The Small Computer Magazine

# kilobaud T.M.

Understandable for beginners . . . interesting for experts

May 1978 / Issue #17 / $2.00 / DM 7,50 / Sfr 8,10 / Ffr 16,0 / UK £2


| | | |
|---|---|---|
| *Ralph Wells* | 22 | **PET's First Report Card** . . . an objective evaluation |
| *Bob Buckman* | 32 | **Scope Power!** . . . a review of Tektronix's Model 922 |
| *J. Tom Badgett* | 36 | **Trials and Tribulations** . . . one businessman's micro blues |
| *Robert Baker* | 42 | **Writing Diagnostic Routines** . . . while your machine is running |
| *Dan Stogdill* | 44 | **Experiments in Software** . . . serial to parallel conversion |
| *Peter Stark* | 48 | **Computer Math Primer** . . . beginner's introduction to number systems |
| *George Young* | 54 | **Kilobaud Klassroom** . . . No. 10: Bus Traffic Control |
| *John Blankenship* | 60 | **Expand Your KIM** . . . Part 5: A/D interfacing (for joysticks!) |
| *Sheila Clarke* | 64 | **What's Happening with the IBM Selectric?** |
| *Dr. Lance A. Leventhal* | 68 | **The Top-Down Approach** . . . with some practical examples |
| *Howie DiBlasi* | 76 | **The North Star Floppy System** . . . an 11-year-old can build it! |
| *Stephen Gibson* | 78 | **A Simple Mailing System** . . . a money-making time-saver |
| *Dr. Adam Osborne* | 84 | **Number Crunching: Two Hardware Solutions** |
| *Ken Barbier* | 90 | **Money Manipulations** . . . keep ahead of those cash-flow problems |
| *Richard Roth* | 94 | **Strings and Things** . . . BASIC conversion techniques |
| *Thomas E. Doyle* | 100 | **5 Minutes or 5 Hours?** . . . sorting techniques compared |
| *Mike Kop* | 104 | **Do-It-Yourself Time-sharing** . . . it's easier than you think |
| *Dave Waterman, Dave Lien* | 110 | **Cassette Recorder Disaster: Ground Loops** |
| *Glen Charnock* | 112 | **A Different Search Technique** . . . don't just try it—benchmark it |

**Publisher's Remarks—4, Editor's Remarks—6, Around the Industry—6, TRS-80 Forum—8, Legal/Business Forum—12, KB Club Calendar—13, New Products—14, Books—16, Letters—17, Kilobaud Classified—114, Contest!—114, Calendar—116**

*Robert Baker*
*15 Windsor Drive*
*Atco NJ 08004*

# Writing Diagnostic Routines

## *while your machine is running*

Ever since personal computer systems first caught on, the area of diagnostic software has been overlooked. Few companies producing kits and hobby systems offer diagnostic programs to test and debug their products should a problem be suspected. Possibly such programs haven't been offered yet because of the degree of customizing each user performs while assembling his or her system. Because of competition in the market, most systems are a conglomeration of bits and pieces interconnected for a particular application. Thus, most hobbyists are forced to write their own diagnostic or maintenance programs (usually after a major problem develops) without really knowing what they're doing. It's very difficult to debug software if the hardware is not working properly!

On mini and larger size computers, the manufacturer usually provides various diagnostic programs to be run at regular intervals by the user as a form of preventive maintenance. The programs are written to detect minor faults before they degrade system operation, and to help isolate and debug major problems when they occur.

### Diagnostic Methods

One of two approaches is generally taken for writing and using diagnostic programs. A *bottom up* approach starts by testing the smallest entity in the system then using that proven-good device to test the next device in the system until all devices have been tested. Individual components are then tested in clusters or subsystems, and finally the entire system is tested, or exercised, as a whole.

A *top down* approach, on the other hand, starts by running a system exerciser to test all devices at once and isolate a problem to a given subsystem. More detailed, device-dependent programs are then run for the particular faulty device or subsystem to further isolate and help debug the problem. Once the problem has been corrected and the device-dependent tests are passed, the system exerciser can be run again to verify that that was the only system fault.

A bottom up approach requires the least amount of working hardware to be useful, but a top down approach takes less time to isolate a given system fault; so there are trade-offs. For either approach, the actual programs could be similar, depending on the system and the application.

### Writing A Test Program

Why not write a collection of test programs while you have a working system to try them on? Debug your programs thoroughly when writing them so you're sure that any problems detected are caused by hardware and not software. Try a bottom up approach first as this should make the programs easier to write. Start out with a few simple programs to check the CPU machine instructions, checking operands, condition codes, etc. Then check data paths to and from the CPU and the various control logic, trying different bit patterns to check for shorted lines. For convenience, you may want to create some of these programs in ROM and have them permanently available. Loading programs would require a major portion of the CPU to be in working order, so using ROM would eliminate that problem.

A quick memory check can verify that RAM memory is working correctly by writing all zeros and all ones to each location, reading it back, and comparing the data. You may want to check another memory pattern such as alternating ones and zeros (10101010) as well as checking memory addressing logic by insuring a test pattern had not been written into another location.

After the CPU and memory have been tested, you can then proceed to test any other devices you may have in your particular system. Test each device separately and thoroughly before going to the next. For starters, try a

program to test your CRT display or video terminal with:

- a character generator check, full lines of each character.
- display memory test (swirl pattern). The first line is a full character set. Each line after the first starts with the next letter in the character set after that used on the line above it. Therefore, each character will be on a diagonal, and will appear in each storage location as the test is run, and the display scrolls. For example:

```
abcdefg . . . . . .
bcdefg . . . . . .
cdefg . . . . . .
defg . . . . . .
```

Other patterns can be added to test special features, etc., depending on the particular display. To test a keyboard you can try a program that:

- displays on your terminal or CRT the code for the key depressed and the actual character.
- asks you to type each character in a set sequence and checks the code received.

Another useful test could print continuous lines of any character typed in by the user. When another is typed in, the printer would change to that character. Similar tests can be written to check the particular features or functions of other displays and terminals. Use easy-to-recognize patterns on printers or terminals and keep the tests simple!

If you have tape drives, cassettes, floppy drives, optical readers or joysticks, don't forget to test them also. Test every device in your system thoroughly, one at a time. Later you can add a simple exerciser to get everything working at once and check for device interactions. My advice is to save this for later when you start to get a feel for what you really want to accomplish and how you want to control your test programs.

### Make It Useful

For whatever devices are being tested, certain basic features should be included for your own convenience and increased usefulness of the programs. Each test within a program should give a clearly defined indication when an error is detected. This can be an error message on a terminal or simply a machine stop at a specific address. If error stops are used, separate halts or stops should be used for each possible error so the address at which the machine stops will indicate *what* error was detected. For added convenience you may even want to generate an error dictionary to list and describe each possible error halt and give some possible causes or cures. Also, keep a log of the errors and causes detected by your programs for later reference. They may save you from debugging the same problem again several months or years later.

Another desirable feature should be the capability to loop on any test for scoping purposes, possibly with a sync pulse generated at the start of each pass. Other features can be added as desired, depending on your particular system and how you want to test it. Try to keep it simple but flexible, and easy to use.

Plan ahead and prepare yourself for the inevitable. Sooner or later you're bound to have a hardware problem, and your local TV repairman will probably not be able to help you, let alone know what you're talking about. An even better idea: Run the tests periodically and catch small problems before they become major ones. Your time spent writing diagnostic programs for your system now will be repaid many times in the future. ■