GENERAL SOFTWARE NOTES

Package I has been upgraded in many ways. And now the machine language debugger, DBG-8800, is included as an integral part of Package I. As a result of this change, DBG-8800 will no longer be priced separately from Package I. Instead, Package I version 3.0 will cost $75 effective immediately. Users who still have Package I or DBG-8800 on order will receive Package I/DBG version 3.0 at no extra charge. All new orders should be placed at the $75 rate (cassette or paper tape). Sale of the source of DBG-8800 on cassette or paper tape has been discontinued.

There has been a number of inquiries as to whether DBG-8800 is useful with BASIC. The answer is no. ALTAIR BASIC has its own debugging facilities designed specifically for debugging BASIC programs.

For those who are interested in more information on DBG, here is a quick example:

(underlined typed by user)

DEBUG

```
1.  $SΔ1Ø/  NOP MVI B,100 <LF>
2.      12/  NOP LXI H, #6000 <LF>
3.      15/  NOP MVI M, 0 <LF>
4.      17/  NOP INX H <LF>
5.      20/  NOP DCR B <LF>
6.      21/  NOP JNZ 15 <LF>
7.      24/  NOP .X
8.      10G
9.        Ø  BREAK @24
10. $ØΔL/ 100 <CR>
11.     B/  Ø  <CR>
12.     F/ 106! ZP
```

In the example above, <CR> stands for carriage return and <LF> for line feed. What the program does is zero out the 100 octal locations starting at location 6000 decimal (# means decimal - line 2). After the program is entered (symbolically!) a break point is set after the last instruction (line 7). Next execution is begun with a G(GO) command. When the memory clear program is done, the break point is hit and DBG types the break point number and the address of the break point (line 9). The user then examines some registers in octal mode (lines 10 & 11). The user then examines the flag (condition code) register and uses the special exclamation point command to see symbolically which flags are set.

This is a good example of how short programs may be "improvised" using DBG. The monitor program save facility could be used to save such improvised programs on paper tape or cassette.

BASIC NEWS

Disk BASIC is running! Thanks to many long hours of coding, typing, and debugging by that microcomputer programmer par excellance, Bill Gates, ALTAIR Disk BASIC has struck a new high in micro software. As mentioned before, Disk BASIC has:

Random files
Sequential files
Program saves and load from disk
Program chaining
etc.

We recommend that you have 20K bytes of memory if you wish to use disk Extended BASIC. BASIC takes about 15K minimum (can be more depending on the number of simultaneous random and sequential files the user wants to have open).

Disk BASIC will always have the cassette and line printer features built in. (No special versions should be ordered.) Disk BASIC is version 3.3 of BASIC. This means it also has:

Octal constants
Console command
Improved random number generator
Cassette numeric array save/load features

and more. These features will not be available in the 4K, 8K, and Extended versions (which will stay at version 3.2), but they will be available in ROM Extended BASIC.

ROM BASIC?

Yes indeed. No prices or delivery dates are available, but we will have BASIC on 12K of ROM. If you like to power your machine down, but don't like to reload BASIC (and can't yet afford a disk), ROM BASIC is the answer! We will have more information in coming newsletters.

For those of you who have left your fantastic compiler or whatever waiting in a drawer, now's the time to get it out! The yearly grand software prize ($1,000 in credit for ALTAIR products) will be announced at the WACC. So dig out that software and send it in! Today!

Now that Disk BASIC is done, work on finishing the DOS is underway. Files are compatible between the DOS and Extended Disk BASIC--in fact much of the same code is used. We now project a delivery date of the DOS version 1.0 of about April 15.

6800 BASIC

Is finished . . . due to the extraordinary efforts of Richard Weiland III, the 6800 now has a BASIC comparable to the 8080's. Size: About 6300 decimal bytes. It is so similar to ALTAIR 8K BASIC, the differences may be summarized on one page.

As those of you who have used 8K BASIC are aware, it is only 5900 bytes, so the 6800 version is slightly bigger (7%). 8080 addicts that we were, we expected the 6800 version to be much bigger. But Ric's efforts proved convincing. It is the concensus of most people that have programmed both CPU's that while the 8080 can be programmed in slightly tighter (and tricky) code, the 6800 is the easier of the two machines for a beginner to learn, and requires only slightly larger memory than the 8080 when programmed by an expert.

No price for 6800 BASIC has yet been set.

Please direct any questions you have about 6800 software to Mark Chamberlin, our resident 6800 systems programmer, who is presently working on the assembler, editor and monitor.

Direct any 8800 Package I questions to Paul Wasmund, who rules the realms of Package I.

SOFTWARE

Version 3.4 Altair BASIC

by Paul Allen

Version 3.4 will be released only in the disk version because version 4.0 will be released within a month for all four versions of BASIC. Version 4.0 will allow cassettes of programs to be interchangeable between the different versions of BASIC (8K, Extended, Disk), so it was decided to release 3.4 only in the disk version until 4.0 was ready. Users who have ordered 3.4 will receive 4.0 (except for those who have ordered Disk 3.4). 4.0 in the Extended and Disk versions will have constant compression and line pointers which should speed up program execution in these versions signifigantly.

3.4 and 4.0 will have all the features of 3.3 which was described in detail in the Disk documentation. This means that the Extended and Disk versions will have long lines (255 characters), the INSTR function, CONSOLE, the WIDTH command for setting terminal width, single quote (') remarks, and multiple assembly language subroutines (DEFUSR). The 8K version , Extended version, and Disk version all have octal constants and CLOADing and CSAVEing of matrices on cassette.

NOTE
The Extended version of 4.0 BASIC will require 16K bytes minimum for execution (Extended BASIC 4.0 itself requires 12K).

BASIC version 3.4 has a number of added features as well as a number of bug fixes.

The bug fixes are:

1.) BASIC (all versions) now works properly with the 4PIO board as described in previous Extended BASIC documentation. The correct status bits are now used, and BASIC does an IN from octal channel 23 to clear the output status bit after each character is output. This IN is done no matter what I/O board is used, so it is not recommended that a board other than a 4PIO be used at I/O port 23.

2.) (Extended, Disk versions) The FRE function now returns a positive number if the amount of free memory exceeds 32K bytes.

3.) (Disk version) When a random file is deleted, all the space used by the random file is freed up. Previously, if a random file was extended incrementally, only the first group (8 records) would be freed when the file was deleted.

4.) (Disk version) When simultaneously accessing two files OPENed on different disks, BASIC sometimes forgot which disk it was currently accessing. This has been fixed.

5.) (3.2 8K and larger versions) Typing in a line with a large number of ? marks could case BASIC to be wiped out. Fixed.

6.) (Disk version) The INSTR function did not free up its string temporaries properly, causing spurious "STRING FORMULA TOO COMPLEX" error messages. Fixed.

7.) (Extended 3.2 only) When subtracting double precision numbers of the same exponent of opposite sign, the sign was incorrect, i.e. PRINT 2-3 gave 1 as an answer. Fixed.

8.) (Disk Version 3.3) Use of the line printer caused unpredictable problems. Fixed.

9.) (Disk version 3.3) Use of the RND function with a negative argument caused the random number generator to return the same value over and over again. Fixed.

10.) (Disk version 3.3) Input or Output to sequential data files caused the current terminal position (POS) to be set to zero. Fixed.

11.) (All versions prior to 3.4 not fixed in 4K 3.4) If a direct GOSUB was given to a subroutine which did INPUT from the terminal, the INPUT would wipe out the direct statement, causing unpredictable results when a later RETURN was executed. Under these circumstances, 3.4 will immediately print OK and return to system level if a RETURN is executed back to a direct statement which has been destroyed by an INPUT.

The features and changes listed below are in order of the version for which they are applicable, i.e. features for 4K version first, 8K next, etc.

Additions to 4K and larger versions

Changes for 8K and Larger Versions

Control-C Interrupts INPUT statements

Control-C is now the only way to interrupt an INPUT statement. If a carriage return is typed in response to an INPUT statement, execution of the program will continue at the next statment after the INPUT without changing the values of the variables specified in the INPUT statement.

Rubout and Control-U

The rubout (octal 177) can now be used instead of backarrow (_) or underline to delete characters on an input line. The difference is that rubout prints each character that is deleted and precedes the first character deleted with a backslash (\). If deletion was in progress using rubouts and a new character is typed, a backslash will be echoed and then the new character will be typed.

Example:

```
100 X=\=X\Y=10
```

(In this case two rubouts were typed after 'X=' had been typed.)

Control-U may now be used to delete a line in the same fashion as the at-sign (@). A carriage return is printed and the current line of input is deleted.

Spaces No Longer Allowed in Reserved Words

Spaces may no longer appear inside reserved words such as THEN or AND. The only exception is GOTO which may have embedded spaces. The reason for this is to avoid statements like:

```
100 R=F OR Q
```

Being LISTed as:

```
100 R=FOR Q
```

With the corresponding SYNTAX (SN) error when the line is executed.

Pause (Control-S) and Proceed (Control-Q)

- CONTINUED -

VERSION 3.4 ALTAIR BASIC - CONTINUED -

When executing a program, Control-S may be used to cause program execution to pause so that output may be examined and then resumed with Control-O. This is especially useful when using high speed CRT terminals. After executing a BASIC statement, Control-S will cause BASIC to pause until Control-Q or Control-C is typed. Control-C will cause a BREAK and return to command level. Control-S and Control-Q are not echoed and have no effect when a program is not being executed.

Hexadecimal Constants

Hexadecimal (base sixteen) constants are now available by preceding the number with &H. If the hexadecimal value contains a character which is not A-F or 0-9 a SYNTAX (SN) error will occur. If the hexadecimal value is greater than 16 bits of signifigance (more than four hex digits), an OVERFLOW (OV) error will occur.

Examples:

```
PRINT &HFF
 255

100 LADDR=ADDR AND _&HFF 'mask off low byte
```

Octal contants may optionally be expressed either with a preceding & or with a preceding &O.

Features Available Only in Extended and Larger Versions

Control-C interrupts LINE INPUT

Control-C is now the only way to interrupt a LINE INPUT and return to command level. In version 3.3, a BEL (Control-G) was used to perform this function.

Control-C and Control-O Printing Changed

Control-C and Control-O now print as ^C and ^O when they are typed. Control-U in the Extended version also prints as ^U.

The Tab (Control-I) Character

Tab (Control-I) is used on either input or output to move the terminal carriage or cursor to the next eight column field on the terminal. The tab stops are columns 1,9,17,25,33, etc.

This is especially useful for formatting lines continued with <line feed>:

```
100<Tab>        FOR I=1 TO 10:<line feed>
<tab><tab>      FOR J=1 TO 10:
<tab><tab><tab>         A(I,J)=0:
<tab>   NEXT J,I<carriage return>
```

LISTs as:

```
100     FOR I=1 TO 10:
                FOR J=1 TO 10:
                        A(I,J)=0:
        NEXT J,I
```

> NOTE
>
> <tab> characters always print as the appropriate number of spaces.

Lower Case Input

Lower case alphabetic characters are now accepted by BASIC. Lower case characters are always echoed as lower case, but when lower case is used as part of a direct command or program statement, translation of lower case to upper case is performed if the lower case character is not part of a quoted string literal, REMark statement, or single quote (') remark.

Thus, a line input as:

```
100 print a,b:rem print out the values of a and b
```

Will be LISTed as:

```
100 PRINT A,B:REM print out the values of a and b
```

or:

```
150 if a$="basic" then 200 'test for BASIC command
```

is LISTed as:

```
150 IF A$="basic" THEN 200 'test for BASIC command
```

Brackets now Allowed as Matrix Subscript delimiters

Brackets [,] are now interchangeable with parentheses as delimiters for matrix subscripts. Thus:

```
100 A[I]=0
```

is equivalent to:

```
100 A(I)=0
```

This has been done for compatibility with other BASICs, notably HP BASIC.

CONTinue Possible after Errors

It is now possible to CONTinue after an error in a direct statement. Also, errors no longer cause loss of the current FOR...NEXT context and subroutine (GOSUB...RETURN) context.

EDIT Command Types BEL on Errors

The EDIT command will now type a BEL character (control-G) if it receives a command which it does not recognize (i.e. Y).

Error Trapping

Often it is desirable to trap execution of errors within a BASIC program in order to take action to recover from the error, or to give a better explanation of why the error occurred than a simple error message.

This facility has been added to BASIC through the use of the ON ERROR GOTO, RESUME and ERROR statments, and with the ERR and ERL variables.

Enabling Error Trapping

The ON ERROR GOTO statement is used to specify which line of the BASIC program the error handling subroutine starts. The ON ERROR GOTO statement should be executed before the user expects any errors to occur. Once an ON ERROR GOTO statement has been executed, all errors detected during the execution of the BASIC program will cause BASIC to start execution of the specified error handling routine. If the<line number> specified in the ON ERROR GOTO statement does not exist, an UNDEFINED STATEMENT error will occur.

Syntax of the ON ERROR GOTO statement:

```
ON ERROR GOTO <line number>
```

Example:

```
10 ON ERROR GOTO 1000
```

Disabling the Error Routine

IF the user desires to disable the trapping of errors he should place an ON ERROR GOTO 0 statement in his program. This disables trapping of errors, and any error will cause BASIC to print an ERROR message and stop program execution.

- CONTINUED -

VERSION 3.4 ALTAIR BASIC - CONTINUED -

If an ON ERROR GOTO 0 statement appears in error trapping subroutine, it will cause BASIC to stop and print the error message which caused the trap. It is recommended that all error trapping subroutines execute an ON ERROR GOTO 0 subroutine if an error is encountered for which they have no recovery action.

> NOTE
>
> If an error occurs during the execution of an error trap routine, the error will immediately be "forced". An error message will be printed for the error detected inside the error trap routine.

The ERR and ERL Variables

When the error handling subroutine is entered, the variable ERR contains the error code for the error. The error codes and their meanings are listed below.

THIS SECTION TO BE ADDED LATER

The ERL variable contains the line number of the line where the error was detected. For instance, if the error occurred on line 1000, ERL will be equal to 1000.

If the statement which caused the error was a direct (immediate mode) statement, the line number will be equal to 65535 decimal.

> NOTE
>
> Neither ERL nor ERR may appear to the left of the = sign in a LET or assignment statement.

The RESUME statement

The RESUME statement is used to continue execution of the BASIC program after the error recovery procedure has been performed. The user has three options. The user may RESUME execution at the statement that caused the error, at the statement after the one that caused the error, or the user may RESUME execution on a different line than caused the error.

To RESUME execution at the statement which caused the error, the user should use:

```
RESUME
```

or

```
RESUME 0
```

To RESUME execution at the statement immediately after the one which caused the error, the user should use:

```
RESUME NEXT
```

To RESUME execution at a line dfferent than the one where the error occurred, use:

```
RESUME <line number>
```

Where <line number> is not equal to zero.

Error Routine Example

The following example shows how a simple error trapping subroutine operates.

```
100 ON ERROR GOTO 500
200 INPUT "WHAT ARE THE NUMBERS TO DIVIDE";X,Y
210 Z=X/Y
220 PRINT "QUOTIENT IS";Z
230 GOTO 200
500 IF ERR=4 AND ERL=210 THEN 520
510 ON ERROR GOTO 0
520 PRINT "YOU CANT HAVE A DIVISOR OF ZERO!"
530 RESUME 200
```

The ERROR statement

In order to force an error to occur in a program, an ERROR statement has been provided. The primary use of the error statement is to allow the user to define his own error codes which can then conveniently be handled by a centralized error trap routine as described above. The format of the ERROR statement is:

```
ERROR <numeric formula>
```

Example:

```
ERROR 5
SYNTAX ERROR
```

When defining his own error codes, the user should pick values which are greater than the ones used by BASIC. Since further error messages may be added to BASIC in the future, it is recommended that error codes which are allocated from the last possible value (255) down to lower codes be used. If the <numeric formula> used in an ERROR statement is less than zero or greater than 255 decimal, a FUNCTION CALL error will occur.

If an attempt is made to print out an error message for an error which is greater than the highest defined system error, an FC error will be printed instead.

Of course, the ERROR statement may also be used to force SYNTAX or other standard BASIC errors.

Assigning String Substrings - The MID$ statement

A new statement has been added that makes it much easier to change a single character or sequence of characters inside a string without altering the other characters in the string. As an added benefit, using such a statement does not incur the numerous string allocations if concatenation is used to perform this function.

The format of the MID$ statement is:

```
MID$(<string variable>,<numeric formula 1>
 [,<numeric formula 2>])=<string formula>
```

Examples:

```
100 MID$(A$,3,2)=" "
500 MID$(N$(I),2)="TEST"
```

<numeric formula 1> specifies the first character of the <string variable> that will be replaced by the <string formula> to the right of the '=' sign. If <numeric formula 1> is greater than the length of the <string variable>, then a FUNCTION CALL error will occur.

The optional <numeric formula 2> specifies how many characters to copy into the <string variable> from the <string formula>.

Characters are copied from the <string formula> into the <string variable>, starting at the character position specified by <numeric formula 1>. They will be copied until either the end of the <string variable> is reached, the end of the <string formula> is reached, or <numeric formula 2> characters have been copied, whichever occurs first.

- CONTINUED -

VERSION 3.4 ALTAIR BASIC - CONTINUED -

More Examples:

```
Suppose T$="TEST"
Then:
MID$(T$,2)="ORT"
T$ now equals "TORT"

or

MID$(T$,3,1)=" "
T$ now equals "TE T"

or

MID$(T$,3,2)="XTEND"
T$ now equals "TEXT"
```

Zero Bytes Allowed in Sequential Disk Files

Zero bytes are now allowed as valid data bytes in sequential data files on the disk. In version 3.3, zero bytes could not be written to sequential files.

Features Added to the DISK Version Only

FILES Command Prints Files Across Line

The FILES command now prints the files on the floppy disk in columns across the page instead of down the page. This is much more convenient for CRT terminals.

*
*

Disk Hardware Notes
-continued from page 13

6. Our dealers now have Pertec FD-400 service manuals. If you suspect difficulty with the FD-400, contact your nearest dealer for his advice and service.

7. If you can't remedy the difficulty, don't try to save postage by just returning the FD-400 alone. Please return your complete 88-DCDD including Cables, Controller Boards, and Drive Chassis. This will allow us to check your system out completely and save you time, money, and hassle.

B. Disk Drive Chassis:

1. On the Buffer Card the most common difficulty is incorrect wiring or incorrectly installed ICs.

2. On the Power Supply Board be sure X1 and X3 are properly installed as indicated on the errata sheet.

3. If you suspect difficulty with the Disk Drive, DO NOT attempt to service it. Any work done on the Pertec FD-400 will void the warranty. Typical service charges for customer damaged FD-400's are $100.00.

4. Do not plug the FD-400 connector in backwards. Be sure to install the polarizing key as the instructions indicate. Plugging in the connector backwards will destroy 5-10 ICs and will cost at least $100.00 for repair.

5. If you must ship the Pertec FD-400 or complete Disk Drive unit, reinstall the Disk door block or strap. Any damage to the mechanism as a result of incorrect shipping typically costs the customer $100.00 in repair charges.

to the CPU. In general this will involve setting the SB and BUS CONTROL bits and selecting the source of the "processor" instruction. It must be noted here that by "processor" instruction we mean any of the bytes that may be required to make a complete 8080 instruction (not just the OP code). There are five sources for "processor" instructions:

CONTROL PROM (via tri-state drivers), enable: S5

Upper address switches (A15--A8), enable: S2

Lower address switches (A7--AØ), enable: S1

Upper address latch, enable: S3

Lower address latch, enable: S4

A typical PROM sequence to complete an examine would be as follows:

1) Control Instruction: Set up SB, BUS CONTROL (BC), S5

2) "Processor Instruction": JMP = 303 (octal)

3) Control Instruction: Set up SB, BC, S1

4) "Processor Instruction": 000_8. The contents of PROM are immaterial here since data is coming from address switches AØ--A7.

5) Control Instruction: Set up SB, BC, S2

-Altair 8800b- continued from page 15

6) "Processor Instruction": A8--A15

7) Control Instruction: Clear Control Latch

8) "Processor Instruction": Stop Code for PROM address counter

FRONT PANEL INTERFACE BOARD

All the lines between the 8800b bus and the Display/Control Board are now buffered through a Front Panel Interface Board. (The bus lines no longer directly drive anything on the Display/Control Board.) The Front Panel Interface Board connects to the Display/Control Board by means of two 34-conductor ribbon cable assemblies, eliminating the wiring harness between the Display/Control Board and the bus.

NEW CPU BOARD

The CPU Board consists of four major functional blocks:

8080A CPU Chip
8224 Clock Generator Chip
8212 Status Latch
Drivers and Receivers

The diagram, Figure 4, shows the relationship between these four blocks. Several points of interest are:

1. The DIG1 signal (see section entitled "New Bus Lines") controls enabling of the input data drivers (DIØ--DI7) from the bus.

2. The ready input to the 8224 (RDYIN) is the logical product of (PRDY) AND (FRDY) AND (XRDY) AND (XRDY2).

3. The bidirectional data bus to (and from) the 8080A is completely buffered (8216s).

The 8080A, the microprocessor chip itself, exercises control over the CPU board and the rest of the system. It executes the instructions stored in memory and controls all the data transfers.

The 8224 clock generator chip provides the two-phase clock (at the specified voltage levels) required by the 8080A. In addition, it synchronizes the READY and RESET inputs to the 8080A and provides a status signal (STSTB) that can be used to load the 8212 status latch. This guarantees that status data will be available as soon as possible in a machine cycle. The master timing reference for the 8224 is an external crystal (18MHz). By changing this crystal it is possible to generate the clocks used by the faster versions of the 8080A: the 8080A-1 (1.3us cycle time) and the 8080A-2 (1.5us cycle time).

-continued on page 21

Altair Business System (cont.)

points. Also available will be a POINT OF SALE option for the Inventory Management system. The Point of Sale option prepares a sales receipt while automatically updating the inventory file and providing a direct sales entry for the general ledger. The Inventory Management system is currently available and the Point of Sale option will be available during the first quarter of 1977.

A TIMEKEEPING system is also available for law or accounting offices or other professionals who bill clients for time and expenses. The Timekeeping system keeps track of time and expenses separately and provides automatic billing and statement generation. The Timekeeping system will directly interface with the general ledger and will be available during the second quarter of 1977.

The component packages of the Altair Business System are available under a one-time fee licensing arrangement. The licensing includes three years of software maintenance. Each package is accompanied by a comprehensive set of documentation including operator guides and systems guides as well as training aids.

The Altair Business System may be seen at your local Altair Computer Center. For additional information see your local Altair Computer dealer or contact the Altair Software Distribution Company, Suite 343, 3330 Peachtree Rd., N.E., Atlanta, Georgia, 30326, phone (404) 231-2308.

Program Progress

How about a program on how to do an ascending and/or descending bubble sort of complex numbers in the form (a,b) and/or a≠b; using BASIC?

Suggested By:

Roger Mann
248 Beacon Hill Drive
Ft. McMurray
Alberta, Canada
R9H 2R1

Altair BASIC 4.0

By Paul Allen

Changes to 4K and 8K versions in 4.Ø

A number of new features have been added to BASIC Version 4.Ø. They include all the non-disk features added to the extended version since 3.2--long program lines, the substring assignment MID$ function, octal and hex constants, etc.

Only minor changes have been made to the 4K and 8K versions for release 4.Ø. There are no user visible changes in the 4K except for the change in sense switch encoding (not described in this article). The main changes to the 8K version are the new editing characters (↑U-Line delete, DEL-Rubout), the improved random number generator, and saving matrices on cassette (CSAVE.A and CLOAD.A).

The material below describes only those features added between 3.4 and 4.Ø to the Disk version.

Editing Input - Control/A

The control-A character may be used to edit a line as it is being typed in. As soon as the control-A is typed, a carriage return line feed exclamation mark space sequence is printed, and the user may edit the input using any of the features of the EDIT command before hitting carriage return.

Example:

```
FOR I=1 TO 1ØØ:?A(J),:NEXT I^A
!  (SJCI<return>)FOR I=1 TO 1ØØ:?A(I),:NEXT I

<FOR loop is executed>
```

If an error is discovered during typing and before carriage return has been typed, control-A may also be used to edit the data at an INPUT Statement.

Dot - The Current Line Number

The dot character (.) may be used when a line number is expected in an EDIT, DELETE, LIST, or DELETE command. Dot is set to the current line when an error occurs, a line is listed, edited or inserted.

Examples:

```
LIST 1ØØ
1ØØ X=X+1
OK
EDIT.
1ØØ

5ØØØ THIS IS A NEW LINE
EDIT.
5ØØØ
```

Automatic Line Insertion - The AUTO Command

When a program is created, program line numbers are usually entered in sequential fashion with a standard increment between the lines. The AUTO command provides for automatic generation of line numbers when entering program lines. The format of the AUTO command is:

```
AUTO [<initial line>[,[<increment>]]]
```

Example:

```
AUTO 1ØØ,1Ø
1ØØ INPUT X,Y
11Ø PRINT SQR(X^2+Y^2)
12Ø ^C
OK
```

Control-C is used to terminate an AUTO command. If the <initial line> is omitted, an initial line of 1Ø and an increment of 1Ø is assumed. If the <initial line> is followed by a comma but no increment follows, the last increment used in an AUTO command will be used.

Continued on Page Six

SPACE$ and STRING$ Functions

Two functions are available for generating a string containing a character repeated N times. This is useful for formatting output and for blank filling fields. The STRING$ function returns a string of its string argument repeated N times. If N is zero, the null string is returned. The string argument must be non-null. Only the first character of the string argument is used to form the result string:

```
STRING$(<numeric formula>,<string formula>)

Example:

PRINT STRING$(1Ø,"A")
AAAAAAAAAA
OK
```

Optionally, a second <numeric formula> may be substituted for the <string formula>. This causes the ASCII value of the <numeric formula> to be used to generate the character to be repeated:

```
PRINT STRING$(1Ø,65)
AAAAAAAAAA
```

The most common case for STRING$ would be to generate a string of spaces. A SPACE$ function is provided to make this convenient:

```
PRINT SPACE$(10);"*"
          *
OK
```

XOR, EQV, IMP

Three new operators have been added to extended BASIC. These operators function in exactly the same way as the AND and OR operators, that is, they force their arguments to integer and then return a sixteen bit integer result. The precedence of these operators is shown in relation to AND and OR:

```
Highest:

AND
OR
XOR
EQV
IMP
```

The truth tables below describe how the bits of the result are formed from bits of the left-hand side (LHS) and right-hand side (RHS) operands.

XOR:

LHS	RHS	Result
1	Ø	1
Ø	1	1
1	1	Ø
Ø	Ø	Ø

EQV:

LHS	RHS	Result
1	Ø	Ø
Ø	1	Ø
1	1	1
Ø	Ø	1

IMP:

LHS	RHS	Result
1	Ø	Ø
Ø	1	1
1	1	1
Ø	Ø	1

The FIX Function

The FIX function takes a numeric argument and returns the truncated integer part of the argument. FIX is equivalent to SGN(X)*ABS(INT(X)).

```
PRINT FIX(-1.1),INT(-1.1)
-1            -2
```

The major difference between FIX and INT is that FIX does not return the next lowest number for negative numbers.

Continued on Page Seven

New Club Missouri-Illinois

The first meeting of the St. Louis Area Computer Club was held October 29 on the campus of Washington University. Approximately thirty people attended the organizational meeting, representing various vocations, ages, and geographic areas (including the Illinois side of the Mississippi river). Jon Elson, president pro tem, remarked that the club may be unique in having a number of obscure computers, making communication among members difficult at any level below the flowchart. Committees on hardware, software, and applications were formed, and questionnaires were circulated to aid in planning tutorials and facilities needed. Members stayed late discussing their favorite topics, with hardware reliability a commonly-voiced concern. Meetings are expected on a monthly basis. Contact:

Lou Elkins
314-427-6116, or
Box 1143,
St. Louis, MO 63188

New Club British Colombia

Anyone having questions about the newly formed British Columbia Computer Society should contact one of the members listed below.

1. Officers:

   President: Karl Brackhaus
   203-1625 W 13th Ave.
   Vancouver, BC, Canada
   V6J 2G9
   (604) 738-9341

   Treasurer: Ken Browning
   605 Spender Drive
   Richmond, BC, Canada
   (604) 271-2637

   Secretary: George Bowden
   1250 Nicola St.
   Vancouver, BC, Canada
   (604) 681-0688

2. Time and Place of meetings:

At 8:00 PM on the first Wednesday of every month in Room 129 of the British Columbia Institute of Technology (BCIT).

LINE INPUT With a String Argument

The LINE INPUT function is now available in the Disk version. It can now also take an Optional string argument which is printed as a prompt:

LINE INPUT [<quoted string literal>;] <string variable>

(Note: LINE INPUT destroys the input buffer.)

Direct INPUT Allowed/Input of Double Precision Numbers

The INPUT statement may now be given in direct mode in the extended version only without causing an "ILLEGAL DIRECT" error.

In BASIC versions prior to 4.Ø, the Double Precision number input routine was not called when double precision values were INPUT, causing imprecise values to be returned. This has since been corrected.

The amount of CPU time required to output a double precision value has also been improved. If the number is less than 1ØØØØ, the number of additions required to scale the number properly has been reduced.

Speed Improvements in the Disk Version

A number of speed improvements have been made to the Disk version. All constants in programs are now converted to a one, two, three, five or nine byte token. All GOTO, GOSUB, THEN and ERL line numbers are now converted to pointers during program execution. This means that the program does not have to be searched for GOTO references.

Changing Program Line Numbers - RENUMber

The RENUMber command allows program lines to be "spread out" to permit the insertion of a new line or sequence of lines. The format of RENUMber is:

RENUM [NN[,MM[,II]]]

NN is the new line number of the first line to be resequenced. If omitted, NN is 1Ø. MM is used to specify which lines in the program will be renumbered. Lines less than MM will not be renumbered. If MM is omitted, the whole program will be resequenced. II is the increment between the lines resequenced. If II is omitted, 1Ø is used.

To RENUMber the whole program to start at line ten with an increment of ten type: RENUM

To RENUMber the whole program to start at line 1ØØ with an increment of 1ØØ, type: RENUM 1ØØ, 1ØØ

To RENUMber lines 5ØØØ and up so they start at line 6ØØØ with an increment of 1ØØØ, type: RENUM 6ØØØ, 5ØØØ,1ØØØ

(Note: Any attempt to RENUMber parts of the program on top of other parts of the program or to create line numbers greater than 65529 will cause a "FUNCTION CALL" error.

All line numbers appearing after a GOTO, GOSUB, THEN ON . . . GOTO, ON . . . GOSUB and ERL <relational operator> will be properly changed by RENUMber to reference the new line number. ON ERROR GOTO Ø statements are not affected by RENUMber.

If a line number appears after one of the statements above but does not exist in the program, the message "UNDEFINED LINE XXXXXX IN YYYYY" will be printed. This line reference (XXXXX) will not be changed by RENUMber.

(Note: The line number YYYYYY may later be changed by RENUM to a different line number.)

PEEKing and POKEing Above 32K

It is now possible to PEEK and POKE addresses above 32767 using a positive argument. Negative arguments may still be used as before.

Continued on Page Eight

NOVEMBER SOFTWARE CONTEST

By Stan Webb

This month Henry Lacy wins both first and second place in the major program category with two useful, well-documented entries. He receives first place for a combination of two programs, a Decimal Support Package which extends the capabilities of a program already in the Altair User Software Library, and a Decimal Output Routine to print the data run through the package.

He takes second place for his Self-Incrementing Hand Loader. This program is an octal (or binary, depending on your viewpoint) loader which requires minimal hardware, that is, only an 8800 (no terminal).

Third place goes to Lee Wilkinson for his simple, useful Accounts Receivable program designed for small businesses.

Darrel Van Buer takes first place in the subroutine for his Inverse Normal Distribution Function subroutine. This routine generates random numbers with a normal distribution.

First Place Major Program

#10-15-761
Author: Henry E. Lacy
Length: 153 bytes/136 bytes
Title: Decimal Support Package (requires #8-18-752)/ Decimal Output Routine

Second Place Major Program

#10-21-762
Author: Henry E. Lacy
Length: 74 bytes
Title: Self-Incrementing Hand Loader

Third Place Major Program

#10-19-761
Author: Lee Wilkinson
Length: 60 lines Altair BASIC
Title: Accounts Receivable

First Place Subroutine

#10-12-761
Author: Darrel Van Buer
Length: 16 lines Altair BASIC
Title: Inverse Normal Distribution Function

Entries Accepted into Library

#11-4-761
Author: Gordon Berry
Length: 32 lines Altair BASIC
Title: Standardized and Weighted Scores

Continued on Page Eight

The HEX$ and OCT$ Functions

Two new functions have been provided to make conversion between numbers and their hex or octal representations easy.

Functions:

```
HEX$(<integer formula>)
OCT$(<integer formula>)
```

Example:

```
PRINT HEX$(255),OCT$(255)
FF              377
```

Both HEX$ and OCT$ return a string value whose characters represent the hexadecimal or octal value of their argument. The resulting string does not have any leading or trailing spaces.

Changes in Error Messages

A new Error message has been added and an old error message changed.

A MISSING OPERAND (Code 29) Error message occurs if an operator is encountered during formula evaluation, but no operand succeeds it:

```
PRINT 2+
MISSING OPERAND
```

The UNDEFINED STATEMENT error message has been changed to print UNDEFINED LINE (UL in the 8K version), which is more appropriate.

Control-I and the Line Printer

Tab (Control-I) now works properly with the line printer and moves the printer carriage to the next eight character field.

New Features and Corrections to the Disk Version

Random Disk data files may no longer be LOADED as a program accidentally, causing unpredictable results. A BAD FILE MODE error will occur.

A new function has been provided in the Disk version to allow retrieval of error parameters when a DISK I/O ERROR occurs. ERR(Ø) returns the disk number of the disk. ERR(1) returns the track number (0-76) and ERR(2) returns the sector number.

The disk number is no longer included in the I/O error message.

If a LINK ERROR occurs while a file is being KILLed, the file name is now deleted from the directory.

The VARPTR Function

The VARPTR(<variable>) function returns the address of the variable given as the argument. If the variable has not been assigned a value during the execution of the program, a FUNCTION call error will occur.

The main use of the VARPTR function is to obtain the address of variable or array so it may be passed to an assembly language subroutine. Arrays are usually passed by specifying VARPTR(A[0]) so that the lowest addressed element of the matrix is returned.

(Note: All simple variables should be assigned values in a program before calling VARPTR of an array. Allocation of a new simple variable will cause the addresses of all arrays to change.)

#10-21-761
Author: Philip Romanik
Length: 30 lines HP BASIC
Title: Random
Random Number Generator

#10-18-761
Author: Jay Lucas
Length: 100 bytes
Title: Memory Test
Assembler memory test, a very thorough one.

#11-4-762
Author: Gordon Berry
Length: 300 bytes
Title: Print Registers

#10-25-761
Author: Byron Johnson
Length: 2 lines BASIC
Title: Extended Precision Square Roots

#10-25-762
Author: Byron Johnson
Length: 7 lines BASIC
Title: BASIC Line Renumbering
Renumbering program for 3.2 Extended BASIC.

#10-27-761
Author: Steven Armbruster
Length: 210 bytes 680 Assembler
Title: Political Influence

CN /Subscriptions

In the October issue of Computer Notes we stated that original subscriptions would expire at the end of December, 1976. Due to unforeseen problems in the development of an entirely new computer mailing system, we have extended all subscriptions to January, 1977. All those customers whose subscriptions will, at this time, expire should receive a notice approximately one month prior to the expiration date. The notice will describe both the terms and cost for an additional one year subscription to Computer Notes. Those persons who have already sent in their renewal orders will have these entered into the computer during the month of December, and they will become effective starting with the month of February. If you have any additional questions, please do not hesitate to contact us.