LEVEL SEVEN DOCUMENTATION

This document is believed to be correct and every effort has been made to ensure the information contained herein is accurate. However, no responsibility can be accepted for inaccuracies or omissions.

LEVEL SEVEN is a new and very powerful software package for the TRITON computer system. It consists of a 2k monitor with 24 functions and an 8k extended BASIC.

AVAILABLE EXCLUSIVELY FROM:

TRANSAM

TRANSAM COMPONENTS LTD.
12 CHAPEL STREET, LONDON, NW1
TEL: 402 8137

LEVEL SEVEN FIRMWARE

LEVEL SEVEN BASIC & MONITOR

PAUL BAXTER (TRANSAM)

LEVEL SEVEN MONITOR

The first section of this documentation deals with the operation and application of the 7.2 monitor, the second section is devoted to the extended BASIC.

INTRODUCTION

The Triton level seven monitor is a 2k operating program intended to carry out essential management routines and provide the user with a powerful set of commands for entering, debugging, and running machine code programs. It also has single keystroke vectors to TRAP for more extensive assembly language writing and to level seven BASIC for programming in the high level language.

GETTING IT GOING

The level seven firmware comes in 10x2708 type EPROMS, the monitor is supplied in 2 EPROMS marked MONITOR 7.2 A and MONITOR 7.2 B. The A rom is inserted into the socket on the main board for IC21 which is the socket furthest from the expansion socket, the B rom is placed in the socket for IC24 (the one nearest to the expansion socket). Note: Users with a parallel version will have the 7.2A chip replaced with one marked 7.2P. This goes in the same position as the A.

The level seven BASIC is in 8x2708 EPROMS and needs to be fitted in an EPROM card on the motherboard. The card may be placed at any position on the motherboard, but the jumper (or switch) should be set at the topmost position (block 7) which is the position nearest to the plug on the card. The EPROMs are placed in the card starting BASIC 7.2 A in the position of IC7 (top right hand socket), BASIC B goes in IC8 next to it, BASIC C goes below A etc.

Take care when inserting the roms as it is very easy to bend the pins when inserting. Check the roms again, then plug the rom card into the motherboard and switch on. The welcome message below should be displayed.

```
TRITON V7.2
END:XXXX     (XXXX is a hexadecimal address)
FUNCTION? P Q J E R C G A D U H L F T S I O W M V X N Z K
-
```

Triton is now waiting for you to enter one of the letters shown in the FUNCTION message. The message END:XXXX is to inform the user of the amount of memory on-line at power-up. This is the amount of memory that will be assumed until another reset occurs. It is important to power up the motherboard at the same time or before the main board as otherwise the Triton will only assume memory on the main board. At power-on Triton performs a non-destructive memory test to determine how much memory is available. This means that RESET can be pressed at any time without the fear of destroying the contents of memory. This also means that memory will contain a random pattern at switch on. If it is desired to set memory to a certain value refer to the E function.

XXXX is the address at which memory failed. This will be 2ØØØ withthe on-board ram or 4ØØØ with one ram card in position 1 on the motherboard and so on.

Reference is made in following sections to 'The main loop'. This is the waiting section of the monitor. Triton prints the FUNCTION message and awaits an entry from the user.

USING TRITON

The functions and their use are described below; it is recommended that these are studied carefully if you are used to any of the previous monitors as there are some differences in operation. After the description is an example of the monitor in action. This should enable you to take full advantage of the many features of the level seven monitor.

P - program.

This function allows entry and displaying of the content of memory in hexadecimal (shortened to hex) - its main use is for the entry and checking of small blocks of memory. For longer programs, the Q function is more suited.

To call this function, type P. The message 'START:' will be printed below and the computer waits for you to enter a start address for programming. The address is entered in Hex and may be up to 4 characters long. If more than 4 characters are entered the last four entered are used; if less than 4 are entered leading zeros are appended (e.g. If 23 is entered the address ØØ23 is used). If an invalid character is entered it terminates the address (e.g. if X is entered the address will be ØØØØ).

If during entry, a mistake is made, the character can be deleted by use of the DEL key. The character can then be re-typed and when the new address is correct press the RETURN key and the address entered will be displayed below together with the data in that location in the form:
ØØØØ 31

Note that both the address and data are in hex. The user may then enter new data, or retain the old data. To retain the data in that location press RETURN; the computer will then step to the next location in memory, displaying it as before. If you wish to step back one location the up-arrow (↑) should be entered as the first character (shift (↑)). If the value in that location is to be changed, it should be entered followed by RETURN or up-arrow (depending on whether you wish to step forward or back). If the data is only 1 character a leading Ø is added if 2 characters or more the last 2 characters are used. This method for entering addresses and data applies to the other functions with the exception of Q (see next section). To exit from the program mode (or, indeed from any function) enter Control-C. (The CTRL key held whilst C is pressed). The FUNCTION message will be displayed.

Q - Quick Program

This function has been included as a tool primarily for the entry of programs or strings of data and is not intended for changing single bytes. To enter the function from the main monitor loop type 'Q' - TRITON will respond with 'START:' You can then enter an address as detailed above. When the 'return' key is pressed the monitor will respond as in the program mode and you may step backwards or forwards as above. The difference comes in entering data - instead of having to press 'return' between bytes, the monitor will enter and step forward as soon as two digits have been entered - in addition invalid characters (not Ø-9 or A-F) will be ignored. If a valid character is typed incorrectly you cannot use the 'Del' key - instead you must press 'return' followed by '^' to go back to that location again. To exit from the function type control C as above. You may step forward or back through memory after entering the first digit without altering the content of that location.

BOOKS ON ASSEMBLY LANGUAGE PROGRAMMING AND BASIC AVAILABLE FROM TRANSAM

REF.NO	TITLE	PRICE £
CB98	A QUICK LOOK AT BASIC	10.95
CB41	ADVANCED BASIC	6.00
CB97	ACCENT ON BASIC	4.95
CB163	BASIC PRIMER	6.95
CB65	BASIC COMPUTER GAMES	5.50
CB95	BASIC - A UNIT FOR SECONDARY SCHOOLS	4.45
CB40	BASIC BASIC	6.50
B18	BASIC A HANDS-ON METHOD	7.78
B22	BASIC FROM THE GROUND UP	6.60
B24	BASIC WITH BUSINESS APPLICATIONS	9.74
B25	BASIC WITH STYLE (PROGRAMMING PROVERBS)	4.07
B26	BASIC WORKBOOK	4.39
B206	BITS 8080 PROGRAMMER CODING PAD	1.50
CB37	BEGINNING BASIC	6.50
CB150	PROGRAMS THAT WORK (IN BASIC)	3.15
B79	GUIDED TOUR OF PROGRAMMING IN BASIC	4.13
CB70	GAME PLAYING WITH BASIC	5.50
CB36	INSTANT BASIC	7.20
CB161	MORE BASIC COMPUTER GAMES	5.50
CB168	MICROSOFT BASIC	6.75
CB126	PIMS (Personal Information Management System (BASIC)	6.81
CB28	STEP BY STEP INTRODUCTION TO 8080 SYSTEMS	6.45
CB39	SOME COMMON BASIC PROGRAMS	6.45
CB48	SCELBI 8080 GOURMET GUIDE & COOKBOOK	7.95
CB96	TEACHERS' BASIC MANUAL	6.45
CB184	THE LITTLE BOOK OF BASIC STYLE	5.25
CB31	8080 BUGBOOK	7.65
CB50	8080 PROGRAMMING FOR LOGIC DESIGN	6.30
CB53	8080/8085 ASSEMBLY LANGUAGE PROGRAMMING	6.95

****

J - Jump to BASIC L7.2

This function causes a jump to BASIC level seven. This function will respond with the message:
BASIC L7.2.
END ADDRESS ?
The computer will then wait for you to enter an address in hex at which you wish to stop BASIC

using memory, this is used to free memory for use by machine-code subroutines etc. If you wish BASIC to use all memory, enter a RETURN and the message 2554 BYTES FREE
READY
>
will be displayed and the computer is now in BASIC (2554 bytes is the message with 3k of on-board ram). To exit to the monitor enter control-C.
If an address less than 17ØØ or greater than the end of memory is entered, all memory is used.
This function must be used before X and will always clear memory whereas X will not.

E - Erase Memory

This function is used to initialise a block of memory to any given value. When this function is called up, the computer will request a start address, an end address and a byte value - the start and end addresses are both 4 digit hex-addresses as detailed previously, the byte value is any two digit hex byte. When 'return' has been pressed after entering the byte, the monitor will proceed to fill memory between the two addresses (inclusive) with that byte. When the function is finished the oscillator will sound.
N.B. For correct operation the start address must be below the end address as the byte is written to memory commencing at the 'START': address and incremented until it equals the end address. Therefore if the 'START:' is given at 3000 and the end as 2000 all memory will be set to the value except between 2ØØØ - 3ØØØ! This also applies to the S,H.F and N functions.

R - Register Operation

When this function is entered the monitor will wait for another key to be pressed to indicate which register operation is required. If the space bar is hit all registers and flags will be displayed - a typical printout is shown below:

```
A   C   B   E   D   L   H   SP    PC
ØØ  1Ø  FF  57  Ø5  57  AØ  1468  1647 SAP
```

You will note that the register pairs (HL,DE,BC) are displayed low order byte first - the letters refering to the register PC - Program counter, SP - Stack pointer are printed above. The letters following the program counter represent the flags - if the flag is set the letter is present. The 5 are listed below:

S = Sign Flag Z = Zero A = Auxilary Carry
C = Carry P = Parity

To modify a register from the main loop, type 'R' followed by the letter of the register you wish to modify e.g. to modify the accumulator, type 'RA'. The value of the accumulator would then be printed with the cursor one space along, you may then enter a new value for the accumulator followed by 'return'. If you do not wish to modify the register you may exit by typing 'return' without entering any data. The monitor will signify an error condition and the register will not be changed. The same applies to the 16 bit registers SP or PC. To modify these registers type 'P' for program counter and 'S' for stack pointer. To display the

flags from the main loop, type 'RF' and the flags set, if any, will be displayed. If you do not wish to change the flags, enter control C. To clear all the flags, type return or to set some flags type in their letters (as detailed above). The letters need not be in any order. An invalid character in the string will not be flagged as an error but will simply terminate the update i.e.: PZBCA will set the parity and zero flags only, resetting the others.

C - Continue from a breakpoint

This function requires no other data and loads up the registers in the 8080 with the pseudo-registers in memory (those displayed in the R function). The last register to be loaded is the program counter, causing an immediate jump to the address held in PC register.

G - Goto Address

This function will execute a machine code program at a specified address. When the letter is entered a start address will be requested; once the address has been entered by pressing 'return' the monitor checks for any errors and if none are found a carriage return followed by a line feed will be printed and the program will start to execute.

A - ASCII String into Memory

This function allows you to enter a string of characters into memory directly from the keyboard. The monitor will request a start address. When this has been entered correctly and the 'return' key is pressed the monitor will print a carriage return followed by a line feed and wait for you to enter some characters. As each character is entered it is displayed and entered into memory starting at the address specified. To delete characters use the 'DEL' key or control H. When the string has been entered an 'EOT' character (Ø7H) must be entered to denote the end of the string. To obtain this character type control D - this will be entered into memory and the monitor will then print END:XXXX. Where XXXX is the address of the byte after the 'EOT' character, this should be the begin address of any subsequent strings. If 'EOT' is not required on the end of a string, terminate it with 'control C'. The 'control C' will not be entered and the end address is that of the byte immediately after the last character.

D - Display String

This function imitates a PSTRNG monitor utility (see later) and will print out any string from the address entered until it encounters an 'EOT' character such as those entered using the 'A' function. When this function is entered the start address of the string will be requested. When this has been entered a CR/LF sequence will be printed followed by the string. Unless it is known that a string exists at the entered address, it is best to use the 'N' function to find any strings. When this function is entered the start address of the string will be requested. When this is entered a CR/LF sequence will be printed followed by the string.

U - Useful function (Port Operations)

This function allows you to read and write to any port. When this function is entered the port number will be requested. This must be entered as a two digit hex number followed by 'return'; the data at that port will be displayed and you may either hit 'return', which will display the port data again (this is useful to see if the data at the port is changing) or alternatively, a two digit hex number may be entered - followed by 'return'. This data will then be written to the port and the data at the port will be read and displayed again. To exit from this function enter control C.

H - HEX Dump

This function produces a formatted hex-dump of memory between any two addresses. This allows for greater density than the 'L' function. The monitor will request a start address followed by an end address and will then produce a dump- of memory (in hex) from the start address to the end address (inclusive). The dump is formatted so that the start of each line (with the exception of the first) will have the address XXXØ; therefore all lines will contain 16 bytes (except the first and last). To exit from this function before it has finished type control C. To halt the printout temporarily enter control 'S' (X-OFF) to resume printout type control 'Q' (X-ON). These actions apply to all printout with the addition that when the printout has been halted with control 'S' a new line can be started by pressing the return key whereupon a CR/LF sequence will be printed. This is most useful when the output is diverted to a teletype or similar printer when a long line would otherwise be overtyped at the end stop. The printout can then be resumed with control 'Q'. When the dump has finished the oscillator will sound to inform you.

L - List Memory

This function will list short sections of memory by simply entering an address. When this function is entered a start address will be requested from the user. When this has been entered that address will be printed followed by the data in that location. This sequence will be repeated on subsequent lines for the next 15 locations and below that the prompt 'more?' will be printed. If you type 'Y' the next 15 locations will be listed, if you enter any other letter the function corresponding to that letter will be entered, if any other character is typed (e.g. the space bar) the error message will be printed and the monitor will enter the main loop.

F - Find Bytes

This function allows you to search memory between two addresses for any string of bytes. When this function is entered a start and end address are requested followed by the 'BYTE:' prompt. You may then enter a string of bytes separated by commas and the last byte terminated by pressing 'return'. The monitor will then start to search between the two addresses (inclusive) for that byte string and print out the locations (if any) where the string starts. When the search has finished the oscillator will sound to inform you.

T - Trap

Trap is the Triton Resident Assembly Language Package - it resides on an 8k eprom card on the motherboard (see catalogue or documentation for details). This function will enter TRAP displaying the list of options available - no memory initialisation is performed by entry to TRAP and therefore program integrity is maintained.

S - Shift Memory

This function is more than a memory move instruction; it is an 'intelligent' move. When this function is entered a start and end address are requested from you. When you have entered the address of the block you wish to move (inclusive) the prompt 'TO:' will be printed - you then enter the address you wish to move the block to. After pressing 'return' the monitor will move the block and when finished beep the oscillator to tell you.

I - Input Tape (in standard TRITON format) to memory

This function will allow you to enter prerecorded tapes into TRITON's memory starting at 1602. When 'I' is typed from the main loop the tape control relay will be switched off and a tape header requested. When this has been entered correctly (use the DEL key where appropriate) press 'return'. The tape relay will be switched on and the message 'FILES FOUND' will be printed. Note the cursor drops onto the next line. The tape will now be searched for a header corresponding to that typed in. As each header is found it is printed out and if the two do not correspond a CR/LF is printed and the monitor waits for the next header. If the monitor recognises the header the cursor stays on the same line whilst the program is loaded. When the tape has finished loading, the message 'END' is printed out, the oscillator will sound and the monitor will re-enter the main loop waiting for another command. If during loading an error occurs (due to a noise spike or bad tape etc.,) a CR/LF and a question mark will be printed - note that only one question mark will be printed no matter how many errors are detected. This may also mean that the monitor may not recognise the end of the tape when it comes. If this happens you must use the Reset switch since the monitor does not read thekeyboard - this also applies to the 'O' function and whilst the monitor is looking for a header or loading the tape.
N.B: It is not advisable to try to load a tape when the output is directed to a printer since if the printer may not accept characters as fast as the tape is sending (30 characters per second) the header will not be printed correctly or recognised - therefore use the 'V' function first.

O - Output a program to tape (in standard Triton format)

This function will allow you to save machine code programs starting at 1602. When you type 'O' the tape relay will switch off and the tape header will be requested. When this has been entered start the tape recorder, then hit the return key - the monitor will switch on the recorder, wait for 5-6 seconds and then proceed to write the program in the TRITON standard format (see later). When the data has been written out the monitor will wait for 5-6 seconds and then switch off the relay,

beep the oscillator for a moment and say 'END' before returning to the main monitor loop.

W - Typewriter Mode — Type 'W' and the monitor will print all characters entered until a control 'C' which aborts back to the monitor: The monitor will print onto the printer if it is selected.

M - Motor Switch — This function will toggle the tape-motor control relay - i.e. if it is off it will be turned on and vice versa. Once this has been done the monitor will print its function message and wait for another command.

V - VDU Switch — This function will direct the output to a printer as well as to the VDU. It therefore switches the printer output on and off. Even if you have no printer it can still be of use in slowing down the output by changing the value at locations 1402/3 the printout speed may be varied (this does not apply to V6.2P). It is initially set up to 110 Baud - for formula etc., see later. At reset the printer is off.

X - Extended Basic — This function enters BASIC without clearing memory or initialising vectors and therefore the J function should be used upon first entry to BASIC. To exit to the monitor from BASIC type control 'C'. N.B: To delete a character in BASIC you can now use the 'DEL' key instead of control H. In EDIT note the top right key (5FH) will give the delete character function.

N - Neat Dump of Strings — This function will produce a formatted text dump of memory. The monitor will request a start and end address and will produce a formatted dump between those 2 addresses (for details of format see H function). All printable upper case ascii characters (20 to 5FH) are printed as they are, otherwise a period (.) will be printed in its place - the characters are separated with spaces. When the dump has finished the tone will sound.

Z - Zap A Prom. — This function is to beused in conjunction with the Eprom programme card (available from Transam), it requests a start address and will then program the 2708 eprom with that and the next 1023 locations in memory. Two errors are possible; 'PROGRAM ERROR' - indicating that the device has a bit set to 'Zero' that should go to 'one' - this is not possible although a logic one may be changed to a zero - the device will have to be erased. The other is a 'READ ERROR' this occurs if the data programmed into the device does not correspond to that in memory after the device has been programmed. If all goes well the monitor will say 'END' and sound the tone - the tone will also be sounded in the event of an error.

K - Keyboard Unshift — This function is mainly of use to those using Triton with a printer and/or the lower case graphics Rom as a letter writer or word processor. Typing 'K' will reverse the effect of the monitor's shift on the keyboard so it will act as a normal type-

writer i.e. the lower case small - press shift to get upper case - this only affects the letters, numbers are as before. To get back to normal type Shift-K. At reset the shift is set to normal - upper case on unshift keys.

HERE ENDETH THE LESSON.....

That concludes the description of the functions - if you've ploughed through those - congratulations!

The following section contains notes on the use of the monitors various functions as examples. This is followed by notes on use, circuits, specifications, monitor routines and addresses.

SETTING AN EXAMPLE.

In this section we shall use an example given in listing 1 to demonstrate the use of the monitor. This program inputs a character from the keyboard and maps it on the VDU.

A few brief words of explanation are in order for those of you unfamiliar with this type of listing - it is called an assembler listing and is split into 6 columns:-

ADDRESS	DATA	LABEL	MNEMONIC	OPERAND	COMMENT
1	2	3	4	5	6

These listings are produced by the TRAP assembler. Columns 1 and 2 are those produced by the assembler, the first column listing the address, the data being from 1 - 3 bytes so one complete instruction is on one line - this makes the program much easier to read. The other columns are the original text, the 3rd column is optional and is used for referencing subroutines etc., the 4th and 5th columns are for the instruction and the 6th column contains any comments.

First of all clear memory to Ø so we can see what we are doing so type E 16ØØ, FFFF as the end (for good measure) and ØØ as the Byte, after a couple of seconds the function message will be printed after a beep - now list from 1600 to check this; type 'Y' to the 'more?' prompt, then type 'Q' to enter the program. Type the digits checking the addresses against those listed. Once entered use control 'C', then check using the L function to list again. If any bytes are entered wrongly change those with the 'P' function. Once we have entered the program we must enter the 2 strings - the first one starts at 1622, type A 1622 (RET) ENTER A CHARACTER (CONTROL-D). The end address should be 1634 so put the 2nd string in by typing A 1634 (RET) AGAIN (Y/N)?(CNTRL-D) -OK we should now be ready to run. Type G-16Ø2. If the program has been entered correctly the screen should clear and the message 'ENTER A CHARACTER' will be displayed on the second line - hit a key and the VDU will be mapped continuously, hit INT2 and the screen should settle down and the registers displayed - the PC will be somewhere between 16ØC and 161Ø and the HL register at 1000 which is the first position on the VDU - this is where the problem is - we haven't incremented it so the program will never end - we must insert an INX H instruction at 160D; but first we must make room for it - type S 160D 1642 160E so we are moving the block from 160D - 1642 up one byte - having done this we can insert (using the P function) 23 at 160D and run once again. The message will be printed with a graphic character preceeding it. This is because we moved the message by one byte, so we will have to change the

address at 1604/5 - we can find where the strings have moved to by using the N function from 1610 to 164F. Put the correct addresses in 1604/5 and 1618/9 - having done this try again - the correct message is issued but we seem to have problems with the character - let's try a breakpoint - the best place to put one is at 160F. Using the P function put D7 at 160F and run again - we have now found the problem - the accumulator has been loaded with the H register so we have lost the original character. However, if we save the character in the B register and map the B register instead, this should solve the problem - we need to open out another gap at 1609, so using the S function move the program up and insert thebyte 47 at 1609 and change the byte at 16ØD to 7Ø (MOV M,B).

```
ØØØØ                ; PROGRAM TO DEMONSTRATE L7
ØØØØ                ;
ØØØØ                      ORG     16ØØH
ØØØØ                ;
16ØØ                INCH    EQU     ØØØBH
16ØØ                RESPRP  EQU     ØØESH
16ØØ                SCREEN  EQU     1ØØØH
16ØØ                ;
16ØØ 44 16                  DW      ENDPRG   ;MARK END FOR TAPE
16Ø2 CF             CLRSCN:RST      1        ;CLEAR SCREEN
16Ø3 11 24 16               LXI     D,STRING;SET TO PRINT STRING
16Ø6 CD E5 ØØ               CALL    RESPRP   ;PRINT AND GET CHARACTER
16Ø9 47                     MOV     B,A      ;SAVE CHARACTER
16ØA 21 ØØ 1Ø               LXI     H,SCREEN;SET HL TO START OF VDU
16ØD 7Ø             MAP:    MOV     M,B      ;PUT THE CHARACTER ON VDU
16ØE 23                     INX     H        ;ON TO NEXT LOCATION
16ØF 7C                     MOV     A,H      ;LOOK AT HIGH BYTE
161Ø FE 14                  CPI     14H      ;END OF VDU?
1612 C2 ØD 16               JNZ     MAP      ;NO- CARRY ON
1615 CD ØB ØØ               CALL    INCH     ;YES- WAIT FOR A KEY
1618 11 36 16               LXI     D,MESSG  ;AND ASK IF MORE
161B CD E5 ØØ               CALL    RESPRP   ;GET REPLY
161E FE 59                  CPI     'Y'      ;YES?
162Ø CØ                     RNZ              ;NO- BACK TO MONITOR
1621 C3 Ø2 16               JMP     CLRSCN   ;ELSE- DO AGAIN
1624                ;
1624 45 4E 54       STRING:DB       'ENTER A CHARACTER',Ø4H
1627 45 52 2Ø
162A 41 2Ø 43
162D 48 41 52
163Ø 41 43 54
1633 45 52 Ø4
1636 41 47 41       MESSG: DB       'AGAIN (Y/N) ?',Ø4H
1639 49 4E 2Ø
163C 28 59 2F
163F 4E 29 2Ø
1642 3F Ø4
1644                ;
1644                ENDPRG:DS       Ø
1644                        END
INCH  ØØØB   RESPRP ØØE5    SCREEN 1ØØØ   ENDPRG 1644
CLRSCN16Ø2   STRING 1624    MAP    16ØD   MESSG  1636
```

(above) LISTING TWO (below) LISTING ONE

```
0000                ; PROGRAM TO DEMONSTRATE L7
0000                ;
0000                        ORG     1600H
0000                ;
1600                INCH    EQU     00BBH
1600                RESPRP  EQU     00E5H
1600                SCREEN  EQU     1000H
1600                ;
1600 42 16                  DW      ENDPRG  ;MARK END FOR TAPE
1602 CF             CLRSCN:RST      1       ;CLEAR SCREEN
1603 11 22 16               LXI     D,STRING;SET TO PRINT STRING
1606 CD E5 00               CALL    RESPRP  ;PRINT AND GET CHARACTER
1609 21 00 10               LXI     H,SCREEN;SET HL TO START OF VDU
160C 77             MAP:    MOV     M,A     ;PUT THE CHARACTER ON VDU
160D 7C                     MOV     A,H     ;LOOK AT HIGH BYTE
160E FE 14                  CPI     14H     ;END OF VDU?
1610 C2 0C 16               JNZ     MAP     ;NO- CARRY ON
1613 CD 0B 00               CALL    INCH    ;YES- WAIT FOR A KEY
1616 11 34 16               LXI     D,MESSG ;AND ASK IF MORE
1619 CD E5 00               CALL    RESPRP  ;GET REPLY
161C FE 59                  CPI     'Y'     ;YES?
161E C0                     RNZ             ;NO- BACK TO MONITOR
161F C3 02 16               JMP     CLRSCN  ;ELSE- DO AGAIN
1622                ;
1622                STRING:DB       'ENTER A CHARACTER',04H
1634                MESSG: DB       'AGAIN (Y/N) ?',04H
1642                ENDPRG:DS       0
1642                        END
```

Having done this change 1610 to FE - now all that remains is to change the addresses of the strings (again) so use the F function to search from 1600 to 1650 for 45,4E and 41,47 which are the first 2 bytes of the 1st and 2nd string respectively. Change 1604/5 and 1619/A again and run - it should now do what we said it would - and correspond to listing 2.

Now do a hex jump from 1600 to 1650 to find where the program ends and put this address in 1600/2 - we are now ready to save the program. Type 'O' and give the program any tape header start the recorder and press 'return' to send the data. To retrieve the program use the I function - whilst loading the tape deliberately stop and start the tape - a question mark should be shown on the next line - indicating a load error - only one question mark will appear. This concludes the example.

NOTES ON USE:

1. When a program is run a 'ret' instruction (C9) will return to the monitor (provided the stack is not reset).
2. When using the 'F' function a match will always be found at 1410 as this is where the match pattern is held.
3. After using TRAP it is best to do a reset as some of the interupt vectors may have been destroyed.
4. The monitor checks for a program in ROM 2 (IC22) before printing out the function message. If the first byte in the second rom is 31 (LXI SP) then the program in ROM 2 will be executed.
5. If an attempt is made to jump to non-existent memory (FF) this will be vectored back to the monitor (as with all the interrupts) so vastly diminishing the chances of destroying the program.
6. Tiny BASIC will not run with this monitor.

7. An explanation of the 'S' function - if the byte sequence 31,70,1A is held in memory and it is to be moved forward (UP) one byte to give 31,31,70,1A it must be moved from the top first otherwise we get 31,31,31,31 but if we want to move the other way we must start at the bottom.

   N.B.: TOP : FFFF BOTTOM : ØØØØ UP : TOWARDS THE TOP

   DOWN : TOWARDS THE BOTTOM

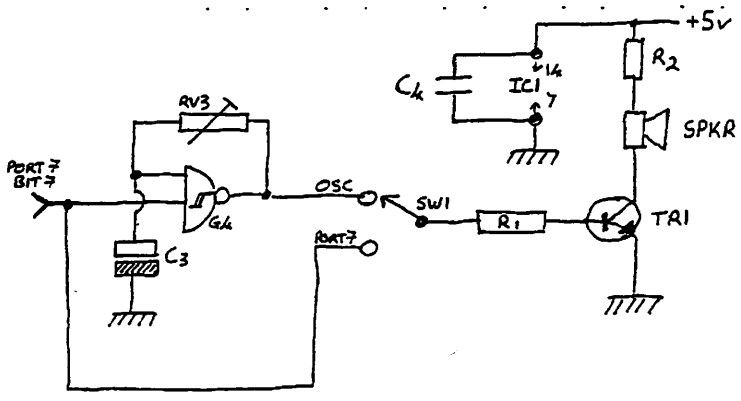
8. The G function can be used for repeatedly executing a program by loading the stack pointer with 1470 and the PC with the address of the program - NOTE that in this case a 'return' to the monitor is not permissible - instead a jump to START must be used.

9. SETTING PRINTER MODE FROM PROGRAMS

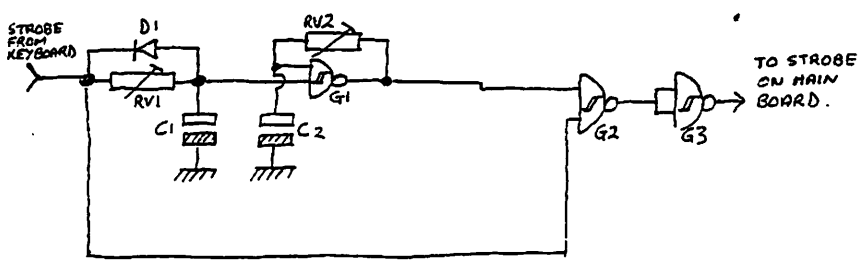
```
POKE $1401,$55 :REM PRINTER ON
POKE $1401,$AA :REM PRINTER OFF
3E55            MVI A,55H
32 Ø1 1A        STA 1401H    ;PRINTER ON

3EAA            MVI A,ØAAH   ;PRINTER OFF
32Ø1  1A        STA 1401 H
```

10. The Y function is vectored through ram at 1473 so to change the vector change the locations at 1474/5.

KEYBOARD AUTO REPEAT AND OSCILLATOR



R1 = 1K ± 10%
R2 = SEE TEXT
RV1,2,3 = 10k PRESET
C1 = 100µF 16V ELECT
C2 = 4.7µ 16V TANT
C3 = 1µ 16V TANT
C4 = .1µ POLYESTER
D1 = IN4148
TR1 = 2N3053
G1-4 = 74LS132
SPKR = 3-16 OHMS SPEAKER.



HOW IT WORKS

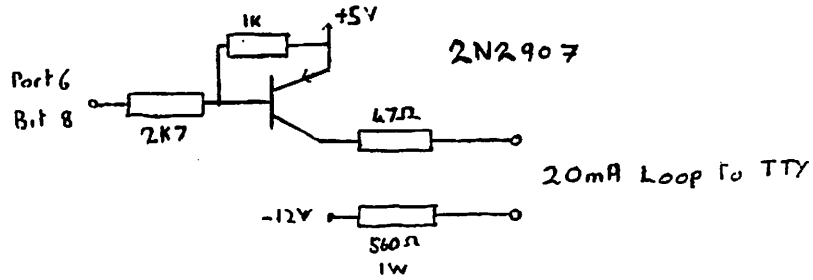
AUTO REPEAT - with no strobe (input LOW) C1 is discharged and the input is now the output is high from G1 and C2 is fully charged. G3 o/p is high and the o/p G3 (acting as an inverter) is low. When a strobe arrives C1 starts to charge up (via RV1) but whilst it is charging (time determined by setting of RV1) the o/p of G1 is still high, since both inputs to G are high the output goes low - this is inverted by G3 sending a steady strobe to the input port. If now the strobe input goes low again, C1 discharges rapidly via D1 and the output goes low again. If, however, the input is maintained for long enough for C1 to charge up, both inputs from G1 are high and the output goes low - this sends the output (by G2+3) low again, this starts to discharge C2 - when C2 has discharged the input goes low sending the output high and giving a high output again - this starts to charge C2 again and so on until the strobe is removed. ∠RV2 determines the rate of repeat.

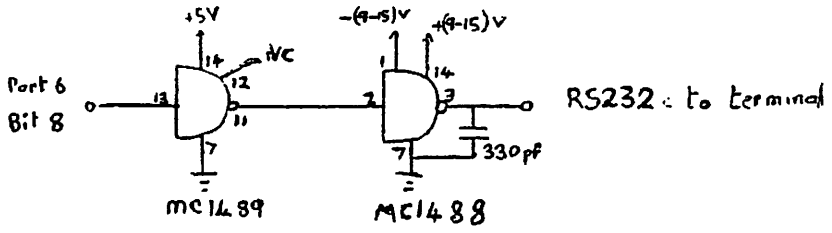
OSCILLATOR

The principle of operation is similar to that of the repeat, but when the input (BIT7) is low the output of the gate is held high, but as soon as the output goes high the circuit starts to oscillate. Since C3 is small this occurs rapidly giving an audio frequency (adjustable by means of RV3). This is amplified by TR1 to drive a small speaker. R2 should be chosen to give the appropriate tone - any value from 20R to 100R should be appropriate. Alternatively, a 30R resistor may be placed in series with a 100R wire wound potentiometer to act as a volume control. The switch included is optional but is desirable to allow the speaker to be switched direct from port 7 (for music programs etc). The oscillator should sound on an error and on the other functions detailed previously.

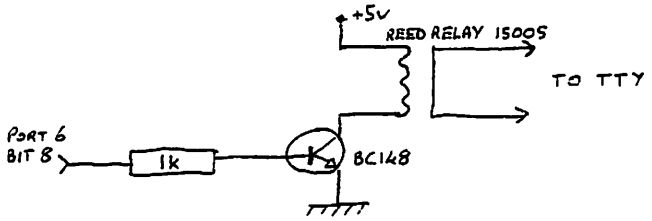
CONSTRUCTION

The circuit is reasonably simple and should pose no construction problems. Veroboard is one suitable method. The prototype was built on vero-strip board and mounted between the switches and transformer next to the speaker. The components should be mounted resistors first - the IC should be socketed - put C4 as close to IC1 as possible - this will stop weird effects when the oscillator sounds with auto repeat. Take care to insert the capacitors and diodes correctly. If the diode is reversed the repeat will start immediately.





The two circuits shown on the previous page are to interface the serial output from Triton to a 20ma or RS232 terminal. On certain TELETYPES* making the circuit below should suffice where the TTY supplies its own 20ma.

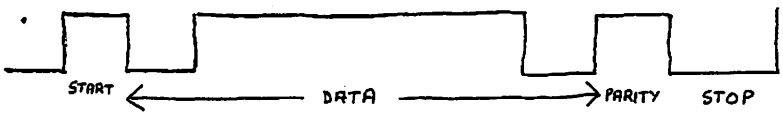


The reed relay is the same type as used on the tape I/O on Triton.

SERIAL I/O

The level seven monitor provides a serial output to a printer. This can be turned on or off by using the V function, it can also be changed in programs as mentioned previously. The serial output appears as one bit on Bit 8 of Port 6 it is overidden in software and the speed of output can be easily changed by altering a value stored in RAM - see later.

The data is inverted - i.e. logic Zero is +5 and one OV. The format is as follows:-



Each character starts with a start Bit followed by 8 data bits followed by a fake parity Bit (always 1) this is suitable for teletypes and most printers where parity checking is not needed. Two stop bits are sent at the end of each character with the exception of a carriage return where a continuous stop bit is sent for 3 character periods to allow the mechanism to return properly. This should be more than adequate for most machines.
N.B. A special version of the monitor is available for high speed parallel printers such as the TRANSAM BD80. It should be noted that the character is output to the printer (if selected) before the VDU.

When the computer is reset, the serial o/p speed is set to 110 baud - Bit time = 9.09ms = 10 characters per second, to alter the speed the formula below should be used to calculate the value - converted to hex, and put into locations 1402/3.

$$Tbit = \frac{1}{\text{Baud rate}} \text{ sec}$$

$$tcy = \frac{9}{\text{clock rate}} \text{ sec}$$

tcy - 1.255uS for 7.168MHz clock
or 0.5uS for 18MHz clock

$$n = \frac{Tbit}{tcy} - 107$$

$$\frac{\quad}{24}$$

eg for 110 Baud

$$Tbit = \frac{1}{110} = 9.09ms$$

$$n = \frac{9.09mS}{1.26uS} - 107$$

$$\frac{\quad}{24}$$

$$= 297 \quad \text{or } 0129H$$

*Trade Mark of TELETYPE CORP.

The minimum value (0001) gives 6K baud at 7mHZ or 15K baud, at 18mHZ.

TAPE FORMAT

The standard Triton format is used for input and output in the 7.1 monitor - this is detailed below.

Each Bit consists of a start Bit, 8 Data Bits, an odd parity Bit followed by 2 stop Bits. The actual record format is 5-6 seconds of continuous mark tone, 64 sync characters (ODH), program end address low, high, program, 5-6 seconds mark tone. The end address is stored in 16ØØ,Ø1, and this is automatically set up by tiny BASIC but must be preset by the user in machine code programs (see example) For BASIC L7.1 the tape routines are contained within the BASIC and these should be used for loading and storing L7.1 BASIC programs.

MONITOR ROUTINES AND ADDRESSES

MEMORY MAP:

This is a memory map of the L7.1 system.

The L7.1 monitor occupies locations ØØØØ to Ø3FF

and ØCØØ to ØFFF, the space from Ø4ØØ to ØBFF is left for user expansion (see previous section).

	Address
	FFFF
BASIC 7.1	EØØØ
TRAP	CØØØ
Available for off-board expansion	2ØØØ
On board user ram	16ØØ
Monitor/BASIC ram	14ØØ
VDU	1ØØØ
Monitor 'B'	ØCØØ
User roms	Ø4ØØ
Monitor 'A'	ØØØØ

Useful addresses are listed below:

1401	DISSW	55 = PRINTER ON	AA = VDU ONLY
1403	SPEED	Baud rate	
1410	BUFFER		
143Ø	INT 3		
1433	INT 4		
1436	INT 5		
1439	INT 6		
143C	INT 7		
1472	KEYSFT	8Ø = NORMAL	7F = TYPEWRITER SHIFT
1476	INVEC	INPUT VECTOR	
1479	OUTVEC	OUTPUT VECTOR	

PORT ADDRESS

Ø -	Keyboard Input only special routine needed
1 -	Tape I/O Uart status input
2 -	Tape I/O Uart data strobe output
3 -	Led port output On = Logic zero
4 -	Tape I/O uart receive data enable input
5 -	VDU output Strobe routine needed
6 -	Serial O/P on Bit 8 Bit 7 spare
7 -	Bit 8 = Relay Bit 7 = Oscillator (speaker) (see circuit)
8 - 15	Decoded on board by 74LS154 (needs extra decoding)

16 - FF Available for extensions

MONITOR UTILITIES

ADDRESS	NAME	COMMENTS
ØØØØ	RSTØ	Reset address. Sets up vectors - checks memory and prints initialisation message after clearing screen.
ØØØ8	RST1	Clears VDU and resets cursor returns after suitable delay with all registers intact.
ØØØB	INCH	A call routine which waits for a character to be typed in from the input device and returns with the character in the accumulator all other registers intact.
ØØ1Ø	RST2	Interrupt 2 - Jumps to INIT routine, saves registers and prints them on VDU and printer (if selected) then enters main loop.
ØØ13	OUTCH	Character output routine; outputs to the VDU (and printer if selected) the character in the accumulator, returns with all registers intact.
ØØ18	RST3	INTERUPT 3; vectored to 143Ø for user jump vector. (NOTE: on parallel printer versions this interrupt is not available to the user).
ØØ1B	INDATA	Will wait for characters to be entered from input device; echo on the VDU and put into memory pointed to by the DE register pair. On a carriage return the routine terminates; a space followed by an EOT character (04H) is stored in memory with the DE register pointing to the EOT character; the return is not stored.
ØØ2Ø	RST4	= Interrupt 4; vectored to 1433 for user operations.
ØØ23	PDATA	Prints a string stored in memory pointed to by the DE register pair; returns on EOT with all registers intact with the exception of the DE pair; points to character <u>after</u> the EOT - allowing the routine to be called again for a string following the first.
ØØ28	RST5	= Interrupt 5; vectored to 1436 for user operations.
ØØ2 B	PSTRNG	Similar to Pdata but a CR/LF sequence is output before the string is printed.
ØØ3Ø	RST 6	= Interrupt 6; vectored to 1439 for user operations.
ØØ33	PCRLF	Will output a CR/LF sequence to the output device(s); returns all registers intact.
ØØ38	RST 7	= Interrupt 7; vectored to 143C for user operations.
ØØ3B	ECHOCH	Will input a character from the input device and print it on the output device before returning it in the accumulator - all other registers intact.

The above subroutines are fixed utilities and are guaranteed to remain at these addresses if another monitor is produced.

The routines listed below are not guaranteed to remain in these positions if subsequent versions of the monitor are released.

Routines marked with an asterisk (*) are in the same position as the V5.1 monitor.

*URTOUT	ØØ3E	Will output the character in the accumulator to tape; returns registers intact.
*TAPOFF	ØØ4A	Will switch the tape motor relay off - returns with A=Ø all other registers intact.
COMP	ØØBF	Will compare the DE to the HL register pair; returns C set if DE HL Z set if DE=HL and neither set if HL DE. A is lost.
*FIVSEC	ØØC5	Waits for 5-6 seconds before returning all registers intact.
RESPRP	ØØE5	Will print the CR/LF sequence followed by the string pointed to by the DE register pair and will then wait for a character to be entered, echo it and return with the character in A; the DE register pair will point to the location of the EOT character - all other registers intact.
ERROR	Ø15B	Will print a CR/LF followed by the message 'Error' and beep the oscillator before returning to the main loop.
ERR1	Ø15E	As above but without the leading CR/LF.
PRBEEP	Ø15A1	Will print the string pointed to by the DE register pair, beep the oscillator and return to the main loop.
BEEP	Ø154	Will sound the oscillator and return to the main loop.
START	Ø174	Re-enters the monitor main loop.
STRTAD	Ø2Ø8	Will print the prompt 'START:' and get a 4 digit address in the HL pair DE + A lost - aborts to monitor on error.
GETADR	Ø2ØB	Will print the message pointed to by the DE pair and get an address as above.
G2BYT	Ø2ØE	Will print a space and wait for an address to be entered as above.
STEND	Ø22A	Will get a start and end address, end on top of stack - start below that. HL,DE,+A registers lost.
COMPI	Ø231	Similar to COMP but jumps to beep if DE-HL are equal.
PRADAT	Ø23F	Prints the address pointed to by the HL pair and the data in that location on the next line. A is lost, others intact.
PRTLOC	Ø242	Prints the data pointed to by the HL pair - A lost.
ENDADR	Ø251	Prints the message 'END:' followed by the address in the HL pair. A and DE lost.
PRADI	Ø254	Prints the address in the HL pair.

PHEXSP	Ø259	Prints the data in the accumulator followed by a space. A lost.
PRTSPC	Ø256	Prints a space. A - lost.
INSPBF	Ø27B	Prints a space then gets a string in the Buffer; returns on 'return' with a space followed by an EOT. DE points to EOT character. DE + A lost.
INBUFF	Ø270	Prints the character in A and gets a string in buffer as above. DE + A lost.
PRTDAT	Ø2AB	Prints the accumulator in hex - A lost.
TAPOUT	Ø2C2	Turns the tape relay off, requests a header - gets header, turns relay on, waits 5 seconds, sends 64 sync characters followed by header and returns. A,B,D,E lost.
TAPIN	Ø2E1	Turns the tape relay off, requests and gets a header - turns the tape relay on, prints the message 'FILES FOUND:' and searches the tape, printing all headers. Returns when a match is found. A,B,D,E lost.
URTER	Ø365	Gets a character from tape and checks for an error condition, prints out a CR/LF followed by '?' N.B. The C register must be loaded with '?' prior to calling this for the first time but not changed until tape loading is complete. B lost.
TAPRD	Ø3E6	Gets a character from the tape and prints it on the VDU - repeats indefinitely.
*TAPON	Ø3F6	Turns tape relay on. A returns 8ØH. All others intact.
GETACC	ØC14	Expects to find 2 hex digits in buffer converts and puts in A. B + DE lost.
*UARTIN	ØE62	Gets character from tape in A - other registers intact.
*TAPHDR		Turns relay off, requests and gets a header in buffer. DE + A lost.
ERRMSG	ØED6	Start address of string which prints 'ERROR'; call at ØED7 to omit space. (Use via PDATA or PSTRNG).
STRMSG	ØECF	Start address of string which prints 'START:' Use as above.
ACKA	ØFDS	Start address of string which prints 'HEADER:' Use as above.
ACKD	ØE49	Start address of string which prints 'END', Use as above.

Notes for use with programs in TRITON manual.

1. Using interrupt 3.

   Interrupt 3 is now vectored to 1430 so you should first insert the bytes C3,18,16 in locations 1430-2 before using the program.

2. Duo-Decimal program.

   Change the instruction at 1615 to RET by changing 1615 to C9.

3. Keyboard/Led Program version 2.

   Change the instruction at 160E in the alternative ending to RZ by changing the byte at 1610 to C8.

4. Alphabet twelve times using 1/0 - change instruction at 1615 as above.

5. Alphabet using memory mapping.

   Change the instruction at 1626 as in 3.

6. Modem echo test.

   Change the instruction at 1602 to call uartout by changing the bytes at 1603/4 to 3E and ØØ - change the instructions at 1606 to call uartin by changing the bytes at 1607-8 to 62 and ØEH.

7. Test tape playback program.

   Is now contained in the monitor at Ø3E6.

8. Test tape record program.

   Change the instructions at 1602, 160D, 1612 to call uartout by changing the bytes following each of these addresses to 3E and ØØ.

IMPLEMENTATION

Level Seven is available direct from TRANSAM COMPONENTS LIMITED at 12 Chapel Street, London, N.W.1. They can be supplied in exchange for programmed EPROMS (see catalogue for details) TRAP is also available from TRANSAM.

TRITON LEVEL SEVEN BASIC

INTRODUCTION

The TRITON level seven BASIC is an extended BASIC with full mathematical and string functions plus features not available on any other 8K basics together with high speed execution.

OVERVIEW

The number range (using scientific notation) is 10×10^{38} to 10×10^{-38} (approx) with an accuracy of 6½ digits (actually 24 binary bits), the last digit is rounded(if necessary). Numbers are displayed in conventional form unless the number exceeds the value 999999 or is smaller than .01 in which case the display will be in scientific format (sign, mantissa, exponent, sign, exponent).

Numbers are stored in binary resulting in faster execution, greater accuracy and more compact storage than other methods (such as B.C.D.).

Strings are also supported as an integral part of the BASIC and an extended range of string functions are available (including the very-powerful INSTR function). A full range of mathematical functions are also supported.

GETTING IN

To enter BASIC see the monitor section which should be read in full before continuing. Only if you have read this documentation fully and still have problems, should you contact TRANSAM.

You should now be in the command level of BASIC (the prompt '>' has been printed and the computer awaits your command...)

BACK TO BASICS

Entering lines

A full line editing facility is available at all times when entering lines for BASIC whether in EDIT, INPUT or in the command level. This is achieved by using certain reserved characters for control purposes; these are detailed below:

CNT-H (or DEL)	steps the cursor back one position in the line.
CNT-I	steps the cursor forward one position in the line.
CNT-K	steps the cursor forward to the end of the line.
CNT-L	steps the cursor back to the start of the line.
CNT-J	opens a space at the cursor position and moves all characters up to make room.
= (top key right)	will delete the character before the cursor and close the line up.

Any other character is taken as a character to be placed in the line and steps the cursor on one position.

If any attempt is made to insert characters beyond the end of a line or backspace beyond the start of the line the commands are ignored.

To enter a line press the RETURN key, this will enter all characters to the left of the cursor,alternatively cnt-] can be used to enter the whole line regardless of the cursor position.

ESCAPE CODES

The ESC key is used when entering lines for entering a whole word with two keys; the ESC key is followed by another key which will signify the word to be entered. Most keys will produce words; these are listed below. CNT-C will cause a return to the monitor as usual. Note that control characters do not perform their

usual EDITING functions if preceded by ESC.

CNT A	PRINT	CNT B	AUTO	CNT D	CLEAR
CNT E	CLS	CNT F	CALL	CNT G	DIM
CNT H	DATA	CNT I	DUMP	CNT J	DEF
CNT K	EDIT	CNT L	ERASE	CNT M (RET)	END
CNT N	FOR	CNT O	GOTO	CNT P	GOSUB
CNT Q	GET	CNT R	HELP	CNT S	INPUT
CNT T	IF	CNT U	LPRINT	CNT V	LIST
CNT W	LINE	CNT X	LOAD	CNT Y	LET
CNT Z	NEXT	CNT [	OFF	CNT ESC	OUT
CNT]	ON	CNT ↑	POKE	CNT =	RUN
space	RETURN	!	RESTORE	"	REM
#	READ	$	RANDOMIZE	%	RESUME
&	REPLACE	'	RENUM	(	STOP
)	SWAP	*	SAVE	+	SCR
,	TRACE	-	VDU	.	WIDTH
/	NOT	Ø	THEN	1	ELSE
2	TO	3	STEP	4	USING
5	MOD	6	AND	7	OR
8	XOR	9	+	:	-
;	*	<	/	=	
>	<	?	=	=	@
A	FN	B	PI	C	TAB(
D	SPC(	E	BAS	F	ASC
G	ATN	H	COS	I	CHR
J	EXP	K	FRE	L	HEX
M	INP	N	INT	O	LEN
P	LOG	Q	PEEK	R	RND
S	SGN	T	SIN	U	SQR
V	STR	W	TAN	X	VAL
Y	LEFT	Z	RIGHT	[	MID
/	INSTR	]	STRING		

ABBREVIATIONS

In addition to the above escape codes, many of the words can be abbreviated by using the] key. The abbreviations are listed below (all words are listed in full during LIST).

]	PRINT	A]	AUTO	C]	CONT
CL]	CLEAR	CA]	CALL	D]	DIM
DA]	DATA	DU]	CUMP	E]	EDIT
ER]	ERASE	F]	FOR	G]	GOTO
GOS]	GOSUB	H]	HELP	I]	INPUT
L]	LPRINT	LI]	LIST	LO]	LOAD
M]	MID	N]	NEXT	O]	OFF
P]	POKE	PE]	PEEK	R]	RUN
RE]	RETURN	RES]	RESTORE	RA]	RANDOMIZE
S]	STOP	SW]	SWAP	SA]	SAVE
T]	TRACE	TH]	THEN	V]	VDU
		W]	WIDTH	X]	XOR

FORMATTING

BASIC level seven has a unique space-saving feature which permits faster execution and more compact storage stoether with a fixed format listing resulting in greater readability and freedom of formatting when entering lines. Each reserved word (those listed as words under ESC codes) is stored as one byte and spaces are not stored (except in quoted strings or in REM statements) and are automatically converted to full words on LIST and spaces are added according to certain rules - giving a listing which will correspond exactly to another, no matter whether the original was

typed with or without spaces.

Spaces are optional and may be placed anywhere in line with certain exceptions - these are that a reserved word may not contain spaces and if a combination of letters could be wrongly interpreted as another word: for example 6 IF F oR T THEN 43 should not be entered as: 6 IFFORTTHEN as this will be interpreted as IF FOR T THEN. This should therefore be entered as 6IFF ORTTHEN43 which will be interpreted correctly. Note that if a keyword is entered with an imbedded space it will not be interpreted and can cause weird syntax errors. Where the listing appears correct but yields a SYNTAX ERROR IN LINE 1Ø re-enter the line by using EDIT and exit from the line with CNT-]. If this does not work the error must exist in some other place.

BASIC BASIC

VARIABLES

BASIC uses variables as a means for conveying values in programs. A variable is a symbolic name given to a value; the value that the name represents may be changed at any time in the program and is stored in decimal at all times so resulting in greater flexibility over systems which have limited type variables. Variables must start with a letter and can contain any number of letters or numbers thereafter. Note however, that only the first two characters are unique to a variable and therefore CAT and CANDY are the same variable (CA). Variables may not contain reserved words; some examples of valid and invalid variables are given below:

Acceptable Variables	Unacceptable Variables
A	31 (begins with a digit)
C2	28 (numeric constant)
XX	ONE (contains keyword ON)
COPY	
UXZC	

Undeclared variables which are accessed are given the value Ø.

STRING VARIABLES

String variables are formed in the same way as numeric variables and are followed by a dollar ($). The same rules apply as for numerics; the variable A1 and 1$ are distinct variables and may be used together, if a string variable is undeclared, it takes the value of a null string (""). This assures that later changes or additions will not yield errors or unpredictable results.

ENTER

To enter a line in basic is a simple operation. Lines may be of two forms; the first is an immediate command and will be executed immediately the RETURN key is pressed to enter the line; this is also referred to as the DIRECT ENTRY mode. Try entering the immediate command PRINT "the dragon" followed by the RETURN key - Triton should then do as you say and print 'the dragon' on the next line. Below this it will print READY to tell you it has finished executing the program and awaits your next command. Certain commands may not be used as direct statements (DEF, INPUT and LINE input) and others are illogical (DIM, DATA etc..) but may be used. Any command may be put in a program (including RUN, EDIT and SCR)

The second form is as follows. To enter a line as part of a program type a line number before the commands; this will cause

the line to be stored in memory as part of the program and will not be executed immediately. TRY entering the following line:

```
1 GOTO 1
```

the computer will merely re-prompt you with > and wait for your next command. It has entered the line into memory and we can check this by executing a list command; type LIST and the line entered will be listed. To run the program enter the command RUN; the computer will then seem to do nothing. Don't worry; if you look at the program you will see why; we have put the computer in an infinite loop - it is 'chasing its' tail' so to speak. To stop the program we must take some action. Press the top right hand key on the keyboard. This is called the break key - the message BREAK IN LINE 1 will be printed to tell you that the program has been interrupted in line 1; we can continue the program where we left off by using the CONT command. This will again lock the computer in a loop so we must use the break key to stop it (note that to break from an INPUT statement use CNT-D). To get rid of the program type SCR; this will delete all lines from the program, you can verify by using the LIST command.

FEATURES

Level seven BASIC has many unique and useful features not found on other 8k basics. Some of the more unusual features are detailed hereunder.

HEX CONSTANTS

Level seven supports hex constants in a program and these are denoted by a preceeding dollar sign. These can be used as normal constants (H=$c9) as data (1Ø DATA $CD,$ØØ,$13) or as an entry for an INPUT command.

HEX STRINGS

Hex strings may be created for any integer from Ø to 65535. The HEX function will produce a string with the hex value of the argument.

These two features make base-conversion unnecessary in the monitor and indeed hex constants in the program make is much easier to read. Both features can of course be used in the immediate mode to give conversions.

DATA FILES

Data files are simply implemented in level seven by the unique feature of being able to save and load string and numeric arrays in the program. A very powerfull SAVE and LOAD structure permits great flexibility in data storage without restricting its power.

COMMANDS

All commands amy be used in a program (Including SCR) so, for instance a program may list part of itself and then execute that part. Certain commands do, however, not resume the program and exit to the command level. These commands are: SCR, STOP, END, EDIT and AUTO.

NOTE: The list of words given previously are not all implemented in release 1 of the BASIC; these commands will give a SYNTAX ERROR is used and no plans are made to implement any of these extra words. The words in question are:
HELP, RESUME, REPLACE, RENUM, USING and MOD.

ARITHMETIC OPERATIONS

Where a formula for a calculation contains more that one operation certain rules are used in deciding the order of operators execution. In any formula BASIC will perform operations in the following order:

1. Parentheses are the highest priority and any expression in parentheses will be executed before the rest of the line. This also applies to nested parentheses; the innermost expression will be evaluated first.
2. Exponentiation
3. Unary minus
4. Multiplication or division (of equal priority)
5. Addition or Subtraction
6. Logical operators in the order NOT, AND, OR or XOR (OR and XOR same priority)

If the rules do not clearly designate the order of evaluation, the expression will be evaluated from left to right. The expression X^Y^Z

1. A^B = step 1
2. (result step 1)^C=step 2

PARAMETERS

The commands LIST and SCR may be justified with parameters to restrict the range of their operation. If no parameters are specified the operation is performed on the whole of the program (i.e. the whole program is listed or deleted). If one operand is specified it is performed on that line, or the next line following it (unless it is beyond the end of the program), so for instance if a program contains lines 1Ø, 2Ø and 4Ø and the command SCR 3Ø is issued line 4Ø will be deleted (the same applies to list). If two operands are specified (separated by a dash (-)) the operation is performed for the first line (or the next one) through to the last line (or the one before it). A summary is given below:

1. LIST will list all lines
2. LIST 1Ø will list line 1Ø (or the next line)
3. LIST 1Ø-4Ø will list from line 1Ø (or next) to 4Ø (or preceding)
4. LIST -1Ø will list from the start to line 1Ø
5. LIST 1Ø- will list from line 1Ø to the end.

AUTO

Level seven has a useful and easy to use AUTO command for supplying line numbers when entering programs. Auto entered on its own will cause 1Ø to be printed out, and each time return is pressed it will step 1Ø. To change the start to 1ØØ enter AUTO 1ØØ. To change the step to 5 enter AUTO 1Ø,5. This will start at line ten and increment in steps of five.

If the line number exceeds 65529, Basic will complain with an OVERFLOW ERROR.

If the line number being printed already exists in memory an asterisk is printed before the line to warn the user. He may take one of the following steps if this happens:

1. Ignore it and enter the line. This will replace the line already in store
2. Leave the AUTO mode by entering CNT-D.
3. Delete the line by typing RETURN.
4. Leave the line in memory by issuing a CNT-L followed by a RETURN.

To quit from AUTO enter CNT-D.

PRINT

This is BASIC's main output command and is used for printing values strings etc..

LPRINT LPRINT follows the same rules as PRINT with the addition that the line will be sent to the printer as well as to the vdu. The printout always returns to the vdu only after an LPRINT. Wherever PRINT is used LPRINT may be substituted.

PRINT used alone (or followed by a colon) will result in a CR/LF being printed (Carriage Return & Line Feed). PRINT may be abbreviated to ? .

Characters enclosed in quotes are printed as entered. As in all quoted strings if the last character on the line is a quote (") it may be ommited to save space - eg: PRINT "HELLO THERE" and PRINT "HELLO THERE will both produce the correct message but the second example uses one less byte.

Two separators are used to separate items in a PRINT statement. These are a comma (,) and a semi-colon (;). The comma will move the cursor to the next tab position (every 8 characters). A semi-colon will print a single space unless it is the last character in the print statement in which case it will not move the cursor. If no separator is used no spaces will be output, so for instance an answer to a sum can be put next toa quoted string as in PRINT "THE ANSWER = "A+B which would produce something like THE ANSWER = 32 which is much more readable and uses less storage than the usual semi-colon separator.

Note for use with other BASIC's: Some other BASIC's use a semi-colon to separate items without printing a space. In this case the semi-colon may be ommitted so the string "HELLO ";A ;" HOW ARE YOU" may be entered as either "HELLO "A " HOW ARE YOU" or "HELLO":A :"HOW ARE YOU" both of which will produce the same printout and both of which save 2 bytes over the original.

If it is wished to print 2 (or more) strings combined without a space either PRINT A B or PRINT A +B may be used; the first one is preferable since it does not use extra space (if there is not enough room a NO STRING SPACE ERROR will be printed, but this cannot happen with the first example. Also the second example uses one extra byte).

Two functions are available in PRINT which produce formatting, the first, TAB will move the cursor to the column specified in the argument (providing is has not passed it already) and SPC will print the specified (upto 255) number of spaces.

Numbers are printed as detailed before with leading and trailing zeroes suppressed.

Scientific notation consists of the sign, mantissa,sign,exponant. EG 12345000 would be printed as 1.2345E+7.

INPUT

The command used to get replys from the user, used for both string and numeric data.

Multiple inputs are allowed and the elements (variables) to be entered are separated by commas.

e.g. INPUT A,B$,H

This input will cause the default prompt (?) to be printed the user could then enter the data separated by commas, as on the input statement (EG 12,FRED,33,763) and would assign 12 to a 'FRED' to B$ and 33.768 to H. If the user only entered one or two items (EG 12,HARRY), BASIC would re-prompt for the remaining items with a question mark. If too many items were entered the excess items are ignored without producing any form of error message. If the user wishes to supply his own prompt then this is enclosed in quotes before a semi-colon and the variable. This would then be printed WITHOUT the question mark and without any extra spacing.

E.G. INPUT "WHAT IS YOUR NAME ? ";NAME$

If a RETURN is typed without any other characters being entered for a string, a null string ("") is returned, if for a numeric value, the value Ø is returned. If a numeric value is required and string data is entered BASIC will re-prompt with the original prompt.

LINE INPUT

A special version of INPUT for strings may be supplied with a prompt in quotes and ONE STRING variable. It then prompts the user with the prompt supplied and will save all characters upto the RETURN in the string variable named. This includes leading spaces, quotes ("), colons (:) and commas (,) all of which are not used on a normal string INPUT. As before a single RETURN will produce a null string.

GET

This is a special, real time feature for getting information without waiting for a RETURN. This feature opens up many possibilities in the fields of interactive games and simulations. The GET command produces no prompt and does not echo any characters typed. When used it returns immediately, whether a key has been pressed or not. If a numeric variable is specified the GET will return 1 to 9 if the keys 1 to 9 are pressed; otherwise it will return Ø (as with INPUT). If a string variable is specified any key pressed will be returned as a single character string. If no key is pressed a nul string ("") is returned. Only one argument is allowed.

EG 1ØØ GET A$:IF A$ ="Y" THEN PRINT "YES" ELSE GOTO 1ØØ

will wait for Y to be pressed and print YES before continuing.

RUN

Used alone will clear all variables and arrays and execute the program from the lowest numbered statement. If a line number is specified then execution will commence at the line specified. (if the line specified does not exist the error message MISSING LINE ERROR would be displayed).

The use of RUN performs an implicit CLEAR.RUN may be used in a program.

CLEAR

Clear has two distinct forms,both of which clear all variables, strings and arrays. Used alone it will free all free memory for use. When used with a number it will reserve that number of bytes for exclusive use of strings. This defaults at power-on to 5Ø and can be altered by the user to any figure (so long as there is sufficient memory). If an attempt to use too much space is made, NO SPACE ERROR is printed. The number of free bytes for strings can be determined by using the FRE(A$)) function. If there is not sufficient space for strings (indicating a larger CLEAR number is required) the error message NO STRING SPACE will be printed.

SCR

Unique to LEVEL SEVEN BASIC this feature allows for deletion of blocks of lines as wellas the whole program. (For parameters see list parameter section). SCR will clear the whole program as normal. Note that line numbers quoted as parameters are INCLUSIVE.

LIST

Allows blocks of lines or the whole program to be listed in the standard BASIC format (for parameters see previous section).

READ

Used in conjunction with the DATA statement allows data to be placed within a program and obtained in the same manner as an input statement. If there is insufficient data in the program to satisfy a read statement a MISSING DATA error is generated, if a mismatch occurs between the READ and DATA statements (ie the variable is numeric and the data is string) the message SYNTAX ERROR IN 1ØØØ (or whatever line contains the data) is printed.

RESTORE

Allows the user to re-use data by resetting the point at which data is to be read in the program. Used alone it will reset the position to the first DATA statement (performed automatically by RUN). If it is followed by a line number, it will go to that line (providing it exists).

DATA

The complement to READ statement above, may contain mixed data in the same line. The string data must be enclosed in quotes if the string contains any spaces or commas or colons; otherwise they are optional. (Note that the trailing quote may be omitted if it is the last character on the line).

EG 233 DATA 12,hello,HELP," THIS IS A ,,STRING

DIM

Used to reserve space for matrices, which may contain up to 255 dimensions and each dimension up to 65535 elements (memory allowing). The elements must all be the same type. Arrays (matrices) are named in the same way as variables (see previous section). Several different arrays may be dimensioned at once by separating the separate arrays with commas. A dimension is not necessary if the individual dimensions are less than 1Ø. If an element is accessed from an array before it is dimensioned it will be dimensioned to 1Ø elements per dimension (actually 11 because of the zero subscript). It is always best to dimension the array first for readability and to save space. Also note the use of the zero element which is present in all arrays. If an

attempt is made to read an element not in an array (too big, not enough dimensions or too many dimensions) the error message BAD SUBSCRIPT ERROR will be printed. If an attempt is made to dimension an array twice or to dimension an array after using an element (in which case it will automatically dimension to 11 or 11x11) the error message REDIMENSION ERROR will be printed.

GOTO

Used to transfer execution to another line. The expression following the word GOTO must refer to a valid line otherwise an error will occur. Any numeric expression may be used.

EG GOTO 1ØØ GOTO 1ØØØ+A*2Ø

The GOTO command can also be used to run a program without resetting the variables and arrays.

GOSUB

Used to execute a subroutine. The argument form is as shown above for GOTO

RETURN

Used to end a subroutine; returns to the point after the corresponding GOSUB.

FOR

Used with NEXT for loop operations defaults to a step of 1 will always execute at least once.

EG FOR A=1 TO 1Ø

STEP

Used to alter the step after execution of a for next loop, if ommitted defaults to 1

EG FOR +01Ø TO ½ STEP -1

NEXT

Used to terminate a FOR loop may be followed by the variable of the FOR statement, defaults to the innermost loop (the last FOR statement). If no preceeding FOR is present the error MISSING FOR ERROR will be printed.

NESTING

Is limited only be memory size and is typically in the region of 300-400 levels.

EG NEXT NEXT COUNT NEXT I,J (special case for multiple for loops).

IF

This is the conditional test to determine the course of action to be taken by the program.

The THEN is optional as a qualifier but may be needed in certain circumstances.

```
EG  IF A=+1 THEN 30     IF A>=B PRINT "THEY ARE EQUAL"
    IF A<=S4 THEN A=4 :GOTO C   IF A<>L THEN PRINT "YOU LOSE
```

ELSE may also be used and will be executed if the comparison is false.

```
EG  IF A>N THEN PRINT "YES" ELSE PRINT "NO
    IF A<G-5/2 THEN 4Ø ELSE 8Ø
```

Note that if the qualifier (either THEN or ELSE) is used as a conditional GOTO, the number must be a constant. To use an

expression, GOTO should be used.

EG IF A=GG GOTO A*1Ø ELSE GOTO GG*1ØØ

Else may be nested.

EG IF A=2 GOTO 34 ELSE IF A=12 GOTO 47 ELSE PRINT "ERROR":END

If ELSE is not present execution will continue on the next line if the compairson is false. If ELSE is used it must appear on the same line as the IF. Execution will continue on the next line when an ELSE is encountered.

CLS

Clears the vdu, incorporates a delay. No arguments needed.

VDU

Used for sending ascii codes to, and memory mapping the vdu. Must contain 2 arguments, separated by commas, the first argument must lie in the range Ø to 1Ø24 and determines the position of the character to be mapped. If Ø is used the character is output in the normal way through the port. This is used for sending control characters which could not be put in a PRINT statement.

EG To send a line feed only to the terminal, use VDU Ø.$ØD.

The second argument is the character to be sent, or mapped. It must lie in the range Ø to 255. Characters greater than 127 are the same as Ø to 127. Eg 128 is the same as Ø.

EG FOR POS=1 TO $4ØØ:VDU POS,$2Ø:NEXT :REM SLOW CLEAR OF VDU

POKE

Used for changing memory; consists of two arguments separated by a comma. The first argument is the address and must be between Ø and 65535 ($ffff). The second argument is the value to be put at that address and must lie in the range Ø to 255 ($FF).

OUT

Used for outputting values to the 8080 ports; consists of two arguments separated by commas, both of which must be in the range Ø to 255 ($FF). The first argument gives the port number, the second the value to be sent to the port.

CALL

Used for interfacing user written machine code routines to the main BASIC program. Consists of 3 arguments separated by commas. The first argument must lie in the range Ø to 65535 ($FFFF) and is the address to be called. The second argument must lie in the same range, and is passed to the user program in the DE register pair. The third parameter must be a variable and is used to store the value of the DE pair on return. All registers may be destroyed (except the SP) and a return to BASIC is accomplished with a RET instruction at the end of the user program ($C9).

EG CALL $3ØØØ,A,A CALL C,V,I

END

Used to stop execution of the program and may be put anywhere in the program.

CONT

Used to continue execution of a program after it has been halted by a stop command, by typing CNT-D on an INPUT statement or by hitting BREAK during execution. May only be used if the program has not been changed. If the program has been changed, or an error has occurred, the error message CONT ERROR will be printed.

EDIT

Used to recall a line into the input buffer for editing in the normal way (see line editing). EDIT is followed by the number of the line to be edited. The line is then printed out as it would be in the LIST command. If the line is too long to fit in the buffer an OVERFLOW ERROR is given and the line remains intact and unchanged. The line is printed out and the cursor moved to the start of the line. Editing may be carried out in the normal way; the line number can be edited to facilitate copying lines without re-typing the whole line, or small changes can be made.

LET

Used to assign a value to a variable, (optional).

ON

Another form of conditional GOTO, amy be used with GOTO or GOSUB. Takes the form .. ON var GOTO (GOSUB) num,num,num... var is any variable and num is a line number. Var may be an expression. The expression is evaluated and a GOTO (or GOSUB) executed to the N'th line number.

EG The following code can be replaced by an ON command.

```
1ØØ IF A=1 THEN 1ØØØ
11Ø IF A=2 THEN 11ØØ
12Ø IF A=4 THEN 2ØØ6
13Ø IF A=5 THEN 2Ø88
14Ø IF A=6 THEN 3ØØØ
15Ø PRINT "ERROR ------"
```

Can be replaced by:

```
1ØØ ON A GOTO 1ØØØ,11ØØ,15Ø,2ØØ6,2Ø88,3ØØØ
15Ø PRINT "ERROR ------"
```

Note that line 15Ø was inserted if A was 3. If the number (A) is less than 1 or greater than the number of lines listed then execution continues after the last number (the next statement). With GOSUB the RETURN instruction will cause a return to the next statement after the ON GOSUB.

DEF

Def is sued to define the user function. The function has one local parameter (one that cannot be used by the rest of the program), and the statement takes the form
DEF FNR(X)=INT(X*RND(1))
FN is the common name for all user functions and is followed by a name (names for functions are made in the same way as variables) in this case R. X is the local parameter and is distinct from the variable X which may co-exist. Wherever X is used on the right hand side of the equals sign the parameter is substituted. This means that the variable X cannot be used and this should be borne in mind when writing programs. Any other variables may be used on the right hand side. The parameter need not be used at all (a dummy parameter). The user defined function is used as in any expression like:
A=FNR(4) which result in A taking a random value from Ø to 3.
The function must be declared before it is used. A DEF statement may not be used as an immediate command.

REM

Used for putting comments into programs, the rest of the line after a REM is ignored and stored as it is typed.

BASIC FILES

File handling in Level Seven BASIC is made easy by the use of SAVE and LOAD for arrays (see next section). These can be string or numeric and usually the average program can make do with one of each.

Given below is a short example of how to go about implementing data files:

```
1Ø CLS:PRINT "DEMO PROGRAM
2Ø INPUT "1=LOAD 2=SAVE 3=EDIT ";T:ON T GOTO 3Ø,1ØØ,2ØØ
3Ø INPUT "LOAD TAPE AND PRESS RETURN";A$:LOAD("FIL1")@D$,D
4Ø GOTO 2Ø
1ØØ INPUT "LOAD TAPE AND PRESS RETURN";A$:SAVE("FIL1")@D$,D
11Ø GOTO 2Ø
2ØØ INPUT "ITEM NO. ";T
21Ø PRINT "ITEM";T;D$(T)TAB(2Ø)"QTY="D(T);:INPUT"ENTER NEW QTY";D(T)
22Ø INPUT "ANOTHER ?";A$:IF LEFTS(A$,1)="Y"THEN2ØØ ELSE 2Ø
```

This would be the basis of a very simple stock-keeping program but it should give you the idea of how to write a practical program (you would need to add DIMs to the program as well as some means of entering the item names to the program above.)

SAVE

Used for saving user programs or data, it has many unique features. The tape format used is special and not compatible with other formats. This means that programs written for levels 4, 5 or 6 will have to be typed into level seven; machine code programs may still be loaded from the monitor in the normal way. O may not be used for saving a BASIC program.

SAVE may have a specified name or, if none is specified, the default header '$$$' is sent. The header is enclosed in round brackets and may be any string expression. If no extra parameters are specified the BASIC program in memory is saved.

EG SAVE will save the program with the header '$$$'
SAVE ("PROG *1") will save the program with the header 'PROG*1'
SAVE (NAME$) will save the program with the header NAME$.

If an attempt is made to specify a number a TYPE MISMATCH ERROR will be printed.

To save a block of memory (such as a maching code program) the addresses are placed after SAVE (the header if used)

EG SAVE 16ØØ, 18ØØ will save the block from 1600 to 1800 (hex).
SAVE ("TEST")A,B will save the block from A to B with the header 'TEST'

To save an array it must have been previously dimensioned and is saved in full. The array name is preceded by an '@' to specify an array.

EG SAVE @A will save the numeric array A with the header '$$$'.
SAVE (DAT$)@ST$ will save the string array ST$ with the header DAT$
SAVE ("ART"+A$)@A will save the numeric array with the header 'ARY'+A$

SAVE may be placed in a program or used as an immediate command. When it is executed it switches the motor on, waits 6 seconds and then sends the data. If the list of items to save is followed by a comma, the SAVE will be executed on the next item, with only 1 6 second gap.

EG SAVE @$,("AR")@D$ will have a six second gap, the header $$$ followed by numeric array $ followed by a 6 second gap followed by the header 'AR' and string array D.
SAVE, will save the program with the header '$$$' twice.
The tape relay is switched off five seconds after the last item in the list has been sent.

LOAD

Used to recover files saved by use of the save command. Note that it will not load files from TRAP, L6 or tiny BASIC L5 or L4.

The format for the load command is the same as for the save command with the exception that programs do not have an address following them, even if you are loading a machine code program. To load a program, only the header must match and the program will be loaded to the original address from which it was saved.

If the file loaded was a BASIC program it will run automatically from the lowest line, if not, execution will continue at the next statement.

LOAD used alone will search the tape for a program (BASIC or Machine code) and load the first program it encounters into memory. If a header is specified (as in SAVE) this header will be searched for, the message:
FILES FOUND:
will be printed and the computer will search the tape for a program. As each program is found the title will be printed out. If the computer is loading the file, the cursor will stay on the same line. If not, it will drop onto the next line and wait for another header.

If an array is specified then the computer will search for an array on the tape and load it. The array must be large enough to take all the elements of the array being loaded from tape. If not, the message OVERFLOW ERROR will be printed. Also, the type (string or numeric) of the array must match, otherwise the message TYPE MISMATCH ERROR will be printed. The no. of dimensions in the array does not matter, so long as the total number of elements is greater than or equal to the number of elements in the array saved.

When BASIC saves an array it saves the first dimension first, then the next element in the second dimension, etc. For example in an array set to DIM A(3,3) it saves in this order:

A(0,0) , A(1,0) , A(2,0) , A(3,0) , A(0,1) , A(1,1) , A(2,1) , A(3,1) , A(0,2) , A(1,2) , A(2,2) , A(3,2) , A(0,3) , A(2,3) , A(3,3) .

This could be saved from array A and reloaded into array X which had been set to DIM X(15) (although it could have been bigger) and A(0,0) =X(0) and so on. Any additional elements (not loaded from tape) are left unchanged, so for instance A could be loaded into an array with 3Ø elements and only the first 16 would be changed.

EG LOAD will search for a program, if it is a BASIC program it will RUN.
LOAD @A will search for the first array and load it.
LOAD ("FI*") will search for the program 'FI*'.

ERRORS

BASIC will check for any errors occuring during the loading of a file from tape. If any are detected, a question mark (?) is printed on the line below the header of the file being loaded to inform the user of the fact. Only one query will be printed

no matter how many errors are detected and BASIC will not abort from the load but wait until the data has been loaded and then print the message LOAD ERROR and return to the command level. It may be that the program has only missed one byte in which case it is only a question of using EDIT to correct the mistake. However, it may be that it has corrupted a pointer or line number, in which case it is best to exit to the monitor with CNT-C and re-enter BASIC with J and try again. (SCR may not work if the program pointers have been corrupted).

BREAKING

The SAVE command may be exited at any time by using the break key. Load may be exited whilst it is searching for a file or printing the header, but when a program is loading the RST-2 will have to be used (or RESET) since a break during a file load could corrupt memory.

BEEP BEEP

For users with the bleeper circuit fitted (details in documentation components from Transam) the bleeper will sound after a SAVE has finished and after each file has been loaded with LOAD.

DEBUGGING

Several commands are available on Level seven to aid in the fast development of programs and the efficient debugging thereof.

STOP

This will cause BASIC to print the message BREAK IN LINE x (where x is the line in which stop occurs) and to return to the command level. Execution may be continued from after the STOP by using CONT.

TRACE

This command causes BASIC to print all line numbers as they are executed. They are printed in square brackets (to distinguish them) as each line is encountered.

OFF

The command OFF will stop the action caused by TRACE. This is the condition at power-on and entry to BASIC by J.

DUMP

This is a very useful command. Used to find what's what in a program. It will print all variables used, together with their values (string or numeric). Also arrays are printed with their total size. Note that the dimensions appear in the opposite order to that used in a DIM statement (DIMA(2,3) will give A(3,2))

here is an example of a dump

```
A=3.14159
A$=HELLO
NA$=GEORGE P. INGLISH
C(5,2)
D$(5Ø,5Ø)
```

OTHER USEFUL FEATURES

SWAP

Used to swap two variables, strings, array elements etc. Both sides must be of the same type otherwise a TYPE MISMATCH ERROR is generated.

EG SWAP COUNT,C1 SWAP NAME$,TMP$ SWAP A(4,3),A(X,Y) etc.

ERASE

Used for removing an array from memory, so that it may be re-dimensioned or may give additional space for other operations. Generally used after a SAVE to save the contents of the array. More than one item may be specified, arrays being separated by commas.

EG ERASE R,Y$

RANDOMIZE

Used to set the random number generator at an unpredictable point. Note that the BASIC will always return the same number after entering, but from thereon the result is not predictable (see RND).

WIDTH

When BASIC sends its output to a printer the width of the page is assumed to be 64 characters (as on the vdu) but this can be altered by using the WIDTH command, the number following the command is used as the width (0-255)

EG WIDTH 8Ø will set up for an 80 column printer.

****

That concludes the description of the commands and functions available to the LEVEL SEVEN BASIC user. Other features of the language will now be dealt with, followed by a list of functions and finishing with some tips and hints.

MATHEMATICAL OPERATIONS

All arithmetic is carried out on floating point numbers and where possible they are represented in normal format. If they go out of the range of normal representation they are printed in normal scientific format (see previous section). Expressions may be as complex as necessary with the main limitation being the length of a single line. The number of parentheses is not limited, nor are other nested functions and recursive execution is permitted by the stack-orientated structure of the BASIC.

ARITHMETIC OPERATORS

The full range of mathematical operators are present (+,-,*,/) which may be used for numeric calculations (+ alone may be used for string concatenation). In addition exponentiation is supported (X to the Y is entered X^Y), in addition comparators and logical operators are allowed (see previous section for priority).

LOGICAL CAPTAIN

Logical operators are generally used on IF statements (IF A=B OR B+C=R THEN) but they operate in a logical fashion on signed 16 bit integers; comparators yield either -1 (TRUE) or Ø (FALSE); therefore NOT -1 gives Ø. AND, OR, NOT, and XOR all operate on a bitwise principle. Here are some examples:

63 AND 16 =16 63=111111, 16=10000 so giving 63
4 OR 2=6 4=100,2=10 ans =110 =6.
3 XOR 2=1 3=11 ,2=10 ans =001 =1.

-for a fuller explanation of logical operations see the book list in the monitor section.

EASY AS ...

PI is available as a predefined variable, and may not be redeclared. It has the value 3.1415927 and will yield correct results when used with the trigonometric functions.

FUNCTIONS

The functions available on the LEVEL SEVEN are listed below and standard trigonometric functions are handled in radians. The following routine can be used to convert from radians to degrees.

X/PI*18Ø will give x in degrees X*PI/18Ø will convert to radians.

OTHER FUNCTIONS

The following functions can be calculated by using the formulas given below:

FUNCTION	FUNCTION IN BASIC
SECANT	SEC(X) = 1/COS (X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(-X*X+1))
INVERSE COSINE	ARCCOS(X) = ATN(X/SQR(-X*X+1))+1.57Ø8
INVERSE SECANT	ARCSEC(X) = ATN(SQR(X*X-1))+(SGN(X)-1)*1.57Ø8
INVERSE COSECANT	ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1 *1.57Ø8
INVERSE COTANGENT	ARCCOT(X) = -ATN(X)+1.57Ø8
HYPERBOLIC SINE	SINH(X) = (EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X) = (EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X)+EXP(-X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2+1
INV.HYPERBOLIC SINE	ARGSINH(X) = LOG(X+SQR(X*X+1))
INV. HYPERBOLIC COSINE	ARGCOSH(X) = LOG(X+SQR(X*X-1))
INV. HYPERBOLIC TANGENT	ARGTANH(X) = LOG((1+X)/(1-X))/2
INV. HYPERBOLIC SECANT	ARGSECH(X) = LOG((SQR(-X*X+1)+1)/X)
INV. HYPERBOLIC COSEC.	ARGCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X
INV.HYPERBOLIC COTAN.	ARGCOTH(X) = LOG((X+1)/(X-1))/2

INTRINSIC BASIC FUNCTIONS

ABS(X) Gives the absolute value of X. (X may be a constant, variable or expression). Will return X if X =Ø, -X otherwise.

INT(X) Returns the largest integer less than or equal to X.
EG INT(.85)=Ø 1.9 = 1 -3.1 = -4 etc..

To round a number add .5
EG to round a number X to D decimal places use:
PRINT INT(X*1Ø^D+.5)/1Ø^D

RND(X) Will return a random number between Ø and 1 (sometimes Ø never 1), if X'Ø the last number is returned, otherwise a new number is returned.
(B-A)*RND(1)+A will generate a number from A to B.

SGN(X) Gives +1 if X>Ø, Ø if X=Ø, -1 if X<Ø.

SIN(X) Gives the sine of X in radians.

COS(X) Gives the cosine of X.

TAN(X) Gives the tangent of X.

ATN(X) Gives the arctangent of X.

EXP(X) Gives the constant E (2.71828) raised to the power X.

The largest value X may take is 87.3365.

LOG(X) Gives the natural (base E) logarithm of X.
LOG(X)/LOG(Y) gives the base Y logarithm of X.
EG common (base 10) log of 5 is LOG(5)/LOG(1Ø)

FRE(X) Gives the number of free (unused bytes) left for the program if the expression X is numeric or the number of free bytes for strings if the expression is string.
EG FRE(Ø) or FRE(A$)

PEEK(X) Returns the value held in the location X in memory.

INP(X) Returns the value currently at port X (X must be Ø-255)

STRING FUNCTIONS

ASC(X$) Returns a single ASCII character as its numeric value
EG if X$="?" then ASC($) would give 63. If the string is longer than one character the first character is used if the string is null ("") an error will occur.

CHR$(X) Returns a single character whose ASCII value is X (for ASCII codes see page 45 of the TRITON manual).

STRING$ (X,Y) Returns a string X characters long of character Y.

STR$(X) Gives a string equal to the value X as it would be printed by a PRINT statement
EG STR$ (2+3.5) would give "5.5"

VAL Used to convert a numeric string to a number.
EG VAL("5.5") gives 5.5
If the first character is not +,-,$ or a number the value Ø is returned.

LEN(X$) Returns the length of string X$. If X$ is null Ø is returned.

LEFT$ (X$,Y) Returns the Y leftmost characters in X$. If Y is greater than the number of characters in X$ (LEN(X$)) X$ is returned.

RIGHT$ (X$,Y) Returns the Y rightmost characters in X$. If Y >=LEN(X$) X$ is returned.

MID$ (X$,Y) MID$ with 2 arguments returns a string from character Y to the end of X$. If Y>LEN(X$) a null string is returned.

MID$ (X$,Y,Z) Returns a string from character Y in X$ for Z characters. If Y> LEN(X$) a null string is returned.

INSTR(X$, Y$,Z) Returns the position of the first occurance of Y$ within X$ starting at position Z in X$ (Z is optional) If Y$ is not in X$ or either of the strings are null a value of Ø is returned.
EG IF INSTR(ANS$,"YES")THEN PRINT "I AGREE"

ERROR MESSAGES

The following is a list of error messages with possible causes and fixes.

Error messages are in two forms, the first which occurs during an immediate command takes the form
XXXX ERROR
Where XXXX is the error message, the second form occurs during execution and takes the form
XXXX ERROR IN LINE YYY
Where YYY is the line number of the line in which the error occured.

SYNTAX ERROR

This is the most common error and indicates a mistake such as missing bracket or an incorrect character; once found is easy to fix.

BAD SUBSCRIPT ERROR

Occurs when referencing an element not in that particular array; may also be due to too many or too few dimensions being specified. EG an attempt to reference A(1,1,1) when A was dimensioned as A(2,2).

REDIMENSION ERROR

An attempt was made to DIMension an array that had already been dimensioned. Can also occur if an array was not dimensioned before it is referenced (see section of DIM statement). EG A(3)=2 is encountered before DIM A(1ØØ).

BAD ARGUMENT ERROR

Another common error, due to the argument for a function or statement being out of range. Some common causes are:

a) a negative argument for a subscript E.G. LET C(-1)=3
b) an unreasonably large subscript (>32767)
c) LOG negative or zero argument.
d) SQR negative argument
e) a reference to CALL,INSTR,MID$,LEFT$,RIGHT$,STRING$,INP,OUT, PEEK,POKE,TAB,SPC,VDU,ON..GOTO or ON..GOSUB,with an improper argument.

DIRECT ERROR

Occurs if you try to use DEF or INPUT or LINE INPUT as a direct command. Whilst these cannot be used, other commands are pointless when used as direct statements (REM,DATA,etc..).

MISSING FOR ERROR

A NEXT statement was encountered without a corresponding FOR.

MISSING GOSUB ERROR

A RETURN statement was encountered before a GOSUB.

MISSING LINE ERROR

An attempt was made to reference a line which does not exist in memory. Can occur on the following statements. GOTO,GOSUB,RUN,RESTORE,THEN,ELSE or EDIT.

OVERFLOW ERROR

The overflow error has a general meaning and two specific meanings. The specifics will be dealt with first.

A) The line specified in EDIT was too long to fit in the buffer.
B) The line number being calculated by AUTO got too large.
C) The result of a calculation was too large to be stored or printed. If a number underflows the result Ø is given with no error.

NO SPACE ERROR

Program too large, too many GOSUBs, too many FORs too many levels of parentheses or too complex an expression or any combination of the above, the amount of space can be increased by decreasing the size of the CLEAR command (if used).

NO STRING SPACE ERROR

The space left for strings was less than the length of the string being stored; more space can be reserved by using a CLEAR statement or one with a larger argument (may conflict with above).

MISSING DATA ERROR

A read was encountered without any corresponding DATA statements or all the data had been used. A RESTORE statement can be used to re-read the data or extra DATA statements added.

MISSING FUNCTION ERROR

An attempt was made to use a user function which had not been defined.

DIVIDE ERROR

An attempt was made to divide by Ø (can occur if the TAN(PI/2) is attempted.)

CONT ERROR

An attempt was made to execute a CONT command after an error had occurred or the program was modified.

STRING OVERFLOW ERROR

The length of a string exceeded 255 characters.

STRING ARGUMENT ERROR

A string argument was too complex, split the expression into two or more shorter parts.

TYPE MISMATCH ERROR

The right and left hand parts of an expression must be of the same type (string or numeric) or an attempt was made to supply a function with the wrong type of data.

LOAD ERROR

The tape load program detected a read error, the program was still loaded and may be corrected, or an attempt made to re-load.

SAVING SPACE

1) Use multiple statement lines, each line has a fixed overhead of 5 bytes, therefore each line combined is 5 bytes saved.

2) Delete all REM statements, each character in a rem statement uses one byte (including spaces).
   EG 13Ø REM THIS IS A COMMENT uses 23 bytes.
      21Ø PRINT RES :REM PRINT SUM uses 11 bytes including the colon.

3) Use variables instead of constants C uses one byte 5.3246 uses 6. This also improves the speed of a program.

4) The END statement is optional and can be ommitted to save space.

5) Reuse variables. A variable once declared uses space and if a temporary variable is used in one part of a program use it again in another. Also use one temporary string variable for user replies.

6) Use GOSUBs to execute common portions of code.

7) Use User FUNCTIONS for common expressions.

8) USE the zero elements in arrays. EG A(Ø) B(Ø.Ø) etc..

STORAGE ALOCATION

Simple (non array) variables use 6 bytes; a string variable uses 6 bytes of main store plus one byte per character in the actual string in the string store.

Matrices use a minimum of 12 bytes with elements using 4 bytes each plus 2 bytes for every dimension.

A user function uses 6 bytes.

Each keyword (PRINT,SAVE,LIST,TAN etc) uses one byte; other characters use 1 byte each (spaces are not stored except in REMs).

Each active FOR..NEXT loop uses 22 bytes
Each active GOSUB (one that hasn't returned) uses 6 bytes.
Each parentheses encountered uses 4 bytes and each temporary result calculated uses 12 bytes.

SPEED HINTS

The hints given below will increase the execution speed and in many cases reduce storage requirements as well.

1) Delete all REM statements; these slow BASIC down by having to ignore them.

2) USE VARIABLES INSTEAD OF CONSTANTS - THIS WILL INCREASE SPEED BY A LARGE AMOUNT. It is much faster to recall the value of a variable then work out the value of a stored number in ascii.

3) Variables used often should be declared at the start of the program. The first variable declared will be the quickest to find.

4) Use NEXT without an index variable, BASIC does not need to check for the correct index.