COMPUTER TECHNOLOGY

On-line FORTRAN for the PDP-8¹

RALPH NORMAN HABER, STEVEN H. BARRY, and THOMAS UHLMAN,²
UNIVERSITY OF ROCHESTER,
Rochester, New York 14627

Generally, writing programs for most small data acquisition computers is difficult and time consuming. In part, this has been due to the unavailability of interpretive languages, since they have been unusable when many special input-output operations are needed. Recent changes in the PDP-8 family FORTRAN compiler permit such experiments now to be written in FORTRAN, at a great saving of programming and debugging time, while gaining substantial power in terms of logical control. A typical experiment is described in some detail, with an event-time flowchart of its operation. A flowchart of a FORTRAN program and the program itself are presented to illustrate how the system accomplishes tasks necessary for the design. A brief description is given of the interface components needed for the experiment. Both the specific program described and the concepts illustrated are applicable to other experimental paradigms.

One drawback of small data acquisition computers for on-line experimentation has been the difficulty of writing programs quickly and easily. Because such experiments inherently need specialized input-output instructions, interpretive languages (e.g., FORTRAN) have not been particularly useful. This difficulty has led many Es to shy away from computers for research control and has also greatly reduced their productivity even after such computers are in the laboratory.

A recent small compiler modification in DEC's version of the 4K-FORTRAN for the PDP-8 family computers now permits transfers from FORTRAN to DEC's assembly language or to machine language for subroutines to carry out processing of a special nature (e.g., a software character-generator program for a CRT) or for subroutines to issue and process specialized input-output operations. Whenever a PAUSE statement is followed

by a decimal number, the program will now treat this as a jump to a subroutine beginning at the octal equivalent of that address. A link is established at the beginning of the subroutine, so at its end, program control is returned to the next statement in the FORTRAN program following the PAUSE.

The relatively small change in the compiler makes the very powerful FORTRAN logic and arithmetic capability available to on-line users. It further permits programs to be written and operated without the extensive special training usually required. Thus, the computer becomes a tool with which to do other things, rather than the most complex and time-consuming component in the laboratory. Nelson, Barry, and Perrott (1969), following Doorne and Sanders (1968) have specified several criteria that must be met if the small-data computer is to justify its place in the experimental laboratory. Programming ease is a central one.

The full import of this development of FORTRAN control capabilities is illustrated by comparisons between programs written in PAL/MACRO and FORTRAN on the bases of programmer time needed, ease of user access, logical power, and ability to extend the program to other experimental designs. Even inexperienced computer users spend relatively little time in program development when introduced to the combined FORTRAN and modularized MACRO structure discussed below. Once written, the program can be easily used by inexperienced people with but a few

minutes' instruction. The inherent power of the easily understood and English-language-based FORTRAN logic makes possible one-statement specifications of logical functions requiring tens or even hundreds of statements in the analogous PAL/MACRO form. Finally, the program structure discussed below has already been extended to four other experimental designs of very different nature (one of which used the computer to produce tones of varying frequency) with conversion times of between 2 and 4 h.

This paper presents a description of a basic FORTRAN program written to operate a three-field backward-masking problem in vision, using a 4K PDP-8 computer and some modular machine-language subroutines. While the description in this paper is very specific, the general format of the program is applicable to many areas of psychology using widely disparate experimental paradigms. Specifically, the program illustrated here is prototypic of experiments that involve control of multiple events in real time. From the computer's vantage point, it matters not whether the events are lights, as in the present example, or gates on a runway, or reinforcement contingencies, or a communication network. All that changes are the connections made from the interface to the specific apparatus components to be operated.

Figure 1 illustrates the timing sequences for an experiment in backward masking. A S is dark adapted and is told to fixate a dimly lit fixation point on an otherwise dark background. When he initiates a trial

Fig. 1. The temporal sequence of three channels for a visual backward-masking experiment.

Fig. 2. A time-event chart for a general three-field visual backward-masking experiment.

intertrial interval, the teleprinter types out which buttons were pressed and their latencies in milliseconds from the offset of the last light in the stimulus sequence. This entire record occupies one line on the teleprinter. The values for the next trial are then typed out and the sequence is repeated. Up to 1,000 trials can be run during a 1-h session.

To run a session, the computer is turned on, the program loaded along with the FORTRAN operating program, and the proper paper tape for the trials to be run placed in the paper-tape reader. When the program is started, it asks for several overall constants for the session (e.g., tape number, end of tape number, intertrial interval), waits for the S's startswitch, and then begins.

Figure 2 illustrates a flowchart of the program for this experiment. Not all of the fine details are included in the flowchart (such as the formatting for the printing of the trial parameters, output printing, or on-line data analysis). This particular version of the program used the teleprinter and keyboard with the trial parameters punched on paper tape and read in trial by trial. Other input arrangements are easily specified.

This experiment uses a three-channel tachistoscope interfaced to a PDP-8. Some of the principles of the interface are described in Haber (1968) and will be expanded briefly here. The lights are operated by a jam-transfer buffer (see Fig. 3, left panel) in which each of the three bits of the accumulator is connected through a pair of AND-gates (opened by a particular output pulse generated by an output instruction) to a flip-flop which in turn is connected to a driver and then through a cable to one of the three channels of the tachistoscope. The choice of light is determined by the number in the accumulator at the time the output instruction is executed by the program. This interface permits any combination of lights or other events to be operated simultaneously and does not require that one light be turned off before the next is turned on. Its advantage is that the selection of the output devices to be operated is determined by software, rather than by extensive and expensive hardware selection components in the interface. While only three channels are shown, up to 12 events can be selected independently by the single output instruction. The events themselves can be altered by unplugging this cable from the interface and substituting another one connected to a different apparatus.

by depressing a startbutton (actually a footswitch), a sequence of one, two, or three superimposed flashes are presented. These display stimuli are capable of masking each other under appropriate temporal asynchronies. Varied on each trial are the lights to be turned on, with their durations and the temporal asynchrony between them specified. The S's response is to specify (by pushing the corresponding button) which of the one or more lights presented he saw. He is required to press the buttons as quickly as he can in whatever order he wishes. If he saw no lights, he is instructed to depress the fourth button. The latencies of each buttonpress are recorded during a fixed intertrial interval during which he may respond.

The experiment consists of a sequence of such trials. The exact specification of the parameters of each trial can be determined prior to the beginning of the experiment and stored in memory, or contained on paper or magnetic tape and read under FORTRAN program control just prior to the beginning of each trial. A trial-by-trial printout of the parameters and the responses can be obtained on the teleprinter, or these can be stored in memory or on tape and printed later. Data can be analyzed on-line or later. When a trial-by-trial record is printed, the values of each parameter for the current trial are typed out. The lights are displayed after S presses the startbutton. The S's responses are recorded, and at the end of the

Fig. 3. A schematic of the interface needed to operate this experiment, divided into an output control for lights, an input control for buttons, the device selectors, and a pulse-generating control. DEC flip-chip components are illustrated but pin numbers are deleted for clarity.

The buttons are sensed by an input buffer, somewhat as a mirror image of the output system. It requires a button, which, when pressed, makes a number of contacts (e.g., eight would permit differentiation between 256 buttons into the same buffer) plus a strobe contact, common to all buttons, which must be made after the others have settled. (We use the Microswitch KB series for this purpose.) When any button is pressed the strobe contact closure is sensed and integrated by a Schmitt trigger which sets a button flag flip-flop. This flag is gated through to the skip bus where it will cause the next instruction in the program to be skipped if the gate is opened by an input instruction (INBUT, 6611) issued in a button-sensing subroutine. The contact closures on each button are selectively wired in a binary code (e.g., Button 1 would connect only Contact 1, Button 2 only Contact 2, Button 3 only Contacts 1 and 2, and so forth, for as many contacts as are available). If the buttons are checked as soon as the INBUT instruction produces a skip, then no buffer is needed, since the button code can be decoded while the S still has his finger depressed. To permit later checking, a flip-flop buffer is needed to store the code before it is strobed into

the accumulator. Figure 3 (center-left panel) illustrates a three-bit input buffer (capable of sensing up to 7 buttons) with a storage buffer. If storage is not needed, then the flip-flop buffer and the AND-gates just below it are not needed.

When the program learns from the skip facility that a button has been pressed, it can proceed to load the accumulator with the unique number of the particular button pressed. Without the storage facility, all that is needed is to issue the input instruction, LODBUT, which opens the AND-gates from the button to the accumulator.

Real time is measured by a pulse-generating clock, set to issue a pulse every millisecond. When the clock is turned on (by another output instruction) the pulses are sensed in the skip bus. They can then be counted or accumulated under either subroutine or FORTRAN program control (see Fig. 3, right panel).

Table 1 contains a FORTRAN program for this experiment with comments on the functions of most of the instructions. To conserve space in this paper, instructions are omitted concerning printing of the data sheet, the automatic-pacing option, the option to read trial parameters and store data on magnetic tape, and procedures for

data analyses. Since these involve rather standard FORTRAN procedures and instructions, no novel coding is required. The program in Table 1 looks even longer than it is, since many of the instructions are normally combined onto one line, but have been separated here for expository purposes. All numbers are expressed in integer format. While this is mandatory for all numbers used in subroutine input-output processes, floating-point expressions may be used in the conventional manner within FORTRAN.

Table 2 presents the five input-output subroutines addressed by the PAUSE statements, along with some comments to explain briefly their function. These subroutines could be used with this interface (see Fig. 3, center-right panel) for any type of experiment involving up to 12 output events and up to 256 differently coded input devices.

It should be obvious that this FORTRAN program can be modified for a large variety of experiments. The simplicity of the subroutines and the ease with which other subroutines can be added makes it possible to write programs for on-line experiments with little special programming knowledge of a data-acquisition computer. The programs

are short and easy to debug, and easily taught to neophyte programmers or Es.

Guidelines concerning the relative lengths of FORTRAN programs are hard to use. While the FORTRAN program in this example itself is short, the FORTRAN operating program, which also has to be in core, takes up most of the space. To determine the length, and the amount of free core left for PAL/MACRO subroutines, the FORTRAN Symbolprint program is run immediately after compilation. It will list the locations of all dimensions and indices, the address of each numbered statement, and the lower and upper bound of unused core memory. It is into this latter space that subroutines are placed. The other information is also necessary since the subroutines usually make reference to the location of some of the dimension statements, so that trial values can be transferred between FORTRAN and the subroutines.

The particular FORTRAN program illustrated in this article, but including formatting statements to print the data sheet and the automatic pacing option, leaves 774₁₀ memory locations free. Of these, the five subroutines in Table 2 require 40₁₀. If substantial on-line data analyses are required, more of the remaining free core would be taken up by the FORTRAN program expansion. With the use of the Symbolprint program, one can determine how to juggle the relative sizes of the FORTRAN and the subroutines to fit the available space.

The program illustrated in this paper uses real-time features, but those are entirely controlled by the PAL/MACRO subroutines. Since one cannot determine ahead of time the execution time of FORTRAN statements, it is difficult though not theoretically impossible, to use FORTRAN alone for this purpose. As a rough rule of thumb, FORTRAN statements take about ½ msec to execute (as compared to .003 msec for a typical PAL/MACRO instruction). In this example, FORTRAN statements control the logical flow of the program. Whenever input or output operations or precise timing is needed, then the PAUSE facility is utilized.

To provide some information on the temporal interaction of FORTRAN statements and the PAL subroutines, several loops in the program have been measured. For example, Statement 6 and the unnumbered statement following it

Table 1
A FORTRAN Program for a Backward Masking Experiment

END = 0	C; Part I: Initialize constants and variables	Trial number of last trial on tape
URTR = 0		Intertrial interval—time for S to respond
EVENT = 0		Trial number of current trial
LITE = 0		Number of events in current trials
ITIME = 0		Code of event to be displayed
IBUTTON = 0		Number of milliseconds event is to be displayed
ISBUTT = 5		Number of button S pressed for his responses
Dimension IDURAT (5)		Start Button is number 5
		Five locations to store up to 5 durations for current trial
Dimension LTCODE (5)		Five locations to store up to 5 codes for the events for current trials
Dimension JBUTT (4)		Four locations to store up to 4 button responses made for current trial. Buttons are coded as 1-4.
1; Format (I)		An integer format is defined
2; Accept 1, IEND, ITI		Read in from the TTY keyboard the last trial number and the intertrial interval to be used
	C; Part II: Get values for current trial and test for Start button	
3; Accept 1, ICURTR, IEVENT		Read from paper tape the current trial number and the number of events in trial. These will also be typed on teleprinter
4; DO 5 K = 1, IEVENT		
Accept 1, LTCODE(K), IDURAT(K)		For each event, read in the code to operate event and the duration of the event. These are stored in the Dimensions previously established.
5; Continue		
6; Pause 3456		Jump to button subroutine at octal 6600 (= decimal 3456) to see if subject pressed the start button.
IF (IBUTTN-ISBUTT) 6, 7, 6		If no button is pressed, or one other than number 5, go back and listen some more. If button is number 5, go to 7.
	C; Part III: Presentation of stimuli	
7; DO 9 J = 1, IEVENT		For each event in sequence get code and duration of event.
LITE = LTCODE (J)		
ITIME = IDURAT (J)		
Pause 3472		Jump to subroutine to execute event.
Pause 3488		Jump to subroutine to count out the duration of the event.
IF (J-IEVENT) 9, 8, 9		If all events are not completed, do loop again.
8; LITE = 0		If last event is done, then turn last event off (set light code to 0).
Pause 3472		Jump to subroutine to execute event.
9; Continue		
	C; Part IV: Measure and record reaction time	
DO 10 L = 1, 4		Set the value of the four button storage locations to 0, in preparation to store next responses.
JBUTTON (L) = 0		
10; Continue		
IRT = 0		Set reaction time counter to 0.
11; Pause 3504		Jump to subroutine to start a one millisecond time count.
Pause 3456		Jump to subroutine to check to see if S pressed any buttons.
IF (IBUTTN) 12, 13, 12		If no button has been pressed (e.g., IBUTTN = 0), jump to 13, otherwise go to 12.
12; JBUTTN (IBUTTN) = IRT		Store the value of reaction time counter in the register for the button pressed.
13; IRT = IRT + 1		Increment reaction time counter by 1.
IF (IRT-ITI) 14, 15, 15		Has enough time been given to S to respond? If not, go to 14; if yes, to 15.
14; Pause 3520		Jump to subroutine to wait out rest of millisecond. Return when millisecond is up.
Go to 11		Loop through response routines for another millisecond.

Table 1 (Continued)

15; DO 16, M = 1, 4	C; Part V: Type results
Type 1, JBUTTN (M)	Type the reaction times in integer format for
16; Continue	when each of the four buttons was pressed.
	Zeros will be typed for each button not
	pressed.
17; FORMAT (/)	Teleprinter will line feed and carriage return
Type 17	when this format addressed.
IF (ICURTR - IEND) 3, 18, 18	Has last trial been run? If not, go to 3 for next
	trial. If yes, go to 18.
18; FORMAT ("END OF TAPE")	Type out "end of tape."
Type 18	
Go to 2	Jump to beginning to be ready to read in a new
END	tape.

Table 2
Input-Output Subroutines Used with FORTRAN Programs

Location	Contents	Symbolic Coding	Comments
BUTTON Listening Subroutine			
6600	0000	Button, 0000	/ Linking address when subroutine is entered. Address of next FORTRAN statement is stored here.
6601	7200	CLA	/ Clear accumulator.
6602	3613	DCA I IBUTTN	/ Deposit zeros in IBUTTN
6603	6612	BUTT?	/ Was a button pressed? Skip next instruction if it was.
6604	5600	JMP I BUTTON	/ No, exit subroutine by jumping to address stored in 6600.
* 6605	6604	BUTBCR	/ Yes, clear button buffer.
* 6606	6614	BUTBLD	/ Load button buffer.
6607	6601	BUTT IN	/ Load accumulator with button code.
6610	3613	DCA I IBUTTN	/ Deposit button code in IBUTTN, available to FORTRAN.
6611	6611	BUTCLR	/ Clear button sense flag.
6612	5600	JMP I BUTTON	/ Return to FORTRAN.
6613	7565	IBUTTN	/ Octal address of IBUTTN, as given in symbol print of FORTRAN program.
LIGHT Operation Subroutine			
6620	0000	LIGHT, 0000	/ Linking address.
6621	7200	CLA	/ Clear accumulator.
6622	1626	TAD I LITE	/ Load accumulator with code for current event.
6623	6602	DISPLAY	/ Turn on light through jam transfer in interface. See Figure 3 for interface.
6624	7200	CLA	/ Clear accumulator again.
6625	5620	JMP I LIGHT	/ Return to FORTRAN.
6626	7567	LITE, 7567	/ Octal address of LITE.
DURATION Subroutine			
6640	0000	DURAT, 0000	/ Linking address.
6641	6631	CLKON	/ Turn on clock, which will give a pulse every millisecond.
6642	6622	CLKCLR	/ Clear clock flag—ignore all previous pulses from clock.
6643	6621	CLKSKP	/ Has one millisecond elapsed?
6644	5243	Jump back 1	/ No, wait some more.
6645	2652	ISZ I ITIME	/ Yes, add one to ITIME. Is it now zero?
6646	5242	Jump back 4	/ No, continue another millisecond.
6647	6624	CLKOFF	/ Yes, turn off clock.
6650	6622	CLKCLR	/ Clear clock flag again.
6651	5640	JMP I DURAT	/ Return to FORTRAN.
6652	7566	ITIME, 7566	/ Octal address of ITIME.
Millisecond Start Subroutine			
6660	0000	MSSTRT	/ Linking address.
6661	6631	CLKON	/ Turn clock on.
6662	6622	CLKCLR	/ Clear clock flag.
6663	5660	JMP I MSSTRT	/ Return to FORTRAN.
Millisecond Finish Subroutine			
6700	0000	MSFNSh	/ Linking address.
6701	6621	CLKSKP	/ Has 1 MSEC elapsed?
6702	5301	JMP Back 1	/ No, wait some more.
6703	6624	CLKOFF	/ Yes, turn off clock.
6704	6622	CLKCLR	/ Clear clock flag.
6705	5700	JMP I MSFNSh	/ Return to FORTRAN.

*Delete if button buffer is not used.

form a loop during which the startbutton is awaited. If no button is pressed, then these two statements cycle indefinitely. The duration of the loop is 1.42 msec, of which .01275 msec is spent in the subroutine. Nearly all of this time is consumed by the IF statement, because of its several logical tests.

Another more complex example is the longer loop to count reaction time. It begins with Statement 11 and ends with the statement following Statement 14. This loop, consisting of three PAUSE, two IF, two Equality, and one Go To statement takes 4 msec, exactly, to execute. Of that time, the subroutines require .01275 msec to sense the absence of a button, .015 msec more if a button was pressed, .012 msec to start the time count, and .01725 msec to end the time count—a maximum total of .057 msec. From Statement 11 to Statement 14, between 3 and 4 msec elapse. Since the subroutine to end the time count will signal exactly when a msec has elapsed, this time is rounded to exactly 4 msec. Thus, reaction time in this program is counted in units of 4 msec. It could be reduced to a finer unit if necessary by changing the order of the statements.

The program listed here, even its abbreviated form, will run as is. It requires a PDP-8, 8/S, 8/I, or Lab-8, with the basic 4K-size memory. With only very slight modifications it could operate on any other DEC computer. The interface, including a power supply and mounting panel needed to run this experiment, or any like it using these buffers, costs less than \$800 plus, of course, the tachistoscope, the response buttons, and the cable.

REFERENCES

- DOORNE, H. V., & SANDERS, A. F. PSARP, a programable signal and response processor. *Behavior Research Methods & Instrumentation*, 1968, 1, 29-32.
- HABER, R. N. An on-line computer in a visual perception laboratory. *Behavior Research Methods & Instrumentation*, 1968, 1, 86-93.
- NELSON, M. A., BARRY, S. H., & PERROTT, D. R. Automated system for acoustic research. *Behavior Research Methods & Instrumentation*, 1969, 1, 298-299.

NOTES

1. This development was supported in part by a research grant from the United States Public Health Services (MH 10753) and a grant to purchase the computer from the National Science Foundation (GB 4547) to the first author.
2. Now at the University of North Carolina.