A microcomputer-controlled laboratory: Software

JO W. TOMBAUGH, RICHARD F. DILLON, BARRY MILLMAN,
and WILLIAM R. BEZANSON

Carleton University, Ottawa K1S 5B6, Canada

The software used in a microcomputer-controlled laboratory is described. Primary emphasis is placed upon describing the capabilities of a powerful, easy-to-use programming language appropriate for controlling research in a wide variety of applications. It is particularly appropriate for physiological and operant laboratories. Design of the language and accompanying operating system follow a rigorous modular, structured approach, with most coding done in high-level languages. As a result, the code is easy to maintain and expand. In addition, it is not difficult to adapt the programs to other microprocessors to take advantage of a rapidly improving technology.

In this paper we describe powerful easy-to-use software developed for operating a laboratory control system (Dillon, Millman, Tombaugh, Ferguson, & Bezanson, 1979). In designing this software system, we hoped to satisfy a number of requirements. First, we wanted the language to be useful in a wide variety of applications, including those in which analog-to-digital (A/D) and digital-to-analog (D/A) equipment were involved. In particular, all the capabilities normally found in special-purpose signal-averaging equipment were desired. Second, we wanted a language to be easy for a nonprogrammer to read and learn to use. A great deal of time and discussion with nonprogrammers was directed toward this goal, culminating in the decision that nonprogrammers are typically unfamiliar with the idea of flow of control through a program. Rather, they prefer to identify an external event and the conditions that cause the event to occur. Third, extensive, easy-to-understand error diagnostics, which are always desirable, were viewed as essential for nonprogrammers. Finally, the laboratory computer should communicate with a remote large computer that has the latest statistical packages and languages available. Consequently, the software needed to transfer files between the laboratory and the Carleton University central computer was developed.

This project was supported by NRC grants to W. E. Webster and Brian Tansley, and by grants from the Department of Psychology, Carleton University. Computer time on the Carleton University Honeywell/Xerox Sigma 9 was provided by a grant from the President's Computer Fund. Requests for reprints should be addressed to Jo W. Tombaugh, Department of Psychology, Carleton University, Ottawa, Ontario K1S 5B6, Canada. B. Millman is now at Data Kinetics, 31 Arthur Street, Ottawa, Ontario, Canada. The authors thank David Parlour, a student in the Carleton University Department of Engineering, for his substantial contributions to the development, coding, and testing of various aspects of the system.

In addition to the above functional requirements, several design criteria were adhered to as we planned and programmed the software. To insure that the programs would be easy to debug and maintain, highly readable, and easy to expand, the system was designed to be highly modular and structured. In addition, a high-level language, rather than assembly language, was used for coding. This choice led to faster implementation and made the programs more portable; they can be more easily adapted to different computer configurations than programs coded in assembler language. Such portability is important because hardware improvements are occurring rapidly and a large software package should not be tied to one microprocessor that may soon be obsolete. We now plan to shift from a M6800 microcomputer to the M6809 when it becomes available, because of its vastly improved capabilities. In a similar fashion, we would like to be able to transfer the language to a number of microcomputers and even minicomputers with a minimum of work.

The laboratory system software consists of (1) a program to allow the user to set up experiments, (2) a program to run experiments, and (3) a program allowing transfer of files between the system and the central campus computer.

We are implementing two further interactive programs. The first allows modification of already existing user programs, a job now done with a text editor. The second provides simple data editing capability and off-line listing capacity by the laboratory system rather than by the central computer.

Before describing the laboratory software, we outline the general-purpose system software that is available for the microcomputer.

GENERAL-PURPOSE SYSTEM SOFTWARE

Those familiar with minicomputers realize that

manufacturers of microcomputers do not presently provide high-level software with their computers. However, a number of independent vendors offer high-quality software at very reasonable prices. Since we essentially built our own computer, we had to obtain the necessary software. The floppy disk system from Southwest Technical Products has a convenient disk-based operating system called FLEX that allows random access, named files, and a fairly complete disk version of BASIC (see Southwest Technical Products Corporation, 1978). An editor, assembler, text-processing system, and disassembler on diskettes were purchased from Technical Systems Consultants, the authors of the FLEX operating system (Technical Systems Consultants, 1978). One of the first programs we wrote enabled us to transfer files between our system and the central computer. This link allowed us to develop much of our software on the larger computer and provided access to data analysis programs and mass storage superior to anything we could provide on the microcomputer.

Our software on the central computer is written in PASCAL, a language available for many microcomputers (Bowles & Hollan, Note 1). The decision to write as much software as possible on a central computer allowed us to devote the microcomputer system time exclusively to development of the run-time system. Also, researchers are able to write new experiment-control programs on the central computer without disrupting ongoing research. When PASCAL is available for the

Motorola 6800 or 6809 computer, we will provide a microcomputer-based version of this software for users who prefer to work independently of the central computer.

Most of the run-time system executed on the microcomputer is written for a cross-compiler on the central computer in a subset of PL/I, called MPL, purchased from Motorola (see Motorola Semiconductor Products, Inc., 1976). With a flexible operating system, the system software is equal or superior to that provided with many minicomputers.

LABORATORY SOFTWARE

Experimental Control Programs

Overview. The user programs his experiments in a high-level user language. An interpreter provides prompts and line-by-line diagnostics for the researcher. The language itself is designed to simplify programming of uncomplicated evoked-potential research, while providing powerful advanced capabilities for those interested in complex experimental manipulations. In evoked-potential research, electrical activity in the brain is recorded, starting from a synchronizing event such as a light flash.

To keep a user language as straightforward as possible, definitions of stimuli, responses, assignments of values to variables, and data collection are specified separately from the experimental procedures. Table 1

Table 1
Example of an Experiment Control Program Written in the User Language

1. program name: EVOKED TWO				
2. input from keyboard or file? KEYBOARD				
3.		definitions		
4. stimuli				
5. name	relay #	time on	time off	# pulses
6. LIGHT	7	1S		
7. WARNING	4	500M	500M	3
8. A/D input				
9. name	channel #	time between samples		# samples
10. OCCIPLOBE	3	1M		200
11. TEMPLOBE	8	5M		120
12. EEG = OCCIPLOBE, TEMPLOBE				
13. A/D data definitions				
14. note: computations beyond raw and average are signal mean, variance, maximum, minimum, latency to peak				
15. name	a/d name	computation		record during
16. occiplobe	occiplobe	raw		
17. templobe	templobe	raw		
18. aveocciplobe	occiplobe	average		PAUSE
19. avetemplobe	templobe	average		PAUSE
20. OCCIPEAK	OCCIPEAK	MAXIMUM		PAUSE
21.	control statements			
22. event name	caused by		results in	
23. START	STARTSWITCH		PULSE ON/OFF WARNING	
24. SESSION	START + 15S		LOOP 15	
25. ... TRIAL	SESSION		LOOP 20	
26. EVOKED	TRIAL		TURN ON/OFF LIGHT	
27.			TAKE EEG	
28. ITI	EVOKED + 10S		NEXT TRIAL	
29. ... PAUSE	TRIAL DONE + 40S		NEXT SESSION	
30. FINISH	SESSION DONE		PULSE ON/OFF WARNING	
31. END	FINISH DONE		STOP	

Note—Lowercase letters represent computer-initiated text.

shows an example for an evoked-potential experiment in which two channels of analog data are recorded. Fifteen trials with signal averages computed over 20 A/D recordings are included. A warning is presented at the beginning and end of the session, and signal averages (but not the raw data) are stored permanently on disk. In this and later examples, computer-initiated text is shown in small letters; user-written instructions are capitalized. For example, in Line 1, "program name:" is written under program control and EVOKEDTWO is typed by the user. Line numbers are used here to facilitate description of the language but are not indicated in actual program listings. There are three sections to the program: (1) specification of the name of the experiment, which will be used in the future to bring the user program into computer memory (Lines 1 and 2), (2) definitions (Lines 3-20), and (3) control statements (Lines 21-31). Notice in Line 22 that three columns are labeled "event names," "caused by," and "results in." That is, using Line 23 as an example, the event named START is caused by closure of the STARTSWITCH and results in pulsing some kind of WARNING on and off. For now, this example is presented to provide an overview of the general approach taken in our event-oriented language in which external occurrences such as switch closures cause an event and its related operations to occur. This approach avoids the necessity for the experimenter to code waiting loops and allows concurrent processing of several events. Later, we will return to the example in Table 1 for a detailed explanation of individual instructions. This task is postponed until a more systematic presentation of the full range of capabilities has been presented.

Events. Various conditions cause an event to occur. Event names can be up to 16 characters in length and are supplied by the user. Events are triggered (i.e., activated or caused to occur) by switches, responses, the passage of time following the start of another event, or by the passage of time following the completion of another event. Time can be specified in seconds (S) or milliseconds (M). Default is to seconds. If no time is specified, it is taken to be 0 sec. For example, assuming that TRIAL is an event name used elsewhere in the program and BARPRESS is defined as a response, some acceptable forms and their interpretations are shown below.

event name	caused by	Explanation
RED	TRIAL	RED begins immediately after TRIAL begins.
STIMULUS	BARPRESS	STIMULUS begins when a BARPRESS occurs.
REINFORCE	BARPRESS CORRECT	REINFORCE begins when a correct BARPRESS occurs. (How correct responses are defined will be shown later.)
PUNISH	BARPRESS NOT CORRECT	PUNISH begins when a BARPRESS defined as NOT CORRECT occurs.

WAIT	TRIAL + 10S	WAIT begins 10 sec after TRIAL begins.
REWARD	TRIAL + 10S OR BARPRESS	REWARD begins 10 sec after TRIAL or following a BARPRESS, depending on which occurs first.
LAST	TRIAL DONE + 200M	LAST begins 200 msec after the event TRIAL is completed.

An event such as REWARD in the last example is limited to one activation. That is, REWARD will be triggered by either a BARPRESS or the passage of 10 sec from the start of TRIAL, but not both. In general, each event is in one of three states, as follows.

STATE	NOTATION (using event name REWARD)	DESCRIPTION
Not active	NO REWARD	The event is not active
Active	REWARD	Active and in progress
Done	REWARD DONE	Previously active and completed

The user may explicitly limit access to one event based upon the status of other events, as follows.

event name	caused by	Explanation
REWARD	BARPRESS IF NO WAIT	REWARD will begin when a BARPRESS occurs if the event WAIT has not yet been activated.
PAUSE	BARPRESS OR TRIAL + 10S IF WAIT DONE	PAUSE begins 10 sec after TRIAL or following BARPRESS only if the event WAIT has been activated and completed.

In addition to limiting access to events based on the status of other events, access can be limited by testing the values of variables. Variables, like events, are represented by names. However, the names represent a numeric value or a code associated with a particular stimulus or response. How values are assigned to variables is discussed later. In future examples, the names COUNT and NUMBER represent numeric variables.

The possible comparisons of variables to values are as follows.

NOTATION	MEANING
=	equal to
< >	not equal to
<	less than
< =	less than or equal to
>	greater than
> =	greater than or equal to

The user may also use AND or OR conjunctions in his expression. Finally, NOT may be used to invert an expression. NOT must immediately follow IF and applies to the entire expression that follows. Some typical examples are shown below.

event name	caused by	Explanation
REINFORCE	BARPRESS IF COUNT <= 20	REINFORCE occurs following BARPRESS if the variable COUNT is less than or equal to 20.
DELAY	TRIAL + 10S OR BARPRESS IF NEXT DONE AND NUMBER = 10	DELAY triggered by TRIAL + 10 or BARPRESS, whichever occurs first, only if the event NEXT is completed and the variable NUMBER equals 10.
CONFUSE	TRIAL IF NOT NEXT DONE AND COUNT = 10	The limiting condition is the opposite of that in the last example. That is, CONFUSE will occur following TRIAL if the expression (NEXT DONE AND NUMBER = 10) is false. Care should be taken to avoid confusion between NO, which modifies the following event name only, and NOT, which modifies the remainder of the expression.

The last column in a control statement, labeled "RESULTS IN," designates the actual operations that happen in the experiment. There are two types of functions if the CAUSED BY conditions are satisfied: operations such as turning on stimuli or assigning values to variables, and control instructions for repeating series of events or looping.

Operations. The operations fall into five functional categories, as shown in Table 2. For all of these operations, the operation code in Table 2 is followed by the name of a definition that must be supplied in the DEFINITION section of the program. As is the case for event names, definition names are provided by the user and can be up to 16 characters in length.

Whether a name represents an event or a definition is established by context. Each definition name must appear in the left column of the definition section, while each event name is defined when it occurs in the left column of the control statement section.

For each operation, Table 2 gives the operation code (e.g., TURN ON) that will appear in the RESULTS IN column of the control statement section.

event name	caused by	results in
TRIAL	START +5S	TURN ON LIGHT

LIGHT is a definition name and must occur in the definition section, when the computer prompts for definition of stimuli, as follows.

definitions				
stimuli name	relay #	time on	time off	# pulses
LIGHT	3			

In the definition section, the computer will prompt with the name of each functional class of operations

Table 2
Operations That Can Be Specified in the "results in" Column of the Control Statements and the Form of Corresponding Definitions in the Definitions Section

I Stimulus Control Operations				
A. Operations:				
TURN ON				
TURN OFF				
TURN ON/OFF				
TURN OFF/ON				
PULSE ON/OFF				
PULSE OFF/ON				
B. Definition prompt:				
name	relay #	time on	time off	# pulses
II Response Control Operations				
A. Operations:				
NOTICE				
IGNORE				
B. Definition prompt:				
name	switch #			
III A/D Input Operations				
A. Operation:				
TAKE				
B. Definition prompt:				
name	channel #	time between samples	# samples	
IV D/A Output Operations				
A. Operation:				
SET D TO A TO				
B. Definition prompt:				
name	channel #	axis (X or Y)		
V Assignment Operations				
A. Operations:				
SELECT				
CHOOSE				
CHOOSE CORRECT				
B. Computer prompts:				
1. For SELECT				
note: types are random, shuffled, or sequential,				
give one list value and its number of occurrences				
per line. maximum = 100.				
name	type	list	# occurrences	
2. For CHOOSE and CHOOSE CORRECT				
note: give paired list names, then paired list items,				
one pair per line.				
choose name 1	from name 2	list1	list2	
VI A/D Data Operations				
A. Operations:				
COMPUTE				
RECORD				
specified entirely through definition				
B. Computer prompts:				
note: computations beyond raw and average are				
signal mean, variance, maximum, minimum, latency				
to peak				
name	A/D name	computation	record during	
VII Simple Parameter Operations				
A. Operations:				
TIME				
COUNT				
TRACE				
SUM				
specified entirely through data definitions				
B. Definition form				
1. For TIME				
name	from name 1	to name 2	record during	
2. For COUNT				
name	event, response, or list name			record during
3. For Trace				
name	list name			record during
4. For SUM				
name	value name			record during

Table 2 Continued

VIII Loop Control
A. Operations:
LOOP
NEXT
TERMINATE
B. Definition form is not applicable.

and the headings. The user presses the return key without entering information to advance to the next category.

All examples given in this section of the paper show a small segment of the definition section, including the computer-initiated text in small letters and the segment of the control statement in which the definition is actually used. Definitions and control statements actually appear in different sections of the program.

Codes in the first four categories of Table 2 deal with stimuli and responses. The following example shows how a tone called BUZZER can be pulsed on and off 10 times (assuming that a buzzer is attached to Relay 3 on the interfacing box).

definitions				
stimuli name	relay #	time on	time off	# pulses
BUZZER	3	1000M	500M	10

control statements		
event name	caused by	results in
ALARM	PANICBUTTON	PULSE ON/OFF BUZZER

Not shown in Table 2 is the fact that a name can be used to refer to several stimuli or responses. For example,

definitions				
stimuli name	relay #	time on	time off	# pulses
TONE	7	3S		
REDLIGHT	11	1S		
GREENLIGHT	2	3S		
STIMULI = TONE, REDLIGHT, GREENLIGHT				

control statements		
event name	caused by	results in
TURNON	TRIAL + 500M	TURN ON/OFF STIMULI

In this example, all three stimuli (named TONE, REDLIGHT, GREENLIGHT) will be activated simultaneously by the instruction TURN ON/OFF STIMULI. As specified in the definition, REDLIGHT will turn off in 1 sec, while TONE and GREENLIGHT will end after 3 sec.

A switch closure or response will be ignored unless the instruction NOTICE is given.

definitions		
responses name		switch #
LEFT		4
RIGHT		2
BARPRESS	=	LEFT,RIGHT

control statements		
event name	caused by	results in
SETUP	TRIAL + 500M	NOTICE BARPRESS
REWARD	BARPRESS	TURN ON/OFF HOPPER

Following a NOTICE instruction, switch closures will result in activation of REWARD until either an event that results in IGNORE BARPRESS is encountered, or the loop in which the NOTICE instruction was given is terminated. (Looping will be described in the next section.)

A particularly powerful feature of the language concerns the two assignment statements shown in Part V of Table 2, SELECT and CHOOSE. The SELECT instruction allows a name to be assigned an item from a list. The "item" can be a time, value, stimulus, or response. For example,

definitions			
selection name	type	list	number
ITI	RANDOM	5S	1
		10S	2
		15S	1

control statements		
event name	caused by	results in
NEW	TRIAL + 2S	SELECT ITI
LATER	NEW + ITI	TURN ON LIGHT

In this example, the variable ITI will be given a new value each time the event NEW occurs. This value will be 5, 10, or 15 sec, with the value selected at random with replacement from one 5, one 15, and two 10s.

Another utilization of SELECT shows how a stimulus can be varied from trial to trial. A user wishing two red and two green stimuli selected at random in a block of four trials would include the following in his program.

definitions				
stimuli name	relay #	time on	time off	# pulses
RED	4			
GREEN	8			

selection name	type	list	# occasions
STIMULUS	SHUFFLED	RED	2
		GREEN	2

control statements		
event name	caused by	results in
NEWTRIAL	START	SELECT STIMULUS
		TURN ON STIMULUS

SHUFFLED is a random selection without replacement, in this case, from a set consisting of two red and two green stimuli. At the end of each four trials, the full set will again be available.

A SEQUENTIAL selection is similar to SHUFFLED, in that the items in the list are used one at a time. However, in this case the order of the item always remains the same as specified in the list.

Another assignment statement is CHOOSE. This instruction involves a pairing of two lists, such that the item in List 1 is chosen because it is associated with a value in List 2. To illustrate, the next example shows how a stimulus can be chosen on the basis of the value of a variable named COUNT.

definitions				
stimuli names	relay #	time on	time off	# pulses
RED	7			
GREEN	3			
BLUE	11			

choose			
choose name 1	from name 2	list 1	list 2
STIMULUS	COUNT	RED	5
		BLUE	10
		GREEN	

control statements		
event name	caused by	results in
TRIAL	START	CHOOSE STIMULUS TURN ON STIMULUS

If the variable COUNT is less than or equal to 5, STIMULUS = RED, if COUNT is greater than 5, but less than or equal to 10, STIMULUS = BLUE. For all higher values, STIMULUS = GREEN. The final item in List 1 is the value selected if the value in List 2 is not found (e.g., COUNT greater than 10). As is true of the SELECT instruction, the variables in List 1 and List 2 can be values, response names, stimulus names, or the results of a calculation. When the second list consists of values, the values should be ordered from lowest to highest, since the appropriate value is chosen based on a less than or equal to comparison. When the second list represents names of stimuli or responses, selection is based on equality only, since order has no meaning. Finally, a special form of CHOOSE, CHOOSE CORRECT, can be used when the item to be selected (List 1) is a response, as shown in the next example.

definitions				
stimuli name	relay #	time on	time off	# pulses
RED	7			
GREEN	12			
WHITE	3			

responses		switch #
name		
LEFT		4
RIGHT		6
CENTER		5

select name	type	list	# occurrences
LIGHT	RANDOM	RED GREEN	1 1

choose correct	choose name 1	from name 2	list 1	list 2
KEYPECK	LIGHT		LEFT RIGHT CENTER	GREEN RED WHITE

event name	control statements caused by	results in
TRIAL	START + 5S	SELECT LIGHT TURN ON LIGHT NOTICE KEYPECK
REWARD	KEYPECK CORRECT	TURN ON/OFF HOPPER
PUNISH	KEYPECK NOT CORRECT	TURN ON/OFF SHOCK

If the GREEN LIGHT is selected, the event REWARD will be activated by a LEFT KEYPECK or PUNISH will be activated by a RIGHT KEYPECK or CENTER KEYPECK.

In general, the events CHOOSE and CHOOSE CORRECT provide a simple-to-use implementation of select case programming structure (Dillon, 1979). Further, the actual "cases" are in lists in the definition section, so that an experimenter can easily modify particulars, such as the pairing of stimuli and correct responses in a program without otherwise affecting the scheduling of events.

Parts VI and VII in Table 2 show how A/D data and simple parameters are named, computed, and recorded. By default, the A/D data for each channel will be available for display at run-time, but will not be recorded. In addition, the signal average will be computed over trials and will be recorded at the end of the experiment. The user can modify or add to the list of default data computations. It is possible to compute and display the mean, variance, maximum, or minimum value of a signal. In addition, the sample number on which the maximum value is obtained (LATENCY TO PEAK) can be recorded.

In specifying data computation, the column RECORD DURING is used to specify when data should be transferred to a buffer for disk storage. If this column is left blank, the computation is available for display or use within the program, but is not recorded for later retrieval. Use of the words NOT NEEDED avoids computation of signal averages. The following example shows a signal is recorded only if no eye movement has occurred. It also illustrates the default computations that can be associated with any A/D event and how they are modified.

definitions			
A/D input name	channel #	time between samples	# samples
EEG	4	20M	100
EYES	12	50M	20

A/D data definitions

name	A/D name	computation	record during
eeg	eeg	raw	
eyes	eyes	raw	
aveeeg	eeg	average	KEEP
aveeyes	eyes	average	
EYEBLINK	EYES	MAXIMUM	

event name	control statements caused by	results in
TRIAL	START + 50	TAKE EEG TAKE EYES
KEEP	TRIAL DONE IF EYEBLINK < = 10	
NEXT	TRIAL + 10S	

In this example, the data of interest (AVEEEG) are recorded at event KEEP, but KEEP only occurs if EYEBLINK (the maximum signal obtained from the eyes) is quite small. It should be noted that the event TRIAL is not DONE until the A/D recordings are completed and the required computations are completed.

It is possible to define and operate on simple parameters, as well as analog data, as shown in Part VII of Table 2. The user specifies what type of parameter operation he requires (TIME, COUNT, SUM, or TRACE) and is prompted accordingly. As is also true for analog data, no mention of these operations is required in the control statements. The following examples show how two of these operations are used.

parameter operations—type = time, count, sum, or trace
type: TIME

name	from name 1	to name 2	record during
LATENCY	TRIAL	BARPRESS	
TOTALTIME	START	END	END

type: SUM

name	value name	record during
TOTAL LATENCY	LATENCY	END

In this example, the latency between the start of a trial and barpress is computed, but not recorded. It is, instead, placed in a sum called TOTAL LATENCY, which is recorded during event END.

The operations COUNT and TRACE are particularly useful in conjunction with SELECT and CHOOSE definitions, as the next example illustrates.

definitions			
select name	type	list	# occurrences
COLOR	RANDOM	RED	1
		GREEN	1
		BLUE	1
		YELLOW	1

parameter operations
type: COUNT

name	event, response, or list	record during
COLORCOUNT	COLOR	END

type: TRACE

name	list
COLORORDER	COLOR

With the COUNT instruction, the list of colors and the number of times each was selected are recorded during the event END. With TRACE, the new value selected from the list COLOR is recorded each time a selection is made from the list.

This completes our discussion of the various operations, except for a special class known as looping (Part VIII of Table 2).

Looping. Another term that could have been used for looping is "repeating." A simple example shows its importance.

event name	caused by	results in
LOOK	NEWTRIAL	TURN ON/OFF TONE TAKE EEG
ITI	LOOK + 10S	

The above event, LOOK, represents the heart (one trial) of an evoked-potential experiment in which A/D input begins (TAKE EEG) at the same time as a tone is sounded (TURN ON/OFF TONE). Normally, a user is interested in repeating such operations several times in an experiment. To accomplish 20 repetitions, the following is required (except for the optional addition KEEPON, to be explained later).

event name	caused by	results in
NEWTRIAL	START + 5S	LOOP 20
... LOOK	NEWTRIAL	TURN ON/OFF TONE TAKE EEG
... ITI	LOOK + 10S	NEXT NEWTRIAL
KEEPON	NEWTRIAL DONE	TURN ON TONE

In this example, NEWTRIAL names a compound event or loop that extends to the event ITI. The "body" of a loop always extends from the loop name (e.g., NEWTRIAL where the LOOP operation is given) to the event that contains the instruction NEXT (e.g., ITI). The latter instruction starts the loop again.

In the above example, LOOP 20 is a short form meaning LOOP while the number of loops is less than or equal to 20; the loop occurs 20 times before NEWTRIAL is DONE. That is, the compound event NEWTRIAL remains active until the test conditions or test clause associated with the loop instruction are met. After the 20 repetitions, the event KEEPON is activated by NEWTRIAL DONE. KEEPON need not be in the example. It is coded here to show that an event (KEEPON) can be made to happen when a loop is DONE. However, there is no requirement that the end of a loop lead to subsequent events; there is no "flow of control," as in normal programming. Thus, it is permissible to have concurrent processing of independent loops, as follows.

event name	caused by	results in
LOOPA	START + 500M	LOOP 20
...	...	...
... ENDA		NEXT LOOP A

```

LOOPB      START + 500M      LOOP 50
.
.
.
... ENDB
KEEPON     START + 30S      NEXT LOOP B

```

In this example, LOOP A and LOOP B begin together, are processed independently, are "DONE" independently, and have no further effect on flow of control. This capability is very useful in programming complex stimulus events and such operant procedures as concurrent schedules of reinforcement.

It is also possible to nest one compound event inside another. An example of this type of processing is contained in Table 1 and will be described later. Extensive error diagnostics are provided to make sure that no improper transfers take place while looping. For example, transfer into the middle of a loop, inappropriately bypassing a test of the conditions that must be met for the loop to be repeated, cannot be permitted. Thus, an error message occurs if any event or response is mentioned in the "caused by" column that does not belong to the loop.

In addition to counting the number of loops (e.g., LOOP 30), the expression that follows LOOP can test the status of other events (e.g., LOOP WHILE NO ALARM) or the values assigned to variables (LOOP WHILE COUNT > 50). The full set of operators described previously (=, <, <=, >, >=) can be used. Several test conditions can be specified, and these can be connected by AND or OR. Finally, the entire expression can be inverted by placing NOT immediately following WHILE. Some typical examples follow.

event name	results in	Explanation
TRIAL	LOOP WHILE NUMBER < 10 OR X = 5	Looping will continue while NUMBER is less than 10 or X = 5.
WAIT	LOOP WHILE NOT NUMBER = 10	Loop while NUMBER not equal to 10.
SESSION	LOOP WHILE NOT NUMBER = 10 AND NO TRIAL	Loop while NUMBER not equal to 10 OR TRIAL OR TRIAL DONE.

Finally, LOOP used alone results in continuous looping with no test conditions following NEXT LOOP. Such a loop requires a TERMINATE instruction, described next, to avoid endless looping.

The test clause following LOOP is only tested at the start of each loop. Consequently, a loop that requires 15 sec to complete might need to be terminated before the 15 sec are up. The next example shows a loop that terminates after 10 loops or after an alarm button is pressed.

event name	caused by	results in
NEWTRIAL	START + 10S	LOOP 10
... LOOK	NEWTRIAL	TURN ON STIMULUS NOTICE PANICBUTTON TERMINATE NEWTRIAL
... ALARM	PANICBUTTON	NEXT NEWTRIAL
... BACK	LOOK + 1S	
KEEPON	NEWTRIAL DONE	

The event ALARM only occurs if the response key named PANICBUTTON is pressed. If this happens, the NEWTRIAL loop is DONE and KEEPON is activated.

Run-time parameters. A last convention for the language should be mentioned. Researchers frequently have parameters of experiments that they would like to vary from day to day. For example, a researcher may wish to have 100 trials 1 day and 50 the next. In our language, this flexibility is obtained by replacing the parameter value with a question mark at the time the program is translated. For example,

event name	caused by	results in
TRIAL	START + ?	TURN ON LIGHT

At run-time, the above two lines are displayed for the user, who can specify the value he requires that day.

Example. We will now return to the sample experiment presented in Table 1 and briefly describe the operations involved. Lines 21-31 represent the control statements for the experiment. The event START is caused by depression of the start switch on the laboratory control system, and it results in a WARNING being pulsed on and off (Line 23). The event SESSION begins 15 sec after START and involves 15 repetitions of its "body," that is, the events between SESSION and PAUSE (Lines 24-29), where PAUSE results in NEXT SESSION.

The event TRIAL begins when each SESSION starts and is repeated 20 times (Lines 25-28). Thus, within each repetition of the SESSION loop, there are 20 repetitions of the TRIAL loop. Each TRIAL loop causes an event EVOKED in which a LIGHT is turned on, then off (Line 26), and EEG recordings are taken (Line 27). The event ITI occurs 10 sec following the beginning of the event EVOKED (Line 28), so that each TRIAL loop requires 10 sec. The event TRIAL is DONE when the required 20 loops are completed. The event PAUSE (Line 29) occurs 40 sec after TRIAL DONE and results in NEXT SESSION. Finally, FINISH (Line 30) is caused by the completion of the 15 SESSION loops, and the experiment is terminated.

The part of Table 1 labeled DEFINITIONS (Lines 3-20) fills in procedural details and indicates what data will be collected. As an example, WARNING (Line 7) is connected to Terminal 4 on the interface panel. It is pulsed three times with both on and off times of 500 msec. As another example, in Lines 8-12, the program specifies that the EEG recording consists of two channels of A/D input. OCCIPLOBE is the name of one of the channels, and it is connected to Terminal 3 of the A/D subsystem. For that channel, samples of the signal are taken every 1 msec until 200 samples are gathered. The type of A/D data summary required by the user is specified in Lines 13-20. Since lowercase text comes from the computer, most of Lines 13-19 represents default specifications. The user must fill in the RECORD DURING column in order to have data saved on disk storage. By leaving the column blank,

the user is specifying that the data will not be saved, although he can display the signal on a graphics display during the experiment by specifying PLOT OCCIPLOBE or PLOT TEMPLOBE.

Unusual features. This concludes the presentation of the user's language, which is characterized by two unusual features. First, use of "come here when" programming instead of "go to" programming emphasizes changes imposed by external events rather than flow of program control. Second, a great emphasis has been placed on eliminating "select case" instructions from the control statements, using lists of data and SELECT or CHOOSE instructions instead. That is, statements such as the following are unnecessary.

```
IF LIGHT = RED, THEN CORRECT BARPRESS
= LEFT
```

```
ELSE IF LIGHT = GREEN, THEN CORRECT
BARPRESS = RIGHT
```

```
ELSE IF LIGHT = WHITE, THEN CORRECT
BARPRESS = CENTER
```

Of course, this power must be achieved by making the software work harder. In our system, the program that sets up the experiment establishes flow of control, removing this burden from both the users and the real-time system that runs the experiment. This choice was not arbitrary. The real-time system, like the user, is continually meeting deadlines and operating under time constraints, so that efficiency is an important consideration. It is to this system that we now turn for a brief description.

Running Experiments

The run-time system reads in a previously defined file, asks the user to define any values that were indicated with question marks, writes out headers to the data disk (time, date, subject number, condition number, etc., under user control), and runs the experiment. A run-time panel allows the operator to change parameters as the experiment progresses. Analog information is displayed on an oscilloscope using a display module. Specific routines involved in performing these functions are briefly described: (1) A foreground SCAN system continually watches for various software flags to be set and collects statistics that indicate the amount of time the central processor is idle and the maximum lengths of a number of critical queues. (2) An interrupt service routine determines the interrupt sources and transfers control to appropriate device handlers. (3) The device handlers perform any needed input or output, reset the interrupt flag, and set any needed software flags. (4) A series of subroutines are called from the user program to perform the various operations, such as TURN ON, SELECT, and so on.

To run an experiment, the researcher must be familiar

with a small set of Teletype commands to display on a scope, and he must understand the functions performed by five switches mounted on a console and described previously (Dillon et al., 1979).

Keyboard commands are used to display data, by naming the data. To display two A/D curves simultaneously, the researcher types

```
P1 = NAME1
```

```
P2 = NAME2
```

He can move curves around for comparison purposes with commands such as

```
P1 UP 5
```

```
P2 LEFT 7
```

Except for I/O commands isolated in small sections of the device handlers, the run-time system is written entirely in MPL. By following this procedure, the user can produce code more quickly. In addition, as new developments in hardware become available, we will be able to recompile for the newer machines. We now plan to convert to the Motorola 6809 as soon as it is available.

CONCLUSION

We have described the language in which a user specifies experiments to be run on our microcomputer-based laboratory control system. We have also indicated some aspects of the run-time system that controls experiments, and we have briefly described some associated software, for example, a file transfer program.

The user language is high level and application oriented. By high level we mean that the programmer is not concerned with machine details, such as word sizes, number of registers, assembler syntax, and so on. Accordingly, the usual pain of low-level language programming is lessened. By application oriented, we mean that the language was specifically designed for psychology laboratory experiments. This can be seen in the use of the terms "STIMULUS" and "RESPONSE" in defining the digital input and output to the interfacing, and in the names used for both events and definitions in the examples. Note, however, that names are selected by the user, so that someone writing a program to control the rollers in a steel mill would use terminology familiar in that environment. Thus, we have defined a language that recognizes the specialized terminology of various applications and allows specification of control in a very natural readable manner.

The major limitation at present is that the language does not have instructions allowing the display and recognition of alphanumeric material. Expansion in this

direction requires development of string-handling capabilities so that the programmer can CHOOSE, SELECT, ACCEPT, or DISPLAY words or sentences. These capabilities will be included in an expanded version of the language.

REFERENCE NOTE

1. Bowles, K. L., & Hollan, J. D. *An introduction to UCSD PASCAL: A (nearly) machine-independent software system for micro- and minicomputers*. Paper presented at the eighth annual meeting of the National Conference on the Use of On-Line Computers in Psychology, San Antonio, November 1978.

REFERENCES

- DILLON, R. F. Structured flow of control in a laboratory control language. *Behavior Research Methods & Instrumentation*, 1979, **11**, 234-240.
- DILLON, R. F., MILLMAN, B., TOMBAUGH, J. W., FERGUSON, W. R., & BEZANSON, W. R. A microcomputer-controlled laboratory: Hardware. *Behavior Research Methods & Instrumentation*, 1979, **11**, 293-300.
- MOTOROLA SEMICONDUCTOR PRODUCTS, INC. *MPL language for the M6800 microprocessor*. Phoenix, Ariz: Author, 1976.
- SOUTHWEST TECHNICAL PRODUCTS CORPORATION. *Flex user's manual*. San Antonio, Tex: Author, 1978.
- TECHNICAL SYSTEMS CONSULTANTS. *Flex advanced programmer's guide*. West Lafayette, Ind: Author, 1978.