Interactive Microware, Inc.
P.O. Box 771
State College, Pa 16801
(814) 238-8294

*** A GUIDE TO BASEX ***

BASEX is a new, easy-to-learn language for 8080-type microcomputers. BASEX programs are almost as easy to write as BASIC programs; in fact, it is possible to translate most programs from BASIC into BASEX. The BASEX compiler is interactive; it combines the functions of entering, editing, compiling and executing your programs. Since the compiler can decompile its own code very rapidly, there is no need to have a separate source program.

The greatest advantage of using BASEX is that programs run 5 to 20 times faster than similar BASIC programs. Most programs that would normally have to be written in assembler language can now be written much more easily in BASEX. As an added bonus, the BASEX run-time routines are only 2K bytes long, so BASEX programs typically require about 6K bytes less memory than similar programs that are run with an 8K BASIC interpreter.

In spite of the speed and compactness of BASEX programs, most of the powerful features of BASIC are available to BASEX users. Array variables, text strings, arithmetic and logical operations on signed 16-bit integers and versatile I/O functions are available (see the instruction list on the reverse side). In addition, BASEX allows variable names of any length, block memory searches, block memory transfers and named subroutines that can pass multiple arguments to and from the calling program. Although BASEX does not support floating point number calculations or trigonometric functions, it is easy for the BASEX user to add new functions for custom applications.

The BASEX manual includes a detailed explanation of the language and fully commented listings of the assembler language run-time routines and the BASEX compiler, which is itself written in BASEX! You also get sample programs and a well documented LOADER program that is capable of relocating BASEX programs and compressing them into the smallest possible amount of memory.

PRICES:	Manual, "A Guide to BASEX"	$ 8
	Compiler and Loader Programs	
	On North Star diskette	$25
	On Meca tape	$25
	On other media (please ask)	$35

(OVER...)

BASEX PROGRAM COMMANDS

```
X,Y and Z are variables or constants; A( is an array; S$ is a string
"TEXT" is a string constant; variable names can be any length.

>> Pseudo-Operations <<
REM !BASEX! (comments)  DIM A( 10 S$ 80 (dimension arrays & strings)
*** START (program labels)  END (end of program)

>> Arithmetic Commands << (Results are in the Accumulator)
ADD X+Y (add) SBT X-Y (subtract) MLT X*Y (multiply) DIV X/Y (divide)
ABS X (absolute value) INC X (increment) DEC X (decrement)

>> Logical Commands << (Results are in the Accumulator)
AND X&Y (logical AND)   NOT X (complement)
IOR X#Y (inclusive OR)  XOR X%Y (exclusive OR)

>> Load and Store Commands <<
SET X=Y or SET A( 1=X or DEF X=Y A( 1=X ... (equate variables)
DAT A( X=Y Z ... (initialize an array)
BRD X (read byte at X)  BRT X=Y Z ... (write bytes from X onward)
WRD X (read word at X)  WRT X=Y Z ... (write words from X onward)

>> String Commands <<
STR X=S$ Y      or STR A( X=S$ Y (string to variable transfer)
STR S$ X X=Y    or STR S$ X X=A( Y (variable to string transfer)
STR S$ X Y=S$ Z or STR S$ X Y="TEXT" (string to string transfer)
CMP S$ X Y-S$ Z or CMP S$ X Y-"TEXT" (compare strings)
LEN S$ X (length of string)  VAL S$ X (numeric value of string)

>> Program Control Commands <<
JMP CC X (jump to X on condition CC = EQ, NE, LT, GT, LE, or GE)
FOR X=Y......TIL X+Y>Z (loop)     GTO Y (unconditional branch)
CAL X Y Z ... (call subroutine X) GET Y (pass Y to subroutine)
PUT Z (return Z to caller)        RET (return from subroutine)

>> Input/Output Commands <<
INP X A( Y S$ Z ... (input)       CHR X (print character)
PRT X A( Y S$ Z "TEXT"... (print) TAB Y (tab to column Y)
XIN X (read input port)   WAT X Y (wait for condition Y on port X)
XOT X (write output port) DSI/ENI (disable/enable interrupts)

>> Memory Search and Move Commands <<
BSR X Y Z (search from X to Y for byte Z)
WSR X Y Z (search from X to Y for word Z)
MOV X Y Z (move memory from X through Y to Z)
```

BASEX COMPILER COMMANDS

```
LST or LSM (list all or part of the program or symbol table)
DMP or NTR (dump or enter one or more memory values)
INS or DLT (insert or delete one or more program statements)
SIZ (print program size)  LOC (set program pointer)
RUN (run the program)     MON (exit to monitor)
```

Interactive Microware, Inc.
P.O. Box 771
State College, Pa 16801
(814) 238-8294

*** A GUIDE TO BASEX ***

BASEX is a new, easy-to-learn language for 8080-type microcomputers. BASEX programs are almost as easy to write as BASIC programs; in fact, it is possible to translate most programs from BASIC into BASEX. The BASEX compiler is interactive; it combines the functions of entering, editing, compiling and executing your programs. Since the compiler can decompile its own code very rapidly, there is no need to have a separate source program.

The greatest advantage of using BASEX is that programs run 5 to 20 times faster than similar BASIC programs. Most programs that would normally have to be written in assembler language can now be written much more easily in BASEX. As an added bonus, the BASEX run-time routines are only 2K bytes long, so BASEX programs typically require about 6K bytes less memory than similar programs that are run with an 8K BASIC interpreter.

In spite of the speed and compactness of BASEX programs, most of the powerful features of BASIC are available to BASEX users. Array variables, text strings, arithmetic and logical operations on signed 16-bit integers and versatile I/O functions are available (see the instruction list on the reverse side). In addition, BASEX allows variable names of any length, block memory searches, block memory transfers and named subroutines that can pass multiple arguments to and from the calling program. Although BASEX does not support floating point number calculations or trigonometric functions, it is easy for the BASEX user to add new functions for custom applications.

The BASEX manual includes a detailed explanation of the language and fully commented listings of the assembler language run-time routines and the BASEX compiler, which is itself written in BASEX! You also get sample programs and a well documented LOADER program that is capable of relocating BASEX programs and compressing them into the smallest possible amount of memory.

PRICES:	Manual, "A Guide to BASEX"	$ 8
	Compiler and Loader Programs	
	On North Star diskette	$25
	On Meca tape	$25
	On other media (please ask)	$35

(OVER...)

BASEX PROGRAM COMMANDS

X,Y and Z are variables or constants; A(is an array; S$ is a string
"TEXT" is a string constant; variable names can be any length.

```
>> Pseudo-Operations <<
REM !BASEX! (comments)  DIM A( 1Ø S$ 8Ø (dimension arrays & strings)
*** START (program labels)  END (end of program)

>> Arithmetic Commands << (Results are in the Accumulator)
ADD X+Y (add) SBT X-Y (subtract) MLT X*Y (multiply) DIV X/Y (divide)
ABS X (absolute value) INC X (increment) DEC X (decrement)

>> Logical Commands << (Results are in the Accumulator)
AND X&Y (logical AND)   NOT X (complement)
IOR X#Y (inclusive OR)  XOR X%Y (exclusive OR)

>> Load and Store Commands <<
SET X=Y or SET A( 1=X or DEF X=Y A( 1=X ... (equate variables)
DAT A( X=Y Z ... (initialize an array)
BRD X (read byte at X)  BRT X=Y Z ... (write bytes from X onward)
WRD X (read word at X)  WRT X=Y Z ... (write words from X onward)

>> String Commands <<
STR X=S$ Y      or STR A( X=S$ Y (string to variable transfer)
STR S$ X X=Y    or STR S$ X X=A( Y (variable to string transfer)
STR S$ X Y=S$ Z or STR S$ X Y="TEXT" (string to string transfer)
CMP S$ X Y-S$ Z or CMP S$ X Y-"TEXT" (compare strings)
LEN S$ X (length of string)  VAL S$ X (numeric value of string)

>> Program Control Commands <<
JMP CC X (jump to X on condition CC = EQ, NE, LT, GT, LE, or GE)
FOR X=Y......TIL X+Y>Z (loop)    GTO Y (unconditional branch)
CAL X Y Z ... (call subroutine X)  GET Y (pass Y to subroutine)
PUT Z (return Z to caller)        RET (return from subroutine)

>> Input/Output Commands <<
INP X A( Y S$ Z ... (input)       CHR X (print character)
PRT X A( Y S$ Z "TEXT"... (print)  TAB Y (tab to column Y)
XIN X (read input port)   WAT X Y (wait for condition Y on port X)
XOT X (write output port) DSI/ENI (disable/enable interrupts)

>> Memory Search and Move Commands <<
BSR X Y Z (search from X to Y for byte Z)
WSR X Y Z (search from X to Y for word Z)
MOV X Y Z (move memory from X through Y to Z)
```

BASEX COMPILER COMMANDS

```
LST or LSM (list all or part of the program or symbol table)
DMP or NTR (dump or enter one or more memory values)
INS or DLT (insert or delete one or more program statements)
SIZ (print program size)  LOC (set program pointer)
RUN (run the program)     MON (exit to monitor)
```

Interactive Microware, Inc.
P.O. Box 771
State College, Pa 16801
(814) 238-8294

*** The BASEX Tape and Disk Guide ***

The North Star disk handler offers these features:

1. Up to four drives can be accessed and the directory of any drive can be listed.
2. Named files can be created, deleted, loaded, saved or checked for accuracy by comparison with memory.
3. As many as five files can be open simultaneously for read and/or write operations.
4. Once a file has been opened, 256 byte records may be written, read or verified in sequential or random order.
5. A BASEX program may load and execute a new program which may or may not overlay the old program and data arrays.
6. The BASEX/North Star disk handler occupies only 512 bytes of memory in addition to the normal disk operating system and the user program.
7. A Basexerciser sample program demonstrates execution of all disk operations from the keyboard.

The Meca tape drive handler provides these features:

1. Up to four tape drives can be accessed and the directory of any drive can be listed.
2. The tape on any drive can be mounted, unloaded, rewound or initialized to zero the directory.
3. Named files can be loaded, saved with automatic verification or overlayed on the same region of the tape if space permits.
4. A BASEX program may load and execute another program which may or may not overlay the old program and data arrays.
5. The BASEX/Meca tape handler occupies only 270 bytes in addition to the normal tape operating system and the user program.
6. A Basexerciser sample program demonstrates execution of all tape operations from the keyboard.

The fast random access capability of the North Star disk drive together with the economy and large capacity of the Meca tape drive provide a mass storage capability that is hard to match at double the price. If you own both of these peripherals, you can combine disk and tape operations in your BASEX programs, as exemplified by a third Basexerciser routine which is included with this package.

(OVER...)

BASEX Tape and Disk Guide

Table of Contents

PRICES:	Manual	$2Ø
	Programs on North Star disk	$15
	Programs on Meca tape	$15

NOTE: You will also need the BASEX compiler to run these programs.

Interactive Microware, Inc.
P.O. Box 771
State College, Pa 16801
(814) 238-8294

*** The BASEX Tape and Disk Guide ***

The North Star disk handler offers these features:

1. Up to four drives can be accessed and the directory of any drive can be listed.
2. Named files can be created, deleted, loaded, saved or checked for accuracy by comparison with memory.
3. As many as five files can be open simultaneously for read and/or write operations.
4. Once a file has been opened, 256 byte records may be written, read or verified in sequential or random order.
5. A BASEX program may load and execute a new program which may or may not overlay the old program and data arrays.
6. The BASEX/North Star disk handler occupies only 512 bytes of memory in addition to the normal disk operating system and the user program.
7. A Basexerciser sample program demonstrates execution of all disk operations from the keyboard.

The Meca tape drive handler provides these features:

1. Up to four tape drives can be accessed and the directory of any drive can be listed.
2. The tape on any drive can be mounted, unloaded, rewound or initialized to zero the directory.
3. Named files can be loaded, saved with automatic verification or overlayed on the same region of the tape if space permits.
4. A BASEX program may load and execute another program which may or may not overlay the old program and data arrays.
5. The BASEX/Meca tape handler occupies only 270 bytes in addition to the normal tape operating system and the user program.
6. A Basexerciser sample program demonstrates execution of all tape operations from the keyboard.

The fast random access capability of the North Star disk drive together with the economy and large capacity of the Meca tape drive provide a mass storage capability that is hard to match at double the price. If you own both of these peripherals, you can combine disk and tape operations in your BASEX programs, as exemplified by a third Basexerciser routine which is included with this package.

(OVER...)

BASEX Tape and Disk Guide
Table of Contents

PRICES:	Manual	$2Ø
	Programs on North Star disk	$15
	Programs on Meca tape	$15

NOTE: You will also need the BASEX compiler to run these programs.

BASEX: A Fast, Compact Interactive Compiler for Microcomputers

BY PAUL K. WARME
423 Kemmerer Road
State College, PA 16801

BASEX is a new intermediate-level language for microcomputers that combines some of the best features of both BASIC and EXecutable machine language code. It is almost as easy to use as BASIC and yet, it is nearly as fast and versatile as assembler language. The BASEX compiler is *interactive*; that is, it permits you to enter, list, edit and run your program without the help of any auxiliary programs, such as editors or linkage editors. In this respect, BASEX is mbre like BASIC than FORTRAN, PASCAL and other compilers. Most BASEX commands resemble their counterparts in BASIC language, and this facilitates learning the language and translating programs from BASIC into BASEX.

The main advantages of BASEX are its speed (typically 7 to 20 times faster than BASIC) and the compact size of BASEX programs (typically at least 6K bytes smaller than similar programs written in BASIC). This difference in size is due to the fact that the BASEX compiler does not have to be resident during execution and the run-time subroutines occupy only 2K bytes of memory. However, since the BASEX compiler and run-time routines together use only 8K of memory, it is generally more convenient to leave the compiler in memory while debugging your BASEX programs. A microcomputer with 16K bytes of memory is adequate to make full use of the power of BASEX. Since the BASEX compiler is able to decompile its own object code very rapidly, you only need to save one copy of the program (the object code). This is a distinct advantage over FORTRAN and other compilers, which save separate source, object and executable versions of the same program.

The Origins of BASEX

About a year ago, I was having a lot of fun with my IMSAI 8080 microcomputer until I tried some BASIC programs to draw pictures on my video terminal and play electronic music. Unfortunately, BASIC was too slow to permit smooth motion of video images and accurate timing of musical notes. After buying a chess program written in BASIC, I found that it took about ten minutes to make a move. The Bulls and Cows game (similar to Mastermind) in *101 Basic Games* often took 10 to 30 minutes to make a move, much to my consternation. In short, I discovered that there is a rather large class of problems that demand greater execution speed than BASIC can deliver. Unfortunately, many of the things that I wanted to do, such as interactive graphics, electronic music and word processing, fall in this category.

At the time, it seemed that programming in assembler language was the logical alternative to BASIC, although I dreaded the thought of writing large programs (>8K) in assembler language. When I started translating one of these large programs into assembler language, it was quite discouraging. I had to reinvent the wheel at every turn, writing routines to input and output numbers and text, multiply, divide, etc. Then I noticed that certain operations were done repeatedly in different parts of the program and could be called as subroutines, thus avoiding repetitious coding. After thinking about this, it occurred to me that most of these same subroutines would be needed in just about any large assembler language program.

From this point, it was easy to conclude that assembler language programming would be much easier and faster if all of these useful subroutines could be made available in any program. A method was needed to pass arguments to the subroutines; the most straightforward way of doing this would be to insert the addresses of the arguments immediately after the subroutine call. This approach requires that a region of memory be set aside for a *symbol table*; that is, a table containing the names of variables and their present values.

Pulling on the Bootstraps

The next step was to write a set of subroutines that would be generally useful in almost any program and then write a compiler in BASIC that would translate three-letter mnemonic names of each subroutine into a CALL to the corresponding subroutine. The compiler also had to copy each new variable name included in the user program into the symbol table and leave a two-byte space for the value associated with that name. Once the compiler was working in BASIC, I used it to translate the compiler into BASEX (in other words, a series of CALL's to BASEX subroutines). If this sounds circular, well, it is.

I now had a compiler written in BASEX which worked about 20 times faster than the compiler written in BASIC. The speed advantage of BASEX was already paying off! In order to put some finishing touches on the BASEX compiler, the original version was copied to a different part of memory and the original compiler was used to modify the relocated copy. Incidentally, writing a compiler in a new language is a good test of its versatility and power. In fact, I decided to add a few more subroutines to the run-time package during the process of translating the compiler into BASEX.

An Overview of BASEX Program Commands

Table I summarizes the commands available to BASEX users. Notice that all of the command mnemonics are three letters long and the data types include 16-bit signed constants and variables (-32,767 to 32,767), one-dimensioned arrays of up to 255 values, string variables of up to 255 characters and string constants of any length. One unusual feature of BASEX array names is that the "(" is treated as part of the name, much like the "$" at the end of a string name. Thus, the closing parenthesis,), is not needed after the element. For example, DATA(1 means the first element of the array called DATA(. Variable names may contain any number of characters (or numbers), which makes BASEX programs more descriptive than BASIC programs. Since variable names are saved only once (in the symbol table), the memory cost of using longer, more meaningful names is slight.

Pseudo-Operations:

The BASEX pseudo-operations are used only by the compiler; they are skipped over during execution of the program. The REM command is for comments, which are set off by exclamation marks. The DIM command tells the compiler how much space to save in the symbol table for arrays and strings. The *** command marks an entry point in the program which can be addressed by the GTO, JMP and CAL commands. This feature of using names instead of numbers for entry points adds to the readability of BASEX programs. The END command indicates the end of a procedure or program.

Arithmetic and Logical Commands:

All of the arithmetic and logical commands place their result in the accumulator, which is referred to by the letter A (this is the only reserved variable name). The accumulator, A, is a 16-bit variable that can be used just as any other BASEX variable. The sign of the most recent arithmetic or logical operation is retained for conditional branching via the JMP command. The arithmetic commands allow addition (ADD), subtraction (SBT), multiplication (MLT) or division (DIV) of 16-bit signed constants or variables. Commands are also available to determine the ABSolute value (ABS), INCrement a value by one (INC) or DECrement by one (DEC). The logical commands (AND, NOT, IOR, XOR) operate simultaneously on all bits of a 16-bit constant or variable.

Load and Store Commands:

BASEX provides several ways to load and store values in memory. The SET command is equivalent to the LET command in BASIC. The DEF (DEFine) command allows multiple SETs to be combined in one command. However, it is more convenient to use the DAT command to assign values to successive elements of an array. The BRD (Byte ReaD) command works like the 8-bit PEEK or EXAM functions in BASIC, while the WRD command gives the BASEX user the added capability of reading 16-bit words anywhere in memory. The complementary functions of storing bytes or words in memory are performed by the BRT (Byte wRiTe) and WRT commands. These commands are analogous to the POKE or FILL commands in BASIC, but it is easier to store multiple bytes or words using BASEX, since multiple values can be stored with a single command.

String Commands:

Basex needs only one very powerful string command, STR, to duplicate the ASC, CHR$, LEFT$, MID$ and RIGHT$ string functions used in some versions of BASIC. Characters can be transferred back and forth between strings and variables (or arrays). Substrings are specified in terms of their first and last character positions in a given string. The CMP command CoMPares the alphabetic order of two strings and allows conditional branching via the JMP command, based on the outcome of the comparison. Other useful string commands are the LEN command to determine the LENgth of a string or substring and the VAL command, which returns the numeric value of a string in the accumulator, A.

Program Control Commands:

One of the most useful program control commands is the JMP command, which branches to a labelled (***) entry point if a particular condition has resulted from the most recent arithmetic, logical or compare command. The conditions that can be tested for are EQual (EQ) to zero, Not Equal (NE), Less Than (LT), Greater Than (GT), Less than or Equal (LE) and Greater than or Equal (GE) to zero. The GTO command can be used for unconditional branching. Structured programming buffs may choose not to use the GTO command, but after all, if you know where you're going, the GTO is the quickest way to get there. However, structured programming techniques can be used with BASEX.

Instead of the FOR. . .NEXT loop structure of BASIC, BASEX uses the FOR and TIL commands. The FOR command sets the initial value of the loop variable, while the TIL command adds or subtracts any value from the loop variable and then checks to see whether the terminating value has been exceeded. If not, the program branches to the statement following the corresponding FOR command. FOR. . .TIL loops can be nested to any depth and there are no restrictions on jumping into or out of loops.

BASEX has a much better facility for calling subroutines than BASIC. For starters, subroutines are called by name, rather than by statement number, so the subroutine name can

```
                    BASEX-PROGRAM COMMANDS

X,Y and Z are variables or constants; A( is an array; S$ is a string
"TEXT" is a string constant; variable names can be any length.

>> Pseudo-Operations <<
REM !BASEX! (comments)  DIM A( 10 S$ 80 (Dimension arrays & strings)
*** START (Program labels)  END (End of program)

>> Arithmetic Commands << (Results are in the Accumulator)
ADD X+Y (add) SBT X-Y (subtract) MLT X*Y (multiply) DIV X/Y (divide)
ABS X (absolute value) INC X (increment) DEC X (decrement)

>> Logical Commands << (Results are in the Accumulator)
AND X&Y (logical AND)   NOT X (complement)
IOR X#Y (Inclusive OR)  XOR X%Y (Exclusive OR)

>> Load and Store Commands <<
SET X=Y or SET A( 1=X or DEF X=Y A( 1=X ... (equate variables)
DAT A( X=Y Z .... (initialize an array)
BRD X (read byte at X)  BRT X=Y Z ... (write bytes from X onward)
WRD X (read word at X)  WRT X=Y Z ... (write words from X onward)

>> String Commands <<
STR X=S$ Y      or STR A( X=S$ Y (string to variable transfer)
STR S$ X X=Y    or STR S$ X X=A( Y (variable to string transfer)
STR S$ X Y=S$ Z or STR S$ X Y="TEXT" (string to string transfer)
CMP S$ X Y-S$ Z or CMP S$ X Y-"TEXT" (compare strings)
LEN S$ X (length of string)  VAL S$ X (numeric value of string)

>> Program Control Commands <<
JMP CC X (jump to X on condition CC = EQ, NE, LT, GT, LE, or GE)
FOR X=Y.......TIL X+Y>Z (loop)     GTO Y (unconditional branch)
CAL X Y Z ... (call subroutine X)  GET Y (pass Y to subroutine)
PUT Z (return Z to caller)         RET (return from subroutine)

>> Input/Output Commands <<
INP X A( Y S$ Z ... (input)        CHR X (print character)
PRT X A( Y S$ Z "TEXT"... (print)  TAB Y (tab to column Y)
XIN X (read input port)   WAT X Y (wait for condition Y on port X)
XOT X (write output port) DSI/ENI (disable/enable interrupts)

>> Memory Search and Move Commands <<
BSR X Y Z (search from X to Y for byte Z)
WSR X Y Z (search from X to Y for word Z)
MOV X Y Z (move memory from X through Y to Z)

                    BASEX COMPILER COMMANDS

LST or LSM (list all or part of the program or symbol table)
DMP or NTR (dump or enter one or more memory values)
INS or DLT (insert or delete one or more program statements)
SIZ (print program size)  LOC (set program pointer)
RUN (run the program)     MON (exit to monitor)
```

reflect its purpose. The first argument after a CAL command is the entry point name, defined elsewhere in your program by a *** command. The optional second and subsequent arguments specify variables to be passed to or returned from the subroutine. Within the subroutine, you can use the GET command to receive a value from the calling program or use the PUT command to return a value. Since the variables named in the GET and PUT commands may be different from the names used in the CAL command, you can reserve "local" variable names for use in subroutines. However, any subroutines that are compiled at the same time and share the same symbol table may access any variable without using the GET and PUT commands. This allows "global" variables to be shared by subroutines. The RET command is used to RETurn from a subroutine.

Input/Ouput Commands:

The I/O commands in BASEX make it easy to communicate with terminals and instruments of all types. To customize BASEX for your terminal, you simply fill in the address of your routines to input and output one character, as described in the manual. The INP command INPuts one line of characters or numbers, ending with a carriage return, and stores them in the variables listed after the INP. The PRT command PRinTs a list of variables or constants, separated by spaces. The PRN command does the same, but omits the space between items. You may space over to a particular column by using the TAB command or print a single CHaRacter with the CHR command.

If you wish to communicate directly with I/O ports, you can INput from port X with the XIN command or OuTput to port X with the XOT command. In both cases, the byte to be communicated is placed in the accumulator, A. The WAT command WAiTs until the input port specified by its first argument gives a non-zero value when ANDed with its second argument. You can also ENable or DiSable Interrupts with the ENI and DSI commands.

Memory Search and Move Commands:

Since the block memory search and move commands are extremely fast, they are indispensible for writing compilers, editors and other programs where speed is important. The BSR command SeaRches for a particular Byte (8 bits) within a specified region of memory, while the WSR command SeaRches for Words (16 bits). If the specified byte or word is found, its address is placed in the accumulator, A, and the condition flags are set to the Less Than (LT) condition so that a JMP command can test whether the search was successful. If the requested byte or word is not found, the Greater Than (GT) condition is flagged. The MOV command MOVes a group of bytes from one region of memory to another. It gives correct results even when the new region of memory overlaps the old region.

Using the BASEX Compiler

The interactive BASEX compiler makes it easy to write and debug programs. Each time BASEX is restarted, it prints "RANGE?" and expects you to enter four numbers to designate the first and last addresses of your program and the first and last addresses of its symbol table. Let's call these addresses PRG1, PRG2, SYM1 and SYM2, respectively. BASEX works with decimal numbers, since most people think most readily in base 10. If your BASEX program is already loaded in memory and you have previously declared its memory space, you can type a 0 after the "RANGE?" prompt, and then a carriage return (here denoted <CR>). In this case, BASEX will print the current values of PRG1, PRG2, SYM1 and SYM2, just as a reminder.

If you want to write a new BASEX program, you should respond to the "RANGE?" prompt by typing the starting address of the program (PRG1), a space, and then type the same value for PRG2, followed by a space. The space is the preferred separator between numeric arguments, although any non-numeric character will do. Now, you should type the address of the last byte in memory that you want to set aside for the new program, a space, and then type the same number once more, followed by <CR>. This sets SYM1 and SYM2 to the top of memory because the symbol table builds downward from the top of memory. The BASEX compiler will now erase any program that previously existed between PRG2 and SYM1.

Whether you are starting a new program or revising an old one, the value of PRG2 is next printed, followed by a question mark. The "?" always signals that a BASEX program (the compiler, in this case) is waiting for user input. You can now type any three-letter BASEX command mnemonic, a list of arguments and then <CR>. The BASEX compiler checks to see whether your command line begins with a compiler command (discussed below) and if so, performs the requested action and then retypes the value of PRG2 and "?" once more.

If your command line contains a *program* command, the appropriate CALL to a BASEX run-time subroutine is inserted in your program, followed by the symbol table addresses of any arguments specified in the command line. If any new variable names occur in the argument list, they are copied into the symbol table. Next, the compiler updates PRG2 to the address of the next available program location, prints it and waits for another command line after the "?" prompt. Notice how the automatic line number feature of BASEX facilitiates program entry.

BASEX Compiler Commands

The BASEX compiler commands allow the user to list and modify programs. The SIZ command prints the values of PRG1, PRG2, SYM1 and SYM2 in the following format:

```
PROGRAM ##### #####   SYMBOLS #####
      #####
```

The LOC command allows you to set PRG2 to any position in your program, so that the next program command will be inserted there. If you want to LiST your entire program, you can type LST; if you type one number after LST, the single command at that location will be listed or, if you type two numbers after the LST command, all program commands between those limits will be listed. The LSM command is used to LiSt the entire SyMbol table and one or two arguments may follow the LSM to list selected parts of the symbol table. As each variable name is listed, its location in the symbol table and the current value of that variable are also printed. One useful debugging technique is to insert an END command at some point in your program and then, when your program halts, list all or part of the symbol table to determine the current values of your variables. You can now alter the values of certain variables, list the program or modify the program. In most cases, you can still resume your program after the END command without harmful side-effects.

The INS and DLT compiler commands are used to INSert or DeLeTe program lines. Both require two arguments, which specify the first and last memory locations to be inserted or deleted. These locations can be determined from the program listing. The INS command moves the last part of the program to a higher place in memory, prints the range of the relocated program segment, sets PRG2 to the first location to be inserted, prints PRG2 and then waits after the "?" prompt for one or more new program commands to be entered. When you finish entering new commands, you should use the DLT command to delete the unused locations up to the relocated program segment. In cases where the altered program will be the same-size or shorter than the original one, you can use the LOC command (instead of the INS command) to set PRG2 to any location in your program, enter the changes, and then delete any unwanted commands with the DLT command.

The DMP command is available for printing the decimal values stored between any two memory addresses that you specify. A carriage return and line feed are printed automatically whenever the address is evenly divisible by ten. The NTR command can be used to eNTeR one or more decimal values into sequential memory bytes, starting at the memory address specified by its first argument. When you want to RUN your program, you merely type the RUN command, followed by the desired starting address. Since your program has already been compiled as you entered it, there is no delay before your program begins. The address given after RUN can point to the first byte of any command in your program or any other program. You can restart BASEX at any time by typing RUN 0 or just RUN. After your BASEX program finishes, it will jump to location 0 and restart BASEX.

The MON command allows you to exit conveniently to your system monitor. In view of the great diversity of mass storage devices now available, BASEX contains no mass storage handlers. However, you can use your system monitor to save and load BASEX programs or you can even use BASEX to write a tape handler, as will be shown by one of the examples coming up.

BASEX Error Messages

In order to maintain fast program execution and small program size, BASEX is somewhat weak on error checking. Nevertheless, it is a rare thing for a BASEX program to run wild, because after each BASEX command is completed, a check is made to be sure that the next byte in the program is a CALL instruction. Any other instruction will cause the program to halt, and error dump will be printed and then the BASEX compiler will be restarted. Other run-time errors (of 18 different types) will also halt and print and error dump, consisting of the values of all the registers, the program location where the error was detected, the error type number and the top three words on the stack. In addition, when the BASEX compiler detects any of nine different types of program errors, it prints a two letter error code and ignores that command line.

Relocating LOADER for BASEX Programs

It is frequently desireable to be able to move a BASEX program to a position different from that at which it was compiled. A special program called the LOADER (written in BASEX) is provided for this purpose. Although the standard version of the LOADER overlays the BASEX compiler, it can be used to relocate the compiler or the LOADER itself. The LOADER commands include MOV, to MOVe a program or its symbol table to a new location, PRG to modify the PRoGram and SYM to modify the SYMbol table to take into account its new location in memory. The DMP, NTR, RUN and MON commands described above are also available in the LOADER.

The FIX command is the most powerful LOADER command; it moves the program to a new location, removes all pseudo-operations (REM, DIM, and *** labels) and moves the symbol table to directly follow the program. As each variable name is moved, its new location is printed and the name is abbreviated to its last letter to conserve space. Comments are also omitted to make the program smaller. After the FIX operation, the program can no longer be listed or changed by the BASEX compiler, nor can it be relocated by the LOADER. This provides a measure of protection for proprietary software.

EXAMPLE: Memory Test Program

Let's write a simple BASEX program to write every possible 8-bit byte in every memory location within a selected region of memory and then read it back to verify its correctness. Of course, any errors should be reported by giving the erroneous value, its location and the correct value. In order to compare BASEX with BASIC, we'll do it first in Altair BASIC, version 3.2.

```
10 REM MEMORY TEST IN BASIC
100 INPUT "FIRST AND LAST BYTES";FIRST,LAST
110 FOR I=0 TO 255
120 FOR J=FIRST TO LAST
130 POKE J,I
140 IF PEEK(J)=I GOTO 160
150 PRINT "BYTE",J,"WAS",PEEK(J),
    ";SHOULD BE",I
160 NEXT J
170 NEXT I
180 PRINT "DONE"
OK
RUN
FIRST AND LAST BYTES? 16501,16600
DONE
OK
```

In BASIC, this program took 250 seconds to test 100 memory bytes (25,600 write, read and compare operations). Now, let's translate this program into BASEX and observe how similar the two programs are.

```
*GO BASEX
RANGE ? 11000 11000 11500 11500
11000 ? REM !MEMORY TEST IN BASEX!
11006 ? PRT "FIRST AND LAST BYTES"
11012 ? INP FIRST LAST
11020 ? FOR I=0
11028 ? FOR J=FIRST
11036 ? BRT J=I
11044 ? BRD J
11050 ? SBT A-I
11058 ? JMP EQ OK
11066 ? BRD J
11072 ? PRT "BYTE" J "WAS" A ";SHOULD BE" I
11088 ? CHR 13
11094 ? *** OK
11100 ? TIL J+1>LAST
11112 ? TIL I+1>255
11124 ? PRT "DONE"
11130 ? CHR 13
```

```
11136 ? SIZ
PROGRAM 11000 11136     SYMBOLS 11387 11500
11136 ? LSM
11387 DONE" 11387
11395 ;SHOULD BE" 11395
11409 WAS" 11409
11416 BYTE" 11416
11424 OK 11100
11429 J 0
11433 I 0
11437 LAST 0
11444 FIRST 0
11452 FIRST AND LAST BYTES" 11452
11476 MEMORY TEST IN BASEX! 11476
11136 ? RUN 11000
FIRST AND LAST BYTES ? 16501 16600
DONE
RANGE ? 0
PROGRAM 11000 11136     SYMBOLS 11387 11500
```

The execution time in BASEX is 29 seconds for 100 bytes. This is 8.6 times faster than BASIC, even though this version of BASIC is the fastest one tested by Tom Rugg and Phil Feldman in their article, "Basic Timing Comparisons . . . Information for Speed Freaks," published in *Kilobaud* (June, 1977, p. 66). When I coded their longest program, Benchmark Program 7, in BASEX, it ran in 7.1 seconds, whereas Altair BASIC took 51.8 seconds and other versions of BASIC took up to 235.6 seconds. Apple's 6K integer BASIC took 28.0 seconds; this may be a better comparison with BASEX, since neither of these use floating point arithmetic. However, like all benchmark programs, there is a certain amount of variation in performance, depending on the nature of the program. In all of the cases that I have tested, BASEX runs 7 to 20 times faster than Altair BASIC, Version 3.2.

EXAMPLE: Intel Format Tape Routines with Checksum

The following program reads, writes and verifies tape files in Intel format. It is fast enough to work with my National Multiplex CC-7 data recorder running at 2400 baud (try that with BASIC!). It will also work with paper tape or other flavors of cassette tape. The use of descriptive names for entry points and subroutines makes the program fairly easy to understand.

In Intel format, the first byte of a record is a colon and all subsequent bytes are in ASCII hexadecimal form. Bytes 2 and 3 are the record length, bytes 4 through 7 are the load address and bytes 8 and 9 are the record type (0 here). Next comes the data, two ASCII digits per byte, and then the checksum, which is the negated 8-bit sum of all the digits since the colon. At the end of each record, there is a carriage return and line feed.

The verify operation is the same as the read operation, except that the values read from the tape are just compared to the contents of memory. Subroutine INCHAR reads one ASCII digit and OUTCHAR writes one digit on the tape. Both of these routines must be customized for the user's I/O ports. Subroutines READCHK and WRTCHK read and write memory bytes as two hex digits and update the checksum, while READHEX and WRTHEX do the same thing without contributing to the checksum. Subroutine HEX converts ASCII hexadecimal digits to binary, whereas HEXOUT does the reverse conversion and goes on to output the result.

The program and symbol table occupy 1,712 bytes and they can be compressed to only 1,097 bytes by means of the FIX command in the LOADER program.

```
PROGRAM 12288 13406     SYMBOLS 13453 14000
13406 ? LST
12288 REM !INTEL FORMAT TAPE ROUTINES!
12294 DIM ANSWER$ 6
12302 DEF TAPE 2 STATUS 3
12314 *** BEGIN
12320 CHR 13
12326 PRT "READ/WRITE/VERIFY/STOP"
12332 INP ANSWER$ 1
12340 CMP ANSWER$ 1 1 "R"
12352 JMP EQ READ
12360 CMP ANSWER$ 1 1 "W"
12372 JMP EQ WRITE
12380 CMP ANSWER$ 1 1 "V"
12392 JMP EQ VERIFY
12400 GTO 0
12406 *** VERIFY
12412 SET VRFLAG=1
12420 GTO READ1
12426 *** READ
12432 SET VRFLAG=0
12440 REM !READ NEXT BLOCK!
12446 *** READ1
12452 SET CHKSUM=0
12460 CAL INCHAR
12466 SBT A-58
12474 JMP NE READ1
12482 CAL READCHK
12488 SBT C-0
12496 JMP EQ BEGIN
12504 SET NBYTES=C
12512 CAL READCHK
12518 MLT C*256
12526 SET ADDRESS=A
12534 CAL READCHK
12540 ADD C+ADDRESS
12548 SET ADDRESS=A
12556 CAL READCHK
12562 FOR BYTE=1
12570 CAL READCHK
12576 SBT VRFLAG-0
12584 JMP EQ READ2
12592 BRD ADDRESS
12598 SBT C-A
12606 JMP EQ READ3
12614 PRT "V" "ERROR AT" ADDRESS
12624 GTO BEGIN
12630 *** READ2
12636 BRT ADDRESS=C
12644 *** READ3
12650 INC ADDRESS
12656 TIL BYTE+1 NBYTES
12668 CAL READHEX
12674 ADD A+CHKSUM
12682 AND A&255
12690 JMP EQ READ1
12698 PRT "R" "ERROR AT" ADDRESS
12708 GTO BEGIN
12714 REM !READ WITH CHECKSUM!
12720 *** READCHK
12726 CAL READHEX
12732 SET C=A
12740 ADD C+CHKSUM
12748 SET CHKSUM=A
```

```
12756 RET
12760 REM !READ 2 HEX DIGITS!
12766 *** READHEX
12772 CAL INCHAR
12778 CAL HEX
12784 MLT A*16
12792 SET C=A
12800 CAL INCHAR
12806 CAL HEX
12812 ADD A+C
12820 RET
12824 REM !CONVERT ASCII TO HEX!
12830 *** HEX
12836 SBT A-58
12844 JMP LT HEX1
12852 SBT A-7
12860 *** HEX1
12866 ADD A+10
12874 RET
12878 *** INCHAR
12884 WAT STATUS 2
12892 XIN TAPE
12898 AND A&127
12906 RET
12910 REM !WRITE TAPE!
12916 *** WRITE
12922 PRT "WRITE RANGE"
12928 INP FIRST LAST
12936 SET A=0
12944 FOR BYTE=1
12952 CAL OUTCHAR
12958 TIL BYTE+1 10
12970 REM !WRITE NEXT BLOCK!
12976 *** WRITE1
12982 SET NBYTES=16
12990 SBT LAST-FIRST
12998 JMP GE WRITE2
13006 REM !TERMINATOR!
13012 SET A=58
13020 CAL OUTCHAR
13026 SET A=48
13034 CAL OUTCHAR
13040 CAL OUTCHAR
13046 GTO BEGIN
13052 *** WRITE2
13058 SBT A-15
13066 JMP GE WRITE3
13074 ADD A+16
13082 SET NBYTES=A
13090 *** WRITE3
13096 SET CHKSUM=0
13104 SET A=13
13112 CAL OUTCHAR
13118 SET A=10
13126 CAL OUTCHAR
13132 SET A=58
13140 CAL OUTCHAR
13146 CAL WRTCHK NBYTES
13154 DIV FIRST/256
13162 CAL WRTCHK A
13170 CAL WRTCHK FIRST
13178 CAL WRTCHK 0
13186 FOR BYTE=1
13194 BRD FIRST
13200 CAL WRTCHK A
13208 INC FIRST
13214 TIL BYTE+1 NBYTES
13226 SBT 0-CHKSUM
13234 AND A&255
```

```
13242 SET C=A
13250 CAL WRTHEX
13256 GTO WRITE1
13262 REM !WRITE WITH CHECKSUM!
13268 *** WRTCHK
13274 GET C
13280 ADD C+CHKSUM
13288 SET CHKSUM=A
13296 REM !WRITE 2 HEX DIGITS!
13302 *** WRTHEX
13308 DIV C/16
13316 CAL HEXOUT
13322 SET A=C
13330 *** HEXOUT
13336 AND A&15
13344 SBT A-10
13352 JMP LT ASCII
13360 ADD A+7
13368 *** ASCII
13374 ADD A+58
13382 *** OUTCHAR
13388 WAT STATUS 1
13396 XOT TAPE
13402 RET
13406 END 13406 ? LSM
13453 ASCII 13374
13461 HEXOUT 13336
13470 WRITE 2 HEX DIGITS! 13470
13492 WRITE WITH CHECKSUM! 13492
13515 WRTHEX 13308
13524 WRTCHK 13274
13533 WRITE3 13096
13542 TERMINATOR! 13542
13556 WRITE2 13058
13565 WRITE1 12982
13574 WRITE NEXT BLOCK! 13574
13594 OUTCHAR 13388
13604 LAST 16100
13611 FIRST 16101
13619 WRITE RANGE" 13619
13634 WRITE TAPE! 13634
13648 HEX1 12866
13655 CONVERT ASCII TO HEX! 13655
13679 HEX 12836
13685 READ 2 HEX DIGITS! 13685
13706 READ WITH CHECKSUM! 13706
13728 READHEX 12772
13738 ERROR AT" 13738
13750 READ3 12650
13758 READ2 12636
13766 BYTE 6
13773 ADDRESS 16101
13783 256 256
13789 NBYTES 5
13798 C 0
13802 READCHK 12726
13812 INCHAR 12884
13821 CHKSUM 0
13830 READ NEXT BLOCK! 13830
13849 READ1 12452
13857 VRFLAG 1
13866 VERIFY 12412
13875 V" 13875
13880 WRITE 12922
13888 W" 13888
13893 READ 12432
13900 R" 13900
13905 READ/WRITE/VERIFY/STOP" 13905
13931 BEGIN 12320
```

(cont'd on page 39)

(cont'd from page 31)

```
13939 STATUS 3
13948 TAPE 2
13955 ANSWER$ 6 S
13970 INTEL FORMAT TAPE ROUTINES! 13970
13406 ?
```

Other Applications of BASEX

The BASEX manual includes a very useful sample program that allows you to convert numbers from any number base between 2 and 36 to any other base and do arithmetic (+, -, *, /) or logical operations (&, #, %) in any number base. This 1K-byte program will do everything the $60 TI Programmer calculator will do and more! I have also converted that chess program and the Bulls and Cows game into BASEX; both run about 10 times faster than they did in BASIC. That factor of 10 is the difference between fun and boredom. BASEX was also ideal for writing a powerful word processor that I call PRO-TYPE (see below). In order to use my North Star disk and Meca tape drives with BASEX programs, I have written some utility routines that permit random or sequential access to disk or tape files, etc. These things have a way of eating up time, so the interactive graphics and electronic music programs haven't yet been translated to BASEX, but someday . . .

Hardware Requirements and Availability

The current version of BASEX was written for the 8080, but it also runs on Z80 or 8085 microcomputers. However, it is fairly straightforward to translate the 2K bytes of run-time subroutines to run on other microcomputers. Since the BASEX compiler and LOADER are written in BASEX, they should work with minor modifications on any other microprocessor, once the run-time subroutines are available. It should also be very easy to translate any user application programs from one dialect of BASEX to another. The conversion for the 6502 is already underway and if there is sufficient interest, BASEX may be made available for other microprocessors, as well. A second limitation of the current version of BASEX is that it resides in the lowest 8K of memory (locations 0 to 5 and 64 to 1FFF), which presently precludes its use with some computers that have ROM down there (e.g., the Radio Shack TRS-80, Heath H-8).

The BASEX manual is being published by Byte Publications, Inc. and will be available from them or from your local microcomputer store. It includes a more detailed explanation of each command, sample programs and fully commented listings of the assembler language run-time subroutines, the compiler and the relocating LOADER program. A full explanation of how the BASEX user can add his own customized commands to BASEX is included in the manual. It also contains the object code for these programs in bar code format. For those of you who want to be the first on your block to try BASEX, I have a limited number of copies of the preliminary, unedited version of the manual available for $10. North Star disks or Meca tapes containing all of the programs are available for $25 from Interactive Microware, Inc., 116 South Pugh Street, State College, PA 16801. Other disk and tape formats will also be available as soon as possible. This company also markets the program that allows BASEX programs to access North Star disk or Meca tape drives and the PRO-TYPE word processor, which is written in BASEX.

BASEX Tape and Disk Guide UPDATE #1

1. Page 5 For DOS version 3, location 2BØ1 should be 5Ø, not 51 as indicated. In order to implement the new North Star DOS version 4, make the following changes in the disk handler:
2AFE=5D, 2BØ1=A2, 2BØ8=47, 2B1F=ØØ, 2B2Ø=ØØ, 2B21=ØØ,
2B2F=7B, 2B3Ø=22, 2B32=12, 2B53=B3.

2. Page 11 Change the instructions for altering the BASEX compiler as follows:

    1. NTR 8179 68 83 75
    2. NTR 2136 2 42

3. Page 2Ø In addition to the changes indicated for using a tape operating system at a location other than 7ØØØH, location 2C2Ø must be set to XF, where X is the first digit of the origin of the relocated tape operating system.

4. Page 39 Location 2545 (decimal) in the BASEX compiler must be changed (NTR 2545 39) in order to recognize the DKL, DKR and DKW (or TPL, TPR and TPW) commands.

Feel free to call me at 814-863-ØØ74 or 814-238-8294 if you have any questions or comments about these programs.

Paul K. Warme