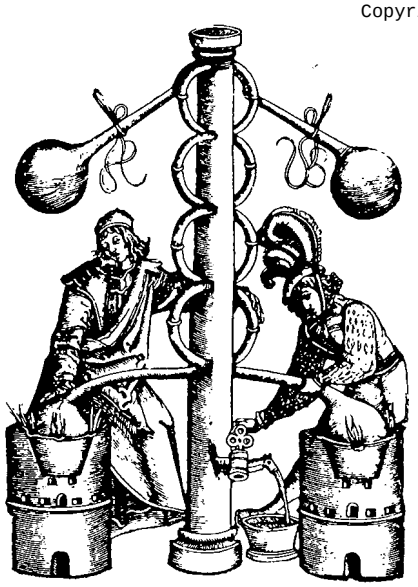

Microcomputer interfacing

Assembly language or BASIC

Which way to go?

By David G. Larsen, Peter R. Rony, Jonathan A. Titus, and Christopher A. Titus

As THE APPLICATION of microcomputers continues to grow, more and more users are asking what the best language is for programming the microcomputer. There are currently only two languages that have widespread support, assembly language and the BASIC* language.

In choosing a language, there are a number of advantages and disadvantages associated with each that must be considered. The two languages should be compared on the basis of cost of programming, memory requirements, machine independence of programs, program relocatability, users' libraries, logicaloperations, use of nonstandard peripherals, and speed of execution. This listing is not in order of importance. In fact, there are probably items of interest that have been omitted. For each particular microcomputer application, the user may be interested in only one of the items, such as memory (storage) requirements, while others may feel that all of the items are important and must be considered.

Cost of programming. The cost of programming a microcomputer in any language includes the cost of preparing flowcharts, entering the program into the microcomputer (using the assembly language editor or the BASIC interpreter), debugging the program, and producing the user's manuals, program listings with comments, and other documents that are needed to make the programs useful to others. Programs written in the BASIC language are generally much shorter than assembly language programs that have an equivalent function, so program entry costs should be much lower. Since one BASIC statement may be the equivalent of up to 50 or more assembly language instructions, debugging costs should be fewer, and program listings and comments shorter. Therefore, in terms of programming costs, a program written in the BASIC language should be less costly than the same program written in assembly language.

Memory requirements. Most BASIC programs available for microcomputers are in the form of interpreters, which means that the interpreter and the application program written in the BASIC language must be in the computer's memory at the same time. Available BASIC interpreters require between 6000 and 12,000 memory locations for storage. There are some BASIC interpreters available that require only 2000 memory locations, but these shorter interpreters lack many of the features required for data acquisition, control, or data processing. Along with the BASIC interpreters, the BASIC application program may require an additional 500 or 1000 memory locations for storage. A similar application program written in assembly language may require only 2000 or 3000 memory locations for storage. The important distinction between assembly language and BASIC at this point is that only 2000 to 3000 memory locations are required for the assembly language program, as opposed to a BASIC program, which requires from 6500 to 13,000 memory locations (the BASIC interpreter and the BASIC application program). As you can see, with a larger program, BASIC looks better, since the amount of memory required to store the BASIC interpreter becomes relatively smaller.

On the other hand, for short programs BASIC requires a great deal of memory. A short assem-

Mr. Larsen, Department of Chemistry, and Dr. Rony, Department of Chemical Engineering, are with the Virginia Polytechnic Institute and State University. Mr. J. Titus and Dr. C. Titus are with Tychon, Inc.

'BASIC is a registered trademark of the Trustees of Dartmouth College.

bly language program that requires only 500 memory locations might be equivalent to a BASIC program that requires 6500 memory locations (interpreter plus the application program). Memory costs money, consumes power, and may fail.

Machine independence. Many project managers are becoming increasingly wary of assembly language programs that cost $10,000 or $20,000 and can be used with only one type of microcomputer. For instance, an assembly language program written for the 8080 will not execute on a 6800-, 8008-, 6502-, or 9900-based computer. Assembly language programs are very computer dependent. However, programs written in the BASIC language can be independent of the actual computer on which they are run. That is, a BASIC program written on an 8080-based microcomputer will probably execute properly on a 6800-, 6502-, or 9900-based microcomputer. At the moment, however, application programs that are written in the BASIC language are not totally machine independent, since the BASIC interpreters that are available for the various microcomputers mayor may not have identical functions. Thus, some BASIC interpreters have a "log" function while others may not. If no log function were available, one could be added in about 25 BASIC steps. For the most part, it is easier to modify a BASIC program for a different microcomputer system than it is to modify an assembly language program for a different microcomputer system.

Program relocatability, As assembly language programs grow in complexity, data storage areas, stack areas, and subroutines often have to be moved about in memory, frequently requiring reassembly of the entire program. This is particularly true for an 8080 assembly language program, simply because the 8080 uses absolute addressing. For an assembly language program written for the 6800 or 6502, which uses relative addressing, this reassembly process may not be required, but is often recommended. For a BASIC application program, the data storage areas and stack areas are allocated as the program is interpreted by the BASIC interpreter. Therefore, the programmer does not have to worry about where data values or subroutine return addresses are stored. The BASIC interpreter remembers where all data values are stored. Variables, constants, and arrays can be added, changed, or deleted, and the BASIC interpreter keeps track of where they are stored.

Users' libraries. Many minicomputer programmers have resisted using a microcomputer, simply because there were no comparable users' libraries for microcomputers. What most programmers fail to realize is that many of the programs in these libraries, particularly the programs submitted by the users, are very difficult to use. Many of these programs require a special peripheral device, special peripheral interface, certain types of memory at specific addresses, or extended mathematical capabilities. Unfortunately, this is also true of many of the programs in the various microcomputer users' libraries. Programs are submitted to the libraries just to gain entry into the library, and the programs are not judged on their merits. Of course, if one program in the users' library saves $10,000 or $20,000, then the users' library is certainly worthwhile. However, do not depend on a program in a users' library until you have run it through its paces on your microcomputer system.

Logical operations. Most microcomputers can perform AND, OR, Exclusive-OR, and compare logical operations. However, many BASIC interpreters cannot perform these operations, which may be particularly important if your microcomputer system must be used to control a number of devices. If the microcomputer has to monitor 50 sensors and switches and control five pumps, 15 valves, and 32 lights based on the states of these sensors and switches, it may be easier to write the program in assembly language. Of course, logic operations can be performed in the BASIC language by using a sequence of mathematical operations. However, if this has to be done in the BASIC application program, the logical operations may take tens or hundreds of milliseconds.

Using nonstandard peripheral devices. If a nonstandard peripheral device is interfaced to the microcomputer system, then an assembly language subroutine (a device handler) can probably be written to control it. In all probability, a BASIC interpreter will not already include a device handler for this device. Instead, each BASIC application program must be written so that it can call an assembly language subroutine that operates the special device. Some BASIC interpreters have some very sophisticated features when it comes to "patching-in" or adding an assembly language subroutine to the BASIC application program; others do not. Chances are that if you do any interfacing yourself, you will have to write the assembly language subroutine for the peripheral device. If the BASIC language is used, then the BASIC interpreter must be able to access this device,

```
        *010 000
TEST1,  LXISP    ILOAD THE STACK POINTER WITH A
        STACK    IRIW MEMORY ADDRESS.
        O
        LXIH     ILOAD REGISTER PAIR H WITH
        350      /1,000 DECIMAL
        003
AGAIN,  CALL     /CALL THE READ SUBROUTINE SO THAT
        READ     I A CHARACTER CAN BE INPUT
        O
        DCXH     /DECREMENT THE COUNT
        MOVBA    ISAVE THE VALUE IN THE B REGISTER
        MOVAH    IGET THE M5BY OF THE COUNT
        ORAL     IOR IT WITH THE LSBY OF THE cotnr
        JNZ      ITHE cotsr I5 NON-ZERO, 50
        AGAIN    IGET ANOTHER DATA VALUE.
        O
        HLT      IREAD 1,000 VALUES, SO HALT

        IN       /INPUT THE STATUS WORD OF THE
        021      /PERIPHERAL DEVICE
        ANI      ISAVE ONLY ONE OF THE FLAGS
        004
        JZ       ITHE FLAG IS ZERO, SO WAIT FOR
        READ     IIT TO BECOME A LOGIC ONE
        O
        IN       ITHE FLAG IS A ONE, INPUT THE
        020      lEIGHT-BIT DATA VALUE
        RET      I AN0 THEN RETURN
```

Figure 1 *An assembly language that reads eight-bit values from a peripheral device.*

through the assembly language steps.

Speed of execution. The actual speed of program execution may be an important consideration. **In** general, the speed at which a BASIC application program can be executed is determined by the particular BASIC interpreter and computer used. An interpreter that uses integer numbers will be faster than one that uses floating-point numbers. A simple BASIC statement may require I or 2 msec in which to be executed, while the equivalent assembly language operation may require only 30 or 40 μsec. **If** the microcomputer must digitize an analog signal ten times each second and do a slight amount of data processing, then either assembly language or the BASIC language will probably work. **If** a signal must be digitized a thousand times each second, for long periods of time, then the BASIC interpreter will probably not be able to handle this data rate, while an assembly language program probably will.

Assembly language/BASIC benchmarks. As our two benchmarks, we will use admittedly simple programs. The first benchmark program simply reads an eight-bit data value from a peripheral device. The assembly language TEST1 program is listed in *Figure 1.* **In** this program, register pair H is loaded with a decimal count of 1000. The READ subroutine is then called. When the peripheral device's flag is a logic one, the 8080 inputs the eight-bit data value into the A register and then returns from the subroutine. When the 8080 does return, the count in register pair H is decremented and then checked to see if it is zero. **If** not, the 8080 jumps back to AGAIN so that another data value can be read from the peripheral

```
10 FOR I=1 TO 1000
20 LET A = CALL (15000)
30 NEXT
40 END
```

Figure 2 A *BASIC program that reads eight-bit values from* a *peripheral device.*

device. Note that the program does not do anything with the data value read from the peripheral device. This benchmark was written simply to measure the time required to read a value from a peripheral. We have assumed that the peripheral device is a fairly fast one. The equivalent BASIC program is listed in *Figure 2.*

Note that this example does not have any input/output instructions. This is because the BASIC interpreter was not written so that it could directly access the peripheral device. Therefore, the BASIC program accesses the peripheral device by calling an assembly language subroutine starting at memory location 15,000'0. This assembly language subroutine is very similar to the READ subroutine in Figure 1. The total elapsed time required by either benchmark is divided by 1000 to determine the time required to read a single eight-bit value. For the assembly language benchmark, the time required to read one data value was 46.5 usee. For the BASIC benchmark, 13 msec were required to read one data value. By using a different BASIC interpreter, this time might be cut in half.

The second benchmark performs an 11-point "moving average" on an array that contains 256 eight-bit data values. The assembly language program requires 165 memory locations for both the program and some temporary storage locations, and is not listed because of its length. The equivalent BASIC program is listed in *Figure* 3. The assembly language version of this benchmark required only 1.5 sec to smooth the 256 data values in the array, while the BASIC program required 105 sec. One notable difference between the two programs is that the BASIC interpreter uses floating-point math,

```
10 DIM A( 256)
20 FOR I=1 TO 256
30 LET A(I)-CALL(14000)
31 NEXT
32 LET I-I
33 FOR J-I TO 8
34 PRINT A( I),
35 LET 1-1+1
36 NEXT
37 PRINT " "
38 IF 1<257 THEN 33
45 PRINT "ARRAY FORMED"
50 DATA .074,.111,.182,.333,.666,1,.666,.333,.182,.111,.074
55 X-I
60 SaO
61 FOR I - I TO 11
70 READ C
80 LET S-S+(C*A(X»
90 LET X-X+I
100 NEXT
105 RESTORE
110 LET A(X-6)-S/3.732
120 LET X-X-10
130 IF X<247 THEN 60
135 PRINT "ALL VALUES ARE FILTERED"
140 LET 1=I
145 FOR J=I TO 8
150 PRINT %I%,A(I),
155 LET 1-1+I
160 NEXT
165 PRINT " "
170 IF 1<257 THEN 145
175 END
```

Figure 3 *A BASIC program that smooths* a *256-entry array.*

so the values in the array were smoothed as floating-point numbers. The assembly language program used integer math subroutines. The initial data values were eight-bit values (0.4070 accuracy), so there was really no reason to use floating-point numbers. However, our BASIC interpreter, along with most other BASIC interpreters, can only operate on floating-point numbers.

In conclusion, there are a number of points that must be examined when choosing between assembly language and the BASIC language. Both have their advantages and disadvantages. **In** fact, you have probably already thought of many points that we have not included here. Ideally, given the time and money, you should try to solve your particular problems using both assembly language and the BASIC language and choose the solution that will best meet your goals. However, few of us have the time or money to do this, so we have to compromise and use benchmarks to point us in the right direction.

In general, if memory costs do not bother you and relatively slow operation can be tolerated, then BASIC is probably the better choice. **If** the price of the product that is to use the microcomputer is to be very competitive, then assembly language may be better. **In** addition, assembly language gives you the opportunity to execute a program as fast as possible; you are not slowed down by an interpreter. As more programmers realize the advantages and disadvantages of both languages, the demand for an efficient compiler language (BASIC, FORTRAN, CONYERS, or C, etc.) will increase. **In** a few years, this may become the method by which a majority of microcomputers are programmed.