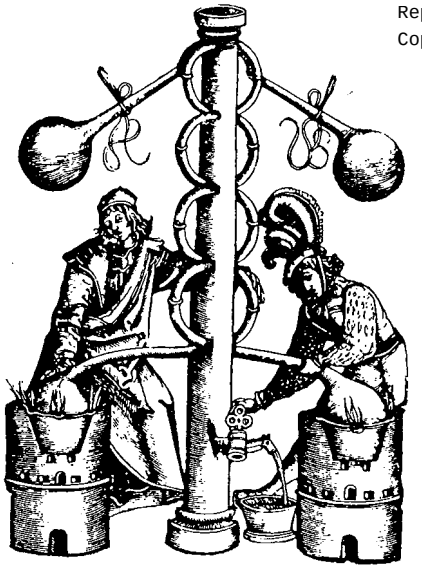
Reprinted with permission from American Laboratory, Vol. 9, No. 10, pp. 116, 118.



Microcomputer interfacing

Preparing programs

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

ONE of the problems facing many microcomputer users is the preparation of software for particular applications. The software examples provided in past columns are short enough to have been put together or *assembled* by hand, i.e., each mnemonic was translated into its octal, hexadecimal or binary equivalent. Addresses for jumps, calls and input!output devices are easily added or changed since the computer programs are short and the addresses are probably listed in sequential order on the rought draft. Unfortunately, not all software preparation is this easy. Many application programs can be many thousands of steps long. This column will initiate a discussion of the aids available for microcomputer program development.

One of the biggest problems in software development is the clear, concise statement of the problem and its solution. All of the desired results, inputs, outputs and the complete program flow-including all decision making steps-must be considered before the programming is started. This can be in outline or block diagram form, but a flow chart will prove much easier to follow. A typical flow chart is shown in *Figure 1.*

After the problem has been well thought out and a solution put in flow chart form, a decision must be made. Is the program short enough to be easily translated by hand? In many cases, particularly where the programs are simple, hand assembly makes sense. In other cases software development aids called *editors* and *assemblers* are faster and more efficient. To understand how editors and assemblers work, consider the process we use to put together this column.

The first step is an outline of the subject so that we can cover it well in the short column format. A handwritten copy is then typed, corrected, retyped, and perhaps corrected and typed a final time. The illustrations and examples are formulated and drawn separately. This is the *editing* process. When writing a column, it is best to avoid references such as "the example below" or "the table on the following page." When the column is composed or *assembled,* references to a specific table or figure are much easier to follow.

Computer software is developed in much the same way. An editor program is used, either on a microcomputer or a time-sharing system, to edit the individual program steps. The editor can correct program steps, change steps, or insert and delete steps, just as a secretary would do with

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

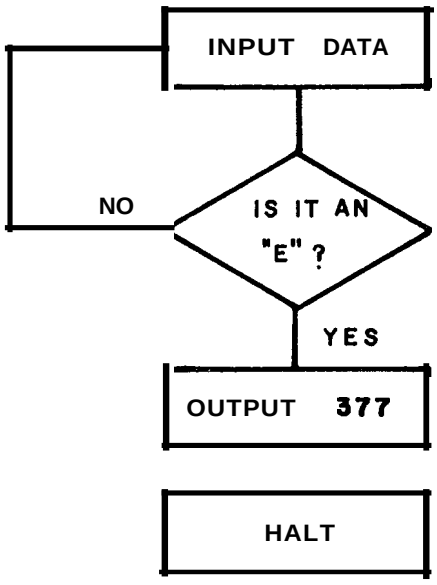


Figure 1

Editor	A program that allows edit functions such as addition of a line or character to a program, insertion, deletion, etc. It permits you to alter your program. The input data could be anything from programs or reports to raw instrument data.
Assembler	The program that converts the assembly language code into machine code, accepting mnemonics and symbolic addresses instead of actual binary values for addresses, instructions and data.
Monitor	A program that controls the operation of the various programs available. The monitor will be able to access the editor, assembler or other programs.
Debugger	A program that allows the user to observe the program flow and the results of the program's operation in a step-by-step mode. A debugger may be used to change data or instructions, alter registers, etc.
Breakpoint	A special instruction that may be inserted in a program to break off the normal program control and return control to a debug-type program. When a breakpoint is executed, the debug program will indicate what the computer was doing at that point.
Cross-assembler	An assembler program that will generate the binary code of a program for a different type of computer. For example, an 8080 cross-assembler might operate on a PDP-8 minicomputer.

a manuscript. The editor program is generally unaware that you are writing a computer program, since you can use most editors to write a letter, prepare mailing lists, etc. When using an editor to prepare a program in mnemonic form, *symbolic addresses* are often assigned to software tasks within the program. In this way the actual value of the addresses for subprograms or subroutines may refer to the letters, LOOP, as the starting address of a time delay loop. Allowing us to use symbolic addresses for program steps means that the program may be changed without regard to the actual numeric values of addresses.

The assembler program must be such that it accepts information from the editor and generates an output in a form compatible with your computer. Just as you assemble short programs a step at a time, so does the assembler. The assembler contains a table of mnemonics and their equivalent values. For example, an 8080 assembler would translate an MVIA instruction into 076 octal. The assembler also assigns real, 16-bit addresses to your symbolic addresses, such as LOOP. When using symbolic addresses you must be sure to have a program step for each symbolic address and you must assign an address if you use a symbol. You cannot assign the same "name" to more than one address. Most assemblers will recognize a *redefined symbol* or an *undefined symbol* and will produce an error message to let you know what needs to be corrected.

The final assembler output will be in punched paper tape, cassette or disk form ready to run on your system. Most assemblers will also produce a listing of the program showing the address of each step, the data in each successive location, a symbolic address name, and the mnemonic plus any comments. A typical assembler output is shown in *Table 1*.

After a program has been assembled, it will probably have to be debugged in order to operate properly. The program checkout and debugging can be painful without additional software "tools." Computer control panels often prove useful, but reading binary codes can become tedious, and many computers have no external controls and readouts. As an alternative, *debugging programs* are available for most microcomputers that allow you to change instructions, list blocks of data or instructions, and single-step through a program one instruction at a time.

One feature of many debug programs is the ability to establish a *breakpoint* in the software being tested. When the computer reaches a breakpoint, the instruction at that address is executed and an output device such as a teletypewriter lists the contents of important, internal CPU registers. Breakpoints are very useful since they indicate not only that the computer reached a certain point in the software, but what the computer was doing when it got there. If a breakpoint is set in the normal program flow and it is not reached, there is something wrong with the program. In this case, the breakpoint would be moved closer and closer to the start of the program until the error is found. When the error is found, it may be corrected by using the debug program to change an instruction, data, etc.

When the program is operating correctly, the debug program should have the means of saving it on paper tape, a cassette, or other medium. It should also be able to read such programs back into memory. In any case, when errors are found, you will probably want to re-edit and reassemble the software to produce a complete, error-free documented listing.

Since most programs will contain errors, it may be a good idea to have the debug program as a permanent part of your computer. It is prudent to store a debug-type program in read-only memory (ROM or PROM) since "run away" programs being tested might alter the debug software and necessitate its having to be loaded again. There are many debug or *monitor* programs available; Intel Corporation's Insite software library lists at least four. The editor/assembler programs may also be resident in PROM and the low cost of both read/write memory and PROM "chips" suggests that many users will keep standard system programs such as editors, assemblers and debug resident in their system. The alternative is a paper tape,cassette or disk-based software package that must be read into memory before each use.

Cross-assemblers will also generate an assembled program, but for some other computer. For example, a PDP-11 might be able to cross-assemble 8080 microcomputer programs. Cross-assemblers can be powerful programs since some incorporate simulation programs to test the program, too.

The program we use for testing programs is DBUG written by C.A. Titus, and the assembler output shown in our program examples is that produced by the Tychon Editor/Assembler (TEA).' Both are resident in our 8080 system on PROM chips.

Table 1

Software example showing a typical assembler **output**

```
                        *003 000
003 000 061   START.  LXISP   ISYMBOLIC ADDRESS OF START
003 001 377           377
003 002 000           000
003 003 333   LOOP,   IN      / INPUT DATA FROM PORT 5
003 004 005           005
003 005 376           CPI     ICOMPARE IT TO 026
003 006 026           026
003 007 312           JZ      11 F IT MATCHES GO TO -DETECT-
003 010 015           DETECT
003 011 003           0
003 012 303           JMP     IIF IT DOESN'T MATCH. GO TO
003 013 003           LOOP    /LOOP AND CHECK AGAIN
003 014 003           0
003 015 171   DETECT. MOVAC
003 016 323           OUT
003 017 007           007
003 020 166           HLT
```

References

I. TITUS, C.A., "DBUG, an 8080 interpretive debugger," (E&L Instruments, Inc., Derby, Conn., 1977).