BY SAMUEL D. FENSTER AND
LINCOLN E. FORD, M.D.

SALT

A threaded interpretive language interfaced to BASIC *for research laboratory applications*

Research laboratories such as ours have become increasingly dependent on computers. This spiraling dependence has been fueled by the decreasing cost and increasing availability and power of microcomputers. A major impediment to the further use of computers is the lack of adequate software. To overcome this limitation, we developed a method of combining compiled and interpretive higher-level languages for general use in a diversity of laboratory applications.

In our previous work with minicomputers, control programs were written in a higher-level language, such as BASIC, with assembly-language subroutines for each specialized task. When two tasks were to be performed in immediate sequence, we either wrote a new routine combining the two earlier versions or made two separate calls to the routines from BASIC. Rewriting programs requires a great deal of programming time, but making repetitive calls from BASIC frequently took unacceptably long execution times. To eliminate both of these problems, we decided to develop a means of linking assembly-language routines together so that they could be used in a single CALL statement from BASIC. Before we had progressed very far, we found that we had reinvented the threaded interpretive language (TIL), this time in a form that was interfaced directly to BASIC.

THREADED INTERPRETIVE LANGUAGES

TILs consist of a set of "words." Primary words, called *primitives*, consist of machine-language subroutines. More complex words, *secondaries*, consist of a sequence of calls to primitives. Higher-level words may call primitives or previously defined secondaries in any order. They consist simply of a list of calls to starting addresses of the words. Program execution is rapid because all the primitives are run in machine language and the only extra time is that which is required to pass parameters to each routine and to proceed from one routine to the next.

A TIL has two distinct functions, *compiling* and *running*. Compiling consists of creating the lists of tasks to be performed in sequence and placing the necessary parameters in a place where the primitives can obtain them. Running consists of carrying out the tasks in sequence. The major difference between types of threaded interpretive languages is the way the program passes from one routine to the next by a process called threading (see reference 1). We use the technique called subroutine threading. When running, the program simply makes a subroutine call to each word in sequence. Every word, both primary and secondary, ends with a return (RET) instruction. Since the sequence of routines in our application is initially called from BASIC, the final RET statement in the list returns program control to BASIC.

It might be asked why we do not use one of the existing TILs. The principal reason is that we require a more interactive control language in the laboratory, where every possible turn of events cannot be anticipated when a program is written. Another reason is that we find looping and conditional branching restrictive and difficult to use in other TILs. Finally, other higher-level languages contain commands that are convenient for performing routine tasks but are not readily available in most TILs.

BASIC

BASIC was developed as a language for beginners. In fact, most people can learn to write programs in BASIC within a few hours. This is an advantage in a university laboratory populated partially by students, where inexperience is common and personnel turnover is high. It is not, however, the major reason for using BASIC as the principal language in the laboratory. Its major virtue in this setting is a characteristic that might usually be considered a drawback, namely, that it is

(continued)

When Sam Fenster isn't working as a computer programmer at the University of Chicago, he is a sophomore majoring in mathematics and computer science at Columbia University. Lincoln E. Ford is associate professor of Medicine and Cardiology at the University of Chicago.

You can write to the authors at the University of Chicago, Section of Cardiology, Department of Medicine, Hospital Box 249, 950 East 59th St., Chicago, IL 60637.

BASIC *has become the standard language of many small computers, and thus is constantly being improved.*

an interpretive language. This means that every command in a program is translated to machine language immediately before it is carried out. The need for separately interpreting each command slows program execution greatly, but it also makes the programs highly interactive. That is, a program can be stopped at any point and the operator will know precisely where he is because his original program is unaltered.

Once a BASIC program has been interrupted, the computer can be run in immediate mode. The operator can instruct the computer to carry out commands one at a time. In this manner, it is possible for the operator to determine the value of a variable, change a variable, or even change the path of program flow. The ability to interrupt a computer that is interfaced to laboratory apparatus is invaluable when the operation of the apparatus is not entirely predictable, as is often the case.

A final advantage of BASIC is that it has become the standard language of many small computers, including the IBM Personal Computer (PC). As a result, it is constantly being improved and has many useful features.

One disadvantage of some forms of BASIC is their limited memory capability. The version used on the IBM PC can address only 64K bytes. Although this is sufficient space for most programs, laboratory apparatus can frequently generate enough data to make this space seem cramped. Subsequent analysis can usually reduce the data to a more manageable size, but it is necessary to find some way to hold the data without severely limiting the program area. To this end we have designed SALT to store raw data in areas of "high" memory above the BASIC space. An entire block of data can be transferred between this space and disk. Portions of the data in a block can also be transferred between high memory and BASIC arrays. This ability to move large amounts of data between high memory and disk and to operate on smaller parts of it in BASIC arrays provides a type of virtual memory, greatly expanding the capability of BASIC.

The major disadvantage of BASIC is its slowness. While running programs in an interpretive language, the computer cannot respond fast enough for many laboratory applications. To overcome this slowness, machine-language subroutines control the computer when it is interacting with the laboratory apparatus. A difficulty with using simple subroutines occurs when two or more of them are to be run in sequence. Program control returns to BASIC after each subroutine call. This return costs valuable time, and more importantly, the time required may be somewhat unpredictable, especially when parameters must be passed from BASIC to the subroutines. For time-critical operations in which subroutines are to be run sequentially, it was previously necessary to write additional subroutines, combining earlier ones in the proper sequence. As the number of computer applications in the laboratory increased, the continued rewriting of subroutine sequences became costly. To overcome this difficulty, we have developed a method of calling subroutines from a master machine-language routine, with one CALL statement from BASIC.

SALT

SALT is a laboratory TIL. As with all TILs, SALT's two distinct functions are compiling and running. All compilation is performed by a single assembly-language routine called from BASIC. This routine, named LOADER, creates secondary machine-language words that are subsequently called from BASIC. A secondary word consists of a sequence of CALL instructions, each followed by the starting address of the routine being called and any necessary parameters. Thus, a call from BASIC to a secondary word initiates a second call to the first word in a sequence. The final instruction in a list is RET, bringing the program back to BASIC. It is not possible to call any of the primitive subroutines directly from BASIC. The LOADER routine must first translate the name of the routine to its starting address and create a secondary word consisting of a machine-language CALL instruction followed by all necessary parameters and RET at the end.

The secondary words in a program are recompiled each time the program is run. Thus, programs in SALT are actually created and stored from within BASIC programs. It is not possible with SALT, and probably not desirable in general, to create secondary words that retain their identity outside the individual program. To use SALT, a programmer must be familiar only with the general requirements of the assembly-language routines he wishes to run and with the requirements of the LOADER command. He need not know about any previously written programs or about the specifics of the assembly-language programs. [*Editor's note: See the end of the article for details on obtaining copies of the* SALT *software package.*]

At present, we have more than 100 separate assembly-language primitives. These are grouped into functional categories described in detail below.

HARDWARE

The software described here was written specifically for an IBM PC equipped with a Tecmar Lab Master board, 512K bytes of memory, and two floppy-disk drives. It was developed out of a need for an easy and efficient way to make use of the Lab Master. The general principles can, however, be applied to any similar computer system. Almost all laboratory applications can be described as

(continued)

SPS

*SPSS/PC runs or
IBM PC/XT and AT
SAS is a registerec

a combination of the following functions: control of experiments, including timing and synchronization of external events, and setting external voltages; data acquisition, usually through the digital conversion of electrical analog signals; data storage; and data analysis. All of these functions can be implemented with the hardware and programs described here, except that the topics to be described under the heading of data analysis are limited to a few routines for simple arithmetic procedures and display that might be used as part of more sophisticated analyses.

The Lab Master board has four discrete functions required for laboratory application: an analog-to-digital (A/D) converter with 16 channels of input; two digital-to-analog (D/A) converters; a 24-channel digital input/output (I/O) device; and a chip with five programmable counters and an internal 1-MHz clock, whose basic frequency can be divided either by powers of 10 or powers of 16. Most of the software routines that we will describe are directed at implementing the four functions of the board. The remainder are used for performing simple arithmetic operations on blocks of numbers and for moving blocks of data between storage areas.

Details of Operation

SALT consists of a single compiling routine, LOADER, and a large number of small subroutines that perform very specific operations during program execution. The small subroutines are no different from any other type of assembly-language routine except that many are designed to be used in sets. For example, one subroutine may set up a counter for timing operation and another may be required to start the actual timing. The aspect of SALT that makes it unique is the LOADER routine that establishes the secondary words that are called from BASIC.

LOADER

The LOADER routine that compiles the secondary words converts all other subroutine names to addresses, eliminating the time required to look up addresses during execution. It then uses the addresses in a sequence where each one is the operand of a CALL, which is followed by any necessary parameters. A RET is placed at the end of the sequence, and the starting address of the entire sequence in memory is placed in a table. The use of memory by the LOADER routines, as well as the memory allocation for SALT and the A/D data operation, is shown in figure 1. As illustrated, separate memory areas are reserved for secondary routines and for their addresses when SALT is loaded into memory. The LOADER routine fills these areas as it compiles secondary words. In the present version of SALT, those reserved areas are relatively small because there is no need in our application to have them any larger. In principle, the only limitation to the size of these areas is the amount of available memory.

The parameters passed to the routine can be of three types: fixed integer values placed after the routine call; positions of binary bits to be set in a byte that is then placed after the routine call; and BASIC integer variables whose addresses are stored after the call. The following example illustrates the operation of LOADER.

In this example, a word called DR.2 (driver routine number 2) will first configure the 24 I/O lines into three 8-bit

(continued)

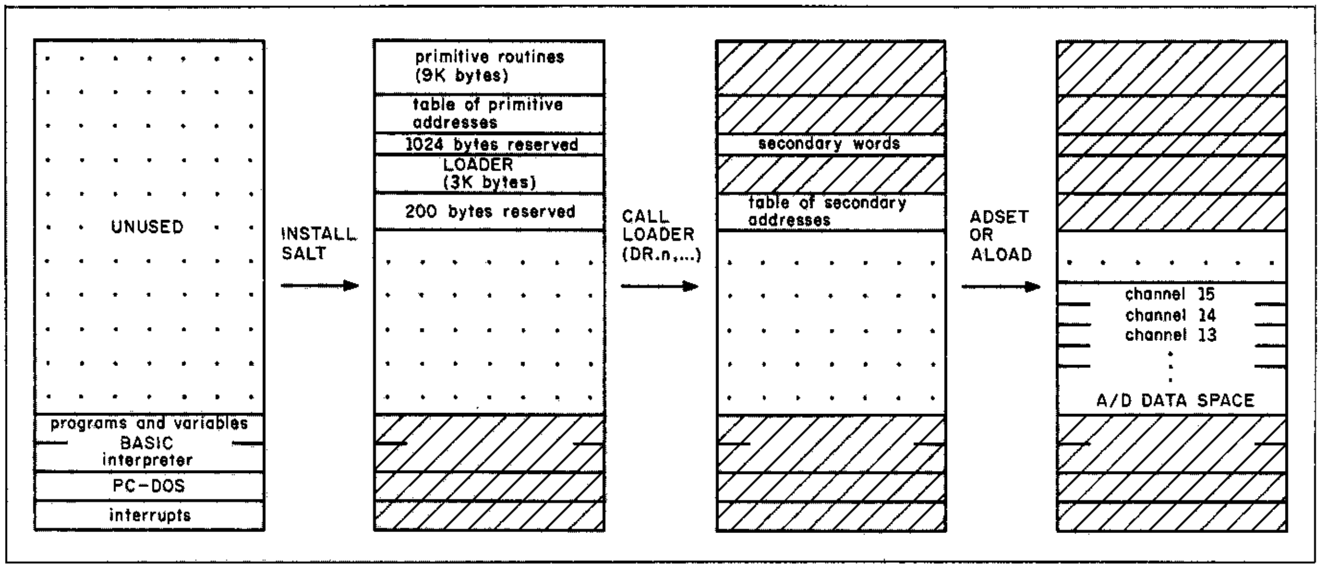


Figure 1: *Installing* SALT *with a* BASIC BLOAD *instruction puts the primitives and* LOADER *routines at the high end of memory and reserves space for the compiled secondary words and for a table of addresses of the secondary words.* A CALL *to* LOADER *creates a secondary word and places the starting address of the word in a table. The* ADSET *or* ADLOAD *routines allocate data space in memory immediately above* BASIC. *The data space is arranged into equal-size channels.*

ports such that the first (A) port is input and the remaining two (B and C) are output. Second, DR.2 initializes counter number 1 to count 20 milliseconds. Finally, DR.2 stops the counter at its initialized value to wait for the command START to begin counting. The BASIC command to perform this sequence is

```
CALL LOADER (DR.2,IOSET.1.0.0,
TIME.1.20.3, STOP.1).
```

LOADER would then create a secondary word in memory consisting of the following sequence:

```
CALL 1003
        1
        0
        0
CALL 1097
        1
       20
        3
CALL 2034
        1
RET
```

The starting address of this sequence would be placed in the second position of the *driver table*, a position reserved specifically for the address of the secondary word called DR.2. The value 1003 is the starting address of the IOSET primitive that initializes the I/O ports, and the parameters 1.0.0. specify the desired configuration. The value 1097 is the starting address of TIME, the routine that sets a counter to count from the Tecmar clock. The parameter 1 specifies the first counter, and the parameter 20 gives the number of pulses to count. The parameter 3 specifies the millisecond rate. The STOP routine, which begins at address 2034, holds counter number 1 at its pre-set count until a START.1 command begins the count.

Once the initialization is complete, another secondary word can use parts of the Lab Master board for other purposes. For example, a word called DR.3 can be written to: wait for I/O line A2 to go high; read a voltage value on A/D channel 10; put the same voltage on D/A channel 1; wait 20 milliseconds using counter number 1, which was initialized by DR.2; and set I/O channels B2 and B7 high. The BASIC command to create this sequence would be

```
CALL LOADER
(DR.3, RHA.2, VAD.10, D.VOLT,
  VDAV.1, D.VOLT, START.1, WAIT.1,
  WHB.2.7)
```

The two secondary words DR.2 and DR.3 can be called from BASIC separately, or another secondary word can be written to call them in sequence with a single call. This latter option would be compiled with the command CALL LOADER (DR.4, CDR.2, CDR.3), which would use the addresses in positions 2 and 3 of the driver table to create the assembly-language sequence

```
CALL 812
CALL 871
RET
```

CALL DR.4 is the BASIC command that would execute the entire sequence of tasks before returning to BASIC.

The LOADER command has very specific requirements, some of which have been imposed for ease of the initial programming, some of which have been created for ease of subsequent programming, and some of which have been imposed by the idiosyncrasies of IBM BASIC. For example, all of the secondary words are called DR.*N* where *N* is an integer between 0 and 100, and the commands DEFINT D and LOADER = 3 must precede the first call to LOADER in the BASIC program. The starting address of the LOADER routine is 3. The necessity for using the DEFINT D statement has been imposed to eliminate the need for typing the "%" character following each DR definition and each BASIC integer variable parameter, such as D.VOLT. The term DR is always used to name the secondary words so that the LOADER program can find the beginning of the parameter list following the words CALL LOADER. A CALL statement from BASIC pushes the entire parameter list in the statement onto the stack. The LOADER routine examines the stack to find the first variable, which always begins with DR. Once this is found, the integer following DR. is used to identify the secondary word.

Table 1: *Functional categories of SALT's primitive subroutines.*

File and data management
Timing
Analog-to-digital conversion
Digital-to-analog conversion
Digital input/output
Arithmetic procedures
Miscellaneous routines

PRIMITIVE SUBROUTINES

The individual primitive can be grouped under the separate functional categories listed in table 1. They are described below in detail by category. The listings of the individual routines are not given here because they require too much space and because they are, for the most part, specific to the Lab Master board.

When these routines were developed, the highest priority was given to speed, because they were designed to be used in real-time applications. To achieve this speed, the routines were made as simple as possible. The number of Test and Conditional Jump instructions was kept to a minimum because they take a moderate amount of program execution time. The desired simplicity was achieved at the expense of some redundancy in the program. For example, there are nine separate routines to write nonvariable data to the three digital I/O ports. These nine routines differ from each other in only two characteristics, the I/O port to be written to, and the type of output, i.e., whether the specified channels are to be made high, made low, or to toggle (change from the previous value). Since there are three alternatives for each of the three channels, there are nine combinations. Each of the routines is short (about 18 bytes), so that the memory required for nine

(continued)

New

Tru
your
the G
level
ANSI
COB
can b
and M
comp

Hig
progr
speed
COB
on the
And
handl
the hi
end us

It's r
AT&T
Micros
MS-D
to get t
compu
operati

separate simple routines is not much greater than that for a single routine that would accept the I/O port and the type of output as parameters. A larger, single routine would, however, require much more execution time.

Execution times were measured for all the time-critical routines that might be used during an experiment. These were estimated in two ways. Some execution times were measured from failures to make A/D recordings on a regular schedule. A ramp voltage was put into one channel of the A/D converter, which was programmed to make conversions at regular intervals that had been established by the frequency of the internal clock. The intervals between clock cycles were decreased in increments of 1 microsecond (µs) until the A/D failed to make a conversion. To detect a failure, the recorded ramp voltage was output to the oscilloscope using the D/A routine. A missed sample was recognized as a double step between voltage levels in the displayed ramp, as shown by the arrows in figure 2. This method could also be used to measure execution times of routines by directing the computer to perform the routines between sampling periods. A second, more straightforward, method of measuring execution times was to use a primitive called COUNT that returns the value in the buffer of a specified counter without influencing the counter's operation. By setting the counter to count down from a large number at microsecond intervals, it was possible to estimate execution times with a resolution of 1 µs. The COUNT routine took 68 µs ± 1 µs each time it was called. The execution time of routines inserted between two calls to the COUNT routine was therefore calculated by subtracting 68 µs from the measured interval.

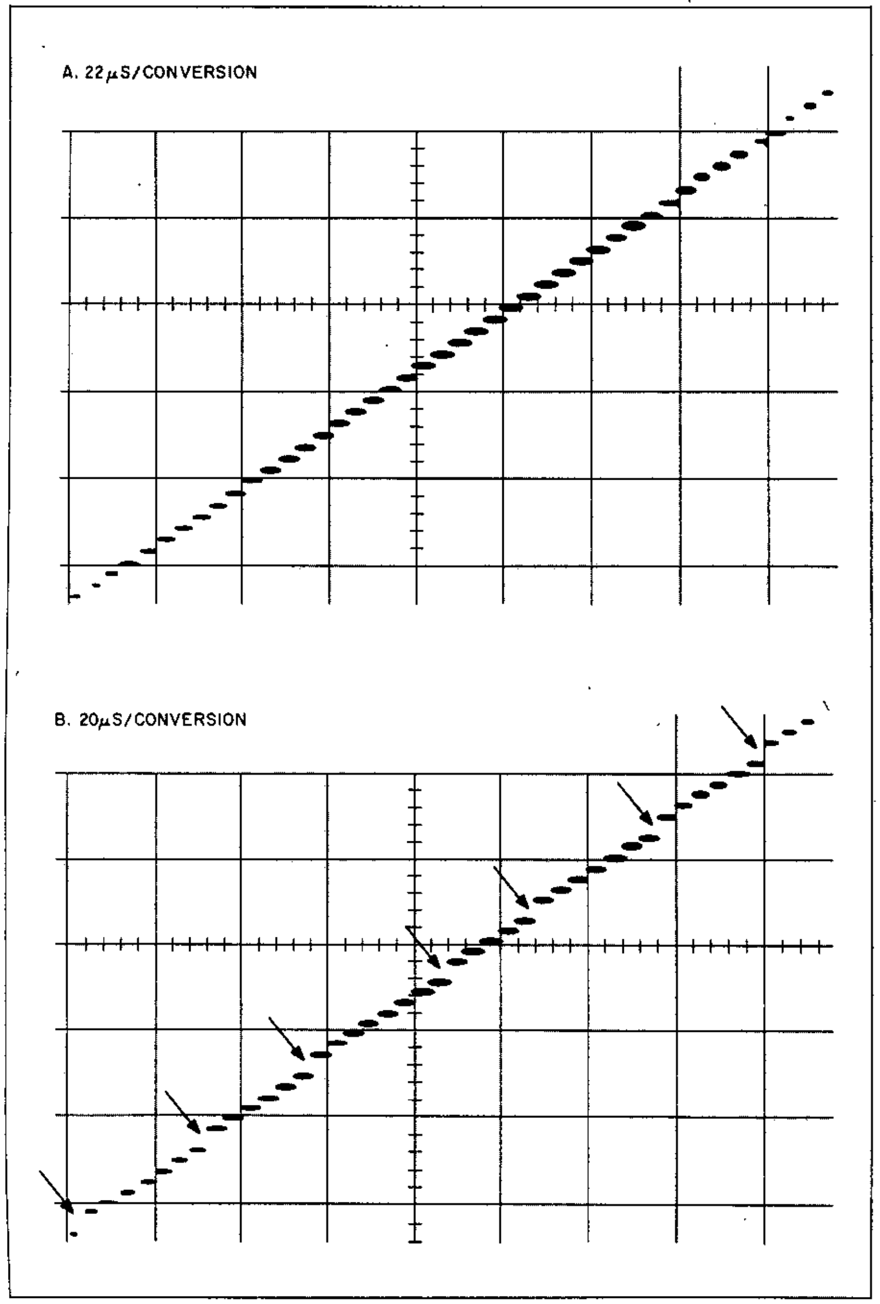


Figure 2: A *voltage ramp was recorded at two sampling intervals, differing by only* 2 *µs. The digitized data was displayed at the same rate for both. Failure to make a conversion, indicated by the arrows in* B, *resulted in a complete sample being missed after every 5–8 conversions. The voltage step after a missed sample is twice normal.*

It might be expected that the execution time for a given routine could be calculated from the number of clock cycles required for each instruction in the routine. In general, the routine took 30 to 50 percent longer than expected. A possible explanation for these longer times has been given in *PC TECH JOURNAL* (see reference 2). Further explanations are beyond the scope of this article. In addition, the execution times were somewhat variable. A possible cause of this variability in timing arises from memory-refresh cycles that occur at unpredictable intervals during program execution. This variability may cause a routine to take variable periods of time, so that variations in execution times will not occur regularly or predictably. Thus, it was necessary to

(continued)

measure execution times repetitively and to estimate an average and a range of speeds.

FILE AND DATA MANAGEMENT
All data is handled as 2-byte integer samples. A few primitives operate on a single number. For example, one primitive sets a voltage value on one of the D/A channels and another records the voltage of a single A/D channel. Most data is handled as long arrays. Before such data can be recorded, a space must be reserved for it in high memory using the ADSET routine, as shown in figure 1. This primitive requires two parameters to establish the number of channels and the number of samples per channel. Once recorded, the data can be transferred to disk using the SAVEF primitive, which creates a disk file using a filename previously specified as a BASIC string variable. The first 2 bytes in the file are used to record the number of samples per channel. When a file is read from disk using the LOADF primitive, a buffer of the correct size is first created in high memory, and the data is read into it. To use the recorded data in a BASIC program, it is necessary to move the data from its buffer in high memory to a BASIC integer array (e.g., DATAARRAY). This is done with the FETCH primitive, which requires two parameters. The first is an integer that designates the channel number, the second (e.g., D.N2) is a BASIC variable that has been set by the BASIC VARPTR function to the starting address of the BASIC array. The BASIC instruction

```
D.N2 = VARPTR(DATAARRAY(0))
```

should immediately precede a CALL to a secondary containing the FETCH primitive because BASIC moves its arrays around unpredictably. The number of the first sample to be transferred from a channel is placed in the zero position of the BASIC array using a BASIC instruction. Samples are transferred sequentially until the defined array is filled or until the end of the channel is reached. In this manner, only a portion of a record is transferred at one time, so that only a small amount of BASIC array space need be used. Another instruction called STORE performs the inverse of FETCH. It transfers data from a BASIC array to a previously defined data space in high memory.

Two additional routines, SAVEF and LOADF, transfer arrays of data in high memory to and from disk. The arrangement of data into channels is the same as that produced by the ADSET routine (see figure 1).

There is a primitive called SWITCH that allows the operator to switch be-

twee
buff
buff
a tot
is a
ADS
LOA
the
BASI
data
fer a
BASI
with

TIMI
Preci
all la
ing i
clock
Lab M
cy ca
10 or
plicat
use c

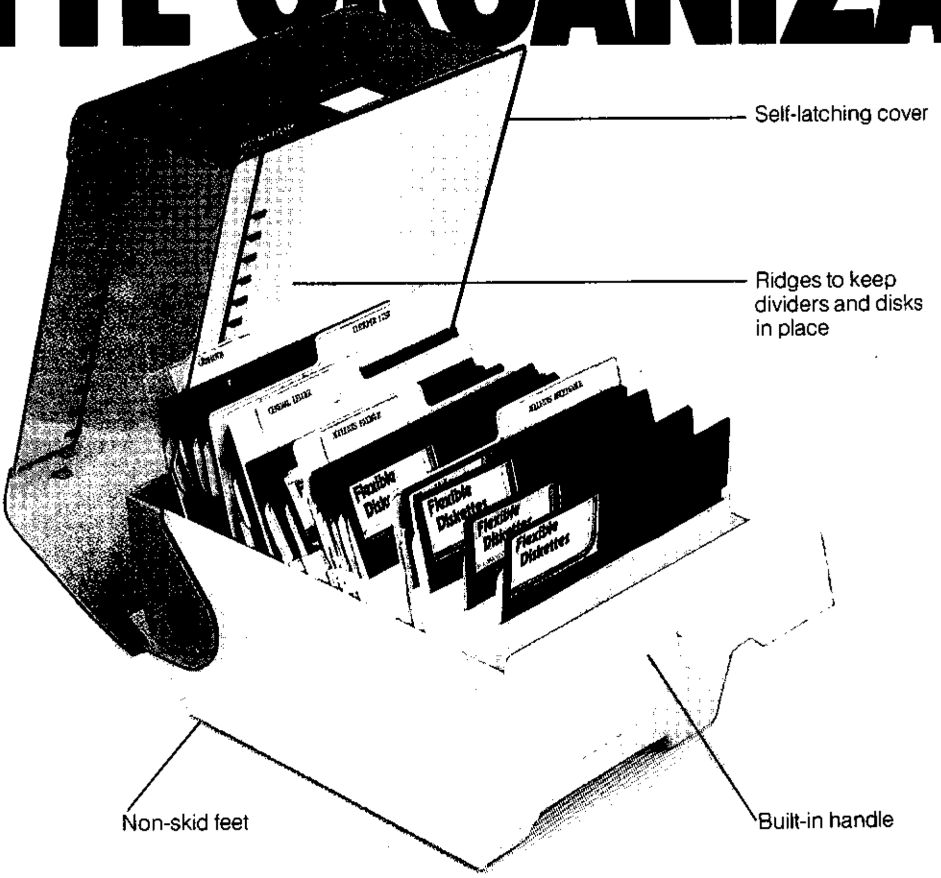
DISKETTE ORGANIZATI

Media Mate.™ The best diskette file for the money.

For organizing, protecting and transporting your diskettes, the Amaray Media Mate is the best file you can own. And you can buy it for just $15.95*

The Media Mate is available in two sizes, to hold up to fifty 5¼" diskettes, or thirty 3½" diskettes.

*Suggested retail price, Media Mate 5 Media Mate 3



AMARAY
Makers of Media Mate

Disk Org you

Whe
diskett
organi
them c
Amara
chang
along v
The
for 8," 5
Ask
Media
wherev
and co
are sol
4-AMA
neares
Interna
14935
Redmo
*Suggestec

Inquiry 2

The
to be
placed
SIC ar-
Sam-
y until
til the
In this
cord is
t only
space
uction
erse of
BASIC
l data

EF and
n high
he ar-
s is the
DSET

VITCH
ch be-

tween two simultaneously existing buffers in high memory. Since each buffer can be as large as 64K bytes, a total of 128K bytes of high memory is available. The combination of ADSET, FETCH, STORE, SAVEF, LOADF, and SWITCH greatly expand the amount of space available to BASIC for handling data. Because data can be transferred between buffer and disk and then to and from BASIC, the routines provide BASIC with a form of virtual memory.

TIMING

Precise timing is essential in almost all laboratory applications. Such timing is made available by a 1-MHz clock in the timer-counter chip on the Lab Master board. The basic frequency can be divided either by powers of 10 or powers of 16. To keep the application simple, we have chosen to use only the decimal divisions. The divided frequencies are available in two forms. They can be divided further by numbers ranging from 1 to 16 and made available externally as an "F-out" square wave. We have not yet found this provision useful and instead have chosen to perform timing using one of the five counters in the chip. There were two reasons for this decision: The F-out pulse is not available internally to software, and more importantly, it cannot be synchronized to external events. The 1-MHz clock and its dividers operate continuously so that they cannot be synchronized at all. The counter can, however, be made to begin counting in synchronization with an externally applied signal. By counting a large number of high-frequency pulses, it is possible to synchronize the counters to within the limits of the basic frequency. Since the counters can count to two pulses, it is possible, in principle, to achieve 1-μs accuracy of synchronization. In practice, the accuracy is limited to 3 μs because about 3 out of every 15 μs (14 out of every 72 central-processor clock cycles) are used by the computer for memory refresh, and it is not possible to control when the 3-μs interruptions will occur.

The counters on the Lab Master board are complex and have 18 different modes of operation. Despite this complexity, the chip containing the five counters, the 1-MHz frequency generator, and the frequency divider is controlled using two 1-byte ports. While it would be possible to initialize the counters directly from BASIC by sending data to the two control bytes, the complexity of the timers makes it much easier to use separate subroutines to establish each of the modes of operation.

(continued)

ZATION FROM AMARAY™

ching cover

s to keep
s and disks
e

n handle

DiskBank.® Organization as you need it.

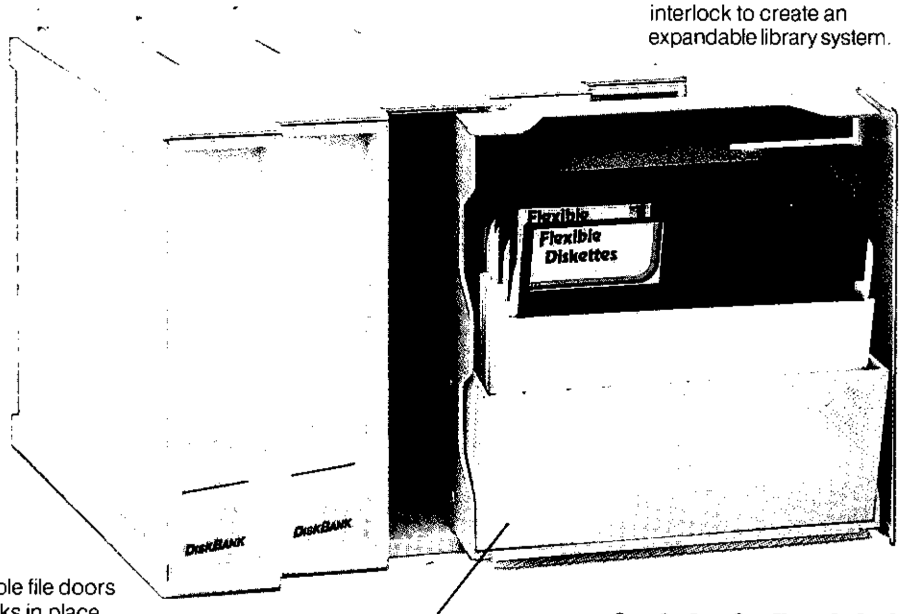
Whether you have 10 diskettes—or 210—you can organize them, and keep them organized, with the Amaray DiskBank. It changes and grows right along with your needs.

The DiskBank is available for 8," 5¼" and 3½" diskettes.*

Ask for the Amaray Media Mate or DiskBank wherever computers and computer products are sold. Or call 800-4-AMARAY for the dealer nearest you. Amaray International Corporation, 14935 N.E. 95th Street, Redmond, WA 98052.

Single module fits in a briefcase.

Modules stack and interlock to create an expandable library system.



Adjustable file doors hold disks in place whether DiskBank is placed vertically or horizontally.

AMARAY
Makers of DiskBank

*Suggested retail price, System 5— $29.95; System 3—$15.95.

Inquiry 22

The complexity of the counters arises from the combinations of available functions. Each counter can count either internal pulses or external pulses, each can count repetitively or give a single count, and each counter can be gated either internally or externally. Finally, each counter has two 16-bit registers from which it obtains its count.

In our application the counters are used for two main purposes. The first is the timing of D/A and A/D conversion. Since conversions are usually made at regular intervals, the counters are usually set to run continuously. In most cases, the duration of the pulses is not critical because the digital conversions are triggered on the changing edge of the counter pulses, so that only one of the two counter registers is used at one time to generate the correct frequency. In one application, the counter pulses are also applied to sample-and-hold circuits used to synchronize the conversion. In this case the pulses from the counter must be sufficiently long to hold all the external circuits in the hold mode until the analog values are converted. Both counter registers are used, one to establish the frequency and the other to set the pulse duration. In another application, the two counter registers are used to establish two separate frequencies. Switching between the two frequencies is accomplished with gating pulses applied externally. The use of these gates eliminates the need for reprogramming the counters and thereby hastens program execution.

The second application of the counters is to control external events by providing pulses of specified duration at specified times. The counter registers are used to establish pulse duration as well as the delay between some initiating event and the onset of the pulse. The counters can be very useful for this purpose because they operate independently of the computer's central processing unit once they have been started. This independent operation speeds program execution greatly.

A/D CONVERSION

The simplest A/D primitives convert a single analog voltage value on one A/D channel and place the digitized integer value in the BASIC variable space. All the other A/D primitives make sequential conversions at a frequency determined by a counter on the timer chip. The principal need for speed in laboratory applications is in the rapid accumulation of digitized data from electrical analog signals. There are four separate routines for this purpose and each has separate advantages with respect to simplicity, speed, and synchronization. All of the routines have been written in a way that makes it possible to interrupt

(continued)

Fancy Font $\sum_{k=2}^{n} \left| \frac{\phi(k)}{k-1} \right|$

Social *Invitations* Emphasis α ♤

Calligraphic β ♡

γ ◇

Лучшая Diplomas δ ♧

¡Español! ε §

Stylish Fancy Font ζ †

Clean *handwritten* COMPUTE

BLOCKS

Dot Matrix Made Beautiful

All the type in this ad was printed on an inexpensive dot matrix printer with Fancy Font, the program that works with almost any word processor to produce high resolution, proportionally spaced, letter quality printing.

No special hardware or installation is required, so you'll be using Fancy Font as soon as you get it.

Fonts, including Roman, Sans Serif, **Bold**, *Italic*, *Script*, Old English and more, from 8 to 24 points come standard with Fancy Font.

FLEXIBLE. Fancy Font comes with a complete set of over 1500 mathematical, foreign language and other special symbols.

Hundreds of additional fonts in sizes from 6 to 72 points are available, at a nominal additional charge, from our growing font library.

You can edit any character and also create your own characters or logos, up to 1 inch by 1 inch.

Powerful formatting features let you center, justify, wordwrap and type flush left or right, with or without running headers and footers — even with different fonts and sizes on the same line.

COST EFFECTIVE. You buy Fancy Font from your local dealer or direct from us, the software developers, for only $180.

You get near typeset quality at a small fraction of the time and cost of using art or typesetting services.

Fancy Font runs on PCDOS, MSDOS and CP/M systems with Epson, Toshiba and compatible dot matrix printers.

You'll be turning out great looking reports, newsletters, presentations, letters, mathematical texts, overheads or invitations after getting your copy of Fancy Font. The applications are limited only by your imagination.

Call or write now to order Fancy Font or ask for additional information including actual samples and independent reviews.

CALL TODAY 1-800-351-0500 MasterCharge and Visa Accepted

*Soft*Craft, Inc.

222 State Street, Madison, WI 53703 (608) 257-3300

Fancy Font is a registered trademark of SoftCraft, Inc.

ter,

fer color

ctionally
ons from
rs, sheet
nore.
electable
rial inter-
protocols
l Okidata
emulate
(-PLUS™.
capable
ut modi-

nters are
or rapid,

om 3000

ynesboro,
ginia, call

uiry 195

rate of conversion. Since the output is displayed repetitively on an oscilloscope, it is important to make this routine as fast as possible. In addition, it is highly desirable to have the sample output rate be a round number of milliseconds. The D/A routines can operate without failure at 48 μs per conversion and so can be used conveniently at 50 μs per conversion. At this rate two channels of 1000 samples each require 50 milliseconds for display. The two-channel repetition rate is thus about 10 per second. The repetition rate is actually a little lower because the oscilloscope requires a few milliseconds to reset its beam to the beginning of each trace. A repetition rate of 12 to 18 Hz is not flicker-free, but is also not too uncomfortable to scrutinize.

Digital I/O Primitives

The I/O ports are used in our laboratory to transfer 5-volt TTL (transistor-transistor logic) pulses between the computer and other apparatus. This transfer of digital signals enables the computer to control the experiments, or in some cases, to be directed by external events. Frequently the digital pulses must be transferred during other time-critical operations, such as making rapid A/D recordings. For this reason, these routines were made as simple and therefore as short as possible. As explained above, this need for simplicity resulted in a large number of very similar primitives. The simple routines take about 31 μs. If the routine is to be run during A/D sampling, an additional 50 μs is required to stop and start the sequential A/D routines.

Arithmetic Primitives

A group of primitives perform simple arithmetic manipulations on entire channels of data located in high memory. The manipulations include addition, subtraction, multiplication, and division by constants as well as integration and differentiation. These routines can be very useful in making displays, finding maxima, and detecting trends.

(continued)

New Release
4.1

Microstat®

We've continually improved Microstat since it was introduced in 1978, and the latest release includes many new features you've wanted.

- Interactive and Batch Processing
- Expanded Data Management Subsystem with New Data Transforms
- Reading data files created by other programs
- 3 types of Analysis of Variance
- Time Series
- Crosstabs and Chi-Square
- Factorials, Permutations, and Combinations
- Hypothesis Tests
- Data sets that can exceed memory
- Multiple Regression (including Stepwise)
- Scatterplots (including best fit regression)
- Correlation Analysis
- 12 Nonparametric tests
- 8 Probability Distributions
- Descriptive Statistics
- Easy Installation

Microstat's algorithms have been designed to prevent numeric overflow errors and yield unsurpassed accuracy. Microstat's price is $375.00 including the user's manual and is available for the Z80, 8086, 8088 CPU's and CP/M80, CP/M86, MS-DOS, and PC-DOS. To order, call or write.

ECOSOFT INC.

6413 N. College Ave. • Indianapolis, IN 46220
(317) 255-6476

VISA

Trademarks: Microstat (Ecosoft), CP/M (Digital Research), MS-DOS (Microsoft), PC-DOS (IBM), Z80 (Zilog), 8086, 8088 (Intel).

IF POWER FAILS, DATASAVER® TAKES OVER!

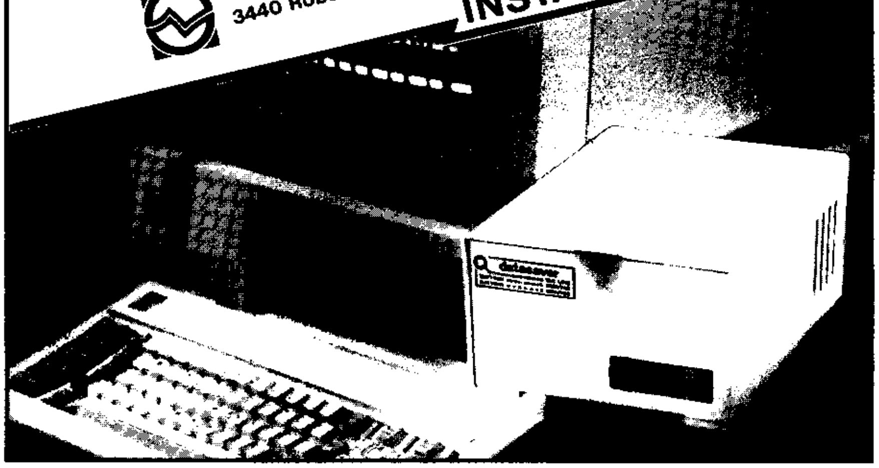
PROTECTION – saves data during power failures.
– saves hardware from overvoltage transients.
PORTABILITY – allows mobile or extended holdup time using auxiliary 12 volt battery.
FEATURES – internal battery provides 5 min. + operating time – AC line conditioning
– audible and visual alarms, interrupt signal – compact, desktop styling
– no installation required.

200 WATT — $495 / 90 WATT — $350
For special applications and product information, call 805-541-4160.

Instant power order line
805-541-4161

CUESTA SYSTEMS, INC.
3440 Roberto Court, San Luis Obispo, California 93401

INSTANT AC POWER



There are a few miscellaneous routines for operations that could also be run from BASIC, but which have been included in SALT so that program control would not have to return to BASIC each time they were to be run. These routines print messages on the monitor, sound the beeper, etc. They can be used, for example, to provide warning sounds and error messages in SALT.

FUTURE IMPROVEMENTS

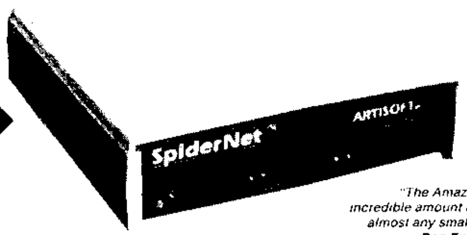
This first version of SALT contains all the necessary routines for interfacing a computer to laboratory apparatus and for manipulating blocks of data, but it contains very little else. It is obvious that more primitive routines for data analysis would be useful, and that the language might be useful outside the lab. Rapid routines for displaying records with cursors on the displays, finding maxima and minima in records, etc., would greatly speed some types of analysis. High-speed mathematical routines, such as fast Fourier transforms, could also be extremely useful in some specialized applications. Nonlaboratory applications would include all forms of assembly-language routines to be run in batches under control of an interactive higher-level language, such as BASIC. The use of assembly-language routines for performing repetitious procedures and for handling large blocks of data can greatly hasten program execution time and expand the memory space available to BASIC, without sacrificing its interactive qualities. Fortunately, the structure of the language permits the simple addition of the necessary primitives. Once the routine is written, its name, starting address, and parameter format are simply added to the tables of primitives. Since the assembly-language routines are short, it will generally not be necessary to remove old routines to make room for new ones. The current version of SALT occupies about 12K bytes of instruction space, of which 9K bytes are used for primitives. As written, the program can fill up to 64K bytes of higher memory with assembly-language programs, so that the space occupied by the primitive routines can be expanded over fivefold before economy of space becomes a consideration. [*Editor's note: You can obtain a copy of* SALT *on disk and documentation of its operation by sending* $50 *to Sam Fenster,* 4949 S. *Woodlawn Ave., Chicago,* IL 60615.] ■

REFERENCES

1. Ritter, Terry, and Gregory Walker. "Varieties of Threaded Code for Language Implementation." BYTE, September 1980, page 206.
2. Smith, Bob, and Tom Puckett. "Life in the Fast Lane: Techniques for Obtaining Timing Information with Microsecond Resolution on the PC." *PC TECH JOURNAL*, April 1984, page 63.

FINALLY!

Communications Hardware and Software that's Inexpensive, Easy to Use and Guaranteed.



"The Amazing SpiderNet ... an incredible amount of utility available to almost any small-sized application."
Ron Exner, 12/84 Hardcopy

SNARE™—SpiderNet's Callback Security System **$595**
- Protects ANY dialup computer system from unauthorized intruders
- Authorized users are called back at specific locations
- SNARE secures three modem lines at less than $200/line
- Up to three separate computers can use SNARE simultaneously
- Uses Hayes-compatible modems or others like DEC's DF03
- Stores 70 authorized users, expands to 150 (300+ soon)

SpiderNet—Printer/Plotter Sharing Unit **$495**
- Shares expensive RS-232 plotters, daisy wheel or laser printers
- Makes laser printers affordable for your office or department
- Two to five micros or minis can share a peripheral
- Connect and use—no software modifications required!
- Optional 64K buffer for spooling

SpiderNet—Computer and Peripheral Networking **$495**
- Six port, intelligent software-controlled RS-232 switch
- Share multiple peripherals between systems
- Interconnect computers to share data and files
- Three pairs of ports can be connected simultaneously
- Links ports at different baud rates
- Programmable: perfect for custom RS-232 control application

SpiderNet—Multiplexer, Terminal Concentrator and more **$495**
- Five to one RS-232 multiplexer/demultiplexer
- 5×1 or 4×2 concentrator for expanding terminal ports

ENVOY™—Telecommunications Software **$49.95**
- Access to electronic mail, remote systems and data networks
- Error free, text and binary file transfers via XMODEM or ANSI X3.28
- Smart terminal mode with capture buffer, autodial and autologin
- Easy to use, menu driven, compact and high speed
- Utilities menu for copy, type, print, erase and rename files
- For IBM PC, PCjr, PC compatibles, Sanyo MBC-55X, CP/M-80 or -86

30-day money-back guarantee on all products

ARTISOFT INC
2828 N. Country Club #105
Tucson, Arizona 85716
(602) 327-4305

Inquiry 37 for Dealers.
Inquiry 38 for End Users.