     AB(0,2)
00 00 00 00     AB(1,2)
00 00 00 00     AB(0,3)
00 00 00 00     AB(1,3)

C2 41           Name of array AB$ in reverse
25 00           Bytes used for array (0025 = 37)
02              2 dimensions
02 00           Size of second dimension including zero element
04 00           Size of first dimension including zero element
00 00 00 00     AB$(0,0)
00 00 00 00     AB$(1,0)
00 00 00 00     AB$(2,0)
00 00 00 00     AB$(3,0)
00 00 00 00     AB$(0,1)
00 00 00 00     AB$(1,1)
00 00 00 00     AB$(2,1)
00 00 00 00     AB$(3,1)
```


******** Usage of the stack for GOSUB/RETURN and FOR/NEXT ********

GOSUB and RETURN usage of the stack

When a GOSUB is executed the address of where to RETURN to and the number of the line to RETURN to are PUSHed on the stack as follows:-

```
HIGH MEMORY:     XX XX   Address of where to RETURN to
                 XX XX   Number of line to RETURN to
STACK POINTER:   8C      GOSUB token as marker
```

This GOSUB block remains on the stack until a RETURN is executed at which point BASIC looks back through the stack until it finds a GOSUB block and then sets the stack there and then POPs the line number and the address of the statement after the GOSUB and continues execution.

The fact that the stack is set to this GOSUB block kills all active FOR loops which were set up inside the subroutine.

FOR and NEXT usage of the stack

When a FOR is executed the address of the first statement in the loop, the line number of the loop statement, the TO value, the STEP value and the sign of the STEP are all PUSHed onto the stack as follows:-

```
HIGH MEMORY:     XX XX           Address of first statement in loop
                 XX XX           Line number of loop statement
                 XX XX XX XX     TO value in floating point
                 XX XX XX XX     STEP value in floating point
                 XX              Sign of STEP
                 XX XX           Address of index variable
STACK POINTER:   81              FOR token as marker
```

This FOR block remains on the stack until a matching NEXT is executed. When next is executed BASIC looks back through the stack to find the matching FOR block. If it is found then the STEP value is added to the value of the index variable and the result is compared with the TO value. With the use of the TO value and the sign of the step BASIC knows if the loop has been completed or not. If it has not been completed then the FOR block remains on the stack until the loop has been completed. The stack is set to point to this FOR block and effectively kills all FORs nested within this loop. When the loop is completed the FOR block is removed from the stack and execution continues from after the NEXT instruction.

If the FOR block cannot be found then a "?NF Error" occurs.

******** Now for the exciting bit - The ROM ********

Name	Addr	What the routine does
INIT	E019	Copy the initial work space conditions into the work space RAM at 1000 to 105F.
MSIZE	E036	Output "Memory size" prompt and get response from user. If a number is given jump to TSTMEM otherwise start at the beginning of RAM and test each successive byte until the end of RAM is found. When the end has been found jump to SETTOP.
TSTMEM	E05B	Get supplied address. If any bad characters output "?SN Error" and re-initialise otherwise try to write a D9H byte into that address. If D9H is not read back then there is no RAM at that address so go to MSIZE to ask again. If D9H is read back then drop through to SETTOP.
SETTOP	E06D	Test for minimum memory requirement and if not enough jump to MSIZE to ask again otherwise save this address as the highest available memory location, allocate 50 bytes for string space and output the sign on message and the number of bytes free. Then drop through to WARMST.
WARMST	E0AE	Clear run-time registers and jump to PRNTOK.
BAKSTK	E356	Look back through stack for FOR or GOSUB blocks and exit with HL pointing to the block.
CHKSTK	E38A	Check for "C" levels of stack space and output "?OM Error" if not enough stack space.
DATSNR to TMERR	E3A7 to E3BF	I like this section of code because you jump into the routine at one of the entry points, this loads the "E" register with the error number and the "LD BC,nn" opcode (01H) skips all the following "LD E,n" instructions. It is very efficient.
ERROR	E3C1	Output error code for error number in the E register, output the line number if needed and drop through to PRNTOK.
PRNTOK	E3F8	Output "Ok" message and drop through to GETCMD.
GETCMD	E405	Get a direct statement or BASIC line, CRUNCH the text and if it is a direct statement go to EXCUTE to execute it. If it is a BASIC line then move it to the program text area.
SRCHLN	E499	Search for line number in register DE.
NEW	E4B9	Set PROGND to start of program text area to effectively remove program from memory and execute the null program to clear all the other pointers.
RUNSFT	E4C5	Execute program from fitst statement.
INTVAR	E4C9	Clear out variables and string space an reset DATA pointer.

CLREG	E4DF	Reset stack to top of memory, clear out temporary strings, disable CONTinue and execute program.
PROMPT	E4FC	Output "? " prompt for INPUT and go to get input line.
CRUNCH	E509	Move text string from HL to BUFFER crunching reserved words into tokens as you go.
ENDBUF	E5B8	Mark the end of BUFFER with three nulls.
GETLIN	E5F2	Get an input line from monitor.
TTYLIN	E607	Get an input line by character and save it in BUFFER.
CPDEHL	E68A	Compare the DE register with the HL register returning flags:- Z - Registers are equal, C - DE > HL and NC- DE <= HL
CHKSYN	E690	Make sure the next byte in the code string is the sam as the byte following the "CALL CHKSYN". If not - output "?SN Error".
OUTC	E69B	Output character in the A register to the terminal and output a CRLF if the current terminal width is exceeded. Character is not output if CTLOFG is set.
CLOTST	E6CC	Get an input character and flip CTLOFG if it is control "O".
LIST	E6DD	LIST BASIC program from start (or specified line number) LINESN lines at a time. It does not take into account that graphics may appear in quotes, DATA or REM statements so the graphic character is treated as a reserved word token and expanded to the complete word.
FOR	E779	Assign the initial value to the index variable and then set up a FOR block on the stack.
RUNCNT	E7F2	Main execution driver loop. This gets the next statement (May be after ":" or on next line) and gives it to EXCUTE.
EXCUTE	E816	Test current statement. If it is a key word then find the address of the routine from WORDTB and call it. If it is a letter then call LET to try to assign a variable. Otherwise generate "?SN Error".
GETCHR	E836	Get next character in code string and return flags:- Z - End of statement (":" or null) C - Character is a digit otherwise NC.
RESTOR	E846	RESTORE routine. Set DATA pointer to start of program or to specified line if a line number is given.
TSTBRK	E861	Test for break key. If pressed test for control "S" (pause). If another break pressed then stop execution and output break message.
STOP	E870	STOP routine. Flag "STOP" an join END.

END	E872	END routine. Flag "END". Save address of break and set continue address and line. If from STOP or break then output "Break" message and if not a direct statement then output line number also.
CONT	E89E	CONT routine. Get continue address, if it is zero then output "?CN Error" else set address and line as current and continue.
NULL	E8B1	NULL routine. Set number of nulls to be output after CRLF.
CHKLTR	E977	Test next character in code string and return flags:- NC - Letter, C - Not a letter.
FPSINT	E97F	Get an integer 0 to 32767 from next character.
POSINT	E982	Get an integer 0 to 32767 to DE.
DEPINT	E985	Test for integer 0 to 32767 and leave it in DE.
DEINT	E98B	Test for integer -32768 to 32767 and leave it in DE.
ATOH	E9A5	Read an ASCII integer 0 to 65529 into DE from code string.
CLEAR	E9CA	CLEAR routine. "CLEAR" Clear variables, "CLEAR s" also reserves string space (where t is 0 to 32767), "CLEAR,t" sets top of memory (where t is 0 to 32767), "CLEAR s,t" reserves string space and sets RAM top.
RUN	EA10	RUN routine. "RUN" run from start of program, "RUN n" run from specified line.
GOSUB	EA1C	GOSUB routine. Create a GOSUB block on the stack and GOTO specified line.
GOTO	EA2D	GOTO routine. Get ASCII line number and continue execution from that line.
RETURN	EA4B	RETURN routine. Look back through stack for a GOSUB block and if one is found POP line number and address in line and continue execution from there. Otherwise "?RG Error" occurs.
DATA	EA70	DATA routine. Flag DATA and drop through to common code.
REM	EA72	REM routine. Flag REM and look for end of statement, DATA statement ends with ":" or null, REM statement ends only with null (End of line).
LET	EA87	LET routine. Get variable name, make sure "=" follows, evaluate expression and assign it to the variable.
ON	EAE1	ON GOTO / ON GOSUB routine. Get integer expression (0 to 255). Look through list of ASCII line numbers until the Nth value is found. If value N exists GOTO or GOSUB that line. Otherwise drop through to next statement.

IF	EAFF	IF routine. Evaluate expression. If it is zero then dop through to the next line. Otherwise GOTO line number if specified else go to execute the statement.
PRINT	EB23	PRINT routine.
INPUT	EBFD	INPUT routine. Test for direct mode, if in direct mode then no INPUT allowed ("?ID Error"). Otherwise output prompt string if one is given, Get an input line and join READ routine to assign values to variables.
READ	EC2C	READ routine. Get address of next DATA item and flag READ. Read in items an assign to variables (used for INPUT and READ). If any errors occur use READFG to see if:- "?SN Error" or "?OD Error" are needed for READ or "Redo" or "Extra ignored" are needed for INPUT.
FDTLP	ECD2	Find next DATA statement.
NEXT	ECF6	NEXT routine. Look back through stack for a matching FOR block. If one is found then add STEP to index variable and compare it with the TO value. If the loop has not been completed then continue execution from the first statement in that loop. Otherwise remove the FOR block from the stack. Then look for another variable (Eg "NEXT B,A"). If found then re-enter NEXT routine to process this loop. Otherwise continue execution from the statement after the NEXT.
GETNUM	ED41	Get a numeric expression.
TSTNUM	ED44	Make sure current value is numeric ("?TM Error" if not).
TSTSTR	ED45	Make sure current value is a string ("?TM Error" if not).
CHKTYP	ED46	Make sure current value is of the type selected by TYPE.
EVAL	ED5A	Evaluate an expression using algebraic logic (Do multiplication and division before addition and subtraction. Etc.). Leave the result in FPREG and set TYPE to string or numeric.
FNOFST	EE33	Enter with offset into FNCTAB. If it is "POINT (X,Y)" then jump directly to "POINT (X,Y)" routine (This SHOULD have been a jump via the POINT reflection). If it is LEFT$, RIGHT$ or MID$ then evaluate the string expression and get the "," after it then go to the string function. If it is any other function then get its address from FNCTAB and call it.
SGNEXP	EE70	Get sign of number and return in the D register:- FF - Negative, 00 Positive.

DIM	EF28	DIM routine. Flag "create" variable and join common [illegible].
GETVAR	EF2D	Look for specified variable. Use FORFLG to:- 1. Not allow subscripted FOR loops. 2. Look for an FN function definition. 3. Look for an array name for CLOAD* and CSAVE*. 4. Or just look for the variable as supplied.
FRE	F0D0	FRE(X) routine. If X is a numeric expression then return the amount of memory between ARREND and the stack. If X is a string expression then do a "Garbage collection" and return the number of unused bytes in the string area.
POS	F0FE	POS(X) routine. Return the current cursor position (CURPOS).
DEF	F106	DEF FN routine. Define a user defined function. This is not legal in direct mode.
DOFN	F133	Call an FN function. Save any previous argument name and value on the stack, Evaluate the function an then restore the previous argument name and value.
IDTEST	F17B	If in direct mode then output "?ID Error" else return.
CHEKFN	F189	Make sure "FN" follows and flag find FN function definition.
STR	F19A	STR$ routine. Convert current numeric value to ASCII and create a temporary string for it.
SAVSTR	F1AA	Save current string in string area.
MKTMST	F1BF	Make a temporary string.
PRNUMS	F20F	Print number string at HL.
PRS	F210	Print string at HL.
TESTR	F229	Test if enough string space. If insufficient then GRBAGE collect and test again. If still insufficient space then output "?OS Error".
GRBAGE	F253	Do a "Garbage collection" on string area. Scan through all string variables and shift all active strings to the top of the string area thus leaving the free space in one block.
CONCAT	F306	Concatenate two strings (Eg A$+B$).
GETSTR	F350	Get a string routine.
LEN	F382	LEN(X$) routine.
ASC	F391	ASC(X$) routine.
CHR	F3A2	CHR$(X) routine.

LEFT	F3B2	LEFT$(X$,X) routine.
RIGHT	F3E2	RIGHT$(X$,X) routine.
MID	F3EC	MID$(X$,P[,L]) routine. MID$(X$,P) returns string from character P to the end. MID$(X$,P,L) returns string from position P for L characters.
VAL	F41C	VAL(X$) routine.
INP	F441	INP(X) routine. Set up port number in work space and call INPSUB skeleton. Return value from port.
POUT	F44D	OUT P,N routine. Set up port number in work space and call OUTSUB skeleton to output value to the port.
WAIT	F453	WAIT P,A,X routine. Set up port number in work space. Call INPSUB to get byte, XOR value with X if X is supplied and then AND the value with A. If this value is zero then go back to WAITLP until value is non-zero.
FNDNUM	F481	Get next number from code string.
CSAVE	F4C3	CSAVE routine. If CSAVE* jump to array save routine ARRSV1. Evaluate string expression for name and save program.
CLOAD	F4F9	CLOAD routine. If CLOAD* jump to array load routine ARRLD1. See if CLOAD? and save status. Evaluate string expression for file name (None given - any file will do) and search for that file. When found load it into memory.
PEEK	F5A3	PEEK(X) routine.
POKE	F5AA	POKE A,V routine.
ROUND	F5BB	Round number up by adding 0.5 to it.
ADDPHL	F5BE	Add floating point number at HL to the current value in FPREG.
SUBPHL	F5C4	Subtract the current value from the value at HL and leave the result in FPREG.
PSUB	F5C8	Subtract FPREG from the value on the stack and leave the result in FPREG.
SUBCDE	F5CA	Subtract FPREG from the value in BCDE and leave the result in FPREG.
FPADD	F5CD	Add the value in BCDE to the value in FPREG.
MINCDE	F60D	Subtract value in FPREG from value in BCDE.
NORMAL	F638	Normalise value in BCDE.
PLUCDE	F672	Add value in FPREG to value in BCDE.
COMPL	F67E	Complement value in BCDE.

LOG	F6C7	LOG(X) routine. Get LOG of FPREG. Scale the number to be between 0 and 1. Add SQR(1/2) to value. Divide into SQR(2). Subtract from 1. Compute the sum of series using LOGTAB for coefficients. Subtract 0.5 from result. Re-scale the number and then multiply by LOG(2).
MULT	F706	Multiply FPREG by value on stack.
FPMULT	F708	MULTIPLY FPREG by value in BCDE.
DIV	F767	Divide the value on stack by FPREG and leave the result in FPREG.
DVBCDE	F768	Divide the value in BCDE by FPREG and leave the result in FPREG.
TSTSGN	F813	Test sign of number in FPREG.
SGN	F822	SGN(X) routine.
ABS	F838	ABS(X) routine.
STAKFP	F844	Move value in FPREG to stack.
PHLTFP	F851	Move value at HL to FPREG.
FPBCDE	F854	Move value in BCDE to FPREG.
BCDEFP	F85F	Move FPREG to BCDE.
LOADFP	F862	Move value at HL to BCDE.
FPTHL	F86B	Move Value in FPREG to HL.
SIGNS	F879	Set sign of result depending on signs of operands.
CMPNUM	F88E	Compare numbers.
FPINT	F8BB	Get integer of FP value.
INT	F8E6	INT(X) routine.
ASCTFP	F91A	Convert ASCII floating point number into binary.
NUMASC	F9B8	Convert floating point binary into ASCII.
SQR	FAAC	SQR(X) routine. Uses SQR(X) = X ^ 0.5
POWER	FAB5	Raise base BCDE to the power FPREG. Uses X^Y = EXP(Y*LOG(X)).

EXP	FAFA	EXP(X) routine. Scale value to be between 0 and 1 Compute sum of series using EXPTAB for coefficients. Re-scale number.
SUMSER	FB5B	Sum the series using table of coefficients at HL. ((N1 * X^2 + N2) * X^2 + N3) * X^2 ... + Nn
SUMSER1	FB6A	Sum the series using table of coefficients at HL. ((N1 * X + N2) * X + N3) * X ... + Nn
RND	FB8B	RND(X) routine.
COS	FC00	COS(X) routine. Uses COS(X) = SIN(X+PI/2).
SIN	FC03	SIN(X) routine. Divide angle by 2*PI. Get fraction part. Move other quadrants around and set what result will be. Sum the series using table of coefficients at SINTAB. Adjust result.
TAN	FC67	TAN(X) routine. TAN(X) = SIN(X) / COS(X).
ATN	F67C	ATN(X) routine. If value > 1 then get 1/value and set PI/2 - angle. Sum the series using table of coefficients at ATNTAB. Negate result if original value was > 1.
WIDTH	FDA5	WIDTH routine. Set terminal width but NOT commas width.
LINES	FDAD	LINES routine.
DEEK	FDBC	DEEK(X) routine.
DOKE	FDC7	DOKE A,V routine.
SCREEN	FDE6	SCREEN X,Y routine.
SCRADR	FE11	Get screen address from row and column in DE and BC.
INLINE	FEE8	Get an input line from NAS-SYS.
GETXYA	FF15	Get (X,Y) for SET,RESET and POINT. Return address on screen and a bit mask.
SETB	FF40	SET(X,Y) routine.
RESETB	FF55	RESET(X,Y) routine.
POINTB	FF79	POINT(X,Y) routine.
XYPOS	FF96	Convert (X,Y) to a row and column on screen.

A PROBLEM WITH POWER–ON RESET ON THE NASCOM 2

C. Bowden

I recently discovered a hardware problem on my system that could cause a lot of difficulty in some circumstances. It seems that the fault could be fairly common, as I found out that another Nascom 2 user local to me had also had problems.

The power-on-reset (POR) on my system had never worked reliably, from the beginning, but I had not bothered to look for the cause, as it did not trouble me too much and it is not easy to get at my Nascom 2 card. I simply used the manual reset. This continued even after I multi-mapped my system as described in earlier issues of this newsletter. I had adopted the habit of always switching to Nas-Sys (i.e. standard Nascom), before switching off, so that the next time the machine was switched on, resetting would not cause a CP/M boot, with SIMON looking for non-existent disks, thus avoiding 'HEAD BANGING'.

My normal procedure was to switch on, and press reset TWICE(!!) in quick succession. Nas-Sys would announce itself, and I could proceed to use whatever system I desired, using the System Switch. (It was necessary to use 2 resets to get Nas-Sys booted correctly.)

A few weeks ago though, I started to interface a Gemini GM822 RTC module, and I had trouble with Clock Data corruption. In the search for the cause, I decided that it was time to get the POR working, since I did not know what the Z80 was doing between the manual Resets, and this could be contributing to the data corruption. It did not take long to get the POR working, once I had got the N2 card out of its case. In order to test it in this position though, I had to run the card on its own, with POR set to address 0000H, and with Nas-Sys in its normal socket. The POR appeared to work correctly. However, when the system was reassembled in its case, with the POR set to operate at 2000H/4000H (depending on the DIL switch LSW1 and my system switch), there were still some very peculiar results.

With Nas-Sys selected (reset jump to 4000H), a manual reset was STILL needed after the POR. However, with CP/M selected (reset jump to 2000H), the POR appeared to work correctly. In order to see what was happening under Nas-Sys, I connected up a TV to the modulated O/P of the N2, and my video monitor to the IVC card O/P. On running up the system, I discovered that the POR loaded SIMON and looked for a disk, irrespective of the system switch position. I still had to make the second, manual reset to get Nas-Sys loaded correctly. Without a logic analyser or some other suitable testgear, it is impossible to be sure where the POR was going, but it seems likely that it was going to 0000H, and then the Z80 was charging up through RAM until it found some program to execute. In my case this was the CP/M boot ROM at 2000H on the paged in EPROM card. (Which would be found irrespective of the position of the system switch.)

Having found out what was happening, the next question was why? The trouble was almost certainly due to some malfunction of the reset-jump-multiplexer. I spent some time poking about with a 'scope and logic probe, and tried changing chips, but all with no result. A colleague who runs identical hardware had no problems on his system. He loaned me his EPROM card to try, but the results were just the same, so the trouble seemed to be on the CPU card. After a lot more thinking and poking about I decided to see if anyone knew the answer, so I rang Nascom and a couple of other people, but the fault seemed unknown to them, so apart from a couple of suggestions, they could not help.

I had been offered the loan of a CPU card by another friend, which I had not previously taken up because it meant a ten mile round trip, but it seemed the next logical step, so I borrowed the card, and tried it. It ran perfectly. I took it out, and decided to check the components of the POR circuit. They were identical to mine EXCEPT for the capacitor C3. On Phil's card this was 100uf instead of the 68uf supplied and as shown in the circuits and component lists. I rang Phil. Yes, he had had trouble with reset at some time in the past and had probably altered C3, but he could not remember the details. I changed C3 to

150uf, as I did not have anything else of suitable value handy, and put the card back into the case. This time the POR worked perfectly, and it has done ever since. (It should be equally valid to increase R13 to say 12K/15K.)

I have not analysed the action of the POR circuits of the N2, but it seems that the value of the time constant of C3/R13 can be critical. The time constant C2/R11 in the manual reset circuit is identical in component values. In consideration of the 'cure' for this fault it seems that C3/R13 should have a longer time constant than C2/R11 for correct circuit action. By the laws of averages, this should be true for 50% of the Nascom 2's with 10K/68uf fitted in both positions, allowing for component tolerances. In any case, if this assumption is true, the fault will only be noticed in a proportion of those N2's that are trying to reset to some other address than 0000H.

I am not an expert in digital logic. Perhaps someone can send in an article explaining the operation of the circuit, with some timing diagrams, as my cure may not be the best one for this problem. It does work though.

I would like to thank Alan Stevens and Phil Harvey for the loan of cards, which helped me to isolate the problem. I hope that this article may help others who have experienced similar problems with the power-on-reset on their system.

LARGE RAM SYSTEMS USING THE MAP CARD

by Richard Beal

This note refers to the review of the MAP 256K RAM card on pages 40 to 44 of Vol 2 Issue 1 of 80-BUS News. A problem has arisen when using the Gemini GM813 CPU/64K RAM card with more than one MAP RAM card. The reason is that the GM813 does not provide an A19 signal - see "The Great A19 Debate" on page 43 of that issue for a detailed explanation. This means that the 64K of RAM on the GM813 is addressed both at the bottom of the one megabyte of addressable memory, and also half way through. This obviously causes problems, since if two 256K MAP cards are added, the last 64K of the second card is overlaid by the bottom 64K on the GM813.

The solution is to disable the 64K on the 813, and alter the header plugs on the MAP cards so that they provide memory starting at OK, 256K, 512K and 768K. This is very simple and MAP will happily tell you how to do this.

The best way to disable the memory on the 813 is a simple hardware modification to permanently disable the RAM. MAP will provide a recommended alteration. Alternatively, the 813 RAM could be addressed on a different 64K page, using the original paging scheme which has four pages of 64K. This has the disadvantage that as well as the GM813 needing modifying, a patch to RP/M is needed, which makes it nonstandard. I don't approve of that, but MAP may prefer to adopt this solution.

You may feel that it is incredibly wasteful to effectively throw away 64K of RAM, but if you have a system with 512K or more, you probably don't mind!

Note that with Gemini's own GM833 512Kbyte RAM-DISK board you can go upto 8 Megabytes with no mods. to either the GM813 or GM833!

SERIAL INTERFACE PROBLEMS MADE EASY

by Richard Beal

The serial interface is a simple and convenient way to connect together computers, VDU terminals, modems, acoustic couplers, printers and many other devices. The RS232 or V24 standard is universal and works very well at speeds of at least 9600 bps. The hardware designer has only to incorporate a UART on a board, and a few wires to a 25 pin connector, and the cost is very low. The software needed is very simple, and nothing can go wrong!

But anyone who has tried to plug together two computers using the RS232 link has probably experienced problems. The difficulty is that the standard is intended for the connection of terminals, such as VDUs or printer terminals, to a host computer, or to a modem linked to a host computer. But is a microcomputer a terminal or a host? When linked to a printer it is clearly the host, but when linked to a modem it is a terminal. The connections for hosts and terminals are completely different, and this is where the problems start.

This article tries to explain how to persuade all these devices to talk to each other using RS232, and how to wire up the plugs for all possible cases.

The first problem is that there are two types of 25 pin connector, one male and the other female. Unfortunately most manufacturers are now so confused that they seem to randomly fit male or female connectors to their computers. In fact Gemini have got rather confused themselves, as you will see. A general rule is that hosts should have female sockets that need a cable with a male plug, and terminals should have male sockets that need a cable with a female plug. Another confusing fact is that the sex of a plug is determined by the 25 tiny pins and not by the shape of the entire plug. Also the pin numbering is forwards on one type and in reverse on the other, as you would expect. The pin numbers can usually be read near the pins, if you have good eyesight.

The normal RS232 connecting lead consists of a male plug at the host end and a female plug at the terminal end. The cable links pins 2, 3, 4, 5, 6, 7, 8 and 20 to the same pins at the other end. This cable can be used to link modems to terminals, or computers to printers. This assumes that neither manufacturer got confused. If you try this lead with a Gemini Galaxy, it may work, but you are taking a slight risk, because the socket on the Galaxy is wired up partly as a host and partly as a terminal.

Connections to the Galaxy serial socket

Pin 2	Data from device into the Galaxy
Pin 3	Data to device from the Galaxy
Pin 7	Ground (always)

Now this is correct for a host, and since it is a host type socket, so far so good.

Pin 4	RTS out
Pin 5	CTS in
Pin 6	DSR in
Pin 8	CD in
Pin 20	RTS out

Now these look correct, and one can sympathise with the designer for deciding to connect these up to the corresponding lines from the UART. But UART manuals for some reason always assume that you are building a terminal and never a host. In fact if you are connecting to a printer, pins 4 and 20 carry signals from the printer, and pins 5, 6 and 8 are expecting to receive signals

from the Galaxy. Now it does not matter that 5, 6 and 8 are in each case an input connected to an input, but it may matter that 4 and 20 are outputs connected to outputs, and these outputs may be attempting to assert different values. Fortunately RS232 devices seem very resilient, and I have not heard of any damaged printers because of the two devices arguing over lines 4 or 20.

Similarly if you want to connect the Galaxy to a modem or acoustic coupler, then not only is the socket the wrong type, but lines 2 and 3 need to be reversed. But pins 4, 5, 6, 8 and 20 are now correct.

How to wire up the connectors correctly

The ideal solution is to wire two sockets from the present one. One would be as a host, and the other as a terminal. The socket acting as a terminal is identical to the present socket with two changes. Firstly it is a male socket, and secondly the lines to pins 2 and 3 are reversed. The socket acting as a host is identical to the present socket, except that the RTS line from the Gemini is wired to pins 5, 6 and 8 of the socket, providing the signals expected by a terminal. Also RTS should be wired back to the CTS, DSR and CD lines leading to the Gemini UART, since this provides the signals that it expects. Pins 4 and 20 on the socket contain signals from the terminal which should not be connected to the Gemini board. The RTS line from the Gemini should not be connected.

To make this clearer, here are the exact connections from the Gemini serial connector to a host socket, which should be female.

Gemini Serial Connector			Cannon D (Female)	
Data in	3	<-----	2	Data from terminal kbd
Data out	6	----->	3	Data to printer
RTS out	10	--X X--	4	RTS out
CTS in	8	<---->*	5	CTS in
DSR in	9	<---->*	6	DSR in
Ground	11	<---->	7	Ground
CD in	1	<---->*	8	CD in
DTR out	12	--* X--	20	DTR out

X means no connection. In addition to the connections shown, the DTR signal from the Gemini is connected to 5, 6 and 8 in the Cannon connector.

A particular advantage of this wiring is that the DTR signal from the Gemini provides a useful indication of the state of the Gemini RS232 interface. When the Gemini is Reset, the Data out line gives an invalid data signal, since it is not yet conditioned to even operate at the correct speed. But the DTR signal informs the terminal that the host is not ready, and the terminal will ignore the input. For example when connected to a Diablo terminal, pressing Reset causes the audible alarm on the terminal to sound. Then a moment later the DTR signal is restored and the terminal will function correctly. This is most important if there is no IVC in the system, and a serial terminal is being used as the console, either of RP/M or of CP/M. If you simply fix the input signals at the terminal so that it is always ready, then the initial messages may be corrupted for about ten characters until bit alignment is achieved.

Serial Connections for the Nascom 2

These connections do not work for the Nascom 2 serial connector, since this is not fully compatible with the Gemini. [Ed. - The Nascom 2 does not have modem control signals, and although a handshake signal can be provided, this is not at RS232 levels.] However you may find it useful to modify an N2 to conform

to this standard. The only feature lost is the 20mA interface, which is obsolete and unlikely to ever be needed. Cut the tracks leading to pins 1 (was drive out, now CD in), 9 (was 20 mA in, now DSR in), and 12 (was 20mA out, now DTR out). Then connect pin 2 to pin 12, giving 5V on the DTR out signal. Check your connections carefully as you will not wish to blow up your Nascom. Some cables connect 5, 6, 8 and 20 at the terminal end. If the terminal puts 12V on its DTR then this could be fed back to the N2 5V supply with disastrous results, so be careful!

Special Purpose Leads

It is possible to build special leads to connect a host to a host or a terminal to a terminal. In each case every device believes the other to be what it expects.

In both cases pin 7 (ground) is connected to pin 7 at the other end. Pin 2 at one end is connected to pin 3 at the other, and vice versa, since the inputs and outputs must be reversed. Do not connect any other wires between the devices.

The host to host lead could be used to connect two computers together, using their printer sockets - if they are wired up correctly! At each end, connect pin 6 (DSR out) to pins 4 and 20 (RTS in and DTR in). This fools each host into believing that it is connected to a terminal.

The terminal to terminal lead has pin 20 (DTR out) connected to pins 5, 6 and 8 (CTS in, DSR in, CD in) at each end. This fools each terminal into believing it is connected to a host.

Other Articles

There are many articles explaining various aspects of the RS232 interface. There is a useful section in D. R. Hunt's article in the 80-BUS news Vol 1 Issue 3, pages 32 to 36.

Software

When the computer is acting as a host, the normal operating system software can use the serial interface to drive a printer. But it is not so obvious what to do then the computer is to act as a terminal device. In this case a program is run which makes the computer act as a terminal. A comprehensive terminal emulation program is given in full below. Another program which is very useful is MODEM7, which is available from the CP/M Users Group, and is easily made to operate on a Nascom or a Gemini. This allows files to be transferred down phone lines (using modems or acoustic couplers), and it automatically recovers from line errors by patiently retransmitting the data when needed.

Terminal Emulation Software

It can be a problem to find software which allows a computer to act as a simple terminal. This is not as simple as you might at first think, and therefore an example for the Gemini IVC is printed below.

Now for the program which turns a Gemini into a dumb terminal. This is not as simple as it seems for several reasons. Firstly as soon as the speed gets high enough, the display cannot keep up with the incoming data, and characters start to be lost. Also this program allows characters to be echoed to a serial printer which may be very slow, for example 10 cps. Another complication is that the terminal may operate in half duplex or full duplex. This program allows for all these options, as you can see from the comments at the start of it. It can be assembled using M80.

```
----- Start of program listing -----

        page 60
        title TERMB  Buffered Terminal Program
        .z80
; Program for a simple half or full duplex terminal.
; Operates at slow and fast speeds.  Tested at up to 9600 bps.
; Execute with command TERM H for half duplex, otherwise it is full duplex.
; Type Control-C to exit to CP/M.  Patch 0103H to change this character.
; Type Control-P to toggle printer.  Patch 0104H to change this character.

; Configuration is Gemini IVC for display and keyboard.
; The UART is an 8250, or a 6402.
; The printer may be either the CP/M list device, or a serial printer
; which may have an 8250 or 6402 UART.
; If the printer is the CP/M list device, then this must be fast enough
; to keep up with the input data.
; Code could be added for a Centronics printer, scanning for serial input
; while waiting for the printer to be ready.

false   equ 0
true    equ not false

; UART type - 8250 if true, 6402 if false
u8250   equ false       ; UART to host/modem
p8250   equ false       ; UART for serial printer, if selected

; Printer type - CP/M list device if true, serial printer if false
cpml    equ false

; Ports for UART to host/modem
        if u8250
uartd   equ 0b8h        ; Data
uarts   equ uartd+5     ; Status
        else
uartd   equ 11h         ; Data
uarts   equ uartd+1     ; Status
        endif

; Ports for serial printer
        if p8250
puartd  equ 0b8h        ; Data
puarts  equ puartd+5    ; Status
        else
puartd  equ 01h         ; Data
puarts  equ puartd+1    ; Status
        endif

; Ports for IVC
pvdud   equ 0b1h        ; Data
pvdus   equ 0b2h        ; Status

; Keyboard command characters
ichex   equ conc        ; Exit to CP/M
ichpr   equ conp        ; Toggle printer

; CP/M routines
list    equ 5           ; List output
condir  equ 6           ; Direct console I/O
jwboot  equ 0000h       ; Warm boot
jbdos   equ 0005h       ; CP/M
f1      equ 005dh       ; Parameters

; Characters
conc    equ 03h         ; Control-C
bell    equ 07h         ; Bell
bs      equ 08h         ; Backspace
lf      equ 0ah         ; Line feed
cr      equ 0dh         ; Carriage return
conp    equ 10h         ; Control-P
esc     equ 1bh         ; Escape
cul     equ 1ch         ; Cursor left

; Working storage
chex:   defb ichex      ; Exit to CP/M - can patch this value
chpr:   defb ichpr      ; Toggle printer - can patch this value
abuf:   defw buffer     ; Input position in buffer
obuf:   defw buffer     ; Output position in buffer
sprt:   defb 0          ; Printer toggle - 0 to not print
param:  defb 0          ; H for half duplex, otherwise full duplex
vbusy:  defb 0          ; FF when VDU is busy answering keyboard request
        defs 32         ; Stack
stack:

; Start
start:  ld sp,stack

; Load parameter entered in command line
        ld a,(f1)
        ld (param),a

; Start of main processing loop

; Scan for serial input
term2:  call serin

; Examine buffer
term3:  ld hl,(abuf)
        ld de,(obuf)
        or a
        sbc hl,de
        jr nz,term3a
        ld hl,buffer    ; Empty, so reset buffer
        ld (abuf),hl
        ld (obuf),hl
        jr term6
term3a: ld a,(de)       ; Get char from buffer
        inc de
        ld (obuf),de
```

```
; Output to console
term4:  and 7fh          ; Strip parity
        cp bs            ; If BS make CUL
        jr nz,term4a
        ld a,cul
        jr term5
term4a: cp cr            ; OK if CR
        jr z,term5
        cp lf            ; OK if LF
        jr z,term5
        cp bell          ; OK if Bell
        jr z,term5
        cp 20h           ; Otherwise ignore non-printing chars
        jr c,term2
        cp 7fh
        jr z,term2

; Output to display
term5:  ld e,a           ; Save for printing
        push de
        call vdu
        pop de

; Print
        ld a,(sprt)      ; Test toggle
        or a
        call nz,print

; Test keyboard
term6:  call keyin
        or a             ; No input
        jr z,term2
        ld hl,chex       ; Exit to CP/M
        cp (hl)
        jp z,jwboot
        ld hl,chpr       ; Toggle printer
        cp (hl)
        jr nz,term10
        ld a,(sprt)
        cpl
        ld (sprt),a
        jr term2

; Output to UART
term10: call serout

; To display if half duplex
        ld e,a
        ld a,(param)
        cp "H"
        jr nz,term2
        ld a,e
        jr term4
```

```
; I/O routines

; Check keyboard
keyin:  ld a,0ffh        ; Flag VDU busy
        ld (vbusy),a
        ld e,esc         ; Check if char entered
        call vdu
        ld e,"k"
        call vdu
        call getivc
        or a
        jr z,key2        ; No input
        ld e,esc         ; Get character
        call vdu
        ld e,"K"
        call vdu
        call getivc
key2:   push af
        xor a            ; Flag VDU normal
        ld (vbusy),a
        pop af
        ret

; Get character from the IVC
getivc: call serin       ; Scan for serial input
        in a,(pvdus)
        rla
        jr c,getivc
        in a,(pvdud)
        ret

; Send character in E to IVC
vdu:    push de
        call serin       ; Scan for serial input
        pop de
        in a,(pvdus)     ; Check status
        rra
        jr c,vdu
        ld a,e
        out (pvdud),a    ; Output character
        ret

; Print character in E
print:  if cpm1
        ld c,list        ; Output to CP/M list device
        jp jbdos
        else
        ld a,e           ; Output to serial printer
        or a             ; Make even parity
        jp pe,sp2
        xor 80h
sp2:    push af
sp4:    call serin       ; Scan for serial input
        in a,(puarts)
        if p8250
        bit 5,a
        else
        bit 6,a
```

```
        endif
        jr z,sp4
        pop af
        out (puartd),a
        ret
        endif

; Serial input routine
serin:  in a,(uarts)
        if u8250
        rra
        else
        rla
        endif
        ret nc
        in a,(uartd)
        ld hl,(abuf)    ; Store in buffer
        ld (hl),a
        inc hl
        ld a,(jbdos+2)  ; Check buffer size limit
        dec a
        cp h
        jr nz,st4
        ld a,(vbusy)    ; If VDU busy, cannot signal error, so ignore
        or a
        ret nz
        ld e,bell       ; Ignore char and sound bell
        ld c,condir     ; (Probably losing more chars)
        jp jbdos
st4:    ld (abuf),hl    ; Update buffer position
        ret

; Serial output routine
serout: or a            ; Make even parity
        jp pe,ser2
        xor 80h
ser2:   push af
ser4:   call serin      ; Scan for serial input
        in a,(uarts)
        if u8250
        bit 5,a
        else
        bit 6,a
        endif




        jr z,ser4
        pop af
        out (uartd),a
        ret

; Buffer
buffer:

        end start
```

```
W
L
0010 {  ************************
0020    * The Towers of Hanoi *
0030    * by J Hunt   12/6/82 *
0040    ************************
0050  }
0060 {$L-,I+,C-}
0070 PROGRAM TOWERSofHANOI;
0080 CONST
0090   maxdisc=7;
0100   mdisc2p1=15;
0110   flen=47;  {maxdisc*6+5}
0120 TYPE
0130   INT=INTEGER;
0140 VAR
0150   disc:ARRAY[0..maxdisc] OF
0160            ARRAY[1..mdisc2p1] OF CHAR;
0170   n :ARRAY[1..3] OF INTEGER;
0180   floor:ARRAY[1..flen] OF CHAR;
0190   qry,pchar,dchar,bchar:CHAR;
0200   peg:ARRAY[1..3,0..maxdisc] OF INTEGER;
0210   ndiscs,move:INT;
0220   ok:BOOLEAN;
0230 {}
0240 PROCEDURE title;
0250 BEGIN
0260   POKE(£0BCA,
0270     ' +++ The TOWERS of HANOI +++');
0280 END;
0290 {}
0300 PROCEDURE screen(x,y:INT);
0310 CONST cursor=£0C29; vll=£080A;
0320 BEGIN
0330   IF (x<1) OR (x>48) THEN HALT;
0340   IF (y<1) OR (y>16) THEN HALT;
0350   POKE(cursor,vll + 64*(y-1) + x-1);
0360 END;
0370 {}
0380 PROCEDURE clearline(x,y:INT);
0390 BEGIN
0400   screen(x,y);
0410   WRITE(' ':48-x);
0420   screen(x,y);
0430 END;
0440 {}
0450 FUNCTION readint:INT;
0460 VAR r:REAL;
0470     c:CHAR;
0480     digits:SET OF CHAR;
0490     sign:INT;
0500 BEGIN
0510   digits:=['0'..'9'];
0520   r:=0.0;
0530   sign:=1;
0540   READLN;
0550   REPEAT READ(c) UNTIL c<>' ';
0560   IF c='-' THEN BEGIN
0570     sign:=-1;
0580     READ(c);
0590     END
0600   ELSE IF c='+' THEN
0610     READ(c);
```

```
0620    WHILE c IN digits DO BEGIN
0630      r := r*10.0 + ORD(c) - ORD('0');
0640      READ(c);
0650    END;
0660    IF r>MAXINT THEN r:=MAXINT;
0670    readint:=ROUND(r) * sign;
0680  END;
0690  {}
0700  PROCEDURE setgame;
0710  VAR i,j,d:INTEGER;
0720    PROCEDURE setdiscs;
0730    VAR i,j:INTEGER;
0740    BEGIN
0750     FOR i:=0 TO maxdisc DO BEGIN
0760       FOR j:=1 TO mdisc2p1 DO
0770         disc[i][j] := ' ';
0780       disc[i][maxdisc+1] := pchar;
0790       FOR j := 1 TO i DO BEGIN
0800         disc[i][maxdisc+1-j] := dchar;
0810         disc[i][maxdisc+1+j] := dchar;
0820       END;
0830     END;
0840    END;
0850  BEGIN {setgame}
0860    pchar:=CHR(255); dchar:=CHR(128);
0870    bchar:=CHR(129);
0880    setdiscs;
0890    FOR i := 1 TO flen DO
0900      floor[i] := bchar;
0910    PAGE;  title;
0920    WRITE('1':maxdisc+2,'2':mdisc2p1,
0930          '3':mdisc2p1);
0940    screen(1,3);
0950    FOR i:=0 TO maxdisc DO BEGIN
0960      d:= i - maxdisc + ndiscs;
0970      IF d<0 THEN d:=0;
0980      WRITE(' ',disc[d]);
0990      WRITE(pchar:maxdisc+1,pchar:mdisc2p1);
1000      WRITELN;
1010    END;
1020    WRITELN(floor);
1030    n[1]:=ndiscs; n[2]:=0; n[3]:=0;
1040    FOR i:=1 TO ndiscs DO
1050      peg[1,i] := ndiscs+1-i;
1060    FOR i:=1 TO 3 DO
1070      peg[i,0] := 999;
1080    move:=1;
1090  END;
1100  {}
1110  FUNCTION movedisc(p1,p2:INT):INT;
1120  VAR error:INT;
1130  BEGIN
1140    error:=-1;
1150    clearline(5,15);
1160    IF p1=p2 THEN
1170      WRITE('That is no move!')
1180    ELSE IF n[p1]=0 THEN
1190      WRITE('No discs on this peg!')
1200    ELSE IF peg[p1,n[p1]] > peg[p2,n[p2]]
1210      THEN
1220      WRITE('Not allowed!')
1230    ELSE BEGIN
1240      error:=0;
1250      move:=move+1;
1260      n[p2] := n[p2]+1;
1270      peg[p2,n[p2]] := peg[p1,n[p1]];
1280      n[p1] := n[p1]-1;
1290  screen((p1-1)*mdisc2p1+2,3+maxdisc-n[p1]);
1300      WRITE(disc[0]);
1310  screen((p2-1)*mdisc2p1+2,4+maxdisc-n[p2]);
1320      WRITE(disc[peg[p2,n[p2]]]);
1330    END;
1340    movedisc:=error;
1350  END;
1360  {}
1370  PROCEDURE demonstrate;
1380  VAR i:INT;
1390    PROCEDURE pause;
1400    VAR i:INT;
1410    BEGIN
1420      FOR i:=1 TO  5000 DO;
1430    END;
1440    {}
1450    PROCEDURE longpause;
1460    VAR i:INT;
1470    BEGIN
1480      FOR i:=1 TO 6 DO pause;
1490    END;
1500    {}
1510    PROCEDURE moveheap(count,p1,p2,use:INT);
1520    BEGIN
1530      IF count>1 THEN
1540        moveheap(count-1,p1,use,p2);
1550      screen(5,14);
1560      WRITE('Move:',move:4);
1570      IF movedisc(p1,p2) <> 0 THEN HALT;
1580      pause;
1590      IF count>1 THEN
1600        moveheap(count-1,use,p2,p1);
1610    END;
1620  BEGIN {demonstrate}
1630    screen(5,13);
1640    WRITE('First from peg 1 to peg 3...');
1650    longpause;
1660    moveheap(ndiscs,1,3,2);
1670    clearline(5,13);
1680    WRITE('And now back to peg 1...');
1690    longpause;
1700    move:=1;
1710    moveheap(ndiscs,3,1,2);
1720    clearline(5,13);
1730    move:=1;
1740    longpause;
1750    clearline(5,14);
1760  END;
1770  {}
1780  PROCEDURE instruct;
1790    PROCEDURE howmany;
1800    VAR ok:BOOLEAN;
1810    BEGIN
1820      REPEAT
1830        clearline(5,13);
1840      WRITE('How many discs do you want? ');
1850        ndiscs:=readint;
1860        ok:=(ndiscs>0) AND (ndiscs<=maxdisc);
1870        clearline(5,15);
1880        IF NOT ok THEN
1890          WRITE('Think again!');
1900      UNTIL ok;
1910    END;
1920  BEGIN  {instruct}
1930    PAGE; title; WRITELN;
```

```
1940    WRITELN('   A number of discs, all dif',
1950            'ferent in size,');
1960    WRITELN('are placed in order on a fixe',
1970            'd peg, with the');
1980    WRITELN('largest at the bottom. There ',
1990            'are two empty pegs');
2000    WRITELN('in line with the first. Your ',
2010            'task is to move');
2020    WRITELN('all the discs to peg 3, one a',
2030            't a time, such');
2040    WRITELN('that at no time a larger disc',
2050            ' rests on a');
2060    WRITELN('smaller disc.');
2070    WRITELN;
2080    WRITELN('The maximum number of disc',
2090            's is',maxdisc:2,'.');
2100    howmany;
2110  END;
2120  {}
2130  PROCEDURE play;
2140  VAR ok:BOOLEAN;
2150      err,p1,p2,i,min:INT;
2160  BEGIN
2170    REPEAT {until game over}
2180      REPEAT
2190        clearline(5,13);
2200        WRITE('Move:',move);
2210        screen(15,13);
2220        WRITE('Move disc from peg ');
2230        p1 := readint;
2240        ok := (p1>0) AND (p1<4);
2250        IF ok THEN BEGIN
2260          screen(36,13);
2270          WRITE('to ');
2280          p2 := readint;
2290          ok := (p2>0) AND (p2<4);
2300        END;
2310        clearline(5,15);
2320        IF NOT ok THEN
2330          WRITE('Dont be silly!');
2340      UNTIL ok; {valid peg numbers}
2350      err:=movedisc(p1,p2);
2360    UNTIL (n[2]=ndiscs) OR (n[3]=ndiscs);
2370    min:=1;
2380    FOR i:=1 TO ndiscs DO min:=min*2;
2390    min:=min-1;
2400    clearline(15,13);
```

```
2410    IF move=min+1 THEN
2420      WRITE('Well done, you made no ',
2430            'mistakes.')
2440    ELSE
2450      WRITE('It can be done in ',min,
2460            'moves!');
2470    screen(5,14);
2480    IF n[3]=0 THEN
2490      WRITE('But that is the WRONG PEG!');
2500  END;
2510  {}
2520  BEGIN
2530    REPEAT
2540      instruct;
2550      setgame;
2560      REPEAT
2570        clearline(5,13);
2580        WRITE('Shall I demonstrate? ');
2590        READLN; READ(qry);
2600        clearline(5,15);
2610        ok:=(qry='N') OR (qry='Y');
2620        IF NOT ok THEN WRITE('What?');
2630      UNTIL ok;
2640      IF qry='Y' THEN BEGIN
2650        demonstrate;
2660        REPEAT
2670          clearline(5,13);
2680          WRITE('Are you ready to play ',
2690                '(Y/N)? ');
2700          READLN; READ(qry);
2710          ok:=(qry='Y') OR (qry='N');
2720          clearline(5,15);
2730          IF NOT ok THEN WRITE('What?');
2740        UNTIL ok;
2750        END
2760      ELSE qry:='Y';
2770      IF qry='Y' THEN play;
2780      REPEAT
2790        clearline(32,14);
2800        WRITE('Again (Y/N)? ');
2810        READLN; READ(qry);
2820        ok:=(qry='Y') OR (qry='N');
2830        clearline(32,15);
2840        IF NOT ok THEN WRITE('What?');
2850      UNTIL ok;
2860    UNTIL qry='N';
2870    PAGE;
2880  END .
```

NASCOM BASIC CROSS–REFERENCE LISTING

M. J. R. Gibbs

The listing on the following pages is for a utility that I wrote to give a variable cross-reference listing of a Nascom BASIC program. One of the problems of Nascom BASIC is that only the first two characters of a variable name are used, so that AQS, AQ and AQ1, which are all valid names, are considered identical. A number of people make this mistake in their programs - I have certainly done so, and for a beginner it is often not obvious what is wrong.

This program may help in this respect, and I have also found it useful when developing or changing a program sometime after it was originally written. It is also very useful when developing a large program where a large number of variable names can become a problem.

The program is located in RAM between 0C80H and 0F65 and will give a sorted list of variable names (only the first two characters being displayed to avoid confusion). String variables are identified by a '$' and array variables by a '(', followed by a list of the statement numbers in which that variable is referenced.

For example, the following slightly modified benchmark program:-

```
100 REM BENCHMARK 7
110 PRINT "S"
120 K = 0
130 DIM MQQQ(5)
140 K = K+1
150 A = K/2*3+4-5
160 GOSUB 230
170 FOR L = 1 TO 5
180 MQQQ(L) = A
190 NEXT L
200 IF K < 1000 THEN 140
210 PRINT "E"
220 END
230 RETURN
```

Would result in a screen containing:-

```
            < NASCOM CROSS REFERENCE >

A           150  180
K           120  140  150  200
L           170  180  190
MQ(         130  180
```

The program is very easy to run. Leave BASIC using the MONITOR command. Load the utility from tape or disk (once loaded it can be left for future use as BASIC does not use that portion of RAM). Run the program by entering EC80, which will give you the maximum of a screenful of information for studying. To continue press any key, which will give successive screens until the listing is complete, and then return to the monitor. BASIC can be re-entered if desired by using the warm start command (Z).

One word of warning. This program generates a lot of data. This is stored at the end of the BASIC program and in normal use presents no problem. However, in the case of a very large program the data can overflow the available space with unpredictable results. You are therefore advised to save any large BASIC program before running this utility. I have never had any program corrupted, but it is possible.

```
0F63            ;       ********************************
0F63            ;       * NASCOM BASIC CROSS REFERENCE *
0F63            ;       * ============================ *
0F63            ;       *          M.J.R.GIBBS         *
0F63            ;       *          20'05'83            *
0F63            ;       ********************************
0C80                    ORG   £0C80
0C80                    LOAD £8C80
1000            BASIC   EQU   £1000           ;BASIC LOCATION
0018            NASSYS  EQU   £18
007B            BLINK   EQU   £7B
006A            CRLF    EQU   £6A
005B            STOP    EQU   £5B
0030            CRT     EQU   £30
0028            PRS     EQU   £28
0C29            CURPOS  EQU   £0C29
0BCA            TOPLNE  EQU   £0BCA
000C            CLEAR   EQU   £0C
000D            CRET    EQU   £0D
0022            QUOTE   EQU   '"'
008E            REM     EQU   £8E
000E            LINES   EQU   14
002F            VDUW    EQU   8*6-1
002F            ;       ********************************
002F            ;       *      CROSS REFERENCE PROG    *
003F            ;       ********************************
0C80 2AD610             LD    HL,(BASIC+£D6);HL = A(END OF PROG)
0C83 110005             LD    DE,£500         ;DE = £500
0C86 19                 ADD   HL,DE           ;CALC STACK POS
0C87 22500F             LD    (STACK),HL      ;LOAD IT
0C8A CD200F     XREF    CALL  HEAD
0C8D 0021FE10           LD    IX,BASIC+£FE    ;IX = BASIC SOURCE SCAN
0C91 2AFA10             LD    HL,(BASIC+£FA);HL = 1ST  ADD POINTER
0C94 7C                 LD    A,H             ;CHECK IF PROG AVAILABLE
0C95 B5                 OR    L               ;ie   1ST ADD = 0
0C96 2003               JR    NZ,X10
0C98 CF                 RST   NASSYS          ;NO   PROG AVAIL
0C99 5B                 DB    STOP
0C9A 00                 NOP
0C9B 2A5C0F     X10     LD    HL,(STACK)      ;SETUP STACK
0C9E 22570F             LD    (LEN),HL        ;SET  POINTER
0CA1 2AFC10             LD    HL,(BASIC+£FC);HL = 1ST LINE NO
0CA4 22610F             LD    (LINENO),HL     ;SET  FIRST LINE NO
0CA7 DD7E00     LOOPX1  LD    A,(IX+0)        ;A   = CHR
0CAA FE00               CP    £00             ;IS   00 ie EOB
0CAC CAA40D             JP    Z,XEOB
0CAF 1804               JR    X30             ;MAIN SCAN LOOP
0CB1 DD23       NEXTX1  INC   IX              ;TRY  NEXT LOC
0CB3 18F2               JR    LOOPX1
0CB5 FE8E       X30     CP    REM             ;IGNORE REM
0CB7 2008               JR    NZ,X40
0CB9 AF                 XOR   A               ;SCAN FOR NEXT EOB
0CBA DD23       LOOPRM  INC   IX              ;REM  FOUND
0CBC DDBE00             CP    (IX+0)          ;IS   IT EOB
0CBF 20F9               JR    NZ,LOOPRM       ;NO   TRY AGAIN
0CC1 C3A400             JP    XEOB            ;YES  NEXT BLOCK
0CC4 FE22       X40     CP    QUOTE           ;IGNORE IN QUOTES
0CC6 2011               JR    NZ,X50          ;NOT  QUOTE
```



```
0CC8 0023       LOOPQU  INC   IX              ;LOOK FOR " OR EOB
0CCA DD7E00             LD    A,(IX+0)        ;GET  CHR
0CCD FE00               CP    £00             ;IS   EOB
0CCF CAA40D             JP    Z,XEOB
0CD2 FE22               CP    QUOTE           ;IS   OUT OF QUOTE
0CD4 28D8               JR    Z,NEXTX1        ;YES  TRY FOR VAR
0CD6 C3C80C             JP    LOOPQU          ;CONT
0CD9 FE41       X50     CP    'A'             ;IS   ASCII 1-Z
0CDB 38D4               JR    C,NEXTX1
0CDD FE5B               CP    'Z'+1           ;FIRST CHR IS ALPHA
0CDF 30D0               JR    NC,NEXTX1       ;NO   TRY AGAIN
0CE1 325F0F             LD    (WORK1),A       ;LOAD 1ST CHR
0CE4 3E01               LD    A,£1            ;SET  TYPE VARIABLE
0CE6 32600F             LD    (TYPE),A        ;LOAD TYPE
0CE9 3E20               LD    A,' '           ;BLANK OUTWORK1
0CEB 215E0F             LD    HL,WORK2        ;HL = A(NEXT VAR)
0CEE 77                 LD    (HL),A          ;LOAD BLANK
0CEF DD23       GOTX20  INC   IX              ;EXAMINE NEXT
0CF1 DD7E00             LD    A,(IX+0)        ;A   = NEXT CHR
0CF4 FE00               CP    £00             ;IS   EOBLK
0CF6 2834               JR    Z,LOADIT        ;YES  EOB
0CF8 FE5B               CP    'Z'+1           ;IS   ALPHA OR NUM
0CFA 3030               JR    NC,LOADIT       ;YES  LOAD IT
0CFC FE20               CP    ' '
0CFE 28EF               JR    Z,GOTX20        ;IGNOR SPACES
0D00 FE24               CP    '$'             ;IS   STR$
0D02 2816               JR    Z,STR           ;YES
0D04 FE28               CP    '('             ;IS   ARRAY
0D06 281C               JR    Z,ARRAY         ;YES
0D08 FE30               CP    '0'             ;2ND  NUMERIC
0D0A 38E0               JR    C,LOADIT        ;NO   MUST BE END
0D0C FE3A               CP    '9'+1           ;NUM  9
0D0E 30D4               JR    C,GOTX30
0D10 FE41               CP    'A'             ;2ND  ALPHA
0D12 38E8               JR    C,LOADIT
0D14 77         GOTX30  LD    (HL),A          ;LOAD 2ND VAR
0D15 215A0F             LD    HL,JUNK         ;IGNORE JUNK
0D18 18D5               JR    GOTX20          ;TRY  FOR NEXT
0D1A 3A600F     STR     LD    A,(TYPE)        ;SET  STRING
0D1D CBD7               SET   2,A             ;BIT 2 = STR
0D1F 32600F             LD    (TYPE),A
0D22 18C8               JR    GOTX20
0D24 3A600F     ARRAY   LD    A,(TYPE)        ;SET  ARRAY
0D27 CBCF               SET   1,A             ;BIT 1 = ARRAY
0D29 32600F             LD    (TYPE),A
0D2C 2A5C0F     LOADIT  LD    HL,(STACK)      ;HL = A(STACK)
0D2F ED5B5E0F           LD    DE,(WORK2)      ;IY = VAR hl
0D33 B7         LD10    OR    A               ;CHECK FOR END OF STACK
0D34 ED48570F           LD    BC,(LEN)        ;BC = A(END STACK)
0D38 ED42               SBC   HL,BC           ;IS   END OF STACK
0D3A 09                 ADD   HL,BC
0D3B 2853               JR    Z,INSZ          ;YES  MUST NEED INSERT
0D3D E5                 PUSH  HL              ;IX = HL
0D3E FDE1               POP   IY              ;IY = STACK LOC
0D40 FD4601             LD    B,(IY+1)        ;BC = THIS VAR
0D43 FD4E00             LD    C,(IY+0)
0D46 EB                 EX    DE,HL           ;KEEP HL
0D47 B7                 OR    A
```

```
0D48 ED42           SBC  HL,BC        ;IS   THIS THE SAME
0D4A 09             ADD  HL,BC
0D4B EB             EX   DE,HL        ;RESTORE HL
0D4C 3842           JR   C,INS7       ;NO   HI SO MUST INSERT
0D4E 200A           JR   NZ,LOAD20    ;NO   LO TRY NEXT
0D50 3A600F         LD   A,(TYPE)     ;CHECK SAME TYPE
0D53 FD8E02         CP   (IY+2)
0D56 3838           JR   C,INS7       ;NO   HI SO INSERT
0D58 280A           JR   Z,INS2       ;YES  LOAD ONLY LINENO
0D5A 3EFF     LOAD20 LD   A,#FF        ;NOT  FOUND TRY NEXT
0D5C EDB1           CPIR              ;DOWN TO #FFFF eob
0D5E BE             CP   (HL)         ;CHECK 2ND BYTE
0D5F 20F9           JR   NZ,LOAD20    ;TRY AGAIN
0D61 23             INC  HL           ;INDEX PAST #FFFF
0D62 18CF           JR   LD10         ;TRY  NEXT BLOCK
0D64 ED5B610F INS2  LD   DE,(LINENO)  ;DE = LINE NUMBER
0D68 3EFF           LD   A,#FF        ;A   = #FF EOB
0D6A EDB1     X60   CPIR              ;SCAN FOR EOB
0D6C BE             CP   (HL)         ;CHECK SECOND BYTE
0D6D 20FB           JR   NZ,X60
0D6F 2B             DEC  HL           ;SET  HL = A(#FFFF)
0D70 E5             PUSH HL           ;IY = HL
0D71 FDE1           POP  IY
0D73 FD46FF         LD   B,(IY-1)     ;BC = LAST LINE NO
0D76 FD4EFE         LD   C,(IY-2)
0D79 EB             EX   DE,HL        ;TEST IF SAME AS LAST
0D7A B7             OR   A
0D7B ED42           SBC  HL,BC        ;CHECK IT
0D7D 09             ADD  HL,BC
0D7E EB             EX   DE,HL        ;RESTOR HL
0D7F CAA70C         JP   Z,LOOPX1     ;YES  ALREADY GOT IT
0D82 3E02           LD   A,2          ;INSERT IT IN STACK
0D84 CDAE0E         CALL SHIFT        ;CREATE SPACE
0D87 FD7300         LD   (IY+0),E     ;LOAD NEW NUMBER
0D8A FD7201         LD   (IY+1),D
0D8D C3A70C         JP   LOOPX1       ;REMOVE FOR LISTING
0D90 3E07     INS7  LD   A,#07        ;LOAD NEW BLOCK
0D92 CDAE0E         CALL SHIFT        ;MOVE STACK
0D95 E5             PUSH HL           ;KEEP HL
0D96 D9             EXX               ;KEEP REGS
0D97 D1             POP  DE           ;DE = LOAD LOC
0D98 215E0F         LD   HL,WORK2     ;HL = A(BLOCK)
0D9B 010700         LD   BC,#0007     ;BC = L(BLOCK)
0D9E EDB0           LDIR
0DA0 D9             EXX               ;RECOVER REGS
0DA1 C3A70C         JP   LOOPX1       ;REMOVE FOR LISTING
0DA4 010500   XEOB  LD   BC,#0005     ;BC = L(00 AA AA NN NN)
0DA7 DD8E01         CP   (IX+1)       ;IS   END OF PROG
0DAA 2005           JR   NZ,XEOB10
0DAC DD8E02         CP   (IX+2)
0DAF 280E           JR   Z,PRINT      ;YES  GO PRINT THE RESULTS
0DB1 DD09     XEOB10 ADD  IX,BC       ;INDEX IX TO NEXT BLOCK
0DB3 DD66FF         LD   H,(IX-1)     ;HL = NEW LINE NO
0DB6 DD6EFE         LD   L,(IX-2)
0DB9 22610F         LD   (LINENO),HL  ;SET  LINENO
0DBC C3A70C         JP   LOOPX1
0DBF CD200F   PRINT CALL HEAD
0DC2 DD2A5C0F       LD   IX,(STACK)
```



```
0DC6 3EFF           LD   A,#FF        ;FOOL FIRST BLOCK
0DC8 DD77FF         LD   (IX-1),A     ;LOAD DUMMY #FFFF
0DCB DD77FE         LD   (IX-2),A
0DCE 060E           LD   B,LINES      ;ALLOW PAGING
0DD0 ED43550F       LD   (LCNT),BC    ;SET  LINE COUNTER
0DD4 3E20     LNE10 LD   A,' '        ;A   = BLK
0DD6 21EE0E         LD   HL,LINE+1    ;BLANK OUT LINE
0DD9 11EF0E         LD   DE,LINE+2    ;DE = 2ND LOAD POINT
0DDC 012F00         LD   BC,VDUW      ;BC = LEN OF LINE-1
0DDF 77             LD   (HL),A       ;BLANK FIRST  LOC
0DE0 EDB0           LDIR              ;BLANK REST OF LINE
0DE2 FD21EE0E       LD   IY,LINE+1    ;IY = START ADD
0DE6 3EFF           LD   A,#FF        ;IS FIRST IN BLOCK
0DE8 DDBEFF         CP   (IX-1)       ;CHECK EOB MARKER
0DEB 202C           JR   NZ,LNE40     ;NO
0DED DDBEFE         CP   (IX-2)       ;TRY  2ND BYTE
0DF0 2027           JR   NZ,LNE40
0DF2 DD7E01         LD   A,(IX+1)     ;LOAD VAR NAME 1ST
0DF5 FD7700         LD   (IY+0),A     ;PUT  IN PRINT
0DF8 DD7E00         LD   A,(IX+0)     ;LOAD VAR NAME 2ND
0DFB FD7701         LD   (IY+1),A     ;PUT IN PRINT
0DFE DD23           INC  IX
0E00 DD23           INC  IX
0E02 DD7E00         LD   A,(IX+0)     ;A   = TYPE
0E05 DD23           INC  IX
0E07 CB57           BIT  2,A          ;IS   STRING
0E09 2805           JR   Z,LNE30
0E0B 0E24           LD   C,'$'        ;LOAD PRINT '$'
0E0D FD7102         LD   (IY+2),C
0E10 CB4F     LNE30 BIT  1,A          ;IS   ARRAY
0E12 2805           JR   Z,LNE40
0E14 0E28           LD   C,'('        ;LOAD PRINT '('
0E16 FD7103         LD   (IY+3),C
0E19 FD21F40E LNE40 LD   IY,LINE+7    ;SET PRINT INDEX
0E1D DD6601   LNE50 LD   H,(IX+1)     ;HL = LINE NUMBER
0E20 DD6E00         LD   L,(IX+0)
0E23 DD23           INC  IX           ;INC TO NEXT
0E25 DD23           INC  IX
0E27 23             INC  HL           ;CHECK FOR EOB
0E28 7C             LD   A,H          ;ie   #FFFF+1 = 0
0E29 B5             OR   L
0E2A 2816           JR   Z,NXTLNE     ;FINISH THIS LINE
0E2C 2B             DEC  HL           ;RESET HL BACK
0E2D CD700E         CALL ENCODE       ;LOAD PRINT AREA
0E30 FD23           INC  IY           ;ALLOW SPACE
0E32 FDE5           PUSH IY           ;IS   END OF LINE
0E34 E1             POP  HL
0E35 011C0F         LD   BC,LNEND     ;BC = A(END OF LINE)
0E38 B7             OR   A
0E39 ED42           SBC  HL,BC
0E3B 38E0           JR   C,LNE50      ;NO TRY NEXT NUMBER
0E3D CDED0E         CALL LINE         ;PRINT LINE
0E40 181B           JR   NXTLN2       ;DO  PAGING
0E42 DDE5     NXTLNE PUSH IX          ;HL = IX
0E44 E1             POP  HL
0E45 ED4B570F       LD   BC,(LEN)     ;CHECK END OF STACK
0E49 B7             OR   A
0E4A ED42           SBC  HL,BC
```

```
0E4C 2815              JR   Z,ENDLST
0E4E FDE5              PUSH IY              ;HL = IY
0E50 E1                POP  HL
0E51 01F40E            LD   BC,LINE+7       ;EMPTY LINE ?
0E54 B7                OR   A
0E55 ED42              SBC  HL,BC
0E57 CA040D            JP   Z,LNE10
0E5A CDED0E            CALL LINE
0E5D CD0A0E    NXTLN2  CALL PAGE
0E60 C3D40D            JP   LNE10
0E63 CDED0E    ENDLST  CALL LINE
0E66 DF                RST  NASSYS
0E67 7B                DB   BLINK
0E68 3E0C              LD   A,CLEAR         ;CLEAR THE SCREEN
0E6A F7                RST  CRT
0E6B 00                NOP
0E6C 00                NOP
0E6D DF                RST  NASSYS
0E6E 5B                DB   STOP
0E6F 00                NOP
0E6F           ;       ********************************
0E6F           ;       *        SUBROUTINE ENCODE *
0E6F           ;       ********************************
0E70 DDE5      ENCODE  PUSH IX              ;KEEP INDEX
0E72 DD21400F          LD   IX,NUM          ;IX = POWER
0E76 3E01              LD   A,01            ;SUPRESS LEADING ZERO'S
0E78 32590F            LD   (LZIND),A       ;SET  IT ON
0E7B 0605              LD   B,05            ;B  = LOOP CNT
0E7D AF        ENCLP1  XOR  A               ;A  = 0
0E7E DD5601            LD   D,(IX+1)        ;DE = POWER
0E81 DD5E00            LD   E,(IX+0)
0E84 3C        ENCLP2  INC  A               ;A  = D COUNT
0E85 ED52              SBC  HL,DE           ;FIND DIGIT
0E87 30FB              JR   NC,ENCLP2       ;ANY  MORE
0E89 19                ADD  HL,DE           ;RESTORE NUM
0E8A 3D                DEC  A               ;RESET COUNT
0E8B FE00              CP   00              ;SUPRES LEAD 0
0E8D 2007              JR   NZ,ENCT10       ;NOT  ZERO
0E8F 3A590F            LD   A,(LZIND)       ;IS   ZERO SUP ON
0E92 FE00              CP   00              ;0    = OFF
0E94 200D              JR   NZ,ENC20        ;SUP  ON
0E96 C630      ENCT10  ADD  A,30            ;CONV ASCII
0E98 FD7700            LD   (IY+0),A        ;LOAD BUF
0E9B FD23              INC  IY              ;INDEX TO NEXT
0E9D AF                XOR  A               ;A  = 0
0E9E 32590F            LD   (LZIND),A       ;LEADING ZERO IND OFF
0EA1 1802              JR   ENCNXT
0EA3 FD23      ENC20   INC  IY              ;YES  INSERT BLANKS
0EA5 DD23      ENCNXT  INC  IX              ;INDEX TO NEXT
0EA7 DD23              INC  IX              ;       POWER
0EA9 10D2              DJNZ ENCLP1          ;REPEAT
0EAB DDE1              POP  IX              ;RECOVER IX
0EAD C9                RET
0EAD           ;       ********************************
0EAD           ;       *         SUBROUTINE SHIFT *
0EAD           ;       ********************************
```



```
0EAE E5        SHIFT   PUSH HL              ;KEEP LOC
0EAF D9                EXX                  ;SAVE REGS
0EB0 D1                POP  DE              ;KEEP DE FROM OLD REGS
0EB1 0600              LD   B,0             ;BC = LENGTH OF SPACE
0EB3 4F                LD   C,A
0EB4 2A570F            LD   HL,(LEN)        ;HL = A(END OF STACK)
0EB7 B7                OR   A
0EB8 ED52              SBC  HL,DE           ;END OF STACK ?
0EBA 19                ADD  HL,DE
0EBB 2817              JR   Z,SHIFTE
0EBD 09                ADD  HL,BC           ;HL = A(NEW END OF STACK)
0EBE E5                PUSH HL              ;KEEP END
0EBF 2A570F            LD   HL,(LEN)        ;HL = A(END OF STACK)
0EC2 B7                OR   A
0EC3 ED52              SBC  HL,DE           ;HL = L(TO BE MOVED-1)
0EC5 23                INC  HL              ;HL = L(TO BE MOVED)
0EC6 E5                PUSH HL              ;BC = HL
0EC7 C1                POP  BC
0EC8 D1                POP  DE              ;RECOVER A(NEW END)
0EC9 2A570F            LD   HL,(LEN)        ;HL = A(OLD END)
0ECC ED53570F          LD   (LEN),DE        ;SET  LEN TO NEW LENGTH
0ED0 EDB8              LDDR                 ;MOVE IT ALL 'A' BYTES
0ED2 D9                EXX                  ;RECOVER REGS
0ED3 C9                RET
0ED4 09        SHIFTE  ADD  HL,BC           ;SET NEW END
0ED5 22570F            LD   (LEN),HL
0ED8 D9                EXX
0ED9 C9                RET
0ED9           ;       ********************************
0ED9           ;       *         SUBROUTINE PAGE  *
0ED9           ;       ********************************
0EDA ED4B550F  PAGE    LD   BC,(LCNT)       ;REC  RET ADD
0EDE 1008              DJNZ PAGE10          ;DEC  LINE COUNTER
0EE0 DF                RST  NASSYS          ;WAIT FOR INPUT
0EE1 7B                DB   BLINK
0EE2 00                NOP
0EE3 CD200F            CALL HEAD
0EE6 0600              LD   B,LINES         ;RESET LINE COUNTER
0EE8 ED43550F  PAGE10  LD   (LCNT),BC       ;KEEP IT
0EEC C9                RET
0EEC           ;       ********************************
0EEC           ;       *         SUBROUTINE LINE  *
0EEC           ;       ********************************
0EED EF        LINE    RST  PRS
0EEE                   DS   VDUW-1
0F1C           LNEND   DS   1
0F1D 0D.               DB   CRET
0F1E 00                NOP
0F1F C9                RET
0F1F           ;       ********************************
0F1F           ;       *         SUBROUTINE HEAD  *
0F1F           ;       ********************************
0F20 3E0C      HEAD    LD   A,CLEAR
0F22 F7                RST  CRT             ;CLEAR THE SCREEN
0F23 00                NOP
0F24 00                NOP
0F25 21310F            LD   HL,MSG1         ;WRITE HEADINGS
0F28 11D30B            LD   DE,TOPLNE+9
```

```
0F2B 011A00          LD   BC,MSG2-MSG1
0F2E EDB0            LDIR                ;LOAD IT
0F30 C9              RET
0F31 3C204E41
0F35 53434F4D
0F39 2043524F
0F3D 53532052
0F41 45464552
0F45 454E4345
0F49 203E       MSG1 DB   '< NASCOM CROSS REFERENCE >'
0F4B            MSG2 EQU  $
0F4B 1027       NUM  DW   10000
0F4D E803            DW   1000
0F4F 6400            DW   100
0F51 0A00            DW   10
0F53 0100            DW   1
0F55            LCNT DS   2
0F57            LEN  DS   2
0F59            LZIND DS  1
0F5A            JUNK DS   2
0F5C            STACK DS  2
0F5E            WORK2 DS  1
0F5F            WORK1 DS  1
0F60            TYPE DS   1
0F61            LINENO DS 2
0F63 FFFF       EOBLK DW  #FFFF
```



```
ARRAY  0D24 BASIC  1000 BLINK  007B CLEAR  000C
CRET   000D CRLF   006A CRT    0030 CURPOS 0C29
ENC20  0EA3 ENCLF1 0E7D ENCLF2 0E84 ENCNXT 0EA5
ENCODE 0E70 ENCT10 0E96 ENDLST 0E63 EOBLK  0F63
GOTX20 0CEF GOTX30 0D14 HEAD   0F20 INS2   0D64
INS/   0D90 JUNK   0F5A LCNT   0F55 LD10   0D33
LEN    0F57 LINE   0EED LINENO 0F61 LINES  000E
LNE10  0DD4 LNE30  0E10 LNE40  0E19 LNE50  0E1D
LNEND  0F1C LOAD20 0D5A LOADIT 0D2C LOOPQU 0CC8
LOOPRM 0CBA LOOPX1 0CA7 LZIND  0F59 MSG1   0F31
MSG2   0F4B NASSYS 0018 NEXTX1 0CB1 NUM    0F4B
NXTLN2 0E5D NXTLNE 0E42 PAGE   0EDA PAGE10 0EE8
PRINT  0DBF PRS    0028 QUOTE  0022 REM    008E
SHIFT  0EAE SHIFTE 0ED4 STACK  0F5C STOP   005B
STR    0D1A TOPLNE 08CA TYPE   0F60 VDUW   002F
WORK1  0F5F WORK2  0F5E X10    0C9B X30    0CB5
X40    0CC4 X50    0CD9 X60    0D6A XEOB   0DA4
XEOB10 0D81 XREF   0CBA
```

```
0C80  2A D6 10 11 00 05 19 22  5C 0F CD 20 0F DD 21 FE  *......"\.. ..!.
0C90  10 2A FA 10 7C B5 20 03  DF 5B 00 2A 5C 0F 22 57  .*..|. ..[.*\."W
0CA0  0F 2A FC 10 22 61 0F DD  7E 00 FE 00 CA A4 0D 18  .*.."a..~.......
0CB0  04 DD 23 18 F2 FE 0E 20  0B AF DD 23 DD BE 00 20  ..#.... ...#... 
0CC0  F9 C3 A4 0D FE 22 20 11  DD 23 DD 7E 00 FE 00 CA  .....".  ..#.~.....
0CD0  A4 0D FE 22 28 DB C3 C8  0C FE 41 38 D4 FE 5B 30  ...."(.....A8..[0
0CE0  D0 32 5F 0F 3E 01 32 60  0F 3E 20 21 5E 0F 77 DD  .2_.>.2`.> !^.w.
0CF0  23 DD 7E 00 FE 00 28 34  FE 5B 30 30 FE 20 28 EF  #.~...(4.[00. (.
0D00  FE 24 28 16 FE 28 28 1C  FE 30 38 20 FE 3A 38 04  .$(..((..08 .:8.
0D10  FE 41 38 18 77 21 5A 0F  18 D5 3A 60 0F CB D7 32  .A8.w!Z...:`...2
0D20  60 0F 18 CB 3A 60 0F CB  CF 32 60 0F 2A 5C 0F ED  `...:`...2`.*\..
0D30  5B 5E 0F B7 ED 4B 57 0F  ED 42 09 28 53 E5 FD E1  [^...KW..B.(S...
0D40  FD 46 01 FD 4E 00 EB B7  ED 42 09 EB 38 42 20 0A  .F..N....B..8B .
0D50  3A 60 0F FD BE 02 38 38  20 0A 3E FF ED B1 BE 20  :`....00 .>.... 
0D60  F9 23 18 CF ED 5B 61 0F  3E FF ED B1 BE 20 FB 2B  .#...[a.>.... .+
0D70  E5 FD E1 FD 46 FF FD 4E  FE EB B7 ED 42 09 EB CA  ....F..N....B...
0D80  A7 0C 3E 02 CD AE 0E FD  73 00 FD 72 01 C3 A7 0C  ..>.....s..r....
0D90  3E 07 CD AE 0E E5 D9 D1  21 5E 0F 01 07 00 ED B0  >.......!^......
0DA0  D9 C3 A7 0C 01 05 00 DD  BE 01 20 05 DD BE 02 28  .......... ....(
0DB0  0E DD 09 DD 66 FF DD 6E  FE 22 61 0F C3 A7 0C CD  ....f..n."a.....
0DC0  20 0F DD 2A 5C 0F 3E FF  DD 77 FF DD 77 FE 06 0E   ..*\.>..w..w....
0DD0  ED 43 55 0F 3E 20 21 EE  0E 11 EF 0E 01 2F 00 77  .CU.> !......./.w
0DE0  ED B0 FD 21 EE 0E 3E FF  DD BE FF 20 2C DD BE FE  ...!..>.... ,...
0DF0  20 27 DD 7E 01 FD 77 00  DD 7E 00 FD 77 01 DD 23   '.~..w..~..w..#
0E00  DD 23 DD 7E 00 DD 23 CB  57 28 05 0E 24 FD 71 02  .#.~..#.W(..$.q.
0E10  CB 4F 28 05 0E 28 FD 71  03 FD 21 F4 0E DD 66 01  .O(..(.q..!...f.
0E20  DD 6E 00 DD 23 DD 23 23  7C B5 28 16 2B CD 70 0E  .n..#.#|.(.+.p.
0E30  FD 23 FD E5 E1 01 1C 0F  B7 ED 42 38 E0 CD ED 0E  .#....B8........
0E40  18 1B DD E5 E1 ED 4B 57  0F B7 ED 42 28 15 FD E5  ......KW...B(....
0E50  E1 01 F4 0E B7 ED 42 CA  D4 0D CD ED 0E CD DA 0E  ......B.........
0E60  C3 D4 0D CD ED 0E DF 7B  3E 0C F7 00 00 DF 5B 00  .......{>.....[.
0E70  DD E5 DD 21 4B 0F 3E 01  32 59 0F 06 05 AF DD 56  ...!K.>.2Y.....V
0E80  01 DD 5E 00 3C ED 52 30  FB 19 3D FE 00 20 07 3A  ..^.<.R0..=.. .:
0E90  59 0F FE 00 20 0D C6 30  FD 77 00 FD 23 AF 32 59  Y... ..0.w..#.2Y
0EA0  0F 18 02 FD 23 DD 23 DD  23 10 D2 DD E1 C9 E5 DY  ....#.#.#.......
0EB0  D1 06 00 4F 2A 57 0F B7  ED 52 19 28 17 09 E5 2A  ...O*W...R.(....*
0EC0  57 0F B7 ED 52 23 E5 C1  D1 2A 57 0F ED 53 57 0F  W...R#...*W..SW.
0ED0  ED B0 D9 C9 09 22 57 0F  D9 C9 ED 4B 55 0F 10 00  ......"W.....KU...
0EE0  DF 7B 00 CD 20 0F 06 0E  ED 43 55 0F C9 EF 00 00  .... ....CU.....
0EF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0F00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0F10  00 00 00 00 00 00 00 00  00 00 00 00 00 0D 00 C9  ................
0F20  3E 0C F7 00 00 21 31 0F  11 D3 0B 01 1A 00 ED B0  >....!1.........
0F30  C9 3C 20 4E 41 53 43 4F  4D 20 43 52 4F 53 53 20  .< NASCOM CROSS 
0F40  52 45 46 45 52 45 4E 43  45 20 3E 10 27 E8 03 64  REFERENCE >.'...d
0F50  00 0A 00 01 00 00 00 00  00 00 00 00 00 00 00 00  ................
0F60  00 00 00 FF FF 00 00 00  00 00 00 00 00 00 00 00  ................
0F70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

AUNT ALICE'S AGONY COLUMN

D. Parkinson

Is there anyone out there? (Dr Dark believes you are all a figment of the circulation manager's imagination. [Ed. - Circulation manager - who's he?].) I know there are at least five of you, I've got your letters here courtesy of our esteemed Editor. The following few pages attempt to answer some of the questions asked in these letters, and are based on the premise that if one person has actually taken the trouble to write in with a question, there must be at least a thousand of you out there with the same question on your lips. So, as these pages are to be fueled by your questions/comments/hints/tips, send them in - even if it is only one sentence on the back of an old envelope. Do not expect a personal reply. (Letter writing is not my forte). You may get a reply via 80-BUS News in due course. If a few people ask the same question there will almost certainly be a reply via the News (unless there are obvious signs of collusion like the same handwriting!).

This issue I start with a hardware bug in GM812 (the Gemini IVC) that was discovered sometime ago by Richard Beal, and has also been found by Nils Nazoa-Ruiz. It will only manifest itself if you are using interrupts, and only then in special circumstances. If you happen to be unlucky you will find that the IVC is reset everytime an interrupt occurs. The explanation for this lies in the way the IO addresses are decoded on the card. This is done via a PROM (IC25) which is enabled by IORQ. With the standard PROM the four outputs are used to signify addresses 00-07 (NASIO), address B1 (data port), address B2 (control port), address B3 (card reset). The B1 and B2 outputs are qualified by RD and WR before they are used, but the B3 output is taken straight to the RESET input of the IVC's Z80. If you look up the interrupt behaviour of the Z80 in its data sheet you will see that the Z80 acknowledges an interrupt by putting out an /IORQ signal along with an /M1, a unique signal combination that is used at no other time. Unfortunately if the lower 8-bits of the address bus just happen to hold the address B3, then the IVC is reset.

The cure is simple (in theory), and that is to disconnect pin 14 of IC25 from GND, and connect the backplane /M1 signal to it via a spare inverter on the board. Pin 14 is an additional enable input (active low), and with this modification the decode PROM will be inhibited during any cycle where M1 is low, thus preventing the interrupt acknowledge cycle being misinterpreted by the IVC. The normal IO operations will continue to work as before.

Next a letter from Alan Mackay - here is someone who is actually using his computer to compute. I reproduce his letter below in the hope that someone out there can help.

"The most useful of the features of BASIC as originally developed at Dartmouth College was the provision of matrix operations; multiplication, transposition and inversion being each accomplished by single statements. Most implementations for microcomputers do not provide these essential routines. Can anyone tell me how I can do matrix inversion on a Nascom 2? Does anyone have assembly language programs for matrix operations in BASIC or PASCAL? More generally, is there a library of serious mathematical subroutines - such as the NAG (Numerical Algorithms Group) provides for Algol and Fortran on large computers - for the Nascom 2? Is there anyone doing serious calculations, as opposed to computer games, on a micro?"

- replies to Alan Mackay, 22 Lanchester Road, Highgate, London N6 4TA. I recall a product MAP14Z that was available in 1980/1 from Enertech Ltd, 32 Gildridge Road, Eastbourne, East Sussex BN21 4SH. This was a 40-bit floating point package available for the Nascom. It did not support matrix operations, and a review of it appeared in INMC80 issue 2 (Sept80-Jan81).

Tony Chamberlain writes in with a list of the undocumented op-codes for the Z80. These have been covered before in various of the glossies and earlier INMC80s/80-BUS News, but if you're new to the Z80 and have missed them and want to know more send a note to me asking for it to be published in a later issue.

Now a request - "What is the best controller to purchase, first for a DOS, and later CP/M - MAPVFC, GM809, GM829, LUCAS LOGIC ?" My biased reply to that is GM829 - why? - it's the one I use and the only one I know about in any depth. (GM829 has superceded GM809, but you may be able to get 809s that have been traded in for 829s - check with your dealer).

Seriously all I can suggest is that you draw up a list of your current requirements, your future requirements, the capabilities of the boards, and then make a decision. All the boards mentioned meet the basic requirements of: a) They work. b) They are 80-BUS compatible. This just leaves the important facts of price/facilities/support to influence your choice. I list below some pointers for your thoughts:

Price: Can you afford it/is it worth it? Bear in mind that by the time you've added a DOS, the drive(s) themselves, power supply and connecting cables, a small saving in the price of the controller will become a very small saving in your total outlay. I'm not suggesting that you disregard the price, but in the long term the more important factors are facilities and support (below).

Facilities: What range of drives does it support? (3"/5"/8"/Winchester?). (Note that the new Sony 3" drive looks to the controller like an 8" drive in terms of data transfer rates, although the drive connector is different again being a 26-way one.) Will you ever want to use 8" and 5.25" drives simultaneously from the same controller? (GM829 offers software selectable drive type). If you are likely to change between 8" and 5.25" drives sometime and are happy to change straps, does the board layout allow both drive connectors to be fitted, or do the connector fields overlap?

Support: This is a major factor that isn't always given the attention it deserves. What disk operating system do you want to use? (NAS-DOS, POLYDOS, QDOS, CP/M etc). How does this restrict your choice of controller? If you want to use CP/M eventually, whose implementation of CP/M do you want to use? (Your own/Gemini/Nascom/MAP80/Independent). NOTE quite a lot of the criticism of CP/M that appears in the computing press from time to time should be more rightly directed at the writer of the BIOS rather than Digital Research. A good BIOS with adequate error trapping and various extended features can make a remarkable difference to a user's view of CP/M. A 'plain vanilla' BIOS can leave a lot to be desired.

What hardware do you already own? This is fairly important as for example Gemini's CP/M systems are based upon Gemini's disk controller card and the Gemini IVC. There are versions that will run on Nascoms (with and without IVCs), but they provide no support for the Nascom AVC or the MAP low-cost video card. Similarly the Nascom CP/M system is based on the Nascom disk card and won't support the Gemini IVC. This is the burden you have to bear for owning such a flexible system. The possible hardware permutations are high, and no one software product is likely to support every hardware permutation. Each manufacturer is likely to support his own product(s), and, depending on how he sees the market, may extend his support to include various boards from other manufacturers that may be used in conjunction with his product.

(I gather an optimist rang up Gemini one day to ask if they did an implementation of Digital Research's CP/M to run on a Nascom2 + RAMB + Nascom AVC + Nascom disk card - not a Gemini product in sight!)

Another question in the same letter - "Can a 'virtual disk' work under DOS or must I have CP/M (MAP80)?". I don't believe any of the DOS's support virtual disk, but most CP/M implementations do. The Gemini CP/M currently supports any 'page-mode' RAM (e.g. Nascom RAM B, GM802, MAP256k) for virtual disk, and also the forthcoming GM833 RAM-DISK (see below). For full support of a multiple MAP card system you have to go to MAP or use Richard Beal's SYS. For those who can't get enough of it Mr R.J.Drew (4 Brackenrigg, Armathwaite, Carlisle) has a cut-and-strap method of modifying the MAP RAM cards to provide a total of 4Mbytes for a virtual disk. (This assumes you've got the space/amps/money available for the 16 boards). With Gemini's GM833 you can reach a total of 8Mbytes without any cut-and-strap, but you'll still need the space/amps/money for 16 boards.

I see from another letter that someone out there has been spreading false rumours about CP/M block sizes and disk sizes: CP/M (in version 2.2) supports logical drives up to 8Mbytes in size. CP/M can allocate the disk space in block sizes of 1k, 2k, 4k, 8k, or 16k. The one restriction is that if a block size of 1k is used, then the disk capacity cannot exceed 256 blocks (256k). All the other block sizes can be used with any disk capacity up to the limit of 8Mbytes. One point to be aware of is that CP/M has a little bit of intelligence built in. If the total disk capacity is less than 256 blocks, each directory entry, (which records where each file is on the disk), holds the block numbers as 8-bit quantities. If the capacity is greater than 256 blocks, then each block number is stored as a 16-bit quantity. (i.e. If the capacity of the disk is less than 256 blocks, each directory entry can hold up to sixteen 8-bit pointers, but if the capacity exceeds 256 blocks each entry only holds up to eight 16-bit pointers.)

This is the reason why a block size of 4k was chosen for the Gemini Galaxy & GM825. As these use Quad density drives (i.e. Double density recording and double track density - 96tpi), the option of offering both single-sided (400k formatted) and double-sided (800k formatted) drives was open. If a block size of 2k had been used then the directory formats of single-sided disks (400k capacity - 200 2k blocks), and double-sided disks (800k capacity - 400 2k blocks) would have been incompatible. With a block size of 4k the 'block capacity' of both single and double sided drives are less than 256, thus ensuring that CP/M uses an identical method of recording block numbers in the disk directories. (A single-sided disk can be read/written in a double-sided system, and a less-than-half-full double-sided disk can be read/written in a single-sided system).

PRODUCT PREVIEW TIME

The Gemini GM833 is an 8x8 80-BUS card containing 512kbytes (or 0.5Mbytes, if you prefer!) of dynamic RAM. This RAM cannot be used as conventional system memory, but is configured to appear as a disk. It is accessed via three I/O ports which can be regarded as "track", "sector" and "data". To access the memory all that has to be done is to output "track" and "sector" addresses, and then read or write 128 bytes of data to/from the data port (perhaps using INIR or OTIR in a pseudo DMA transfer).

The software interface matches very nicely to CP/M, and is so simple that an application program in a non-disk based system could drive it directly. Unlike the "page mode" scheme, or the extended addressing modes of GM813 or the MAP256 card, no complex memory management is involved as the RAM-DISK is completely independent of the normal system memory.

CALL FOR ARTICLES

Reverting to the earlier theme, to help me answer yet another question perhaps someone out there (a Dealer even?) would like to produce a summary of the various DOSs & CP/Ms available for 80BUS cards? The obvious things to cover are the hardware they require to run. The extra hardware they can support (e.g. IVCs/AVCs/VFCs/PLUTOs/CLIMAXes/etc). The standard Nascom/Gemini software that can/can't be used under it. How easy it is to customise (can you add a printer?).

An alternative might be for YOU to send in a summary for the DOS you use. A user's view is perhaps better, as a dealer, though he sees all the products, may not have the time/inclination to use everything/anything he sells, and so may miss some of the finer points of the DOSs. Your replies can then be collated, compared, checked and published. This, (if it ever appears!), will no doubt be followed by a deluge of your articles on how to modify X to get it to run under Y.

CALL FOR QUESTIONS

As I said at the start, these pages will be fueled by your letters - so send them in even if it is only one sentence on the back of a postcard. If you want any of the above replies expanded on say so. (More on CP/M directory formats?). If I can't answer your questions I'll try and find someone who can.

Doctor Dark's Diary — Episode 16.

Two "Pascal" compilers slug it out! Is it Wirth reading this?

Regular readers will know that I like Pascal, and that I tend to go on at length about the virtues I think I see in the compiler I usually use. I must have hinted really well that I would like free things to review a while ago, because the postman brought me a nice shiny new compiler to play with, and I am sure I didn't pay for it! This was a very nice surprise, as you can imagine, and I rapidly set about attempting to use it. It seems like a good idea to compare and contrast the two Pascal systems I now have, so that, if you are thinking about changing from BASIC, you can get some sort of idea which of the two would be better for your purposes. Obviously, I am unable to comment on other systems I have not yet seen, and this does not imply any criticism of these others... Serves them right for not sending me freebies though!

The one I already owned is the Hisoft Pascal 4 compiler, sold for £40. The new one is the Poly-Data Compas system, which costs £120, plus VAT. To save a little space, I intend to refer to them as either Hisoft or Compas throughout. If you buy the Hisoft, you get a floppy disc and about fifty five pages of manual, loose pages, that is. Compas also comes on a floppy disc, but has a very smart blue folder which contains, at a guess, a hundred pages. Both of them run under CP/M, so don't get either if you are running an unexpanded Nascom 1.

Hands up all those who wondered why I put quotation marks round the word "Pascal" in the heading. Why? Well neither of these compilers is a full implementation of standard Pascal, and both of them have extras that are not standard. I don't think there is a full micro Pascal about yet, but am willing to accept one free if I'm wrong! What you get are subsets of the language, with additions that the compiler writer thinks you will want. Hisoft only allows FILEs of CHAR, like CP/M's ASCII files. It does not cater for VARIANTs in RECORD types. Compas doesn't have SET, POINTER, or CHAR types. Neither of them will let you pass a procedure or a function as a parameter to an other procedure or function, but I have never particularly wanted to do this latter thing! In fact, I have never met anyone who could tell me in nice short words why anyone would want to complicate a nice simple language by doing such a bizarre thing. I dare say it is all the rage in university computing departments... The SET types are something quite useful, which is not the impression given in many books about Pascal, where examples often feature lists of cheeses. POINTERs are very useful, and their absence from Compas indicates that dynamic data structures are also missing. So if you want to use linked lists, and build them up dynamically, you have to go about it in a less easy way than would be the case with Hisoft. I will describe the various extras facilities the two compilers provide later on in the article.

Hisoft only provide a compiler, which requires ASCII text files as its input, and it is up to you what you use to edit them. Any reasonable word processor will do the job, and only a masochist would try to use ED.COM for the job, so there is a hidden cost. Compas is a more complex system, designed to make program development faster. It has a built in editor, and can stay in memory along with the source program and the compiled result [assuming there is room]. So you can edit your program, compile it, and then run it, all without returning to CP/M. Learning to use a new editor is always a problem, and at first it always seems as though the designer has gone out of his way to choose really unlikely keys to give the editor its commands. The Compas editor still seems very strange to me, as it has two modes of operation, and I always seem to have it in the wrong one. The other problem is that the list of editor commands in the manual is separate from the list of which keys you have to press. This is presumably to make it easier for Poly-Data to change the manual to suit machines with different keyboards. I think it would have been a more friendly idea to put the two sections together, and write a separate manual for each machine the program is to be used on. To be fair, this would not be a problem once one became used to using the editor, but at first it is infuriating.

Both manuals give formal definitions of the syntax of the Pascal the compilers will accept. Hisoft use those syntax diagrams with one way arrows to show you what can go where, while Compas have done a very careful job of writing out a full Backus Naur definition of the language. In case you are not familiar with BNF, it tends to look like this:-

<program> ::= <program heading> <block> .
<program heading> ::= <empty> ! PROGRAM <file and so on and so on...

I find syntax diagrams a million times easier to read, even though they are not as pleasing to the mathematical mind, and I believe they are a better way of doing things. You may disagree, if you are fond of mathematical formalism for its own sake, rather than where it is useful. Both compilers come complete with example programs. The ones Hisoft give you are printed in the manual, so you have to type them in for yourself, if you want to use them. One of them, which is supposed to be able to read a CP/M disc directory into memory, doesn't work with my CP/M, although I am told it does with others. Compas's examples are already on the disc, so all you have to do is compile them and run them. They are very nice examples, too. One of them is a calculator program, which makes your expensive Nascom [or similar] work just like a £5 pocket calculator. Another is a quite useful address and telephone number file handling program, and there is also one that can do hex dumps of files from disc.

The precision of real number calculations is not the same in both systems. Hisoft gives you seven significant figures, while Compas gives you eleven. Of course, if you need even more precise figures than that, you can program your way round the restriction with either compiler, as I mentioned in a previous article. They both use two bytes to store an integer, which gives you the usual -32768 to 32767 range of values. Compas doesn't tell you if an integer variable has overflowed, whereas Hisoft does, unless you set a compiler option to omit overflow checks. This would make the program run faster, but I usually forget to do it, and I have no complaints about the speed Hisoft programs run at! With Compas, you have to set a compiler option if you want to use recursion. I used to think using recursion was just showing off, until I needed it once or twice.

Compas has a data type that is not used in standard Pascal, or Hisoft for that matter, and it is the string. Normally, in Pascal, one uses arrays of characters, and writes routines to manipulate them. It can be done, although I never finished the set of routines I was working on a while ago. The lack of strings in Pascal is a real downer. Compas strings are very much like BASIC strings, although the way it chops strings up [equivalent to left$, right$, and mid$] uses a notation like that of the peculiar BASIC in the Sinclair ZX81. Still, it is there, and strings can be cut up, concatenated and generally messed around as much as you like.

Extensions to Pascal.

Both compilers give you direct access to the contents of memory, Compas does it by means of what they call "the memory array", which behaves much like an ordinary Pascal array, whose contents are the Z80's address space. Hisoft have added the words PEEK and POKE to the language, but these are not the single byte operations users of non structured BASIC know and love. If you poke an address with a data type that takes more than one byte, all the bytes of the source item are poked into successive memory locations. In other words, POKE(£F80A,'Heading') would put the word "Heading" on the top line of the screen. This is very useful!

The Z80 ports can also be accessed using either compiler. Hisoft have the reserved words IN and OUT, while Compas uses another of its special arrays which is referred to as the port array.

Both compilers provide a way to include machine code routines in programs, and call them. This means that they both admit that carefully written machine code does even faster than their standard run time routines, I suppose. There is no shame in that. Still, given the program speed that either compiler gives you, machine code routines are only rarely necessary, when you are using the computer to control some tricky peripheral, for example.

Both compilers have random number routines, but the Hisoft one is primitive, as it just returns the contents of the Z80 refresh register, which often causes programs that use it to be far less random than the programmer wanted them to be. Compas has a function "random(i)" which returns an integer r in the range 0<=r<=i. This is very useful in game programming!

Another of the things Pascal could have done with is a little extension to the CASE statement, so that you could specify what was to happen if the controlling variable didn't match any of the items in the CASE statement. Surprise! Compas has OTHERS, and Hisoft has ELSE. These mean the same as each other:-

```
CASE i OF                          CASE i OF
  1 : WRITE('one');                  1 : WRITE('one');
  2 : WRITE('two');                  2 : WRITE('two');
  OTHERS: WRITE('lots')            ELSE WRITE('lots')
END;
```

Standard Pascal only knows about sequential files, Compas has random access files too. Hisoft have been promising to add these, but have not yet done it, or have forgotten to send me the new version!

In the event that you manage to write a program that is too big for your machine, what can you do? If you have Compas, you can chain programs, which is handy. You can do it with Hisoft programs as well, IF you know enough about using CP/M.

Selling your programs.

If you write it with Compas, you are forced by the licence you have signed to pay Polydata a percentage of the money you get for your program. This is supposedly because the program will contain Polydata's run time routines. In fact, they would be of no use to the buyer on their own, with no way of knowing how to call them. People who sell software with conditions like this attached should be forced to say so in their advertising. Given the price of Compas, I think they ought to be happy with what they have, and not try to make even more money from the work of others. Hisoft do NOT demand money in this way.

Conclusions.

The suitability or otherwise of either of the compilers for your purposes will depend very much on what those purposes are. If you want to write large suites of business programs, and do it without having to write your own routines to do fancy screen handling, string handling and high precision calculations for you, then it will have to be Compas. You will also have all the bother of paying them again if you sell your work. If, on the other hand, you want to learn Pascal, and write amazing games, fast control programs and generally stay close to "Proper" Pascal, then go for Hisoft. Then, when you sell 100,000 copies of your game to all the Spectrum owners (yes, Hisoft Pascal 4 is available for the Spectrum, so you can write games/etc for a huge market!) you can keep all the money, until the tax man finds out. My own preference? Frankly, it is the Hisoft product, and I am looking forward to the day they bring out Modula 2!

Random Rumours (& Truths?)

by S. Monger

I could hardly believe it when I was asked for another contribution so quickly. What could I find to write about so soon after my last load of rubbish? Well, I suppose a reasonable start would be talk about things that have NOT arrived.

Where are they now?

For example, the Sinclair Microdrive! Yes, it's only 15 months(ish) since it was first announced, and I only mention it here because I know of one certain individual who intends to try and interface one to their 80-BUS machine. Could be interesting, and somewhat cheaper than a Winchester. Which reminds me! (Note the subtle way of changing topic - very clever!) In a long, long lost price list from Nascom (or should that be Lucas Logic, or even Lucas/Nascom? - I get confused so easily) I seem to remember a line that said something like "5MB Winchester Disk Drive - price to be advised", and at about the same time there was a Nas-Net leaflet with "optional hard disk" mentioned - where has that got to? And how about Gemini, their last catalogue mentioned "GM818 Serial Board" - no sign of it yet. And nor is there any sign of IO Research's Pluto Pallette, Frame Grabber or IO-Net boards. What are these people all so busy doing that they can't produce these almost forgotten (but not by me!) 'products'?

What are they doing instead?

Lucas must be putting all their current effort into the Lucas LX range. What is that? Well, remember the 'Conspiracy' that I mentioned two issues ago - that's to do with the LX. The original idea was to take a Quantum 2000 case and put a Nascom 2, AVC, MAP RAM board and Gemini GM829 FDC/SASI board inside it. It was first shown at Which? in January, but I don't think that it has been sighted since. Interesting developments afoot?

And Gemini? Well they have launched one or two new things in recent months. There is the GM839 prototyping board at £12.50. Made of fibre-glass, and laid out for high density packing, it's ideal for that washing machine and central heating controller! Then there's their GM833 RAM-DISK board. (It seems they managed to resist the temptation to call it a Virtual Disk board, because of the possible abbreviation of the latter!) This board provides 512K bytes of RAM, port-mapped in such a way as to allow very easy interface software, using the otherwise little-used Z80 INIR and OTIR instructions. Apparently a DIL switch on board allows upto 8 Megabytes of the things. Fancy a dose? Then there's the Gemini Multi-Net (briefly described last issue) and a permutation on the Galaxy called "M-F-B". And what does that stand for? Multi-Format-BIOS of course. A Galaxy is supplied, fitted with (or surrounded by) drives of different TPI (tracks per inch), an optional Winchester, and an optional 8" drive. M-F-B links it all together, and allows easy transfer of software between almost all soft-sectored 5.25" and 8" formats. "Superbrain to Xerox? - No problem. IBM to Osborne? - One moment sir." They are obviously looking at Software houses as their market for the M-F-B, and I am sure that there is little point in printing a price here. (Actually, I don't know it, but I don't want to admit to that.)

And finally, IO Research. Well, to be quite honest I have heard very little about what they are upto. I understand that sales of PLUTO are pretty good, mainly to non-80-BUS computer owners, who also have to buy special interfaces. Why don't these people buy a REAL microcomputer, which this board would plug straight into? Some people have no idea!

Nascom & Gemini USERS

NEW 32K C.M.O.S. BATTERY BACKED RAM BOARD

from

MICROCODE (CONTROL) LIMITED

FEATURES:

EFFECTIVELY REPLACES EPROMS.
Does away with the inconvenience of EPROM programming and the compromise of assigning valuable address space to ROM.

ON BOARD RE-CHARGEABLE Ni-Cad BATTERY RETAINS MEMORY FOR OVER 1000 Hrs.
Battery is automatically charged during power-up periods.

HIGH SPEED OPERATION up to 6 MHz WITHOUT WAIT-STATES.

FULLY NASBUS[1] and GEMINI-80 BUS[2] COMPATIBLE.

PAGE MODE SCHEME SUPPORTED.
The board can be configured to provide one 32k byte page or two completely independent 16k byte pages.
Complete pages of 64k bytes are simply implemented by adding more boards on to the bus.

SOFTWARE and/or HARDWARE READ/WRITE PROTECTION.
4K blocks in either page are link selectable to be aligned on any 4K boundary.

FULLY BUFFERED ADDRESS, DATA AND CONTROL SIGNALS.

MEMORY I.C. SOCKETS ARE LINK SELECTABLE TO SUPPORT ANY 24 PIN 2k byte MEMORY I.C.s.
Thus the board can support up to 32k bytes of any mixture of cmos, nmos rams or 2716/2516 eproms.
All options are link selectable using wire links plugged into gold-plated socket pins, avoiding the risk of damage and the inconvenience caused by soldering links directly to the board.
The printed circuit board is manufactured to the high quality demanded by industrial users and conforms to B.S.9000.
The board comes assembled and tested and is supplied with a minimum of 2k bytes of low-power cmos ram. Fully documented.

AVAILABLE NOW!

PRICES:

Board with 32k bytes	£184.95
Board with 16k bytes	£149.95
Board with 2k bytes	£99.95
TC5517AP Very low power 2k cmos memory I.C.	£6.50

Please add £1.50 (P&P) & VAT @ 15%

[1]nasbus is a trademark of nascom microcomputers a division of LUCAS LOGIC
[2]trademark of GEMINI MICROCOMPUTERS LIMITED

Cheques/PO's made payable to:-

MICROCODE (CONTROL) LTD
40 WELL TERR., CLITHEROE,
LANCASHIRE, BB7 2AD
ENGLAND. 0200 27890

For all Microprocessor Applications

Consultancy, Design and Manufacturing Services to Industry

Call
MICROCODE (CONTROL) LTD
40 Well Terr., Clitheroe,
Lancashire, BB7 2AD
England. Tel. 0200 27890

NEW PRODUCT

MICROCODE (CONTROL) LTD.
40 WELL TERR., CLITHEROE,
LANCS., U.K. 0200 27890

"Aspirins are not the only solution to your backplane problems"

BP14C BACKPLANE

14 SLOT 80-BUS & NASBUS TERMINATED BACKPLANE

BP14C has been designed, using mainframe techniques, to solve microcomputer problems. As the speed of microcomputers increases, the demands on the bussing system become more and more critical. Without the use of a properly terminated backplane, system performance and reliability will inevitably suffer.

BP14C has been designed to overcome the three major backplane problems:

1. Noise generated by reflected signals.
2. Noise generated by crosstalk.
3. Noise generated and transmitted through backplane power rails.

These problems have been attacked on the BP14C by:

1. Terminating ALL active bus signals into a potential balanced R.C. filter.
2. Interlacing all active bus signals with ground 'shield' tracks.
3. Forming a complete ground plane on one side of the backplane and using very wide tracks for the other power rails.

The Backplane measures 14" by 8" and supports up to 14 slots. Smaller lengths can be obtained by a simple 'score and break' operation.

The backplane will fit neatly into a 19" rack with spare space available for a Power Supply Unit.

The backplane features proper implementation of both the interupt and the bus request daisy chains.

BP14C is another product designed in line with Microcode (Control) Limited's commitment to industrial quality 80-bus and NASBUS performance.

Price £47.00 plus £1.50 (Post and packing), plus VAT

Cheques/PO's made payable to:-

MICROCODE (CONTROL) LTD.
40 Well Terrace, Clitheroe,
Lancs., U.K. Tel. 0200 27890