Laboratory Data Collection with an IBM PC

A versatile hardware/software combination

Stephen C. Gates
Illinois State University

You have a new IBM Personal Computer (PC) and you want to use it in the laboratory to collect data from a scientific instrument. How do you do that with a small investment of time on your part and still get a product that is a powerful, useful tool in the laboratory?

I faced that same problem almost two years ago when our chemistry department received its first IBM PC, and I wanted to interface it to a variety of chemical laboratory instruments. We had only one of each type of instrument, so I was faced with the possibility of designing a custom interface for each of 10 or more instruments.

Fortunately, I had interfaced single instruments to a DEC LSI 11/23 and to an Apple, so I knew from my own previous mistakes that a little advanced planning would make this a much simpler project. Specifically, I realized that interfacing can be made much easier by using two simple concepts: first, buy commercially available hardware where possible, and, second, develop general-purpose software that can be used for almost any instrument.

By utilizing these two concepts, I found that even undergraduate chemistry students with little previous computer experience can produce research-quality interfaces, with complete software, in less than one week. If you follow the suggestions provided here, you should be able to design and implement an interface to the instrument of your choice in less time. All you need to do is be able to program in BASIC, FORTRAN, or some other language that allows the use of assembly-language subroutines.

The essential elements of this system are a commercially available data-collection board that fits in one of the slots of the IBM PC, a preamplifier and filter for conditioning the signal from the instrument, and a set of BASIC and assembly-language routines to perform tasks common to all of the instruments to be interfaced.

The utility of this approach arose fairly naturally from some initial design decisions. My major criteria for selection of equipment and software were ease of development and ease of use. Therefore, I judged it to be not cost-effective to spend time developing special-purpose A/D (analog-to-digital) converters, timers, or other equipment. Similarly, I chose to use BASIC for all purposes except the data-collection process itself because of the ease of programming, even for novices; when the programs are completely tested, they are converted to compiled BASIC to greatly increase their execution speed.

In order to encourage a variety of users, I put the (now several) IBM PCs on carts so that the computers can be wheeled from experiment to experiment. Each cart contains a 64K- or 128K-byte IBM PC with a color-graphics monitor adapter and green monitor; dual 320K-byte disk drives; a combination board containing an A/D converter, D/A (digital-to-analog) converter and programmable clock; and a preamplifier and filter combination. A typical system in use is shown in photo 1.

Each of the components on the cart is designed to accommodate interfaces to a variety of instruments. If you are attempting to develop a similar system, it may help to have a description of why I selected each component.

Data-Acquisition Board

Several different manufacturers now market general-purpose data-

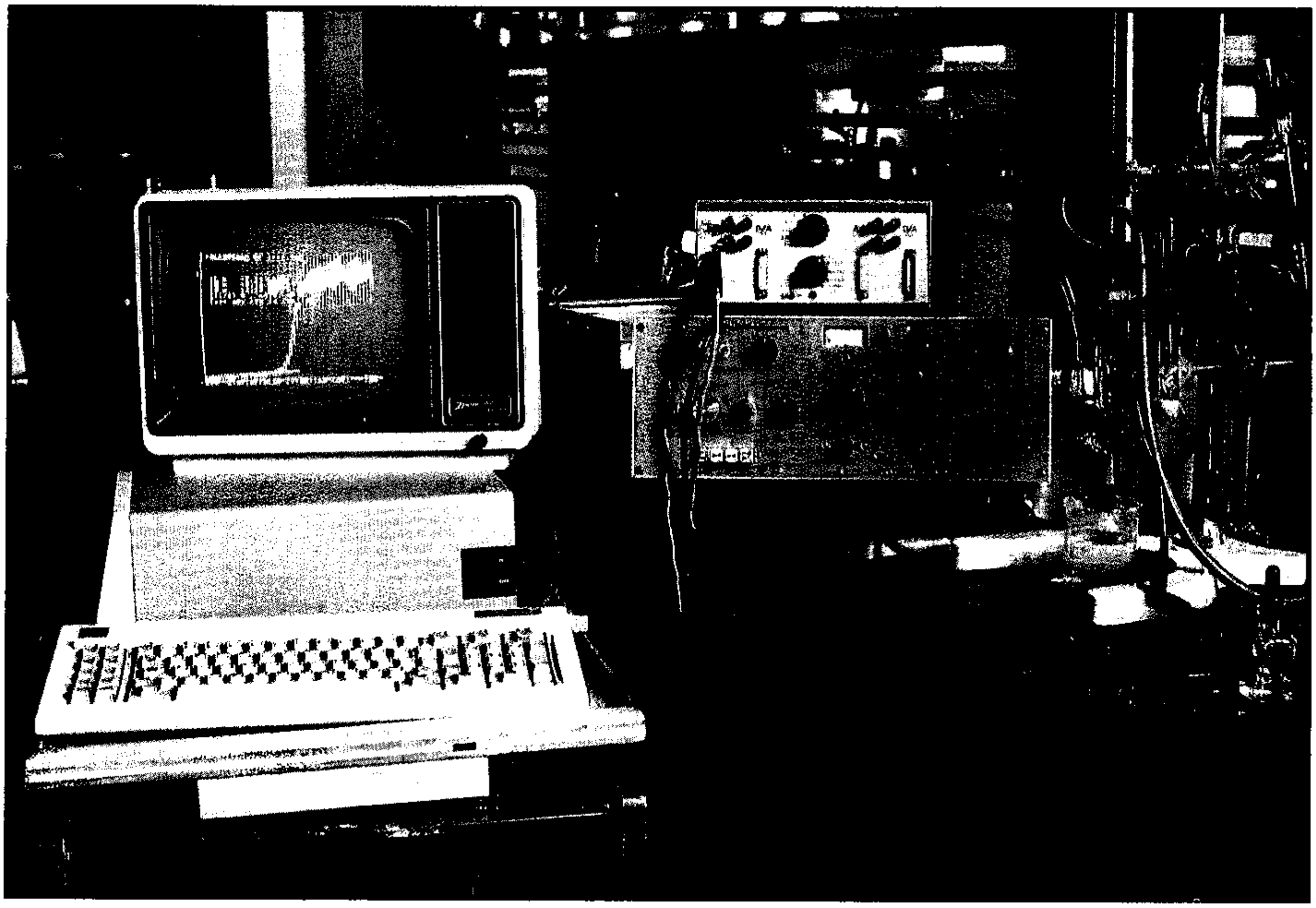


Photo 1: *The general-purpose laboratory interface station can be moved easily from instrument to instrument because it is on a laboratory cart. The IBM PC contains a color/graphics monitor board and a Tecmar Lab Master interface board. The preamplifier box is perched on top of the larger control device for the polarograph, in the center. The electrodes for the polarograph are at the right side of the photo.*

acquisition boards (see reference 4). These usually include a multichannel A/D converter, one or more D/A converters, and a programmable clock as standard features, with options such as programmable gain, higher acquisition rates, and DMA (direct memory access). For most scientific applications, a 12-bit A/D conversion is necessary; 8-bit A/D converters simply do not provide adequate resolution.

In addition, most laboratories now use nonintegrating A/D converters rather than integrating types because of the slow speed of the latter. The primary advantage of the integrating A/D converter is the reduction of noise; however, this can be accomplished instead through appropriate software used with the nonintegrating type. The A/D converters on almost all of the general-purpose data-collection boards now available are of the nonintegrating type.

While not essential, a programmable clock is highly recommended. Although timing can be controlled by carefully timed program loops, usually in assembly language, it is much more easily and accurately achieved in hardware.

For these reasons, I chose to use a Tecmar (6225 Cochran Rd., Cleveland, OH 44139, (216) 349-0600) PC-Mate Lab Master board with a 16-channel, 12-bit nonintegrating A/D converter with no programmable gain and a general-purpose clock/timer. The board also contains two D/A converters and a digital I/O (input/output) section that I do not routinely use, but which you may need if you plan to control the operation of your instrument as well as collect data from it.

Connecting the Interface

In order to use the hardware interface in your lab, you must first connect the interface to the instrument. If the instrument has a recorder output, this is very easy to do; simply connect the A/D converter input to the recorder output wires. For signals below 1 volt maximum, the preamplifier should be interposed between the A/D converter and the instrument.

Often, particularly on more recently designed instruments, both a recorder output and a BCD (binary-coded decimal) or other computer-compatible output exist. If there is a computer output, no A/D converter is needed; instead, a digital I/O board, serial interface, or other hardware is required. Unfortunately, I found that the documentation provided by Tecmar on the digital I/O section of the interface board is almost no help to those who are not already familiar with this type of hardware.

Alternatively, if no suitable output

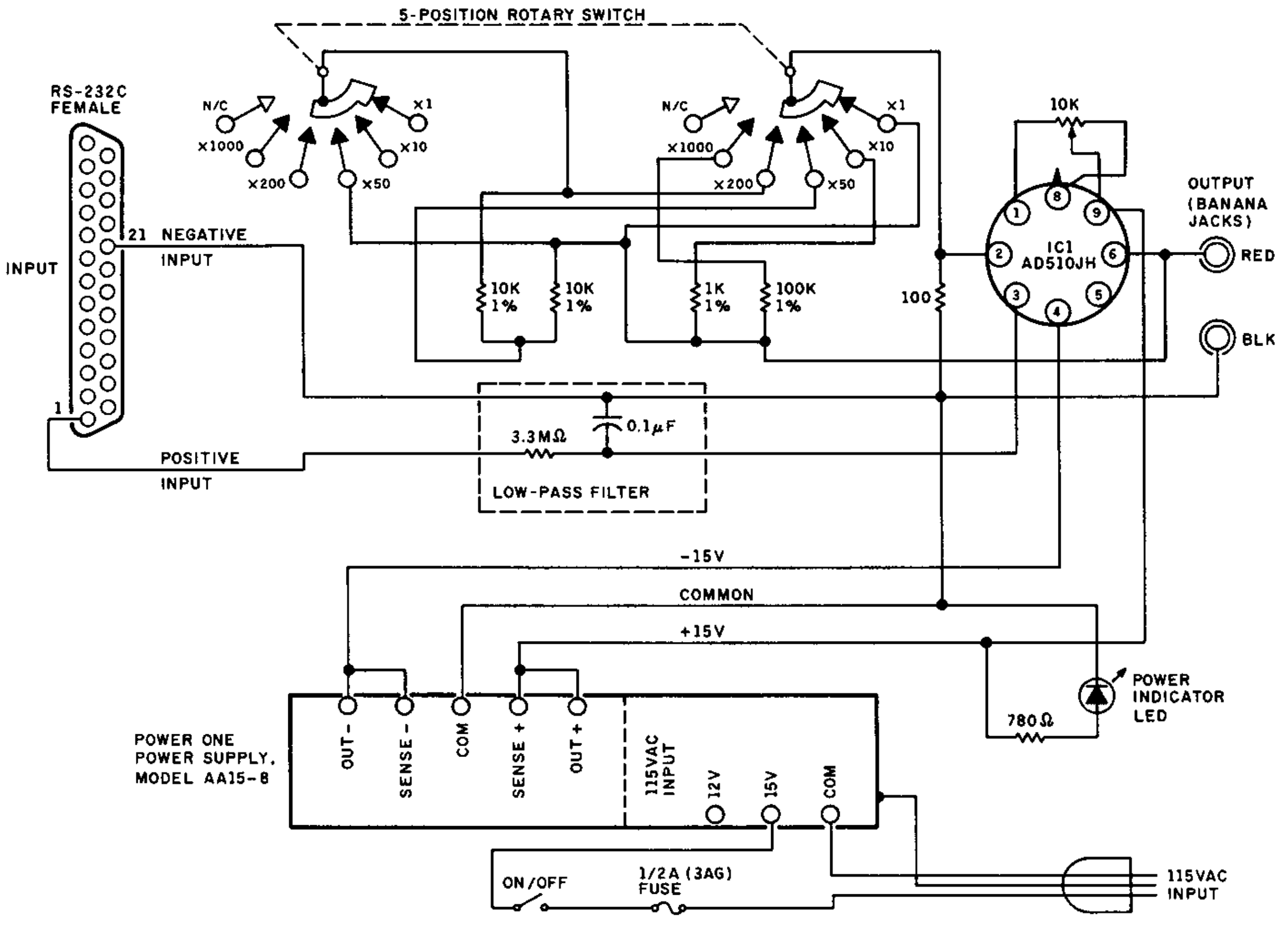


Figure 1: *The schematic diagram for the preamplifier described in the text. The low-pass filter is optional.*

is provided, it may be necessary for someone with knowledge of the electronics of the instrument to locate for you the portion of the circuitry needed to provide a suitable voltage output to the A/D converter. Where possible, this voltage should be in the volt range, rather than in millivolts (mV) or microvolts (μV). Fortunately, most instruments have recorder outputs and consequently are very easy to interface.

Preamplifier

Depending upon the instrument being interfaced and the A/D board being used, varying amounts of preamplification are needed. I designed our system to accommodate a wide variety of possible inputs; hence, a simple amplifier circuit was included to permit five different gains between 1 and 1000. The amplifier schematic is shown in figure 1.

Alternatively, a programmable-gain A/D board may be desirable, although that option is usually much more expensive than a separate amplifier. There is another reason for separate preamplifiers, however. Instruments with full-scale outputs of under 10 mV are common in scientific laboratories because of the widespread availability of 10-mV strip-chart recorders. For these instruments, your best alternative is to build the preamplifier into the instrument itself, or at least to connect it so that it is as near as possible to the instrument. This reduces the amount of noise picked up by the low-level signal lines that, in effect, act as antennas to the various sources of electronic noise in the environment. In general, the shorter the distance between the instrument and the A/D board, the better the signal-to-noise ratio will be in the final data.

Filter

The most general solution to noisy signals is software filtering, because the filter can be varied to best match the noise level. However, particularly for low-level signals and low data-collection rates, e.g., 1-mV signals at 60 Hz (hertz), I have found it useful to have a hardware filter because of the large amount of computation time required for extensive software-based filtering. For such instruments, I use the simple, passive, low-pass filter included in figure 1. This filter has a cutoff frequency of approximately 0.5 Hz, which is adequate for filtering out the most common noise signals that are 60 Hz or higher in frequency. More expensive filters, including active and notch filters, may be desirable for specific applications. Almost any "electronics for scientists" text can be consulted for more details.

Data Collection

One aspect of interfacing that texts often neglect is the need for general-purpose programs to collect, plot, and process the instrumented data.

However, by having a suitable library of general-purpose routines, you can shorten the development time for your specific interface considerably. By using the general-purpose data-collection, smoothing, and display routines described here, you can concentrate all of your efforts on developing the device-specific portion of the software and end up with a higher quality product in a much shorter time than if you "reinvent the chip" for each new interfacing project.

In order to provide high data-collection rates and a real-time plot, I wrote a data-acquisition routine in assembly language. The routine illustrated in listing 1 provides rates up to 2400 Hz with a real-time plot, and up to 20 kHz without plotting. Even faster rates are possible with special hardware settings of the standard Tecmar board, and rates up to 125 kHz are available as an optional feature. However, very few instruments will require higher rates than 20 kHz.

The routine in listing 1 assumes the use of the Tecmar Lab Master data-acquisition board, so that some of the code is device-specific and would need to be modified for use on other systems.

Although the listing is fully documented, several comments are required. First, using the excellent procedure suggested by Rollins (see reference 2), the routine begins with a header section to enable it to be converted by EXE2BIN to a binary file that can be loaded into memory with a BASIC BLOAD command. Second, high-resolution plotting is done using the BIOS VIDEO_IO routine, which is invoked with interrupt 16 (10 hexadecimal).

Three different clock rates are used, depending upon the desired data-collection rate. This is done to achieve maximum precision. For high data rates, the 1 MHz clock in the Tecmar board is used directly. For rates below 31 Hz, a 10-kHz subfrequency of the clock is used; to use the 1-MHz clock directly would require chaining several of the counters together. Rates of less than 1 Hz are counted with a 100-Hz subfrequency.

At very high data rates, it is possible that a conversion may take place

Listing 1: *An assembly-language data-collection routine for use with the IBM PC and the Tecmar Lab Master board.*

```
                TITLE   TIMER
;       S.C.\GATES  DEPARTMENT OF CHEMISTRY, ILLINOIS STATE
;                UNIVERSITY, NORMAL, IL   61761
;       SUBROUTINE TO DO TIMED DATA COLLECTION FROM TECMAR BOARD
;       CALL FROM BASIC WITH CALL OF FORM:
;               CALL TIMER (A%(1),F%,P%,N%,C%,S%)
;       WHERE   A% IS ARRAY WHERE DATA ARE TO BE STORED
;               F% IS OVERRUN FLAG--SET TO ZERO UPON NORMAL EXIT
;                  OTHERWISE SET TO VALUE OF CX REGISTER TO GIVE
;                  NUMBER OF POINTS NOT COLLECTED
;               P% IS 0 TO OMIT REAL-TIME PLOT, OTHER TO PLOT
;               N% IS NUMBER OF POINTS TO BE COLLECTED
;               C% IS CHANNEL NUMBER OF A/D
;               S% IS NUMBER OF DATA POINTS PER SECOND
;                  S% MUST BE <= SPEED OF A/D
;                  IF S% <0 THEN MEANS WANT THAT MANY SEC/POINT
;
CSEG    SEGMENT
        ASSUME CS:CSEG,  DS:NOTHING
HEADER:
        DB      0FDH                ;CODE FOR BLOAD FILE
        DW      0
        DW      0
        DW      RTN_LEN
TEMP    DW      ?                   ;TEMP. STORAGE
PLOT    DW      ?                   ;PLOT FLAG
TEMPSI  DW      ?                   ;TEMP STORAGE FOR SI REGISTER
OVRUN   DW      ?                   ;OVERRUN OF A/D FLAG
;DEFINITIONS:
ADD0    =1808                       ;BASE OF TECMAR BOARD
ADD4    =ADD0+4                      ;A/D CONTROL BYTE
ADD5    =ADD0+5                      ;A/D CHANNEL NUMBER
ADD6    =ADD0+6                      ;A/D START
ADD8    =ADD0+8                      ;CLOCK DATA PORT
ADD9    =ADD0+9                      ;CLOCK CONTROL PORT
;
TIMER   PROC    FAR
        PUSH    BP                  ;SAVE BP
        MOV     BP,SP               ;SET BASE PARAMETER LIST
        MOV     DI,[BP]+6           ;GET DATA POINTS/SEC
        MOV     AX,[DI]             ;   INTO BX REGISTER
        MOV     BX,AX
        MOV     DI,[BP]+8           ;GET CHANNEL NUMBER
        MOV     AX,[DI]             ;   AND STORE AS AX
        MOV     DX,ADD5             ;   AND OUTPUT TO A/D
        OUT     DX,AL               ; (USE ONLY LOWER BYTE)
        MOV     DI,[BP]+10          ;GET NUMBER OF DATA POINTS
        MOV     CX,[DI]             ;  STORE IN CX REGISTER
        MOV     DI,[BP]+12          ;GET PLOT FLAG
        MOV     AX,[DI]             ;  STORE IN MEMORY
        MOV     PLOT,AX
        MOV     AL,128              ;SELECT A/D MODE (DISABLE AUTOINCREMENT,
        MOV     DX,ADD4             ;   EXTERN. START CONVERSION, ALL INTERRUPTS
        OUT     DX,AL               ;   GAIN=1)
        MOV     AX,0                ;SI IS X-VALUE OF POINT TO BE
        MOV     TEMPSI,AX           ;  PLOTTED--SAVE FOR LATER
        MOV     AX,6                ;SET UP HIGH-RES GRAPHICS MODE
        INT     10H
        MOV     DX,ADD6             ;RESET DONE FLIP-FLOP OF A/D
        IN      AL,DX
        MOV     DX,ADD9             ;SET DATA POINTER TO MASTER MODE REGISTER
        MOV     AL,23
        OUT     DX,AL
        MOV     DX,ADD8             ;SET MASTER MODE REGISTER FOR SCALER CONTROL=
        MOV     AL,0                ;   BCD DIVISION, ENABLE INCREMENT, 8-BIT BUS,
        OUT     DX,AL               ;   FOUT ON, DIVIDE BY 16, SOURCE=F1,
        MOV     AL,128              ;   COMPARATORS DISABLED, TOD DISABLED
        OUT     DX,AL
        MOV     DX,ADD9             ;SET DATA POINTER TO COUNTER MODE OF
        MOV     AL,5                ;   REGISTER 5
        OUT     DX,AL
        MOV     DX,ADD8             ;SET COUNTER 5 FOR COUNT REPETITIVELY,
        MOV     AL,33               ;BINARY COUNT,COUNT DOWN, ACTIVE HIGH
        OUT     DX,AL               ;TC, DISABLE SPECIAL GATE, RELOAD FROM LOAD,
        CMP     BX,31               ;CHECK IF >= 31 POINTS/SEC
        JGE     FAST                ;IF SO, JUMP TO FAST
        CMP     BX,0                ;CHECK IF > 0 POINTS/SEC
        JG      MED                 ;IF SO, JUMP
;  BRANCH TO HERE IF POINTS/SEC < 0, MEANS THAT WANT LESS THAN
;  ONE POINT/SEC.
SLOW:   MOV     AL,15               ;SET TO 100 HZ (NO GATE, RISING EDGE
        OUT     DX,AL               ;   OF F5)
        NEG     BX                  ;GET ABSOLUTE VALUE OF BX
        MOV     AX,BX               ;AND MULTIPLY BY 100 TO GET COUNT
        MOV     DI,100
        MUL     DI
        JMP     GO
;BRANCH TO HERE FOR 31 TO 20,000 POINTS/SEC--USE 1 MHZ CLOCK
FAST:   MOV     AL,11               ;COUNT AT 1 MHZ (NO GATE, RISING
        OUT     DX,AL               ;     EDGE OF F1)
        MOV     AX,10000            ;DIVIDE 1,000,000 BY PTS/SEC BY
        MOV     DI,100              ;  GETTING 10E6 INTO DX+AX
        MUL     DI
        DIV     BX                  ;BX=PTS/SEC; RESULT IN DX+AX, BUT
                                    ;   IGNORE DX, SINCE DX=0
```

Listing 1 continued on page 371

Listing 1 continued:

```
        CMP     AX,200          ;DISABLE INTERRUPTS IF >=5000
        JG      FAST2           ;    POINTS/SEC
        CLI
FAST2: JMP      GO
;BRANCH TO HERE FOR 1 TO 30 POINTS/SEC--USE 10 KHZ CLOCK
MED:    MOV     AL,13           ;COUNT AT 10 KHZ (NO GATE, RISING
        OUT     DX,AL           ;    EDGE OF F3)
        MOV     AX,10000        ;CALCULATE NUMBER OF TICKS OF 10,000 HZ CLOCK
        CWD                     ;    PER DATA POINT BY DIVIDING
        DIV     BX              ;    10,000 BY POINTS/SEC
;START CLOCK TICKING AT DESIRED RATE
GO:     MOV     DX,ADD8         ;    AND LOAD COUNTER 5 WITH TICKS
        DEC     AX              ;(COUNT TO ZERO, SO DECREMENT AX
        OUT     DX,AL           ;    FOR CORRECT COUNT)
        MOV     AL,AH
        OUT     DX,AL           ;    8 BITS AT A TIME
        MOV     DI,[BP]+14      ;GET OVERRUN FLAG ADDRESS
        MOV     WORD PTR [DI],0         ;ZERO THE FLAG
        MOV     OVRUN,DI        ;AND STORE THE FLAG ADDRESS
        MOV     DI,[BP]+16      ;GET ADDRESS OF DATA ARRAY
        MOV     DX,ADD9         ;LOAD COUNTER 5 FROM LOAD REGISTER
        MOV     AL,112          ;    AND ARM (START COUNTING)
        OUT     DX,AL
        MOV     DX,ADD4         ;ENABLE EXTERNAL START (PINS 3 + 4 OF
        MOV     AL,132          ;   CONNECTOR J2 MUST BE CONNECTED)
        OUT     DX,AL
;BEGIN DATA COLLECTION; COLLECT UPON EXTERNAL START TRIGGER
DONE:   MOV     DX,ADD4         ;CHECK IF DATA READY
        IN      AL,DX
        CMP     AL, 128         ;BY CHECKING READY BIT (BIT 7)
        JB      DONE            ;LOOP UNTIL READY
        TEST    AL,64           ;SEE IF DATA OVERRUN FLAG SET
        JNE     ERRMESS         ;IF SO, NOTIFY BASIC PROGRAM AND EXIT
        MOV     DX,ADD5         ;YES, DONE, SO GET LOW BYTE OF DATUM
        IN      AL,DX
        MOV     [DI],AL         ;AND STORE IT
        INC     DI              ;GO TO NEXT LOCATION IN ARRAY (1 BYTE LATER)
        MOV     DX,ADD6         ;GET HIGH BYTE AND STORE IT
        IN      AL,DX
        MOV     [DI],AL
        INC     DI
        CMP     PLOT,0          ;DON'T PLOT IF PLOT FLAG=0
        JZ      NOPLOT
;PLOT ROUTINE STARTS HERE
        MOV     TEMP,CX         ;SAVE CX FIRST
        MOV     AH,AL           ;GET HIGH BYTE JUST TAKEN
        MOV     AL,[DI-2]       ;AND LOW BYTE FROM STORAGE SO AX=DATUM
        ADD     AX,2047         ;CALCULATE Y-VALUE TO PLOT =
        CWD                     ;      199-((DATUM+2047)/21)
        MOV     BX,21           ;DIVIDE BY 21--QUOTIENT IN AX
        DIV     BX
        MOV     DX,AX           ;RESULT INTO DX
        NEG     DX              ;NEGATE AND ADD TO 199
        ADD     DX,199
        MOV     SI,TEMPSI       ;GET X-VALUE OF LAST POINT ON SCREEN
        INC     SI              ;GO TO NEXT LOCATION ON SCREEN
        CMP     SI,640          ;TEST IF AT RIGHT EDGE OF 640 X 200
        JL      M1              ;  SCREEN
        MOV     SI,0            ;IF SO, GO TO LEFT EDGE TO PLOT
M1:     MOV     CX,SI           ;GET X-VALUE INTO CX
        MOV     TEMPSI,SI       ;SAVE X VALUE
        MOV     AX,3073         ;AH=12,AL=1 TO WRITE DOT TO SCREEN
        INT     10H             ;PLOT POINT
        MOV     CX,TEMP         ;RESTORE CX
NOPLOT: LOOP    DONE            ;DECREMENT CX AND LOOP IF >0
;BRANCH TO HERE UPON FINISH OR OVERRUN
NOGO:   MOV     DX,ADD4         ;TURN OFF A/D
        MOV     AL,0
        OUT     DX,AL
        STI                     ;RESTORE INTERRUPT SERVICE
        POP     BP              ;RESTORE BP
        RET     12              ;6 ARGUMENTS IN CALL X 2=12
ERRMESS: MOV    DI,OVRUN        ;SET OVERRUN FLAG SINCE A/D GOING
        MOV     WORD PTR [DI],CX   ;TOO FAST
        JMP     NOGO
TIMER   ENDP
RTN_LEN         EQU     $-TEMP  ;LENGTH OF ROUTINE FOR HEADER
CSEG    ENDS
        END     HEADER          ;NEEDED FOR A .BIN FILE CONVERSION
```

Listing 2: *General-purpose data-collection, graphing, and smoothing program in IBM PC BASIC (DOS 1.10).*

```
10 REM GENERAL PURPOSE DATA COLLECTION PROGRAM
20 REM S. GATES, DEPARTMENT OF CHEMISTRY, ILLINOIS STATE UNIVERSITY, NORMAL, IL
30 REM
40 REM Some FOR...NEXT loops are compressed to speed execution
50 CLEAR,31000: BLOAD "TIMER.BIN",31000: TIMER=31000     'Get timer routine
60 DIM A%(1000),B(1000),SG%(9)
70 WIDTH 80:CLS
80 INPUT " Do you wish to process data that have already been collected?";Y$
90 IF Y$="y" OR  Y$="Y" THEN Y=4: GOTO 300
100 INPUT "Enter your name, please"; NAM$
110 D$=DATE$: T$=TIME$: INPUT "Enter the sample identification, please."; S$
```

Listing 2 continued on page 372

before the previous data point has been read from the A/D converter. This is referred to as an overrun. Thus the program must check for the occurrence of an overrun. Upon finding one, the assembly routine sets a flag that can be read by the BASIC program once the data collection is finished.

At very high data-collection rates, interrupt-driven processes occurring in the computer, such as interrupts by the system clock, may interfere with data collection. Indeed, initially this program was limited to 6 kHz until I realized that the interrupts from the system clock were taking too much time. For this reason, at rates above 5 kHz, the subroutine turns off interrupts with a CLI (clear interrupt flag) instruction; when data collection is completed, the interrupts are again enabled by using an STI (set interrupt flag) instruction.

At low-to-moderate data-collection rates, it is useful to have a real-time plot. This is done for each data point collected by loading the low and high bytes of the data point into a register and converting it so that the screen displays a −10-volt A/D reading at the bottom and a +10-volt reading at the top—i.e., so that the full screen is used for the display.

When this assembly-language routine is linked to a higher level program, such as a compiled BASIC or FORTRAN program, only minor changes are required. The Header section must be removed, so that the code starts at Temp. The Timer procedure must be made Public, and the last line of the routine must include an End statement instead of an End-Header statement. After assembly, the subroutine is linked to the calling program using Link in the normal fashion; EXE2BIN does not need to be run in that case.

Sample BASIC Program

A short interpreter BASIC program for the IBM PC that uses the assembly-language routine is shown in listing 2. The program sets aside a region of memory for the routine; the location chosen in line 30 may vary depending upon the amount of memory available in the system. The value 31500 is correct for a 64K-byte system using Advanced BASIC.

After the data has been collected, the overrun flag is checked, and the data is displayed in the high-resolution graphics mode. The data is scaled to fill the entire screen.

Once the data has been collected and displayed, you usually will need to remove high-frequency noise. A simple method for doing this in software is shown in listing 2. It uses a

Listing 2 continued:

```
120 PRINT "Please enter 3 lines of experimental description, including":
    PRINT "Sample preparation, instrument settings, etc."
130 FOR I=1 TO 3: LINE INPUT L$(I): NEXT I
140 INPUT "Enter the channel number (0 to 15)"; C%
150 INPUT "Enter the number of data points to collect.";N%
160 INPUT "Enter the number of data points/second desired. "; S
170 S%=S: IF S< 1 THEN S%=-1!/S   'Convert to proper format for timer routine
180 PRINT "Type any key to start count-down for data collection."
190 I$=INKEY$:IF I$="" THEN 190
200 CLS: FOR I=10 TO 0 STEP -1: LOCATE  12,40 : PRINT I: FOR J=1 TO 500:NEXT J:
    NEXT I                           'Count down;  J loop is delay between counts
210 F%=0                             'Initialize overrun flag
220 P%=1:IF S% > 2000 THEN P%=0     'Plot if < 2000 pts/sec
230 CALL TIMER(A%(1),F%,P%,N%,C%,S%)     'Collect data; all variables MUST be
                                                                      INTEGER!
240 IF F%<> 0 THEN PRINT "Warning--data taken too fast": N%=N%-F%
250 FOR I=1 TO N%
260      IF A%(I) > 32767 THEN A%(I)=A%(I)-65535!
270      A%(I)=A%(I)/.2047:      'Store input as mV, assuming -10 to 10V range
280 NEXT I
290 CLS: PRINT " Enter a 1 to plot data on the screen":
    PRINT " A 2 to store the data in a file.": PRINT " A 3 to smooth the data":
    PRINT " A 4 to get another file":PRINT " A 5 to exit": INPUT Y
300 ON Y GOSUB 340,510,670,770,890
310 GOTO 290
320 '******* SUBROUTINES *********
330 REM Screen plotting routine
340 SCREEN 2 :KEY OFF
350 DEF FNSCALE(Z%)=190-190*(Z%-YMIN)/(YMAX-YMIN)
360 INPUT "Enter the label for the graph",LAB$
370 CLS:YMAX=A%(1): YMIN=A%(1)
380 FOR I=1 TO N%
390      IF A%(I)<YMIN THEN YMIN=A%(I) ELSE IF A%(I)> YMAX THEN YMAX=A%(I)
400 NEXT I
410 YPLOT=FNSCALE(A%(1))
420 PSET (60,YPLOT),0                  'Go to first point
430 FOR I=2 TO N%: XPLOT=60+579*(I-1)/(N%-1): YPLOT=FNSCALE(A%(I)):
    LINE -(XPLOT,YPLOT): NEXT I
440 LOCATE  25,40 : PRINT LAB$;: LOCATE  1,1 : LINE (60,0)-(639,190),,B 'label
    and box plot
450 LOCATE  25,8 : PRINT "1";: LOCATE  25,75 : PRINT N%;: LOCATE  1,1 :
    PRINT YMAX;: LOCATE  24,1 : PRINT YMIN;     'Label axes
460 LOCATE 6,1: PRINT "Type any key to continue";
470 Y$=INKEY$:IF Y$="" THEN 470
480 RETURN
490 REM ********
500 REM Subroutine to store data in a file
510 INPUT "Enter the name of the file in which the data are to be stored.";FILN$

520 OPEN FILN$ FOR OUTPUT AS #2
530 WRITE #2,NAM$,D$,T$               'Save name, date, time
540 WRITE #2, S$                      'Save sample description
550 FOR I=1 TO 3: WRITE #2,L$(I): NEXT I     ' and conditions
560 WRITE #2,N%,S%                   'number of points, sampling rate
570 FOR I=1 TO N%: WRITE #2,A%(I): NEXT I
580 CLOSE #2: RETURN
590 REM ********
600 REM Subroutine to compute second-order 9-point Savitzky-Golay smooth
610 REM    including smooths at both beginning and end of data
620 REM It computes a "smoothed" value for each point by adding together
630 REM      the 4 points on either side of it, plus itself, each multiplied
640 REM      times the corresponding coefficient.
650 REM It then computes the "smoothed" value for each successive point
660 REM      using the original data array.
670 DATA -21,14,39,54,59,54,39,14,-21: 'Savitzky-Golay coefficients
680 RESTORE
690 FOR I=1 TO 9:READ SG%(I):NEXT I    'Get coefficients
700 FOR I=1 TO N%: B(I)=0:DF%=0: FOR J=-4 TO +4: IF I+J< 1 OR I+J>N% THEN 720
710      B(I)=B(I)+A%(I+J)*SG%(J+5):DF%=DF%+SG%(J+5)
720 NEXT J:B(I)=B(I)/DF%:NEXT I        'Divide by sum of coefficients used
730 FOR I=1 TO N%: A%(I)=B(I): NEXT I  'Store back in original array
740 RETURN
750 REM ********
760 REM Subroutine to read previously collected data from disk file
770 INPUT "Enter the name of the file to be processed. " ; FILN$
780 OPEN FILN$ FOR INPUT AS #2
790 INPUT #2, NAM$,D$, T$
800 INPUT #2,S$
810 FOR I=1 TO 3: LINE INPUT #2,L$(I): NEXT I
820 INPUT #2,N%,S%
830 PRINT NAM$,D$,T$: PRINT S$: FOR I=1 TO 3: PRINT L$(I): NEXT I
840 PRINT "Number of points= "; N%, "Points/sec= "; S
850 FOR I=1 TO N%: INPUT #2,A%(I): NEXT I
860 CLOSE #2: PRINT "Type any key to continue"
870 Y$=INKEY$: IF Y$="" THEN 870
880 RETURN
890 END
```

For low-cost RS-232C port expansion, BayTech has...

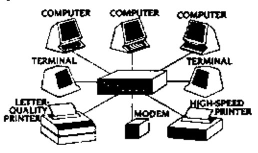
THE RIGHT STUFF

SERIAL PORT EXPANDERS

- Increase I/O ports to 4, 8 or 16.
- Mix & match peripheral devices.
- Port selection through software control; all ports can run with different configurations.
- Models for most applications.

NETWORKING

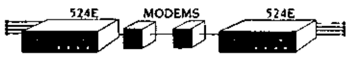
MODEL 528F $619



With its host port and eight peripheral ports capable of any-port-to-any-port interconnection, networking applications with this unit are virtually unlimited.

MULTI-PLEXING

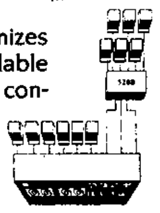
MODEL 524E $319



This unit allows four lines of data to be multiplexed and sent sequentially over a single line, then demultiplexed by a 2nd 524E with automatic distribution to the corresponding peripheral ports.

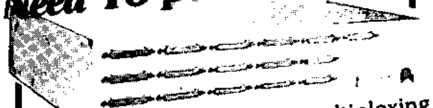
PORT CONTENTION

MODEL 528B $629



This unit maximizes utilization of available ports. Among other configurations, 6 terminals can contend for 3 ports or 7 terminals for 2 ports.

Need 16 ports...$1595



networking, port contention, multiplexing

To place your order or discuss your expansion needs...

1-800-523-2702

BAY TECHNICAL ASSOCIATES, inc.
HIGHWAY 603, P.O. BOX 387
BAY ST. LOUIS, MS 39520
(601) 467-8231

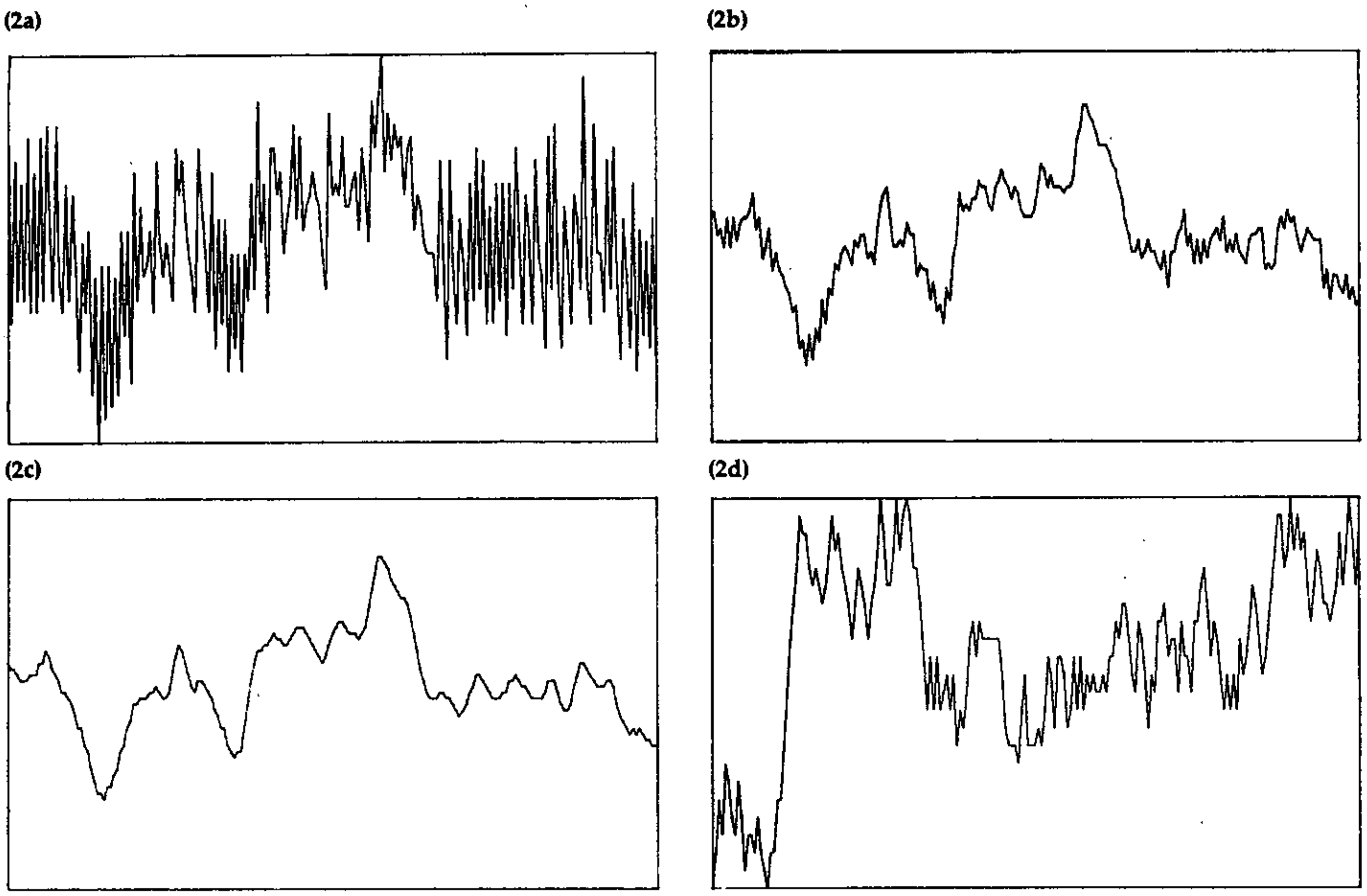


Figure 2: *The effect of filtering on noise levels can be very significant. Figure 2a shows 200 data points taken from an instrument over a period of 20 seconds. Ideally, the signal should be a straight line, but instead shows both long-term and short-term noise. In figure 2b, data from the same instrument is passed through a digital (software) filter once; in figure 2c, it is passed through the filter twice. In figure 2d, data from the same detector is passed through a hardware low-pass filter.*

"Savitzky-Golay" type smoothing algorithm (see reference 3), which is a rapid, easily implemented smoothing technique that is equivalent to fitting a least-squares line through the data. The order of the fit and the number of points included in the fit can be modified to provide varying amounts of smoothing. A second-order, 9-point smooth is the one most often used in my lab. In picking which software filter to use, you may find an article by Cram et al. quite useful (see reference 1). For severe noise problems, other techniques such as ensemble averaging or filtering using fast Fourier transforms may prove useful.

The usefulness of the filtering process is illustrated in figure 2. Figure 2a shows data collected from the detector of a high-performance liquid chromatograph, without filtering. In figure 2d, data was collected from the same detector, but with the low-pass hardware filter being used. In figure 2b, the data is exactly the same as the unfiltered data (figure 2a), except that it has been passed through the Savitzky-Golay second-order, 9-point filter contained in listing 2. In figure 2c, the data from figure 2a has been passed through the Savitzky-Golay filter twice; the reduction in the noise is striking. I often use a combination of hardware and software filtering for optimum results.

Examples of Use

I offer a four-week course to science students that teaches them to interface to a variety of scientific instruments using the techniques described in this article. Students spend one week learning BASIC, two weeks learning the concepts of interfacing and writing simple programs, and one week interfacing the computer to a specific chemical laboratory instrument.

Although the students learn to write data-collection and display routines in BASIC, for their final project they use the Timer routine in listing 1. Using the standardized interfacing system, in one week's time they have written complete data-collection and analysis programs for a number of different instruments, including a pH meter, a UV (ultraviolet)-visible spectrophotometer, a differential scanning calorimeter, a high-performance liquid chromatograph (HPLC), and a polarograph. Even though these programs were written in one week's time, each of these programs is now in routine use in our teaching or research laboratories.

I'll use two examples to show how quickly and easily instruments can be interfaced using this approach.

One student interfaced an IBM PC to a polarograph, using the circuitry shown in figure 1. The polarograph already has a sophisticated preampli-

Using the Tecmar A/D Board

The Tecmar board can be given instructions, and have information read from it, in one of two ways: either the I/O (input/output) mode or the memory-mapped mode can be used. In the I/O mode, various functions of the board are accessed through ports, which are addressed with INP and OUT instructions in BASIC, or IN and OUT instructions in assembly language. In the memory-mapped mode, the functions are accessed at a series of consecutive memory locations; this requires PEEK and POKE instructions in BASIC, or any memory-addressing instruction in assembly language, such as MOV or TEST.

The choice between these two modes is largely a matter of personal preference. The memory-mapped mode is slightly faster but the board is configured at the factory for the I/O mode, which is probably the simpler mode to program. In either mode, you must select the base address, which is the first of 16 consecutive addresses used to communicate with the various functions on the board. The base I/O address set at the factory is 1808. However, other base addresses, as well as the memory-mapped mode, may be selected using the appropriate jumpers or switches.

Other options available on the board include auto-incrementing of the A/D (analog-to-digital) converter (automatically switching the channel from which data is being taken), and the range of the signals coming from or going to the instrument. In addition, three types of inputs to the A/D converter are selectable by appropriate jumper settings: single-ended, pseudo-differential, and true differential. The single-ended setting is normally used, but the differential modes are particularly useful with low-level signals in environments with large amounts of electromagnetic noise. It is also possible to use interrupts to signal the computer when the A/D board has data ready for storage.

The system described in the text uses a −10-V to +10-V bipolar range for the A/D board, clock triggering of the A/D board, and a single-ended input. Only one instrument is normally connected, so the auto-incrementing feature is disabled, as are interrupts. Timer 5 is used to trigger the A/D board.

The clock portion of the Tecmar board provides a general-purpose mechanism for timing various events or for providing timed pulses for triggering various events. At least 18 different modes of operation are possible, each with several options. To the average user, this number of possibilities can prove highly confusing at best.

For triggering the A/D board at specific intervals, however, the process is fairly straightforward. The clock circuitry contains a 1-MHz clock, which is further subdivided either by powers of 10 (BCD scaling) or by powers of 16 (binary scaling), depending upon the option selected. Any one of five counters can be loaded with a count, which is then either incremented or decremented every time the clock "ticks."

For example, with a BCD scaling of divide-by-100, the clock provides a 10-kHz output. Assuming the count is in a downward direction, then the 16-bit counter can be loaded with a value of 99 to provide an output pulse to the A/D board every 0.01 second (i.e., 10 kHz ÷ 100 = 100 Hz). Note that the counter provides an output to the A/D board when it attempts to go below zero (called the "terminal count"); hence, the counter is set to 99 rather than to 100.

To connect the counter pulses to the A/D converter, the output from the specific counter must be directed to the trigger input of the A/D converter. Because of the pin placement on the Tecmar board, the easiest method for doing this is to connect the output of counter 5 to the A/D converter by jumpering pins 3 and 4 of connector J2.

All of the functions of the clock are controlled using two I/O ports accessible to any program. Although these ports are termed control port and data port, both ports are needed to set up the correct timing sequence. In a typical use of the timer, the control port is first directed to point to an internal register called the master mode register. You then select the various control options by loading a 16-bit word into the master-mode register via the data port; this selects options such as whether an 8- or 16-bit I/O bus is being used, what is to be used as a source of the clock frequency, and so forth.

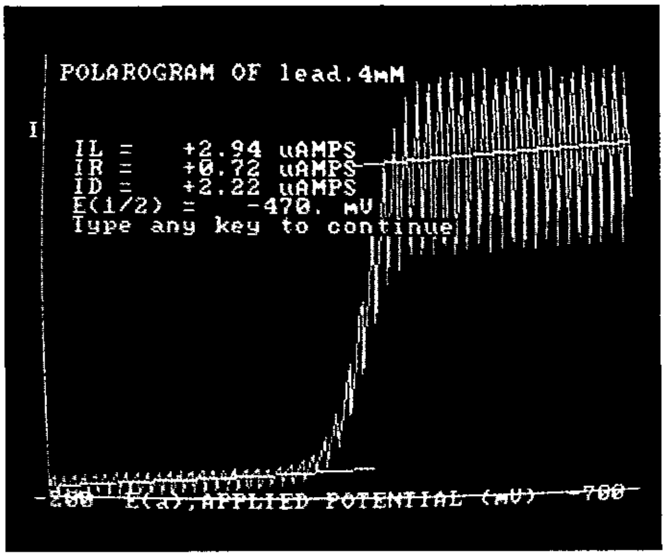
Most of the information, however, is loaded into another internal register, the "counter-mode register." There is one such register for each of the five counters. Hence, the program uses the control port to select which counter-mode register is to be used; in this case, the one for register 5 is selected. The counter-mode register is then loaded, through the data port, with the various options selected for that register. Options include whether to count up or down, whether to count in binary or BCD, and which subfrequency of the clock is to be used. Special options are available if the counters are to be used as a time-of-day clock.

When the program is ready to begin collecting data, the appropriate counter must be loaded with the correct count and "armed," i.e., started counting. Assuming that the A/D converter has been set to recognize the signal from the clock as a trigger by enabling the external start bit, the A/D converter will automatically initiate a conversion (data collection) every time the counter register goes to zero. Hence, the program only needs to wait until the A/D converter signals that it has completed a conversion and then store the data; no timing loops need to be written. The A/D converter will continue to be triggered by the clock until the clock output is turned off by the program.

fier system, so a 10-volt signal could be readily obtained. Hence, the student set the preamplifier on the interface cart to a gain of 1, attached it to the recorder output of the polarograph, used no filtering, and set the Timer routine to collect data for a period of time determined by the potential range scanned.

The major task of the student, then, was to understand the theoretical basis of the instrument readings and to design a program in BASIC to analyze the data. In order to accomplish this, the student had to fit a least-squares line to a sawtooth wave function, determine the inflection point in the curve, and calculate the distance between the two least-squares lines at the inflection point. The A/D readings were then converted to current values in microamperes and the time scale was converted into the applied potential in millivolts.

Students in the analytical chemistry class now use data collected with this system from a series of standard lead samples to calculate the amount of lead in leaded gasoline. Photo 2 shows data collected by a group of students for a standard sample of lead.



```
POLAROGRAM OF lead.4mM

IL =     +2.94 uAMPS
IR =     +0.72 uAMPS
ID =     +2.22 uAMPS
E(1/2) =    -470. mV
Type any key to continue
```

Photo 2: *Students taking our analytical chemistry laboratory course analyze the amount of lead in gasoline using the interface described in the text. The diffusion current (ID on the display) is proportional to the concentration of the lead in the sample.*

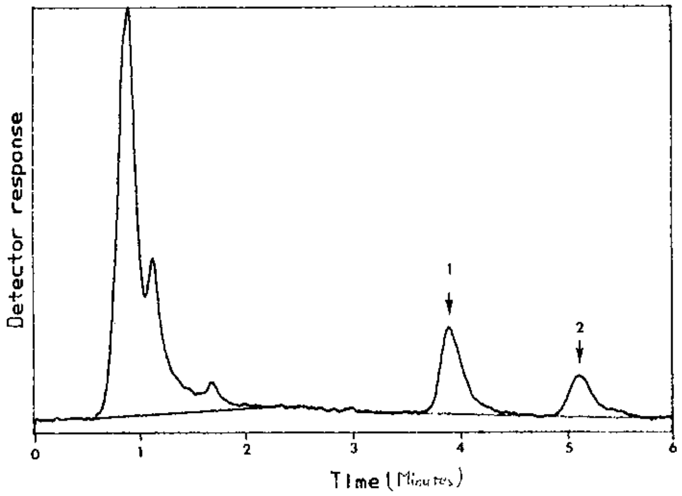


Figure 3: *A common problem in chemical laboratory work is to measure the areas of peaks. Each peak in this figure is integrated by the computer program; the peaks of interest are peaks 1 and 2, which are caffeine and benzyl alcohol, respectively. The benzyl alcohol peak serves as an internal standard for measuring the caffeine. The straight lines under each peak are the baselines determined by the computer during the integration process. The large initial peak is a group of unidentified substances. The sample is a cup of instant coffee.*

A second example of an instrument that students have interfaced is a high-performance liquid chromatograph. The normal output of the HPLC is a 10-mV signal displayed on a strip-chart recorder; hence, the preamplifier was set to a gain of 1000 to provide a 10-volt signal to the A/D converter.

The student writing the program divided it into two sections: a data-collection portion and a data-analysis portion. In the data-collection portion, all of the parameters of the instrument and the sample to be analyzed are recorded, thus providing a permanent record of the conditions of the analysis. The program also asks for the names of the substances being analyzed, if known, and whether an internal standard is being used.

The data collection is done using the assembly-language routine, with a real-time plot of the data. If more than a predefined number of points are collected, the data is "bunched," or averaged, together. The Savitzky-Golay smooth is then performed, and the smoothed data and identifying information are stored in a disk file.

In the second section of the program, the peaks in the data are integrated, and the area of each peak is compared to that of an internal standard. Proper integration involves deciding where each peak starts and stops and then selecting the appropriate baseline to be subtracted from each peak. The results of this process are shown in figure 3. Again, the program is used routinely in our analytical laboratory course; figure 3 shows an analysis of caffeine in coffee performed by a group of students in that course.

Conclusions

One of the many advantages of the revolution in "home" computers is that powerful but inexpensive computers can be used in scientific or industrial laboratories, even by those with relatively limited computer skills. Utilizing off-the-shelf components and simple programming languages, extremely sophisticated data-collection and data-processing systems can be developed very rapidly.

The system described here represents a hardware and software solution to the problem of data collection and analysis in a wide variety of commonly encountered laboratory situations. By making only minor modifications, you should be able to adapt it to other types of hardware and to other types of instrumentation with an extremely wide range of applications, not only in chemistry, but in other scientific and industrial areas as well.■

References

1. Cram, Stuart P., S. N. Chesler, and A. C. Brown III. "Effects of Digital Filters on Chromatographic Signals." *Journal of Chromatography*, volume 126, Amsterdam, Netherlands: Elsevier Scientific Publishing Company Inc., 1976, page 279.
2. Rollins, Dan. "The 8088 Connection." BYTE, July 1983, page 398.
3. Savitzky, Abraham and Marcel J. E. Golay. "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." *Analytical Chemistry*, volume 36, Washington, DC: American Chemical Society, 1964, page 1627. Minor corrections were also published in *Analytical Chemistry*, volume 44, 1972, page 1906.
4. Welch, Mark J. "Expanding on the PC." BYTE, November 1983, page 168.

Stephen C. Gates, Ph.D. (Department of Chemistry, Illinois State University, Normal, IL 61761), is assistant professor of biochemistry. He teaches a course in computer interfacing and does research on computerized chemical analysis of biological samples.

Program Available: *A disk with copies of the programs described in the article is available. Write to the author for information.*