NewBasic

expands TRS-80 Basic

CIE People's Software
Box 159, San Luis Rey CA 92068

NEWBASIC

A set of enhancements for Disk Basic with added utilities for the TRS-80 Model I 32K or 48K system with one or more disk drives running under TRSDOS 2.3, NEWDOS 2.1 or NEWDOS/80.

NewBasic expands TRS-80 Basic

just $29.95 disk version on tape
add $5.95 to have it on disk

By CLARENCE FELONG, KEN BROWN and GARY SHUTE
Modular Software Associates

NewBasic, written by our company, is a set of enhancements to TRS-80 Model 1 Basic, plus added utilities.

Two versions are available from CIE:

- The disk Basic version will run on 32 or 48K RAM systems operating under TRSDOS 2.3, NewDOS 2.1 or NewDOS-80. This version may be obtained for the special introductory price of $29.95—($31.75 CA).
- Level II NewBasic does not include disk utilities and may be purchased for $19.95.

NewBasic contains more than 30 new commands including a spooler, despooler, DOS commands while in Basic, a powerful new trace facility, "single-key" key-word entries, fast new graphics commands, and even a sound output capability.

NewBasic is unique in its linking-loader concept. A user may create many different "custom" versions which may include a few or all of the new commands (disk Basic only). This allows the efficient use of RAM for running very large programs.

Level II NewBasic includes all the commands of the disk version except those which are useful only under a disk operating system. This version fits in about 5K of RAM.

NewBasic is transparent to the user. That is, it exits in low RAM and does not interfere with normal machine-language routines located in high memory.

DISK BASIC

Some of NewBasic's commands are just for disk spinners. These include SPOOLON, DESPOOL, DIR, DLOAD, DRUN, and EXIT.

SPOOLON directs printer output to a user-specified disk file. SPOOLON used in conjunction with DESPOOL allows for more efficient hardcopy output.

Printouts that normally would require a large amount of time to output may be very quickly spooled. Later, the created file may be despooled while other computer operations are being performed. DESPOOL will despool a file to either the RS-232-C interface or parallel (Centronics) port.

Despooling is interrupt driven; normal processing can continue while a file is being printed. Spooling and despooling may be carried on simultaneously.

The DIR command allows the viewing of a diskette's directory. The directory is given a more complete format than TRS-80 users are accustomed. A user can see at a glance how much free space remains on the diskette and the size, in sectors, of every file.

DLOAD and DRUN allow for the loading and executing of object-code files (-CMD).

EXIT enables a return to the DOS READY state.

BACKGROUND UTILITIES

Many complaints aired about the Model I keyboard and display have been answered by NewBasic.

The hard-to-see underline cursor may be replaced with a blinking block cursor—simply use the BLINK command.

Use of the REPEAT command allows auto-key repeat after a key has been depressed about half a second. An audible "beep" may be heard for keystrokes if an external speaker-amplifier is hooked to the cassette-out line (or, try just plugging it into the cassette recorder and depressing Play. Works on some machines—Ed). A keyboard debounce routine is also enabled by the REPEAT command.

The inputting of Basic keywords has been made easy by the use of "single-key" entries. The QUICKKEY command enables the input of whole words by pressing a single key (example: G equals GOSUB). Included in the documentation is a chart which can be attached to the video display for abbreviation look-up. This allows much faster look-up than key stick-on type aids. The touch typist will no longer have to stop and examine the keyboard to find an abbreviation.

GRAPHICS ADDITIONS

Several useful graphics abilities are implemented under NewBasic.

The user may draw a line between two points by use of the LINE command.

The statement LINE 0,0,127,47 draws a line from the upper left of the video display to the lower right. Similarly, the RECT command draws the rectangle perimeter determined by the imaginary diagonal line drawn between the two given points.

The FILL command draws a rectangle and then "fills it in" (i.e. turns the interior graphics blocks on). WTS whites the screen "instantly".

NEW TRACE, PRINTER FEATURES

NTRON is a new trace facility which makes program debugging much easier.

The user may specify which lines of a program are to be traced and the entire line, as it is executed, will be displayed. The value of variables or expressions may also be displayed—during program execution! This command may also be used inside a Basic program for even greater flexibility.

The use of the PON command causes video output to be echoed to the printer. The use of PON and SPOOLON enables the user to keep track of programs run, or for debugging purposes.

SOUND, LOCATOR

Also included in NewBasic is the SOUND command. This allows the output of a specific tone for a requested duration to the cassette port. A handy table of frequencies and durations versus arguments is included in the documentation.

Ever have problems finding a variable or string in a large program? The LOC. command allows the user to find any or all uses of a variable, Basic key word, or string.

Depressing one key allows the user to edit the line containing the "found" item. This enables, for example, the locating of all the instances where the keyword PRINT is used so that the editing of the found word may occur.

NEW GOTO, GOSUB

Two new commands that allow the transferring of program execution are included in NewBasic. NGOSUB and NGOTO may be used to GOTO or GOSUB to a line number, an expression, or even a label. The statement X =10 : NGOTO X causes program execution to goto line 10.

You may also use a statement such as NGOSUB x "ALABEL" to call a subroutine denoted by the label, x "ALABEL". Also, ON ERROR GOTO has been modified to accept expressions as arguments.

STRUCTURED CONSTRUCTS

Structured programming is now possible with a DO-UNTIL construct. DO loops may be nested 10 deep enabling easy development and debugging of programs.

DO-UNTIL coupled with line labels and NGOSUB make the GOTO command unnecessary. GOTO is considered to be a definite no-no by many structure-oriented programmers and when used in complex programs makes debugging a chore.

RS-232-C INPUT-OUTPUT

The RS-232-C interface may now be initialized under Basic by the RS232 command. Rates from 110 to 300 baud with or without parity enabled and different word lengths may be implemented. The RSIN command allows input from either the keyboard or the serial port. The RSOUT command directs all printer output to the serial port.

It is a trivial matter to implement an automatic logon to a time-share system such as The Source or Compuserve and run a terminal program all under control of a Basic program!

This two-line program allows the TRS-80 to be a "smart terminal":

```
10 A$=INKEY$:
   IF A$="" THEN 10 ELSE PRINT A$;
   IF A$<>":" THEN 10
20 A$=INKEY$ :
   IF A$="" THEN 20 ELSE LPRINT A$;
   IF A$=CHR$(13) THEN 10 ELSE 20
```

(Continued on back page)

TABLE OF CONTENTS

NEWBASIC CREATOR

This program produces a NEWBASIC program on disk, composed of user-selected commands. Once a NEWBASIC program has been created, it can be executed as described in EXECUTING NEWBASIC. You need run CREATOR only when you desire a different version of NEWBASIC.

When you execute CREATOR, you will be asked what commands you wish to include in the NEWBASIC program you are creating. You will be shown every keyword in alphabetical order. A few keywords, such as DO (UNTIL) and PON (POFF), are shown in pairs (the second keyword will be displayed in parentheses). After deciding what keywords to include, the NEWBASIC program you have just created will automatically be DUMPed to disk. You are now ready to execute NEWBASIC.

To execute the NEWBASIC CREATOR, type "CREATOR" when you see "DOS READY". (CREATOR occupies memory from 7900H to BFFFH, and therefore will overwrite any custom driver routines, etc, that occupy this region.) The screen will clear, and the following will appear:

```
MODULAR SOFTWARE ASSOCIATES - NEWBASIC CREATOR - VERSION 1.0

KEYWORD                    DESCRIPTION                    SIZE    USE?
BLINK       SWITCH BLINKING CURSOR ON/OFF                 161     ?
```

Notice the information that is displayed for BLINK--this is the same type of data that will appear for every NEWBASIC keyword. First, there is the keyword BLINK, followed by a short description of what BLINK does. Next is the number of bytes (in decimal) that would be added to NEWBASIC if you decide to include BLINK. The size for a particular keyword can vary greatly from one use of CREATOR to another, depending on what other keywords have (or have not) been included at the point when the size is displayed. As an example, the size of EXIT is 6 bytes if DRUN is included, but 103 if it is not included. The size is displayed to give an approximate idea of the size of code associated with a keyword. It is not intended to be an aid in achieving optimal memory usage, since the figure shown can be misleading.

You must now decide whether or not to include BLINK in the NEWBASIC program you are creating. If you wish to include it, you must press the "Y" key; to exclude it press the "N" key. The same type of information for the next keyword will now be shown, and you must decide whether or not to use it. When you have answered "Y" or "N" for every NEWBASIC keyword, the following will appear:

```
TOTAL SIZE OF COMMANDS INCLUDED:  XXXX

FILESPEC TO SAVE TO (PRESS ENTER FOR "NEWBASIC/CMD" DEFAULT)?
```

The total size given is the total byte count which will be taken up by the NEWBASIC program being created <u>once it is executed</u>. All NEWBASIC programs take up the same amount of space on disk,

regardless of their size when they are run. Hence, a NEWBASIC program that has no NEWBASIC commands included will be the same size on disk as one in which every NEWBASIC command has been included.

The NEWBASIC program just created will now be saved to disk. If you press <ENTER> in response to the "FILESPEC ..." question, it will be saved under the name NEWBASIC/CMD. If you desire to have it saved under a different name, you may type in a standard DOS filespec and press <ENTER>. (The filespec may not include a password, however.) An extension of /CIM will automatically be appended if your filespec does not include an extension. A delay will occur in which the relocatable NEWBASIC program is created. An asterisk in the upper right hand corner will blink during the delay to assure you that CREATOR is still functioning properly. After the delay, the disk drive should turn on and the NEWBASIC program will be DUMPed to disk.

IMPORTANT NOTE: The NEWBASIC program to be DUMPed will take up 8 grans on the disk. If you are using TRSDOS or NEWDOS 2.1, the drive motor will remain on if there is not enough room on the disk for the file to be DUMPed. If the DUMP takes longer than 30 seconds, then this has occurred. You must reset the computer and run CREATOR again, after insuring enough file space exists.

EXECUTING NEWBASIC

Once you have run the NEWBASIC CREATOR program and created a NEWBASIC program on disk, it is very simple to execute NEWBASIC and enter your NEW, enhanced, BASIC. The normal "BASIC/CMD" file must exist for the created NEWBASIC program to execute. You execute NEWBASIC just as you would execute whatever BASIC you are using, except that you substitute "NEWBASIC" (if the created NEWBASIC program is called "NEWBASIC/CMD") for "BASIC". If you are using NEWDOS BASIC, you may type the number of files desired, memory size, etc, all on the same line as you normally would to enter BASIC. If you are using TRSDOS, then you should only type "NEWBASIC" and press <ENTER>. Answer the "HOW MANY FILES?" and "MEMORY SIZE?" questions as usual. For both TRSDOS and NEWDOS there will be a short delay to allow you to read the copyright notices of the respective BASICs. The screen will then clear, MODULAR SOFTWARE ASSOCIATES's copyright notice will appear, and below that a number will be displayed ("USER MEMORY START ADDRESS = XXXXX"). This number gives the start address in decimal of your BASIC program text area. You are now ready to use your NEW enhanced BASIC.

IMPORTANT NOTE: DEBUG must be off before executing NEWBASIC, or the computer will "hang". Once you have entered NEWBASIC, you may re-enable DEBUG by CMD"D" if desired.

EXAMPLES:

NEWBASIC
will execute NEWBASIC under all compatible DOS's.

NEWBASIC 15,50000
will execute NEWBASIC and enter BASIC with 15 files and memory size set to 50000. (NEWDOS only!)

MISCELLANEOUS NOTES

NEWBASIC occupies RAM below normal BASIC. This moves BASIC programs higher in RAM than usual. Some programs which use PEEK and POKE do not allow for this fact, and may not run properly under NEWBASIC.

All NEWBASIC commands must be at the start of a BASIC statement. This means that NEWBASIC commands included in IF/THEN/ELSE statements must be preceeded by a colon to indicate the start of a new statement.

In the syntax descriptions that follow, optional arguments are surrounded by braces "{ }".

BLINK

The BLINK command toggles the blinking block cursor on/off. The first time the command is executed, the blinking cursor will appear. The next time the command is executed, the blinking cursor will be replaced with the standard underline character. If BLINK is used again, after the underline cursor has been restored, the blinking block cursor will reappear (and so on...).

EXAMPLES:

```
10 BLINK
```
if this is the first time that BLINK is used, the blinking block cursor will be turned on.

```
10 BLINK
20 INPUT"NAME",A$
30 BLINK
40 PRINT "HELLO, ";A$
```
if the underline cursor is being used prior to running this program, the following will occur: The question "NAME?" will be displayed with a blinking block cursor after it. If the reply to the input statement was "JOHN", the line "HELLO, JOHN" will appear. Note the blinking cursor will have been replaced with the standard cursor.

CALL hex-constant
CALL int-expression
CALL lsb,msb expression
CALL pos-expression

This command allows you to easily call a machine language subroutine. No DEFUSR or USR(X) statement is needed. The subroutine must be in memory at the time of the CALL. You can not directly pass an argument between the BASIC program and the machine language subroutine being called. If you wish to pass one or more arguments, this can be done by POKEing and PEEKing into specified memory locations. The subroutine being called should return to BASIC by the Z-80 RET statement. The CALL statement saves all registers used by BASIC.

EXAMPLES:

```
CALL &HE000
```
will call a subroutine located at hex E000.

```
20 CALL +50000
```
will call a subroutine located at decimal 50000.

```
30 CALL X + PEEK(35000)
```
will call a subroutine located at the sum of the current value of X and the contents of memory location 35000.

```
CALL 0,200
```
will call a subroutine at 0,200 in lsb,msb (decimal) notation.

```
200 CALL -100
```
will call a subroutine located at -100 (65436 decimal).

```
10 X=-100 : CALL X
```
will call a subroutine located at -100 (65436 decimal).

```
1000 X=65436 : CALL X
```
The intent of this program line is the same as in the preceeding examples, but will result in an OVERFLOW error. This is because "X" will be interpreted as an "int-expression". To force "pos-expression" interpretation, the correct usage is CALL +X.

CONV hex-constant
CONV int-expression
CONV lsb,msb expression
CONV pos-expression
CONV "A"
CONV "AA"

This command is used to display different representations of the same number. CONV "A" means convert the single alphanumeric character A. CONV "AA" means convert the 2-character string AA, where AA can be any 2 alphanumeric characters (not necessarily the same).

The following illustrates the display format of CONV:

```
CONV +50000                                          (user entry)
&HC350    " P"   -15536    +50000    80,195        (computer reply)
```

The first number displayed is the hexadecimal (base 16) representation. Next follows the 2 character alphanumeric or graphic representation. The first character displayed corresponds to the most significant byte (msb) of the hexadecimal representation, while the second character corresponds to the least significant byte (lsb). The byte will be displayed as a blank if it is less than 0021H (33 decimal), or greater than 00BFH (191 decimal). An exception is that a zero byte will be considered as an ASCII NUL (""), and will not be displayed. The third representation given is the decimal integer form of the number. This is the number you would use to PEEK or POKE an address above 32767. The fourth number displayed is the positive decimal representation. The fifth (last) representation is the number in decimal lsb,msb format. This number is very useful when PEEKing or POKEing addresses in decimal.

EXAMPLES:

```
CONV "bD"
```
will display the representations of the letters "bD".

```
CONV &HBD
```
will display the representations of 00BDH.

```
10 X=100 : Y=0 : CONV X,Y
```
will display the representations of the lsb,msb number 100,0.

DESPOOL "filespec" {,S}

The DESPOOL command without the trailing ",S" will despool the specified file to a parallel printer. The command with the trailing ",S" will despool the specified file to the RS-232-C serial port. The spooler is interrupt driven; normal processing can continue while a file is being printed. An occasional short pause will occur as the disk file is being accessed for more information. This command allows the operator to compose, edit, or run BASIC programs which may include disk I/O, while a previously spooled file is being printed. The despooling is automatically terminated upon reaching the end-of-file. To abort the despooling operation, hold down the shift key, the down arrow key, and the "C" key simultaneously (this outputs a 'control C').

Only ASCII encoded files may be despooled. BASIC programs saved without the ",A" option may not be despooled. An attempt to despool two files at the same time will result in an ILLEGAL FUNCTION CALL error. CMD"T", which halts interrupts, will cause despooling to pause. CMD"R", which enables interrupts, will cause the despooling to continue.

EXAMPLES:

```
10 DESPOOL"TEXT/FIL"
```
will start the despooling of the file "TEXT/FIL" to the parallel port printer.

```
20 DESPOOL"CODEFILE/TXT.SECRET:1",S
```
will, upon execution, start despooling the file "CODEFILE/TXT" on drive 1 with the password "SECRET" to the serial port.

DIR {:} {drive no.}

The DIR command allows the display of a diskette's directory. As in DOS, if no drive number is specified, the directory of drive Ø will be displayed. If a drive number is specified, it can be immediately after the command (DIR1) or seperated from the DIR by a space (DIR 2). The "normal" colon may also be used (DIR :1). The drive number must be used in a BASIC program.

The display will be formatted with diskette data on the top line of the display, and file data on the lines below. If there are more files than may be displayed at one time, as many as possible will be displayed and pressing any key will display the remaining files. The diskette data will contain the drive number being read, the date the diskette was formatted, and the number of free sectors (grans * 5) remaining on the diskette. The data displayed for a file includes the filename, including extension if any, followed by the size of the file in sectors. System files will not be displayed.

The DIR command will not execute in NEWDOS/8Ø (nor is it needed). If the command is used, a MISSING NEWBASIC COMMAND error will result.

EXAMPLES:

DIR
when this command is given, the directory of drive Ø will be displayed.

DIR 1
will result in the display of drive 1's directory.

1Ø DIR :2 : PRINT"DONE"
will, when run, display the directory of drive 2, if it exists, and then display the line "DONE". If a DIR of a drive number that does not exist is executed the drive motors will remain on and the system will "hang".

DLOAD "filespec"

The DLOAD command is used to load object-code files in RAM and returns control to BASIC. Command (/CMD) files may be loaded and, providing the entry point is known, executed from within BASIC.

EXAMPLES:

```
10 DLOAD"PRINTER/CMD"
```
will load the command file "PRINTER/CMD" into RAM.

```
10 DLOAD "SORT/CIM"
20 CALL &HF000
30 PRINT"SORT DONE"
```
will load the object-code file "SORT/CIM" and execute it (if the start address is hex F000). The line "SORT DONE" will then be displayed.

DO {statement} {statement} ... {statement}
UNTIL (true/false expression)

The DO and UNTIL commands form an iterative (repetitive) loop. The DO command denotes the start of the loop, while the UNTIL command terminates the loop. The body of the loop consists of Ø or more statements between DO and UNTIL. These statements will be executed repeatedly until the (true/false expression) argument following UNTIL is true. (The parentheses surrounding the "true/false expression" after UNTIL are mandatory.) Nesting of DO/UNTIL loops past a level of 1Ø will cause an OVERFLOW error.

If an UNTIL is executed without previously executing a matching DO, an UNTIL WITHOUT DO error will occur. Mismatched DOs and UNTILs can result from transferring (GOTO) out of the DO/UNTIL loop body. To immediately terminate a DO/UNTIL loop, use the following procedure--<u>never</u> transfer (GOTO) out of the loop.

```
1Ø I=Ø
2Ø DO
3Ø   GOSUB 1ØØØ
.
.
1ØØ  IF T=2 THEN I=1Ø : GOTO 2ØØ
.
.
19Ø  I=I+1
2ØØ UNTIL (I=1Ø)
```

In line 1ØØ, the loop terminating condition (I=1Ø) is set to true. Control is then transferred to the line containing UNTIL, where the loop will be properly terminated.

A DO/UNTIL loop should not be imbedded within IF/THEN/ELSE statements, unless the entire loop is contained within the same program line as the IF. The following two examples illustrate this restriction. The first example shows an incorrect use of a DO/UNTIL within an IF/THEN/ELSE. The second example shows how the first example can be written correctly.

```
1Ø IF I=1 THEN : DO
2Ø   PRINT"I=1"
3Ø   INPUT I
4Ø UNTIL (I<>1)
```

The fault in the above program is that line 2Ø will be executed even if I does <u>not</u> equal 1. This is due to the fact that the "scope" of the IF statement only applies to the line in which it appears.

```
1Ø IF I=1 THEN : DO : PRINT "I=1" : INPUT I : UNTIL (I<>1)
```

This program line will execute the DO/UNTIL loop only if I=1. There are, of course, several other methods of coping with the IF/THEN/ELSE restriction, two of which are shown below.

```
10 IF I=1 THEN : NGOSUB #"DO I LOOP"
.
.
999 END
1000 #"DO I LOOP"
1010 DO
1020   PRINT "I=1"
1030   INPUT I
1040 UNTIL (I<>1)
1050 RETURN

10 IF I<>1 THEN : NGOTO #"DONE LOOP"
20 DO
30   PRINT "I=1"
40   INPUT I
50 UNTIL (I<>1)
60 #"DONE LOOP" : REM REST OF PROGRAM FOLLOWS
```

The DO/UNTIL loop construct executes slower than a FOR/NEXT loop, since it is implemented as an add-on to an existing interpreter (LEVEL II BASIC). However, it can provide structure and clarity to a BASIC program, which are often more desireable than speed of execution.

EXAMPLES:

```
10 A$="" : DO : INPUT"WHAT IS YOUR NAME";A$ : UNTIL (A$<>"")
```

will continue to INPUT A$ until something other than <ENTER> is INPUT.

```
10 TRUE=0
20 DO : CLS
30   GOSUB 200
.
.
100 UNTIL (TRUE OR A=B)
```

will execute the DO/UNTIL loop until the variable TR equals -1 or until A=B.

```
10 DO
20   NGOSUB #"INIT GAME"
30   DO : NGOSUB #"PLAY ROUND" : UNTIL (GAMEOVER)
40   NGOSUB #"WINNER"
50 UNTIL (0)
```

is an example of nested DO/UNTIL loops. The DO/UNTIL loop in line 30 is nested within the loop of lines 10-50. This is a nesting level of 2. (Notice that if the subroutine #"PLAY ROUND" has a DO/UNTIL loop, it too would be nested within the loop of lines 10-50.) The DO/UNTIL loop in line 30 will terminate when the variable GA equals -1. The loop of lines 10-50 will "never" terminate, since 0 can never be true (that is, can never equal -1).

DRUN "filespec"

The DRUN command exits NEWBASIC and executes the requested object file. This is a final exit. A command file may not be run with a return to NEWBASIC. If an error occurs during the execution of DRUN a return to DOS READY will result. The complete file name must be used including extension as no defaults are implemented.

EXAMPLES:

DRUN"DIRCHECK/CMD"
will exit NEWBASIC, run the program "DIRCHECK/CMD" and exit to DOS.

10 DRUN "BASIC/CMD"
will exit NEWBASIC and load and execute "BASIC/CMD". This will allow the user to enter "normal" BASIC without exiting to DOS.

EXIT

This command exits NEWBASIC and returns to DOS READY. The EXIT and DRUN commands are the only safe way to exit NEWBASIC other than a "reset".

IMPORTANT: Once NEWBASIC has been exited, there is no way to re-enter NEWBASIC to recover a program. Neither "BASIC *" nor "NEWBASIC *" will succeed in doing so.

EXAMPLES:

```
EXIT
   will exit NEWBASIC and return to DOS.

500 IF A$="STOP" THEN : EXIT
   will exit NEWBASIC if the variable A$ equals "STOP".
```

FILL X1,Y1,X2,Y2 {,R}

The FILL command without the trailing ",R" turns on all the graphics blocks of the rectangle determined by the diagonal line drawn from the point X1,Y1 to the point X2,Y2. The FILL command with the trailing ",R" resets (i.e. erases) the graphics blocks of the determined rectangle. The X-coordinates are numbered from left to right, 0 to 127. The Y-coordinates are numbered from the top to the bottom, 0 to 47.

The arguments X1,Y1,X2,Y2 may be numeric constants, variables, or expressions. FILL uses the integer portion, hence, the arguments need not be integers. The command is valid for X's between 0 and 127 inclusive and Y's between 0 and 47 inclusive. Arguments outside this range will produce an "ILLEGAL FUNCTION CALL" error message.

NOTE: Parenthesis before and after the arguments field (unlike SET and RESET) are not needed and if present will cause an error.

EXAMPLES:

```
10 FILL 0,0,127,47
```
will turn all graphics blocks on (i.e. white the screen).

```
20 FILL 127,47,0,0,R
```
will "clear" the screen as all graphics blocks will be reset.

```
10 FILL X+RND(5),IY+2*Y1,4,Y2*Y2
```
will draw and fill the rectangle determined by the evaluated arguments as long as the argument values obtained are within the limits specified in the text.

```
20 FILL X*(3+Z),Y1+.5,X2,47-Y2,R
```
will reset the rectangle determined by the evaluated arguments with the same cautions as in the preceding example.

HIMEM hex-constant
HIMEM int-expression
HIMEM lsb,msb expression
HIMEM pos-expression

HIMEM allows you to protect high memory so that it will not be used by BASIC. The argument specifies the address of the last byte usable by BASIC. An ILLEGAL FUNCTION CALL error will result if this address is not at least 200 bytes above the start of user memory. HIMEM resets the values of all variables and does a CLEAR 50; caution should therefore be employed when using HIMEM within a BASIC program.

EXAMPLES:

```
HIMEM &HFFFF
```
will allow BASIC to use all of memory (48K machine). This will result in an ILLEGAL FUNCTION CALL error if used in a 32K machine.

```
10 HIMEM +40000
```
will protect memory above 40000 (decimal).

```
10 INPUT X : HIMEM +X
```
will protect memory above the input value of X. Note the use of the positive ("+") sign before X. This is to allow for values greater than 32767. A different way to limit the values of the argument is:

```
10 INPUT X% : HIMEM X%
```
will allow X to range from -32768 to +32767.

#"label"

The pound sign ("#") preceding a quoted-string serves to identify the string as a label. If a line is to be labeled, the pound sign must be the first non-blank character in the line after the line number. The pound sign must be followed by a double quote, a string of 0 to 251 alpha-numeric characters (except a double quote), and the terminating double quote. If the line is not null, there must be a colon seperating the label and the BASIC statement which follows. Labels should be unique, although no check is performed to insure this. If two lines are labeled with the identical label, a reference to the label will always refer to the first (lowest numbered) line. A line should only be labeled once--any subsequent labels will be ignored.

EXAMPLES:

```
100 #"DELAY" : DO : T$=INKEY$ : UNTIL (T$<>"") : RETURN
```
labels line 100 as #"DELAY". It may now be referenced by an NGOSUB #"DELAY" statement.

```
10 #"START"
.
.
5000 NGOTO #"START"
```
labels line 10 #"START". A sample reference is shown in line 5000.

LCASE

The LCASE command toggles the lower case display enable. The LCASE command supports both the Radio Shack and Electric Pencil type lower case hardware modifications. The first time the command is used, lower case letters may be entered and displayed by using the shift key and the appropriate letter. The next time the LCASE command is used, lower case entries are inhibited.

Lower case characters can be used anywhere in composing a BASIC program but can only be displayed by PRINT statements or in REM lines.

EXAMPLES:

```
LCASE
10 PRINT"This is a test."
20 LCASE
```

if LCASE has not been used prior to the LCASE command executed in command mode, lower case will be enabled. Line 10 may be input by using the shift-key entry for all lower case characters. Line 20 will inhibit any further lower case entry or display. When the program is RUN, the line, "This is a test.", will be displayed.

```
10 LCASE
20 STOP
```

will result, if lower case is not enabled before the program is run, in the display of the line, "Break in 20". Note that the message is displayed in both upper and lower case letters.

LINE X1,Y1,X2,Y2 {,R}

The LINE command without the trailing ",R" turns on the graphics blocks of the line determined and terminated by the points X1,Y1 and X2,Y2. The LINE command with the trailing ",R" resets (i.e. erases) the graphics blocks of the determined line. This command has the same limits and parameters as the FILL command.

NOTE: Parenthesis before and after the arguments field (unlike SET and RESET) will cause an error.

EXAMPLES:

```
10 LINE 0,0,127,47
```
will draw a line from the top left of the video display to the lower right.

```
20 LINE 127,47,0,0,R
```
will reset (i.e. erase) the line set in the preceding example.

```
10 LINE 128-1,0,2*0,(4*10)+7
```
will draw a line from the top right of the video screen to the lower left.

```
100 LINE (B*B)-(A*A),C*SIN(Y),X2-X1,Y2,R
```
will reset the line determined by the evaluation of the command's arguments.

LOC. {"string"}
LOC. {string}

This command allows you to locate instances of a string within the BASIC program currently in memory. The LOC. command with an argument locates the first occurrence of the string within the program. LOC. without an argument locates the next occurrence of the last string searched for. The maximum argument string length is 20 characters.

The first argument form of LOC. ("string") should be used if the string to be searched for occurs in a quoted string (PRINT "string" or INPUT "string", for example) or a REMark. Double quotes may not be imbedded within the argument string since they serve as delimiters of the string. The second argument form of LOC. (string) should be used to locate a string that is not within a quoted string or REMark. The two forms of LOC. allow you to distinguish between a BASIC keyword and the corresponding character string, as illustrated in the examples below.

The LOC. command maintains its own current line pointer, but will adjust BASIC's LIST and EDIT line pointer as described below. To start a search for a string, use LOC. with an argument. LOC. will start its search from the first program line. If the string is located, the line containing the string will be LISTed on the screen. BASIC's LIST and EDIT pointer will now be changed to point to the LISTed line. LOC.'s own pointer will also have been changed so that the next LOC. (without an argument) will search beginning with the next line after the line just LISTed. (This means that two occurrences of the same string within the same line will be located only once.) Once the line has been LISTed, you have several choices:

1). Press <BREAK> to end LOC. and return to Command mode or the executing BASIC program (should LOC. have been executed within a program);

2). Press the "E" key to enter the EDIT mode in order to EDIT the line just LISTed. After ending EDIT, you will return to the Command mode as usual;

3). Press any other key to do an automatic LOC. (LOC. without an argument). This will find the next occurrence of the string just located.

If a search for a string fails, no line will be LISTed to the screen, and BASIC's LIST and EDIT pointer will not be altered.

EXAMPLES:

The examples below assume the following program is in memory:

```
10 NGOSUB #"NAME?"
20 PRINT : PRINT "THANK YOU";
30 PRINT "VERY MUCH";NM$
40 GOSUB 100
50 END
60 #"NAME?" : INPUT"WHAT IS YOUR NAME";NM$ : RETURN
```

```
LOC.?                   (Locate the first occurrence of the BASIC
                         token PRINT)
                  Line 20 is LISTed to the screen
<ENTER>                 (Locate the next occurrence of PRINT)
                  Line 30 is LISTed to the screen
<BREAK>                 (Exit LOC.)
                  Command mode entered
LOC."?"                 (Locate the first use of a question mark)
                  Line 10 is LISTed
<SPACE BAR>             (Locate the next occurrence of "?")
                  Line 60 is LISTed
E key pressed           (EDIT line 60)
                  EDIT mode is entered
     .                          .
     .                          .
                  EDIT mode is exited
LOC.                    (Locate the next occurrence of "?")
                  No line is LISTed and Command mode is entered
LOC.GOSUB               (Locate the first use of the GOSUB token)
                  Line 10 is LISTed
<BREAK>                 (Exit LOC.)
                  Command mode is entered
LOC.NM$                 (Locate the first use of the variable NM$)
                  Line 30 is LISTed
<BREAK>                 (Exit LOC.)
                  Command mode is entered
```

NGOSUB line number
NGOSUB #"label"

This command allows for greater flexibility in calling a BASIC subroutine. The argument "line number" may be a constant, variable, or expression. It denotes the BASIC program line number to GOSUB. If the second argument form of NGOSUB is used, '#"label"' indicates that the line number to GOSUB is labeled with #"label". Other than providing additional methods of referring to the subroutine being called, NGOSUB performs identically to GOSUB. However, NGOSUB may not be substituted for GOSUB in an ON n GOSUB command.

EXAMPLES:

```
10 NGOSUB 40000 : NGOSUB 100
20 NGOSUB SN*100
```

will cause calls to subroutines located at line numbers 40000, 100, and the product of the variable SN and the number 100.

```
10 NGOSUB #"INIT"
20 NGOSUB #"PROCESS"
.
.
1121 #"INIT"
.
.
1300 RETURN
.
.
1380 #"PROCESS"
.
.
1420 RETURN
```

will cause the subroutines located at lines 1121 & 1380 to be called.

NGOTO line number
NGOTO #"label"

This command provides additional methods of specifying the line to GOTO. The argument "line number" may be a constant, variable, or expression. It denotes the BASIC program line number to GOTO. The argument '#"label"' refers to the line which begins with the matching #"label". Other than allowing greater flexibility in referring to the line number to GOTO, NGOTO performs identically to GOTO. However, NGOTO may not be used in place of GOTO within an ON n GOTO or ON ERROR GOTO command.

EXAMPLES:

```
10 NGOTO 10
```
will cause an "infinite" loop upon execution.

```
10 #"HERE" : NGOTO #"HERE"
```
will cause an "infinite" loop similar to the previous example.

```
10 NGOTO X
```
will cause execution to transfer to the line specified by the current value of the variable X, assuming such a line exists.

NTROFF
NTRON {expression} {,expression} ... {,expression}
NTRON line range {,expression} {,expression} ... {,expression}

The NTRON command enables a trace facility for debugging BASIC programs. It allows you to specify what lines of your BASIC program are to be traced, as well as providing a unique expression-trace capability. A BASIC program must be in memory at the time the NTRON command is executed, or an ILLEGAL FUNCTION CALL error will occur. The NTROFF command disables this trace facility.

The "line range" argument determines what lines of the BASIC program currently in memory will be traced. (See below for a detailed description of "line range".) If missing, all lines of the BASIC program will be traced. It is important to remember that the line range entered remains fixed, even if lines are added to the BASIC program (until you execute another NTRON). For example, suppose the BASIC program in memory, at the time an "NTRON" (no line range) is executed, has lines ranging from 10 to 50. If you later add lines between lines 10 - 50, these will be traced without doing another NTRON. However, if you add lines outside this range (before line 10, or after line 50), these lines will not be traced unless you execute another NTRON. The above example would still apply even if the command entered had been "NTRON (0:65529)".

The remaining arguments to NTRON constitute the expressions (if any) whose values are to be traced. There may be 0 or more of these expressions. Too many expressions (the limit depends on the sum of the length of the expressions) will result in an OUT OF STRING SPACE error. (You can not increase this limit by CLEARing more string space.) If 2 or more expressions are given, they must be seperated by commas. (Also note that if the first argument (line range) is given, a comma must seperate it from an expression.) The expressions used may be any valid BASIC expression which could properly be put in a single PRINT statement. However, NTRON does virtually no syntax checking on these expression arguments. If you enter an invalid expression, NTRON will more than likely accept it. An error will not occur until the program is run and the first line is about to be traced! At this point you will receive a syntax (or another) error, in what might appear to be a correct line. If you execute an NTROFF and the program line subsequently runs without an error, the cause of the error was probably an improper NTRON expression argument.

NTRON remains active with the arguments given (if any) until an NTROFF is executed. Before a BASIC program line being traced is about to be executed, NTRON's output will be displayed on the video screen. This consists of zero or more lines (depending on the number of arguments given with the NTRON command) of "expression=" expression value, followed by a LIST of the line to be executed. It is important to remember that the values of the expressions (if any) are those that exist before the line is executed. (This is particularly important when the expression contains a division operation. If the divisor has not yet been assigned a value, a

DIVISION BY ZERO error will occur.)

NTRON can be of great value in determining BASIC program bugs. Used within a BASIC program, it can provide for the tracing of several disjointed line ranges. This can be accomplished by simply having numerous NTRON commands with different line ranges throughout the program.

Although most output formatting commands (TAB, ;, USING, etc) can be included within "expression", this is not recommended. In particular, it is very difficult to use USING properly. Any "expression" which appears after USING will be formatted with USING (or cause an error if this can not be done). (Remember that the expression arguments must be expressions which may be entered into a single PRINT statement.) Also, the USING variable must be defined before the first line to be traced is executed, or an error will result. It is best to use expressions which do not alter the output format of NTRON, however if formatting commands are used, be prepared for anomalous results.

EXAMPLES:

```
NTRON
```
will cause a trace of all BASIC program lines currently in memory. No expressions will be traced.

```
NTRON(500:1000),X,Y
```
will cause a trace of all BASIC lines between 500 and 1000 (inclusive). The current values of the variables X and Y will be displayed preceeding the line trace (LIST).

```
NTRON INT(X)+A*B,A$
```
will cause a trace of all BASIC program lines. Before each line is traced, the values of the expression INT(X)+A*B and of the variable A$ will be displayed.

```
10 NTRON(:50),RB,TE$,LEN(A$),MID$(A$,LEN(A$)-1,2)
.
.
50 REM
.
.
1000 NTRON(1000:)
```

will cause lines 10 through 50 to be traced after line 10 is executed. Additionally, the values of the expressions shown will be displayed prior to the line trace display. After line 1000 is executed, only line 1000 and any lines that follow will be traced.

line range

Line range specifies a line or a range of lines of a BASIC program. Whenever used it must be enclosed by parentheses. A line range must be followed by a comma or a colon (whichever is appropriate) if it does not terminate a line. A line range can be given in several forms, as detailed below. Several examples illustrate the usage of each form. The examples use constants for "line number", but it should be understood that variables or numeric expressions may also be used. All examples assume the following program is in memory:

```
10 REM
20 PRINT "THIS IS A SHORT PROGRAM"
30 PRINT "WHOSE ONLY PURPOSE IS TO"
40 PRINT "OCCUPY LINES 10 - 50"
50 REM
```

1). (line number 1 : line number 2)

specifies the range of lines beginning from "line number 1" (inclusive) through "line number 2" (inclusive). If "line number 1" does not exist, the first line number that does exist which is greater than "line number 1" will be used as the lower limit of the range. If "line number 2" does not exist, the first line number that does exist which is less than "line number 2" will be used as the upper limit of the range.

EXAMPLES:

```
(20 : 40)
   includes lines 20, 30 and 40.

(5:25)
   includes lines 10 and 20.

(18:100)
   includes lines 20, 30, 40 and 50.
```

2). (: line number)

specifies the range of lines beginning from the first program line through "line number" (inclusive). If "line number" does not exist, the first line number that does exist which is less than "line number" will be used as the upper limit of the range.

EXAMPLES:

```
(:100)
   includes lines 10, 20, 30, 40 and 50.

( : 28)
   includes lines 10 and 20.
```

(:5)
does not include any line and will result in an error.

3). (line number :)
specifies the range of lines beginning from "line number" (inclusive) through the last program line. If "line number" does not exist, the first line that does exist which is greater than "line number" will be used as the lower limit of the line range.

EXAMPLES:

(Ø :)
includes lines 1Ø, 2Ø, 3Ø, 4Ø and 5Ø.

(45 :)
includes line 5Ø.

4). (line number)
specifies a particular line which must exist. If "line number" does not exist, an error will result.

EXAMPLES:

(1Ø)
specifies line 1Ø as the "line range".

(18)
will result in an error since line 18 does not exist.

ON ERROR GOTO line number

The ON ERROR GOTO command has been modified in several ways, but will still function properly with existing programs. The changes made include:

1). The "line number" argument may be a variable or expression, as well as the usual numeric constant. This change does not apply for ON ERROR GOTO used in Command mode.

2). An ON ERROR GOTO "line number constant" issued in Command mode will be ignored. (An ON ERROR GOTO with a variable or expression given as the "line number" will cause an error).

3). If an error occurs in Command mode, any ON ERROR GOTO in effect will be ignored.

Note: You can not trap NEWBASIC's UNTIL WITHOUT DO or MISSING NEWBASIC COMMAND errors. These errors will always cause execution to terminate.

EXAMPLES:

10 ON ERROR GOTO X+100
will cause execution to transfer to the line number determined by the sum of the current value of X and 100, upon an error during program execution.

ON ERROR GOTO 10000
will be ignored since it was issued in Command mode.

POFF
PON

The PON command, when executed, will cause video output to be echoed to the printer. The printer must be on and ready or the system will "hang". The POFF command is used to disable the video to printer echo. The command may be used with custom printer driver routines (serial,TRS232,etc.) providing the routine has been loaded before issuing a PON. See SPOOLON for the use of it and PON simultaneously.

EXAMPLES:

```
10 PON
20 PRINT"THIS IS A TEST"
30 POFF
```

will display the line "THIS IS A TEST" on the video display and also output the line to the printer.

```
PON : LIST
```

this command will cause a program in memory to be listed on the video display and to also be output to the printer.

QUICKEY

This command toggles the quick key entry method on and off. The first time the command is given, quick key entry is enabled; the next time it is given, quick key entry will be disabled.

Quick key entry allows you to enter entire BASIC keywords with two keystrokes. The first key alerts the computer that you desire to make a quick key entry. This key is either the down-arrow on the left hand side of the keyboard, or <CLEAR> on the right hand side. If a second key is not entered immediately, a blinking asterisk will appear at the current cursor position. This is a reminder that you are in quick key entry mode. If the key you press next is not a defined quick key, then it will be displayed (or the action such as <ENTER> or <CLEAR> will occur) as though you had never initiated a quick key entry. If the key you press is defined, then the associated keyword will immediately be displayed on the video screen. When quick key entry is enabled, the <CLEAR> and down-arrow keys do not function as usual. You can issue a down-arrow (that is, a line feed) or clear simply by pressing the desired key twice rather than once. You may also press <SHIFT> <CLEAR> to issue a clear.

The table below lists BASIC keywords and associated keys, organized alphabetically by the BASIC keyword. An attempt has been made to make quick key entries easily recallable. Many of the keywords are mnemonic (that is, the first letter of the keyword matches the associated key). Exceptions are the BASIC string keywords (that is, keywords with "$" in them) which have been assigned to numeric keys. Other keywords also use mnemonics different from the BASIC keyword (Xfer (X) for GOTO or Bring (B) for LOAD, for example). Some keywords use keyboard position to aid in recalling their assigned quick keys (RETURN (H), for example, is next to GOSUB (G)). Of course, some keywords had to be assigned at random (you can't win them all!).

AUTO	- A	KILL"	- K	READ	- Y
CHR$(	- 4	LEFT$(	- 1	RESET(	- Q
CLEAR	- Z	LIST	- L	RETURN	- H
CLOSE	- J	LOAD"	- B	RIGHT$(	- 3
CMD"	- C	MEM	- M	RUN <ENTER>	- R
DATA	- D	MID$	- 2	SAVE"	- S
DIM	- U	MKD$	- 8	SET(	- W
ELSE	- E	MKI$	- 9	STR$(	- 6
FOR	- F	MKS$	- 0	STRING$(	- 7
GOSUB	- G	NEXT	- N	THEN	- T
GOTO	- X	OPEN"	- O	VARPTR(	- V
INKEY$	- 5	PEEK(	- P	blank:blank	- :
INPUT	- I	POKE	- @		

(The above table may be photocopied and placed in the lower right corner of the video display for easy reference.)

RECT X1,X2,Y1,Y2 {,R}

The RECT command without the trailing ",R" turns on the graphics blocks of the perimeter of the rectangle determined by the imaginary diagonal line drawn from the point X1,Y1 to the point X2,Y2. The RECT command with the ",R" resets (i.e. erases) the determined rectangle perimeter. This command has the same limits and parameters as the FILL command.

NOTE: Parenthesis before and after the arguments field (unlike SET and RESET) are not needed and if present will cause an error.

EXAMPLES:

```
10 RECT 0,0,127,47
```
will draw a rectangle perimeter one graphic block wide around the outside of the video display.

```
20 RECT 127,0,0,47,R
```
will reset the rectangle displayed in the preceding example.

```
300 RECT X+15,C*SIN(D),X2*X2,43
```
will draw the determined rectangle perimeter provided the evaluated arguments are in range.

```
40 RECT 127-I,0+I,I-127,0-I,R
```
will reset the determined rectangle perimeter with the same precaution as in the previous example.

REPEAT

The REPEAT command implements the keyboard auto repeat and "beep" toggle. The first time this command is executed, the routine takes effect. A key depressed for about half a second will repeat. If a speaker/amplifier is hooked to the cassette-out line of the TRS-80 (the line that goes to the AUX on the cassette recorder), a beep will be heard upon depressing a key. The second time this command is executed, the auto repeat and "beep" are turned off. The REPEAT command should not be used with Radio Shack's lower case software as lower case characters cannot be input with REPEAT in use.

NOTE: The motor-on relay in the TRS-80 keyboard is not used. It is recommended that the audio and control cable be attached to the TRS-80 CPU/keyboard rather then the expansion interface. If this is not the case, the "beep" will not be heard unless a motor-on command has been issued at least once to the expansion interface. The seperate relay in the E/I must be turned on for audio output through its rear jacks.

EXAMPLES:

REPEAT
will enable key repeat and "beep" if this is the first time the command is issued.

```
20 REPEAT
30 INPUT"NAME?",A$
40 REPEAT
```

if a REPEAT command has already been issued, as in the preceding example, line 20 will turn off the auto repeat. Line 30 will input A$ without allowing auto-repeat. Line 40 will again enable repeat/"beep".

REST line number
REST #"label"

This command allows for selected reading of DATA statements. The argument specifies a BASIC program line number containing a DATA statement. After the REST command is executed, the next READ will start with the first item in the DATA statement within the line just RESTored.

The "line number" argument form may be given as a constant, variable, or expression. The second REST argument form (#"label") refers to a line which has been so labeled. With either argument form, if the line referenced does not contain a DATA statement, an ILLEGAL FUNCTION CALL error will result.

EXAMPLES:

```
10 REST 40000
```
will reset the DATA pointer so that the next READ will start with the first item in the DATA statement in line 40000.

```
10 REST A(X)*100
```
will cause the next READ to start with the first item in the DATA statement in the line determined by the product of A(X) and 100.

RS232

This command initiates dialog that will result in the initialization of the RS-232-C interface. Current status of the UART sense switches will be displayed within vertical bars. If the baud setting is not within the range possible to set with RS232, the word "OTHER" will be displayed. To use the parameters that were switch set, hit "ENTER" in reply to each question. To change settings, enter the appropriate number. Recommended settings are denoted by an asterisk.

EXAMPLE:

RS232
will result in the following dialog:

```
            RS-232-C INITIALIZATION
BAUD RATE (0=110, 1=134.5, 2=300*) !2!? 0
PARITY (0=ENABLE*, 1=DISABLE) !0!? 0
STOP BITS (1*, 2) !1!? 1
WORD LENGTH (5, 6, 7*, 8) !7!? 7
PARITY (0=ODD, 1=EVEN*) !1!?
```

the numbers after the "?" are user entered. The numbers inside the vertical bars are the current UART switch settings. This dialog has resulted in a baud rate of 110, parity enabled, 1 stop bit, 7 bit word length excluding parity bits, and even parity.

RSIN {,R}

The RSIN command without the trailing ",R" will enable the input of characters from either the keyboard or the RS-232-C interface. Providing processing is not too lengthy between input of characters, rates up to and including 300 baud can be supported. The RSIN command with the trailing ",R" (i.e. RSIN,R) will inhibit input from the serial port. Further input will only be from the keyboard.

This command used in conjunction with the RSOUT command makes possible the implementation of terminal programs, automatic logon to time share networks, and remote terminal input and output in BASIC!

NOTE: Do not implement the RSIN command if a RS-232-C board or similar serial I/O device is not installed. An infinite loop will occur and a system reset will be necessary.

EXAMPLES:

```
10 RSIN : RSOUT
20 LPRINT : LPRINT
30 LINEINPUT A$ : IF A$<>"USER ID?" THEN 30
40 LPRINT"123,45,6789" : REM USER LOGON ID
```

this example shows how simple an automatic logon procedure for a time share system is to implement. In line 10, the RSIN command enables input from the serial port. "RSOUT" routes printer output to the serial port. Line 20 outputs two carriage returns. This is usually necessary to establish baud rate synchronization with a time share network. In line 30, a character stream, up to a carriage return, is stored in the string variable A$. If A$ is not equal to "USER ID?" the program loops at line 30. Once the IF/THEN clause is true, the string "123,45,6789" is output to the serial port in line 40. Further processing could include an interactive terminal program - etc.

```
100 RSIN,R
```

will inhibit any further input from the serial port.

RSOUT {,R}

The RSOUT command without the trailing ",R" will route printer output to the RS-232-C interface. The serial port should be initialized before implementing this command. While any baud rate possible with RS-232-C is supported, no provision has been made for outputting nulls after a carriage return. Nulls are required by some serial printers and other devices. The RSOUT command with the trailing ",R" (i.e. RSOUT,R) will reset printer output to the condition existing before the RSOUT command. Refer to the RSIN command.

EXAMPLES:

```
10 RSOUT
20 FOR I=1 TO 10 : LPRINT"HELLO" : NEXT I
30 RSOUT,R
```

this example uses the RSOUT command in line 10. The 10 "HELLO"'s output to the printer in line 20 will go to the serial output port. Line 30 resets to conditions existing before line 10 was executed.

```
10 RSOUT : PON
20 PRINT"THIS IS A TEST"
30 POFF : RSOUT,R
```

line 10 first routes printer output to the RS-232-C interface and then echoes all video output to the printer (i.e. the serial port). Line 20 outputs the line "THIS IS A TEST" to the video display and the serial port. Line 30 turns off the video to printer echo and reroutes further printer output as per the conditions existing before line 10 was executed.

SOUND frequency,duration

The SOUND command outputs, to the cassette port, a tone of requested frequency and duration. The arguments of this command may be constants, variables, or expressions. SOUND uses the integer portion, hence, the arguments themselves need not be integers. The command is valid for frequency and duration arguments between 0 and 255 inclusive. Arguments outside this range will produce an "ILLEGAL FUNCTION CALL" error.

The motor-on relay in the TRS-80 CPU/keyboard is not used. It is recommended that the audio and control cable be attached to the CPU/keyboard rather then the expansion interface. If this is not the case, the sound will not be heard unless a motor-on command has been issued to the expansion inteface at least once. The audio out cable (AUX IN on the cassette recorder) should be connected to a speaker/amplfier or to the AUX IN of an audio system.

A table of approximate frequencies and durations for SOUND follows:

FREQUENCY ARGUMENTS

argument	tone in Hertz	argument	tone in Hertz
0	14250	17	1040
1	8100	18	1000
2	5500	19	950
3	4500	20	910
4	3500	25	750
5	3000	30	630
6	2600	35	540
7	2300	40	475
8	2100	50	380
9	1850	60	310
10	1700	80	240
11	1500	100	190
12	1425	125	150
13	1320	150	140
14	1250	200	90
15	1170	255	65
16	1100		

DURATION ARGUMENTS

argument	time in seconds	argument	time in seconds
0	.01	50	.36
5	.04	100	.72
10	.08	150	1.08
15	.12	200	1.44
25	.19	255	1.84

EXAMPLES:

```
10 SOUND 19,150
```
will produce a tone of 1000 hertz for about 1 second.

```
100 SOUND X+3*(A+B),255-X
```
will produce a tone of the determined frequency for the determined duration.

SPOOLOFF
SPOOLON "filespec"

The SPOOLON command directs printer output to the specified disk file. The command is disabled and the disk file closed by the SPOOLOFF command. Spooling, used in conjunction with the DESPOOL command, allows for more efficient hardcopy output. Printouts that normally would require a large amount of time to output may be quickly spooled. Later the created file may be despooled while other computer operations are being performed.

SPOOLON may be used before a PON command. This allows all output to the video display to also be output to a disk file. This capability is useful for keeping track of programs run, or for debugging purposes. A POFF command should be issued before SPOOLOFF if PON is previously used and a printer is not "ready". If this is not done, and the printer is not powered up and ready, the system may "hang".

Only one file at a time may be spooled to. If there is a problem initializing the requested file, or SPOOLON has already been issued, an ILLEGAL FUNCTION CALL will occur. If an error such as DISK SPACE FULL occurs during spooling, an error message will be displayed and the spooled-to file will be closed. If PON was used, a POFF command will be issued.

NOTE: The simultaneously use of SPOOLON, PON, and DIR under NEWDOS 2.1 or NEWDOS/80 may, under some circumstances, cause anomolous results.

EXAMPLES:

```
10 SPOOLON"HOLD/TXT"
```
this creates a file "HOLD/TXT" and future output directed to the printer will be placed in the file.

```
100 SPOOLON"JUNK/PRT.PASSWORD:1"
```
will create a file "JUNK/PRT" on drive 1 with the password, "PASSWORD". Printer output will be directed to this file

```
10 SPOOLON"TEST/HLD"
20 PON
30 PRINT"TESTING ... 1,2,3"
40 POFF : SPOOLOFF
```
this program will create the file, "TEST/HLD", output the line "TESTING ... 1,2,3" to both the video display and the file, stop video-to-printer echo, and close the file.

WTS

The WTS command is used to "white" the video screen. (That is, turn all graphics blocks "on".) The current cursor position is not changed.

EXAMPLES:

```
WTS
```
will white the screen "instantly".

```
10 FOR I=1 TO 10
20   WTS : PRINT @148, CHR$(23);"RED ALERT!";
30   FOR J=1 TO 100 : NEXT
40   CLS : PRINT @148, CHR$(23);"RED ALERT!";
50   FOR J=1 TO 100 : NEXT
60 NEXT
```
will display "RED ALERT!" while the video screen alternates between white and black.

GLOSSARY FOR NEWBASIC

ASCII

American Standard Code for Information Interchange. This method of coding is used for textual data.

baud

Signalling speed in bits per second. The term is usually used in conjunction with serial input and output. See RS-232-C.

command file

A DOS file with the extension /CMD. The file consists of Z-80 object code (machine language).

DCB (data/device control block)

An area in RAM associated with an Input/Output device.

expression

A meaningful sequence of one or more constants or variables possibly used with operators and functions.

filespec

A sequence of characters which specifies a particular disk file. It consists of a mandatory filename, of up to eight characters, followed by an optional extension, password, and drivespec.

hex-constant

An integer number expressed in hexadecimal notation (base 16), preceded by "&H". Leading zeros are ignored. The largest unsigned integer which can be expressed as a hex-constant is 65535 (&HFFFF).

int-expression

An expression that when evaluated lies within the BASIC integer range (-32768 to +32767). If the expression value is not an integer, it will be rounded to the closest integer.

#"label"

A string of up to 251 alphanumeric characters preceded by a pound sign and enclosed in double quotes (#"ALABEL"). Any characters other than a double quote may appear within the label defined by the begin and end quotes.

line number

A constant, variable, or numeric expression which specifies a BASIC program line number. Usually this line number must exist within the BASIC program currently in memory, or an error will result. When used within a "line range", however, "line number" need not exist.

line range

A line or a range of lines of a BASIC program. Whenever used it must be enclosed by parentheses. See NTRON.

pos-expression

An expression preceded by a plus sign ("+") indicating that the value of the expression should be evaluated as an integer between 0 and 65535 inclusive.

RS-232-C

Refers to a specific EIA (Electronics Industries Association) standard which defines a widely accepted method for interfacing data communications equipment. The TRS-80 holds its RS-232-C interface in the expansion interface.

Whenever the host computer gives the user-prompt "X", the user's inputs will be sent to the host until a carriage return is entered. Then the cycle starts over.

OTHER UTILITIES

Another useful utility contained in NewBasic is CONV. The CONV command converts any one of the following to all of the other representations—a hexadecimal number (&HXXXX), a number in decimal LSB,MSB form (255,127), an ASCII representation ("A?"), positive and negative integers (from -32768 to 32767), and large positive integers (+0 to +65535).

A lowercase driver compatible with both the Radio Schak and Electric Pencil type hardware modifications is enabled by the LCASE command.

An assembly-language routine may be called without using the cumbersome DEFUSR and USR functions. A simple CALL command is used instead. The CALL may be to positive or negative integer addresses, LSB,MSB representations, hexadecimal addresses, or addresses represented by expressions (i.e. CALL X + 2 x Y).

A partial RESTORE command is also includ*' REST restores the DATA line pointer to its argument address. REST 20 would cause the DATA statement in line 20 to be read next—no mater what the current DATA pointer may be.

All of the above commands and more are included. NEWBASIC adds many new commands to Basic found nowhere else.

It also includes many featues sold separately—and usually for a higher price than the total NewBasic package!

Documentation accompanying NewBasic is detailed, and contains many examples. The disk Basic version is more than 40 pages.

NewBasic is now in stock at CIE and is available for immediate delivery.

Modular Software Associates, developer of NewBasic, is also creator of People's Software tape 6P, the disk patch to Tandy Tiny Basic, and the upcoming games tape for the TRS-80 Color Computer, both of which can be obtained from CIE.