SESSION X

SYMPOSIUM: PROGRAMMING LANGUAGES FOR COMPUTER-AIDED EXPERIMENTATION WITH MEDIUM-SCALE COMPUTERS

N. John Castellan, Jr., *Indiana University*, President

A general experiment programming system for the IBM 1800*

DAVID KIERAS

*Human Performance Center, University of Michigan
Ann Arbor, Michigan 48104*

A system is described that meets some user requirements of programming ease, general applicability, and simultaneous multiple S and laboratory operation. The system is based on an extremely flexible time-sharing real-time monitor and a user-level task-oriented programming language which together free the user from all multiple S bookkeeping programming. The conceptually simple language consists of simple commands for operations at the level of experimental procedure, such as displaying stimuli, collecting and timing responses, providing time delays, and recording data. Other commands support string manipulation, arithmetic, and disk I/O. The system is programmed only for the IBM 1800; however, it represents a successful approach to laboratory computerization.

This paper describes the General Experiment Programming System (GEPS) for the IBM 1800 computer facility at the University of Michigan Human Performance Center. A first version of GEPS saw heavy use for about a year; however, the examples in this paper are coded for the second version of GEPS, which is in the final stages of implementation. The difference between the two versions is a matter of several improvements that do not affect the basic philosophy of the system.

The computer installation, which is intended to allow several Ss and laboratories to be run at once, is based on an IBM 1801 processor-controller consisting of a CPU, 16K of 16-bit words, several data channels, and digital I/O devices. Peripherals include a console typewriter, card reader/punch, and a disk storage drive. The main laboratory is equipped with five S stations, each of which has a keyboard display unit and a set of pushbuttons. A sixth keyboard display is used as an on-line monitoring and control station by the E. The keyboard displays were IBM 2260 units; currently we are converting to ADDS MRD 700 devices. Other laboratory facilities, e.g., a tachistoscope, will be added in the near future.

The remainder of the paper is in three sections. The first discusses the various user requirements that the author constructed GEPS to meet. The second summarizes the GEPS system organization, and the third describes the GEPS real-time programming language.

USER REQUIREMENTS

The primary purpose of GEPS is to render the computer easily usable by the many researchers at the Human Performance Center who are characterized by limited computer experience, a broad range of research interests, and a large volume of experimental work. Meeting the needs of this group requires a very general programming system that is simple enough to learn and use that the researcher can quickly and effectively program his own experiments. The IBM-supplied programming facilities are inadequate: assembler language programming is too esoteric and time consuming for most users, while the FORTRAN-based process control system for the 1800 is difficult to use, slow in execution, and not particularly suitable for programming psychological experiments. For example, effective use of the keyboard displays often requires character string manipulations, which FORTRAN can perform in only a crude fashion.

Running several Ss and laboratories at a time has the important advantages of saving much time, effort, and money. However, a difficult programming problem is involved in the simultaneous processing of multiple independent Ss and laboratories. The user should be thoroughly insulated from this problem: he should be able to write an experiment program as if there were only one S. The system could then execute the program independently for each S, performing all multiple S bookkeeping automatically.

A final requirement is that, since future needs cannot be accurately anticipated, the system must be easily modifiable and not be unduly committed in its basic structure to any particular types of experiments.

Fig. 1. Overall organization of GEPS.

GEPS SYSTEM ORGANIZATION

The overall system organization is shown in Fig. 1. The user writes a GEPS language experiment program which is translated and stored on the disk by the GEPS language assembler. The user also writes an off-line FORTRAN program to load stimulus material on the disk before an experiment run, and FORTRAN dumping subroutines to recover data from the disk after the run. The GEPS monitor, controlled from the computer console, executes the experiment program, performing I/O only with the S stations and disk storage. This organization permits FORTRAN to be used to good advantage for the off-line computational work such as randomization and numeric data handling, while the real-time programming is done in the task-oriented GEPS language.

A brief description of the internal organization of the GEPS monitor will clarify the system philosophy. The system can run several Ss on the same program, each S being at a different point in the program, or Ss can be on different programs; any mixture is permissible. The simple table structure underlying this flexibility is shown

in Fig. 2. Each S has a S table which is part of a closed S table chain and a variable data area. Information that is particular to an individual S is kept in his table and data area, while the program text can be shared. This program sharing requires that the program be in reentrant (read-only) form, not being modified in the course of execution.

Each S table corresponds to a particular S station equipped with particular I/O devices. However, the stations being grouped together into different laboratories is of no consequence, since the system functions at the level of individual Ss, not groups of Ss. Thus, the provisions for independent multiple S running also provide for independent multiple laboratory operation.

A GEPS language program at run time consists of a series of command codes each followed by parameters which refer to either constant information contained in the program text or variable information in the variable data areas. The program is executed interpretively under the supervision of the GEPS monitor dispatcher routine (Fig. 3), which continuously searches the S table chain for a S needing processing. Upon locating a S table, the dispatcher sets a pointer which indicates to the rest of the monitor which S is being processed. If the S needs service, the dispatcher obtains the code for the next command via the pointers in the S table and calls the appropriate command service subroutine. The subroutine obtains parameter information from the program text, performs the actual command function, and returns control. The dispatcher resumes its search with the next S in the chain, eventually returning to each S. Thus, the GEPS monitor essentially time shares between Ss. Since each S's processing is done

Fig. 2. Table structure of the GEPS monitor.

concurrently with, and independently of, the others, processing of a S may be initiated or terminated at any time without interfering with the processing for other Ss.

DESCRIPTION OF THE GEPS LANGUAGE

A GEPS language program consists of a set of definitions followed by a series of command statements which are executed sequentially. The nonexecutable definition statements identify user-defined symbols and their attributes. The possible attributes are character string, substring, or numeric, and either variable or constant. Character strings are contiguous blocks of core storage with two character positions per word. A substring is a subset of another string starting at a specified character position and extending a specified length. Numerics are conventional binary integers. The command statements consist of an optional label, a command key word specifying the function to be performed, and a list of function parameters which may be the defined names of variables or constants, character or numeric literals, or the labels of other statements.

Consider the introductory sample program in Fig. 4. This simple program displays the message "push a button!" to the S, waits for a response, records response latency on a disk file, and repeats this process 10 times. This one program can be executed concurrently for many Ss.

Fig. 3. Dispatcher routine operation.

PROGRAM 1 SIMPLE EXAMPLE

```

SV MESSAGE 6
SV RT 2
NV COUNTER
NV RESPONSE
SC MSG 'PUSH A BUTTON!'
NC #TRIALS 10
START
ERASE
COUNTER=#TRIALS
MOVE MSG MESSAGE
@LOOP DISPLAY MESSAGE 1240 2400
DI.READ.WT RESPONSE
LATENCY 10 RT
DATA.OUT 2 RT @EOF
ERASE
LOOP COUNTER @LOOP
STOP
END
  
```

Fig. 4. Introductory coding example.

The first seven statements of the program define the program number, two string variables (SV) whose length in characters is specified, two numeric variables (NV), a string constant (SC) containing the message, and a numeric constant (NC) having the value 10. The START statement marks the beginning of executable commands, the first of which erases the S's display screen. The MOVE command copies the constant MSG into the string variable MESSAGE, which the DISPLAY command writes on the display screen using the literal numbers as before and after cursor positioning parameters. The next statement is read as "wait for digital input"; no further processing occurs for this S until he presses a button; meanwhile, the system continues to process other Ss. Any S input unrequested by a program is simply thrown away by the system. When the S responds, a code number (standard for all S stations) is placed in RESPONSE identifying which button was pressed. The LATENCY command computes the time in milliseconds since the last output to the S and the last input from the S, divides it by the scale factor 10 to obtain centiseconds, and places the result in the variable RT, which the DATA.OUT command then writes on Disk File 2. The LOOP command counts down the variable COUNTER, dropping through when it reaches zero, otherwise branching to the statement labeled @LOOP (all statement labels begin with an @ sign). When the STOP command is executed, processing terminates for the S.

A highly detailed description of the available commands is not necessary since they are essentially ad hoc, being defined to meet a priori obvious needs and needs that have arisen in the course of implementing particular experiments. The highly modular and basically simple design of GEPS makes adding a new command a

```

PROGRAM 2 ILLUSTRATE STRING COMMANDS
SV INPUT 8
SV REPLY 22
SS REPLY SLOT 13 9
SC MESSAGE 'WHAT'S YOUR NAME?'
SC DAVE 'DAVE'
SC REPLY.TEXT 'YOUR NAME IS'
SC YES 'YES'
NV N
START
MOVE REPLY.TEXT REPLY
@ASK BLANK INPUT
      BLANK SLOT
      DISPLAY MESSAGE 1200 1400
      TV.READ.WT INPUT
      ERASE
      SCAN INPUT DAVE N
      MATCH N 0 @DAVE.NAME
      MOVE INPUT SLOT
      MOVE '? SLOT(9,1)
      COMPRESS SLOT
      DISPLAY REPLY 1200 1400
      TV.READ.WT INPUT
      COMPRESS INPUT
      MATCH INPUT YES @ASK
@DAVE.NAME STOP
      END

```

Fig. 5. Use of the string commands.

relatively minor matter, requiring only the writing of the new command service subroutine in 1800 assembler language and the insertion of a reference to the new command into two tables.

However, the general types of commands defined in GEPS can be briefly described. S I/O commands are used to communicate with the S by means of the keyboard displays and digital I/O. Disk I/O commands implement two types of buffered disk access. The first is conventional direct access; the second supports a sequential file organization with variable record lengths. Execution of the DATA.OUT command writes the designated string, with an identification header, into the next available space in the named disk file. This facility, by allowing data to be recorded for several Ss on the same disk file, helps minimize disk access arm movement. The off-line data dumping programs sort the resulting data mosaic to obtain individual S data. Privileged commands, unlike normal commands, are able to refer to a variable data area or S table belonging to a S other than the one currently being processed. These commands are usually used in a monitor station program which enables the E to control and monitor the S stations. The monitor station is treated exactly like a S station by the GEPS monitor; the difference in function is defined only by the user programming. Program flow commands include commands for looping, branching, subroutine calling, program chaining, interval timing, and stopping. Arithmetic commands compare numbers, generate random numbers, and compute response latencies. In the first version of GEPS, arithmetic took the form of separate commands for basic arithmetic

operations; the second version accepts FORTRAN-style arithmetic statements. A subscripted string name in an arithmetic statement provides the equivalent of a FORTRAN array. String manipulation commands scan, match, move, blank, and compress strings. Subscripting a string name in one of these commands can be used to define a substring. String-numeric communication commands in the first version were required to transfer the contents of a string field to a numeric field, and vice versa. The second version allows using string and numeric references interchangeably. Other commands convert binary numeric values to the character code equivalents, and vice versa.

Two more coding examples will be presented. Figure 5 illustrates the string commands. The program asks the S for his name and echoes it back in the form of a question. If the S answers "YES," the program stops; otherwise the process repeats. If the S gives any part of his name as "DAVE," the program terminates immediately. SS stands for substring—SLOT is defined as the subset of REPLY starting at Character Position 13 and extending nine characters. The program first initializes REPLY, SLOT, and INPUT. Then it displays the question and waits for a response, filling INPUT with no more than the first eight characters the S types on his keyboard. The scan command searches INPUT for the string DAVE and sets N to the character number where DAVE appears as a substring. If DAVE is not found, N is set to zero, a condition tested for by the MATCH command. If the response included DAVE, the program stops; otherwise the response is moved to the first eight characters of SLOT. Then a question mark, appearing as a character literal, is moved to the subscript-defined last position of SLOT, intervening blanks removed with the COMPRESS command, and the resulting REPLY string displayed. Thus, if the S had answered "JOE," he would then see "YOUR NAME IS JOE?" The response to this question is left-justified with the COMPRESS command and matched with the constant YES. The matching operation compares only the first three characters of INPUT with the three-character constant and branches to @ASK if the match fails.

The Sternberg experiment program (Fig. 6) includes comments and so requires only some general explanations. The 1800 hardware works with word units, so I/O must always involve even numbers of characters, requiring that stimulus digits be paired with blanks. The GEN.RANDOM command generates random integers within a range specified by the maximum value. The PERFORM command acts as an embedded subroutine call, branching to the label after setting the numeric variable BACK to the address of the next command. JUMP commands accept both numeric variables and labels, so the JUMP BACK statement transfers control to the first statement after the PERFORM. The DELAY command delays processing on the S for the specified number of deciseconds.

	PROGRAM 3 STERNBERG EXPERIMENT	(COMMENTS)
	SV SET 20	
	SV DATA 8	THESE SUBSTRING STATEMENTS DEFINE
	SS DATA S 1 2	SOME VARIABLES GROUPED TOGETHER
	SS DATA POSNEG 3 2	FOR EASY OUTPUT AFTER EACH TRIAL.
	SS DATA EC 5 2	
	SS DATA RT 7 2	
	NV BACK, PROBE, #TRIALS, X, Y, N, HOLD	NV DEFINITIONS GROUPED HERE
	NC POS.RESP 1	FOR COMPACTNESS. NORMALLY,
	NC NEG.RESP 2	ONE DEFINITION PER CARD.
	NC OUT.FILE# 0	
	SC WARNING '* '	WARNING SYMBOL, NOTE BLANK.
	SC DIGITS '1 2 3 4 5 6 7 8 9 0 '	
	START	
	GEN.RANDOM 0 123	INITIALIZE RANDOM GENERATOR.
	#TRIALS=20	
	ERASE	
3TRIAL	GEN.RANDOM 6 5	CHOOSE RANDOM SET SIZE.
	PERFORM 3SCRAMBLE BACK	CALL SCRAMBLING SUBROUTINE.
	GEN.RANDOM 2 POSNEG	COIN FLIP FOR POSITIVE OR NEGATIVE
	MATCH POSNEG 1 3NEGATIVE	INSTANCE ON THIS TRIAL.
3POSITIVE	GEN.RANDOM 5 PROBE	PROBE DIGIT IN FIRST 5 DIGITS OF SET.
	JUMP 3DISPLAY	
3NEGATIVE	Y=10-S	PROBE IN REMAINING DIGITS.
	GEN.RANDOM Y PROBE	
3DISPLAY	PROBE=PROBE*2-1	COMPUTE CHARACTER 'NUMBER' OF PROBE.
	N=S	
	X=1	
3D.LOOP	DISPLAY SET(X,2) 1240 2400	DISPLAY SINGLE DIGIT FROM SET.
	DELAY 12	DELAY FOR 1.2 SECONDS
	X=X+2	
	LOOP N 3D.LOOP	LOOP TO DISPLAY ALL 5 DIGITS.
	ERASE	
	DISPLAY WARNING 1240 2400	DISPLAY WARNING SIGNAL.
	DELAY 5	
	DISPLAY SET(PROBE,2) 1240 2400	DISPLAY PROBE DIGIT.
	DI.READ.WT X	WAIT FOR PUSHBUTTON RESPONSE.
	LATENCY 1 RT	COMPUTE LATENCY IN MS.
	ERASE	
	MATCH POSNEG 1 3NEGTEST	GO TO RESPONSE TEST.
3POSTEST	MATCH X POS.RESP 3WRONG	GO TO 3WRONG IF RESPONSE NOT RIGHT.
	SKIP	SKIP NEXT COMMAND.
3NEGTEST	MATCH X NEG.RESP 3WRONG	RESPONSE WRONG IF NOT NEGATIVE.
3RIGHT	EC=1	DATA CODE FOR CORRECT RESPONSE.
	SKIP	
3WRONG	EC=0	DATA CODE FOR INCORRECT RESPONSE.
	DATA.OUT.OUT.FILE# DATA 3EOF	OUTPUT S, POSNEG, EC, & RT.
	DELAY 50	
	LOOP #TRIALS 3TRIAL	LOOP FOR NUMBER OF TRIALS.
3EOF	STOP	
3SCRAMBLE	MOVE DIGITS SET	THIS EMBEDDED SUBROUTINE RANDOMLY
	N=10	INTERCHANGES THE DIGITS IN SET.
3A	GEN.RANDOM 10 Y	
	HOLD=SET(Y)	THESE STATEMENTS MOVE TWO CHARACTERS
	SET(Y)=SET(N)	AT A TIME, SINCE THEY ARE ARITHMETIC
	SET(N)=HOLD	STATEMENTS.
	LOOP N 3A	
	JUMP BACK	RETURN TO COMMAND AFTER 'PERFORM'.
	END	

Fig. 6. Program for the Sternberg experiment.

Usually the user wishes to display specific rather than random stimuli to the Ss, so the strategy of on-line stimulus generation is often impractical. In this case, as shown in Fig. 1, off-line programming would be used to generate and load the stimulus materials on the disk. The experiment program would then obtain the stimulus information by means of the disk input commands.

CONCLUSION

Since GEPS is written in 1800 assembler language, it is not directly usable on other machines. However, GEPS does represent a successful approach to making a computerized laboratory more useful to experimental psychologists and thus may be of interest to other laboratories with similar user requirements.