

# PASSING THE LAB TEST

*After a year in a physiology laboratory, a PC gets a strong endorsement*

PETER AITKEN, PH.D.



Because most scientific data can be represented numerically, digital computers have long performed a valuable role in scientific research, assisting in the analysis of experimental data. The rapid improvements in the performance, size, and price of computers have dramatically affected their laboratory applications. Below is a discussion of applications for the small computer drawn from the selection and use of an IBM PC for data collection and analysis in the Integrative Neurophysiology Laboratory at the Duke University Medical Center.

## **THE OLD DAYS**

Not long ago, when the only available computers were bulky and expensive, computing power was centralized and shared. A university would, for example, have one large mainframe, possibly with a number of remote terminals, that was shared by all personnel—administration and faculty researchers. Data analysis was off-line, that is, measurements taken

---

*Peter Aitken is a research associate in the Department of Physiology at the Duke University Medical Center, Durham, NC. His PC-aided research involves the use of animal models for the study of epilepsy.*

# LAB TEST

in the laboratory would first be converted, if necessary, into numerical form to be entered into the computer at a later date—a tedious and time-consuming task. Even so, computer analysis was widely used and represented a great time savings.

If the analog data could be converted automatically to numerical form and then fed directly into the computer, efficiency would greatly improve. Enter analog-to-digital (a/d) converters, which convert an analog signal (voltage or current) to a binary numerical form and thus make laboratory measurements directly available to the computer. Although this method is powerful, it is limited in application by the size and cost of computers, preventing all but well-to-do laboratories from dedicating a computer to do one task.

When relatively small yet powerful computers became available at prices small and medium-sized laboratories could afford, major changes in laboratory uses of computers began to occur. Equally important—if not more so—was the introduction of computer systems with hardware and software specifically designed for laboratory use so that a scientist could exploit the computer's power without being or hiring a computer expert.

Perhaps the best known early example of such a system is Digital Equipment Corporation's PDP-8/Lab series introduced about 12 years ago. It provided laboratory data collection and analysis capabilities at a price ranging from \$10,000 to \$20,000. By today's standards, of course, the price/performance ratio was high and the software cumbersome and limited. But at the time it seemed awesome.

The past few years have seen an explosion in laboratory-oriented computers and accessories. Indeed, the number of products and the number of companies marketing them make selecting a computer system for a laboratory a formidable task, as I found out when asked to select one for the Duke Neurophysiology Laboratory. Preliminary investigations revealed

three fairly distinct categories of available systems, covering a wide range of prices, capabilities, and convenience factors. It quickly became clear that there is not always a direct relationship between price and capability; thus, buyers should have a clear idea of their needs and should

**When relatively small yet powerful computers became available at prices small and medium-sized laboratories could afford, major changes in laboratory uses of computers began to occur. Equally important—if not more so—was the introduction of computer systems with hardware and software designed for laboratory use.**

consider the amount of programming they are willing to do in order to select a system that will do the most for the least cost.

## OUT THERE

The most sophisticated (and expensive) laboratory computer systems are those whose hardware and software have been designed specifically as an integrated package for laboratory use. In this category fall DEC's NINC System, Laboratory Technology Corporation's LABTECH 70, and Hewlett-Packard's 9836. These are powerful, 16-bit systems with integrated interfacing components and data acquisition and analysis software.

Next come integrated data acquisition systems, which are designed to interface with personal computers. These systems, such as Cyborg Corporation's ISAAC for the Apple-II and the Series 500 from Data Acquisition Systems for the IBM-PC and Apple, consist of an external cabinet or con-

sole that connects by cable to one of the computer's slots, the console contains a/d and d/a converters, clocks, Schmidt triggers, and other interfacing hardware. To some extent these systems can be customized to meet the specific needs of the user. They come equipped with a package of software routines for data collection and analysis, instrument control, and graphics. While I have not used the systems in this category, their published specifications compare favorably with those of the fully integrated systems like MINC, usually at about half the price.

The least expensive route to computer data collection is to add a single-board, analog interface to a personal computer and write a program for it. For those not afraid of getting involved with hardware or of writing the necessary software, this approach can result in a powerful system obtained at a surprisingly reasonable price. This is the approach used at the Duke Physiology Laboratory.

## LABORATORY PC REQUIREMENTS

The research projects currently underway here involve testing animals in an effort to advance the understanding of certain diseases of the nervous system such as epilepsy, in order to find more effective medical treatments. Data collection needs vary considerably; at times it is necessary to sample one or two electrical signals rapidly for a short period—1,000 samples over one-tenth of a second. Other experiments require that as many as six or eight signals be sampled slowly, perhaps once per second, over a period of several hours. Further requirements were d/a capacity in order to permit the computer to control external instruments; several high-resolution internal clocks for timing functions; digital input/output lines for further control and sensing; and graphics. These requirements, common to most laboratory situations, would have been well met by any system in the first two catego-

## LAB TEST

ries above. However, it soon became apparent that even the least expensive of these systems was beyond the laboratory's budget.

Because the necessary programming skills were available on our staff, we opted for a system from the third category, a personal board analog interfacing system.

### WHICH PERSONAL COMPUTER?

Considerations of hardware and software availability quickly narrowed the field to two choices: the Apple II+ (the Apple IIe had not yet been introduced) and the IBM PC. At first, the Apple seemed the logical choice because several other nearby laboratories were already successfully using Apples in applications similar to the ones we needed. Thus, not only had the Apple been shown to be capable of performing the tasks we needed, but there was a library of software from which to draw and an informal users group with whom to talk.

The more I talked with people who were using the Apple, the more apparent became their dissatisfaction with the machine's inherent limitations. Although many complaints dealt with the Apple's processing speed, gripes focused more commonly on its memory limitations. With only 64K for operating system, program, and data files, some users found that their data collection and analysis abilities were severely limited. The IBM's vastly larger memory capacity and faster processing speed began to seem all the more appealing.

### HARDWARE ADDITIONS

The next step was to find an analog interfacing board for the PC that would meet the laboratory's needs. Without an appropriate board, the PC's greater speed and memory would be all but lost. At the time of our search (fall, 1982), only one firm, Tecmar, was marketing such boards. Fortunately, one of their models, the Labmaster, offered all the features we needed: 16 to 12 bit a/d channels

**C**onsiderations of hardware and software availability quickly narrowed the field to two choices: the Apple II+ (the Apple IIe had not yet been introduced) and the IBM PC.

(with optional expansion to 256 channels, 5 clocks/counters, 24 digital input/output lines, and a/d conversion speeds up to 40 K Hz. An IBM PC with the Labmaster board seemed to fill the lab's current needs, and it offered future potential. (In addition to the Labmaster, Tecmar also has the Labtender, which offers many fewer features than the Labmaster at a significantly lower price. Data Translation also recently introduced its DT2801 I/O board for the PC, which is similar in capabilities to the Labmaster. Both Tecmar and Data Translation also offer, at additional cost, packages of software routines to implement the functions of the boards.)

Only two software statements are required to program the Labmaster; in BASIC these are the INP and OUT statements. The IBM PC has 65,535 input/output ports, 16 of them are used by Labmaster. Some of these ports, called control parts, are used to send information to Labmaster; control parts, as the name implied, serve to control the board's functions. Some ports are used to receive information from the Labmaster, such as the board's current status or the results of the last a/d conversion.

A major reason for choosing the PC was its memory capacity. For additional memory beyond 64K on the system board, we chose an AST Research MegaPlus board with 256K. Besides memory, this board provides a parallel printer port, a serial port, and a clock/calendar with battery back-up. We use a black and white

monitor, but decided on the IBM Color Graphics board for its graphics capabilities. Two 320K disk drives and an Epson MX-80 printer (with Graftrax to allow printing of graphics), completed the hardware.

### PROGRAMMING LIMITATIONS

Our initial software acquisitions were fairly straightforward—PC DOS with BASIC, the IBM Macro assembler, and a few utilities for functions such as print spooling, RAM disk emulation, and graphics screen dump to the printer. Because I was already familiar with BASIC, I chose that as my programming language, with assembly language subroutines when more speed was necessary.

In the course of writing our first data collection programs, however, the limitations of interpreted BASIC began to surface. Processing speed leaves much to be desired, of course, but of greater concern was the difficulty of interfacing assembly language subroutines to BASIC programs. This problem seemed insoluble until I learned that early versions of the IBM BASIC manual contained incorrect directions for BASIC-assembly language interfacing. Once the correct directions were in hand, the interfacing process worked, but the procedure was still cumbersome.

The other limitation, even more serious, was the BASIC interpreter's 64K maximum size for program plus data. This limitation negated most of the advantage of IBM's greater memory size, and, as my programs grew in complexity, it began to place major constraints on our work.

All of these problems were solved with the purchase of an IBM BASIC compiler. This is a powerful and useful piece of software that, in my view, belongs in the library of anyone using BASIC for serious programming. Although this article is not the place for a detailed description of the compiler, several of its features that relate directly to our problems deserve mention.

AT  
AFFC  
Data A  
Co  
IBM Pe

DA  
The Price

Also  
Accounting

# LAB TEST

First, compiled BASIC runs much faster than interpreted, by a factor of approximately 10. Second, interfacing BASIC and assembly language programs is much simpler because it is done automatically by the linker rather than through the tedious application of the DEBUG program, as when using interpreted BASIC.


Third, compiled BASIC provides 64K for the program plus another 64K for data arrays—a doubling of available memory. Fourth, the compiler allows the omission of line numbers from all program statements except those referred to by GOTOs, GOSUBs, and other branch instructions. This allows for more compact and easily readable source code, as shown in the accompanying BASIC program listings 1, 2, and 3, which follow this article.

Listing 1, ASSEMBLY.TXT, is a complete listing of an 8088 assembly language subroutine named

DSCOPE1, which, when called by a BASIC program, collects and averages sweeps of analog data on Labmaster channel 0; the collection sweeps are triggered by pulses on Labmaster channel 7. In general, this program component is used for fast sampling of short-duration signals. In our lab, it is used to collect and average responses of brain cells that have been triggered by electrical stimulation.

Listings 2 and 3, BASIC1.TXT and BASIC2.TXT, are fragments of a large data collection and analysis program, written in BASIC for the IBM BASIC compiler. BASIC1 is the data collection part of the program, which works in conjunction with the ASSEMBLY.TXT. It sets up a clock on the Labmaster board to run at the desired sampling rate and also passes parameters—such as the address of the array in which the data are to be stored and the number of sweeps to average—to the assembly language

subroutine, DSCOPE.1. BASIC2 permits already-collected waveforms to be plotted on an X-Y plotter. It converts data into appropriate X and Y values and then sends the commands to the digital-to-analog converters on the Labmaster board.

The PC for our lab has served us well. For an investment of approximately \$6,000, we have a data collection and analysis system whose capabilities rival those of systems costing two and three times as much. It has met every need so far, and we have not come close to exploiting all its capacity. During a year of heavy use we have experienced no hardware failures and, with the exception of the problem with the BASIC interpreter mentioned above, no software bugs. This system should be seriously considered by anyone needing a laboratory computer system. 

## LISTING 1 ASSEMBLY.TXT

```

; This program collects and averages sweeps of analog data from
; channel 0 of the labmaster board, with each sweep triggered
; by a pulse on channel 7. It requires that the calling program set
; up clock 5 to output pulses at the desired sampling rate, and
; that the Labmaster be jumpered for external start a/d conversions.
; The calling program must also pass two parameters: the address of the
; array for data storage, and the number of sweeps to be averaged.
; Program now collects 640 points/sweep. Subroutine "beep" beeps the
; speaker after each sweep. Maximum sampling rate=40kHz.

sseg      segment stack      ;set up stack.
          dw      20 DUP (?)

sseg      ends

dseg      segment           ;set up data segment.
          dw      (?)
          dw      (?)
          ends

cseg      segment public 'CODE' ;code segment.
          assume cs:cseg,ss:sseg,ds:dseg

          public dscopel     ;declare "dscopel" public so it can
                              ;be called from another program.

dscopel   proc far
          push bp             ;save register.
          mov  bp,sp
          mov  si,[bp]+8      ;get first parameter passed by
                              ;calling program.

          mov  dx,[si]
          mov  array,dx      ;data array address in "array".
          mov  si,[bp]+6     ;get second parameter passed.
          mov  dx,[si]
          mov  nsmps,dx      ;# sweeps in nsmps.
          inc  nsmps         ;now nsmps=#sweeps+1.

;now we go thru the array and zero all elements.

          mov  bx,array      ;bx points at base of array.
          mov  si,0          ;si indexes array element.
          mov  cx,640        ;cx counts array elements.

```

```

          mov  ax,0
zero_loop: mov  [bx][si],ax    ;move "0" to array element.
          add  si,2          ;point at next element.
          loop zero_loop     ;loop back if cx<0.

;now we wait for a synch pulse on a/d channel 7.

synch_loop: nop
          dec  nsmps         ;nsmps now = # sweeps remaining.
          jz   done          ;if it's 0, we're done.

wait_synch: mov  dx,0714H    ;point at control byte.
          mov  al,10000000B  ;disable autoincrement and
          out  dx,al         ;all interrupts.
          inc  dx            ;point at a/d channel byte and
          mov  al,7          ;specify that next channel
          out  dx,al         ;to convert is #7.
          inc  dx            ;point at a/d start & hi data.
          in   al,dx         ;read high data to reset
                              ;done bit.
          out  dx,al         ;start a conversion.
          mov  dx,0714H     ;point dx at status byte
wait_done:  in   al,dx       ;and read it in.
          cmp  al,10000000B  ;see if bit 7 is set.
          jb   wait_done     ;if not, look again.
          inc  dx            ;point at low data byte.
          in   ax,dx         ;read ch#7 voltage.
          cmp  ax,1024       ;is it > "1024"?
          j!   wait_synch    ;if not, try again.

;now that a synch pulse has been received, we can collect data

get_data:  mov  cx,640      ;cx will count loops.
          mov  si,0         ;si=array offset pointer.
          mov  bx,array     ;bx=array base pointer.
          mov  dx,0714H    ;point at control byte.
          mov  al,10000100B ;enable external starts and
          out  dx,al         ;disable autoincrementing.
          inc  dx            ;point dx at 0715H.
          in   ax,dx         ;read data to clear "done"bit.
          mov  al,0
          out  dx,al         ;specify a/d channel #0.
data_loop: dec  dx          ;point at status byte
in_loop:  in   al,dx        ;and get it.

```