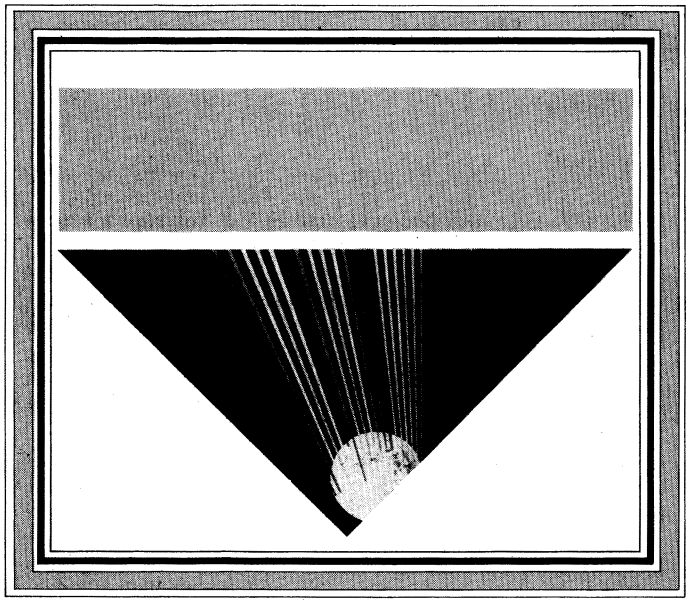


Real-Time Microcomputer Applications Using LLL Basic

Michael D. Maples and Eugene R. Fisher
Lawrence Livermore Laboratory

Introduction

One of the most controversial subjects today is the cost-effectiveness and usefulness of high-level languages in microcomputer applications. Lawrence Livermore Laboratory has been involved with microprocessors since 1972[1,2] and as such has constantly been involved with the evaluation of HLL's. The problem becomes even more complicated when the area of real time data acquisition and control is specifically addressed.

To evaluate this approach, LLL had a Basic 8080 resident interpreter[3]-cross compiler software written. The results have been quite pleasing. The application program generation and debug time has been greatly reduced, and sophisticated assembly language programs are no longer required for every stage of the software development cycle.

Interpreter or assembly language? The following paragraphs describe LLL's experience with this Basic package.

Description

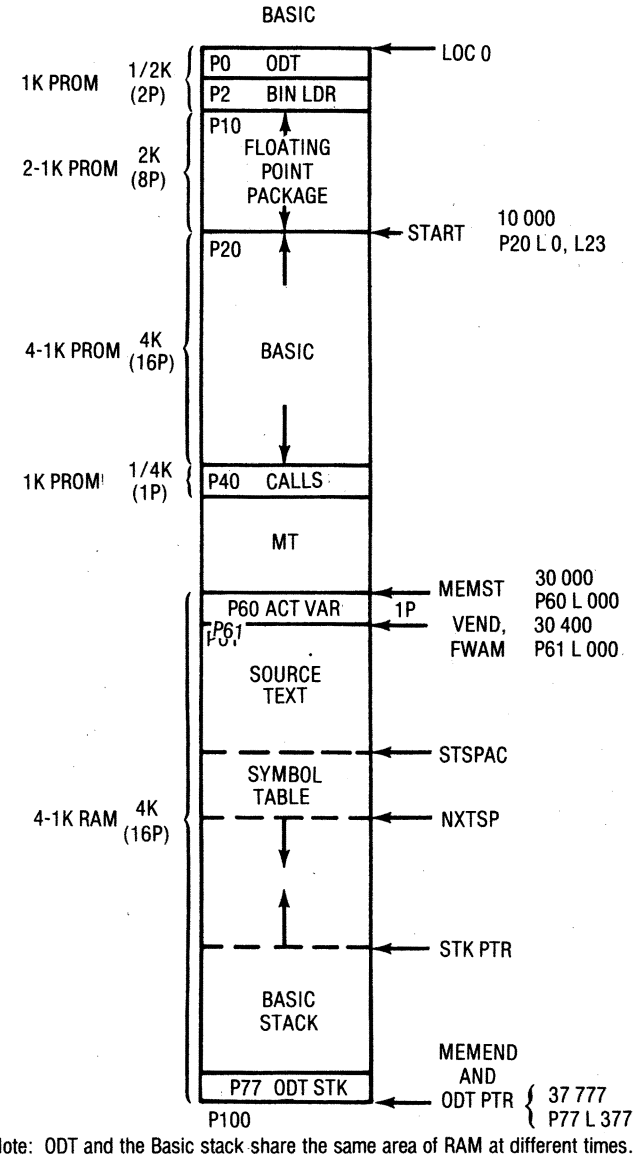
LLL Basic is an interpreter,[3] a code that decodes and immediately executes instructions written in a high-level language. The LLL interpreter executes ASCII programs only. This means that the user program is stored character by character in RAM memory—admittedly *not* a very efficient use of memory. This language allows minimum on-line debug time, but the interpreter must always be resident, a requirement which forces extensive memory overhead (see Figure 1). The resident interpreter requires 5K locations and additional memory for user text space. The language is relatively slow, because each instruction is decoded from its ASCII Basic commands, converted to its machine-equivalent code, and then executed, line by line.

The user develops and debugs his program interactively using the interpretive LLL Basic. Once he has his final version, he runs it through the LLL Basic compiler. The resulting code is in machine language and executes out of programmable-read-only memories. The more efficient code, both in size and in execution speed, is loaded into PROMs for performance of the desired operations (either control or calculative) for the experiment. Using the benchmark program shown in Appendix A, we compared the compiler and interpreter speeds for 1000 iterations. The execution times for the program were 75 and 22 seconds for the LLL Basic interpreter and compiler, respectively.

Being able to write and debug a program with an interpreter while still maintaining the efficiency of a compiler greatly reduces the software effort in microcomputer applications. However, high-level languages are often criticized for not providing good control features. LLL Basic was designed for use in data acquisition and control applications: careful attention was given to allowing good subroutine calls for high-speed, machine-language routines. The second strong feature of the LLL Basic package is the full 24-bit, floating-point arithmetic package,[4] allowing enough dynamic range for good computational capability (refer to Appendix C).

How does Basic compare to assembly language? The microprocessor user may spend a great deal of time condensing his assembly program to save on his per-unit memory cost. Depending on the application, the number of units to be produced, and the need to modify the system's operation at a later date, the programming effort of the user may well be more cost-effective if a high-level language is used.[5]

As a sample comparison, a simple mathematical routine was written in assembly language and in LLL Basic.[6] The assembly language routine is lifted from an actual appli-



Note: ODT and the Basic stack share the same area of RAM at different times.

Figure 1. Basic interpreter memory map (8K PROM, 4K RAM). Memory locations in octal.

cation program. The routine, which performs the simple ideal-gas-law calculation, took one week to program and debug on the Intel 8008.[7] Because of the requirements of the scientific notation package[4] used to perform these calculations, no changes in the routine are made when it is programmed to execute on the Intel 8080. Although the program remains the same, debugging on the 8080 is easier than on the 8008. Therefore, we have amended the original program generation and debug time from five to four days. (The assembly language program is listed in Appendix B; the equivalent LLL Basic program is listed in Appendix C.)

The assembly program is only 100 8-bit words long. Is that too small to be compared with a high-level language equivalent? The general tendency is to assume that high-level languages become cost effective only for programs larger than 2K or 5K bytes. The LLL Basic program took 1 hour to generate, debug, and compile into machine-code ready to be loaded into PROMs. Table 1 shows the relative cost of a sample system using assembly and Basic languages.

Table 1. LLL Basic vs assembly languages.

	LLL-BASIC		ASSEMBLY	
	COST	HRS	COST	HRS
1702-A PROM'S ($50 EACH)	$200	—	$ 50	—
512-BYTE PROM BOARDS ($35 EACH)	70	—	35	—
	$270	—	$ 85	—
LABOR: SOFTWARE EFFORT*	25	1	800	32
TOTAL COST	$295		$885	

*A value of $25 per hour is used by LLL to calculate the cost incurred by a fully-supported engineer.

```
 10 REM "THIS IS A SAMPLE PROGRAM"
 20 REM "SHOWING HOW A ASSEMBLY"
 30 REM "PROGRAM IS CALLED FROM"
 40 REM "BASIC"
 41 LET V2 = 0
 42 LET P2 = 760
 43 LET T2 = 273.15
 50 REM
 60 PRINT "ORIGINAL VOLUME"
 65 INPUT V1
 75 PRINT "ORIGINAL PRESSURE"
 80 INPUT P1
 85 PRINT "ORIGINAL TEMPERATURE"
 90 INPUT T1
100 CALL (1, V1, T1, T2, V2, P1, P2)
110 PRINT "THE VOLUME AT STD PRESS AND TEMP IS"; V2
200 END
```

Figure 2. The LLL-Basic program with assembly call.

```
SUB 1:  POP B       ;   GET STD PRESSURE
        POP H       ;   GET FINAL PRESSURE
        MOV B,C     ;   SET UP FOR DIVISION
        POP D       ;   V2 LOCATION
        MOV C,E     ;   USE FOR RESULT
        CALL LDIV   ;   P1/P2
        MOV D,B     ;   SAVE POINTER
        POP B       ;   GET STD TEMP
        MOV B,C     ;
        POP H       ;   GET ORIGINAL TEMP
        MOV C,B     ;   SCRATCH
        CALL LADD   ;   273.15 + ORG TEMP
        MOV A,L     ;   SAVE RESULT IN B
        MOV L,B     ;   SET UP FOR DIVISION
        MOV B,A     ;   DIVISION = 273.15 + ORG TEMP
        CALL LDIV   ;   RESULT PTR IN C REG
        MOV B,C     ;   TEMP RATIO
        MOV C,L     ;   RESULT
        MOV L,E     ;   P1/P2
        CALL LMUL   ;
        MOV B,C     ;
        POP H       ;   ORIGINAL VOLUME
        MOV C,E     ;   V2 POINTER
        CALL LMUL   ;
        RET
```

Figure 3. Assembly language portion of program.

Figure 2 shows the LLL Basic program with assembly call for performing the ideal-gas-law calculation; Figure 3 shows the assembly program. Note that LLL Basic has its own mathematical capabilities and that the example in Figure 3 is presented only to demonstrate how an assembly language call is accessed from Basic.

```
RUN
ORIGINAL VOLUME
:237
ORIGINAL PRESSURE
:14.7
ORIGINAL TEMPERATURE
:22.3
THE VOLUME AT STD PRESS AND TEMP IS 4.2380 E 00 READY.
```

Figure 4. Actual run of sample program.

LLL Basic reads in the needed variables, places pointers to the variables on the stack (i.e., V1, T1, T2, V2. . . .), transfers control to the assembly patch performing the calculations, clears the stack, and then returns control back to itself. The latter then prints out the final answer (see Figure 4). When fast control or data collection routines are needed, the programmer can easily place a desired assembly program under LLL Basic which has a call statement defined just for interfacing assembly patches to its program.

Besides cost-effectiveness and usability, the execution speed of a high-level language is always a concern to the user. A simple benchmark program (see Appendix A) was written to compare LLL Basic with other Basics and with PDP-8 Focal. Table 2 shows that, at least for computations, LLL Basic is impressive. We must realize, of course, that a final program will be executing the compiler Basic, not the interpreter Basic. We were impressed by the comparison of LLL Basic with Focal, because Focal is and has been used so well in data acquisition and control environments.

Table 2. Comparison of execution times for the benchmark program shown in Appendix C.

INTERPRETER	SECONDS PER LOOP	EXECUTION TIME* RANKING
DATA GENERAL 840 MULTIUSER—BASIC (ONLY ONE USER DURING BENCHMARK)	0.0045	0.25
DEC PDP 11/45—BASIC	0.003	0.15
INTEL 8080 COMPILED—LLL-BASIC	0.02	1.0
INTEL 8080 LLL-BASIC INTERPRETER	0.075	3.4
DEC PDP 8E—FOCAL	0.04	1.7

*Realtive to Intel 8080 compiled-LLL-BASIC.

Table 3 shows the execution times—i.e., the average assembly language times—for the various operations allowed in the floating-point package. The sum of the times required for two additions, one subtraction, one multiplication, and one division equals 19.6 milliseconds—1.12 times faster than the compiled LLL Basic. Every user must judge his application carefully to determine if this increased speed, along with the smaller size of the code, warrants the extra effort required to program in assembly rather than in a high-level interpretive language.

Table 3. Average execution times for scientific notation package using assembly language on Intel Model 8080 with LLL Basic.

OPERATION	MILLISECONDS
ADD	2.4
SUBTRACT	2.4
MULTIPLY	5.4
DIVIDE	7.0

Other commands and capabilities of LLL Basic are shown in Tables 4 and 5. The capabilities listed are the minimum requirements that we felt would allow the language to be useful in our laboratory environment. The interpreter Basic allows all of the listed capabilities and maintains the 5K-byte size restriction we placed upon it. Though modifications are continuing on this interpreter-compiler Basic, four applications have been developed with this system.

Table 4. Interpreter commands of LLL Basic.

COMMAND	ACTION
RUN	BEGIN PROGRAM EXECUTION.
SCR	CLEAR PROGRAM FROM MEMORY.
LIST	LIST THE ASCII PROGRAM IN MEMORY.
PLST	PUNCH PAPER TAPE COPY OF THE PROGRAM.
PTAPE	READ PAPER TAPE COPY OF PROGRAM FROM HIGH-SPEED READER.
CNTRL S	INTERRUPT PROGRAM BEING EXECUTED.

Table 5. Additional capabilities of LLL Basic.

STATEMENT	INTERPRETATION
0 to 32767	MAXIMUM RANGE OF LLL-BASIC LINE NUMBERS.
REM	INDICATES A COMMENT. COMMENTS ARE GENERALLY ENCLOSED IN QUOTES. SOME SPACES ARE IGNORED EXCEPT WHEN ENCLOSED IN QUOTES.
END	INDICATES END OF PROGRAM.
STOP	STOPS THE PROGRAM.
GO TO XX	TRANSFERS TO LINE NUMBER XX.
DIM	DECLARES AN ARRAY. ONLY ONE-DIMENSIONAL ARRAYS WITH AN INTEGRAL NUMBER OF ELEMENTS ARE ALLOWED.
LET	INDICATES AN ASSIGNMENT STATEMENT. ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, OR A SPECIAL FUNCTION MAY BE USED.
IF (expression) THEN XX	IF THE CONDITION OF THE EXPRESSION IS MET, THIS CONDITION STATEMENT TRANSFERS TO LINE NUMBER XX.
INPUT	ALLOWS NUMERICAL DATA TO BE INPUTTED VIA A TERMINAL.
PRINT	ALLOWS NUMERCIAL DATA AND CHARACTER STRINGS TO BE PRINTED ON A TERMINAL.
FOR	CAUSES A PROGRAM TO ITERATE THROUGH A LOOP A DESIGNATED NUMBER OF TIMES.
NEXT	SIGNALS END OF LOOP AT WHICH POINT THE STEP VALUE IS INCREMENTED AND LOOP TERMINATION IS TESTED.
GO SUB NN	TRANSFERS CONTROL TO A SUBROUTINE THAT BEGINS AT LINE NN.
RETURN	RETURNS CONTROL TO THE LINE AFTER THE LAST CO SUB.
CALL	USED TO CALL ASSEMBLY LANGUAGE SUBROUTINES. FOR EXAMPLE, IN "CALL(N, A, B, . . .)," N IS SUBROUTINE NUMBER AS LISTED IN THE ASSEMBLY PATCH TABLE; A, B, ETC., ARE PARAMETERS, CONSTANTS, VARIABLES, OR EXPRESSIONS.
GET (X)	READ 8080 INPUT PORT X.
PUT (Y)	OUTPUT A BYTE OF DATA TO OUTPUT PORT Y.

Applications

The first system developed using the interpreter-compiler was for a temperature controller in the inlet probe of a high-resolution mass spectrometer[8] (see Figure 5). The engineer needed the inlet temperature of a mass spectrometer to cycle in a predetermined fashion. A temperature thermocouple located at the inlet was connected to a digital panel meter that performed the analog-to-digital conversion. LLL Basic, via an assembly language call, reads the digital panel meter. The program then ranges the reading, calculates the actual versus the desired temperature variations, and applies the appropriate amount of heat to keep the inlet temperature tracking correctly (see Figure 6). The hardware for this system took approximately 1-1/2 months to design and debug; the software effort took about 3 weeks. The system took 6K ROM and 3K RAM in its interpretive phase and now consists of 4.5K ROM and 0.5K RAM in its compiled form.

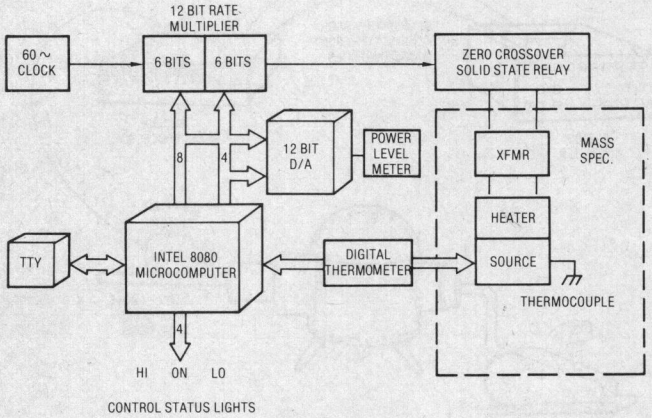


Figure 5. Block diagram of temperature programmer for high-resolution mass spectrometer system.

The next application was for an ion cyclotron resonance experiment (see Figure 7), a completely research-oriented project. Because maximum system flexibility was needed, the engineer chose the Basic language system, thus allowing the scientist to reconfigure the experiment as desired. In this application (see Figure 8), 3 months were required for developing the hardware and 1 month for the software. The software consists of 9K of PROM (Basic and calls) and 8K of RAM (user space for actual Basic program). The system still uses the interpretive Basic and will probably remain in this configuration for some time. Once the experimental procedure is well defined, the software will be compiled and frozen in PROMs.

In these two applications, the engineer designed and developed both the hardware and software system. In the next two applications, the total application program was written by the scientist, not the engineer—an arrangement whose advantages have been noted elsewhere.[5] The engineer provided the scientist with the basic hardware system and the desired assembly language patches and their related Basic call statements. The scientist then programmed the experiment himself.

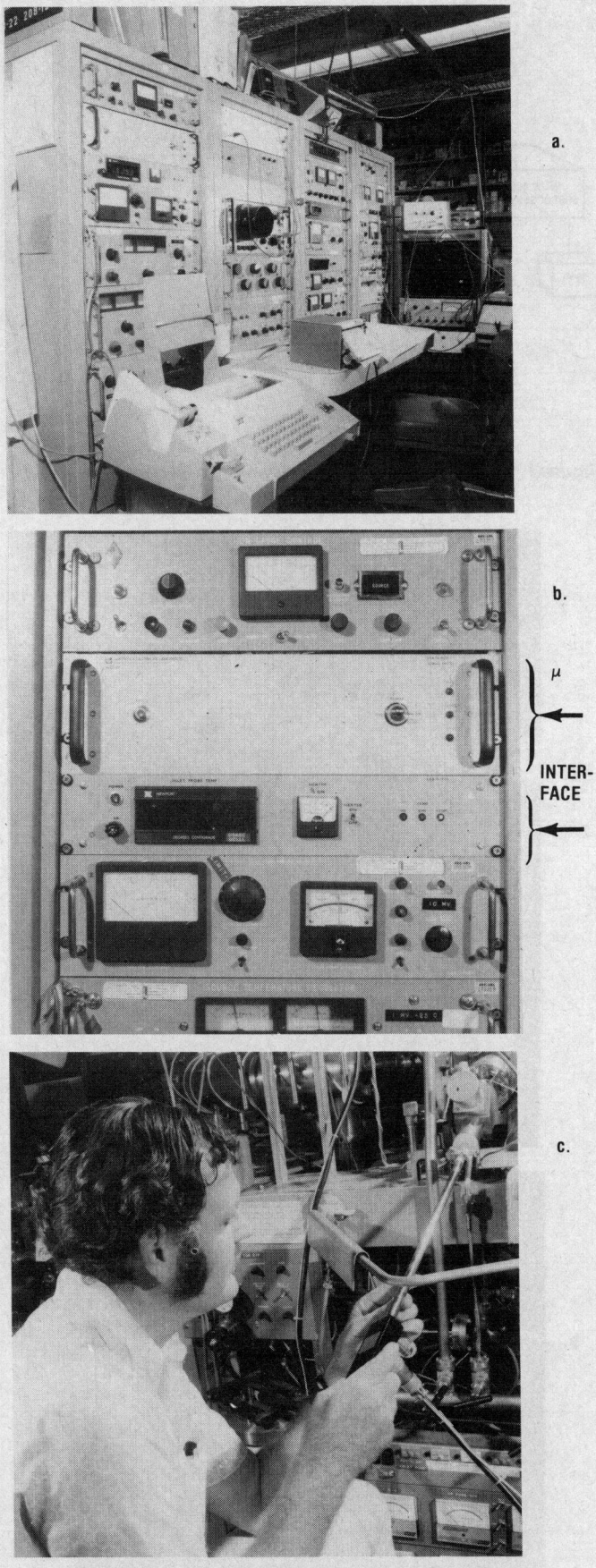


Figure 6. High resolution mass spectrometer showing: (a) control systems, (b) microcomputer controller (upper arrow) and controller interface (lower arrow), and (c) the inlet probe transducer being inserted into the system.

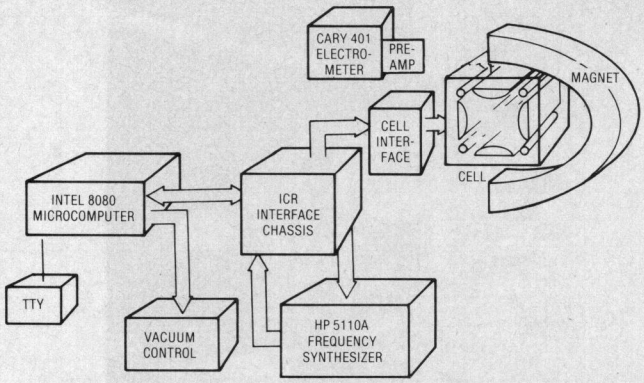
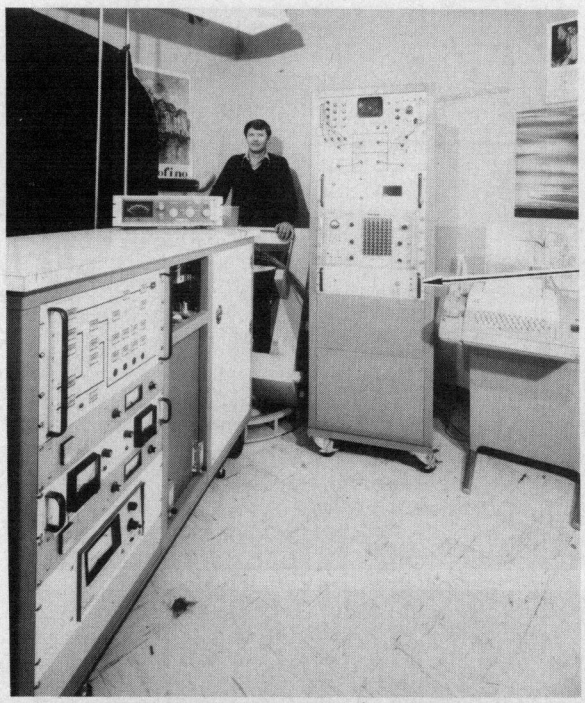


Figure 7. Block diagram of Ion Cyclotron Resonance (ICR) experiment.

a.



b.

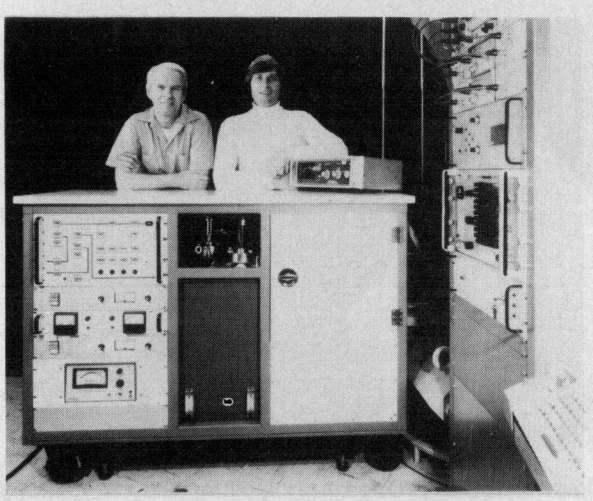


Figure 8. Instrumentation setup used for ICR experiment showing: (a) the microcomputer chassis (see arrow), and (b) the valve control rack.

In the "emission-coupled surface analysis" or ESCA experiment (see Figure 9), the microcomputer system, via its push-button (relay) front panel, monitors and can program the Hewlett-Packard 2100 ESCA control computer. The microcomputer also has control over the sample probe, which contains three samples separated by a 120-degree angle. The scientist has the appropriate Basic call statements for programming and monitoring the ESCA system and for controlling the angle of any sample being examined. He can load three samples and set up a Basic program for analyzing the samples at many different incident angles and under different operating conditions. The run, which may take several hours, can be executed under the control of the microcomputer. The scientist need not be present and can use this time for analyzing and correlating other experimental results. This particular application is presently being programmed by a chemist—who is very excited about controlling his experiment via Basic.

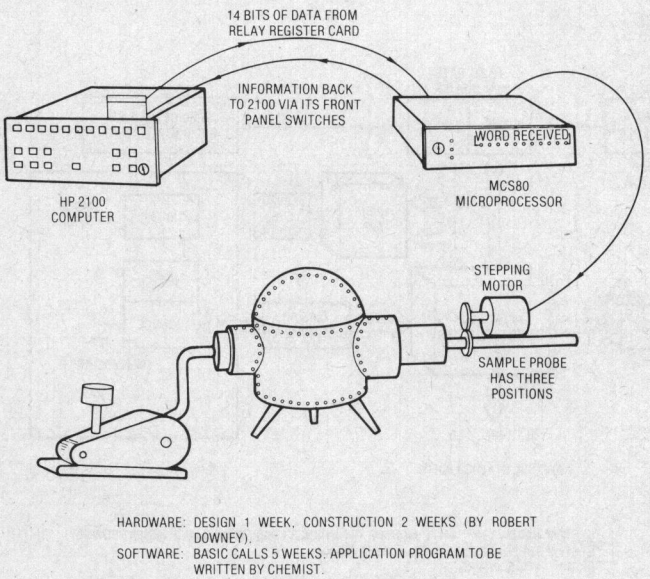


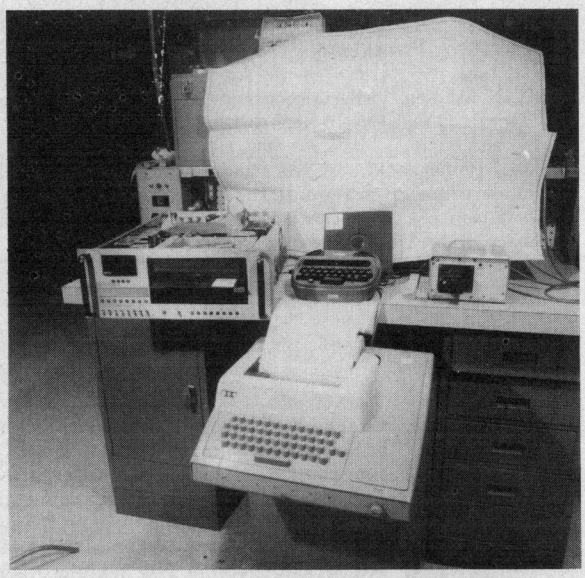
Figure 9. Diagram of system used for Emission-Coupled Surface Analysis (ESCA).

The final application is a portable data acquisition system (see Figure 10) which allows the scientist to configure completely and to reconfigure the acquisition system via the Basic call statements. The 14 call statements in this package include reading the front panel switches, writing the front panel LED's, reading the digital inputs from the experiment, controlling the timing, reading and converting of the analog data, and reading or writing the floppy disk system (see Figure 11). The development software for this system included PLM, Basic, and assembly languages. The complex interfaces between these languages and the hardware are transparent to the user. The user writes his data collection program in Basic. When he completes his experiment, he collects the needed data and passes the system along to another user. Again, the engineer is not involved in the programming.

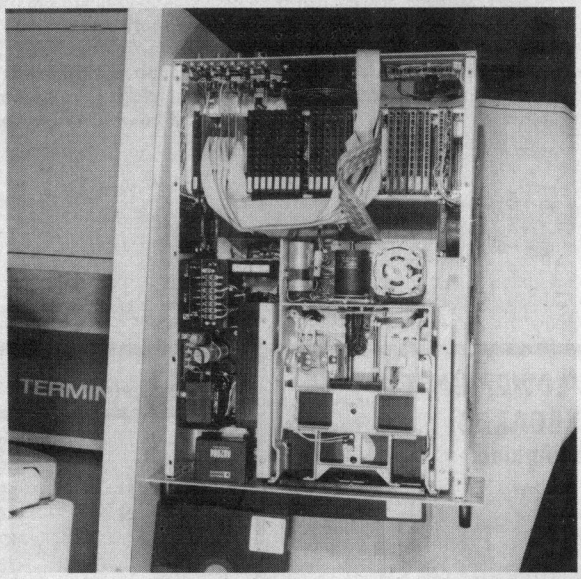
Conclusions

Basic has shown itself to be of great value on laboratory minicomputer systems and is becoming impressive on our microcomputer systems. In an environment where

a.



b.



c.

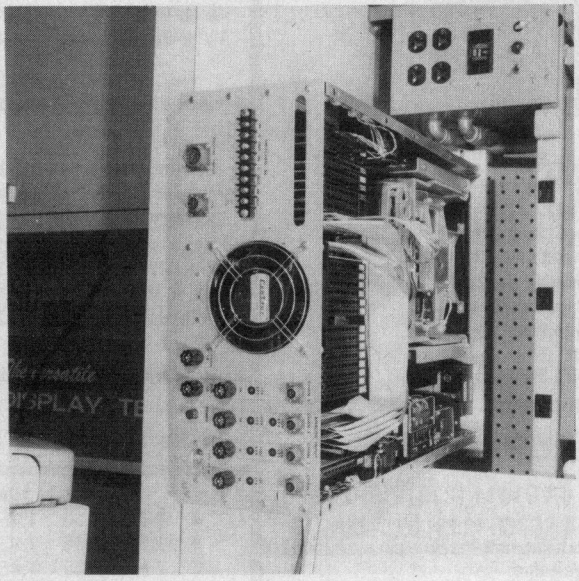


Figure 10. Portable data acqusition system shown in (a), with top view shown in (b), and rear view in (c).

development time is critical, a modular microcomputer system coupled with an interpreter-compiler software system is a necessity. High-level languages such as the LLL Basic described in this paper are viable alternatives to assembly languages and will most certainly earn a position of respect in the eyes of microcomputer users.

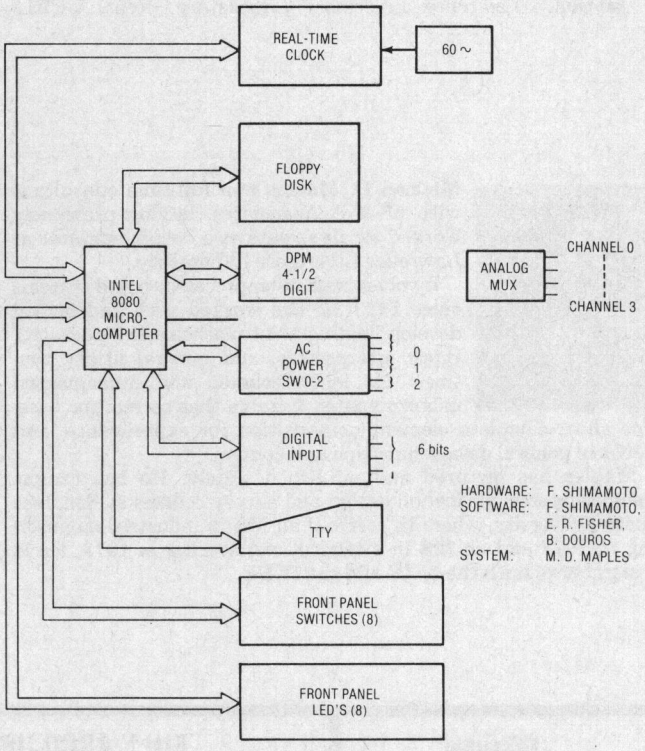


Figure 11. Block diagram of portable data acquisition system.

Acknowledgments

The Basic compiler and interpreter described were developed at the University of Idaho by John Dickinson, Jerry Barber, and John Teeter, under a contract with Lawrence Livermore Laboratory. Jerry Barber, during his summer residence at LLL, was extremely helpful in setting up the interpreter and making it functional. Terry Allison's work in making both the compiler and interpreter operational on our CDC 7600 computers was invaluable. We also thank Bob Downey, Chuck Pomernacki, Faith Shimamoto, and Jim Breshears for allowing us to present their applications in this paper.

This work was performed under the auspices of the U.S. Energy Reserach and Development Administration under Contract No. W-7405-Eng-48.

References

1. E. R. Fisher, "A Modular Approach to Microprocessors," *Digest of Papers,* COMPCON 74 Fall, pp. 55-57.

2. E. R. Fisher, "A System Approach for Processor Implementation," *Proc. Journess d'electronique,* October 1974, pp. 161-175.

3. R. Eckard and J. Barber, "User's Guide to the LLL-BASIC Interpreter," Lawrence Livermore Laboratory, Rept. UCID-17090, April 1976.

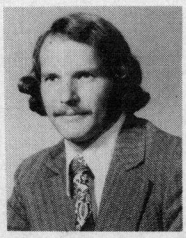
4. M. D. Maples, "Floating-point Package for Intel 8008 and 8080 Microprocessors," Lawrence Livermore Laboratory, Rept. UCRL-51940, October 1975.

5. J. Gibbons, "When to Use Higher-level Languages in Microcomputer-based Systems," *Electronics,* August 7, 1975, pp. 107-111.

6. E. R. Fisher, "High-level Languages in Microcomputer Automation," Lawrence Livermore Laboratory, Rept. UCRL-77868, February 1976, presented at Electro 76, Boston, Mass., May 11-14, 1976.

7. M. D. Maples, "Microprocessors in Computing," *Digest of Papers,* COMPCON 75 Spring, pp. 69-71.

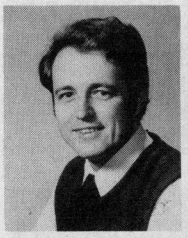
8. C. L. Pomernacki, "A Microcomputer-based Controller for Temperature Programming the Direct Inlet Probe of a High-resolution Mass Spectrometer," Lawrence Livermore Laboratory, Rept. UCRL-79025, January, 1977.



Michael D. Maples is a full-time consultant with M & E Associates, having previously worked for five years as a design engineer at Lawrence Livermore Laboratory.

Involved with microprocessor-based systems since 1973, he has worked with and helped develop floating-point mathematical packages, Basic interpreters, and general utility routines. He also designed and implemented microcomputer systems that correct for baseline shift, calculate element composition for experiments, and perform general data acquisition and control.

Maples has lectured and published widely. He has taught microcomputer application design and survey courses at San Jose State University, where he received an MS in industrial engineering in 1972 and an MS in electrical engineering in 1973. He is a member of both the AIIE and the IEEE.



Eugene R. Fisher is a senior design engineer at Lawrence Livermore Laboratory and a part-time consultant with M & E Associates.

Fisher is a leading proponent of the modular design technique for microcomputer systems used in laboratory automation and data acquisition systems. His modular microcomputer has met wide acceptance in his own and many other national laboratories.

He has designed and implemented a microcomputer floppy-disk system, designed an 8080-based IEEE 488-1975 bus handler, and has spent eight years in minicomputer systems as applied to process control and laboratory automation. He has published and lectured widely.

Appendix A. Sample of benchmark program used for comparison of LLL Basic compiler-interpreters.

```
10   REM "THIS IS A SIMPLE BENCHMARK PROGRAM. IT"
15   REM "MULTIPLIES, DIVIDES, ADDS AND SUBTRACTS."
20   REM
30   INPUT A
40   INPUT B
50   LET C = A + B
60   LET A = A + 1
70   LET E = B/C
80   LET F = A * E
90   LET C = C - F
100  IF A = 1001, THEN 200
110  GO TO 50
200  PRINT "THE LOOP IS DONE AT"
210  END
```

Appendix B. Sample of ideal-gas-law calculation written in assembly language

```
ORG 37400Q
  SCR E QU 30Q
  PAG EQU 4400Q
  PRESS EQU 4Q
  TEMP EQU PRES +4
  VOL EQU 0Q
                    ; READ VOLUME
LXI H,PAG           ; PAGE 11 LOC 0
MVI C,SCR           ; SCRATCH AREA
CALL INPUT          ; READ VOLUME FROM TTY TO
                    ; MEMORY BUFFER
MVI L,PRESS         ; READ PRESSURE
MVI C,SCR           ; SCRATCH
CALL INPUT          ; READ AND STORE IT
MVI L,TEMP          ; READ TEMP
MVI C,SCR           ; SCRATCH
CALL INPUT          ; READ AND STORE IT
MVI L,TEMP+4        ; CALCULATE STD PRESSURE
MOV B,L             ; SAVE IN B
MVI M,276Q          ; LOAD 760 INTO MEMORY
INR L
MVI M,000Q
INR L
MVI M,110Q
INR L
MVI M,012Q
MVI L,PRES          ; DIVIDED
MVI C,TEMP+8        ; RESULT
CALL LDIV
MVI L,TEMP+4        ; STORE 273.15
MOV B,L             ; SAVE IN B
MVI M,210Q
INR L
MVI M,223Q
INR L
MVI M,063Q
INR L
MVI M,0111Q
MVI L,TEMP
MVI C,SCR           ; SCRATCH
CALL LADD           ; ADD TO GET DEGREES KELVIN
MOV C,L             ; STORE RESULT PTR IN C
MOV L,B             ; 273.15 IN B
MOV B,C             ; TEMP IN DEGREES K
MVI C,TEMP+12       ; RESULT
CALL LDIV           ; DIVIDE THEM
MOV B,C
MOV L,TEMP+8        ; PRESSURE
MVI C,PRES          ; TEMPORARY RESULT
CALL LMUL
MOV L,C             ; RESULT PTR TO L
MOV B,VOL
MUI C,TEMP+12       ; FINAL VOL AT STD PARD GING
CALL LMUL
MVI L,C             ; FINAL RESULT
MVI C,SCR
CALL CVRT           ; WRITE ANSWER TO TTY
END
```

Appendix C. Sample of ideal-gas-law calculation in LLL Basic

```
10REM"THIS IS A TEST PROGRAM TO COMPARE PLM ASSEMBLY AND"
20REM"BASIC LANGUAGES FOR WRITING FAST AND EASY"
30REM"TO DEBUG PROGRAMS FOR MICROPROCESSORS."
35REM
40REM"CALCULATE VOLUME USING THE IDEAL GAS LAW"
45REM"VOL2=VOL1*(P1/P2)*(T2/T1)"
50REM
60PRINT"ORIGINAL VOLUME"
65INPUTV1
75PRINT"ORIGINAL PRESSURE"
80INPUTP1
85PRINT"ORIGINAL TEMPERATURE"
90INPUTT1
100REM"NOW THAT THE VALUES ARE INPUTTED LET'S DO THE
      CALCULATION."
105LETA=T1+273.15
115REM"T1 IS IN DEGREES C"
116REM"A IS IN DEGREES KELVIN."
120LETB=P1/760.0
125REM"P1 IS IN MM OF HG."
127REM"P2=760.0 WHICH IS 1 ATM."
130LETC=273.15/A
135REM"C IS T2/T1"
136LETE=C*B
137REM"E=(T2/T1)*(P1/P2)"
140LETV2=E*V1
150PRINT"THE VOLUME AT STD PRESS AND TEMP IS";V2
200END
READY
```