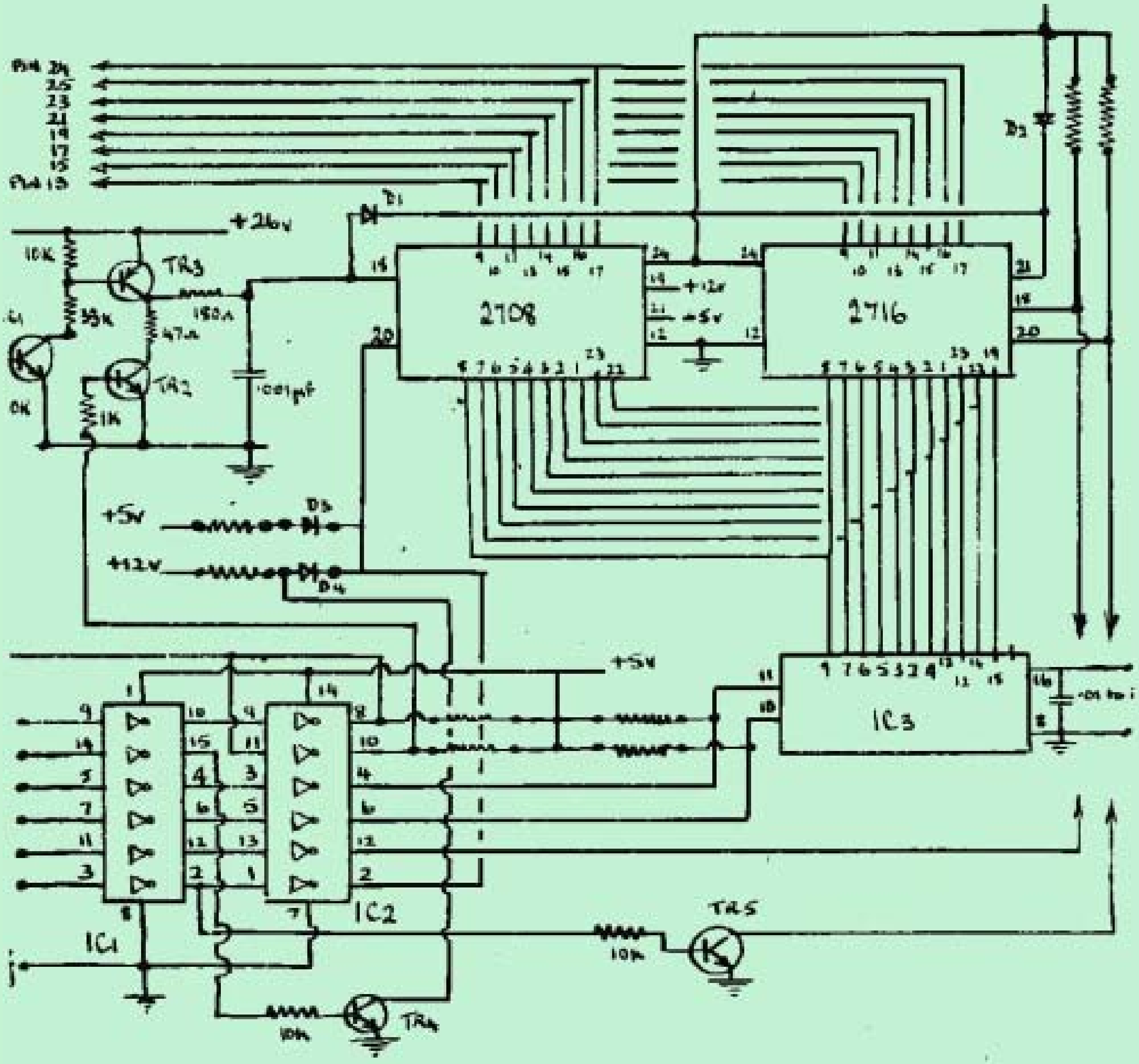
µP MICROPOWER VOLUME 1, NUMBER 4

A MAGAZINE FOR NASCOM USERS



December, 1981 95 p

PROGRAM POWER

NASCOM 1 & 2

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

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

Jailbreak in Space (16K/MC/G)

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

Graphic Golf (16K/B/G)

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

AY-3-8910 Programmable Sound Chip

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

Sound Chip Data Manual (60 pages)

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

Sound Chip Interfacing Board

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

Sound Chip Demo Program (MC)

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

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

ALL PROGRAMS SUPPLIED ON CASSETTE IN CUTS/KANSAS CITY FORMAT

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

PROGRAM POWER
5, Wensley Road
Leeds LS7 2LX.

CONTENTS

EDITORIAL

We must first apologise for the late appearance of this, the December issue of Micropower. New Year's Resolution – we must do better in 1982.

On the basis of the response from readers, and the amount of material we are being sent by contributors, we have decided to produce six issues in 1982 at approximately two-month intervals, the first issue to appear towards the end of February.

The price of Micropower will remain the sa me – 95p per issue, so a year's subscription will be £5.70. This includes postage and packing, but only for the British Isles. Please 65p per copy for Europe, £1.05 per copy for printed paper air mail to the rest of the world.

As always, we are still looking for contributors. Write about your pet projects, hardware or software – what they do, how they do it, how you would like to develop them in future.

There was a time in 1981 when it looked as though Nascom would disappear without trace. Now, with Lucas at the helm, 1982 should be the year when the Nascom system at last 'takes of' and justifies the faith that its many supporters have had in it.

EPROM PROGRAMMER / CHECKER / READER

by C. Bowden

The equipment described in this article is the third EPROM programmer that the author has built in the last couple of years. The first version, which was built about two years ago, worked via the serial interface and was very slow and unreliable. It took over an hour to 'burn' a 2708.

Version number 2, built about a year ago, worked through the Nascom PIO and was much faster, taking only a couple of minutes for the same task. It was also much more reliable, and the software included routines to copy EPROMS to RAM, and to verify them against RAM. One disadvantage, however, was that it would only program 2708 EPROMS, although the basic hardware was suitable for extension to 2K EPROMS, such as the 2716/2516.

The third version, described in this article will work with 2708, 2516 and 2716 EPROMS, and some of the routines can also be used with mask-programmed ROMS which are pin-compatible with 2716s, such as the ROMS used in the NASCOM 2 for Nas-Sys and Graphics. Two extra routines have been added, and the unit will now carry out the following tasks:-

2708/2516/2716 EPROMs

1) Check whether the EPROM is erased, and display a suitable message.
2) Program the EPROM from data in RAM, and display a progress count.

2708/2516/2716 EPROMs, and pin compatible ROMS

3) Transfer the contents of the chip into RAM at a chosen address.
4) Compare the contents of the chip against RAM, and display errors.
5) Dump a copy of the chip contents to a printer on the serial port, displaying memory locations and hex and ASCII data.

The program occupies about 2K bytes of memory space. No attempt has been made to reduce this for the following reasons:-

1) The program has been written so as to minimise the chance of operator error, by offering single-key choices backed up by verification of entry wherever possible. This requires a large number of messages and prompts, which take up a lot of memory space.
2) Because of this, it would have been difficult to keep the size of the program below 1K bytes; 2K seemed to be suitable, as the program would then fit into a 'self-programmed' EPROM.

3) The program would not normally reside in memory (unless, as in the author's case, on an EPROM board that can be paged in or out of the system, see INMC80 No. 4). Normally, it would be loaded from disc or tape.

The listing was written using the CP/M Editor 'ED' for assembly by Macro 80, because program development is easier with these more powerful utilities. It should be easy to alter it to suit ZEAP. The main changes required are:-

Leave out the colons after labels;
Remove the assembler directives END,.Z80,ASEG,.PHASE and .DEPHASE;
Substitute a suitable memory ORG address;
Substitute " in place of ' in compare instructions.

If you have 'ED' or any other editor with MACRO find/substitute commands it should be very easy to make the changes needed to the Source code. Otherwise, there should be few problems. The PIO ports are defined for the I/O board (using ports 14 and 15). These can be changed in the equates section of the listing.

SOFTWARE OPERATION

The program is written so that the user is reminded, by means of messages on the screen, to take suitable precautions when handling chips. User input is in the form of single key replies to prompts on the screen. All entries are echoed on the screen and the user may change them. The various routines are listed in the form of a 'menu'.

When the program is executed, the title is placed on the left side of the top (unscrolled) line of the screen, and warnings about chip handling appear for a couple of seconds. The program then prompts the operator to press key 'C' when he is ready to continue. Whilst waiting for a response, the program display a message warning the operator to switch off when inserting or removing chips. When key C is pressed a prompt asks for the type of EPROM to be handled – key A should be pressed for 2708s, key B for 2516/2716s.

When this entry has been made, the user is asked to verify it by a 'Y' or 'N' response. If 'N' is entered the program repeats the request for the EPROM type, while on receipt of 'Y' the type selected is displayed on the right hand side of the top line, and remains there until changed. In addition the value stored in the program workspace at the location ROMFLG is set to 04H for 1K EPROMs, or 08H for 2K EPROMs.

The 'Menu' for the five routines is then displayed, together with a list of the keys needed to access the routines. Once a routine has been selected and verified, it is then immediately accessed as described below. At the end of each routine a suitable message is output, either confirming the completion, or indicating that errors have occurred. This message is held on the screen for approximately 2 seconds, and then the program jumps back to the label 'RESTRT', and the user is asked if he wants to carry on with the same type of chip. A response of 'Y' will take him back to the menu, while 'N' will return him to the key 'C'/warning mesage routine, followed by chip type selection.

THE ROUTINES

E – CHECK FOR ERASED EPROM

This routine needs no further information from the user and it immediately reads each byte in succession from the chip, checking that its value is 0FFH. If all the bytes are FF, the routine ends with a message saying that the EPROM is erased. If any byte is detected which is not FF, the routine terminates immediately and the message "EPROM" not fully erased" appears. The operation of the routine is very rapid.

The remaining four routines all need a four digit hexadecimal address in order to continue. A subroutine is called when the program enters the chosen routine; this subroutine prompts the operator to enter the required address. It checks the data as it is entered, and only allows valid hexadecimal digits to be stored.

If an error is made during the entry of the four digits, the user must continue until four entries have been made. He will then be given a chance to change the whole entry. On exit from the subroutine, the addres will be in the HL register pair, and also in the workspace at label STOR1.

The address entered will be used as the start of a 1K or 2K block of RAM in the computers memory by the program, transfer and compare routines; for the 'Dump' routine, the normal operating address of the chip should be entered. This address will be printed at the start of each line of data, incrementing by 10H for each line.

P – PROGRAM EPROM FROM DATA IN RAM

After obtaining the address to be used as the start of the data to be put in EPROM and setting up the counters, etc., the routine tests the value at ROMFLG to decide whether 1K or 2K chips are to be programmed. Depending on this value, it selects the appropriate programming routine; two separate routines are required because the programming requirements of the two types of chip are very different. 2708s need each address to be cycled a large number of times (100 – 1000) with a programming pulse of between .1 and 1 millisecond., to produce a total 'burn' time of 100 milliseconds per address. 2516/2716 chips need only one cycle, with a programming pulse of 50 milliseconds per address. It takes about 2 minutes to program either type.

While the programming is going on, a display is put on the screen to show that something is happening. With 2708s the number of programming cycles left is displayed (in hexadecimal). With 2K EPROMS a count is output every 100H (256 decimal), beginning with 00 at the beginning of the first block.

Note that ths software is written for a 4 Mhz clock and NO WAIT STATES. If either are changed, then the value 0E0H in the B register (for the 2708 routine), or 1D00H in BC (2516/2716 routine) will have to be changed. Short delays are written into the software to allow time for line stabilisation or chip ‘set-up’.

C – COMPARE CONTENTS OF EPROM WITH RAM

This routine will compare the EPROM and RAM, byte by byte. It may be used to check for correct programming, or to find small discrepancies between EPROMs and RAM that should be identical. If a mismatch is found the address of the byte in RAM that did not correspond with the ‘EPROM’ will be printed on the screen, and a message that the data did not match will be printed at the end of the routine. If the EPROM and RAM match the routine will end with a suitable message.

T – TRANSFER DAT FROM EPROM TO RAM

This routine will quickly copy the contents of the chip into RAM, starting at the RAM address entered. This data may then be disassembled, modified, relocated, or used to make a back-up copy as desired. On completion of the routine a message is displayed, as there is no other indication that the routine has run its course.

D – DUMP TO PRINTER

This routine was written with the ‘IMP’ printer in mind, and so it interfaces through the serial port. A handshake routine is included (using bit 7 of the keyboard port), to avoid having to set up user routines. The program starts by requesting the normal address of the EPROM. It prints this address, followed by sixteen bytes of data and the ASCII characters correponding to this data (for characters 20H to 7BH; all other characters are printed as ‘.’). A new line is started, and the address is updated, and the routine is continued until all the data in the EPROM has been printed out.

The routine works by reading sixteen bytes from the chip into a workspace buffer (label LINBUF in the source code, pointed to by the IY register). These bytes are then printed, another sixteen bytes read in, and so on.

THE HARDWARE

The author’s unit is built in two parts. A diecast box, 4.5” x 6.5” x 2”, is used as a base and contains a small power supply that provides +5, +12, -5, -12 and +26 volts (the -12 volt line is not needed by the programmer, but it is easy to include and makes the power supply useful for other circuits). The electronics are built on a piece of 0.1” matrix Veroboard, that sits on top of the box. Power could be taken from the computer supply, as only a few milliamps are needed, but a 26 volt generator would still be required. By providing a completely separate power suply the unit can be made much more portable.

Two EPROM sockets are fitted to the unit; one is for 1K chips, the other for 2K chips. To avoid the possibility of damage the sockets should be CLEARLY marked with the chip type, and pin 1 should also be marked. The use of zero insertion force sockets is strongly recommended, to prevent wear and tear on the chips and sockets. Only one socket should be used at a time.

The Z80 PIO has a low drive capability, and buffering is desirable. Pin 20 of the 2708 (WE) rises to 12 volts during programming. Since this is well above TTL levels, the use of a high voltage rated buffer is necessary. The 7406 is suitable, as it is rated at 30 volts. Double buffering is used in the unit as this provides the required number of signal inversions, low loading of the PIO, and the necessary voltage isolation. Buffer IC1 is a CMOS 4049 hex inverter chip; IC2 is a 7406, backed by a couple of BC108/9 transistors to make up the required number of inverters.

IC3 is a CMOS 4040 12 stage binary counter, capable of counting from 0000H to 0FFFH (0 to 4095 decimal). In this unit a count of 03FFH is used for 2708s and a count to 07FFH for 2K chips. The spare output might be used in the future to extend the unit to 4K chips. Port A of the PIO is used for data, and port B for control. Normally port A is set to input and port B to output, but during programming both ports are output.

PORT B BIT ASSIGNMENT

BIT 0 — Controls the 26 volt programming pulse through the high speed switch formed by TR1, TR2 and TR3.

BIT 1 — Controls the WE/$\overline{\text{CS}}$ signal on pin 20 of the 2708 socket, and the $\overline{\text{OE}}$ signal on pin 20 of the 2516/2716 socket.

BIT 2 — A pulse from this bit is used to increment the address counter.

BIT 3 — A pulse from this bit is used to reset the address counter at the start and end of each routine.

BIT 4 — Used to switch the PGM input on pin 18 of the 2516/2716 socket.

BIT 5 — When this bit is 0, it holds pin 20 of the 2708 at 5 volts, during normal READ operations. When set to 1, it allows pin 20 to pull up to 12 volts, for ‘Write Enable’.

The power supply is fitted with a multipole switch on the D.C. outputs. This should be fited in ALL cases, and should be used to switch off the programmer when changing chips. If the programmer is turned on/off with a mains switch when a chip is being inserted or removed, the chip may be damaged, because the low voltage supplies decay more quickly than the 26 volt line; this can result in a 26 volt pulse being written into address 0 of the EPROM, possibly PERMANENTLY!

COMPONENTS FOR THE PROGRAMER.

Transistors		Resistors
TR1, 2	BC 548	4 x 10 Kohm
TR3	BC558	1 x 33 Kohm
TR4, 5	BC109	1 x 180 Ohm
Diodes		1 x 47 Ohm
4 x 1N4148		1 x 1 Kohm
Integrated Circuits		6 x 4.7 Kohm
IC1	CMOS 4049	Capacitors
IC2	TTL 7406	1 x 0.001 μF
IC3	CMOS 4040	5 x 0.1 μF

COMPONENTS FOR POWER SUPPLY

Transformer		Rectifier
15 – 0 – 15 volts, 6VA		1 A Bridge, 50 PIV
Diodes		Voltage Regulators
D1, 2	1N4002 or similar	78L12 (+12V, 100 mA)
Z1	26V, 1 watt Zener	78L05 (+5V, 100mA)
Resistors		79L12 (-12V, 100mA)
2 x 4.7 Kohm		79L05 (-5V, 100mA)
1 x 1 Kohm		Capacitors
1 x 10 Kohm		4 x 0.47 μF
1 x 15 Kohm		2 x 0.22 μF
Single pole mains switch		2 x 4700 μF, 25V
Five pole low voltage switch		2 x 470 μF, 64V

The full circuit diagrams of the programmer and power supply, a Veroboard layout of the programmer, and the software for its operation will be given in the next issue of the magazine.

* * * * * * * * * * * * * *

THE ZILOG Z800

by Rory O'Farrell

Zilog have anounced the Z800, a replacement upgrade for the Z80. Information on the new processor is scant, but in a brief press release Zilog claim the following:-

Three to five times the performance of the Z80A with comparable speed memory,
On chip internal clock; 12 18 & 25 Mhz,
Expanded instruction set that is binary compatible with all Z80 instructions,
Multiply and divide instructions,
On chip memory management and protection unit,
Direct addressing of half a megabyte of memory (524288 bytes),
Programmable bus timing (wait states selectable in software),
Multiplexed address/data bus (i.e., address and dasta lines share the same pins) with Z80 bus signals for easy interfacing to Z80 family chips.

It is claimed that the instruction set is more powerful than that of the Z80, and that it incorporates many of the features of the Z8000. The chip can be used with any of the Z80 or Z8 peripheral chips.

The register structure seems to be very close to that of the Z80. There are two sets of registers, each comprising an accumulator, a flag register, and six general purpose registers. Transfer of data between these duplicate sets is accomplished by the use of 'exchange' instructions. Zilog claim that the result is a faster response to interrupts, and easy implementation of context switching for multi-user processing. In addition there are the interrupt and refresh registers, and two 16 bit index registers. Two implied stack pointers are available: the system stack pointer (which we know and love?) and a user stack pointer. The CPU mode of operation will determine which of these pointers is used. The user stack pointer will facilitate the writing of very efficient high-level language compilers and interpreters.

The allowable data types are bits, BCD digits (nibbles, 4 bits), bytes (8 bits), words (16 bits), and byte strings up to 64 Kbytes long. The standard Z80 instruction set is extended with 8 and 16 bit multiply and divide, and the SET and TEST instruction. In addition, there is a fourth interrupt mode, which provides more flexibility in handling interrupts and traps. The new CPU has a comprehensive trapping structure, allowing for single stepping, system calls, and privileged instruction traps.

The chip offers programmable bus timing. It can insert, under software control, wait states into both memory and I/O transactions. The on-chip clock can also be programmed – an example is quoted of running 6 Mhz memory from the internal 12

Mhz clock. The clock can be controlled from an external crystal, or from an internal oscillator, and is available to the rest of the system. Refresh is provided by the chip; software can select the interval between successive refreshes, or even suspend it entirely!

The on-chip memory management unit provides for a flexible memory structure, by allowing dynamic page relocation, as well as write protect features. The 16 address lines output by the CPU are transformed into 19 bit physical addresses. This large cache of memory will facilitate multiple users (as for example, in schools), or foreground/background processing (playing Adventure and monitoring the nuclear power plant at the same time!).

Delivery of the new CPU is not expected until the first quarter of 1983, and a price has not yet been quoted. It should be stressed that the chip will not be pin-for-pin compatible with the Z80. however, it should not be impossible to make a very compact interface board, although until detailed pinouts and bus timings are published this can only be a dream. If the promise is kept of full software compatibility with the Z80, and three to five times the throughput with the same speed memory, then this will be quite some chip!

---

nascom

* Housed in strong, stylish case with high quality QWERTY keyboard.
* 0.7Mbyte floppy disc system available in matching case.
* Full 8K RAM, expandable to 200K with page mode and RAM boards.
* Factory-built options plus additional range of Nascom-approved hardware and software.

Think of Nascom 3 as an advanced personal computer, built to professional standards and offering the total systems versatility needed by enthusiasts whose imaginations are already ahead of the toy computer field.

Think of Nascom 3 as the powerful heart of a truly versatile educational or business computer system, with added peripherals and an extensive range of firmware and software options.

Or think of Nascom 3 as a custom-structured industrial control unit, well capable of cutting production costs in many key areas.

Nascom 3; reliable, expandable, affordable – and backed by one of Britain's best known engineering groups. Think about it.

OTHER NASCOM PRODUCTS

* Nascom 1 from £125 + VAT
* Nascom 2 from £225 + VAT
* Memory Extension Unit from £80 + VAT
* Disc systems from £375 + VAT
* Input/Output board from £37 + VAT

NEW

* Advanced video controller from £156 + VAT
* Enhanced BASIC from £40 + VAT
* Pascal compiler from £45 + VAT
* Compiled BASIC from £150 + VAT

SPECIAL OFFER IMP PRINTERS £109 + VAT WHILE STOCKS LAST

NASCOM DEALERS

UK

[illegible]

OVERSEAS

[illegible]

Lucas Logic

XTAL BASIC XTRA

by David Elliot

For all you Nascom buffs with Xtal Basic 2.2, here follows a series of articles which show you how to get the best out of this Basic's 'expandability' – it facility for the addition of extra machine code routines. By the time you have added a few custom-built commands, your Microsoft pals will be green with envy. Just to whet your appetite, here are some of the additional commands used with Xtal Basic on my own system, which is a 32K Nascom 1 to which I have added 256 programmable characters (which can be used as high-res graphics):

DOKE	DEEK	RAD	DEG	CLS
SWAP	SETBIT	RESBIT	BIT	AUTO
AOFF	SET	RESET	DRAW	MOVE
SOUND	TUNE	INIT	GCLEAR	SHOW
GXOR	GOR	SCREEN	PON	POFF

Many of these commands are designed to work with the programmable graphics board, and make games programming much simpler.

LOADER PROGRAM

The Xtal Basic handbook explains how to insert the additional command name into the command table, and the vector to the machine code routine into the vector table, but just to make things even easier, here is a machine code loader program which executes from 4E00H.

Xtal Basic is first read in, and then the loader program itself is loaded and executed. It first asks for the name of the command; if no name is entered the loader accepts the following machine code as a subroutine, rather than as an extra command. When a name is entered it is it is automatically added to the end of the command table and the vector is set to beginning of the machine code routine.

When the machine code is being entered the loader automatically prints the address at the beginning of the line and then waits for the code to be typed in. To allow for code relocation, the loader has a command '+', which takes the following 16-bit number and adds the start of the command to it before placing it in memory.

There is limited amount of error-checking to test the numbers entered for the correct number of digits. If an error is detected, the cursor is positioned at the error, which can then be changed, using the Nas-Sys screen editing.

When the machine code routine is complete a full stop is entered. The loader then asks if another command is to be entered. If the response is ‘Y’, the program continues, adding the next command immediately after the last. Otherwise it changes the Basic Text Pointer (at 1283H) to the start of the next 256-byte page. In Xtal Basic the text must start on a page boundary.

```
                    ;
                    ; ***********************************************
                    ; ** COMMAND LOADER BY D. ELLIOT   **
                    ; ***********************************************
                    ;
4E00                          ORG £4E00         ; EXECUTION ADDRESS
4E00   EF           LOADR     RST 40            ; PRINT STRING
4E01   0C                     DEFB 12           ; CLEAR STRING
4E02   456E7465               DEFM“ENTER COMMAND/FUNCTION NAME”
4E1E   0D00                   DEFB 13,0         CRLF, END OF STRING
4E20   DF63                   SCAL INLIN        ; INPUT LINE
4E22   1A                     LD A, (DE)        ; GET 1ST CHARACTER
4E23   FE20                   CP "              ; NAME ENTERED?
4E25   2832                   JR Z,NOTCOM       ; JUMP IF NOT
4E27   21800E                 LD HL, NAMES      ; SET POINTER
4E2A   0EFF                   LD C,-1           ; RESET COUNTER
4E2C   7E           LOOP      LD A (HL)         ; GET NEXT BYTE
4E2D   CB7F                   BIT 7,A           ; START OF WORD?
4E2F   2805                   JR Z,NEXT         ; IF NOT, TRY NEXT
4E31   0C                     INC C             ; INCREMENT COUNTER
4E32   FE80                   CP £80            ; END OF TABLE?
4E34   2803                   JR Z,ADDNAM       ; ADD TO TABLE
4E36   23           NEXT      INC HL            ; INCREMENT POINTER
4E37   18F3                   JR LOOP           ; NEXT CHARACTER
4E39   1A           ADDNAM    LD A (DE)         ; GET NEXT BYTE
4E3A   CBFF                   SET 7,A           ; SET BIT 7
4E3C   77                     LD (HL),A         ; SAVE 1ST CHAR.
4E3D   23           ADD1      INC HL            ; INCR. POINTERS
4E3E   13                     INC DE
4E3F   1A                     LD A,(DE)         ; GET CHARACTER
4E40   FE20                   CP "              ; END OF NAME?
4E42   2803                   JR Z,ADD80        ; IF SO, INSERT £80
4E44   77                     LD (HL),A         ; SAVE CHARACTER
4E45   18F6                   JR ADD1           ; CONTINUE
4E47   3680         ADD80     LD (HL), £80      ; ADD DELIMITER
4E49   59                     LD E,C            ; CALCULATE ADDRESS
4E4A   1600                   LD D,0            ; OF VECTOR
4E4C   EB                     EX DE,HL
4E4D   29                     ADD HL,HL
4E4E   11800F                 LD DE,VECT        ; START OF TABLE
4E51   19                     ADD HL,DE
4E52   ED5B8312               LD DE,(TEXT)      ; SET VECTOR
4E56   73                     LD (HL),E         ; AT END OF BASIC
4E57   23                     INC HL
4E58   72                     LD (HL),D
4E59   EF           NCOM      RST 40            ; PRINT STRING
4E5A   456E7465               DEFM /ENTER MACHINE CODE./
4E6D   0D00                   DEFB 13,0         ; CRLF, STRING END
```

```
4E6F    FD2A8312   0570           LD IY, (TEXT)      ; ZERO POINTER
4E73    FDE5       0580  IN3      PUSH IY            ; CALCULATE ADDRESS
4E75    E1         0590           POP HL
4E76    ED4B8312   0600           LD BC, (TEXT)      ; BASE ADDRESS
4E7A    B7         0610           OR A               ; RESET CARRY FLAG
4E7B    ED42       0620           SBC HL, BC         ; SUB. BASE ADD.
4E7D    DF66       0630           SCAL TBCD3         ; PRINT ADDRESS
4E7F    DF63       0640  IN0      SCAL INLIN         ; INPUT LINE
4E81    CDBB4E     0650           CALL NUM16         ; GET ADD. IN BC
4E84    FD2A8312   0660           LD IY, (TEXT)      ; CALCULATE PROPER
4E88    FD09       0670           ADDIY, BC          ;ADDRESS
4E8A    1A         0680  IN1      LD A, (DE)         ; GET CHARACTER
4E8B    FE20       0690           CP "               ; A SPACE?
4E8D    2003       0700           JR NZ COMND        ; IF NOT, JUMP
4E8F    13         0710           INC DE             ; TRY NEXT
4E90    18F8       0720           JR IN1             ; CHARACTER
4E92    FE2E       0730  COMND    CP ".              ; END OF PROGRAM
4E94    284D       0740           JR Z, END
4E96    B7         0750           OR A               ; END OF LINE?
4E97    28DA       0760           JR Z, IN3          ; GET NEXT LINE
4E99    FE2B       0770           CP "+              ; RELATIVE NUMBER?
4E9B    2014       0780           JR NZ, IN2         ; IF NOT, 8 BIT NO.
                   0785  ;
                   0790  ; IF YES, THEN INPUT A 16 BIT NUMBER
                   0800  ; AND ADD OFFSET TO PROPER ADDRESS
                   0805  ;
4E9D    2A8312     0810           LD HL, (TEXT)      ; GET BASE ADD.
4EA0    13         0820           INC DE             ; START OF NUMBER
4EA1    CDBB4E     0830           CALL NUM16         ; OFFSET IN BC
4EA4    09         0840  CHK      ADD HL, BC         ; CALCULATE ADDRESS
4EA5    FD7500     1010           JR NZ, ERROR       ; IF NOT, ERROR
4ECC    C9         1020           RET                ; RETURN
4ECD    DF64       1030  NUM8     SCAL NUM           ; GET 8 BIT NUMBER
4ECF    380C       1040           JR C, ERROR        ; ERROR DETECTED
4ED1    ED4B210C   1050           LD BC (NUMV)       ; GET NUMBER
4ED5    3A200C     1060           LD A, (NUMN)       ; CORRECT LENGTH?
4ED8    FE02       1070           CP 2               ; TWO CHARACTERS
4EDA    2001       1080           JR NZ, ERROR       ; IF NOT, ERROR
4EDC    C9         1090           RET                ; RETURN
4EDD    ED53290C   1100  ERROR    LD (CURSR), DE     ; POSITION CURSOR
4EE1    189C       1110           JR IN0             ; RE-INPUT LINE
4EE3    EF         1120  END      RST 40             ; PRINT STRING
4EE4    416E6F74   1130           DEFM “ANOTHER COMMAND (Y/N)”
4EFB    1100       1140           DEFB CUL, 0
4EFD    DF7B       1150  END0     SCAL BLINK         ; GET ANSWER
4EFF    FE59       1160           CP "Y              ; IS IT ‘YES’
4F01    2008       1170           JR NZ, END1
4F03    F7         1180           RST CRT            ; PRINT IT
4F04    FD228312   1190           LD (TEXT), IY      ; RESET POINTER
4F08    C3004E     1200           JP LOADR           ; AND CONTINUE
4F0B    FE4E       1210  END1     CP "N              ; ANSWER NO?
4F0D    20EE       1220           JR NZ, END0        ; INPUT AGAIN
4F0F    F7         1230           RST CRT            ; PRINT N
4F10    FDE5       1240           PUSH IY            ; TRANSFER TO HL
4F12    E1         1250           POP HL
4F13    7D         1260           LD A, L            ; IS IT ON A PAGE
4F14    B7         1270           OR A               ; BOUNDARY?
```

```
4F15    2803                JR Z, OK          ; IF SO, END
4F17    24                  INC H             ; ADD 256 TO END
4F18    2E00                LD L,0            ; ZERO LOW BYTE
4F1A    228312    OK        LD (TEXT), HL     ; STORE POINTER
4F1D    DF5B                SCAL MRET         ; RETURN TO MONITOR
        - -   - -   - -   - -   - -   - -   - -   - -
```

The first example of an added command is a routine which provides automatic line numbering. The listing of this routine is followed by a demonstration of the entry of the corresponding machine code using the loader program.

```
                  ;
                  ; ************************************************
                  ; ** AUTO LINE NUMBER BY D. ELLIOT   **
                  ; ************************************************
                  ;
3000                      ORG £3000
                  ;
                  ; ROUTINES IN CRYSTAL BASIC 2.2
                  ;
3000    2BF5      OVEC      EQU £2BF5         ; OUTPUT VECTOR+1
3000    2781      PRTHL     EQU £2781         ; PRINT HL IN DEC.
3000    1761      INNUM     EQU £1761         ; GET NUMBER
3000    154C      TSTCOM    EQU £154C         ; SKIP COMMA
                  ;
                  ; MODIFIED OUTPUT ROUTINE WHICH CHECKS
                  ; FOR A NEW LINE,  AND  THEN  OUTPUTS THE
                  ; NEXT LINE NUMBER
                  ;
3000    FE5D      AOUT      CP £5D            ; NEW LINE?
3002    2064                JR NZ, COUT       ; IF NOT, PRINT
3004    F7                  RST CRT           ; PRINT CHARACTER
3005    222E30              LD (BUFFR), HL    ; SAVE POINTER
3008    213030              LD HL, NOUT       ; CHANGE OUT. VECTOR
300B    22F52B              LD (OVEC), HL
300E    2A2A30              LD HL, (INC)      ; GET INCREMENT
3011    EB                  EX DE, HL         ; INTO DE, AND LAST
3012    2A2C30              LD HL, (LAST)     ; LINE NO. IN HL
3015    D5                  PUSH DE
3016    E5                  PUSH HL
3017    CD8127              CALL PRTHL        ; HL TO SCREEN
301A    E1                  POP HL            ; AND TO BUFFER
301B    D1                  POP DE
301C    19                  ADD HL, DE        ; CALC. NEXT NO.
301D    222C30              LD (LAST), HL     ;  AND STORE ITVN
3020    210030              LD HL, AOUT       ; RESET VECTOR
3023    22F52B              LD (OVEC), HL
3026    2A2E30              LD HL (BUFF)      ; RESTORE POINTER
3029    C9                  RET               ; RETURN
                  ;
                  ; VARIABLE STORAGE
                  ;
302A    0002      INC       DEFS 2            ; INCREMENT
```

```
302C    0002                LAST    DEFS 2          ; LAST NUMBER
302E    0002                BUFF    DEFS 2          ; NEXT CHAR. ADD.
                            ;
                            ; PRINT CHARACTER ON SCREEN AND
                            ; INTO BASIC INPUT BUFFER
                            ;
3030    E5                  NOUT    PUSH HL         ; GET LAST ADDRESS
3031    2A2E30                      LD HL, (BUFF)   ; SAVE CHARACTER
3034    77                          LD (HL), A      ; INCREMENT ADDRESS
3035    23                          INC HL          ; SAVE ADDRESS
3036    222E30                      LD (BUFF), HL
3039    E1                          POP HL
303A    F7                          RST CRT
303B    C9                          RET
                            ;
                            ; AUTO COMMAND
                            ;
303C    110030              AUTO    LD DE, AOUT     ; CHANGE VECTOR
303F    ED53F52B                    LD (OVEC), DE
3043    3EC3                        LD A, £C3       ; SET UP JUMP
3045    32F42B                      LD (OVEC-1),A
3048    CD6117                      CALL INNUM      GET START NUMBER
304B    ED532C30                    LD (LAST), DE
304F    CD4C15                      CALL TSTCOM     ; SKIP COMMA
3052    CD6117                      CALL INNUM      ; GET INCREMENT
3055    ED532A30                    LD (INC), DE
3059    C9                          RET
                            ;
                            ; AOFF COMMAND
                            ;
305A    E5                  AOFF    PUSH HL         ; RESTORE OUTPUT
305B    216830                      LD HL, COUT     ; VECTOR TO NORMAL
305E    22F52B                      LD (OVEC), HL
3061    E1                          POP HL
3062    3EC3                        LD A, £C3       ; SET UP JUMP
3064    32F42B                      LD (OVEC-1),A
3067    C9                          RET
                            ;
                            ; CRT OUTPUT ROUTINE
                            ;
3068    F7                  COUT    RST CRT
3069    C9                          RET
```

The AUTO command format is AUTO xxxx, yyyy, where xxxx is the starting line number, and yyyy is the increment.

The automatic line numbering routines are entered using the loader program as follows:-

ENTER COMMAND/FUNCTION NAME.

ENTER MACHINE CODE

```
0000  FE  5D  20     64  F7  22  +002E  21  +0030  22  F5  2B  2A  +002A
0011  EB  2A  +002C  D5  E5  CD  81     27  E1     D1  19  22  +002C
```

```
0020   21   +0000    22   F5   2B   2A   +002E    C9   FF   FF   FF   FF   FF   FF
0030   E5   2A   +002E    77   23   22   +002E    E1   F7   C9   .
```

ANOTHER COMMAND (Y/N)?

ENTER COMMAND/FUNCTION NAME

AUTO

ENTER MACHINE CODE

```
0000   11   +FFC4    ED   53   F5   2B   3E   C3   32   F4   2B   CD   61   17   ED
0010   53   +FFF0    CD   4C   15   CD   61   17   ED   53   +FFEE    C9   .
```

ANOTHER COMMAND (Y/N)?

ENTER COMMAND/FUNCTION NAME

AOFF

ENTER MACHINE CODE

```
0000   E5   21   +000E    22   F5   2B   E1   3E   C3   32   F4   2B   C9   F7   C9.
```

ANOTHER COMMAND (Y/N)?

No name is entered for the first section of machine code, as this code is a series of subroutines used in the AUTO command. Addresses entered under the '+' option are measured relative to the start of the current section of code being entered. Thus any references to subroutines which precede the current section will be negative. For example, in the second section of code, the AUTO command, there are three references to the subroutines in the first section which are all entered as negative hexadecimal numbers (FFC4, FFF0, FFEE).

The six bytes of workspace used by the AUTO routines can be entered as any six 8 bit values – in the example above they are entered as six FF's.

. . . .

LETTERS

Dear Sir,

Further to Mr. Bowden's letter in issue 3, I must also add my congratulations to Chris Blakmore for his excellent MONITOR.COM, which I call N3.COM allowing me to type N3 and be in Nas-Sys. I cannot help him to convert the RAM ver sion of Zeap 2 to run with the moved video RAM, but I can help anyone who wants to run the EPROM version. The bytes to be changed are:-

D5D4	D62D	D83C	D844	DB1F	DB7D	DBE3	DBF4
DBFD	DC06	DC12	DC68	DCA5	DCAE	DCB8	DDC8
DE06	DE16	DE26	DE59	DE95	DECD		

If the modifications suggested in INMC7 have been carried out, it will be necessary to change the byte at DFBA. Each of the addresses at these bytes will be found to be in the range 08 – 0B; F0 should be added. If it is required to run the EPROM version in RAM for test purposes, the byte at DF52 should be changed to 00.

In order to run D-BUG and Nas-Dis with the revised video RAM, it is only necessary to change the bytes at C01A, C0B3 and C118.

J. T. Nestor, East Kilbride

Dear Ed,

After reading the article on TRS-80 tapes in issue 3, I thought the following might be of interest.

It is possible to read UK101 tapes directly and very simply. Providing the tape has been recorded at 300 or 1200 Baud (most will be at 300), go into mode X from Nas-Sys, initialise Basic with the J command, and then play the tape without entering any commands. The program listing will be printed onto the screen and into memory with the original Basic line numbers. On completion, the program can be modified to run on the Nascom with usually not too much effort!

R. M. Dowling, Welling

Dear Sir,

I recently bought a V&T Superdeck recorder, which came with a demo tape but no instructions. As the company has closed down, is there anyone who can tell me how to connect the Superdeck to my Nascom 2. The twelve connections are marked C, S, R, W, E, D and G, Common, Tx and Rx Clock, and Tx and Rx Data. I'd be grateful for any help.

M. Harrison, 8, Langleys Gardens,
Prestwich, Manchester

FOR SALE	RAM A card with 8K	£45 o.n.o.
	RAM B card with 16K	£70 o.n.o.
	Both in full working order.	C. Bowden

Tel. 0209 860 480 (Evenings) 0209 712 780 (daytime)

HANDS ON

by Viktor

The End of the Beginning (or vice versa?)

READ,DATA & RESTORE

This group of instructions can be extremely useful in certain programming situations, for instance, where a large number of values are to be assigned to variables before the main body of the program can be run. For example, in a program to play music you might want to hold the frequencies of, say, 85 notes in an array. After dimensioning the array, by the command DIM A(84), the values could be assigned as follows:

```
20 A(0)=123:A(1)=125:A(2)=128. . . . . . .A(84)=222
```

It is much simpler, however, to put the values in DATA statements and then READ them all into the array in one go:

```
10 DATA 123, 125, 128 . . . . . . . .
20 DATA . . . . . . . . 216,220,222
30 FOR J=0 TO 84: READ A(J): NEXT
```

Note that because the array contains a member A(0), you can always dimension it to one less than the number of array members needed

When a BASIC program is run, the interpreter looks right through the program and notes the position of all items included in the DATA statements. The DATA statement pointer is then set to the beginning of the list. As each READ statement is executed, the pointer is moved onto the next item. If at any time you want the program to start at the beginning of the list, you just use the command RESTORE. You may also want to READ from various positions down the list. As long as this coincides with a line number you can use RESTORE X (where X is the relevant line number)

A very good example of this was in the Hangman program in the last issue of Micropower, where the author needed first to refer to Numeric data string at line 8000 and then to String data starting at line 9000.

It is possible but not advisable to mix numeric and string items in DATA statements e.g.

```
10 DATA “Fred”,5,“Jim”,7,8,“Harry”
20 READ A$,X,B$,Y,Z,C$
```

However, if you make an error and try to read one type of data into the other type of variable you will get a ‘SYN’ error – and serves you right !!

Program lines similar to the following have been included in many simple computer-based learning tests.

```
10 READ A$, B$
20 PRINT "What is the capital of";A$
30 INPUT C$
40 IF C$=B$ THEN PRINT "Correct": GOTO 10
50 PRINT "Hard Luck –Try again":GOTO 20
60 DATA SCOTLAND, EDINBURGH, EIRE, DUBLIN
70 DATA FRANCE, PARIS, WEST GERMANY, BONN
```

Obviously one would enlarge on the program, with random or sequential testing, some system of marking results and suitable messages to 'humanise' the exercise.

It is self-evident that the basic formula using READ, DATA & RESTORE can be adopted in all such programs.

SET, RESET, POINT

As everyone will know by now, the basic Nascom screen has 16 lines of 48 characters – i.e. 768 screen locations into which you can PRINT or POKE any of the character set. The commands SET, RESET & POINT give you control of a much higher number of smaller areas on the screen. In effect, each character space is divided into 6 (2 horizontal x 3 vertical), and each 'pixel' as they are called, can then be turned on or off by SET and RESET. This gives and effective screen resolution of 96 x 48; computer manufacturers have been known to refer to this as 'high resolution graphics), but this term should really be reserved for the much higher resolution obtainable with bit-mapped graphics or with a programmable character generator.

As an illustration, we can randomly set all the pixels thus:

```
10 CLS
20 REM * SET X TO A RANDOM NO. BETWEEN 0 AND 47
30 X=INT(RND(1)*47)    40 REM* SET Y IN RANGE 0 – 95
50 Y=INT(RND(1)*95)
60 SET(X,Y) : GOTO 20
```

Eventually the screen would be composed entirely of 'set' pixels. Now we can use RESET in a similar fashion. First remove the 'GOTO 20' in line 60 and then enter:

```
70 REM* NOW DO A RANDOM RESET
80 X=INT(RND(1)*47)
90 Y=INT(RND(1)*95)
100 RESET (X,Y) : GOTO 20
```

The pixels will now be randomly reset. To complete the program we could use POINT to test whether certain pixels had been set, and if so start at the beginning again. Remove the 'GOTO 20' in line 100 and then enter:

```
110 REM* TEST A BLOCK OF PIXELS 40 X 30
120 FOR S=15 TO 75 STEP 15
130 FOR T=15 TO 45 STEP 15
140 IF POINT (S,T)=1 THEN 10
150 NEXT T,S
160 GOTO 20
```

PRINTING CHARACTERS FROM THE KEYBOARD

In the manual there is an appendix headed ‘Single Character Input of Reserved Words’. From the list you will see that when typing in a program you can use various combinations of keys to obtain single characters (often referred to as ‘Tokens’) which the computer interprets as instructions. For example:-

? = PRINT   CTRL/GRA/H =GOTO.

This does save time and space when keying in a long program, but also causes a lot of headaches when editing. When the program is listed the tokens are expanded to the full reserved words; consequently lines may exceed 48 characters, and when you try to edit them you loose charactrers from the end. It is all too easy to ‘crash’. In Direct Mode, however, they are always useful. E.g. GRA/Space = LIST.

You might also make use of this facility when designing graphics shapes for use in a program. It is worthwhile marking the chart supplied in the manual with the key designations. For instance, the first two lines (32 chars.) are obtained by depressing the graphics and control keys plus @, A-Z, [, \,], ^ and _.

Let us look at what happens when we press the graphics and control keys. The former sets Bit 7, while the latter flips Bit 6. E.g. The @ character (shift@) is 40 in hex or 01000000 in binary, the bits being numbered 7 to 0 from left to right. Setting Bit 7 gives 1100 0000, or C0 in hexadecimal, and flipping Bit 6 gives 1000 0000 or 80 hex. Thus character 80 hex can be typed with CTRL/GRA/shift@. Similarly, 8F in hexadecimal is obtained with CTRL/GRA/O.

Character FF illustrates the effect of flipping Bit 6 from off to on. The ? has a hex value of 3F (0011 1111). CTRL/? gives 7FH (01111111) , and GRA/CTRL/? gives 0FFH (11111111)

You can always try out various key combinations by typing them directly into a Basic line, and then expanding them by LISTing the ‘program’. For example, type in a line number and then hold the Graphics key down while you enter A,B, C, D, E, F. Now press ‘Return’ and LIST the line. You will find that the interpreter expands it to:-

```
10 COSSINTANATNPEEKDEEK
```

As a final point on this topic, I thought it would be interesting to take a look at the way the BASIC stores reserved words. First enter the command NEW, and then type in the following line:

```
10 PRINT“MIKE”:END:SCREEN 15,12
```

Now reset and tabulate from 10FA hex. You will find the following code:

```
10FA    10 11 0A 00 9E 22 4D 49

1102    4B 45 3A 80 3A 97 20 31

110A    31 35 2C 31 32 00 00 00
```

The first two bytes, 10 11, store the address of the start of the next line, the next two bytes, 0A 00, hold the line number, and then come the bytes which represent the data stored in the line. The first data byte, 9EH, represents the reserved word PRINT; as a line is entered the text is scanned, and if the Basic recognises a reserved word it is replaced by a hexadecimal number in which bit 7 is set. This speeds up programs, because the interpreter can more quickly recognise a reserved word and access the necessary machine code routines.

PRINT is followed by the ASCII codes for “MIKE” (22 4D 49 4B 22) and the separating colon (3A). Two more reserved words then appear, END (represented by 80H) and SCREEN (97H).

The end of the line is marked in the store by a zero, which is followed by a pointer to the start of the next line, a line number, more data, and so on. At the end of the program the zero marking the end of the line is followed by two further zeros in place of the line pointer.

You will notice that although there is no ‘space’ character (20H) after the line number bytes, the LIST command always inserts a single space to improve legibility. No matter how many spaces you put between the line number and the start of the text when you enter the program, the interpreter always removes them - and then puts one back when listing. This makes it difficult to use ‘pretty printing’ – that is, formatting of the text by use of different indentations to make the underlying structure of the program obvious at a glance. You can always indent a line by inserting a colon before the required spaces. For example:-

```
10 :      PRINT “THIS IS AN INDENTED LINE”
```

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

NASCOM USERS

Take a look at the NASCOM APPROVED HS-IN STORAGE SYSTEM. Where else can you get features like these . . .

* **A full on screen instant display of the catalogue.**
* **Auto verification of each file as it is written.**
* **CRC error checking.**
* **Link selectable 2Mhz or 4Mhz option.**
* **Fast data transfer rate of 6000 bps.**
* **Powered from NASBUS.**
* **8" sq NASBUS compatible PCB.**
* **Far more reliable than any floppy disk system.**
* **112K on-line storage with 2 drive system.**

The HS-IN has a Command Set which makes it a floppy-disk "look-alike". It can load an 8K program in under 11 seconds and can store up to 56K (28 files) on each side of tape. Why spend £700 on a floppy disk system when the less expensive HS-IN system has a command set like this . . .

B— Write a Basic file
C— Instant display of catalogue.
D— Delete file.
J— Jump to Basic.
N— Jump to NAS-SYS.
Q— Warm start to NASPEN text editor.
R— Read a file.
T— Transfer file to another drive.
W— Write a file.
X— Exit and rewind cassettes.
Z— Warm start to Basic.

This Mini-Cassette Storage System is technologically far ahead of anything like it on the market and is extremely reliable into the bargain. AND THE COST? Because we have been successful in quantity component purchases we have been able to lower the price until January 31st 1982 (the old price is in brackets).

Single Drive System built and tested **£199** (£230)

Double Drive System built and tested **£279** (£299)

Carriage **£3.50**.

We are Scotland's foremost NASCOM Dealers and keep in stock the full range of NASCOM products as described in the Lucas Logic Advert in this magazine. For the Christmas period and up to January 31st 1982 we are offering a **FREE Statistical Calculator** (without battery) with every NASCOM product worth more than £100 or each series of NASCOM products with a value totalling £100 or more in the same order. AND if you don't want the calculator . . . just 'phone and see if we have something else you need FREE – a book perhaps!

We now have the new NASCOM CASE in stock as well as many more new NASCOM related products.

COMPONENTS AT THE BEST PRICES IN BRITAIN

MICRO-SPARES now have a vast selection of Logic I.C.'s including 74; 74LS and CMOS full range. There are Z80's and support chips as well as resistors, capacitors etc. etc. . . . far too many to list on this page. But to give you an idea of the prices just compare these . . .

	1-199	200 +
2114's (all speeds)	99p	POA
4116's (all speeds)	66p	POA
2708's	1.73p	POA
2716's Single +5v	2.15p	POA
4118's	3.80p	POA

All components are fully guaranteed and are in stock as at 15th December 1981. Orders under £30 please add 50p p. & p. VAT not included. Send SAE for current price list. Official orders from all establishments welcome.
All components in stock sent same day.

NEW

Very shortly now MICRO-SPARES will be selling the all computer RS232C version of the HS-IN. The Mini-Cassette System is just as fast and files can be any length. The machine can be connected to computers, V.D.U.'s, Printers and and other RS232C device. They will take the place of paper tape in loading engineer test programs for instance. Other communication modes are 20mA current loop, IEEE and Z80 bus.

SECOND HAND COMPUTERS

MICRO-SPARES keep a register of users that are buying or selling a computer. Stocks of second-hand machines – all in working order – are available from the very small to the very large at extremely keen prices.

Micro-Spares

19 Roseburn Terrace, Edinburgh EH12 5NG.
Tel: 031-337 5611.

THE MAGIC HEXAGON

Why not solve it the easy way!

By G. P. Robert

The magic hexagon puzzle consists of arranging the numbers 1 to 19 in a hexagonal pattern (see below) in such a way that each row in any direction adds up to 38.

A B C

D E F G

H I J K L

M N O P

Q R S

Unlike magic squares, which can be easily constructed if you know the trick, the solution to the Hexagon does not appear to possess any discernable regular pattern. The number of possible arrangements is a formidable 121,645,100,408,832,000. However, since any arrangement can be rotated six times through 60 degrees, and each of these positions has a mirror image, the actual number of distinctly different arrangements is reduced to a mere 10,137,091,700,736,000!

Inspection of the hexagon reveals one or two relationships which might help in reducing the prodigous task of finding solutions. For example, since the sum of the lines (A+B+C)+(D+E+F+G)+(H+I+J+K+L)=3x38 and (A+D+H)+(C+G+L)=2x38, then it follows that the remaining triangle of numbers (B+E+F+I+J+K) must equal 38. Addition to this triangle of the line (M+N+O+P) gives the large central triangle (B+E+F+I+J+K+M+N+O+P)=2x38. Within this larger triangle the lines (B+E+I+M) and (B+F+K+P) together equal 2x38. B, which occurs in both lines, must therefore equal the small residual triangle (J+N+O). Similar relationships will, of course, exist in each of the 12 possible orientations of the Hexagon.

Use can be made of some of these relationships to shorten the search for a solution by trial and error. The listed program, which will run on a Nascom 1 or 2 under Nas-Sys, and could be adapted to run on any Z80 based microcomputer, systematically fills the hexagon from available numbers stored in a table at the bottom of the screen. At each position where a check can be carried out, an appropriate line or triangle of numbers is added up. If the total is 38 the program proceeds to the next position. If not, then a number is selected and the test is repeated. If the end of the table is reached without a satisfactory number being found the program retreats to the previous position and the search is continued.

For convenience in programming and display, the numbers 1 to 19 are represented by the letters A to S. At the start of the search, and when a solution has been found, the program pauses to allow the position to be noted down. Any key depression will cause the search to be resumed. The program terminates when there are no further possible arrangements.

The first solution will be found in just over four minutes, and the second in another two and a half minutes (at 4 Mhz). Further solutions will require considerably more patience, but those interested enough may wish to run the program exhaustively in order to uncover all the possible arrangements and discover if there exists more than one unique solution.

PROGRAM LISTING

```
0C80    3E 0C                   LD A, 0CH
0C82    F7                      RST ROUT
0C83    01 11 00                LD BC, 11H
0C86    11 1A 08                LD DE 081AH
0C89    21 A9 0D                LD HL, TITLE
0C8C    ED B0                   LDIR
0C8E    3E 3F                   LD A, 3FH
0C90    CD 28 0D                CALL HEXGON
0C93    3E 41                   LD A, 41H
0C95    06 13                   LD B, 13H
0C97    21 99 0B                LD HL, 0B99H
0C9A    77          TABLE       LD (HL), A
0C9B    3C                      INC A
0C9C    23                      INC HL
0C9D    10 FB                   DJNZ TABLE
0C9F    DF 7B                   SCAL BLINK
0CA1    3E 20                   LD A, 20H
0CA3    CD 28 0D                CALL HEXGON
0CA6    03          FORWRD      INC BC
0CA7    3E 14                   LD A, 14H
0CA9    B9                      CP C
0CAA    28 4D                   JR Z, SOLN
0CAC    CD 15 0D                CALL HEXLOC
0CAF    3E 40                   LD A, 40H
0CB1    21 98 0B                LF HL, 0B98H
0CB4    3CONXTNUM               INC A
0CB5    FE 54                   CP “T
0CB7    28 44                   JR Z, REPLAC
0CB9    23                      INC HL
0CBA    BE                      CP (HL)
0CBB    20 F7                   JR NZ, NXTNUM
0CBD    12                      LD (DE) A
0CBE    36 20                   LD (HL), 20H
0CC0    F5                      PUSH AF
0CC1    E5                      PUSH HL
0CC2    21 5B 0D                LD HL, CHKLST
0CC5    09                      ADD HL, BC
```

0CC6	C5		PUSH BC
0CC7	4E		LD C (HL)
0CC8	AF		XOR A
0CC9	B9		CP C
0CCA	20 05		JR NZ, SUMCHK
0CCC	C1		POP BC
0CCD	E1		POP HL
0CCE	F1		POP AF
0CCF	18 D5		JR FORWARD
0CD1	D5	SUMCHK	PUSH DE
0CD2	21 6F 0D		LD HL, SUMLST
0CD5	09		ADD HL, BC
0CD6	46		LD B, (HL)
0CD7	AF		XOR A
0CD8	57		LD D, A
0CD9	C5	ADD	PUSH BC
0CDA	23		INC HL
0CDB	06 00		LD B, 0
0CDD	4E		LD C, (HL)
0CDE	D5		PUSH DE
0CDF	CD 15 0D		CALL HEXLOC
0CE2	1A		LD A, (DE)
0CE3	D1		POP DE
0CE4	82		ADD A, D
0CE5	D6 40		SUB A, 40H
0CE7	57		LD D, A
0CE8	C1		POP BC
0CE9	10 EE		DJNZ ADD
0CEB	D1		POP DE
0CEC	C1		POP BC
0CED	E1		POP HL
0CEE	FE 26		CP 26H
0CF0	28 04		JR Z, RETURN
0CF2	F1		POP AF
0CF3	77		LD (HL), A
0CF4	18 BE		JR NXTNUM
0CF6	F1	RETURN	POP AF
0CF7	18 AD		JR NXTNUM
0CF9	DF 7B	SOLN	SCAL BLINK
0CFB	18 08		JR GOBACK
0CFD	1A	REPLAC	LD A, (DE)
0CFE	CD 20 0D		CALL TABLOC
0D01	77		LD (HL), A
0D02	3E 20		LD A, 20H
0D04	12		LD (DE), A
0D05	0B	GOBACK	DEC BC
0D06	AF		XOR A
0D07	B9		CP C
0D08	28 0A		JR Z, END
0D0A	CD 15 0D		CALL HEXLOC
0D0D	1A		LD A, (DE)
0D0E	CD 20 0D		CALL TABLOC
0D11	77		LD (HL), A
0D12	18 A0		JR NXTNUM
0D14	76	END	HALT
0D15	E5	HEXLOC	PUSH HL
0D16	21 33 0D		LD HL, HEXLST-2
0D19	09		ADD HL, BC

```
0D1A    09                          ADD HL, BC
0D1B    5E                          LD E, (HL)
0D1C    23                          INC HL
0D1D    56                          LD D, (HL)
0D1E    E1                          POP HL
0D1F    C9                          RET
0D20    C5              TABLOC      PUSH BC
0D21    21  58  0B                  LD HL, 0B58H
0D24    4F                          LD C, A
0D25    09                          ADD HL, BC
0D26    C1                          POP BC
0D27    C9                          RET
0D28    06  13          HEXGON      LD B, 13H
0D2A    21  35  0D                  LD HL, HEXLST
0D2D    5E              LOOP        LD E, (HL)
0D2E    23                          INC HL
0D2F    56                          LD D, (HL)
0D30    23                          INC HL
0D31    12                          LD (DE),  A
0D32    10  F9                      DJNZ LOOP
0D34    C9                          RET
0D35    DE  08  E2  08  HEXLST
        E6  08  5C  09
        60  09  64  09
        68  09  DA  09
        DE  09  E2  09
        E6  09  EA  09
        5C  0A  60  0A
        64  0A  68  0A
        DE  0A  E2  0A
        E6  0A
0D5B    00  00  00  01  CHKLST
        00  00  00  05
        0A  00  00  0E
        15  19  1E  25
        2C  31  35  00
0D6F    00  03  01  02  SUMLST
        03  04  04  05
        06  07  03  01
        04  08  06  02
        05  06  09  0A
        0B  03  03  07
        0C  04  02  05
        09  0D  06  04
        05  06  09  0A
        0E  06  05  06
        07  0A  0B  0F
        04  02  06  0B
        10  03  08  0D
        11  04  04  09
0DA9    0E  12          TITLE       DEFM /THE MAGIC HEXAGON/
        54  48  45  20
        4D  41  47  49
        43  20  48  45
0DB9    58  41  47  4F
        4E
        ---------------------------------------------------------------------------
```

By J. Haigh

SINGLE STEP S xxxx

The Single Step command initially uses part of the machine code used by the Execute command, described in article 2. It enters the Execute routine at the point at which it throws away the return address by POPing it into AF; it thus misses out the section which sets the workspace byte CONFLG (£0C26) to -1, and this remains at zero. Continuing with the Execute machine code routine, the Single Step command saves the specified start address in the monitor workspace, loads registers BC, DE, HL, AF and SP from the register save area in the workspace, and pushes the start address onto the top of the stack. The AF registers are then saved while bit 3 of port 0 is set; this activates a TTL circuit which sends a non-maskable interrupt to the processor after four M1 cycles. The AF registers are recovered, and this is followed by a return-from NMI instruction (RETN, ED 45), which, because the start address was pushed on the top of the stack, causes the program to start executing at this address. Three M1 cycles have now occurred (one each time a byte is fetched from memory), so as soon as the next instruction is started the NMI line is activated, and the processor is interrupted at the end of this instruction.

This causes the processor to jump to the NMI handling routine. Here bit 3 of port 0 is reset, and the value stored at CONFLG is tested. If the value is not zero the program must have arrived at this point from an Execute command, and the routine continues as described in the second article. If CONFLG is zero the top 10 bytes of the user stack, which contain the value of the program counter register for the next instruction of the program being single-stepped (pushed on the stack as the return address by the NMI) and the AF, HL, DE and BC registers, are copied to the register save area in the monitor workspace. The contents of these registers, together with the user stack pointer, the interrupt register and the index registers IX and IY, are then printed on the screen.

In Nas-Sys 1 the routine which prints the registers is part of the NMI/Breakpoint handling routine, but in Nas-Sys3 it is written as a separate subroutine which can be accessed by command P or called from other programs. The format of the register print out is also different in the two monitors. In Nas-Sys 1 only the registers are printed out, in the order:-

SP PC AF HL DE BC I IX IY

followed by a string of letters indicating which of the bits in the flag register are set. In Nas-Sys 3 the display of each of the first six register pairs is followed by the sixteen bit value held in the memory location to which the register pair points. Thus if the H register contains £10, the L register contains £00, and memory locations £1000 and £1001 contain £22 and £33 respectively, they will appear in the register display as

1000 3322

Note that the memory bytes are printed in the order (1001), (1000). This is in line with the order in which the registers themselves are displayed, because if the contents of HL were stored at address £1000 by the instruction.

LD (£1000), HL   22 00 10

register L would be stored at £1000, register H at £1001.

After the registers have been displayed, the program returns to the monitor at the routine PARSE, where it waits for a further command input. If you enter command E or S without specifying an address, the execute routine will pick up the address stored in the workspace at £0C69, which is the value of PC saved last time around. This means that execution or single-stepping can be continued from the point where the program operation was suspended by a breakpoint or by the last single step. Once you have carried out one single step you do not need to re-enter the S command – just pressing the Newline key will produce another single step if the last command entered was S. This is particularly useful with Nas-Sys 3, which has a repeat-key routine; just holding Newline down will cause the program to single-step at a rate which can be controlled by the value stored at £0C30,£0C31

TABULATE T xxxx yyyy zz

This command prints out the hexadecimal values of the machine code stored between addresses xxxx and yyyy. The starting address (in hex) and first printed, and this is followed by eight bytes of data. A new line is then started, the updated address is printed, followed by further data bytes, and so on. The process continues until zz lines have been printed, when it pauses until a key is pressed. The routine terminates when the updated address reaches yyyy, or if 'escape' (SHIFT/NEWLINE) is pressed during a pause. Nas-Sys 1 calculates the 'checksum' of the address and data bytes on a line, and outputs this checksum. The checksum cannot be seen in the tabulation because the cursor is backspaced over the two hex digits of the checksum. If you send the tabulated data to a printer which does not backspace, the checksum will be printed. The checksum is quite useful for the direct entry of machine code as it makes it easier to spot errors. You can make your Nascom print the checksum on the screen by routing the output through a short program which suppresses the backspaces, or, if you have access to an EPROM programmer, by removing them from the monitor.

The Nas-Sys 3 tabulate command does not evaluate the checksum, but it has extra facilities for formatting the displayed data. The number of bytes of data output per line can be controlled by a fourth argument; if this argument is nn, there will be 8+nn bytes per line. In addition to the hexadecimal data, Nas-Sys 3 outputs the ASCII or Graphics codes of the data (codes in the ranges £00 - £1F, £7F - £9F, and £FF are output as “.”). A fifth argument, hhll, may be entered to suppress either the hexadecimal (if hh is not zero) or ASCII (if ll is not zero) part of the listing.

If you wish to edit a tabulated listing you must quit the Tabulate command by typing ‘Escape’, and then enter M to get into the Modify routine. You will now be able to move the cursor with the control keys and edit the tabulation. If you are using Nas-Sys 3 it is best to suppress ASCII part of the listing. As this will interfere with the modify command, either by producing error messages or, if the ASCII section contains a full stop, by terminating the modify routine.

USER INPUT/OUTPUT U

Input and output is accessed via pointers to tables which list the routines to be called. With the pointers, which are stored in the workspace at £0C73 (output) and £0C75 (input), set to normal values, as on power-up or after a RESET or N command, input scans the keyboard and serial port while output is sent to the screen. The U command resets the pointers so that routines provided by the user are called before the input or output is performed. The user routines can reside anywhere in memory; the start address of the input routine must be stored in the workspace at £0C7B, that of the output routine at £0C78. These locations normally contain the address of a return instruction in the monitor, so that using the U command without providing the addresses of your routines has no effect.

Although the I/O procedure also uses the remaining routines in the tables, if for any reason you do not wish these routines to be called (for example, you may wish to suppress the screen output), you merely have to set the carry flag in the user routine; the remaining routines will then be skipped. This can have unfortunate consequences – if a printer routine carries out tests which leave the carry flag set for certain characters, these characters will not appear on the screen.

VERIFY V

The address stored in the command table for the VERIFY command is the same as that for the READ command. The two commands use the same code, except that as each data byte is obtained the value stored at location £0C2B, which contains the last command letter entered,is tested to see if it an R. If it is not, the data bytes are not stored in memory, and faulty data cannot corrupt data already in store. If you are calling the READ routine from a program you must store an R at £0C2B or data will not be loaded.

WRITE W xxxx yyyy

The write routine first switches on the cassette LED; it then waits for approximately one second, which allows the cassette recorder speed to stabilise if you are using the LED signal to control your recorder. The output table pointer is then reset to its normal value, so that output is sent only to screen. This ensures that a user routine will not interfere with the operation of the write routine. The address stored at £0C73 is saved on the stack, and restored at the end of the write routine.

After 256 nulls have been output to the serial port the data is output in blocks of 256 bytes, preceded by a null and four 'FF's, a header which gives the start address of the block, the length and number of the block, and a checksum for the header data. The start address and the length and number of the block are printed on the screen, but all other bytes are output only to the serial port; output through this port is by direct call to the serial routine, and does not use the output table and its pointer. The block of data is then output, followed by ten nulls. The purpose of all these nulls is to ensure that if several bytes of data are missed when the tape is read, the start of the next block header, marked by four 'FF's, will not be missed. If there were no nulls the READ routine would continue until it detected the correct number of bytes, accepting one or more of the 'FF's, and it would then miss the following block and only start reading again when the next block start was detected.

When all the data has been written the routine jumps to the end of the READ routine, using the same code to reset the output pointer. If you wish to make several copies of the same data, which is always advisable, you do not need to re-enter the arguments – just enter W and the previous arguments, which are still at 0C0C and 0C0E, will be used again. If you are too lazy to do even this, a simple modification to the WRITE routine will make it unnecessary. Instead of jumping to the end of READ, first execute the following code:-

```
21 10 0C    LD HL, £0C10     ; Point HL to ARG3
35          DEC (HL)         ; Decrement ARG3
C2 XX XX    JP NZ, REWRT     ; Continue if non-zero
C3 YY YY    JP RDEND         ; Jump to end of READ
```

For Nas-Sys 1 the address XXXX is £04EF; for Nas-Sys 3 it is £502. Address YYYY is the end of the READ routine, where the table pointer is reset. If you use the modified READ routine given in the last article, you will find that there is room for extra WRITE code at the end, and the final jump can be a relative one. Incidentally, there is a misprint in the listing on page 30 of the last issue; from line 32 it should read:-

```
CF   R2          RST RIN        ; GET CHARACTER
3C               INC A          ; IS IT FF?
20FA             JR NZ, R1      ; IF NOT, KEEP LOOKING
```

\- - - - - - - - - - - - -

```
10                    R I N G S   O F   H A N O I
20               B Y   S .   H E A D ,   J U N E   1 9 8 1
30 REM
40 CLEAR 500: DIM A(12,2), TR(2,1)
50 REM
60 REM **. . . . . . . Print out rules . . . . . . .
70 CLS: GOSUB 800: PRINT
80 PRINT "The object of the game is to move"
90 PRINT "the rings from pile A to pile B or C
100 PRINT: PRINT "Only the top ring ";
110 PRINT "can be moved."
120 PRINT:PRINT "Larger rings cannot be ";
130 PRINT "put over smaller rings.
140 PRINT:PRINT:PRINT "PRESS ENTER WHEN READY.";
150 INPUT IN$
180 REM
190 REM**. . . . . . . . . Initialise . . . . . . . . .
200 CLS
210 FOR I = 0 TO 10
220 A(I,0) = 1:A(I,1) = 0:A(I,2) = 0
230 NEXT I
240 TR(0,0) = 1:TR(0,1) = 1
250 TR(1,0) = 11:TR(1,1) = 10
260 TR(2,0) = 11:TR(2,1)  =  10:MC = 0
270 GOSUB 600
280 GOSUB 800
290 SCREEN 6,15
300 PRINT "PILE A", "PILE B", "PILE C";
310 REM
320 REM ** . . . . . . . . .  Input move  . . . . . . . . . .
400 GOSUB 920: SCREEN 2,2: PRINT "MOVE FROM ";
410 INPUT N$ 420 GOSUB 1300:TF = IN:IF TF < 0 THEN 400
430 IF A(10,TF)  <>  0  THEN 470
440 SCREEN 1,1
450 PRINT "NO RINGS ON THAT STICK CHUM"
460 GOTO 400
470 GOSUB 920:SCREEN 2,2:PRINT "TO PILE";
480 INPUT IN$
490 GOSUB 1300:TT = IN:IF TT  <  0  THEN GOTO 470
500 GOSUB 1000
510 GOTO 400
580 REM
590 REM ** . . . . . . .  Initialise graphics  . . . . . . .
600 CLS: IT = 0: IP = 14:GOSUB 630
610 IT = 1:IP = 42: GOSUB 630
620 IT = 2:IP = 70: GOSUB 630: GOSUB 800
630 FOR  I = 0 TO 32
640 FOR J = 0 TO A(I/3,IT)
650 SET (J + IP, I+9)
660 SET (IP –J, I+9)
670 NEXT J
680 NEXT I:RETURN
780 REM
790 REM ** . . . . . . .  Print title  . . . . . . .
800 TL$ = "TOWERS OF HANOI   MOVE "+STR$(MC)
810 FOR I = 1 TO LEN(TL$)
820 POKE 3029+I,ASC(MID$(TL$,I,1))
830 NEXT I
840 RETURN
```

```
880 REM
890 REM ** . . . . .  Clear line subroutine  . . . . .
900 SCREEN 1,1:GOSUB 930
910 SCREEN 3,1:GOSUB 930
920 SCREEN 2,2:GOSUB 930:RETURN
930 FOR I = 1 TO 4: PRINT "                        ";:NEXT I
940 RETURN
945 REM
950 REM **  . . .  Move a ring subroutine  . . .
1000 IF TR(TT,1) = 0 THEN 1030
1010 IF TR(TF,1) > TR(TT,1) THEN 1150
1020 IF TF = TT THEN 1130
1030 MC = MC+1: GOSUB 800
1040 A(TR(TT,0)-1,TT) = A(TR(TF,0),TF)
1050 A(TR(TF,0),TF) = 0
1060 TR(TT,1) = TR(TF,1)
1070 TR(TT,0) = TR(TT,0) -1
1080 TR(TF,0) = TR(TF,0) + 1
1090 TR(TF,1) = A(TR(TF,0),TF)
1100 IF A(10,0) + A(10,1) = 0 THEN 1180
1110 IF A(10,0) + A(10,2) = 0 THEN 1180
1120 GOSUB 1400:RETURN
1130 SCREEN1,1:PRINT "DON'T BE SILLY":RETURN
1150 SCREEN1,1:PRINT "YOU CANNOT PUT LARGER";
1160 PRINT " RINGS OVER SMALLER ONES!":RETURN
1180 PRINT "CONGRATULATIONS, YOU HAVE DONE IT"
1190 PRINT "             OPTIMUM 1023 ";
1200 PRINT "YOU RATING IS ";102300/MC;"%"
1210 PRINT "       DO YOU WANT ANOTHER GO (Y/N)?";
1220 INPUT IN$
1230 IF LEFT$(IN$,1) = "N" THEN END
1240 IF LEFT$(IN$,1) = "Y" THEN GOTO 40
1250 GOTO 1220
1260 REM
1270 REM **  . . . . .  Convert key to number  . . . .
1300 IF IN$ = "A" THEN IN=0:RETURN
1310 IF IN$ = "B" THEN IN=1:RETURN
1320 IF IN$ = "C" THEN IN=2:RETURN
1330 IN = -1000:RETURN
1340 REM
1350 REM **  . . .  Move ring graphic sub  . . .
1400 ZZ = TF:GOSUB 1420: ZZ = TT
1420 IF ZZ = 0 THEN IX = 14
1430 IF ZZ = 1 THEN IX = 42
1440 IF ZZ = 2 THEN IX = 70
1450 FOR JX = TR(ZZ,0)*3 TO TR(ZZ,0)*3+2
1460 FOR J = 1 TO A(TR(TT,0),TT)
1470 IF ZZ = TT THEN 1510
1480 RESET (J + IX,JX+6)
1490 RESET (IX-J,JX+6)
1500 GOTO 1530
1510 SET(J+IX,JX+9)
1520 SET(IX-J,JX+9)
1530 NEXT J
1540 NEXT JX
1550 GOTO 900
```

PROGRAM POWER

NASCOM 1 & 2

Nasprint 80

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

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

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

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

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

New Fase (16K/MC/G)

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

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

Starship Command (16K/B/G)

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

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

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

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

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

Moon Raider (MC/G)

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

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

ALL PROGRAMS SUPPLIED ON CASSETTE IN CUTS/KANSAS CITY FORMAT

BARCLAYCARD VISA

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

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