nascom NEWSLETTER

Volume 3 Numbers 5 & 6
June 1984

Lucas Nascom

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

We currently have copies of all the back issues of the Nascom Newsletter and Micropower magazines.

We are now in a position to sell these off at rock bottom prices to clear our shelves.

Volume 1 (Issues 1 to 4)

1 magazine 25p
5 magazines for £1

Volume 2 (Issues 1 to 6)

1 magazine 25p
5 magazines for £1

Our stocks are limited so order now for those missing issues.

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

Back Issues

NASCOM NEWSLETTER Vol 3, No 5 & 6 June, 1984

Contents

Editor - Ian J Clemmett

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

Printed by - Dataform Press

Editorial

Late again, but I suppose that you're probably used to that by now. Sadly, though, this will be the last issue of the Newsletter. Due to a number of reasons including the dying Nascom and the diminishing number of advertisers, I have been forced to combine Volume 3, Nos 5 and 6 into one (thus the almost twice as thick as usual edition).

This combined issue has a number of very interesting articles (as have all the other issues of the magazine come to that). The final part of SysEx is included, there is a very good article on using the AVC memory for program and data storage and then a couple of hardware projects for an EPROM programmer and an emulator.

I haven't got very much news myself. I have been using Nassembler extensively over the last few months and can highly recommend it. If any one has seen or knows where I can buy the 6502 and 68000 packages I would be most grateful if you could let me know. Every where that I have contacted so far has either denied all knowledge of them or said that they were having supply problems.

After a discussion with Microcode, it has been decided that the competition that we jointly sponsored last year will be declared abandonned due to a lack of entries. My apologies to the people that did send in their designs.

Due to demand, we can now offer our back issues at even lower prices (25p each for issues in Volume 1 & 2, or 5 for £1.00) and we still have copies of all Volume 3 (still at £1.25 each I am afraid). See the back issues advert elsewhere in the magazine. For any of you that don't have a full set of the Micropower/Nascom Newsletter, this is probably your last chance (and certainly your cheapest). Order now, while stocks last (or whatever that advertising cliche is).

From the correspondence that I'm currently getting, it seems that the Nascom is being used for control and experimental applications more and more but the number of hobbyist users is dropping off rapidly. The natural life-cycle of a microcomputer I suppose.

Well, I think that just about wraps it up for this, the final editorial of the Micropower/Nascom Newsletter. My thanks go to every one who has contributed in the past and to all the Nascom-associated companies that have supported us with advertising and product reviews. Best wishes to all the people who still know that the Nascom is the best and goodbye.

IJC

SYS-EX A 1K extension program for NAS-SYS monitors. Part 5

by David G. Johnson

This is the final part of the series of articles on the 1K monitor extension to NAS-SYS. All of the major facilities of SYS-EX have been described in the earlier magazine issues. All that now remains is to tidy up a few remaining details and to print the promised hex dump of the monitor extension itself.

SYS-EX is written in relocatable code and will operate correctly at any memory location above 1000H. However, the Nascom memory map allocates locations B000H to B7FFH as the recommended memory locations for extensions to monitor facilities. SYS-EX may conveniently occupy the first half (B000H to B3FFH) of this available space and at the same time provide facilities for the easy use of further extensions from B400H onwards.

Should SYS-EX be located at B000H, two useful and important facilities are available.

1. The hardware reset jump may be set to cold start SYS-EX automatically at switch on and on pressing the RESET button.

2. NAS-SYS 3 provides the "Y" command which jumps to address B000H. The "Y" command can be used directly to cold start SYS-EX.

Cold starting SYS-EX

A cold start is required if the monitor has not already been initialised. e.g. at switch on or after a program has made non standard use of the memory reserved for NAS-SYS workspace.

The execution address for cold starting SYS-EX is 5 bytes on from the start of SYS-EX. e.g. If SYS-EX resides between 1000H and 13FFH, it may be cold started by the NAS-SYS command:

```
E 1005
```

If, and only if, SYS-EX resides between B000H and B3FFH, it can also be cold started at B000H.

Warm starting SYS-EX

A warm start is used when the initialisation of the NAS-SYS workspace and the clearing of the screen memory are not required.

SYS-EX is warm started by executing the program eleven (000BH) bytes on from the start. e.g. If it is installed between 1000H and 13FFH, SYS-EX is warm started with the NAS-SYS command:

```
E 100B
```

A warm start is a useful alternative to the NAS-SYS MRET instruction when ending an assembler program. e.g.

```
0000  C3 0B B0 JP  SYSX
```

Adding further monitor extensions

There are five unused keyboard commands (g j m o p) all of which cause a routine at 0400H beyond the start of SYS-EX to be called. Also, the two called routines USCR1 and USCR2 call a routine at 0400H beyond the start of SYS-EX. If suitable code is located at this address, the commands "g", "j", "m", "o", "p", USCR1 and USCR2 may be used to call a further seven commands, five of which are accessible directly from the keyboard.

As SYS-EX is written in relocatable code, it is possible to copy SYS-EX itself into any suitable location in RAM and then to provide the extension commands in following RAM locations.

On entry to the extension code located beyond SYS-EX, the NAS-SYS stack is in use. Care should therefore be taken to ensure that:

1. higher addresses on the stack are not corrupted
2. the maximum depth of the stack is not exceeded
3. the user code pops the same number of bytes off the stack as it pushes on to it.

A return to SYS-EX may be made by executing a Z80 RET instruction. If the Carry flag is set upon return to SYS-EX, the message "Error" is displayed on the screen prior to the acceptance of further input.

Upon entry to the extension code, various Z80 registers have preset values. These values are as follows.

REGISTER(S)	CONTENTS
A	routine number or ASCII code
HL	value from ARG1
DE	value from ARG2
BC	value from ARG3
SP	0C5FH within the NAS-SYS stack

Fortytwo bytes are available on the stack for use by the extension code. Any calls to NAS-SYS or SYS-EX routines from within the extension code will require a number of stack levels to be available.

Suggested routine to extend SYS-EX at offset 0400H

```
0400  FE 6A          CP   "j
0402  DA ** **       JP   C,gROUTINE
0405  CA ** **       JP   Z,jROUTINE
0408  FE 6F          CP   "o
040A  DA ** **       JP   C,mROUTINE
040D  CA ** **       JP   Z,oROUTINE
0410  FE 82          CP 82H
0412  DA ** **       JP   C,pROUTINE
0415  CA ** **       JP   Z,USCR1
0418  C3 ** **       JP   USCR2
```

Compatibility with NAS-SYS monitors

Compatibility with both NAS-SYS 1 and NAS-SYS 3 monitors is achieved (with one exception) by calling the NAS-SYS routines in the NAS-SYS recommended manner.

The one exception is that SYS-EX requires access to the absolute address of the NAS-SYS INLS routine, which is different in the two versions of NAS-SYS. This is achieved by locating the instruction in the main monitor loop PARSE (by using a variant of the SYS-EX FCEP1 routine) which follows the NAS-SYS call to INLS in both versions of the monitor. The address of INLS is taken from the two bytes prior to this common instruction.

User entry point summary

```
ADDRESS (start      ENTRY POINT
of SYS-EX plus)

0000H . . . . . .   Cold start when installed at B000H
                         only
0003H . . . . . .   Called routine entry point
0005H . . . . . .   Cold start
000BH . . . . . .   Warm start
01E0H . . . . . .   BASIC named program file entry point
                         cold start when installed at B000H
only
0003H . . . . . .   Called routine entry point
0005H . . . . . .   Cold start
000BH . . . . . .   Warm start
01E0H . . . . . .   BASIC named program file entry point
```

```
T1000 1400 1 C 001
  1000 C3 05 B0 18 69 31 00 10 CD 0D 00 31 61 0C EF 18 3D 53 59 53
  1014 2D 45 58 3D 20 0D 00 2A 23 0C 7C B5 28 04 3A 25 0C 77 21 00
  1028 10 22 6B 0C 65 D7 64 01 2B 0C 00 2B 56 2B 5E EB D7 0D 1A FE
  103C 20 28 E7 FE 41 30 05 DF 6B 18 DF E9 FE 7B 30 F7 FE 64 28 07
  1050 D5 13 DF 79 D1 38 EC 1A 32 2B 0C FE 5B 30 09 DF 60 32 0A 0C
  1064 DF 5C 18 BE D7 0B 38 D7 18 B8 E3 7E 23 E3 32 2B 0C D7 00 E1
  1078 FE 61 00 D8 E5 16 00 5F 19 19 00 5E 23 56 E1 19 11 89 FF 19
  108C E5 DF 60 3A 2B 0C C9 E3 54 5D 7E 23 B7 20 FB E3 0E 85 D5 E5
  10A0 CB 51 28 3E 1A 13 B7 28 52 FE 22 28 4A CB 81 CB 71 20 4A BE
  10B4 28 63 CB 49 20 10 23 F5 7C B5 28 05 F1 28 F0 18 DB CB F1 F1
  10C8 18 D6 E1 D1 23 D5 E5 CB 89 18 E8 DF 63 01 01 09 1A FE 22 20
  10DC C1 13 CB D1 18 BC E5 DF 64 E1 38 3B 3A 20 0C FE 03 30 34 B7
  10F0 28 09 3A 21 0C 18 B6 CB 79 20 B2 00 00 C5 F1 E1 D1 F8 D8 20
  1104 09 EF 58 58 58 58 20 0D 00 C9 C5 DF 66 C1 05 20 B7 DF 6A B7
  1118 C9 CB 49 20 9D F1 E5 0C 0C 18 97 CB C1 18 D4 D6 6B 87 26 0C
  112C 6F 5E 7E 23 56 B2 37 C8 D5 DF 60 B7 C9 B7 02 BC 02 CF 03 33
  1140 03 B0 03 D3 00 00 04 14 03 A6 03 00 04 7F 01 8B 01 00 04 FC
  1154 03 00 04 00 04 A7 02 14 02 78 02 E9 03 D9 03 14 02 AE 01 27
  1168 01 27 01 27 01 93 00 9C 00 CB 02 EB 02 22 03 26 03 5D 03 00
  117C 04 00 04 DF 78 DF 72 11 04 00 19 DF 72 B7 C9 D7 29 3C D6 1D
  1190 67 DF 5F DF 5D DF 5D 44 DF 77 3E D3 DF 6F 10 FC EB 41 DF 6D
  11A4 AF DF 6F EB DF 71 DF 5F B7 C9 D7 06 D7 DC DF 57 B7 C9 47 3E
  11B8 18 F7 78 F7 EF 4E 61 6D 65 7C 00 DF 63 21 05 00 19 3E 7C BE
  11CC 28 03 F1 37 C9 54 5D 13 01 2B 00 09 3E 20 2B BE C0 0D 18 FA
  11E0 31 61 0C D7 06 30 02 DF 6B DF 5A CD 8B E9 21 FC FF 19 D8 AF
  11F4 32 0B 0C 2C 21 D6 10 22 0C 0C 2A D6 10 22 0E 0C 3E 77 28 A6
  1208 7B 87 87 C6 72 FE 7A 28 67 32 2B 0C D7 A0 DF 5F DF 78 DF 6A
  121C D7 29 30 FC D7 60 E5 21 BF FF 19 41 1B 23 13 10 0E E1 D7 2A
  1230 21 2B 0C 7E D6 20 77 DF 52 B7 C9 3E 7F BE 28 E9 1A BE 28 E5
  1244 E1 18 D5 11 1B D3 06 03 CF BA 28 03 BB 20 F7 10 F7 CF BB 20
  1258 07 F1 DF 72 DF 5F B7 C9 BA 28 02 37 C9 11 CA 0B 06 2A CF B7
  126C 20 04 E5 EB 18 2D 12 13 10 F4 B7 C9 DF 5F DF 78 D7 C9 38 FC
  1280 18 D8 E5 2A 29 0C 11 C0 FF 19 06 06 36 20 23 10 FB 54 5D 06
  1294 2A B7 28 07 77 23 CF 10 F8 E1 C9 36 20 23 10 FB E1 B7 C9 DF
  12A8 5F DF 78 D7 9A 30 FC 47 DF 6A 78 D7 CD 18 F4 21 0C 0C 1E 0A
  12BC 43 5E 23 56 23 EB DF 69 DF 66 EB 10 F4 18 57 2A 21 0C 3A 20
  12D0 0C FE 03 30 07 26 00 CB 7D 28 13 25 CB 7C 28 0E 2B 7C 2F 67
  12E4 7D 2F 6F 3E 2D 18 05 2A 21 0C 3E 20 F7 FE 20 3E 20 F5 11 0A
  12F8 00 01 FF FF 03 ED 52 30 FB 7B 85 C6 30 F5 60 69 7C B5 20 ED
  130C E1 7C F7 FE 20 20 F9 C9 D7 39 F7 EB DF 64 D7 AF CB 75 20 02
  1320 D7 C9 0E 0D 18 02 0E 00 06 30 3E 15 F7 10 FD A9 F7 B7 C9 D7
  1334 1A D7 26 D8 47 DF 69 F7 DF 66 F7 7C B7 28 07 B8 20 07 CB 7D
  1348 28 03 7D DF 68 18 D3 11 C0 FF 2A 29 0C 19 3E 20 23 BE 28 FC
  135C C9 AF 32 20 0C 47 4F C5 7E FE 2D 20 02 0D 23 3E DF A6 28 23
  1370 7E E3 FE 3A 30 1A D6 30 38 16 EB 21 20 0C 34 62 6B 06 09 19
  1384 38 0A 10 FB 50 5F 19 38 03 E3 18 DA E1 37 C9 E1 79 B7 28 09
  1398 50 58 EB ED 52 CB 7C 28 F0 22 21 0C B7 C9 79 D7 18 54 5D 13
  13AC ED B8 12 C9 D7 0F DF 69 F7 1A DF 68 DF 6A 62 6B 23 ED B0 B7
  13C0 C9 1B 1B D5 EB B7 ED 52 23 44 4D E1 D0 E1 C9 B7 1A ED A0 2B
  13D4 77 23 E0 18 F7 2D 7D FE 03 3F D8 87 21 1A 0C 85 6F 73 23 72
  13E8 C9 3A 0B 0C B7 20 03 DF 63 EB 11 CA 0B 01 30 00 ED B0 B7 C9
  13FC DF 5B 56 30 B1 21 16 14 09 09 4E 23 66 69 CD 15 14 3A 45 10
```

Incorporating ZEAP source files into WordEase III

by A. Doroszenko

The following program converts ZEAP source files into files compatible with WordEase III, making it simple to fully document ZEAP files with all the extra facilities offered by WordEase. A comparison of ZEAP and WordEase file structure shows that conversion from one to the other is fairly easy.

The first two bytes of the ZEAP edit buffer is an offset which when added to the start address of the edit buffer (at £0F00 in ZEAP workspace) points to one more than the end of the file. £00 is used as the end of line marker, and ZEAP line numbers are stored as two bytes which follow immediately after the previous line marker, e.g. 92 01 = line number 0192.

In WordEase the first two bytes of the file (at £1000) is the actual address of the end of file marker (£FF), and the ends of lines are marked by £A0. The ZEAP line numbers have to be converted into 4 byte ASCII strings, so that 92 01 converts to £30 £31 £39 £32 = "0192".

The copious program notes should be sufficient to show how the program works. However, a few further notes are required regarding my system. I have two 64K RAM boards (paged as 0 and 1) and an EPROM board (in page 0). ZEAP runs on page 0, but WordEase is transfered into RAM on page 1 by means of a control program. This explains the OUT commands in the TRANS routine. If your system just has one page remove lines 190-220 and 240-250, and modify BUFP (£0F00) to a higher address after cold starting ZEAP, so that the WordEase file has room below the ZEAP file. Remember though that the WordEase file requires more memory than the ZEAP file, i.e. 4 bytes extra for each format code, and two more bytes per line number.

If you have just one RAM page put the program in any convenient location. If your system is similar to mine then run the program at £0C80, flip to page 1 with the command O FF 22, and then COLD START WordEase. A cold start is necessary because the program as it stands overlaps WordEase's workspace. The WordEase file is unaffected by a cold start. Once in WordEase "Adjust" the text before doing anything else. To simplify the program a little, every time a "Comment" is encountered in the ZEAP file a £A0 marker is placed before it when transfered to the WordEase file. This was to prevent line overflow. If there are more than 47 characters between £A0 markers WordEase will lock up. Note that full line "Comments" should not exceed 42 characters in length.

```
0010        ORG £0CB0
0020        LD SP,£1000      ;STACK IN COMMON MEMORY
0030        LD HL,(£0F00)    ;BUFP IN ZEAP WORKSPACE
0040        INC HL           ;STEP OVER OFFSET
0050        INC HL
0060        LD BC,£1002      ;BOTTOM OF WORDEASE EDIT BUFFER
0070        LD A,£FF         ;PUT £FF AT £1002
0080        CALL TRANS
0090        LD A,£A0         ;PUT £A0 AT £1003
```

```
0100          CALL TRANS
0110          LD IX,COMENT   ;COMMENT FLAG
```

This is the main program loop.

```
0120 LOOP     LD A,(HL)      ;GET THE ZEAP BYTE
0130          CP £00         ;IS IT AN END OF LINE MARKER?
0140          JR Z,ENDLIN    ;JUMP IF YES
0150          CP £3B         ;IS IT A COMMENT DELIMITER?
0160          CALL Z,COMLIN  ;CALL IF YES
0170          CALL TRANS     ;OUTPUT THE CHARACTER TO WORDEASE FILE
0180          JR LOOP
```

This routine transfers a byte in A to the WordEase file. Current WordEase address is in BC.

```
0190 TRANS    PUSH AF        ;SAVE THE CHARACTER
0200          LD A,£22       ;FLIP TO PAGE 1
0210          OUT (£FF),A
0220          POP AF         ;RECOVER CHARACTER
0230          LD (BC),A      ;PUT IT INTO WORDEASE FILE
0240          LD A,£11       ;FLIP BACK TO PAGE 0
0250          OUT (£FF),A
0260          INC HL         ;NEXT ZEAP ADDRESS
0270          INC BC         ;NEXT WORDEASE ADDRESS
0280          RET
```

Deal with end of line marker, and output "^N1".

```
0290 ENDLIN   RES 0,(IX)     ;RESET COMMENT FLAG
0300          INC HL         ;IS NEXT BYTE THE
0310          LD A,(HL)      ; END OF FILE MARKER?
0320          CP £FF
0330          JR Z,END       ;JUMP IF END OF FILE
0340          DEC HL         ;RESTORE POINTER IF NOT
0350          LD DE,NEWLIN   ;PUT"^N1 (£A0)" AT END OF LINE
0360          LD A,£5
0370          CALL CODE
0380          CALL LINNUM    ;DEAL WITH LINE NUMBER
0390          JR LOOP
```

This routine converts a two byte ZEAP line number into a four byte ASCII string.

```
0400 LINNUM   INC HL         ;SECOND BYTE FIRST
0410          CALL GETNO     ;CONVERT BYTE INTO 2 ASCII CHARS.
0420          DEC HL         ;NOW DEAL WITH FIRST BYTE
0430          CALL GETNO     ;CONVERT BYTE
0440          INC HL         ;STEP OVER ZEAP LINE NUMBER
0450          INC HL
```

Check for a label or a full line "Comment"

```
0460          LD A,(HL)      ;CHECK BYTE FOLLOWING LINE NUMBER
0470          CP £20         ;IF IT IS A SPACE THERE IS NO LABEL
0480          JR Z,T30       ; OR COMMENT LINE
```

Labels and "Comments" both tabbed "^T23"

```
0490          LD DE,TAB23
```

```
0500        DEC HL        ;DEC, OR FIRST CHARACTER IS LOST
0510        CALL FORMCD   ;OUTPUT THE FORMAT CODE
```

Check for a full line "Comment"

```
0520        LD A,(HL)     ;GET NEXT CHARACTER
0530        CP £3B        ;IS IT A COMMENT DELIMITER?
0540        JR NZ,GETCHR  ;JUMP IF NOT A COMMENT
0550        CALL TRANS    ;OUTPUT THE ";"
0560        SET 0,(IX)    ;SET COMMENT FLAG
0570        RET
```

The next routine outputs a label; exits when a space is found.

```
0580 GETCHR LD A,(HL)     ;GET THE LABEL CHARACTER
0590        CP £20        ;IS IT A SPACE?
0600        JR Z,T30      ;EXIT IF YES
0610        CALL TRANS    ;OTHERWISE OUTPUT THE CHARACTER
0620        JR GETCHR
```

Opcodes tabbed "^T30"

```
0630 T30    LD DE,TAB30
0640 FORMCD LD A,£4       ;FOUR BYTES IN THIS FORMAT CODE
```

Output the format code.

```
0650 CODE   PUSH AF       ;SAVE NO. OF BYTES TO OUTPUT
0660        LD A,(DE)     ;GET FORMAT CODE
0670        CALL TRANS    ;SEND TO WORDEASE FILE
0680        DEC HL        ;CURRENT ZEAP ADDRESS MUST NOT CHANGE
0690        INC DE        ;NEXT WORDEASE ADDRESS
0700        POP AF        ;NO. OF BYTES TO OUTPUT
0710        DEC A         ;NOW ONE LESS
0720        CP £00        ;ANY MORE IN THIS FORMAT CODE?
0730 .      JR NZ,CODE    ;JUMP BACK IF MORE
0740        INC HL        ;NEXT ZEAP ADDRESS
0750        RET

0760 GETNO  LD D,(HL)     ;SAVE THE ZEAP LINE NUMBER
0770        CALL DIGIT    ;CONVERT TO AN ASCII BYTE
0780        CALL DIGIT    ;DO SAME WITH NEXT NUMBER
0790        LD (HL),D     ;RESTORE ZEAP LINE NUMBER
0800        RET

0810 DIGIT  XOR A         ;CLEAR A
0820        RLD           ;ROTATE NIBBLE INTO A
0830        ADD A,£30     ;CONVERT TO ASCII
0840        CALL TRANS    ;OUTPUT BYTE TO WORDEASE FILE
0850        DEC HL        ;RESTORE ZEAP POINTER
0860        RET
```

This routine checks to see if there has been more than one comment delimiter ";" on one line. If a second one is detected then it is treated as an ordinary character.

```
0870 COMLIN BIT 0,(IX)    ;CHECK COMMENT FLAG
0880        RET NZ        ;RETURN IF 2nd ";"
0890        SET 0,(IX)    ;NOTE 1st ";" FOUND
0900        PUSH AF       ;SAVE THE CHARACTER
```

```
0910         LD DE,ENDMRK   ;OUTPUT "(£A0)^T45"
0920         LD A,£5
0930         CALL CODE
0940         POP AF         ;RECOVER CHARACTER
0950         DEC HL
0960         RET
```

End of file reached, so put "^N1" there, the end of file marker £FF, and save the end of file address.

```
0970 END     LD DE,NEWLIN   ;"^N1"
0980         LD A,£3
0990         CALL CODE      ;OUTPUT THE FORMAT CODE
1000         PUSH BC        ;SAVE CURRENT WORDEASE ADDRESS
1010         LD A,£FF       ;END OF FILE MARKER
1020         CALL TRANS
1030         POP HL         ;RECOVER END OF FILE ADDRESS
1040         LD BC,£1000    ;PUT ADDRESS AT £1000
1050         LD A,L
1060         CALL TRANS
1070         LD A,H
1080         CALL TRANS
1090         SCAL £5B       ;RETURN TO MONITOR
```

Table of format codes.

```
1100 NEWLIN DEFB £5E £4E £31 £20     ;"^N1"
1110 ENDMRK DEFB £A0                 ;END OF LINE MARKER
1120 TAB45  DEFB £5E £54 £32 £33     ;"^T45"
1130 TAB23  DEFB £5E £54 £32 £33     ;"^T23"
1140 TAB30  DEFB £5E £54 £33 £30     ;"^T30"
1150 COMENT DEFB £00                 ;COMMENT FLAG
```

Secondary Uses for the Lucas AVC Graphics Card

by Stan. Gent

The Lucas Advanced Video Controller provides an easily used, moderately high-definition colour graphics facility on the Nascom range of computers.

However, I have seen no mention of the fact that it can also provide a very useful addition to the memory capacity of the machine in non-graphical applications, by virtue of the fact that the board contains 48 kilo-bytes of dynamic RAM, arranged in 3 pages of 16K each.

These three pages are all located at the same address (normally 8000H to BFFFH, although there is provision for changing this by means of hard-wired links). Page selection is via port B2, but only operates after execution of the AVC control software (G32 or G48). As the execution of G32 or G48 seems to require the prior initialisation of Basic, even if all subsequent work is to be in machine code, it is also a wise precaution to set the Basic memory ceiling below 8000H. The response "32767" or less to the question "Memory size?" will achieve this.

Using Nas-Sys, the keyboard command

```
O B2 1
```

will select page 1 (normally the red page),

```
O B2 2
```

will select page 2 (green) and

```
O B2 3
```

will select page 3 (blue), all of which are 16K long.

If the Nascom also has its normal RAM extending over the same address area, O B2 0 will select the normal RAM, giving, in effect, a choice of 4 separate pages of 16K each, in the location 8000H to BFFFH.

Obviously, the page selection can also be carried out within a program, and it is here that the full benefit can be realised. Listing 1 shows one way of achieving this in machine code whilst, in Basic, only a single-line instruction is required as in listing 2. (If using Basic, the memory ceiling limitation referred to earlier should not be applied.)

Don't forget, of course, that if your control program is in the area of memory that you have just paged out, even a Nascom won't know what to do next! The section of memory just paged in must, in this situation, contain the next program instruction and it must be located at the next consecutive address to that in the program counter at the time of changing page.

I have found this facility particularly useful whilst developing programs destined to reside in ROM in my paged EPROM board. Experience has shown that it is all too easy to make mistakes in shifting control from one page to another, and re-programming EPROMs is a tedious business! The paged RAM in the AVC provides an easy way of simulating the paged EPROM, allowing easy testing and modification without the need to re-program an EPROM every time.

Another possible use for the AVC memory is to hold large blocks of data to which the main program (in normal RAM) refers only when required. This could arise, for example, in database programs, word-processor or financial applications ... in fact, wherever memory capacity is being strained by storage of large amounts of data, look up tables, etc.

I hope this article will stimulate other owners of the AVC to suggest more 'unadvertised' uses for this excellent, but poorly promoted, board.

Incidentally, reverting momentarily to the graphics application of the AVC, I am having trouble patching into the DUMP command due, I believe, to an error in the Lucas software or manual. If anyone has dealt with this problem I would be pleased to hear from them.

Listing 1.
Machine code program to page-in the required 16K page of memory and return control to Nas-Sys.

```
        3E xx      LD A,xx
        D3 B2      OUT(B2H),A
        DF 5B      SCAL MRET

                       xx=00 for normal system RAM
                       xx=01 for AVC 'RED' page
                       xx=02 for AVC 'GREEN' page
                       xx=03 for AVC 'BLUE' page
```

Listing 2.
Basic instruction to select the various pages.

```
        OUT 178,xx
```

see above for values of xx.

Another Compass Addition

by Alan Marshall

In volume 3, number 2 of the Nascom Newsletter, I described the addition of two more commands and two more pseudo ops to the COMPASS assembler (version 1.3). This article describes the addition of a further command.

As there were only two 'spare' command letters, the BADCMD facility of COMPASS is used. This was thoughtfully provided by Level 9 as a jump at £0F00 for the addition of extra commands, and the address of this jump is changed to the address of the new routine.

The new routine, the 'Z' command, provides a means of listing label addresses without having to reassemble the program. Once a program has been assembled, pressing 'Z' followed by a Newline will list the labels and their addresses immediately, the number being displayed at a time being controlled by the 'H' command. The command will work before assembly but rubbish will be displayed.

The program as shown below has its origin at COMPASS+£1A53, as it is assumed that the previous additions have been incorporated. If they have not, then the origin is at COMPASS+£197E and line 10 will have to be altered accordingly.

The COMPASS Assembler is unusual in that the labels are not restricted in length, and so the storage of their addresses is also unusual. The usual way is to split the symbol area into 8 byte blocks, six bytes being for the label and two for its address. In this instance, the symbol area is split up into four byte blocks, with two bytes for the address and the other two bytes as the address of the label in the source area. As an example, the label 'RIN' of this program appeared in the symbol table as :

    01 60 08 00

the first two bytes being the label address and the second, the object address. A label address of 00 00 indicates the end of the list and, as the program outputs the object address first, it is erased in lines 36 and 37.

The following listing shows how simple it is to display the labels once the storage is understood. Lines 56 and 57 put the command address in the jump table, so a cold start is necessary for the program to work.

NAS-SEM - Nascom Z80 assembler Page 0000

```
                RIN     EQU     008
                CR      EQU     00D
                ESC     EQU     01B
                ROUT    EQU     030
                TBCD3   EQU     066
                SYMP    EQU     00F71
                PAGSIZ  EQU     00F7A
                COMPSS  EQU     0E500
                RETURN  EQU     COMPSS+0000F
                BADCMD  EQU     COMPSS+000D4
                BREAK   EQU     COMPSS+0013F
                        ORG     COMPSS+01A53
FF53 1A         ZCMND   LD      A,(DE) ;get command
FF54 FE 5A              CP      "Z"     ;is it 'Z' ?
FF56 C2 D4 E5           JP      NZ,BADCMD ;jump if not
FF59 2A 71 0F           LD      HL,(SYMP) ;symbol table
FF5C 3A 7A 0F   ZCMND1  LD      A,(PAGSIZ) ;get page size
FF5F 47                 LD      B,A ;set for repeat
FF60 E5         ZCMND2  PUSH    HL ;save pointer
FF61 23                 INC     HL ;point to object address
FF62 23                 INC     HL
FF63 5E                 LD      E,(HL) ;get low byte
FF64 23                 INC     HL ;point to high byte
FF65 56                 LD      D,(HL) ;get high byte
FF66 23                 INC     HL ;point to next label
FF67 EB                 EX      DE,HL ;get address in HL
FF68 DF 66              SCAL    TBCD3 ;display address
FF6A EB                 EX      DE,HL ;pointer back in HL
FF6B E3                 EX      (SP),HL ;exchange pointers
FF6C 5E                 LD      E,(HL) ;get label address
FF6D 23                 INC     HL
FF6E 56                 LD      D,(HL) ;get high byte
FF6F E1                 POP     HL ;point at next label
FF70 7A                 LD      A,D ;check if end
FF71 B3                 OR      E
FF72 20 06              JR      NZ,ZCMND3 ;jump if not
FF74 3E 1B              LD      A,ESC ;erase false address
FF76 F7                 RST     ROUT
FF77 C3 0F E5           JP      RETURN ;return to COMPASS
FF7A 1A         ZCMND3  LD      A,(DE) ;get label char.
FF7B FE 20              CP      020 ;space ?
FF7D 28 08              JR      Z,ZCMND4 ;jump if it is
FF7F FE 80              CP      080 ;control character ?
FF81 30 04              JR      NC,ZCMND4 ;jump if it is
FF83 F7                 RST     ROUT ;output character
FF84 13                 INC     DE ;point to next character
FF85 18 F3              JR      ZCMND3 ;get it
FF87 3E 0D      ZCMND4  LD      A,CR ;screen next line
FF89 F7                 RST     ROUT
FF8A 10 D4              DJNZ    ZCMND2 ;finish page
FF8C CF                 RST     RIN ;input from keyboard
FF8D FE 1B              CP      ESC
FF8F 20 CB              JR      NZ,ZCMND1 ;get next page
FF91 C3 3F E6           JP      BREAK ;return to COMPASS
                        ORG     COMPSS+0002C
E52C 53 FF              DEFW    ZCMND
```

Poke, Doke ... Moke!

by Garry Rowland

MOKE is a relocatable USR routine for the Nascom ROM Basic version 4.7 that will display menus, title pages and game graphics held in data lists. The data is formatted using offsets and repeat counters to reduce memory requirements to a minimum. The data is read directly from DATA lists; this avoids the duplication of data that results when using string variables or literals.

The MOKE's formatted lists are not quite compatible with the format expected by Basic's READ statement. In choosing the control characters for offset, repeat and terminate - the need for characters that visually indicate their function and not likely to be used in text or graphics was the main consideration. The control characters choosen are:

- / offset
- " repeat
- ! terminate

The " (ditto) repeat control character and the absence of data separators between each charcter in the MOKE list will cause a conventional READ statement to read a block of MOKE data. In a large block of data, calculation of the number of READs required to skip a section of MOKE data could be a bit tiresome. If each new MOKE list started on a new line, the lists can be selected by using the RESTORE n statement, where n is the line number on which the list starts. I don't think there will be many applications where MOKE data lists will need to be read by the READ statement, but if that is required, the MOKE will recognise data separators (though inclusion of them will double the length of your data list!) and & could be substituted for " in the MOKE code.

There could have been a few more features, like repeating blocks of the MOKE list, and fewer compromises; the routine could also have been a lot longer. My maxim is : Long USRs get filed away, short USRs get used.

Using the Moke

DATA	The MOKE reads data items immediately following the DATA statement, including spaces.
,	A data separator must follow any decimal number that is an argument to " or / . It can also be used to separate each item (one character). If the last character on the line is a space then this should be followed by a comma, otherwise the space will be lost when the line is entered.

!	Terminator. The data list will be read by the MOKE until it either finds a terminator or the end of the Basic program text. An OD, Out of Data, error is reported if no terminator or data is found before it gets to the end of the program.
"	Repeat. Must be followed by a decimal number in the range of 1 to 255 (0 gives 256 repeats).
/	Offset. Must be followed by a decimal number in the range of 1 to 65529 (1 to 32767 and -32768 to -7).

An SN, Syntax, error will be reported if a decimal value greater than 65529 is used as an argument to " or /. No error will be reported for the following:

No comma after any number following " or /.
A minus sign or any other character between "/ and argument.
No argument!

The effect of these errors will be seen, but they will not be catastrophic as the memory above 0C00H is protected from 'poking' by the MOKE.

A CRT pointer set to the first point on the screen to be 'poked' should be passed to the MOKE through the USR function. After each 'poke' the CRT pointer is incremented, bear this in mind when calculating offsets. When the MOKE finds the terminator, control is passed back to Basic. The CRT pointer will be passed back set to the address of the last character 'poked' plus one. If the last item in the MOKE list is an offset, the result of the calculation will be returned.

```
100 REM ... MOKE USR DATA with simple demo
110 REM ... load MOKE machine code
120 FOR A=3200 TO 3310 STEP 2
130 READ D:DOKE A,D:NEXT
140 DOKE 4100,3200
150 REM ... demonstration
160 CLS:A=2265
170 RESTORE 2010:D=USR(A)
180 D=INT(RND(1)*4):O=1
190 IF D AND 1 THEN O=64
200 IF D AND 2 THEN O=-O
210 A=A+O:IFA<2058 OR A>2570 THEN A=A-O*2
220 FOR D=1 to INT(RND(1)*28)+4
230 IF (INP(0) AND 127)<>127 THEN 250
240 NEXT:GOTO 170
250 D=USR(2058)
260 FOR D=0 TO 3000:NEXT
270 CLS:END
1000 REM MOKE USR() code
1001 DATA -29747,17129,10827,4316,9086
1002 DATA 11518,-1496,15102,14376,10423
1003 DATA -474,10287,-445,10273,-439,8226
1004 DATA -13051,-5723,1304,13362,7692
1005 DATA 30721,3326,1072,13370,524
```

```
1006 DATA 7427,-3296,-12776,8995,7862
1007 DATA -13818,-7231,24099,22051,21485
1008 DATA 4297,14029,9192,-31746,-18904
1009 DATA -12859,-5520,-18495,-4064,-8680
1010 DATA -23091,-5143,17417,-5299,-24040
1011 DATA -9182,30736,-12991,-3854,201
2000 REM ... graphics demo
2010 DATA "10,/53      /53,  I'M  ,
2020 DATA/53, DYING /53, FOR A /53
2030 DATA  KEY  /53,       /53,"10,!
2040 REM ... screen fill demo
2050 DATA "255, "81, "11, "51, "11, "51
2060 DATA  THANKYOU   "51, "11, "51, "11, "255
2070 DATA "81,!
```

FOR SALE - Nascom 2, 32K RAM with ZEAP, NasDis, DeBug, Naspen and Bits & PCs Toolkit, all in ROM. Boxed with fan, includes a recorder with many tapes. Z80 books, manuals, etc. Cash £175 ono
Ring (Bury) 061-797-3015

FOR SALE - Nascom 2, cased with 48K RAM B, AVC (model B), ZEAP, Naspen, Nasprint 80, NasDis/DeBug, Pascal. Offers over £550
Waterlooville (Hants) 07014 52314

FOR SALE - Nascom 2, 48K RAM, Nas-sys 3, ZEAP, NasDis/DeBug, graphics, Basic Toolkit, 6800 cross compiler, parallel printer cable £250
Tel. Ian Doble, St. Agnes 087255 2121

FOR SALE - Nascom 2 in Kenilworth case with fan, 40K RAM, I/O board with UART + 1 PIO, display monitor. Nas-sys 3, Basic Toolkit, Hullforth, Wordease, V&T assembler and games. Manuals and large number of Nascom related magazines.
All above plus IMP printer & IMPRINT that needs attention for £350
Taylor - Tel. 029668 651 evenings

An EPROM Programmer for 2716 and 2732

by A. Hall

This simple design using the Nascom PIO lets the microcomputer do all the work and reduces the hardware requirements to a 4040 CMOS counter, a few small components, a piece of Veroboard and a 25 volt power supply. It gives you a 'user-friendly', almost foolproof design for just under 1K of object code which can either be stored on tape and operated from RAM or itself programmed into an EPROM for use. The software is written for Nas-Sys 1 and will run on a Nascom 2 or expanded Nascom 1.

The program sets the ports to mode 3, initiates the outputs, checks the 5 volt supply and reminds you to switch on but not insert the EPROM yet.

It asks whether a 2716 or 2732 is to be programmed, reminds you to set the type switch and checks that it is in the correct position.

It then tells you to switch off, insert the EPROM and switch on again and, after checking that the supply has been cut and restored, finally displays the menu of options, and of which may be selected. These are : -

1. Check EPROM completely erased.
2. Dump to RAM for examination.
3. Program and check against source.
4. Return to monitor.

Options 2 and 4 allow use of the normal tabulate function to find unused sections of the EPROM and option 3 programs as many bytes as required from any address in memory to any part of the EPROM.

When the RAM source address, EPROM start address and number of bytes have been entered, a prompt for a final start command is given to allow time to cross-check before committing the EPROM to programming.

During the latter, a pair of rapidly changing symbols is displayed to avoid the impression that nothing is happening. When both symbols are null characters, the programming is complete and the EPROM is checked against its RAM source and a correct or error message displayed before the program returns to the menu.

Port A is used to write data to or read it from the EPROM and Port B uses four bits as outputs to control the 4040 address counter, the programming logic and the voltage switch, and two bits as inputs to check the switch positions. The latter may seem unnecessary, but the wrong type setting is likely to destroy the EPROM when the 25 volt supply is switched on by the transistors so be warned!

It helps if the two diodes are of the Shotky variety to keep the volt drop down but the five transistors are not critical provided that they are rated for 25 volts or above and can pass the charging current of 0.1 MFD capacitor required by the 2732.

The 25 volt source is not specified as it may be convenient to find an available stabilised power unit for occasional use. Fig. 3 shows a simple regulator which will accept anything between 26 and 50 volts, giving out 25 volts at up to 30mA continuously. It would be suitable for a well smoothed voltage doubler from one of the transformer windings on the Nascom power supply.

RAM storage is required at locations 0C81 to 0C8A inclusive for addresses, byte count and EPROM type.

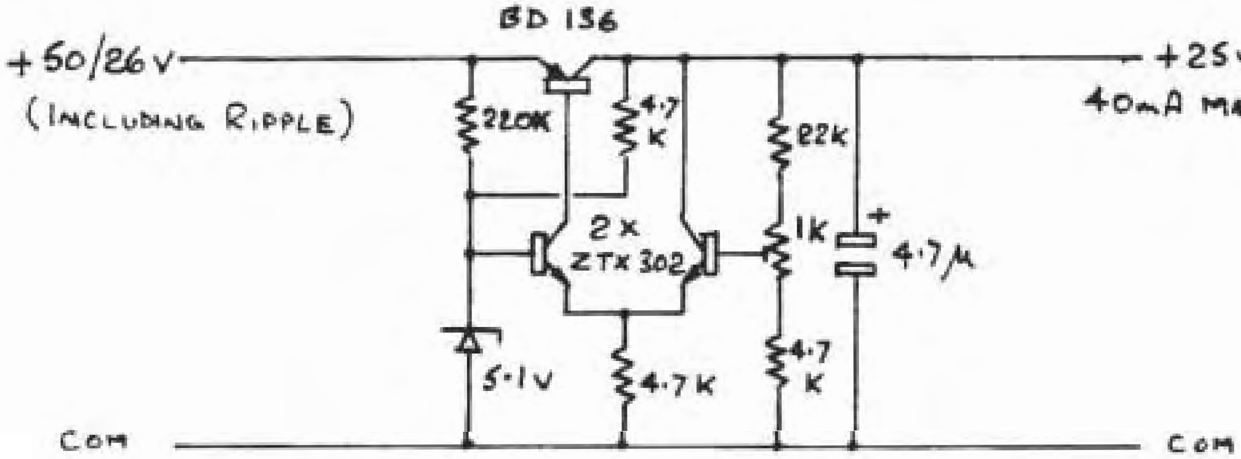
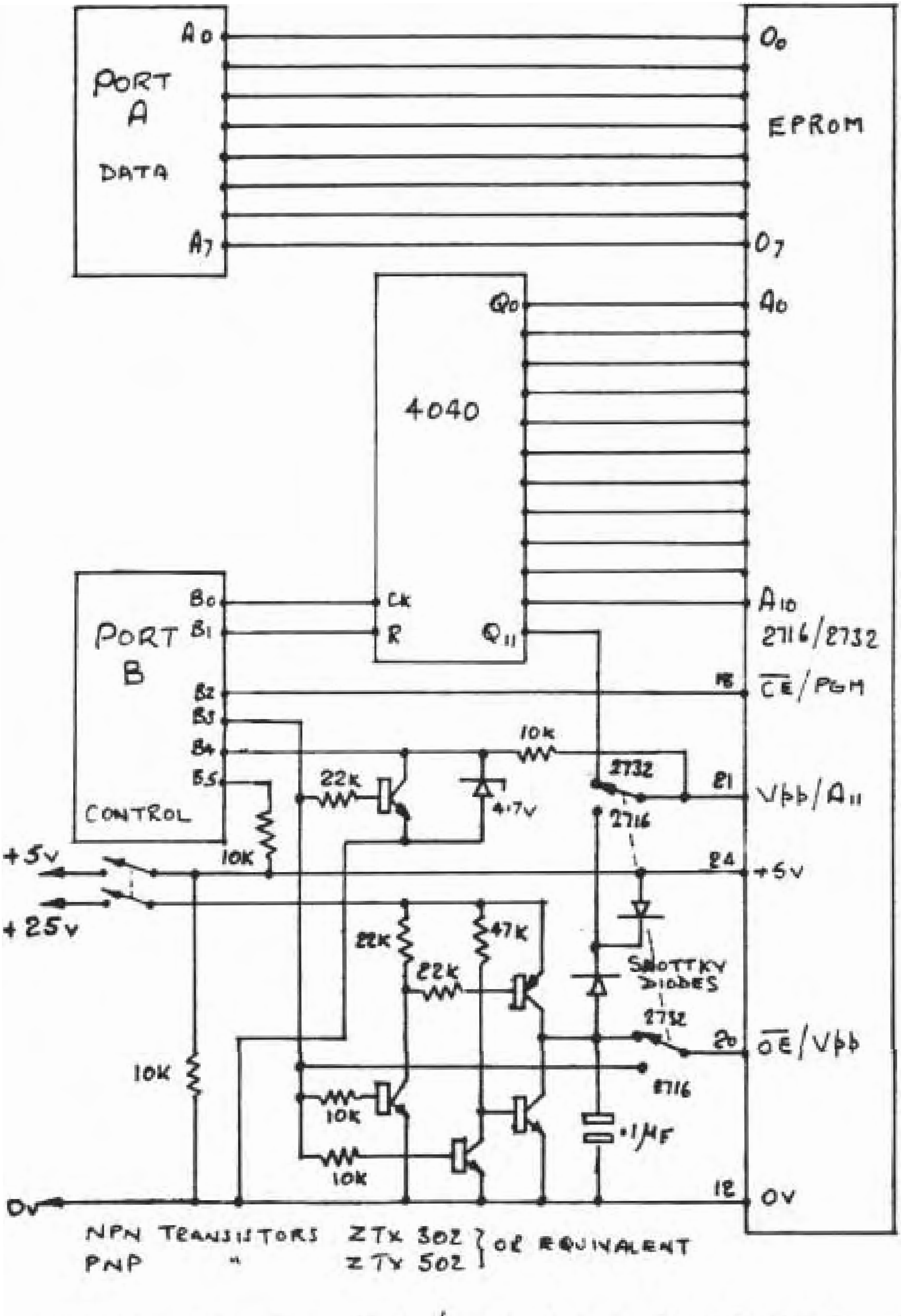


FIG 3. SIMPLE REGULATOR FOR EPROM PROGRAMMER



NASCOM DRIVEN 2716/2732 EPROM PROGRAMMER FIG 2

```
                ; ***EPROM PROGRAMMER FOR 2716/2732***
                ;
                ;INTERFACES
                ;
                ;PORT A : BITS 0 TO 7 DATA
                ;
                ;PORT B : BIT 0  : CLOCK EXT. COUNTER
                ;         BIT 1  : RESET EXT. COUNTER
                ;         BIT 2  : CONTROL CE/PGM
                ;         BIT 3  : CONTROL READ PROGRAM
                ;         BIT 4  : CHECK TYPE SWITCH
                ;         BIT 5  : CHECK 5 VOLT SUPPLY
                ;
                ;RAM BUFFER LOCATIONS :
                ;   0C81-0C84  : EPROM TYPE
                ;   0C85-0C86  : RAM START ADDRESS
                ;   0C87-0C88  : EPROM START ADDRESS
                ;   0C89-0C8A  : NUMBER OF DATA BYTES
                ;
                ;SUBROUTINES :
                ;   CLEAR       : CLEAR SCREEN
                ;   LINE        : CR & LINEFEED
                ;   RESET       : RESET EXT. COUNTER
                ;   RAMAD       : ASK RAM START ADDRESS
                ;   GETAD       : GET ADDRESS INTO BUFFER
                ;   HEXIN       : CONVERT FROM ASCII TO HEX
                ;   PRFLD       : PRELOAD EXT. COUNTER
                ;   CLOCK       : INCREMENT EXT. COUNTER
                ;   DEL         : DELAY 50mS
                ;   BYTE        : GET EPROM SIZE
                ;   ENTER       : MOVE CURSOR BACK
                ;
                        ORG     01000
                ;
                ;INITIALISE PORTS
                ;
1000 3E FF              LD      A,0FF
1002 D3 06              OUT     (6),A
1004 D3 06              OUT     (6),A ; PORT A, M3, ALL INPUTS
1006 D3 07              OUT     (7),A
1008 3E 30              LD      A,030
100A D3 07              OUT     (7),A ; PORT B, M3, 4&5 INPUTS
100C AF                 XOR     A
100D D3 05              OUT     (5),A ; ALL BITS LOW
100F 3E 0C       ASK    LD      A,00C
1011 F7                 RST     030    ; CLEAR SCREEN
1012 DB 05              IN      A,(5)
1014 CB 6F              BIT     5,A    ; CHECK SUPPLY VOLTS
1016 28 1A              JR      Z,WARN; SUPPLY NOT ON
1018 EF                 RST     028    ; SUPPLY ON
1019 44 4F 4E 27        DEFM    "DON'T insert EPROM yet"
102F 00                 DEFB    0
1030 18 29              JR      TYPE
1032 EF          WARN   RST     028
```

```
1033 53 77 69 74                DEFM    "Switch on. DON'T insert EPROM "
1051 79 65 74                   DEFM    "yet"
1054 00                         DEFB    0
1055 DB 05              VON     IN      A,(5)
1057 CB 6F                      BIT     5,A
1059 28 FA                      JR      Z,VON
105B CD 55 13           TYPE    CALL    LINE
105E EF                         RST     028
105F 45 50 52 4F                DEFM    "EPROM type ?"
106B 00                         DEFB    0
                        ;
                        ;CHECK ENTRY & SAVE IN RAM
                        ;
106C DF 7B                      DEFB    0DF,07B ; INPUT FROM KBD
106E F7                         RST     030      ; ECHO TO VDU
106F FE 32                      CP      "2"
1071 20 9C                      JR      NZ,ASK   ; INVALID INPUT
1073 DF 7B                      DEFB    0DF,07B
1075 F7                         RST     030
1076 FE 37                      CP      "7"
1078 20 95                      JR      NZ,ASK
107A DF 7B                      DEFB    0DF,07B
107C F7                         RST     030
107D FE 31                      CP      "1"
107F 28 07                      JR      Z,LAST
1081 FE 33                      CP      "3"
1083 28 0D                      JR      Z,ULT
1085 C3 0F 10                   JP      ASK
1088 DF 7B              LAST    DEFB    0DF,07B ; CHECK LAST DIGIT
108A F7                         RST     030
108B FE 36                      CP      "6"
108D 28 0B                      JR      Z,STOR
108F C3 0F 10                   JP      ASK
1092 DF 7B              ULT     DEFB    0DF,07B
1094 F7                         RST     030
1095 FE 32                      CP      "2"
1097 C2 0F 10                   JP      NZ,ASK
109A 2A 29 0C           STOR    LD      HL,(00C29) ; CURSOR ADDRESS
109D 11 84 0C                   LD      DE,00C84
10A0 01 04 00                   LD      BC,4
10A3 2B                         DEC     HL
10A4 ED B8                      LDDR             ; SAVE VDU ENTRY IN RAM
10A6 3E 0C                      LD      A,00C
10A8 F7                         RST     030      ; CLEAR SCREEN
10A9 EF                         RST     028
10AA 53 65 74 20                DEFM    "Set switch to "
10B8 00                         DEFB    0
10B9 21 81 0C                   LD      HL,00C81
10BC ED 5B 29 0C                LD      DE,(00C29)
10C0 01 04 00                   LD      BC,4
10C3 ED B0                      LDIR             ; DISPLAY EPROM TYPE
                        ;
                        ;CHECK SWITCH SETTINGS
                        ;
```

```
10C5 3A B4 0C     SWIT    LD      A,(00C84) ; LAST DIGIT
10C8 FE 36                CP      "6"
10CA 20 08                JR      NZ,TRY
10CC DB 05                IN      A,(5)     ; CHECK SWITCH
10CE CB 67                BIT     4,A       ; TEST 2716
10D0 20 0A                JR      NZ,GO     ; YES
10D2 18 F1                JR      SWIT      ; NO
10D4 FE 32        TRY     CP      "2"
10D6 DB 05                IN      A,(5)
10D8 CB 67                BIT     4,A       ; TEST 2732
10DA 20 E9                JR      NZ,SWIT   ; MUST BE ERROR
10DC CD 43 13     GO      CALL    CLEAR     ; CLEAR SCREEN.
10DF EF                   RST     028
10E0 53 77 69 74          DEFM    "Switch off supply"
10F1 00                   DEFB    0
10F2 DB 05        VOFF    IN      A,(5)     ; CHECK SUPPLY OFF
10F4 CB 6F                BIT     5,A
10F6 20 FA                JR      NZ,VOFF
10F8 EF                   RST     028
10F9 49 6E 73 65          DEFM    "Insert EPROM & switch on"
1111 00                   DEFB    0
1112 DB 05        SUPP    IN      A,(5)     ; CHECK SUPPLY ON
1114 CB 6F                BIT     5,A
1116 28 FA                JR      Z,SUPP
                  ;
                  ;INTERFACE OK, DISPLAY MENU
                  ;
1118 CD 55 13     MENU    CALL    LINE      ; SUPPLY ON
111B EF                   RST     028
111C 41 6E 79 20          DEFM    "Any key for options."
1130 00                   DEFB    0
1131 DF 62        KEY     DEFB    0DF,062   ; INPUT ROUTINE
1133 30 FC                JR      NC,KEY
                  ;
1135 CD 43 13             CALL    CLEAR
1138 EF                   RST     028
1139 20 20 20 20          DEFM    "          OPTIONS:"
1148 00                   DEFB    0
1149 CD 55 13             CALL    LINE
114C EF                   RST     028
114D 31 20 20 43          DEFM    "1  Check erased."
115D 00                   DEFB    0
115E CD 55 13             CALL    LINE
1161 EF                   RST     028
1162 32 20 20 44          DEFM    "2  Dump to RAM."
1171 00                   DEFB    0
1172 CD 55 13             CALL    LINE
1175 EF                   RST     028
1176 33 20 20 50          DEFM    "3  Program."
1181 00                   DEFB    0
1182 CD 55 13             CALL    LINE
1185 EF                   RST     028
1186 34 20 20 4E          DEFM    "4  NAS-SYS."
1191 00                   DEFB    0
```

```
1192 CD 55 13               CALL   LINE
1195 EF                     RST    028
1196 20 20 20 45            DEFM   "    Enter No. Required:"
11AC 00                     DEFB   0
11AD DF 7B            WAIT  DEFB   0DF,07B  ; INPUT ROUTINE
11AF FE 31                  CP     "1"
11B1 28 0E                  JR     Z,CHECK
11B3 FE 32                  CP     "2"
11B5 28 42                  JR     Z,DUMP
11B7 FE 33                  CP     "3"
11B9 28 6C                  JR     Z,PROG
11BB FE 34                  CP     "4"
11BD 28 35                  JR     Z,NAS
11BF 18 EC                  JR     WAIT     ; NO INPUT
                      ;
                      ;CHECK ALL LOCATIONS 0FF
                      ;
11C1 CD 43 13         CHECK CALL   CLEAR
11C4 CD CC 13               CALL   BYTE
11C7 CD 59 13               CALL   RESET
11CA DB 04            NEXTC IN     A,(4)    ; INPUT FROM EPROM
11CC FE FF                  CP     0FF      ; ERASED ?
11CE 20 14                  JR     NZ,ERRM  ; NO
11D0 CD B9 13               CALL   CLOCK    ; YES
11D3 0B                     DEC    BC
11D4 78                     LD     A,B
11D5 B1                     OR     C        ; LAST BYTE ?
11D6 20 F2                  JR     NZ,NEXTC ; NO
11D8 EF                     RST    028      ; YES
11D9 45 72 61 73            DEFM   "Erased."
11E0 00                     DEFB   0
11E1 C3 18 11               JP     MENU
11E4 EF               ERRM  RST    028
11E5 4E 6F 74 20            DEFM   "Not Erased."
11F0 00                     DEFB   0
11F1 C3 18 11               JP     MENU
                      ;
                      ;RETURN TO MONITOR
                      ;
11F4 CD 43 13         NAS   CALL   CLEAR
11F7 DF 5B                  DEFB   0DF,05B  ; MONITOR RETURN
                      ;
                      ;DUMP EPROM DATA TO RAM
                      ;
11F9 CD 43 13         DUMP  CALL   CLEAR
11FC CD 62 13               CALL   RAMAD
11FF 21 86 0C               LD     HL,00C86
1202 CD 7C 13               CALL   GETAD
1205 2A 85 0C               LD     HL,(00C85) ; RAM POINTER
1208 CD CC 13               CALL   BYTE
120B CD 59 13               CALL   RESET
120E DB 04            NEXTD IN     A,(4)    ; INPUT FROM EPROM
1210 77                     LD     (HL),A   ; INTO RAM
1211 CD B9 13               CALL   CLOCK
```

```
1214 23                    INC     HL
1215 0B                    DEC     BC
1216 78                    LD      A,B
1217 B1                    OR      C         ; LAST BYTE ?
1218 20 F4                 JR      NZ,NEXTD ; NO
121A CD 55 13              CALL    LINE      ; YES
121D EF                    RST     028
121E 44 6F 6E 65           DEFM    "Done."
1223 00                    DEFB    0
1224 C3 18 11              JP      MENU
                     ;
                     ;PROGRAM EPROM
                     ;
1227 CD 43 13        PROG  CALL    CLEAR
122A 3E FF                 LD      A,0FF
122C D3 06                 OUT     (6),A
122E 3E 00                 LD      A,0
1230 D3 06                 OUT     (6),A     ; PORT A SET FOR OUTPUT
1232 CD 62 13              CALL    RAMAD
1235 21 86 0C              LD      HL,00C86 ; HOLDS RAM ADDRESS
1238 CD 7C 13              CALL    GETAD
123B CD 55 13              CALL    LINE
123E EF                    RST     028
123F 45 50 52 4F           DEFM    "EPROM Address ? nnnn"
1253 00                    DEFB    0
1254 CD DF 13              CALL    ENTER     ; MOVE CURSOR BACK
1257 21 88 0C              LD      HL,00C88 ; HOLDS EPROM ADDRESS
125A CD 7C 13              CALL    GETAD
125D CD 55 13              CALL    LINE
1260 EF                    RST     028
1261 4E 6F 2E 20           DEFM    "No. of Bytes ?  nnnn"
1275 00                    DEFB    0
1276 CD DF 13              CALL    ENTER
1279 21 8A 0C              LD      HL,00C8A ; HOLDS NO. OF BYTES
127C CD 7C 13              CALL    GETAD
127F CD 55 13              CALL    LINE
1282 EF                    RST     028
1283 50 20 74 6F           DEFM    "P to Program"
128F 00                    DEFB    0
1290 CF              READY RST     08
1291 FE 50                 CP      "P"
1293 20 FB                 JR      NZ,READY ; TIME FOR CHECKING
1295 CD A6 13              CALL    PRELD
1298 2A 85 0C              LD      HL,(00C85)
129B ED 4B 89 0C           LD      BC,(00C89)
129F 3A 84 0C              LD      A,(00C84) ; 2716 OR 2732
12A2 FE 36                 CP      "6"
12A4 28 07                 JR      Z,PRO6
12A6 FE 32                 CP      "2"
12A8 28 72                 JR      Z,PRO2
12AA C3 18 11              JP      MENU
12AD 3E 08           PRO6  LD      A,008
12AF D3 05                 OUT     (5),A      ; 2716 CONTROL
12B1 7E              LOOP6 LD      A,(HL)
```

```
12B2 D3 04                      OUT   (4),A       ; DATA
12B4 3E 0C                      LD    A,00C
12B6 D3 05                      OUT   (5),A
12B8 CD C2 13                   CALL  DEL         ; PROGRAM PULSE
12BB 3E 08                      LD    A,008
12BD D3 05                      OUT   (5),A
12BF 3E 09                      LD    A,009
12C1 D3 05                      OUT   (5),A       ; CLOCK PULSE
12C3 3E 08                      LD    A,008
12C5 D3 05                      OUT   (5),A
12C7 23                         INC   HL
12C8 0B                         DEC   BC
12C9 ED 43 E4 0B                LD    (00BE4),BC; GRAPHICS
12CD 78                         LD    A,B
12CE B1                         OR    C           ; LAST BYTE ?
12CF 20 E0                      JR    NZ,LOOP6    ; NO
12D1 3E FF                CORR  LD    A,0FF       ; YES. VERIFY
12D3 D3 06                      OUT   (6),A
12D5 D3 06                      OUT   (6),A       ; PORT A. SET FOR INPUT
12D7 3E 00                      LD    A,00
12D9 D3 05                      OUT   (5),A       ; RESET COUNTER
12DB CD A6 13                   CALL  PRELD
12DE 2A 85 0C                   LD    HL,(00C85); RAM POINTER
12E1 ED 4B 89 0C                LD    BC,(00C89); EPROM POINTER
12E5 DB 04                CLOOP IN    A,(4)
12E7 BE                         CP    (HL)        ; DATA CORRECT ?
12E8 20 1C                      JR    NZ,NOTP     ; NO
12EA CD B9 13                   CALL  CLOCK       ; YES
12ED 23                         INC   HL
12EE 0B                         DEC   BC
12EF 78                         LD    A,B
12F0 B1                         OR    C           ; LAST BYTE ?
12F1 20 F2                      JR    NZ,CLOOP    ; NO
12F3 CD 55 13                   CALL  LINE        ; YES
12F6 EF                         RST   028
12F7 50 72 6F 67                DEFM  "Programmed."
1302 00                         DEFB  0
1303 C3 1B 11                   JP    MENU
1306 CD 55 13             NOTP  CALL  LINE
1309 EF                         RST   028
130A 50 72 6F 67                DEFM  "Program Error."
1318 00                         DEFB  0
1319 C3 1B 11                   JP    MENU
                          ;
131C 3E 0C                PRO2  LD    A,00C
131E D3 05                      OUT   (5),A       ; 2732 CONTROL
1320 7E                   LOOP2 LD    A,(HL)      ; COMMENTS AS FOR 2716
1321 D3 04                      OUT   (4),A       ; WITH APPROPRIATE
1323 3E 08                      LD    A,008       ; CHANGES TO ALLOW FOR
1325 D3 05                      OUT   (5),A       ; CONTROL DIFFERENCES
1327 CD C2 13                   CALL  DEL
132A 3E 0C                      LD    A,00C
132C D3 05                      OUT   (5),A
132E 3E 0D                      LD    A,00D
```

```
1330 D3 05                    OUT    (5),A
1332 3E 0C                    LD     A,00C
1334 D3 05                    OUT    (5),A
1336 23                       INC    HL
1337 0B                       DEC    BC
1338 ED 43 E4 0B              LD     (00BE4),BC
133C 78                       LD     A,B
133D B1                       OR     C
133E 20 E0                    JR     NZ,LOOP2
1340 C3 D1 12                 JP     CORR
                     ;
                     ;SUBROUTINES
                     ;
1343 3E 0C           CLEAR    LD     A,00C
1345 F7                       RST    030
1346 11 DD 0B                 LD     DE,00BDD      ; TOP LINE
1349 21 81 0C                 LD     HL,00C81      ; EPROM TYPE
134C 01 04 00                 LD     BC,04         ; NO. CHARACTERS
134F ED B0                    LDIR                 ; WRITE IT
1351 CD 55 13                 CALL   LINE
1354 C9                       RET
                     ;
1355 3E 0D           LINE     LD     A,00D
1357 F7                       RST    030
1358 C9                       RET
                     ;
1359 3E 02           RESET    LD     A,002
135B D3 05                    OUT    (5),A
135D 3E 00                    LD     A,00
135F D3 05                    OUT    (5),A        ; BIT 1 PULSED
1361 C9                       RET
                     ;
1362 EF              RAMAD    RST    028
1363 52 41 4D 20              DEFM   "RAM Address ?   nnnn"
1377 00                       DEFB   0
1378 CD DF 13                 CALL   ENTER
137B C9                       RET
                     ;
137C 06 02           GETAD    LD     B,2          ; TWO BYTES
137E CD 8F 13        CONV     CALL   HEXIN        ; GET DIGIT
1381 07                       RLCA
1382 07                       RLCA
1383 07                       RLCA
1384 07                       RLCA
1385 4F                       LD     C,A          ; SAVE TOP 4 BITS
1386 CD 8F 13                 CALL   HEXIN
1389 81                       ADD    A,C          ; COMBINE DIGITS
138A 77                       LD     (HL),A       ; STORE IN BUFFER
138B 2B                       DEC    HL
138C 10 F0                    DJNZ   CONV         ; GET SECOND PAIR
138E C9                       RET
                     ;
138F CF              HEXIN    RST    008          ; GET INPUT
1390 F7                       RST    030          ; ECHO IT
```

```
1391 FE 30             CP     "0"
1393 38 FA             JR     C,HEXIN      ; REJECT <030
1395 FE 3A             CP     ":"
1397 38 0A             JR     C,NOLET      ; ACCEPT 030-039
1399 FE 41             CP     "A"
139B 38 F2             JR     C,HEXIN      ; REJECT 03A-03F
139D FE 47             CP     "G"
139F 30 EE             JR     NC,HEXIN     ; REJECT >046
13A1 C6 09             ADD    A,9          ; CONVERT TO HEX
13A3 E6 0F       NOLET AND    00F          ; MASK LOWER NIBBLE
13A5 C9                RET
                 ;
13A6 CD 59 13    PRELD CALL   RESET
13A9 ED 4B 87 0C       LD     BC,(00C87)   ; EPROM ADDRESS
13AD 78                LD     A,B
13AE B1                OR     C            ; WAS IT 0 ?
13AF C8                RET    Z            ; YES
13B0 CD B9 13    INCR  CALL   CLOCK        ; NO
13B3 0B                DEC    BC
13B4 78                LD     A,B
13B5 B1                OR     C            ; FINISHED ?
13B6 C8                RET    Z            ; YES
13B7 18 F7             JR     INCR         ; NO
                 ;
13B9 3E 01       CLOCK LD     A,1
13BB D3 05             OUT    (5),A
13BD 3E 00             LD     A,0
13BF D3 05             OUT    (5),A        ; PULSE BIT 0
13C1 C9                RET
                 ;
13C2 C5          DEL   PUSH   BC           ; SAVE BYTE COUNT
13C3 06 0A             LD     B,00A        ; NO. OF LOOPS
13C5 3E E3       DLOOP LD     A,0E3        ; DELAY CONSTANT
13C7 FF                RST    038          ; CALL RDEL
13C8 10 FB             DJNZ   DLOOP
13CA C1                POP    BC           ; 50mS LATER
13CB C9                RET
                 ;
13CC 3A 84 0C    BYTE  LD     A,(00C84)    ; GET EPROM TYPE
13CF FE 32             CP     "2"
13D1 28 04             JR     Z,KB4
13D3 FE 36             CP     "6"
13D5 28 04             JR     Z,KB2
13D7 01 00 10    KB4   LD     BC,01000     ; 2732, 4K BYTES
13DA C9                RET
13DB 01 00 08    KB2   LD     BC,00800     ; 2716, 2K BYTES
13DE C9                RET
                 ;
13DF 2A 29 0C    ENTER LD     HL,(00C29)   ; CURSOR
13E2 2B                DEC    HL
13E3 2B                DEC    HL
13E4 2B                DEC    HL
13E5 2B                DEC    HL           ; BACK 4 SPACES
13E6 22 29 0C          LD     (00C29),HL   ; REPLACE

13E9 C9                RET
```

Cheap RGB Monitor

by Roger Dowling

An RGB monitor on the cheap - providing that you already have a Ferguson TX9 or TX10 colour set!

The Ferguson TX range of televisions privide an easy conversion for RGB use and work very well with the Nascom AVC. The AVC gives an RGB output as standard rather than PAL encoded video and so an ordinary composite video input colour monitor is of no real use without adding an extra encoder board to the AVC which apart from adding to the cost of the board, it will also reduce definition. All the Ferguson TX models except the very early TX9 can be converted as data insertion points have been provided on-board to enable some models to be equipped with teletext reception without fuss. The TX10 has been the standard large screen (22 and 26 inch) chassis for a couple of years and the TX9 has been the very popular chassis used in models from 14 inch upwards.

If you are the possessor of a Ferguson TX set (I believe that all models have TX somewhere on the front or possibly back cover) then you check to see if you have a TX9 or TX10 simply by looking in the back. If there is only one horizontal panel then you have a TX9. The TX10 has two panels, one large horizontal one and a vertical panel at the extreme back of the set which will hinge down to a horizontal position.

To check whether you have an early or late version of the TX9, you must identify the PAL decoder IC which is a 28 pin device and will be found near the middle of the panel. If it is a TDA3560 or TDA3561 then you are in luck. The earlier type is marked PC1365.

The data insertion points on the decoder IC are on pins 13 (R), 15 (G), 17 (B) and a switching input on pin 9. These pins are conveniently wired to a 6 pin plug on the board PL18 although for some obscure reason the plug connections are slightly different between the two types of chassis. I will give details for the TX9 and show differences for the TX10 where they occur.

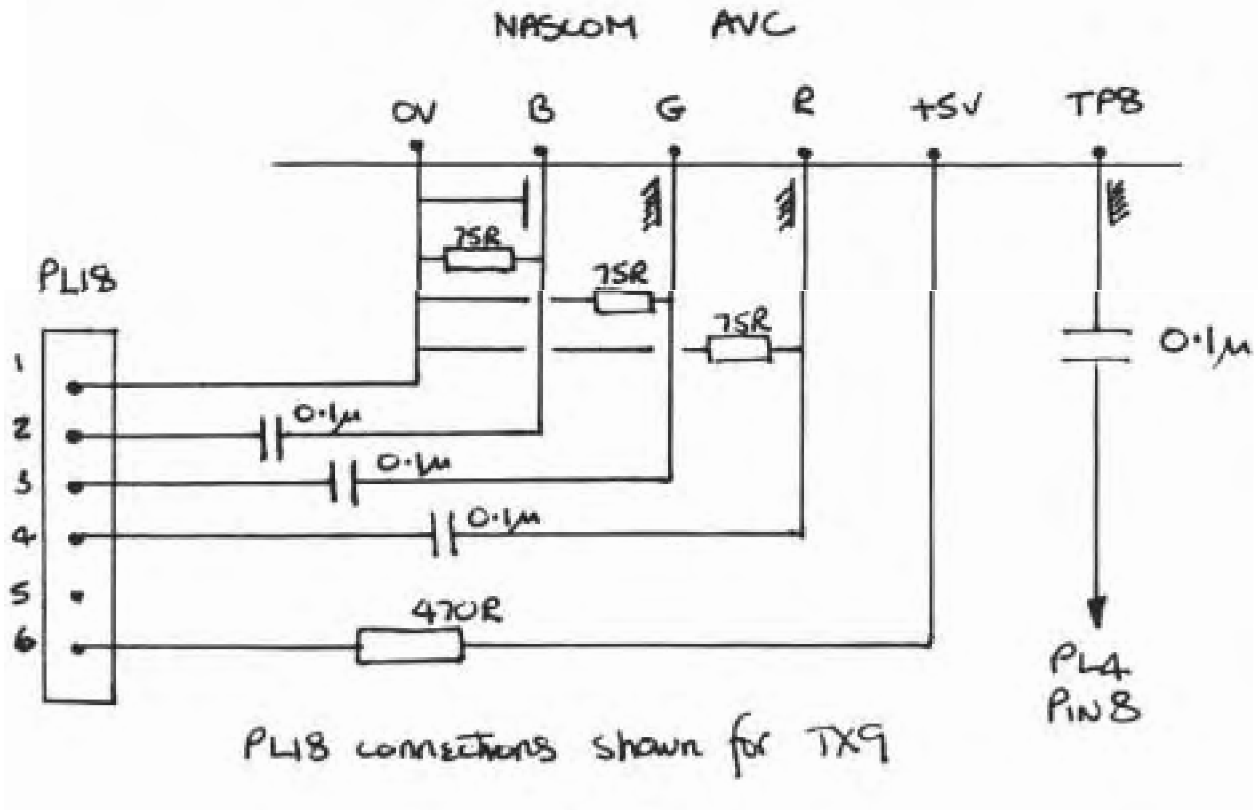
The RGB outputs from the AVC are at a level of 1 volt and are therefore suitable to feed into the data insertion points via isolating capacitors. Loading these outputs with 75R provides the correct contrast level displayed on the CRT. A synchronising signal also needs to be fed to the receiver and can be obtained from TP8 on the Nascom AVC. The i.f. module should be removed from the set. This is a plug-in module consisting of two integrated circuits, one of 16 pins (TDA2540) and one of 8 pins (SL430). It should be easy enough to identify - on the TX9 it is on the far left (looking in from the back) near the mains input plug and directly in front of the tuner. On the TX10 it is on the vertical panel to the right of the tuner.

Syncs from TP8 on the AVC should be wired (via an isolating capacitor) to pin 8 of the socket that the i.f. module plugs into (marked as PL4).

The Red, Green and Blue outputs should be wired, via isolating capacitors) to PL18 pins 4, 3 and 2 respectively. A supply of 5 volts should be taken from the computer via a 470R resistor and wired to PL18 pin 6 (pin 5 on the TX10). This will supply about 3 volts to the decoder IC and internally switch it from video from the i.f. panel to the data inputs. Pin 1 of PL18 is earth. These connections are shown in the diagram below.

For the few that may need their TX set to watch television programmes, it could be arranged to switch the AVC sync and the i.f. video output at PL4 with a CMOS switch operated by the 5 volt supply from the Nascom and, together with this supply also switching the decoder IC between video input and data inputs, an automatic changeover from TV to computer when the computer was plugged in could easily be achieved.

A very important point - before any work of this type is done, the TV set must be isolated from the mains. This not only protects you but also the computer. A suitable torroidial isolating transformer which is small enough to fit inside any model can be obtained from ILP Electronics, Graham Bell House, Roper Close, Canterbury, Kent CT2 7EP. (type number 32030).



Smarten up those Basic Listings !

by D.F. Gutteridge

The following machine code routine enables Basic programs to be listed with indented FOR-NEXT loops, considerably clarifying program structure.

The example shows that a Basic program may be listed in the normal way, and by leaving Basic via the MONITOR command and executing the indent routine at 0CB0H (EC80), an indented listing is produced clearly showing the program structure.

Notes.

1. The routine is fully relocatable.

2. The routine may be interupted using ESC, and the number of lines displayed may be set by the LINES command as usual.

3. The amount of indenting may be altered by changing the entry at 0CBBH and 0CC9H.

4. The routine only picks up FOR and NEXT at the beginning of lines. Messy structures like:

```
170 A=0 : FOR I=1 TO 10 : PRINT "*";
180 NEXT I : PRINT
```

will cause problems. The purpose of the routine is to produce good, visually structured listings to well structured programs.

5. The program is written for Nascom ROM Basic Version 4.7 running under Nas-Sys 1 but it should work with Nas-Sys 3 as well.

```
                 ; MACHINE CODE ROUTINE TO INDENT 'FOR/NEXT'
                 ; LOOPS IN A BASIC LISTING
                 ;
                 ;DAVID GUTTERIDGE
                 ;
                              ORG     00C80
                 RESTAB  EQU     0E143
                 INDENT  EQU     010F9
                 STTBAS  EQU     0105E
                 ;INITIALISE SIZE TO INDENT TO ZERO
0C80 AF                       XOR     A
0C81 32 F9 10                 LD      (INDENT),A
                 ;POINT TO START OF BASIC PROGRAM
0C84 2A 5E 10                 LD      HL,(STTBAS)
0C87 E5                       PUSH    HL
                 ;INITIALISE LINES COUNT
0C88 CD 33 E7                 CALL    0E733
                 ;PROCESS NEXT LINE
0C8B E1          LINE         POP     HL
0C8C 4E                       LD      C,(HL)
0C8D 23                       INC     HL
0C8E 46                       LD      B,(HL)
0C8F 23                       INC     HL
```

```
0C90 78                      LD      A,B
0C91 B1                      OR      C
             ;IF END OF PROGRAM, RETURN TO BASIC
0C92 CA F8 E3                JP      Z,0E3FB
             ;DECREMENT LINES COUNTER
0C95 CD 46 E7                CALL    0E746
             ;RETURN TO BASIC IF 'ESC' PRESSED
0C98 CD 61 E8                CALL    0E861
0C9B C5                      PUSH    BC
             ;PRINT NEWLINE
0C9C CD 81 EB                CALL    0EB81
0C9F 5E                      LD      E,(HL)
0CA0 23                      INC     HL
0CA1 56                      LD      D,(HL)
0CA2 23                      INC     HL
0CA3 E5                      PUSH    HL
0CA4 EB                      EX      DE,HL
             ;PRINT LINE NUMBER
0CA5 CD AD F9                CALL    0F9AD
             ;LOOK FOR 'FOR' & 'NEXT' TOKENS AND
             ; INCREMENT OR DECREMENT INDENT
0CA8 E1                      POP     HL
0CA9 C5                      PUSH    BC
0CAA 7E                      LD      A,(HL)
0CAB FE 81                   CP      081
0CAD 28 08                   JR      Z,MORE
0CAF FE 82                   CP      082
0CB1 28 10                   JR      Z,LESS
0CB3 D7 1A                   RCAL    PRINT
0CB5 18 25                   JR      END
0CB7 3A F9 10        MORE    LD      A,(INDENT)
0CBA C6 02                   ADD     A,2
0CBC 32 F9 10                LD      (INDENT),A
0CBF D7 0E                   RCAL    PRINT
0CC1 18 19                   JR      END
0CC3 D7 0A           LESS    RCAL    PRINT
0CC5 3A F9 10                LD      A,(INDENT)
0CC8 D6 02                   SUB     2
0CCA 32 F9 10                LD      (INDENT),A
0CCD 18 0D                   JR      END
0CCF 3A F9 10        PRINT   LD      A,(INDENT)
0CD2 47              LOOP    LD      B,A
0CD3 B7                      OR      A
0CD4 C8                      RET     Z
0CD5 EF                      RST     028
```

```
0CD6 20                     DEFM    " "
0CD7 00                     DEFB    0
0CD8 05                     DEC     B
0CD9 80                     ADD     A,B
0CDA 18 F6                  JR      LOOP
                 ;PRINT REST OF LINE
                 ;RESERVED WORD TOKENS ARE EXPANDED
                 ; USING THE RESERVED WORD TABLE
                 ;THE REMAINING CODES ARE TREATED AS
                 ; ASCII CODES AND PRINTED LITERALLY
                 ;CODE 00 INDICATES THE END OF THE LINE
0CDC C1          END        POP     BC
0CDD 3E 20                  LD      A,020
0CDF CD 9B E6    CHAR       CALL    0E69B
0CE2 7E          NEXT       LD      A,(HL)
0CE3 B7                     OR      A
0CE4 23                     INC     HL
0CE5 28 A4                  JR      Z,LINE
0CE7 CB 7F                  BIT     7,A
0CE9 28 F4                  JR      Z,CHAR
0CEB D6 7F                  SUB     07F
0CED 4F                     LD      C,A
0CEE 11 43 E1               LD      DE,RESTAB
0CF1 1A          SEARCH     LD      A,(DE)
0CF2 13                     INC     DE
0CF3 B7                     OR      A
0CF4 CB 7F                  BIT     7,A
0CF6 28 F9                  JR      Z,SEARCH
0CF8 0D                     DEC     C
0CF9 20 F6                  JR      NZ,SEARCH
0CFB E6 7F       LEAVE      AND     07F
0CFD CD 9B E6               CALL    0E69B
0D00 1A                     LD      A,(DE)
0D01 13                     INC     DE
0D02 CB 7F                  BIT     7,A
0D04 28 F5                  JR      Z,LEAVE
0D06 18 DA                  JR      NEXT
```

NOTICE

Mr. R.M. Dowling of 11, Westbrooke Road, Welling, Kent DA16 1PR, has asked me to mention that he is interested in forming a NasDos user group to circulate a single disk containing programs and items of interest for its members. If you are interested, please contact him at the above address, enclosing a stamped, addressed envelope.

Factorials !!!!

by J.A. Hart

This program produces the factorial of any number between 2 and 3000. Numbers must be input as four digits ie. 2 is input as 0002 and 3000 as 3000. There is no error checking so reset and re-enter number if incorrect. When running, the number of the factorial to which the computer has calculated is displayed on the top left of the screen and the digit length of that number is shown in the middle of the top line. As it is written for maximum speed and not for efficiency in the use of memory, it requires 48K of RAM starting at 1000H. The 48K is for data storage only as the program is less than 1K long. To avoid using more memory for the program, it alters various subroutines which otherwise would be quadruplicated, therefore, it will not work in ROM. The answer is produced 20 times faster than a similar Basic program.

```
1C80 21 90 1C 11 90 0C 01 00 04 ED B0 00 FF 00 FF 00 00 00 00 21
1C94 00 10 AF 77 23 7C FE E0 C2 96 0C CD 21 0E 3E 01 32 00 10 32
1CA8 00 40 32 00 70 32 00 A0 32 B0 0C AF 32 B1 0C 32 B2 0C 32 B3
1CBC 0C CD 92 0E 21 B0 0C 34 7E FE 0A C2 E6 0C 3E 00 77 23 34 7E
1CD0 FE 0A C2 E6 0C 3E 00 77 23 34 7E FE 0A C2 E6 0C 3E 00 77 23
1CE4 34 00 3E B3 CD 10 0E 3E 10 CD 1A 0E 06 11 CD C3 0D 3E 10 32
1CF8 52 0E CD 50 0E CD 50 0E CD 50 0E 3E B2 CD 10 0E 3E 40 CD 1A
1D0C 0E 06 41 CD C3 0D 3E 40 32 52 0E CD 50 0E CD 50 0E 3E B1 CD
1D20 10 0E 3E 70 CD 1A 0E 06 71 CD C3 0D 3E 70 32 52 0E CD 50 0E
1D34 3E B0 CD 10 0E 3E A0 CD 1A 0E 06 A1 CD C3 0D 06 11 3E 40 32
1D48 67 0E CD 62 0E 3E 70 32 67 0E CD 62 0E 3E A0 32 67 0E CD 62
1D5C 0E 3E 40 32 88 0E CD 83 0E 3E 70 32 88 0E CD 83 0E 3E A0 32
1D70 88 0E CD 83 0E CD AA 0E CD F0 0E FE 00 C2 C0 0C 3A B2 0C FE
1D84 00 C2 C0 0C 3A B1 0C FE 01 C2 C0 0C 3A B0 0C FE 00 C2 C0 0C
1D98 00 00 00 00 00 21 FF 3F 2B 7E FE 00 CA A0 0D 7E C6 30 32 AE
1DAC 0D EF 30 00 2B 7C FE 0F C2 A7 0D 7D FE FF C2 A7 0D EF 0D 00
1DC0 DF 5B 00 3A B0 0C FE 00 C2 D8 0D 21 00 A0 AF 77 23 7C B8 C2
1DD4 CE 0D C9 00 3A B0 0C FE 01 C8 00 00 00 00 21 00 A0 AF 5F 3A
1DE8 B0 0C 57 AF 86 27 15 C2 EC 0D 83 27 77 E6 F0 1F 1F 1F 1F 5F
1DFC 7E E6 0F 77 23 7C B8 C2 E7 0D C9 00 00 00 00 00 00 00 00 00
1E10 32 C4 0D 32 D9 0D 32 E8 0D C9 32 CD 0D 32 E4 0D C9 EF 49 4E
1E24 50 55 54 20 4E 4F 20 20 20 0D 00 CD E0 0E 32 7C 0D CD E0 0E
1E38 32 84 0D CD E0 0E 32 8C 0D CD E0 0E 32 94 0D EF 0D 00 C9 00
1E4C 00 00 00 00 21 00 70 56 36 00 23 5E 72 23 56 73 7C B8 C2 56
1E60 0E C9 21 00 10 11 00 A0 1A 86 27 77 E6 F0 1F 1F 1F 1F 4F 7E
1E74 E6 0F 77 23 13 79 86 27 77 7C B8 C2 68 0E C9 21 00 10 11 00
1E88 A0 C5 78 D6 10 47 ED B0 C1 C9 3E 11 32 F1 0C 32 44 0D 3E 41
1E9C 32 0E 0D 3E 71 32 28 0D 3E A1 32 3F 0D C9 21 00 11 2B 7E FE
1EB0 00 CA AD 0E 7D FE F0 38 01 24 24 7C 32 F1 0C 32 44 0D 32 AC
1EC4 0E C6 30 32 0E 0D C6 30 32 28 0D C6 30 32 3F 0D C9 00 00 00
1ED8 00 00 00 00 00 00 00 00 DF 7B 00 32 E7 0E EF 30 00 3A E7 0E
1EEC E6 0F C9 00 3A B3 0C C6 30 32 D2 0B 3A B2 0C C6 30 32 D3 0B
1F00 3A B1 0C C6 30 32 D4 0B 3A B0 0C C6 30 32 D5 0B 21 FF 3F 2B
1F14 7E FE 00 CA 13 0F 00 00 00 00 2C 7C D6 10 67 3E E0 32 77 0F
1F28 DD 21 66 0F 00 00 00 00 00 11 10 27 CD 58 0F 11 E8 03 DD 34
1F3C 00 CD 58 0F 11 64 00 DD 34 00 CD 58 0F 11 0A 00 DD 34 00 CD
1F50 58 0F 11 01 00 DD 34 00 0E 00 0C B7 ED 52 30 FA 0D 19 3E 30
1F64 B1 32 E4 0B 3A B3 0C C9 00 00 FF 00 FF 00 FF 00 FF 00 FF 00
```

V & T Assembler Modifications

by Alan Marshall

The title of this article is somewhat of a misnomer as most of the modifications are actually made to Nas-Dis, though they are made to facilitate its use with the V & T assembler.

One apparently miraculous feature of Nas-Dis is its ability to dis-assemble a program directly into a ZEAP source file. Other assemblers are catered for by dis-assembling to tape and then reading the tape into the assembler in its automatic mode. While this works very well, there is no doubt that the ability to dis-assemble directly to the source area is a great advantage and so, using the excellent listing of Nas-Dis, I made the following modifications. These enable the tape version of Nas-Dis to dis-assemble any program directly to the source area of a V & T assembler.

As the V & T works in hex, the '£', or hash, sign is not required, so addresses C4FFH and CF60H are changed from 23H (£) to 30H (0).

The message delimiter used by ZEAP is a "/" while V & T uses "'" so the following addresses need to be changed from 2FH to 27H :-

C6C0
C6D3
C704
CE8F

As the V & T can only put one DEFB on one line then the DJNZ instruction at C6E2/3 needs to be replaced by two NOPs. If this course is followed then the program will assemble with no editing. If, however, the 3 at C6E3 is replaced by 2 then the V & T DEFW can be used, though you must remember to reverse the order of the two bytes while editing. Depending on which course you follow, the 3 at CD6B needs to be changed to either 1 or 2.

The V & T is a little unusual in that it uses EX AF,AFG instead of EX AF,AF' to change to the alternate accumulator so 27H at CA43 needs to be changed to 47H. Another difference is that ZEAP uses a null as the end of line marker while V & T uses 1FH. CEDEH is the place to change.

The last change is the major change and comes about because of the different ways that the assemblers keep track of things. This short piece of code is put at CC65 to suit the simpler V & T requirements:-

```
CC65 2A 4E 0E      LD HL,(0E4E) ;address of source code
CC68 2B            DEC HL
CC69 22 56 0E      LD (0E56),HL
CC6C ED 5B 52 0E   LD DE,(0E52)
CC70 ED 53 4E 0E   LD (0E4E),DE
```

The Nas-Dis routine ALLINO is next copied from CC8C to CC74 and the following code entered. This writes the end of source code address to the screen and puts the cursor up two lines so that when NEWLINE is pressed after initiating V & T, the assembler is ready to go.

```
CC88 20 1A         JR NZ CCA4 ; not finished
CC8A EF            RST PRS ; print to screen
CC8B 20 40 30 20   DEFM ' @0 '
CC8F 00            NOP ; end print
CC90 1B            DEC DE ; source pointer
CC91 EB            EX DE,HL
CC92 DF 66         SCAL TBCD3 ; print address on screen
CC94 EF            RST PRS
CC95 13 1B         DEFW 1B13
CC97 13 1B         DEFW 1B13
CC99 13 1B         DEFW 1B13
CC9B 00            NOP
CC9C DF 5B         SCAL MRET ; return to monitor
```

The version of V & T that I have was produced before Nas-Sys 3 was introduced. Because of this, it suffers from not being able to print the work areas on the top line while running under Nas-Sys. If you would like to display these work areas on the top line then the following small modifications should be made.

When V & T is initially loaded, the 2DH bytes of the Nas-Sys table at 20AE need to be copied to 207E. Copy only 27H bytes to 207E and, as this leaves the Nasbug calls in the same place, the work areas can be printed on the top line.

Because V & T jumps directly to the 'WRITE' subroutine in Nas-Sys instead of through the SCAL routine, the jump address at 209C needs to be changed to 04FB.

NASPEN Print Routine

by Alan Marshall

This program adds several short routines to the straight print routine that I have been using with my printer, a Centronics 737.

The machine-dependent part of the routine has been put at the end, so that if you have some other printer you can replace the 'PRINT' subroutine with your own. The whole program is fully relocatable with the proviso that the two registers immediately precede it. The FNDSTR subroutine is only in the program to make it fully relocatable and a space has been left so that 0C82 to 0C84 can be replaced by 21 YY XX, where XXYY is the address of the first register.

If the FNDSTR subroutine is removed then all relative calls to the PRINT subroutine will nedd to be recalculated. For those of you not familiar with machine code, the Nas-Sys 'A' command can be used. For example, the new address of PRINT would be 0CED. To calculate the new RCAL2 enter 'A 0C8D 0CED'. On the next line would appear '197A 0060 5E'. The last figure is the relative jump so 65H at 0C8E would be replaced by 5EH.

One minor fault with the relative call is that it is unconditional so two further bytes can be saved in lines 57 to 59 and 62 to 64 by replacing the two instructions with 'CALL NZ,LFOUT'. The high nibble of the first register is used as a flag register with the low nibble as a store for the number of spaces to be printed in a margin.

The first flag (bit 4) is set after a CR. The Centronics printer uses a CR as a print instruction, with a LF being output only if there are other characters in the print buffer. If ther are consecutive CRs then the subsequent ones will be ignored by the printer and so the check is made of the flag and, if set, a LF followed by a CR is output. The CR is output in case the following line has 80 characters, in which case there would be an overflow of the buffer and the last character would be printed on the following line.

The second flag (bit 5) is checked after a CR to see if double spacing is required. If so, a LF followed by a CR is output.

The third flag (bit 6) is also checked after each CR. If it is set then a margin of up to 15 spaces is printed, the number being stored in the lower nibble of the register. This is obtained in line 68 where the AND instruction is used to delete the high nibble. A check is made that a number has been entered or the pointer will print a margin of 255 spaces wide. At this point the CR flag is reset as it would cause an extra LF to be printed if double spaced mode.

The fourth flag (bit 7) is set if a page is to be repeated, with the number of repeats being found in the second register. This is checked at the end of each page, the marker for this being 01, which needs to be set in 1010H for the Generate command. If a page marker is sent to this routine then a CR is output as a signal to the printer, then the CR flag is reset in preparation for the next page. A check is then made for the repeat flag. If it is not set then the Naspen Text and Window pointers are set to the next page (lines 49 & 50) and Naspen warm started so that it is ready to print it when asked. If the flag is set then the stack pointer is reset by popping its top two registers. Then, after checking that a number has been entered into the register and that the number requested has not been printed, the program jumps to the Naspen routine to repeat the printing of the page.

The other two routines not yet mentioned are the start and finish routines. The first character to be output by Naspen is a null. When this is received, the routine jumps to LBL6, where the mode flags are checked. The last character is 02 and this is converted to a CR to allow the printer to finish.

While not being a very comprehensive print routine and though the modes have to be set manually before printing, I have found this a very useful addition and I hope that you do too.

```
                    0000 ; NASPEN PRINT ROUTINE
                    0001 ;
                    0002 ;PAGE    : POINTS NASPEN TO NEXT PAGE
                    0003 ;END     : OUTPUTS CR
                    0004 ;CR      : OUTPUTS LF FOR CONSECUTIVE
                    0005 ;FLAGS   : 4 - CONSECUTIVE CRS
                    0006 ;            5 - DOUBLE SPACING
                    0007 ;            6 - MARGIN. NO. SPACES IN 0-3
                    0008 ;            7 - REPEAT PAGE. NO. IN 2ND REGISTER
                    0009 ;
                    0010 PAGE   EQU    001
                    0011 LF     EQU    00A
                    0012 CR     EQU    00D
                    0013 RCAL   EQU    010
                    0014 ENDMKR EQU    0FF
                    0015 WINDOW EQU    01014
                    0016 TEXT   EQU    01018
                    0017 NPENWS EQU    0B806
                    0018 NPRINT EQU    0BABE
                    0019 ;
                    0020        ORG    00C80
0C80                0021 STORE  DEFS   2
0C82 D7             0022 RCAL1  RST    RCAL
0C83 5D             0023        DEFB   FNDSTR-RCAL1-2
```

```
0C84 00          LBL1   NOP    ;SPACE FOR LD HL INSTRUCTION
0C85 FE 01              CP     PAGE ;END OF PAGE ?
0C87 20 2A              JR     NZ,LBL4
0C89 CB A6              RES    4,(HL) ;RESET CR FLAG
0C8B 3E 0D              LD     A,CR
0C8D D7          RCAL2  RST    RCAL ;PRINT CR
0C8E 65                 DEFB   PRINT-RCAL2-2
0C8F CB 7E              BIT    7,(HL) ;REPEAT ?
0C91 C1                 POP    BC ;SAVE RETURN ADDRESS
0C92 D1                 POP    DE ;TEXT POINTER
0C93 23                 INC    HL ;POINT TO NEXT REGISTER
0C94 28 0C              JR     Z,LBL2 ;SKIP IF NOT REPEAT
0C96 7E                 LD     A,(HL) ;NUMBER REQUIRED
0C97 3D                 DEC    A ;PRINTED ONE
0C98 28 16              JR     Z,LBL3 ;FINISHED ?
0C9A FE FF              CP     0FF ;NO NUMBER ENTERED
0C9C 28 12              JR     Z,LBL3
0C9E 77                 LD     (HL),A ;SAVE NUMBER PRINTED
0C9F C3 BE BA           JP     NPRINT ;REPEAT
0CA2 3E FF       LBL2   LD     A,ENDMKR
0CA4 EB                 EX     DE,HL ;TEXT POINTER
0CA5 23                 INC    HL
0CA6 23                 INC    HL ;POINT TO START OF NEXT PAGE
0CA7 BE                 CP     (HL)
0CA8 28 06              JR     Z,LBL3
0CAA 22 14 10           LD     (WINDOW),HL ;NASPEN TO NEXT PAGE
0CAD 22 18 10           LD     (TEXT),HL
0CB0 C3 06 B8    LBL3   JP     NPENWS
0CB3 FE 0D       LBL4   CP     CR
0CB5 20 31              JR     NZ,LBL8
0CB7 D7          RCAL3  RST    RCAL ;PRINT CR
0CB8 3B                 DEFB   PRINT-RCAL3-2
0CB9 CB 66              BIT    4,(HL) ;CHECK CR FLAG
0CBB 28 02              JR     Z,LBL5
0CBD D7          RCAL4  RST    RCAL ;IF SET PRINT LF
0CBE 1A                 DEFB   LFOUT-RCAL4-2
0CBF CB E6       LBL5   SET    4,(HL) ;SET CR FLAG
0CC1 CB 6E       LBL6   BIT    5,(HL) ;DOUBLE SPACING FLAG
0CC3 28 02              JR     Z,LBL7
0CC5 D7          RCAL5  RST    RCAL ;PRINT LF
0CC6 12                 DEFB   LFOUT-RCAL5-2
0CC7 CB 76       LBL7   BIT    6,(HL) ;MARGIN FLAG
0CC9 28 23              JR     Z,LBL9
0CCB 7E                 LD     A,(HL) ;GET NUMBER OF SPACES
0CCC E6 0F              AND    00F ; FORM LOW NIBBLE
0CCE C8                 RET    Z ;NOT IF 255 SPACES
0CCF CB A6              RES    4,(HL) ;RESET CR FLAG
0CD1 47                 LD     B,A ;NUMBER OF SPACES
0CD2 3E 20              LD     A,020
0CD4 D7          RCAL6  RST    RCAL ;PRINT SPACES
0CD5 1E                 DEFB   PRINT-RCAL6-2
0CD6 10 FC              DJNZ   RCAL6 ;PRINT NEXT SPACE
0CD8 C9                 RET
0CD9 3E 0A       LFOUT  LD     A,LF
0CDB D7          RCAL7  RST    RCAL
0CDC 17                 DEFB   PRINT-RCAL7-2
```

```
0CDD 3E 0D                 LD     A,CR
0CDF 18 13                 JR     PRINT
0CE1 E1           FNDSTR   POP    HL ;RETURN ADDRESS
0CE2 E5                    PUSH   HL
0CE3 01 FC FF              LD     BC,STORE-LBL1 ;DIFFERENCE
0CE6 09                    ADD    HL,BC ;HL POINTS TO FLAG REG.
0CE7 C9                    RET
0CE8 CB A6        LBL8     RES    4,(HL) ;RESET CR FLAG
0CEA FE 00                 CP     0 ;FIRST CHARACTER ?
0CEC 28 D3                 JR     Z,LBL6 ;CHECK DS & MARGIN FLAGS
0CEE FE 02        LBL9     CP     2 ;END OF TEXT ?
0CF0 20 02                 JR     NZ,PRINT
0CF2 3E 0D                 LD     A,CR
0CF4 F5           PRINT    PUSH   AF
0CF5 DB 05        PRINT1   IN     A,(5)
0CF7 CB 47                 BIT    0,A
0CF9 20 FA                 JR     NZ,PRINT1
0CFB F1                    POP    AF
0CFC CB FF                 SET    7,A
0CFE D3 04                 OUT    (4),A
0D00 00 00                 DEFW   0
0D02 CB BF                 RES    7,A
0D04 D3 04                 OUT    (4),A
0D06 00 00                 DEFW   0
0D08 CB FF                 SET    7,A
0D0A D3 04                 OUT    (4),A
0D0C C9                    RET
```

Assembler / NASPEN Routines

by Alan Marshall

There are, no doubt, many people reading this article who do their programming using an assembler. There is a possiblity that there are some who can write without having to move blocks of source code around. There is even a possiblity that some write comments on each line of code that they write, as they write it. For those of you who are bad programmers, like me, the following two programs may be of some use.

Although these programs are written for the V & T assembler, I am sure that they can easily be adapted to work with ZEAP. The differences are, I believe, that the ZEAP source code starts several bytes after the start of its file address and uses a null instead of 1FH as its end of line marker.

The first is a short program that copies the source code of a V & T assembler to a Naspen file.

First cold start Naspen and leave it then, to start the program type E D00 XXXX YYYY where XXXX is the start of the source code and YYYY is the end of it, taken from the top line of the screen. The first check is that the correct number of arguments has been entered to save crashing the program. After that, the characters are copied from source code to Naspen file with checks for the end of each line, where the 1FH marker is changed to 0DH for Naspen. TBCD3 is then used to put the line number on the screen in ASCII which is then copied into the Naspen file, together with its space to give a correct presentation.

At the end of the source code the Naspen 20H and FFH markers are added, the address of the end of text put in the Naspen store at 101A and then Naspen warm started. You can then use the editing facilities of Naspen either to add comments or to move large blocks around.

When you have finished, leave Naspen and use the second program to copy the text back to the source area for the assembler.

Type E D50 XXXX to start. XXXX is the address of the source area start address and after another check of the number of arguments, the program starts by missing out the first three bytes of the Naspen file. The reason for this is that the ASCII representation of the line number takes four bytes, while the number itself is stored as two bytes. The third byte to be omitted is the extra space inserted for presentation in the previous program. Each character is then copied across in sequence until a CR is found. This is changed to 1FH for the assembler and again the first three bytes of the next line are omitted. When the end of text marker, FFH, is found, the screen is cleared and the end of source code address and renumber command put on the screen so that, as soon as the assembler is entered, pressing NEWLINE twice will set all the parameters of the assembler.

Should you wish to edit a listing, replace the print reflection with this short program so that it prints its output into your Naspen file. One word of warning, it romps through memory at a very high rate of knots so make sure that you have lots available. you may also need to reset the limit address at 1012.

```
PUSH HL ; save pointer
LD HL,(STORE) ; Naspen pointer
LD (HL),A ; print character
INC HL
LD (STORE),HL ; save pointer
POP HL
RET
STORE DEFW 1020 ; start of Naspen file
```

There are only eleven bytes to that routine but very useful ones. Don't forget to set 1020 in the store or your listing could be anywhere.

```
                 ; V&T/NASPEN ROUTINE
                 ;
                 CLS     EQU     00C
                 CR      EQU     00D
                 SCAL    EQU     018
                 PRS     EQU     028
                 ROUT    EQU     030
                 MRET    EQU     05B
                 TBCD3   EQU     066
                 ;
                 ARGN    EQU     00C0B
                 ARG2    EQU     00C0E
                 ARG3    EQU     00C10
                 CURSOR  EQU     00C29
                 ;
                 NPENWS  EQU     0B806
                 ;
                         ORG     00D00
0D00 3A 0B 0C            LD      A,(ARGN)  ;CHECK NO. OF ARGUMENT
0D03 FE 03               CP      3
0D05 28 02               JR      Z,L1
0D07 DF                  RST     SCAL
0D08 5B                  DEFB    MRET
0D09 2A 0E 0C    L1      LD      HL,(ARG2) ;START OF SOURCE CODE
0D0C 11 20 10            LD      DE,01020  ;START OF NASPEN FILE
0D0F ED 4B 10 0C         LD      BC,(ARG3) ;END OF SOURCE CODE
0D13 18 15               JR      L4        ;BEGINNING OF LINE
0D15 7E          L2      LD      A,(HL)
0D16 FE 1F               CP      01F       ;END OF LINE ?
0D18 28 05               JR      Z,L3
0D1A 12                  LD      (DE),A    ;PUT CHAR IN NASPEN
0D1B 13                  INC     DE        ;GET NEXT CHAR
0D1C 23                  INC     HL
0D1D 18 F6               JR      L2
0D1F 3E 0D       L3      LD      A,CR      ;NASPEN CR IS 0D
0D21 12                  LD      (DE),A
0D22 13                  INC     DE        ;POINT TO NEXT LINE
0D23 23                  INC     HL
0D24 B7                  OR      A         ;CHECK FOR SOURCE END
0D25 ED 42               SBC     HL,BC
0D27 09                  ADD     HL,BC
0D28 30 18               JR      NC,L5
0D2A 7E          L4      LD      A,(HL)    ;LOW NIBBLE OF LINE
0D2B 23                  INC     HL
0D2C E5                  PUSH    HL
0D2D 66                  LD      H,(HL)    ;HIGH NIBBLE OF LINE
0D2E 6F                  LD      L,A       ;HL = LINE NUMBER
0D2F C5                  PUSH    BC
0D30 DF                  RST     SCAL      ;PRINT HL IN ASCII
0D31 66                  DEFB    TBCD3
0D32 3E 17               LD      A,017
0D34 F7                  RST     ROUT      ;POINT TO START LINE
0D35 2A 29 0C            LD      HL,(CURSOR);POINT HL TO SCREEN
0D38 01 05 00            LD      BC,5      ;LINE NO. + SPACE
```

```
0D3B ED B0        0054          LDIR                  ;COPY ASCII NO. TO NPEI
0D3D C1           0055          POP     BC
0D3E E1           0056          POP     HL
0D3F 23           0057          INC     HL            ;NEXT CHAR
0D40 18 D3        0058          JR      L2
0D42 EB           0059 L5       EX      DE,HL         ;END OF SOURCE
0D43 36 20        0060          LD      (HL),020      ;NASPEN MARKERS
0D45 23           0061          INC     HL
0D46 36 FF        0062          LD      (HL),0FF
0D48 22 1A 10     0063          LD      (0101A),HL;NASPEN END OF TEXT
0D4B C3 06 B8     0064          JP      NPENWS
                  0065 ;
                  0066 ;NASPEN/V&T ROUTINE
                  0067 ;
0D4E 3A 0B 0C     0068          LD      A,(ARGN)
0D51 FE 02        0069          CP      02
0D53 28 02        0070          JR      Z,L6
0D55 DF           0071          RST     SCAL
0D56 5B           0072          DEFB    MRET
0D57 2A 0E 0C     0073 L6       LD      HL,(ARG2)     ;START OF SOURCE
0D5A 11 23 10     0074          LD      DE,01023      ;START OF NASPEN +3
0D5D 1A           0075 L7       LD      A,(DE)
0D5E FE 0D        0076          CP      CR            ;END OF LINE
0D60 20 0A        0077          JR      NZ,L8
0D62 13           0078          INC     DE            ;LOSE TWO CHARS
0D63 13           0079          INC     DE
0D64 1A           0080          LD      A,(DE)
0D65 FE FF        0081          CP      0FF           ;END OF NASPEN FILE
0D67 28 08        0082          JR      Z,L9
0D69 13           0083          INC     DE            ;LOSE EXTRA SPACE
0D6A 3E 1F        0084          LD      A,01F         ;SOURCE END OF LINE
0D6C 77           0085 L8       LD      (HL),A
0D6D 13           0086          INC     DE            ;GET NEXT CHAR
0D6E 23           0087          INC     HL
0D6F 18 EC        0088          JR      L7
0D71 36 1F        0089 L9       LD      (HL),01F
0D73 23           0090          INC     HL            ;END OF SOURCE CODE
0D74 EF           0091          RST     PRS
0D75 0C           0092          DEFB    CLS
0D76 0D           0093          DEFB    CR            ;MOVE 3 LINES DOWN
0D77 0D 0D        0094          DEFB    00D,00D
0D79 20 40 30 20  0095          DEFM    " @0 "        ;SOURCE CODE END
0D7D 00           0096          NOP
0D7E DF           0097          RST     SCAL          ;PRINT MARKER IN ASCII
0D7F 66           0098          DEFB    TBCD3
0D80 EF           0099          RST     PRS
0D81 0D 0D        0100          DEFB    00D,00D
0D83 20 4E        0101          DEFM    " N"          ;PREPARE TO RENUMBER
0D85 00           0102          NOP
0D86 21 0A 08     0103          LD      HL,0080A      ;RESET CURSOR
0D89 22 29 0C     0104          LD      (CURSOR),HL
0D8C DF           0105          RST     SCAL
0D8D 5B           0106          DEFB    MRET
```

HIGH SCORE
5375
(Playing variation 6)

REVENGE OF THE DROSOPHILA

Save the fruit from the flies, but suffer the consequences if you squash all the flies!
A new action game for Nascom, featuring synchronised animation and 8 game variations.

Just 5 out of a sequence of 11 frames running at approx. 6 frames per second.

There's a multistorey warehouse full of fruit and only one bad fruit on each floor. It's the Runner's job to go in and save the fruit from the impending swarm of mutant fruit flies (Drosophila). Maggot's hatch from the bad fruit and, after a vulnerable period racing around the maze, change into the deadly mutant Drosophila. These flies then look for more fruit to make bad... and that's where your problems start.

The Runner must battle against the life cycle of these monster flies - he looses a life each time he squashes a fly and the only way to get another life is by eating one of the fruit he's trying to save! The game doesn't last long if the Runner squashes all the flies, or the first maggot, the Quibbles see to that. Quibbles are slime moulds (mutant, of course) and will take all of the Runner's lives, no matter how many he has, in one collision. This provides a great incentive to control the fly population rather than indulge in 'run away' squashing.

The Runner

He always starts out from the lift. There will be just one bad fruit among the good fruit on the floor; but if any flies were left on the previous floor, just under half of these flies will also be on the floor. The Runner is always on the move, he will change direction to avoid bumping into a wall if none of the appropriate direction key's are held down. If the automatic pick up option is selected, he will also pick up any fruit he comes across. The Runner can hold a maximum of two fruit. He will not pick up any fruit if his hands are 'full', if the fruit is bad or if he is not in position when he bends over. The Runner can return to the lift at any time to put down the fruit. Only once the fruit has been left at the lift will any points be given. Squashing a maggot will not gain the player any points, but the Runner won't loose any lives either. Squashing a Drosophila will give the player less points than that given for saved fruit and loose the Runner a life. When the Runner eats a fruit, he gets an extra two lives. If eating a fruit would give the Runner more than five lives the fruit is refused and the command ignored.

Drosophila

They will search for good fruit to make bad. When there is no good fruit left, the Runner will have the fruit flies undivided attention. The propagation option provides fast or slow breeding Drosophila, though the duration of each stage of the life cycle gets shorter as the good fruit is used up.

Quibbles

So mean they are made optional! If the Runner squashes a fly, or the first maggot, and no flies are left, a Quibble will be placed on the floor (off screen). If the option is not selected, a fly takes the Quibbles place. Quibbles multiply instantly each time they absorb a fruit. The speed they move at depends on how much fruit is left. You can't argue with a Quibble, they'll take all of the Runner's lives in one swoop. How do you survive the Quibbles? If the fruit counter goes zero, the Runner just might make it back to the lift. Help may come in the form of maggots. Maggots and Quibbles ignore each other, but when the maggots change into flies - the flies will eat the Quibbles wherever they find them, but won't go looking for them.

Fruit

The fruit counter gives the number of good fruit on the floor. It is decremented each time a fruit is picked up, made bad or absorbed by a Quibble. When the counter goes zero, the header will flash as an additional warning that no fruit is left and that the Runner should be taken back to the lift. There is no time limit, but the Runner will not be able to go on to the next floor unless he makes it back to the lift. If, when he gets there, there are no flies left, the score for that floor will be doubled (Note: When the Quibble option is not selected and the fruit counter is zero, no more flies will appear on the floor). Any extra lives the Runner has will be carried over to the next floor. The number in the lift shows the floor you are on.

A brief description of the game, details of the score header and game variations are given at the beginning of the game. There is also the option to change the key's used in play at the start of each new game. Nascom's with an NMI button have a 'game reset' feature. Pressing the NMI button will re-run the game, restoring the key's used in default and resetting the scores.

The key's used in default are in a table at $1FF3_H$ and can be changed as described on the cassette flap. The key's must be changed, and the value $7F_H$ at $23DF_H$ changed to $3F_H$, if your Nascom 1 keyboard doesn't have the extra cursor key's.

This game is set to run on a 4MHz micro, but can be changed to compensate for different CPU clock rates - Enter at 3403_H:

01_H for 1MHz clock
06_H for 2MHz clock
12_H for 4MHz clock
33_H for 8MHz clock

The frame rate can be changed independently of the clock rate. It has been set to a default value of 6fps. To change the frame rate, enter at 3401_H:

58_H for 4 fps. (slow motion)
$3F_H$ for 5 fps.
32_H for 6 fps.
29_H for 7 fps.
21_H for 8 fps. (slapstick!)

These values apply at all clock rates and are approximate.

The tape is recorded at 300 baud on side one and 1200 baud on side two. If the N1 tape format was requested, this will be recorded twice on side two - side one will be left at 300 baud in case you later change to the CUTS interface. Execute at 3173_H.

Revenge of the Drosophila is monitor independent and is supplied with a tape reader for loading under monitor's other than NAS-SYS.

The game requires a 16K Nascom 1, 2 or 3 with NASGRA ROM (version 1).

Incredible Value at just **£8** post free

Available only by mail order from:

GARRY ROWLAND
24 Parsloes Avenue, Dagenham RM9 5NX

An EPROM Emulator for the Nascom

by P. Burgess

Introduction.

This article describes the design and construction of a unit of hardware intended to temporarily replace an EPROM during the developement and debugging of firmware for dedicated Z80 CPU based projects. Detail is given to allow the construction of a unit, including an example of a loading program in Z80 assembler code, and connections to a Nascom. The article is also intended to act as a basis for experiment for anyone who' might wisj to extend the design.

Design Considerations.

My own computer interests lie in the application of microprocessors to small design projects, often employing a Z80 CPU plus minimal RAM and EPROM configurations. A single 2716 EPROM is often enough for the whole program, ie. 2048 (2K) bytes. A Nascom plus and EPROM programmer (such as the Gemini one) makes quite a fair development system. However, when testing a newly built piece of hardware (or a 'target' system, as it is known) it can be necessary to repeatedly blow and erase EPROMs to find faults and debug the program. What is required is a chunk of memory which will look like an EPROM to the target system, but to which a host computer can read and write quickly. This would then replace the EPROM until program development is complete.

Several such emulators are commercially available, but tend to be expensive, as they incorporate their own monitors, keyboards and displays. The Nascom, though, already has its own powerful machine code facilities, so this design relies entirely on the Nascom to manipulate the data, resulting in a very simple (7 common lsTTL chips) design, at, perhaps, the cost of a little flexibility.

The device is controlled from a Z80 PIO, in this case, ports A and B of a Nascom 1, although, of course, other ports may be used.

As drawn, the emulator plugs in directly in place of the single supply rail 2716 EPROM, although it could be extended to 2732 devices by adding extra RAM and decoding. Components are not critical; higher density RAM packages such as the 6116 have become available since the prototype was built and could be worth looking at.

Principle.

The EPROM is emulated by a 2K by 8 bit block of RAM. The data and address lines of the RAM are accessible both by the target hardware and by the host computer, in our case a Nascom. The host can read or write to the memory but the target system can only read.

When the target system is accessing the RAM, the host is switched off and is unable to affect the emulator. When the host is in control, the target system cannot access the RAM.

In order to simplify the address line switching, the host addresses the RAM via a binary counter. The drawback is that the RAM locations must always be addressed in sequence. In practice, this is not a problem as the entire emulator is loaded quickly from an image in the Nascom memory.

The resulting hardwareis, therefore, simple and cheap, yet able to perform a function normally requiring expensive and specialized equipment. (Commercial emulators usually have their own CPU and I/O devices. In the unit described here, the monitor functions already present in the Nascom are used instead.)

Circuit Description.

Referring to the schematic diagram, ICs 8 to 11 form the 2K emulator memory. Since each chip is a 1K by 4 bit memory, they are arranged as two pairs, ICs 8 and 10 form the first 1K block and 9 and 11 the second. Address lines A0 through A9 are common to all 4 chips whilst A10 selects the block to be accessed, by driving the Chip Select lines via an inverter. (Part of IC 6). Two NAND gates (half of, IC 7) are also included in the Select lines to allow the host computer to control them.

The 11 address lines are then multiplexed between the computer and the target system by ICs 2, 3, and 4. (74LS244 tri-state, non-inverting buffers.) Each package contains 8 buffers grouped in sets of 4, hence IC 3 is split 'in half' to provide the necessary configuration. One buffer is spare from each half of IC 3. Note - one of these is used up later on to control the output buffer, IC 5.

The Output Enable lines (pins 1 and 19) of the sets of buffers are connected by an inverter (again part of IC 6) so that only one set may be enabled at one time. The line called SWITCH performs this function and is under control of the Nascom along with the other control lines.

When SWITCH is LOW, the address lines from the target system (via SKT 2) are able to address the RAM. When SWITCH is HIGH, these inputs are turned off and the outputs of IC 1 drive the address lines instead.

The RESET and CLOCK inputs of IC 1 are brought out to the host computer via SKT 1. The WRITE ENABLE (WE, active LOW) inputs of the memory chips are connected together and also go out to the Nascom via SKT 1 as do the 8 data lines and the ground connection. (Power may be obtained from the host if its supply is adequate, although the prototype had an independent supply. About 200mA at 5 volt is required.)

The eight data lines are connected directly to a PIO port on the Nascom without an intermediate buffer. Thsi simplifies the design but care must be taken when writing driver software: see later.

A buffer is definitely needed between the data lines and the target system. This function is performed by IC 5, a further 74LS244. The data flow is of course in one direction only, out of the emulator and into the target system. When SWITCH is HIGH and the computer is in charge, part of IC 3 gates off the Enable to IC 5 completely.

Since the standard 2716 EPROM ha$ two select pins (Chip Select and Power Down) which both must be low to access the EPROM, the remaining two NOR gates of IC 6 are included to simulate this function. Finally, capacitors C1, C2 and C3 are included for supply rail decoupling. One buffer in IC 3 and two NAND gates in IC 7 remain spare. I thought of using one of the gates to drive an LED to show the state of the SWITCH control line.

Constructing.

Virtually any of the usual means of building circuits may be used, the original was built on a piece of plain matrix board about 5 inches square using hand wire-wrapping. It does not take too long this way but care is obviously needed to avaoid wiring errors as they are difficult to find later on. Before commencing wiring, the two disc ceramic decoupling capacitors C2 and C3 should be fitted and connected directly to the supply pins of ICs 10 and 11. The supply and ground connections are then dealt with followed by the address, data and control connections.

Connection from the Nascom to the emulator is via a 16 pin DIL socket, SKT 1 (shown by triangular connections on the diagram). Connections to the target hardware are via a 24 pin DIL socket, SKT 2, which is wired to correspond to the pinouts of a standard 2716 EPROM. The prototype uses a link between the 5 volt supply and pin 24 of SKT 2. This allows the target hardware to draw power from the emulator. Clearly, it has to be disconnected if the target system and the emulator have their own supplies!

The emulator is connected to the target system using a 12 inch length of ribbon cable fitted with a 24 way pin header at each end, wired pin to pin.

On completing the wiring, I suggest powering up the emulator board without any chips inserted. The supply pins may be checked at this stage at each IC socket for correct polarity. The chips may then be inserted one at a time whilst monitoring supply current for excessive drain.

A cable may be made up with a 16 pin header to connect to SKT 1 at one end and an appropriate connector for ports A and B of the Nascom, in my case a Canon type. The connection diagram shows the pin numbers for direct connection to the two 16 pin headers on a Nascom 1. Finally, you can put the whole thing in a box if you want to be slick!

Using the Emulator.

Being simple-minded, I tend to use one extremely basic loader routine which transfers a 2K block of RAM from the Nascom to the emulator. The routine is relocatable but resides in ROM at address D42F which happens to be convenient in my system. The routine is so basic that, as it stands, the data to be transferred must reside at address 4000H in the Nascom memory. It would be easy enough to pick up an extra parameter with a Nas-Sys E command to improve this...

The following table gives the functions of the emulator control lines, driven by the lower five bits of Port A, which is configured as an output port at all times. Port B is used for data transfer and must be configured for output only when loading data, otherwise it is left set up for input.

Control Lines

Port A bit	Name	Function
0	SWITCH	Enables access by host when HIGH
1	ADCLK	A low to high transition on this line advances IC 1's count by one
2	ADRES	A high level on this line resets IC 1
3	/WE	Write Enable for the RAM, active low
4	CS	Chip Select for RAM. Note active HIGH

Port B bits 0 to 7 are connected to data bits 0 to 7 of the emulator respectively.

To load the emulator, the program must generate the following sequence:-

1. Port A is configured as an output.
2. The RAM is deselected and a Reset applied to ADRES.
3. SWITCH must be HIGH to put the computer in control.
4. Pointers are set to indicate the starting address of the data to be transferred and the number of bytes (normally 2K or 800H).

5. Port B is then configured to output and the first byte placed onto the emulator data bus.

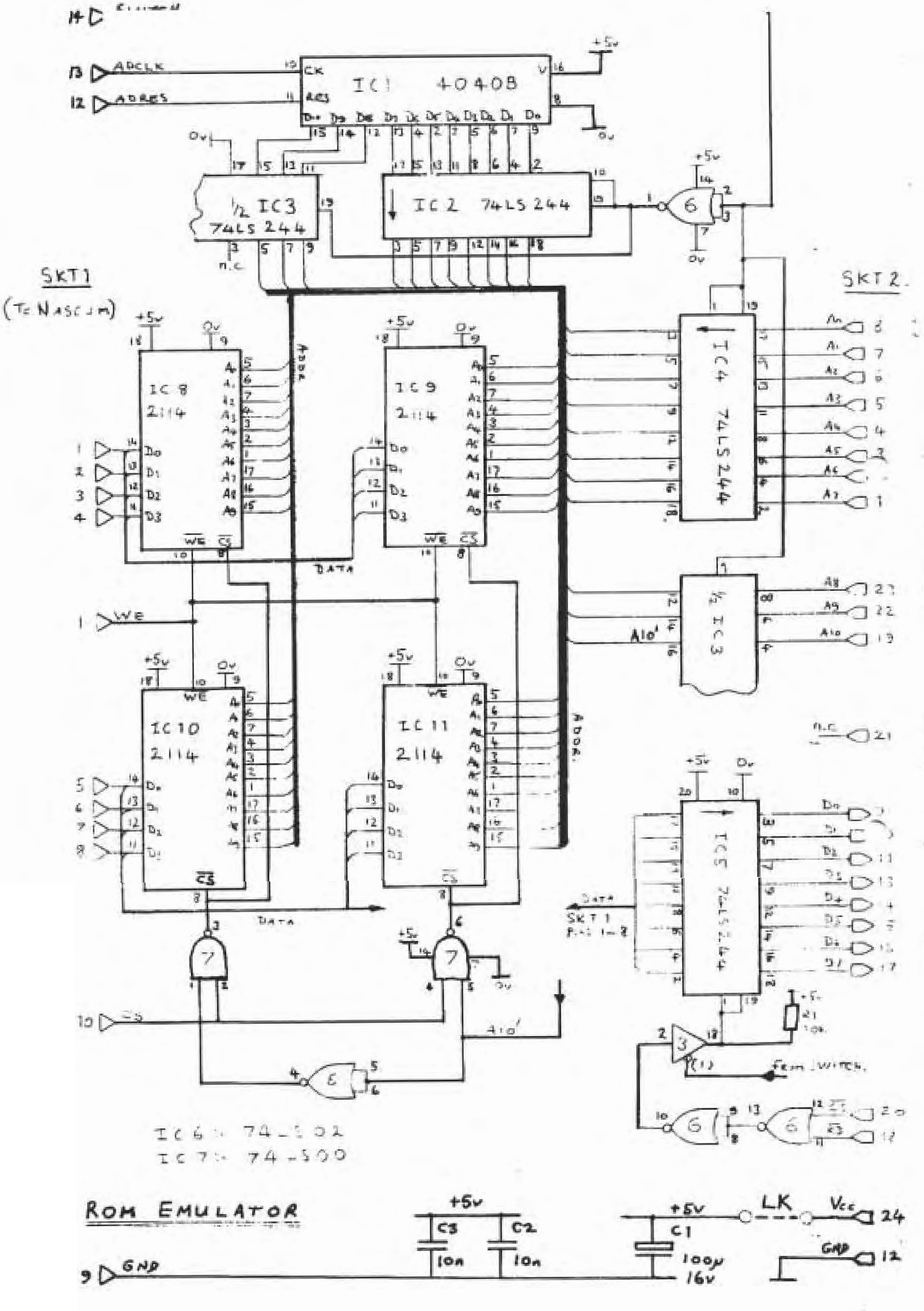
6. The WE signal is taken LOW and the CS signal HIGH to load the first byte.
7. The counter reset is removed and the ADCLK line is taken HIGH and then returned LOW to advance the address count.
8. The next byte is fetched and loaded and so on until the full 2K bytes are transferred.

A similar sequence may be employed to read back the contents of the emulator RAM into the host Nascom.

To use the emulator to develope hardware, short test routines may be written using ZEAP, tested by single-stepping if necessary and then re-assembled with origin 0 but located in the Nascom RAM at 4000H by means of ZEAP's P option. Executing the loader routine will then transfer the program into the emulator. The target hardware may then be powered up and its own processor reset. The target hardware will then be able to access the program as if it were in EPROM. The program may readily be changed until the hardware has been thoroughly tested. The final program can then be written and debugged using the same procedure and then eventually blown into an EPROM.

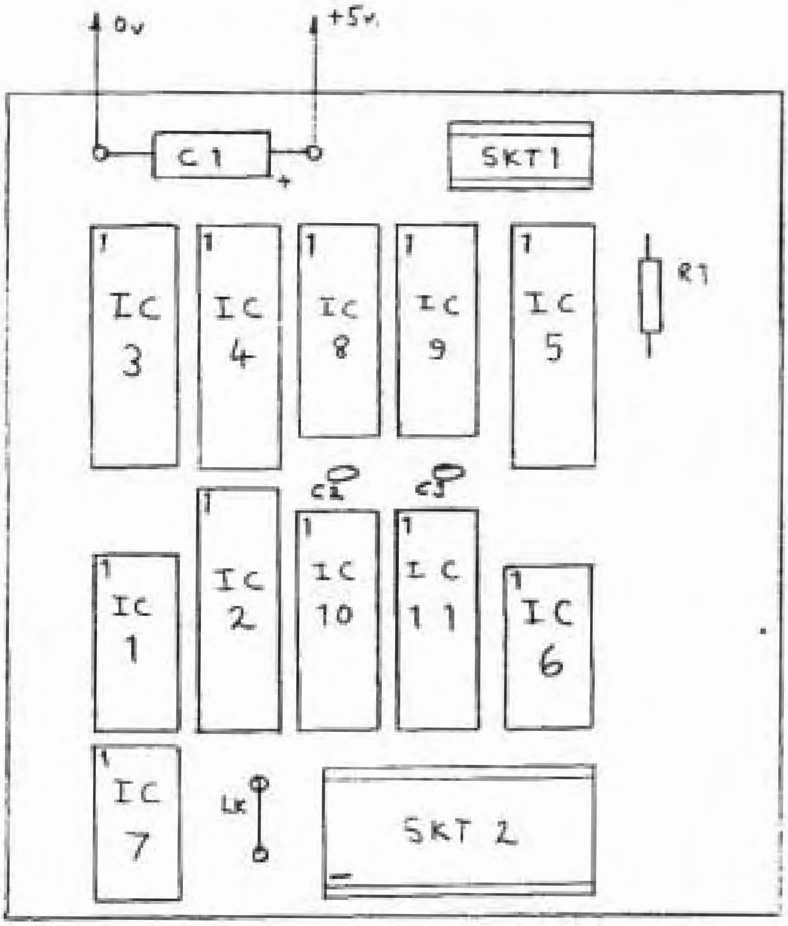
General Points.

The emulator as described has been in use now for some months and has helped to develope several small Z80 based projects. Although its minimal nature does not offer the sophisticated facilities of more expensive devices, I have found it to be a great time saver and now regard it as invaluable for home projects. Better software is definitely required. This will be a future aim.

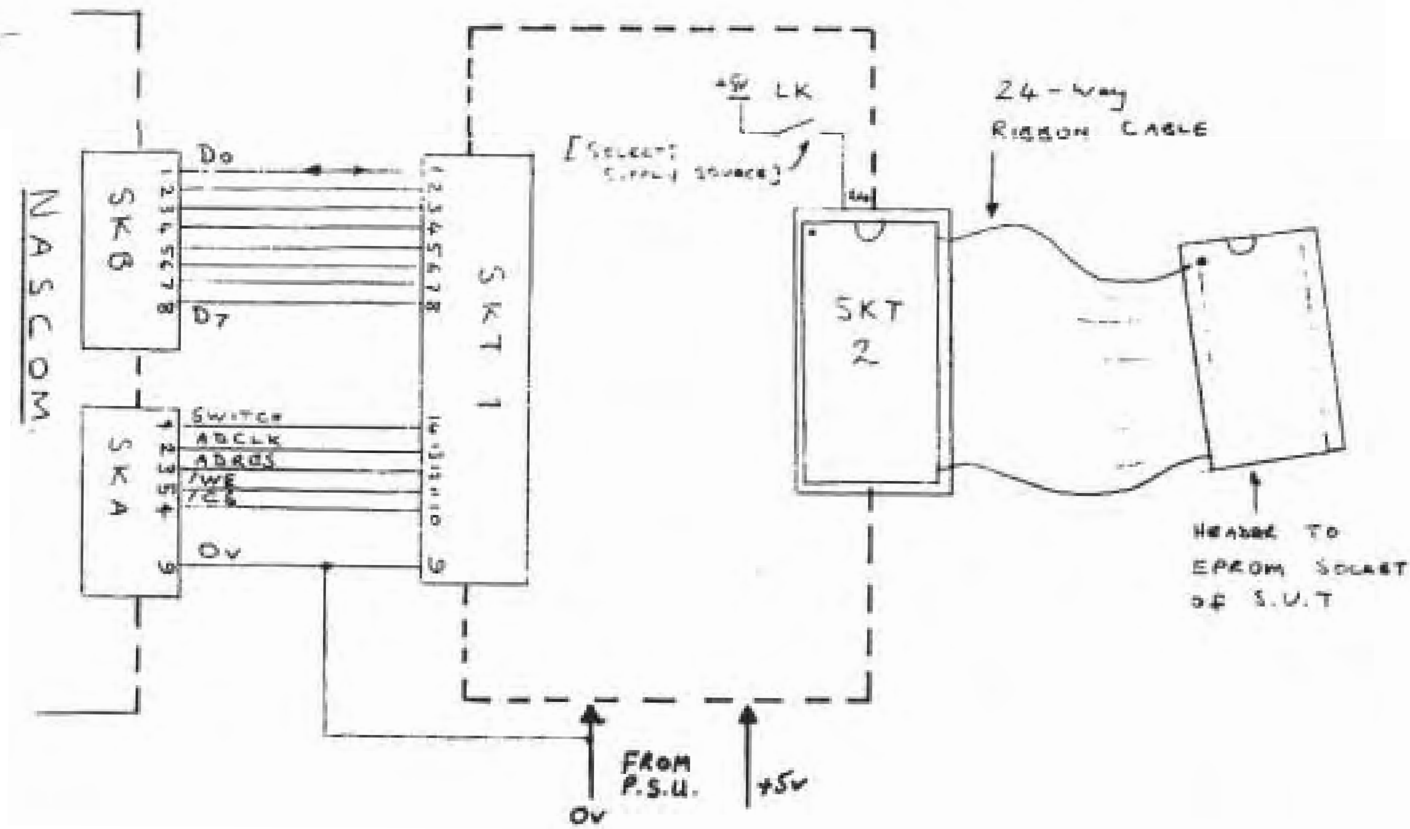


ROM EMULATOR

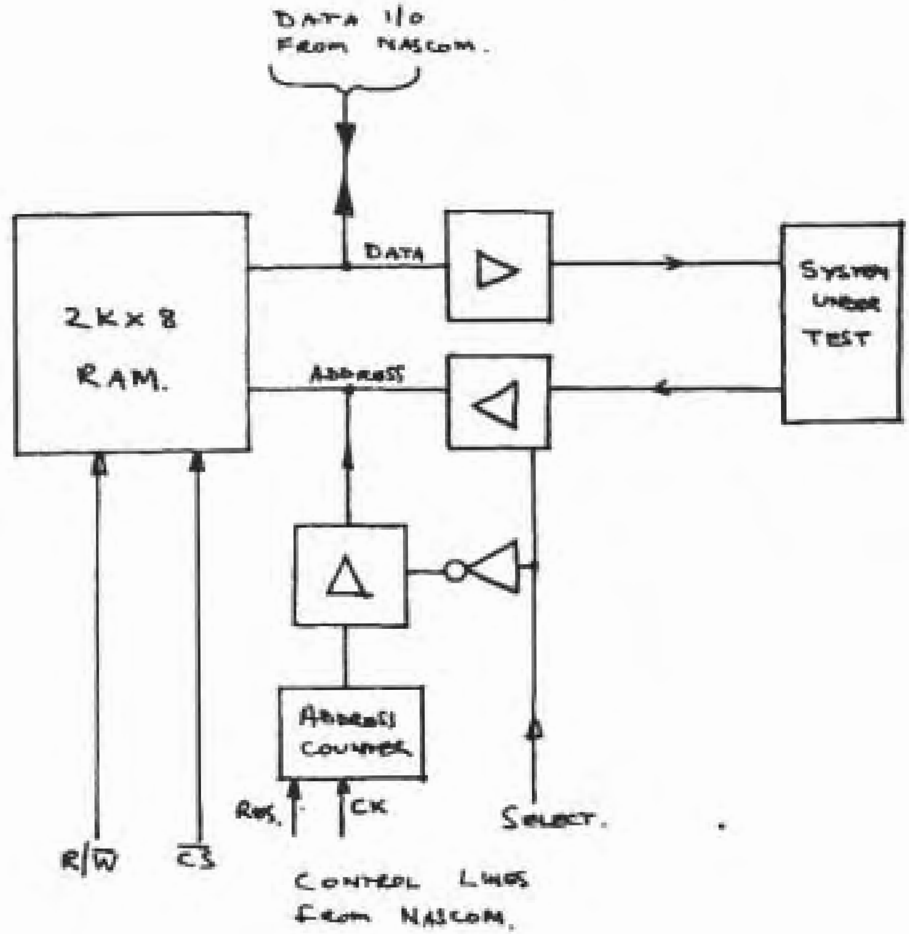
EMULATOR LAYOUT



SYSTEM CONNECTIONS



PRINCIPLE OF ROM EMULATOR FOR NASCOM.



PARTS LIST:-

IC 1	CD 4040B,	CMOS COUNTER.	+ 16-pin Socket
IC 2	74LS 244,	TRI-STATE NON-INV. BUFFER	+ 20 - " "
IC 3	"	"	" "
IC 4	"	"	" "
IC 5	"	"	" "
IC6	74LS02	QUAD. NOR GATE.	+ 14 - " "
IC7	74LS00	" NAND GATE.	" "
SKT1	16-PIN DIL SKT.		
SKT2	24 - " " "		
C1	100 μf. 16v. ELECTROLYTIC CAPACITOR.		
C2, C3	10nf. CERAMIC CAPACITOR.		
R1	10K 1/4 w. RESISTOR.		
1 off	24 pin HEADER → 24-pin HEADER CABLE, 12" LONG.		

```
                    ;***ROUTINE TO WRITE TO THE EMULATOR***
                    ;
                          ORG   00000
0000 3E 0F          START LD    A,00F     ;SET PIO TO OUTPUT
0002 D3 06                OUT   (6),A     ;PORT A SET TO OUTPUT
0004 D3 04                OUT   (4),A     ;00F TO RESET EMULATOR
0006 D3 07                OUT   (7),A     ;PORT B SET TO OUTPUT
0008 21 00 40             LD    HL,04000  ;INITIALISE DATA POINTER
000B 01 00 08             LD    BC,00800  ;INITIALISE BYTE COUNTER
000E 3E 0B                LD    A,00B     ;RESET ADDRESS COUNTER
0010 D3 04                OUT   (4),A     ;OUTPUT TO EMULATOR
                    ;
0012 7E             NEXT  LD    A,(HL)    ;GET BYTE FROM MEMORY
0013 D3 05                OUT   (5),A     ;SEND IT
0015 3E 03                LD    A,003     ;WRITE ENABLE
0017 D3 04                OUT   (4),A     ;SEND IT
0019 3E 13                LD    A,013     ;ENABLE EMULATOR RAM
001B D3 04                OUT   (4),A     ;SEND IT
001D 3E 1B                LD    A,01B     ;WRITE ENABLE HIGH
001F D3 04                OUT   (4),A     ;SEND IT
0021 3E 0B                LD    A,00B     ;DISABLE EMULATOR RAM
0023 23                   INC   HL        ;INCREMENT DATA POINTER
0024 3E 09                LD    A,009     ;INCREMENT ADDRESS PTR
0026 D3 04                OUT   (4),A     ;WITH A LOW TO HIGH EDGE
0028 00                   NOP             ;WAIT
0029 3E 0B                LD    A,00B     ;ADCLK BACK HIGH
002B D3 04                OUT   (4),A     ;SEND IT
002D 0D                   DEC   C         ;DECREMENT BYTE COUNTER
002E 20 E2                JR    NZ,NEXT   ;LOOP
0030 05                   DEC   B         ;
0031 20 DF                JR    NZ,NEXT   ;LOOP
                    ;
0033 3E 0F                LD    A,00F     ;RESET CONDITION IN A
0035 D3 04                OUT   (4),A     ;SEND IT
0037 3E 7F                LD    A,07F     ;SET PIO TO INPUT
0039 D3 07                OUT   (7),A     ;
003B 3E 1E                LD    A,01E     ;SET SWITCH LOW
003D D3 04                OUT   (4),A     ; TO ENABLE EMULATOR
003F DF 5B                DEFB  0DF,05B   ;RETURN TO MONITOR
```

Doctor Knowall Remembers...

The other day, I was caught short for something to read while trying to beat my Rubics cube record of 45 minutes. So, while manipulating my cube with the right hand, I picked up the nearest thing handy... Can you quess what it was ? No, it was the Basic manual. I was suprised to find that basic provides no less than 18 error messages. Of course, I had no seen any of these in years, but I was suprised to find a function that I had almost forgotten about! For some reason, I couldn't remember ever using it...and I doubt if anyone else has ever used POS either.

POS() returns the 'X' position of the printhead and because printers don't usually backspace or have full cursor movement, POS() will not necessarily return an 'X' value corresponding to the cursor position on the screen. Cursor and printhead are independent creatures. This can cause problems if POS() is used in input routines, but POS can be useful in output routines:

```
10 REM OUTPUT DATA, IN DATA LIST FORMAT
20 L=1000: REM LINE ON WHICH LIST IS TO START
30 FOR ADDR=START TO FIN STEP 2
40 IF POS(0)=0 THEN PRINT L;"DATA";:L=L+10
50 IF DEEK(ADDR)<0 THEN PRINT " ";
60 PRINT STR$(DEEK(ADDR));",";
70 IF POS(0)>40 THEN PRINT CHR$(8): REM GET RID OF LAST COMMA
80 NEXT ADDR:END
```

Run it, break when you have a screenful of data and enter each DATA line displayed. Run it again if there is more data to follow. Don't anyone mention USR!

Doctor Knowall's Definitive Guide to INKEY$ without USRs.

USRs are people who run versions of Pascal written in Forth, which in turn are written in Basic! In short, USRs don't like Basic. For the benefit of purists struggling along without an INKEY$ function, I plunge into the depths of my immense pool of knowledge ... and find not one, but two ways of detecting single key presses from Basic.

The Quick and VERY Simple way ...

If you don't mind using a limited range of keys, the keyboard can be read directly from port 0 using the INP() function. Because of timing, only the first row of keys can be read this way. The following program shows which keys are used and the bits that are assigned to them.

```
1000 KEY$(0)="(BACKSPACE)"
1010 KEY$(1)="(ENTER)"
1020 KEY$(2)="(=)"
1030 KEY$(3)="(CONTROL)"
```

```
1040 KEY$(4)="(SHIFT)"
1050 KEY$(5)="(@)"
1060 KEY$(6)="(HOME)"
1070 KEYROW=255-INP(0):REM invert bits read from port 0
1080 IF KEYROW=0 THEN 1070
1090 FOR BIT=0 TO 6
1100 IF 2 BIT AND KEYROW THEN PRINT KEY$(BIT);
1110 NEXT BIT
1120 PRINT:GOTO 1070
```

When all that is required is for the program to wait until Enter or Newline is pressed, you only need the following line :

```
10 IF INP(0) AND 2 THEN 10
```

This will loop until Enter is pressed.

... and a naughty way.

The Basic interpreter must be modified if a genuine INKEY function is required. The Microsoft Basic that the Nascom uses was adapted from a program written in 8080 machine code. The 8080 doesn't have an 'IN r,(C)' instruction, so a small skeleton subroutine is used instead. The subroutine looks like this:

```
103E  DB  ;IN (n)
103F  ..  ;port number inserted here by INP() function
1040  C9  ;return
```

This resides in RAM, as can be seen from the addresses. If DBH is changed to DFH, a Nas-Sys RST SCAL will be executed. The monitor subroutine called being dependent on the number passed by the INP() function. Any subroutine that doesn't require an argument can be used. The useful routines and their numbers are:

```
98  SCAL IN
123 SCAL BLINK
93  SCAL TDEL
```

Example

```
10 POKE 4158,223
20 KEY$=CHR$(INP(98)): REM or INP(123)
30 IF KEY$>"z" THEN 20
40 PRINT KEY$
50 IF KEY$<>CHR$(13) THEN 20
60 DELAY=INP(93)
70 POKE 4158,219: REM restore normal use of INP() fuction
```

Line 30 is required when using SCAL IN as this routine returns values greater than 128 when no key is pressed.

Well, you now have no excuse for ever being a USR again! The two ways of detecting a key press from Basic are so incredibly obvious, I can't imagine why no one else has ever thought of them before!

Cheap Printout 1

by Mark Horsman

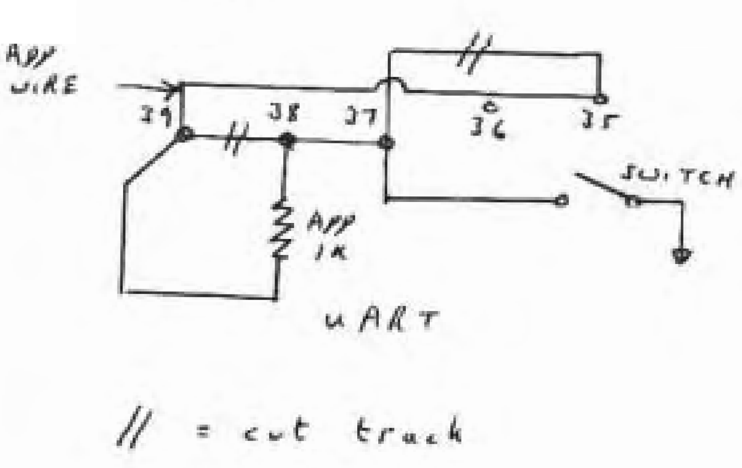
Having the usual need for cheap hardcopy, I bought myself a Creed 75R for about £20.

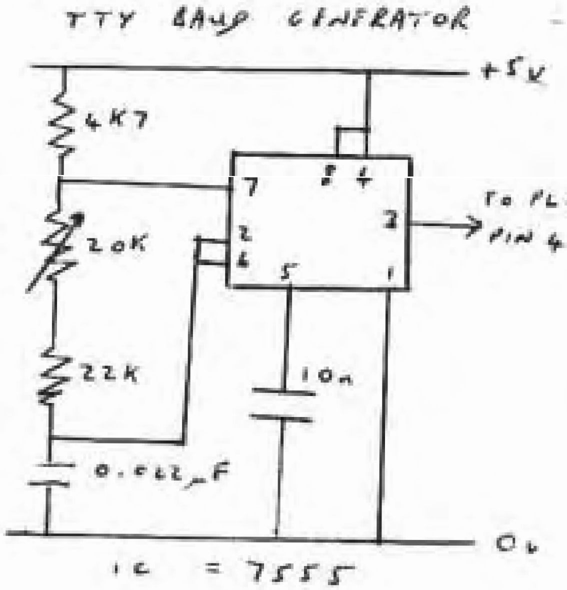
"Great.", I thought, "Now how do I drive the thing?"

Personal Computer World, May 80 and Practical Computing, July 80 both had articles using one bit of a parallel port to output the 5 bit Baudot code, shifting one bit out at a time and generating the start and stop bits. Thsi meant various timing loops that needed trimming, seemingly depending on the weather, to produce the correctly sync'ed output. This, I thought, was not the most elegant solution, so I looked at the UART. In the Nascom 2 this is configured for 8 bit use only but is capable of 5, 6, 7 and 8 bit use depending on the voltages on pins 37 and 38. So, surgery is called for. Cut the track between pins 39 and 38, cut the track between pins 37 and 35, join pins 39 and 35 with a short length of fine wire and add a 1K resistor between pin 1 and pins 37 and 38. This leaves us still in 8 bit mode. Now add a switch between pin 38 and ground such that when open we have 8 bits but when closed we have 5 bit mode, Next we need a circa 16 x 50Hz clock to drive the UART. This is generated with a 7555 (CMOS 555 timer) using a multiturn pot to trim the frequency. This is wired to PL2 pin 4 or TP4. The output is taken from PL2 pin 12 (20mA Loop Out).

Finally, the software. This uses a lookup table, a flag to determine whether letters or figures last sent and a routine to convert such as > to .GT., etc.

P.S. If you live in a terraced house, move - the noise is horrendous.





```
                   ;***CREED 75R PRINTER DRIVER***
                   ;
                   FIGS    EQU     01B
                   LETS    EQU     01F
                   SCAL    EQU     018
                   SRLX    EQU     06F
                   ;
                           ORG     00D80
0D80 F5            CREED   PUSH    AF          ;SAVE REGISTERS
0D81 DD E5                 PUSH    IX
0D83 C5                    PUSH    BC
0D84 E5                    PUSH    HL
0D85 DD 21 D9 0D           LD      IX,FLAG     ;CHARACTER FLAG
0D89 DD 4E 00              LD      C,(IX+0)
0D8C CB BF                 RES     7,A         ;CHANGE ANY GRAPHICS
0D8E 26 0D                 LD      H,00D
0D90 6F                    LD      L,A
0D91 7E                    LD      A,(HL)
0D92 CB 77                 BIT     6,A         ;IF CR OR LF NO SHIFT
0D94 20 2E                 JR      NZ,OPCRLF;  NEEDED
0D96 CB 7F                 BIT     7,A         ;LETTER OR FIGURE
0D98 28 1F                 JR      Z,LOUT
0D9A CB 41                 BIT     0,C
0D9C 20 13                 JR      NZ,ICOUT    ;IF SAME SHIFT THEN
0D9E F5                    PUSH    AF          ; SEND
0D9F 3E 1B                 LD      A,FIGS      ;IF NOT SEND SHIFT
0DA1 0E 01                 LD      C,001       ; CHARACTER
0DA3 DF            FCOUT   RST     SCAL
0DA4 6F                    DEFB    SRLX
0DA5 DD 71 00              LD      (IX+0),C    ;PUT SHIFT INTO FLAG
0DA8 F1                    POP     AF
0DA9 CB 6F                 BIT     5,A         ;IF <OR> SEND EXTRA T
0DAB 28 04                 JR      Z,ICOUT
0DAD DF                    RST     SCAL
0DAE 6F                    DEFB    SRLX
0DAF 3E 10                 LD      A,010       ;T
0DB1 DF            ICOUT   RST     SCAL
0DB2 6F                    DEFB    SRLX
0DB3 E1            POP     POP     HL
0DB4 C1                    POP     BC
0DB5 DD E1                 POP     IX
0DB7 F1                    POP     AF
0DB8 C9                    RET
0DB9 CB 41         LOUT    BIT     0,C         ;IF WRONG SEND SHIFT
0DBB 28 F4                 JR      Z,ICOUT     ;ELSE CHARACTER
0DBD F5                    PUSH    AF
0DBE 3E 1F                 LD      A,LETS
0DC0 0E 00                 LD      C,00
0DC2 18 DF                 JR      FCOUT
0DC4 FE 44         OPCRLF  CP      044         ;SPACE
0DC6 28 E9                 JR      Z,ICOUT
0DC8 3E 02                 LD      A,002       ;SEND CR,LF
0DCA DF                    RST     SCAL
0DCB 6F                    DEFB    SRLX
```

NAS-SEM - Nascom Z80 assembler Page 0001

```
0DCC 3E 08       0054       LD     A,008
0DCE DF          0055       RST    SCAL
0DCF 6F          0056       DEFB   SRLX
0DD0 AF          0057       XOR    A
0DD1 06 02       0058       LD     B,002
0DD3 DF          0059 SOUT  RST    SCAL      ;SEND NULLS FOR MECH.
0DD4 6F          0060       DEFB   SRLX
0DD5 10 FC       0061       DJNZ   SOUT
0DD7 18 DA       0062       JR     POP
0DD9             0063 FLAG  DEFS   0
```

Cheap Printout 2

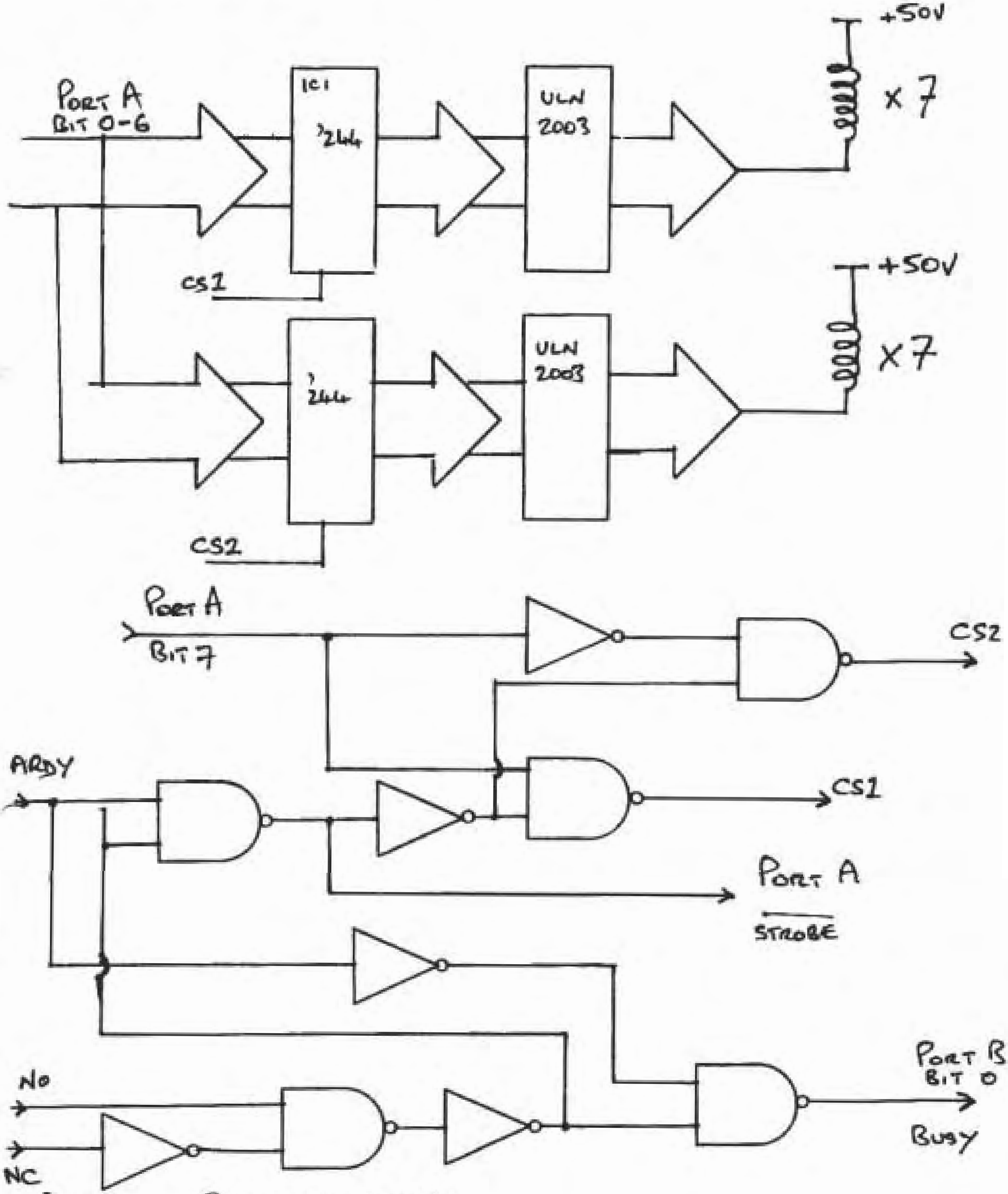
by Mark Horsman

While an old Creed 75R proved adequate for listings etc, it was hardly suitable for letters and reports. Therefore, seeing an ad. in Wireless World for IBM printers, I trekked to Essex and bought myself an old 3982 Terminal printer. I also received with it some suggestions on interfacing. These struck me as somewhat crude and rather incomplete, having no software with them. So, I set about designing my own.

Firstly, the hardware. Octal buffers to switch between characters and control and then 7-way Darlington drivers (ULN2003) to drive the operating coils. The timing contacts on my machine where not all c/o as per the handouts. So, they are just inverted as necessary and added to ARDY to generate a Busy signal. An enable signal for the octal buffers is derived and switched with bit 7 to determine control or character. The TTL is powered from the Nascom and the coils from a simple 48V ps. The worst part is stripping out and identifying all the wiring, then stuffing it back neatly.

Secondly, the software driver. (The nut behind the wheel). This is based on a look up table. Nulls are returned unused. A linefeed after a CR is also thrown away. Full stops are sent either case to improve speed. Character case is compared with the printer and any appropriate change sent. Finally, a routine for printer busy is ended with a short delay which I found necessary to ensure complete settling after Busy is finished.

IBM INTERFACE.



PRINT CONTACTS RETURNED TO +5V VIA 1K RESISTORS.

```
                      0000 ;***IBM PRINTER DRIVER***
                      0001 ;
                      0002          ORG     0AD00
                      0003 ;TABLE OF IBM CODES
AD00 00 BA BA BA 0004          DEFB    000,0BA,0BA,0BA ;000
AD04 BA BA BA BA 0005          DEFB    0BA,0BA,0BA,0BA ;004
AD08 10 BA 04 BA 0006          DEFB    010,0BA,004,0BA ;008
AD0C 04 0B BA BA 0007          DEFB    004,00B,0BA,0BA ;00C
AD10 BA BA BA BA 0008          DEFB    0BA,0BA,0BA,0BA ;010
AD14 BA BA BA BA 0009          DEFB    0BA,0BA,0BA,0BA ;014
AD18 BA BA BA BA 0010          DEFB    0BA,0BA,0BA,0BA ;018
AD1C BA BA BA BA 0011          DEFB    0BA,0BA,0BA,0BA ;01C
AD20 20 BA 96 D6 0012          DEFB    020,0BA,096,0D6 ;020
AD24 DB 85 D7 16 0013          DEFB    0DB,085,0D7,016 ;024
AD28 D0 D4 DA DF 0014          DEFB    0D0,0D4,0DA,0DF ;028
AD2C 03 20 BA DB 0015          DEFB    003,020,0BA,0DB ;02C
AD30 54 5F 5A 5B 0016          DEFB    054,05F,05A,05B ;030
AD34 55 56 52 57 0017          DEFB    055,056,052,057 ;034
AD38 53 50 87 07 0018          DEFB    053,050,087,007 ;038
AD3C BA BA BA B3 0019          DEFB    0BA,0BA,0BA,0B3 ;03C
AD40 D5 B3 C0 C3 0020          DEFB    0D5,0B3,0C0,0C3 ;040
AD44 C7 C6 8B 8F 0021          DEFB    0C7,0C6,08B,08F ;044
AD48 C4 92 81 C2 0022          DEFB    0C4,092,081,0C2 ;048
AD4C C5 9F CA B5 0023          DEFB    0C5,09F,0CA,0B5 ;04C
AD50 86 82 9D B4 0024          DEFB    086,082,09D,0B4 ;050
AD54 C1 CB 9B 90 0025          DEFB    0C1,0CB,09B,090 ;054
AD58 CF B4 D1 BA 0026          DEFB    0CF,0B4,0D1,0BA ;058
AD5C BA BA BA A0 0027          DEFB    0BA,0BA,0BA,0A0 ;05C
AD60 16 33 40 43 0028          DEFB    016,033,040,043 ;060
AD64 47 46 0B 0F 0029          DEFB    047,046,00B,00F ;064
AD68 44 12 21 42 0030          DEFB    044,012,021,042 ;068
AD6C 45 1F 4A 35 0031          DEFB    045,01F,04A,035 ;06C
AD70 06 02 1D 34 0032          DEFB    006,002,01D,034 ;070
AD74 41 4B 1B 10 0033          DEFB    041,04B,01B,010 ;074
AD78 4F 04 51 BA 0034          DEFB    04F,004,051,0BA ;078
AD7C 87 BA BA BA 0035          DEFB    087,0BA,0BA,0BA ;07C
                      0036 ;
                      0037 ; ENTRY POINT - SAVE REGS.
                      0038 ;
AD80 F5               0039          PUSH    AF
AD81 C5               0040          PUSH    BC
AD82 E5               0041          PUSH    HL
AD83 DD E5            0042          PUSH    IX
                      0043 ;
                      0044 ;THROW AWAY NULLS
                      0045 ;
AD85 B7               0046          OR      A
AD86 28 6E            0047          JR      Z,POP
                      0048 ;
                      0049 ;WHERE AM I? STACK HOLDS THIS ADDRESS
AD88 D7 00            0050          RCAL    WAI
AD8A E1               0051 WAI      POP     HL
AD8B 2E 00            0052          LD      L,00
                      0053 ;
```

```
                    0054 ;PUT IT IN IX AS WELL
                    0055 ;
AD8D E5             0056        PUSH   HL
AD8E DD E1          0057        POP    IX
                    0058 ;
                    0059 ;CONTROL OR CHARACTER
                    0060 ;
AD90 FE 21          0061 CP21   CP     021
AD92 30 1C          0062        JR     NC,CHAR
                    0063 ;
                    0064 ;FETCH IBM CODE
                    0065 ;
AD94 6F             0066        LD     L,A
AD95 7E             0067        LD     A,(HL)
                    0068 ;
                    0069 ;IS IT CR ?
                    0070 ;
AD96 FE 08          0071 CPCR   CP     008
AD98 20 06          0072        JR     NZ,CPLF
AD9A DD CB 00 C6    0073        SET    0,(IX+0)
AD9E 18 4C          0074        JR     OP
                    0075 ;
                    0076 ;IS IT LF ?
                    0077 ;
ADA0 FE 04          0078 CPLF   CP     004
ADA2 20 44          0079        JR     NZ,OP1
ADA4 DD CB 00 46    0080        BIT    0,(IX+0)
ADA8 28 3E          0081        JR     Z,OP1
                    0082 ;
                    0083 ;IF LAST CHAR CR THEN THROW AWAY LF
                    0084 ;
ADAA DD CB 00 86    0085        RES    0,(IX+0)
ADAE 18 46          0086        JR     POP
                    0087 ;
                    0088 ;CONVERT CHARACTERS
                    0089 ;
ADB0 CB 7F          0090 CHAR   BIT    7,A
                    0091 ;
                    0092 ;IF BIT 7 SET THEN NON-PRINTING
                    0093 ;
ADB2 28 02          0094        JR     Z,GETIBM
ADB4 3E 2E          0095        LD     A,"."
                    0096 ;
                    0097 ;FETCH IBM CODE
                    0098 ;
ADB6 6F             0099 GETIBM LD     L,A
ADB7 7E             0100        LD     A,(HL)
                    0101 ;
                    0102 ;IF FULL STOP SEND EITHER CASE
                    0103 ;
ADB8 FE 8A          0104        CP     0BA
ADBA 28 2C          0105        JR     Z,OP1
                    0106 ;
                    0107 ;UPPER OR LOWER CASE ?
```

```
                    0108 ;
ADBC CB 7F          0109        BIT     7,A
ADBE 20 14          0110        JR      NZ,UC
                    0111 ;
                    0112 ;I/P PRINTER CASE
                    0113 ;
ADC0 F5             0114 LC     PUSH    AF
ADC1 DB 05          0115 LC1    IN      A,(5)
                    0116 ;
                    0117 ;IF WRONG SEND LC
                    0118 ;
ADC3 CB 4F          0119        BIT     1,A
ADC5 28 0A          0120        JR      Z,SENDL
ADC7 D7 33          0121        RCAL    RDY
ADC9 3E 02          0122        LD      A,2
ADCB D3 04          0123        OUT     (4),A
ADCD D7 2D          0124        RCAL    RDY
                    0125 ;
                    0126 ;CHECK CASE CHANGED
                    0127 ;
ADCF 18 F0          0128        JR      LC1
                    0129 ;
                    0130 ;CASE CORRECT, RECOVER CHAR AND SEND
ADD1 F1             0131 SENDL  POP     AF
ADD2 18 12          0132        JR      SENDCH
                    0133 ;
                    0134 ;UPPER CASE CHARS.
                    0135 ;
ADD4 F5             0136 UC     PUSH    AF
ADD5 DB 05          0137 UC1    IN      A,(5)
ADD7 CB 4F          0138        BIT     1,A
                    0139 ;
                    0140 ;CHECK CASE
                    0141 ;
ADD9 20 0A          0142        JR      NZ,SENDU
ADDB D7 1F          0143        RCAL    RDY
                    0144 ;
                    0145 ;IF WRONG SEND UC CHAR
                    0146 ;
ADDD 3E 01          0147        LD      A,001
ADDF D3 04          0148        OUT     (4),A
ADE1 D7 19          0149        RCAL    RDY
                    0150 ;
                    0151 ;CHECK CASE CHANGED
                    0152 ;
ADE3 18 F0          0153        JR      UC1
                    0154 ;
                    0155 ;IF CASE CORRECT SEND CHAR
                    0156 ;
ADE5 F1             0157 SENDU  POP     AF
                    0158 ;
                    0159 ;BIT 7 SET EQUALS CHAR
                    0160 ;NOT SET EQUALS CONTROL
                    0161 ;
```

```
ADE6 CB FF           SENDCH SET    7,A
                     ;
                     ;LF LAST CHAR. FLAG
                     ;
ADE8 DD CB 00 86     OP1    RES    0,(IX+0)
ADEC D7 0E           OP     RCAL   RDY
ADEE D3 04                  OUT    (4),A
ADF0 FE 0B                  CP     00B       ;WAS CHAR A CR ?
ADF2 20 02                  JR     NZ,POP    ;IF SO, WAIT A WHILE
ADF4 D7 06                  RCAL   RDY
ADF6 DD E1           POP    POP    IX
ADF8 E1                     POP    HL
ADF9 C1                     POP    BC
ADFA F1                     POP    AF
ADFB C9                     RET
                     ;
                     ;WAIT FOR PRINTER CYCLE
                     ;
ADFC F5              RDY    PUSH   AF
ADFD DB 05           RDY1   IN     A,(5)
ADFF CB 47                  BIT    0,A
AE01 20 FA                  JR     NZ,RDY1
AE03 AF                     XOR    A
AE04 06 0E                  LD     B,14
AE06 FF              RDY2   RST    038
AE07 10 FD                  DJNZ   RDY2
AE09 F1                     POP    AF
AE0A C9                     RET
                     ;
                     ;INITIALISE PIO
                     ;
AE0B F5                     PUSH   AF
AE0C 3E 0F                  LD     A,00F
AE0E D3 06                  OUT    (6),A
AE10 3E 4F                  LD     A,04F
AE12 D3 07                  OUT    (7),A
AE14 F1                     POP    AF
AE15 C9                     RET       ;RETURN TO CALLING PROGRAM
                     ;      RST    018    ; OR MONITOR
                     ;      DEFB   05B
```

Gener-80 Review

by IJC

Gener-80 is a Z80-Assembler/Editor package for the Nascom from Seven Stars Publishing. The package is supplied on tape along with an A4 photocopied manual which, although not comprehensive, is adequate.

The program is 6.75K long and the copy I was sent loaded with only a slight volume adjustment to my recorder. The minimum system requirements are a Nascom (1 or 2) with Nas-Sys and Cottis Blandford equivalent interface and 8K or more RAM starting from 1000H. A 300 baud copy is also supplied on the tape. The program loads at 1000H which means that there are no problems transferring it to disc. When executed, the program does a self-check and if all is OK it comes up with a sign-on message and proceeds to ask about workspace. The manual is not very clear about how to define your workspace size. You need to define a source (SCE) area and an object (OBJ) area. The purpose of this is to restrict the code to these areas. The source area can be anywhere in memory that is RAM and does not overlap with Gener-80 or the object area. Likewise for the object area. The manual gives the example of £D00 £DFF for the source area which gives the impression that it is purely workspace for temporary storage etc. In practice it is the source code area and therefore wants to be as big as possible in most cases. Likewise for the object area.

Once these have been designated, you are asked for an offset. This is used during assembly in the expected way to 'offset' the object code.

Once the program is happy with these values, you are presented with a menu.:-

```
COMMAND? A  D  E  L  L?  M  O  P  PA  PAT  S
```

This gives you the option of:-

- A - assemble the source program
- D - delete the source program
- E - edit the source program
- L - load a source program from tape
- L? - verify a source program on tape
- M - move a block of source program
- O - object workspace and assembler offset re-definition
- P - print source program

PA - print an assembled source program
PAT - print an assembled source program with sorted symbol table
S - save source program to tape

The editor mode may be entered either just with E or E followed by a string. Movement about the source file is via the cursor keys and various combinations of control keys to move forward and backward by lines, pages and the complete text. Lines and characters may obviously be deleted and inserted. Points that I found un-natural were that only text on a line up to the cursor were entered and if a line began with a space, an error was flagged and the cursor disappears which looks remarkably like a program crash. It was only after frantic searching of the manual that I found that CS has to be typed before you get a cursor back after an error is flagged.

As mentioned previously, each line of source must begin in character position 1 which means that the listing looks, in my opinion, a bit untidy and not too clear.

The D(elete), M(ove), P(print source), PA(ssembled source) and S(ave to tape) commands all allow the use of optional labels which enable only sections of the source to be worked on. A very useful touch for the print and save commands.

The assembler handles all the Z80 nmenonics plus the usual DEFB, DEFW, DEFS, DEFM, END, EQU and ORG. Labels are up to 6 alphanumerics starting with a letter and must be followed by a colon and a space. Numbers are either decimal or hex (preceded by £) and are equated to 16 bits.

Gener-80 is cassette based and consequently all text is loaded/saved to tape. No technical data is given in the manual to help anyone who wanted to make the assembler disc based. It is also set up for the default Nascom serial printer but, as explained in the manual, it goes via 0C78 so can easily be intercepted and re-directed to your own print routine.

All in all Gener-80 is an adequate assembler package. It does not go for any of the fancy frills like macros etc but as an assembler it does its job well. I think that more technical information is needed for it in this day and age to make it easier for people to patch in their own routines for additional pseudo-mnemonics and I/O. All in all, simple but adequate.

Gener-80 is available from Seven Stars Publishing who hail from 15 Gloucester Avenue, London NW1 7AU.

XTAL IF/THEN/ELSE

by Robert Gill

The following provides a routine for XTAL Basic to allow IF/THEN/ELSE structures to be used :-

```
1283 01 30                          ;HTEXT

0E80 C5 4C 53 45

0F80 1C 2D

1966 CA 01 2D                       ;divert main IF/THEN routine

2D01 3E 00        LD A,0
2D03 23           INC HL
2D04 BE           CP (HL)           ;test of end of line
2D05 CA 1B 2D     JP Z,2D1B
2D08 2F           CPL
2D09  BE            CP   (HL)          ;test   for   user-defined
reserved word
2D0A CA 11 2D     JP Z,2D11
2D0D 2F           CPL
2D0E C3 03 2D     JP 2D03
2D11 23           INC HL
2D12 3E 80        LD A,080          ;this can be changed if ELSE
2D14 BE           CP (HL)           ; command not represented by
2D15 CA 69 19     JP Z,1969         ; the two byte token FF 080
2D18 C3 01 2D     JP 2D01
2D1B 2B           DEC HL
2D1C C3 87 18     JP 1887
```

The routine is simple and only contains a couple of minor bugs:-

1. IF <EXPR> THEN REM ..... :ELSE .....
The routine ignores REM if the IF/THEN proves false and an ELSE statement follows.

2. IF <EXPR> THEN PRINT "xy"
If a REM or PRINT statement used after a THEN contains the characters of 0FF and 080, the routine will misinterpret that as an ELSE command.

The routine is fast causing a time delay under worst possible conditions of 1.3%.

Back Issues

We currently have copies of all the back issues of the Nascom Newsletter and Micropower magazines.

We are now in a position to sell these off at rock bottom prices to clear our shelves.

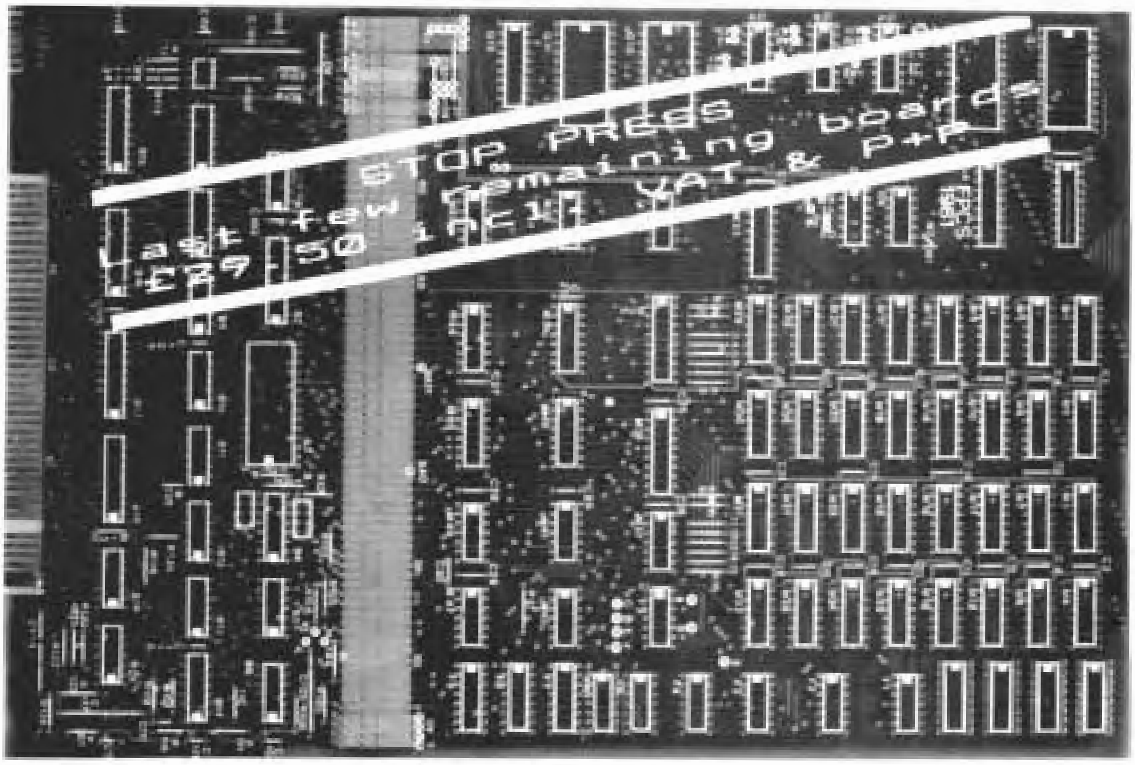
Volume 1 (Issues 1 to 4)

1 magazine 25p
5 magazines for £1

Volume 2 (Issues 1 to 6)

1 magazine 25p
5 magazines for £1

Our stocks are limited so order now for those missing issues.



64 KILOBYTE RAM and BUFFER CARD

with PROGRAMMABLE GRAPHICS

This 64K RAM card is suitable for the Nascom 1 or 2. The double sided glass-fibre P.C.B., 302 mms (12 ins.) by 203 mms (8 ins.), holds up to 4 blocks of 16 Kb dynamic RAM (4116). When all four blocks are fitted the whole of the Z80 address field is occupied by RAM. The on board mapper allows parts of this address field to be selectively inhibited in either read or write mode, or both. The mapper divides the address field into 4K blocks, and any two selected blocks can be further subdivided into 2 x 2K blocks.

The graphics section is entirely separate from the dynamic RAM, but it can be mapped in at any chosen 2K boundary. It can use an EPROM (2716) to give a pre-programmed character set, or static RAM (2 x 4118, or 6116) to provide user-programmable characters.

For the Nascom 2 the memory and graphics section can be separated from the "buffer" section; the resulting 8 x 8 card can be plugged into a standard Nasbus (80-bus) edge connector For the Nascom 1 the bottom 8 x 4 ins. section of the card provides full buffering between the Nascom 1 43-way connector and Nasbus. In addition the following extra facilities are also provided:-

1 Power-on jump; this allows the processor to execute a program at any preset 4K boundary on power-on or reset.
2 Synchronised Reset; the reset pulse is synchronised with the processor M1 cycles, to prevent corruption of data in dynamic RAM
3 Wait state generator; one wait state can be added to memory or input/output access
4 ROM socket; a 28 pin or 24 pin socket can be placed at position B3, and via a series of links this can accommodate a 2716, 2732, 2764 or the standard Nascom Basic ROM
5 Input/output; a partial decode is provided which allows for 64 input/output addresses.

The 64K RAM card is available now, price £39.50, from

MICRO POWER Ltd.,
8/8A, Regent Street, Leeds LS7 4PE
Tel. (0532) 683186
Please add 55p p/p and V.A.T. at 15%.