SESSION X

DIGITAL EQUIPMENT CORPORATION USERS' GROUP

DORIS AARONSON, *New York University, President*

A simple LSI-11 system for real-time control, interevent time storage, and data analysis

WILLIAM L. PALYA

Jacksonville State University, Jacksonville, Alabama 36265

and

ALLAN F. DOYLE

Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

A simple laboratory computer system based on a Digital Equipment Corporation LSI-11, floppy disk, DRV11 parallel input-output board, and the RT-11 operating system is described. Interface to experimental devices is provided through a lab-built relay driver and relay closure sensing interface. An extensive high-level software package provides an easy-to-use control language (e.g., stimuli can be controlled with a simple "TURN ON" or "TURN OFF" instruction) and easy-to-use FORTRAN subroutines for data exploration (e.g., "IFIND" searches a data file for a particular event). The control software automatically generates, codes, and stores a complete log of every input and output event and its time of occurrence in each of five simultaneously running experiments. This provides the capability to reanalyze data in light of hypotheses not available when the experiment was designed. The FORTRAN subroutine library for data exploration provides a conditional and iterative search facility to sift out events or sets of events from the data file for analysis. Standard FORTRAN statements perform arithmetic operations on the resulting data.

The present system and its subsequent elaboration (Doyle & Palya, 1980) provide a researcher with the ability to easily generate, maintain, and analyze complete interevent time records of operant research. This system consists of a computer and mass storage device that interacts with experimental chambers through custom logic and relay interfaces. Under a sophisticated operating system, custom software conducts experiments in response to English-like statements. Custom-analysis software allows an investigator to obtain any desired analysis of the data after the experiment is completed. The system consists mostly of off-the-shelf hardware and user languages written in FORTRAN.

The authors are greatly indebted to Elizabeth Palya for assistance in all phases of producing this system and this manuscript. Requests for reprints, software, and schematics should be sent to William L. Palya, Department of Psychology, Jacksonville State University, Jacksonville, Alabama 36265.

Almost the entire configured system can be purchased from any number of system houses (Palya, 1978b).

COMMERCIAL HARDWARE

The system is built around a Digital Equipment Corporation (DEC) KD11-F or LSI-11 central processing unit (CPU), as is illustrated in Figure 1. Memory is expanded beyond the 4,096 (4K) words of memory contained on the CPU board to the maximum allowable 28K words. In addition, an optional floating-point and double-precision arithmetic chip has been added to the CPU board.

Permanent data and program storage is provided by a DEC RX01 dual floppy disk subsystem. It contains two disk drives that accept 8-in. floppy disks, each capable of storing 128K words, or 256K bytes.

The user interacts with the system through a Lear Siegler ADM 3 video terminal which displays 24 lines of

Figure 1. A block diagram of a simple LSI-11 system.

text with 80 characters/line. The terminal is interfaced to the computer by means of a DEC DLV11 serial line unit. The purpose of the DLV11 is to convert between the parallel data format used internally and the serial data format required by the terminal. Hard-copy output is produced by a Centronics 761 printer that plugs into an auxiliary connector in the back of the video terminal.

When a computer is first powered up, it must be loaded with a small program that loads into the computer a larger program from the mass storage device. This, in turn, loads into memory the operating system, a process known as "bootstrapping." The bootstrap program can be entered manually each time the system is powered up. Alternatively, a module (REV11-C) can be purchased that carries out the bootstrap process automatically. We decided on this option.

In order to control experiments, three DRV11 parallel input/output (I/O) boards were added to the system. Each DRV11 has 16 outputs and 16 inputs at TTL logic levels of ground and +5 V. Custom circuitry interfaces the DRV11s to the chambers.

The CPU communicates with all other modules over a bus. The LSI-11 bus (Qbus) is an approximate subset of the PDP-11 UNIBUS. Other special-purpose modules are commercially available for the Qbus (Palya, 1978a, 1978b).

CUSTOM HARDWARE

We use three-key pigeon chambers with 12-stimulus displays behind each key. All signals in or out of these

chambers go through relays mounted on the back of each chamber. These relay interfaces also contain any necessary power supplies. Three mercury-wetted relays operated by keypecks provide contact closures to the computer; 14 relays operated by the computer activate all events in the chambers. One output relay controls all power to the chamber by switching the incoming 110-V line. One relay operates the food magazine, magazine light, and houselight. The additional 12 relays can be connected to any of the 36 key lights through a patch panel. Each chamber therefore provides three inputs to the computer and receives up to 14 outputs.

A relay driver and relay closure sensing interface, previously described by Palya and Doyle (1978), provides a link between the +5-V low-current signals available on the DEC DRV11 parallel boards and the voltage and current requirements of the relays on the experimental chambers, as is illustrated in Figure 2. The interface is functionally very simple. The outputs of the DRV11 activate high-current drivers, which in turn operate the relays. Contact closures provided by keypecks set flip-flops on the interface. The status of these flip-flops is read by the computer.

COMMERCIAL SOFTWARE

The computer and associated interfaces are virtually useless without an operating system and some high-level language. A software operating system coordinates the resources of the computer to provide the user with an environment within which application programs can be easily developed. RT-11 is an excellent operating

Figure 2. The relay driver and relay closure sensing interface.

system available from DEC. It is a supervisory system (see Figure 3) that contains general overhead facilities, file handling programs, and timekeeping services. It also includes user utilities, such as a text editor, a linker, and a librarian. RT-11 also provides a FORTRAN compiler that allows the user to write programs in a superset of standard ANSI-FORTRAN. FORTRAN allows a programmer to concentrate more on application than on the drudgery of programming in the language of the computer. An additional advantage of FORTRAN is that it is machine-independent. Programs written in FORTRAN on the LSI-11 for the most part will run on almost any other computer with a FORTRAN compiler.

Figure 4 illustrates the process of producing a runnable FORTRAN program under RT-11. Programs are

entered and stored on the mass storage device by using the text editor (EDIT). These "source" programs are written in FORTRAN. Once a source program has been entered, it can then be compiled. A compiler translates the English-like statements of the source program and compiles them into object code, a machine language code interspersed with "abbreviations" of frequently used blocks of machine language code. These abbreviations refer to (or call) subroutines. The actual subroutines are maintained in a library file stored on the disk. Therefore, before the program can be run, it must be linked. This process inserts the blocks of required machine language code in place of each of the subroutine calls, thereby producing an "executable image" or a "run module." The executable image contains all

Figure 3. The RT-11 operating system.

Figure 4. A block diagram of the process of producing a runnable FORTRAN program under RT-11.

the information needed for the processor to run the program under RT-11. A RUN command to RT-11 loads this form of the program from the mass storage device into main memory and begins its execution.

CUSTOM SOFTWARE

We used FORTRAN to write the software necessary to control experiments while collecting complete interevent times. One method of using this software would have required that the researcher specify the interconnected control flow of the five simultaneously running experiments. That would have been suboptimal in that the entire program would have been useless if one of the experiments was changed.

BRDRUN

We chose to implement a multiprocessing system in which up to five control programs can be executed independently and in parallel. While running, each program is effectively the only program in the system and each is written as if it were continuously executing. Actually, each independent program is activated consecutively by the multiprocessing supervisory control program for the time it takes to execute one logical packet of instructions, such as the sequence that turns on a stimulus. In this way each program is allocated its share of the available time minus a small amount for the overhead required to coordinate the "context switch." Each time slice is approximately 5 msec.

BRDRUN, the control program, provides a function beyond simple multiprocessing. The instruction set executed by our five independent programs is optimized for experiment control rather than being PDP-11 instructions. BRDRUN is the supervisor of five separate but coordinated special-purpose computers. These five computers are called "virtual" machines because they are created by software rather than implemented in hardware. These virtual machines are "emulated" in FORTRAN, and the programs for these five machines are FORTRAN data files. Each instruction for a virtual machine is emulated at run time by a series of FORTRAN

instructions contained in the BRDRUN control program.

Figure 5 illustrates the program flow of BRDRUN. BRDRUN emulates the first machine for one virtual

Figure 5. The program flow of BRDRUN.

Table 1
BRDRUN User-System Dialogue

```
.RUN BRDRUN
BRDRUN OPERATIONAL      7-NOV-79      15:00:00

ENTER INPUT FILE NAME (.CLO)
•AUT01.CLO

ENTER DATA FILE NAME (.TRL)
•AU0001.TRL

SESSION #? 1

BOX 1 - BIRD #? 88
        TOTAL REINFS? 50
        POWER BIT? 1

BOX 2 - BIRD #? 356
        TOTAL REINFS? 30
        POWER BIT? 400

BOX 3 - BIRD #? 9
        TOTAL REINFS? 25
        POWER BIT? 1

BOX 4 - BIRD #? 421
        TOTAL REINFS? 60
        POWER BIT? 20

BOX 5 - BIRD #? 17
        TOTAL REINFS? 45
        POWER BIT? 1

PAUSE - - WHEN BOXES READY, TYPE RETURN

- - - - - BOXES RUNNING - - - - -

- - - - - BOXES DONE - - - - -
```

Note—User input is in large type.

machine instruction, then emulates the second machine, and so on. Between the deactivation of one virtual machine and the reactivation of the next, or context switch, the input interface is examined by BRDRUN. If new flip-flops are set, the action is logged into the data file, and the flip-flops are cleared. BRDRUN then performs an instruction sequence that screens out contact bounce to avoid false inputs.

Data collection is handled automatically by BRDRUN. When a session begins, BRDRUN records the time the experiment starts, the subject numbers, and the data. BRDRUN also records the time of occurrence of every subsequent event. Thus, at the conclusion of an experiment, every stimulus change and every recordable behavior input has been logged in a compressed-time format. The entire experiment can, in a sense, be recreated, since all the information available to the computer at run time is subsequently available to the researcher.

Table 1 provides the user-system dialogue involved in running each set of experiments with BRDRUN. BRDRUN first determines the programs to be run and the name of the output data file. It then requests the session number. Then for each box, it requests: the bird number, the number of reinforcers to be administered, and the bit number that powers up the box. BRDRUN then pauses, waiting for birds to be loaded. When carriage return is pressed, the experiments begin, and messages typed to that effect. When the last experiment is terminated, it indicates that the boxes are done.

CLOC

A researcher can write programs for the five virtual machines running under the BRDRUN supervisory control program by writing in the language of the virtual machines. This is analogous to writing LSI-11 programs in LSI-11 machine language. But this approach is laborious and error prone.

Just as DEC supplies high-level language compilers that translate languages such as FORTRAN into machine language, we wrote a program (computer language for operant conditioning, or CLOC) to translate English-like statements into virtual machine language.

Using the text editor, EDIT, the researcher describes the desired experiment in the CLOC language. CLOC is then used to compile the program into code that runs on the virtual machines. The entire instruction set for CLOC is presented in Appendix A. A sample source program for running a variable-ratio 100 (VR100) schedule of reinforcement is shown in Table 2. The heading, BOX 1, serves to identify for BRDRUN the experimental chamber to which the program refers. All the data subsequently generated by this program are tagged with a 1 to distinguish them from the data generated by the other four programs. TURN ON ST05 causes the physical stimulus event associated with Stimulus 5 for that box to be presented (in this case a white key light). RANDOM INTO VR01, BOTTOM IN0001, TOP IN0200 sets Variable 1 equal to a random number between 1 and 200. COUNT VR01 PECKS ON KEY IN0002 causes the program to wait until the number of pecks on Input 2 has reached the number in Variable 1. REINF ST05 OFF, ST02 FOR RL0004.0 RESET ST05 THEN 002 is the reinforcement sequence and is several instructions executed as a group by BRDRUN. This instruction turns off Stimulus 5, then turns on Stimulus 2 (in our case, the magazine light and magazine) for 4.0 sec, then reinstates Stimulus 5. THEN the program loops back to the second statement in the program. The STOP statement serves to delimit the instructions associated with Box 1 from those for other boxes. BRDRUN automatically terminates the session after the number of reinforcers specified in the start-up dialogue.

Configuration

In order to allow the researcher to specify stimulus number and input number instead of actual hardware addresses and particular bits on the interface, there has

Table 2
Box 1 Source Program for a VR100 Schedule

```
BOX 1
TURN ON ST05
RANDOM INTO VR01, BOTTOM IN0001, TOP IN0200
COUNT VR01 PECKS ON KEY IN0002
REINF ST05 OFF, ST02 FOR RL0004.0 RESET ST05 THEN 002
STOP
```

to be some way of specifying to CLOC the mapping between the two. This is the configuration file. When connecting the computer to the experimental chambers, the experimenter chooses the events in each chamber to be connected to individual relays and the relays to be connected to particular drivers and flip-flops on the interface. A special file that specifies the relationship between the hardware and the stimulus (ST) and key (KEY) numbers used in programming is created when using CLOC. CLOC appends this configuration file to the virtual machine code when it compiles a program. To avoid repeating this somewhat lengthy process, a previously created configuration file can be used if the hardware has not been changed. An example of the user-system dialogue to develop a configuration file is provided in Table 3.

Each time CLOC is run, it asks the user if a configuration file is to be made or an old one used. If a configuration file is to be made, CLOC asks for the name it is to be given. It then asks through which memory locations the input and output will pass. These are the addresses of the parallel boards. CLOC then requests the bit number for each of the 16 possible outputs for each box. Finally, it requests the bit number for each of the three possible inputs from each box. A carriage return without a preceding number indicates that the input or output is not used.

Table 4 provides the user-system dialogue involved in running the CLOC compiler. CLOC first determines whether or not the user wishes to create a new configuration file, and if not, which previously created

Table 4
CLOC Compiler System-User Dialogue

```
.RUN CLOC

DO YOU WISH TO CHANGE CONFIGURATION FILE?
NO

ENTER CONFIGURATION FILE NAME
*AUTO1.CFG

ENTER INSTRUCTION FILE NAME
*AUTO1.INS

LIST SOURCE CODE? YES

BOX 1
STOP
BOX 2
  TURN ON ST03
  RANDOM INTO VR01 BOTTOM RL0005.0 TOP RL0120.0
  WAIT VR01 SECONDS
  RANDOM INTO VR02 BOTTOM RL0001.0 TOP RL1000.0
  COMP VR02 TO RL0500.0 IN0006, IN0006, IN0021
  TURN OFF ST03
  TURN ON ST07
  WAIT RL0004.0 SECONDS
  TURN OFF ST07
  TURN ON ST10
  WAIT RL0004.0 SECONDS
  TURN OFF ST10
  TURN ON ST08
  WAIT RL0004.0 SECONDS
  REINF ST08 OFF ST02 FOR RL0004.0 RESET ST03 THEN 016
  .
STOP
.
BOX 5
STOP

ENTER OUTPUT FILE NAME
*AUTO1.CLO

LIST OBJECT CODE? NO
.
```

Note—User input is in large type.

file is to be used. It then obtains the instruction file to be compiled. Source code can be listed at this point. CLOC then requests the name to give to the output file, compiles the program, and can list object code if desired.

Figure 6 illustrates the overall flow of using a program in the CLOC language. AUTO1.INS is entered using the EDIT facility. This program specifies the experimental conditions and is stored on the disk. The user then creates a configuration file (AUTO1.CFG) by running CLOC and responding to the prompts appropriately. Next, the CLOC compiler is run specifying the configuration file and instruction file for input and the name of the compiled program (AUTO1.CLO) that is to be created. BRDRUN is run each time subjects are to be run. The user specifies the compiled instructions for that set of birds and the name of the data file (AU0001.TRL) to be output to the disk.

SPLIT

BRDRUN does not maintain separate data files on the mass storage device for each of the five experiments it runs at one time. The data from all five chambers of any one run are mixed into the same file. Therefore, before analysis can take place, the data must be split up into separate files for each experiment. This is

Table 3
CLOC Configuration System-User Dialogue

```
.RUN CLOC

DO YOU WISH TO CHANGE CONFIGURATION FILE?
YES

ENTER NAME OF CONFIGURATION FILE * AUTO1.CFG
DRCSR = 167760
INPUT = 167764
BOX 1 = 167772
BOX 2 = 167772
BOX 3 = 167772
BOX 4 = 167772
BOX 5 = 167782

1,1 = 1
1,2 = 2
1,3 = 4
1,4 = 10
1,5 = 20
.
1,15 =
1,16 =
2,1 = 1000
2,2 = 2000
.
5,16 =

KEY LEFT, BOX 1 = 1
KEY CENTER, BOX 1 = 2
KEY RIGHT, BOX 1 = 4
KEY LEFT, BOX 2 = 10
.
KEY RIGHT, BOX 5 = 200
.
```

Note—User input is in large type.

Figure 6. A block diagram of the process of producing a runnable control program in the CLOC language under RT-11.

accomplished with a small utility program called SPLIT. The resulting data file is a two-dimensional array of numbers. Each consecutive row represents a consecutive event. The first two columns of each row are numerical codes that identify the type of event, and the other columns identify the time it occurred coded into standard DEC format with 60-Hz resolution. Table 5 illustrates the user-system interaction under SPLIT. SPLIT requests the name of the input file. It then determines whether or not a listing is to be printed, and it consecutively asks if each box is to be split out of the data file. If a box is to be split, SPLIT requests the output file name. It then lists header information and an event-by-event list with its time of occurrence of everything that happened in that box during that session. Listings need not be obtained; in that case only a disk file is created.

EXPLOR

A library named EXPLOR provides FORTRAN subroutines that allow the researcher to easily explore the interevent time data resulting from an experiment. This is accomplished by arranging all events on a time line and providing a "pointer" that can move along that time line. Using EXPLOR subroutines, the pointer can be moved forward and backward in time, or moved to some time or event while cumulating the occurrences of another event (such as keypecks). The complete sub-

Table 5
SPLIT User-System Dialogue

```

.RUN SPLIT
ENTER DATA FILE NAME (.TRL)*AU0001.TRL
DO YOU WANT LISTINGS? YES
SPLIT OUT BOX 1? NO
SPLIT OUT BOX 2? YES
ENTER FILE NAME FOR IT*AU0001.SPL
VERSION 2      DATA 11/ 7/79
SESSION NUMBER 4      START TIME 15:00:00:00
BIRD NUMBER    23      ADDRESSES 167760 167764 167752
CODES 1-10
20 40 100 200 400 1000 2000 4000 10000 20000
CODES 11-20
0 0 0 0 0 0 0 0 0 0
KEY LEFT 0 0 0 0 0 0 0 0 0 0
:
4 -3 0: 0: 6:58
4 7 0: 0: 0: 1
41 64 0: 0: 1: 8
4 -7 0: 0: 2:54
4 10 0: 0: 0: 1
41 64 0: 0: 0:25
41 64 0: 0: 0:21
4 -10 0: 0: 3:16
4 8 0: 0: 0: 1
41 64 0: 0: 0:32
41 64 0: 0: 0:19
41 64 0: 0: 0:20
4 -8 0: 0: 2:50
4 2 0: 0: 0: 0
4 -2 0: 0: 4: 0
:
SPLIT OUT BOX 3? NO
SPLIT OUT BOX 4? NO
SPLIT OUT BOX 5? NO

```

Note—User input is in large type.

Table 6
Sample EXPLOR Program

	PROGRAM ITI	
C		
C		
C	SAMPLE ANALYSIS PROGRAM	
C		
	CALL INIT	!INITIALIZE - OPEN FILE
C		
C	MODULE TO COUNT TOTAL PECKS	
C	ON KEY CENTER DURING ITI	
C	FOR 5 TRIALS	
C		
	ITITOT = 0	!ZERO VARIABLE
	DO 100 I = 1,5	!DO LOOP 5 TIMES
	CALL IFIND (1, 3)	!MOVE FORWARD UNTIL CODE
C		3 (BEGINNING OF ITI)
	CALL IPCODE (1, 2, -3, IPKS)	!MOVE FORWARD, COUNT
C		PECKS ON KEY CENTER
C		UNTIL CODE 3 GOES OFF.
C		CUMULATE PECKS IN
C		VARIABLE IPKS
	ITITOT = ITITOT + IPKS	!TOTAL PECKS IN VARIABLE
		ITITOT
C		
100	CONTINUE	
C		
C		
C	PRINT OUT TOTAL PECKS	
C		
	WRITE (7, 1000)ITITOT	
1000	FORMAT ('0', 'TOTAL ITI PECKS ON CENTER KEY', I6)	
C		
	CALL DONE	!CLOSE FILE
	END	

routine library is presented in Appendix B. A short program is presented in Table 6. The first instruction (CALL INIT) is a subroutine call that requests from the console terminal the data file to be analyzed and establishes a link to that data file. ITITOT = 0 resets the value of the ITITOT counter. The DO 100 and CONTINUE bracket a loop. Everything in this loop is done five times. CALL IFIND moves the pointer forward until it encounters Code 3, which is the beginning of an inter-trial interval (ITI). The CALL IPCODE instruction moves the pointer forward, counting keypecks until the ITI ends. It cumulates the number of pecks encountered in the ITI in the variable named IPKS. The ITITOT = ITITOT + IPKS instruction is a running total of the ITI pecks. The WRITE and FORMAT instructions output the contents of the ITITOT counter after typing

"TOTAL ITI PECKS ON CENTER KEY." The CALL DONE and END instructions terminate the program.

REFERENCES

- DOYLE, A. F., & PALYA, W. L. A hierarchical LSI-11 system for real-time control, interevent time storage, and data analysis. *Behavior Research Methods & Instrumentation*, 1980, 12, 221-231.
- PALYA, W. L. Configuring an LSI-11 system. *Mini-Micro Systems*, October 1978, pp. 63-74. (a)
- PALYA, W. L. A source handbook for Digital Equipment Corporation LSI-11-compatible products. *Behavior Research Methods & Instrumentation*, 1978, 10, 488-509. (b)
- PALYA, W. L., & DOYLE, A. F. A relay driver and contact closure sensing interface for an LSI-11 DRV11 parallel I/O board. *Behavior Research Methods & Instrumentation*, 1978, 10, 485-487.

Appendix A CLOC Instruction Set

CONVENTIONS

CLOC can evaluate the following types of numbers. A number must have the number of digits indicated.

Octal	OC_____	6 digits	000000-177777
Integer	IN_____	4 digits	-999-3,276
Real	RL_____	4 digits-point-digit	-999.9-3,276.7

Inputs	Key IN0001-Key IN0003	Any Key = IN0000
Stimuli	ST01-ST16	
Markers	IN0021-IN0040	
Clocks	Clock IN0001-Clock IN0010	
Variables	VR01-VR20	

! denotes that the line following is a comment and should be ignored by the compiler.

OUTPUT INSTRUCTIONS**Stimuli**

TURN ON ST__
TURN OFF ST__

ST01-ST16. These instructions turn on or turn off whatever event is connected to that relay. The configuration process specifies which relay is associated with each of the 16 potential ST numbers for each box.

Reinforcement

REINF ST __ OFF, ST __ FOR RL _____ RESET ST __ THEN ____

This instruction is executed as a single instruction by BRDRUN. It turns off a stimulus, turns on a stimulus for some time, turns on a stimulus, then directs program flow to some instruction. Use of the REINF instruction causes a counter called REINF to be incremented each time this instruction is executed. Before the session begins, BRDRUN asks for the number of reinforcers to be delivered to a chamber. When REINF reaches this number, that chamber is automatically powered down and the program for that chamber is halted.

Marker

MARKER IN_____

IN0021-IN0040. This instruction has no effect other than to put a marker in the data file. It enables EXPLOR to move conveniently move to a point in time that has no other event change without backspacing.

INPUT INSTRUCTIONS

ANY KEY IN0000 IN_____, IN_____, IN_____
if KL if KC if KR

This instruction waits for the first peck on ANY KEY and transfers program control to the indicated line number, depending on which keypeck was received. Program flow is directed to the line number indicated first if the peck was key left, to the second if key center, and to the third if key right.

COUNT IN_____PECKS ON KEY IN_____THEN_____

This instruction holds program flow until the required number of pecks occur on the specified key. It then transfers program control to the indicated instruction.

TIMING INSTRUCTIONS

WAIT RL_____SECONDS
WAIT VR__SECONDS

These instructions hold program flow until the required time elapses.

START CLOCK IN_____

IN0001-IN0010. This instruction starts 1 of 10 clocks. The clock can subsequently be examined. This instruction enables the program to go on, whereas a WAIT statement stops the program until the time elapses.

VALUE ASSIGNMENT INSTRUCTIONS

VR__ = IN_____
VR__ = RL_____

This instruction sets a variable equal to an integer or a real number.

RANDOM INTO VR__BOTTOM IN_____, TOP IN_____
RANDOM INTO VR__BOTTOM RL_____TOP RL_____

These instructions set a variable to a random number between the bottom and top value specified. The number (either integer or real) is drawn from a rectangular distribution.

GAUSS INTO VR__STDDEV IN_____, MEAN IN_____

GAUSS INTO VR__STDDEV RL_____, MEAN RL_____

These instructions set a variable to a random number drawn from a Gaussian distribution with the specified mean and standard deviation. The number can be integer or real.

ARITHMETIC INSTRUCTIONS

Addition	VR__ + IN_____	VR__ + RL_____
Subtraction	VR__ - IN_____	VR__ - RL_____
Multiplication	VR__ * IN_____	VR__ * RL_____
Division	VR__ / IN_____	VR__ / RL_____

These instructions simply add, subtract, multiply, or divide a variable by the specified number. The number can be integer or real. The previous value of the variable must be of the same type.

INCREMENT

INCREMENT VR__

This instruction increments a specified variable. Incrementing is faster than adding, but it is to be used only with integer numbers.

CONDITIONAL BRANCH INSTRUCTIONS

COMP VR__ TO OC_____	IN_____, IN_____, IN_____
COMP VR__ TO IN_____	IN_____, IN_____, IN_____
COMP VR__ TO RL_____	IN_____, IN_____, IN_____
COMP VR__ TO VR__	IN_____, IN_____, IN_____

< = >

These instructions compare the value of a variable with a number and transfer program execution, depending on the result of that comparison. If the specified variable is less than the number, then the program is transferred to the line number specified first. If the variable is the same, then the program is transferred to the second. If the variable is greater, then the program is transferred to the third.

COMP CLOCK IN_____TO RL_____

IN_____, IN_____, IN_____

< = >

This instruction compares the specified clock with some real number time. It then transfers program control to the indicated line number, depending on the outcome of the comparison. If the clock is less than the real number, program execution is transferred to the first line number; if the same, the second; and if greater, the third.

COMP REINF TO IN_____GO TO IN_____, IN_____

< >

This instruction compares the number of reinforcers that have already occurred to an integer and transfers program control to the line number indicated, to the first if less, to the second if greater or equal.

EXECUTION CONTROL INSTRUCTIONS

THEN_____

This instruction unconditionally directs program flow to the indicated line number and, if used, must be appended to the end of another instruction.

BOX__

This instruction tells CLOC which of the five programs follows. The programs must be arranged consecutively: Box 1-Box 5.

STOP

This instruction removes the chamber from further consideration by BRDRUN. It must be the last statement in each program.

END

This instruction must be the last statement at the end of the five programs.

Appendix B

CONVENTIONS

Stimuli on =	1-16
Stimuli off =	-1--16
Markers on =	21-40
Markers off =	-21--40
Peck =	41
Key 0	any key
Key 1	key left
Key 2	key center
Key 3	key right

Time is structured as a four-element array. Any operations on time are on the four-element array. Carries or borrows are automatically carried out appropriately.

ITIME =	TIME (1) hours
=	TIME (2) minutes
=	TIME (3) seconds
=	TIME (4) ticks

MODULES

When the following modules are invoked as functions rather than subroutines, they return +1 normally and return -1 if an end of file or beginning of file has been encountered.

When movement of the pointer is required, the J argument specifies whether the movement shall be in the forward (J = +1) or in the backward direction (J = -1).

EXPLOR uses Logical Unit 2 for disk I/O. User programs must not use this number. User programs can use Logical Unit 3, 4, and so on.

PROGRAM OVERHEAD

Initialize

CALL INIT

This must be the first statement in the program (other than comments and the PROGRAM statement). It opens files and reads header information.

Done

CALL DONE

This must be the next-to-last statement in the program. It closes files.

End

END

This must be the last statement in the program.

READ HEADER INFORMATION

CALL RDHDR (IVR, IMO, IDA, IYR, ISESS, ITIME, IBIRD)

This module reads the header information on the data file. Subsequently, the analysis program has access to that information through the indicated variable names.

IVR =	Analyze version number
IMO =	month (2 digits)
IDA =	day (2 digits)
IYR =	year (2 digits)
ISESS =	session number (5 digits)
ITIME =	time (4-element array)
IBIRD =	bird number (5 digits)

VERIFY

CALL IVERIF (IBOX, ICODE, ITIME)
or I = IVERIF (IBOX, ICODE, ITIME)

This module reads the data record immediately following the pointer. Subsequently, the program has access to each of the elements of that data record through the indicated variable names.

IBOX = the box number if the event was an output or "41" if the event was an input.

ICODE = the integer code if the event was an output or the key number if the event was an input.

ITIME = time in a four-element array.

POINTER MOVEMENT

Move Pointer to Beginning of Data File

CALL BEGIN

This module moves the pointer to the beginning of the data file.

Move Pointer to a Time

CALL MOVE (J, ITIME)
or I = MOVE (J, ITIME)

This module moves the pointer to the specified time. It does not count anything.

ITIME = time at which to stop.

Move Pointer to a Code

CALL IFIND (J, ICODE)
or I = IFIND (J, ICODE)

This module moves the pointer to the specified code. It does not count anything.

ICODE = code at which to stop (+1--41 or -1--40)

Move Pointer to a Peck

CALL IFNDPK (J, KEY)
or I = IFNDPK (J, KEY)

This module moves the pointer to a specified peck. It does not count anything.

KEY = the peck at which to stop (0 for any, 1, 2, or 3)

EVENT COUNTING

Cumulate Pecks to a Code

CALL IPCODE (J, KEY, ICODE, IPECK)
or I = IPCODE (J, KEY, ICODE, IPECK)

This module cumulates the pecks on a specified key up to a specified code.

KEY = the pecks that are to be cumulated (0 for any, 1, 2, or 3)

ICODE = the code at which to stop

IPECK = is returned, it is the total number of pecks counted up to the specified code

Cumulate Time to a Code

CALL ITLTIM (J, ICODE, ITIME, KEY)
or I = ITLTIM (J, ICODE, ITIME, KEY)

This module cumulates the elapsed time up to a specified code. The pointer can remain where it is or it can be moved to the specified code.

ICODE = code at which to stop

KEY = if the destination event is a peck, then key is 0 (any), 1, 2, or 3

ITIME = is returned and is the total elapsed time up to the specified code

J = +1 search forward moving pointer

J = -1 search backward moving pointer

J = +2 search forward leaving pointer where it was
 J = -2 search backward leaving pointer where it was

ARITHMETIC OPERATIONS

For all arithmetic operations:

I = +1	positive answer
I = 0	answer was zero
I = -1	negative answer
I = -2	overflow

Add Time

CALL ITMADD (ITIME, JTIME, KTIME)

or I = ITMADD (ITIME, JTIME, KTIME)

This module adds two times: $ITIME + JTIME = KTIME$.

Subtract Time

CALL ITMSUB (ITIME, JTIME, KTIME)

or I = ITMSUB (ITIME, JTIME, KTIME)

This module subtracts two times: $ITIME - JTIME = KTIME$.

Multiply Time

CALL ITMMUL (J, JTIME, KTIME)

or I = ITMMUL (J, JTIME, KTIME)

This multiplies time by an integer: $J * JTIME = KTIME$.