Behavior Research Methods, Instruments, & Computers
1993, 25 (2), 127-136

A powerful, inexpensive experiment controller or IBM PC interface and experiment control language

WILLIAM L. PALYA and DONALD E. WALTER
Jacksonville State University, Jacksonville, Alabama

A very powerful, but inexpensive, advanced technology (16 MHz 80C188EB) experiment controller is described. It can be programmed in either a BASIC-like (ECBASIC) or an ALGOL-like (ECL) procedure specification language. It provides 1-msec resolution, optional transparent total data logging, and is designed to function as a remote peripheral processor in a network with virtually any computer acting as the network supervisor (e.g., IBM compatible or Macintosh). Each serial port on the host computer can support up to 10 simultaneous experiments. The various optional I/O modules provide for opto-isolated normally open or normally closed switch operation input, high current output, as well as D/A and A/D functions. Our I/O modules can also be plugged directly into an IBM PC parallel port by using a small adaptor board. In this way, the advantages of using ECBASIC or ECL to control experiments are available to researchers who wish to simply interface their host computer directly to the apparatus in order to minimize expense.

This paper presents the details of our new experiment control computer. The functionality of our previous controller (Walter & Palya, 1984) has been greatly enhanced. Two control languages are supported, the hardware can be used either in a network mode with IBM-compatible or Macintosh computers, or as a simple interface between an IBM PC and an experimental apparatus. Several I/O options are available, and 1-msec resolution is provided. The ruggedized $\frac{3}{32}$-in. thick boards are commercially manufactured, solder masked and silk screened. Significant new features of the procedure specification languages include interrupt routines that provide a convenient way to control concurrent processes, and six letter variables.

There are a vast number of ways that computers facilitate the conduct of inquiry in psychology. In many cases, prespecified outputs are presented to a subject and the resulting behavior is used to alter subsequent outputs or is simply recorded. Computers and software have been optimized for two types of research within this paradigm. In one, typically labeled *experiment generation*, humans respond on a keyboard in response to stimuli presented on a CRT. As a result, special-purpose hardware is unnecessary. MEL and SuperLab are typical of this class of system. The other type, typically labeled *experiment control*, processes a variety of digital events, such as turning on and off lamps or buzzers, and receiving a variety of digital switch operations, such as leverpresses. Occasionally, analog inputs and/or outputs are used. The Med PC system and the controller described in the present paper are of this latter type.

Most of the rationale underlying the appropriateness of our implementation philosophy and the details of how to use our system have been extensively covered for our previous system and will not be repeated here. The interested reader is referred to Cooper, Garcia, and Gibbon (1988), D'Andrea and Knepton (1988), Palya (1988), Pevey (1988), Walter (1988), Walter and Palya (1984), and Weisman and Palya (1988). The present paper will present only an overview of the research environment and the features available with the new hardware and software. A very extensive user's manual (Palya & Walter, 1993) provides detailed descriptions of how to set up and use the current hardware and software. The language reference sections for ECBASIC and ECL provide format information and examples of each instruction. The networking sections for ECBASIC and ECL indicate how to set up the network for each and how to use the PC and Macintosh support software. The final section provides detailed information on the hardware itself.

Our Implementation Philosophy

There were three major decisions in the evolution of our implementation philosophy. The initial and most fundamental (illustrated in Figure 1) was the choice between a time-sharing solution and a distributed-network solution. A time-share solution uses a single CPU and relatively complex software and hardware to carry out multiple experiments simultaneously. Worst-case latency to service a response increases with each additional experiment; when it exceeds acceptable limits, an additional computer is added.

This research was supported by NSF Grant DIR-8915226 to W. L. Palya. The authors thank Josey Chu for the graphics used in the presentation and in this proceedings paper, and we gratefully acknowledge Elizabeth Palya for contributions in all phases of this project. Correspondence and requests for reprints should be sent to W. L. Palya, Department of Psychology, Jacksonville State University, Jacksonville, AL 36265 (e-mail: palya@sebac.jsu.edu).

A distributed network, on the other hand, uses a small, special-purpose controller (hardware interface plus microprocessor) at each experimental station, and interconnects and supervises them over a single wire. We chose a distributed-network solution because it maximized the reliability, simplified the hookup, and provided dramatically faster worst-case response. Latency to service a response does not increase with each additional experiment in that each apparatus has its own CPU.

The second decision involved the nature of the procedure specification language. The choice (illustrated in Figure 2) was between a state-notation solution and an algorithmic-language solution. State notation specifies the nature of each experimental state and the rules for transition between them; an algorithmic language specifies various experimental operations and path decisions. We chose to implement an algorithmic procedure specification system because it was less cryptic, more researchers would have prior experience with an algorithmic language, and those programming skills could be used for other applications, such as data analysis.

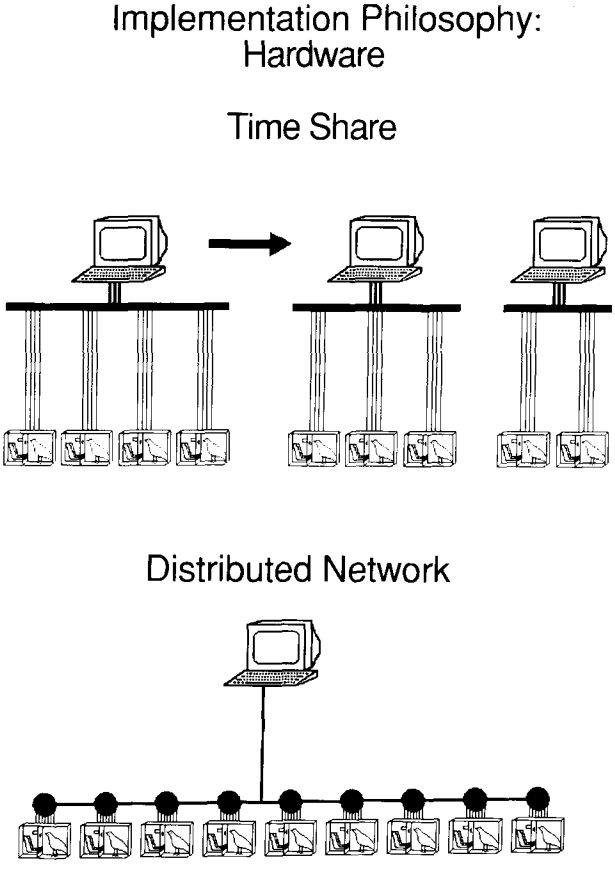

Figure 1. A schematic illustration of a time-shared and distributed-network research environment. Time-shared solutions have a relatively complex, centralized interface and long interconnects to the experimental apparatuses. A distributed network, on the other hand, has a simple, single-wire interconnect with each apparatus. Time-share solutions are obligated to add additional compute power to maintain worst-case response times as tasks increase or get more demanding. Distributed networks can simultaneously run a great many experiments that are computationally, extremely complex, that require extremely short delays to service an input, and that require complete data recording.

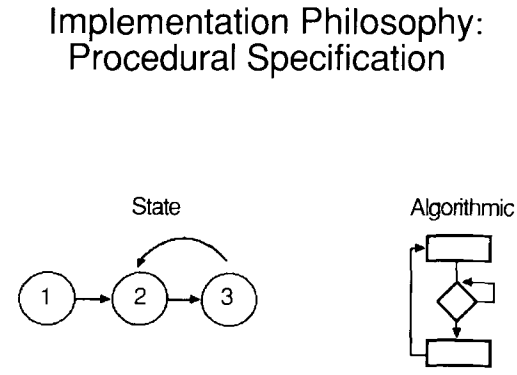

Figure 2. An illustration of the two most popular methods to specify an experimental procedure: state notation versus algorithmic notation.

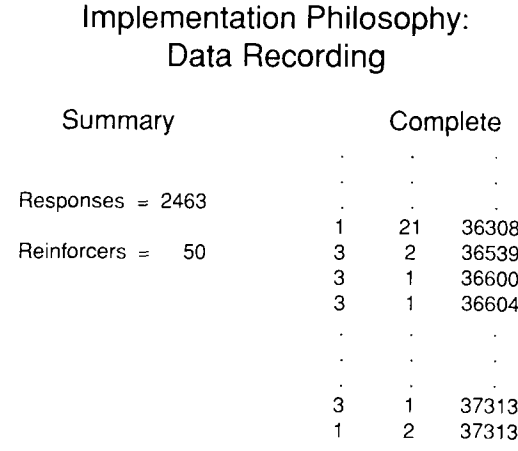

Figure 3. An illustration of two approaches to data collection. A set of measures (e.g., total responses, number of reinforcers, total running time, etc.) versus total data collection (e.g., 363,083 msec into the session, Light 21 came on and was followed at 365,397 msec into the session by a peck to Key 2; pecks on Key 1 occurred at 366,002 msec, 366,045 msec, . . . , and 373,130 msec, followed by reinforcement at 373,130 msec).

The final major implementation decision, illustrated in Figure 3, involved the nature of the data saved. The choice was between recording only a few preselected data summaries (e.g., total number of responses) versus maintaining a complete record of each event and when it occurred (e.g., at 363,083 msec into the session, Light 21 went on, and it was followed by a peck on Key 2 at 365,397 msec, etc.). We chose to make the recording of both summary recordings and complete event transcriptions simple. The facility to do total data recording enables more detailed and more interactive analysis, as well as post-analyses in light of new information. We provided for summary recording because it was easy to implement, it makes for extremely simple data analysis, and some researchers may actually prefer it.

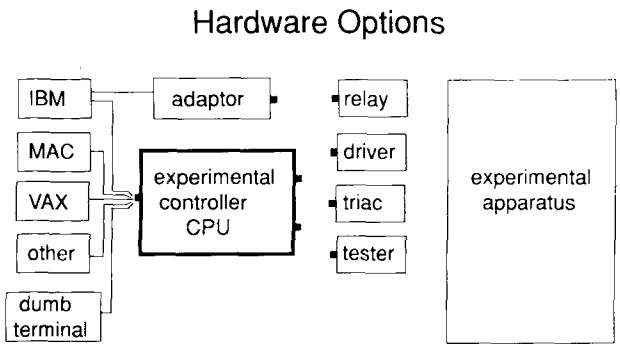

Figure 4. A schematic illustration of the configuration options available with the 80C188EB Experiment Controller. There are four daughter boards; they provide 2-A relays, 300-mA drivers, 5-A triacs, and an I/O tester. These boards can be connected either to the experiment controller or directly to an IBM PC through an adaptor board plugged into the parallel port. The experiment controller can be connected to a wide variety of computers, as well as to a dumb terminal, if a stand-alone system is desired.

Hardware Options

As illustrated in Figure 4, there are three ways that our hardware can be used to conduct research.

1. As a distributed network: In this case, each experimental apparatus is a self-contained, remote computer supervised and integrated by some host computer. The experiment controllers can be connected to virtually any computer ranging from a TRS80, Macintosh, or PC to a Vax. Day-to-day activities, such as writing procedure specifications, data archiving, and data analysis are carried out on the host computer using its resources. An experiment controller CPU and the desired I/O interface are typically mounted on the experimental apparatus. The controller is then connected to the host computer through a single wire.

2. As an interface for an IBM-compatible computer: In this case, an adaptor and the desired I/O interface are mounted on the experimental apparatus. The adaptor is then connected to an IBM-compatible computer's printer port through a flat cable.

3. As a stand-alone computer: Our experiment controller can be programmed with a simple "dumb" terminal attached to the serial port. In this case, an ECBASIC program is entered line by line and run exactly as any BASIC program. However, programs cannot be stored, and data must be printed to a local printer or manually transcribed from the terminal screen. The tradeoff in this case is less convenience for less expense.

The controller (18 × 16 cm) is based on a 16-MHz 80C188EB. It executes 8,088 instructions, has 128K bytes of RAM, 128K bytes of ROM, two RS-422 communication ports, and a built-in serial line diagnostic monitor. Interchangeable daughter boards, which provide the ability to directly connect to most research equipment, can be connected to the controller. The controller supports up to two of our digital I/O daughter boards plus almost any of the commercially available "SBX" family daughter boards. A wide variety of analog/digital and digital/analog, as well as parallel and serial SBX boards, are available from a number of suppliers.

Our daughter boards can also be connected directly to an IBM PC by using a small adaptor board (11.5 × 10 cm) plugged into the parallel printer port. In this way, researchers can run a single experiment using the power and convenience of ECBASIC or ECL at minimum expense. The major cost is that the existing CPU must be dedicated to experiment control while an experiment is running. An advantage of this as a "start-up" approach is that as research is expanded and funding becomes available, experiment controllers can be added with virtually no lost investment and only a minimal wiring change.

Each of the different daughter boards provides at least 20 outputs and four optically isolated inputs. If desired, the daughter boards can be partially populated in order to reduce costs. The four versions provide different current capabilities. The driver interface board (18 × 7.5 cm) contains 20 300-mA drivers, one 2-A relay, and one 10-A relay. The relay interface board (20 × 10 cm) contains 20 2-A relays and one 10-A relay. The triac interface board (23 × 13.5 cm) contains 20 5-A triacs, one 2-A relay, and one 10-A relay. A diagnostic tester/program development module (16.5 × 8 cm) is also available. It contains 24 LED indicators and four pushbutton input switches.

Installation

Our view is that the many wires between an apparatus and the control equipment should be relatively short, out of the way, and not subject to a lot of movement. Most reliability problems in computer/electronic equipment originate in wires and connectors. Therefore, the boards are ruggedized and were designed to be mounted in or on the experimental apparatus itself. We mount them, without an enclosure, on the back of the pigeon "intelligence panels" or on the back of the chambers. A typical installation would use an I/O board permanently mounted on the intelligence panel and attached to the various inputs and outputs through wires only a few centimeters long. Connections are made to a pluggable, screw-type, terminal strip on the daughter board. The daughter board would then be connected to the experiment controller, which can be also be mounted on the intelligence panel or some other convenient place (such as on the back of the experimental enclosure). The interconnection is with pluggable, insulation displacement, flat cable jumpers. A small, 5-V power supply is then attached to the experiment controller. A second power supply is used to provide whatever power is used to operate the lights and feeder for the experimental apparatus. It is attached directly to the daughter board. This allows researchers maximum freedom with respect to apparatus voltage and maximum "noise" isolation for the experiment controller. We mount our power supplies on the side of the pigeon chamber so that the on/off switch is convenient. In the just-described configuration, the apparatus is a self-contained,

remote, networkable station, which can be plugged in at any place along a single, four-conductor network link. Our chambers are about 60 m from the host computer, and nine chambers are plugged into the same wire.

As an alternative option, the I/O daughter board mounted on the intelligence panel may be connected to our parallel port adaptor board and plugged directly into an IBM PC or IBM PC clone. The adaptor board is powered by virtually any wall outlet 9-VAC power adaptor. ECBASIC or ECL is then run on the IBM PC itself, and events in the apparatus are directly controlled through the parallel port.

Using the System

The conduct of inquiry typically involves three distinct activities. Initially, the experimental procedure must be specified: for example, "turn on output AAA, if behavior BBB then do CCC, and record DDD." Second, the procedures must be enacted: for example, "run subject AAA with procedure BBB in apparatus CCC and label the obtained data file DDD." Finally, data analysis could involve, for example, "select data AAA, calculate BBB, and graph CCC." Each of these activities will be covered in turn.

Procedure specification. The researcher invokes a word processor or editor and enters the procedure specification program in the same way that any other text is entered. All the facilities of the editor can therefore be used. For example, frequently used blocks of code can be cut from one program and pasted into another program. The resulting program is then stored on the disk as a simple ASCII file.

Procedure enactment. The researcher invokes a communication program that transfers files between a disk and a serial port. Our software, ExpRun, was specifically written for supervising experiments run on our controllers. A version is available for the IBM PC, Macintosh, and Vax. Experiments are enacted by responding to its prompts for apparatus number, procedure, subject, weight, and resulting data file name. In addition to loading and running the designated procedure in the designated apparatus, ExpRun provides a variety of interconnect facilities and supports both summary data recording and transparent total data recording. If complete event logging is desired, classes of events designated in the ECBASIC or ECL procedure specification, such as pecks to key left, are automatically and transparently logged and uploaded to the designated disk file. Any other program that links serial ports and disk files could be used to provide a subset of these functions. Modem communication programs such as ProCom and White Knight can be used to load ECBASIC programs and retrieve summary data.

Data analysis. If summary data had been obtained, then the data files can be imported directly into Excel, JMP, Systat, and so on. The experiment controller allows the formatting of data for import into virtually any commercially available software. If, on the other hand, complete event records had been obtained, then the data must first be preprocessed. A program must be written that goes through the raw data file extracting the information of interest, such as, for example, the total number of pecks and total number of pecks in green that were followed by yellow. These data files can then be directly plotted or imported into an analysis program. We do not currently provide a select utility, but we have found the software to select data to be relatively easy to write.

Procedure Specification

There are three ways to specify an experimental procedure to be carried out by the experiment controller. Experiment Control BASIC, or ECBASIC, is the simplest and easiest to use. Specific experiment control instructions were developed so that the ease of using BASIC would be available, without the cumbersome problems typically associated with attempting to do real-time experiment control in Dartmouth BASIC. ECBASIC is very fast; it provides for 1-msec resolution, while automatically and transparently providing for total data collection. Experiment Control Language, or ECL, is an ALGOL-like language conceptualized specifically for running experiments. It also provides for 1-msec resolution and total data collection. However, it can carry out many more instructions per second than can ECBASIC and is therefore the preferred option if worst-case response latencies of less than a few milliseconds must be maintained, while also carrying out compute intensive feedback. A third option is to tailor ECBASIC or ECL to specific research needs. Source code is available to "not-for-profit" users who are in extremely demanding situations and who wish to optimize either to their needs. This third option is neither necessary nor advisable for the average researcher.

ECBASIC. ECBASIC is very much like all other BASICs. Assignment of variables, program flow, and data handling are the same. Any textbook on BASIC will provide examples of how to accomplish various tasks, and it is very likely that a BASIC guru is handy to any laboratory. This, in fact, was a major reason that we chose to work within the syntactical rules of BASIC for one of our procedure specification languages.

Appendix A provides a list of the program mode commands, functions, and system control commands available with ECBASIC.

The traditional BASIC program commands are available. FOR and NEXT set up and delimit a loop. GOTO unconditionally redirects program flow. GOSUB and RETURN provide the ability to exit program flow from a variety of places, execute a common subroutine, and subsequently return to the exiting point in the program. IF THEN provides for branching based on a comparison. ON allows for multiple path branching. The READ, DATA, and RESTORE commands provide the facility to use an array of predetermined values in a program. SLEEP halts program execution for a time period. Because ECBASIC is designed for real-time experiment control, this instruction is not the problem it was in Dartmouth BASIC. REM provides for comments in a program. DIM

allocates space for variables. STOP terminates program execution. END signifies the last line of a program.

A number of input/output commands have been added to BASIC. Most are self-explanatory. TURNON turns on and TURNOFF turns off one of the 48 outputs. PULSON and PULSOFF pulses on or off a particular output for some time and then returns that output to its previous state. LINK links outputs such as the magazine light and magazine together so that they can be conveniently manipulated as a single output. FREEZE, BLANK, and THAW are for storing the status of the outputs, turning off all outputs, and subsequently returning them to their previous state, as is often desired for a reinforcement cycle.

COUNT initiates the transparent counting of an input into a variable and is extremely useful. ECBASIC can automatically and transparently increment a designated variable every time a response occurs on a particular input. For example, following COUNT 1,PECKS, all responses on Key 1 automatically increment the variable PECKS. The researcher need only base decisions on the current value of that variable or on the difference between its current value and some previous value at an anchor point. CLICK transparently pulses a designated output on every occurrence of the specified input without further program intervention. This instruction can be used for feedback clicks or for running a cumulative recorder. CLOCK transparently stores the time of each occurrence of a specified input in a variable without further program intervention. This instruction can be used to conveniently store interresponse times when transparent, complete data logging is not used.

ONKEY and ONTICK specify interrupt routine pointers and allow the researcher to easily write concurrent procedures. The procedures for each of the concurrent processes can be written independently. If one of the specified events, such as a left-key peck or right-key peck, occurs, the program will "go sub" to the designated interrupt routine, and can then do whatever is appropriate in response to that event. In this way, the program can wait in a loop and whenever an appropriate input occurs, it can do the code appropriate for that behavior and subsequently return to the wait loop. ONKEY interrupts on an input, and ONTICK interrupts on a time.

A number of instructions have also been added to BASIC to facilitate the transmission of information up the network link (automatically and transparently stored to the disk file under the file name specified when the experiment was started by ExpRun). The experiment control program need not keep track of what file to write to, it need only put items into its output stream. REPORT specifies an input or timer as one that should be automatically logged to the network link each time it occurs. MARKER is a pseudoevent that is not necessarily associated with anything presented to the subject and is available for use when it simplifies the analysis of complete data logs. For example, if a marker is output 1 min before an event, then the rate in the minute before an event can be calculated without doing a backspace through the data stream after the event is found. SEND allows the program to send data other than the items designated in REPORT up to the appropriate disk file. MONIT allows the status of some variable to be monitored in real time while the experiment is being run. PRINT allows a program to output or print data to the network line when only summary data are collected. In this way, sending data to the appropriate disk file on the host computer is exactly the same as sending it to a printer. A hard copy of the resulting disk file can be subsequently obtained by a printing utility on the host computer, or the data can be imported directly to a data analysis package. LPRINT outputs data to a local printer if one is attached to the experiment controller's second serial port. In this way, a self-contained experiment that runs many months could directly output daily or weekly logs without the use of a host computer.

Instructions for more advanced programmers are available. STICK and HTICK start and halt a timer that can be used to generate an interrupt at the specified time. POKE stores a particular value into a specific memory location. CALL provides for a call to an assembly language routine. CLEAR clears the values from all the variables. ALLOC forces the allocation of a block of memory to the program. INPUT reads numerical values from the network line. OUTPB and OUTPW transmit a byte and a word, respectively, to the SBX port. CONFIG allows each input to be configured as normally open or normally closed and to have a specified debounce time.

Functions are also provided in ECBASIC. A function stands for some specific value and is therefore essentially the same as the variable *A* after a LET A=10 statement. A function can be used in any arithmetic expression just as if it were a number. ABS(X) is evaluated as the absolute value of *X*. SGN(X) is evaluated as a -1 or $+1$, depending on the value of its argument. RANGE(X,Y) is evaluated as a random number in the range *X* to *Y*. PROB(X) is evaluated as a 1,with the probability specified by *X*. RAND() provides a random integer. DRAND() provides a double precision random number. SELECT() selects a number from an array without replacement. MOD(X,Y) provides the remainder of *X* divided by *Y*. TIME(X) is an extremely powerful and therefore useful function. It is evaluated as the time since variable *X* was used to store the current time or to "anchor" time. For example, if at any point in the program, B is set equal to the current time or "time since 0" (B=TIME(0)), then TIME(B) or "time since B" can subsequently be used to provide the time since B. No clocks need ever be checked and no timers need ever be incremented by the procedure specification written by the researcher. ECBASIC does all of that transparently by itself.

The following functions are provided for the more advanced users. ADDR() returns the address of the specified variable. PEEK(X) returns the contents of the specified memory location. INCH() returns the value of a character read from the network line. LINCH() returns the value of a character read from the second serial port. INPB() and INPW() input a byte and a word, respectively, from

the SBX port. FREE() returns the amount of free memory.

System control commands are most applicable to the version of ECBASIC, which runs on the PC and controls the experimental apparatus through the printer port. An ECBASIC program can be stored on the disk with SAVE and retrieved with LOAD. The user can return to DOS with QUIT. The program space can be initialized with NEW, programs can be executed with RUN, and programs can be LISTed to the screen. Additionally, special network control commands (unnecessary to the average researcher) are available. ECHO and NOECHO are provided for efficient network communication. OFF directs a controller to disconnect from the network, and OFFRUN directs a controller to disconnect and then begin running the procedure specification program. The networked controllers LIST to the network and LLIST to a local printer.

If ECBASIC is run on the host and the experimental apparatus is connected via an interface plugged into the printer port, then both the CRT and keyboard are supported by ECBASIC. As a result, they can be used as part of the experimental apparatus. PRINT writes to the CRT, and INPUT and INCH read from the keyboard.

Appendix B provides a listing of a realistically complex experimental procedure implemented in ECBASIC. The task is a two-key, concurrent schedule with a changeover delay (COD). This procedure specifies that responses to either of two inputs occasionally produce a reinforcer. The appropriate time for a response to one or the other is independent and unpredictable (much like two one-armed bandits with payoffs governed by some timing mechanism). An added requirement is that following a change to the other alternative, no response will be reinforced for 2 sec. An attempt was made to solve it in a way that was neither flat-footed nor arcane. The following text is included to facilitate a careful examination of the program listing, rather than being self-explanatory.

Because it is good programming practice in ECBASIC to put interrupt routines at the beginning of a program for faster execution, the program starts (line 10) by jumping around those routines to line 1000. It then begins by allocating space for an array of variable-interval (VI) values and VI constants (1000), specifying the number of reinforcers (1010), and the mean VI duration (1020). Next, it sets up an array of Fleshler-Hoffman values for each of the concurrent VIs. (Our procedure is to use root values and expand them to the desired VI. Only six are used in this example to shorten the length of the listing [1030–1060].) It then specifies that each occurrence of input 1 and input 2 is to be temporally tagged and put into the data stream (1070, 1080). Markers that designate the beginning of each VI (1090, 1100) are then put into the data stream. It then specifies that the current VI interreinforcement intervals for the two keys are each to be a random element from the array of potential VI values (1110, 1120). The program then specifies where to go whenever there is a peck on Key 1 (1130) and whenever there is a peck on Key 2 (1140). The actual experiment begins by initializing the reinforcement in progress flag (2000) and the time remaining in the COD (2010). In order to keep track of which key was pecked last for the COD, LASTKY is initialized to zero (2020). The two VI values are started decrementing toward zero by starting Tickers 1 and 2 (2030, 2040). The houselights and the blue keylights behind the left and center keys are illuminated (2050), and then (unless reinforcement is occurring) the program waits for a peck (3000) in, what is in effect, a while loop. If a peck on Key 1 occurs, ECBASIC will go to subroutine at line 100 (as specified in the ONKEY 1,100 instruction [1130]). If a peck on Key 2 occurs, ECBASIC will go to subroutine at line 300 (as specified in the ONKEY 2,100 instruction [1140]).

Suppose an input on Key 1 occurs. As directed by the ONKEY 1 instruction, ECBASIC starts executing at line 100 and does the "what-to-do-when-Key-1-occurs procedure." In this example, the experimental procedure prohibits pecks immediately following a switch to a different key from being reinforced. As a result, the first thing ECBASIC does is to determine if the previous peck had been to the other key. If it had been, then the LASTKY pecked indicator is updated (110), the COD time remaining is set to 2 sec (120), the COD is begun decrementing (130), and program flow returns to the main wait loop. If it had not been to the other key (100), the LASTKY pecked indicator is updated (150); if it has been long enough since the first peck in this series (160) and if the VI time has elapsed (170), it will be reinforced (180 and the following statements). So that the program can get a new VI value for the appropriate schedule, the obtained reinforcer is labeled (180). The module of code that starts reinforcement is then entered. The two VI tickers are halted (190, 200). All outputs are turned off (210). The magazine light and magazine are operated (220). The start time of the reinforcement cycle is noted (230), and the program returns (240) to the main while loop at 3000. In this case, reinforcement is on (3000), so the program takes one of two paths depending on whether the reinforcer was from VI1 or VI2. If it had been VI2 (4000), then it would send a marker indicating that to the event stream (4030), sleep for the rest of a 4-sec reinforcement cycle (4040), and then turn off all outputs (4050) and decrement the reinforcement counter (4060) so that ECBASIC will know when to terminate the session. If enough reinforcers have occurred (4070), the network host is informed that the session is over (5000); otherwise, a new VI value for the just-reinforced schedule (4080, 4120, 4130) is obtained, and the program execution goes to the reinitialization section (4140).

ECL. Experiment Control Language, or ECL, was developed to enable researchers with tasks that require extremely short, worst-case delays but that are also computationally intensive to carry out. ECL forgoes the very simple and standardized syntax of BASIC in order to provide faster execution. As can be seen in Appendix C, many of the instructions in ECL are the same as those

in ECBASIC; the syntax is more like ALGOL. The specific syntax of ECL was the result of our attempt to generate a language that was easy to use, that executed very quickly, and that was reasonable to implement. Because ECL has the feel of a number of structured languages, researchers with prior experience and/or access to reference materials or gurus familiar with PASCAL or ALGOL should find the specification of experimental procedures relatively easy with ECL.

ECL has a variety of program flow commands. Labels can be used to provide a symbolic destination for program flow. GOTO redirects program flow to a label. SUBROUTINE defines a user subroutine as all the code between the subsequent BEGIN and END. It is terminated by a RETURN. PRIORITY can be used to set the priority level of a subroutine. CALL can be used to redirect program flow to a user subroutine. For example, CALL OBSERV executes the code in the OBSERV subroutine and then returns to continue program execution. WHILE reexecutes the same line of code or executes all statements between its BEGIN and END as long as its condition is nonzero. FOR executes a block of statements a number of times. It is essentially equivalent to a FOR/NEXT loop in BASIC. For example, FOR I=1,10 executes the code following its semicolon, or the code within its BEGIN END boundaries, 10 times. The IF statement conditionally executes a block of statements. If the condition is true, the THEN block is executed. If it is false, the next consecutive instruction is executed or the ELSE block is executed (if an ELSE is listed). CONTINUE forces the next loop iteration in a FOR or WHILE loop. BREAK forces an exit from a FOR or WHILE loop. STOP terminates the program. IDLE halts program flow until an interrupt occurs. WAITIN halts program flow for a specified time or until a specified input. It returns the initiating event as the value of a variable. For example, the instruction WAITIN(a,timlmt,key1,key2) will wait until the time specified by "timlmt," an input on "key1," or an input on "key2." When one of those three events happens, program flow is continued. The value of "a" can then be used to direct program flow accordingly. The instruction CONC is basically WAITIN optimized for running concurrent schedules. Program flow falls through when "key1" occurs after "time1," "key2" occurs after "time2," or "key3" occurs after "time3." A COD can be specified in the instruction. The value of "a" can then be used to direct program flow.

A number of input/output instructions are available with ECL. INPUT specifies which hardware input (1 through 8) is to increment which variable, whether the input is a contact make or break, the debounce time, and whether or not the response is to be automatically recorded. The variable (e.g., LFTPKS) specified in INPUT can subsequently be used in expressions. For example, when the pecks are less than some number past the starting value, one block of code can be executed, and when past that value, some other block of code can be executed. TURNON turns on the specified output. PULSON turns on an output, halts program flow for a duration, and then turns the output off. For example, a 4-sec reinforcement cycle could be implemented with PULSON (REIN,4000). TURNOFF turns off the specified output. PULSOFF turns off an output for a duration. The COMBINE command ties together outputs so that activating any one output activates them all. For example, the magazine and magazine light could be combined: operating the magazine would subsequently also operate the magazine light. CLICK sends a pulse to the designated output on every occurrence of the designated input.

FREEZE saves the status of all the outputs. BLANK turns off all outputs, and THAW restores the frozen condition of all the outputs. These instructions are useful when turning off all outputs for a short duration, such as during reinforcement or time out. SEND puts a data item into the data stream to be automatically collected by ExpRun (the communication module) in the host computer. For example, SEND(DATA(J)) can be used to pass an element of a data array to the host computer. MARKER puts a marker into the data stream. For example, MARKER(42) puts Marker 42 into the data stream. This marker can subsequently be used to indicate to the data-analysis program that the second link of a tandem schedule had been entered at that point in the data stream.

ECL also provides random-number functions, all of which can be used in expressions. RAND returns a random number between 0 and 1,000. XRAND returns a random number between 0 and 100,000,000. SEED forces a specific seed to the random number generator in order to obtain repeatable, random sequencing. RANGE returns a random number within the specified range. For example, a random set of waits from 1 to 10 sec could be implemented with SLEEP(RANGE(1,10)*1000). With PROB, a path can be set to occur with a specified probability. XPROB provides probabilities across the same extended range as XRAND. For example, PROB(500) will cause the block of code following a THEN to be executed half the time; otherwise, the code following ELSE will be executed.

Several commands are provided for timekeeping. SLEEP halts program flow for a specified time. For example, SLEEP(1000) halts program flow 1 sec. Variables that can be set to automatically decrement every 1 msec are convenient to implement time bases that run only during a portion of the schedule. For example, a VI value that is to run during some parts of the schedule can be easily implemented with a ticker. STICK starts a ticker decrementing a specified variable from its initial value. This process is automatic and does not require program intervention. The variable can subsequently be used to make decisions based on time. HTICK halts the specified ticker. TIME is equivalent to the same instruction available in ECBASIC. TIME will return a time difference between the current time and any expression. For example, TIME(FISTRT) provides the time since FISTRT. The instruction makes it easy to specify that, for example, if the elapsed time since the anchor point is less than 60 sec, one block of code could be ex-

ecuted; otherwise, the next consecutive instruction would be executed. Real time cannot be lost and cumulative timing errors cannot occur with this instruction, in that it represents the total ticks since the beginning of the session. SLEEP halts program flow. In contrast, a time duration implemented with a ticker or with TIME need not "lock up" the program. Even though most timing and counting is done transparently by ECL itself, it is sometimes handy to have the program carry out activities while timing.

ABS(X) is evaluated as the absolute value of *X*. SGN(X) is evaluated as a −1 or +1 depending on the value of its argument. MOD(X,Y) provides the remainder of *X* divided by *Y*. TICKS is a predefined variable that is incremented every millisecond. Variables must be allocated or defined at the beginning of a program with either the VARIABLE or the WORD statement. They can be simply initialized or can be set to a value. DEFINE can be used to define a symbol as a constant. This clarifies programming by providing names for things that are actually numbers. Output 20 can be defined as red, and subsequently the instruction TURNON RED can be used. REINF is a predetermined variable specified by ExpRun when the program is loaded into the controller. ExpRun prompts the user for the desired number of reinforcers when the experiment begins. The ECL program may then use it to determine when to terminate the procedure.

Appendix D provides a listing of a sample ECL program. It is the same concurrent VI schedule with a COD as was previously implemented in ECBASIC. It was implemented without the CONC instruction in order to provide a realistically complex procedure specification. As can be seen, ECL is "structured" but more verbose than ECBASIC. The text is intended as an adjunct to careful scrutiny of the program listing. The experiment program is in three sections. The first is initialization; the second is a WHILE loop that executes until the desired number of reinforcers for the session have been provided. The final section contains the two subroutines that specify the details of each of the concurrent schedules (in this case, they are the same).

The first two instructions of the program combine the outputs connected to the magazine and magazine light and name that output FOODUP. The third and fourth instructions combine the houselight and the blue keylights behind Key 1 and Key 2 and name that output KEYLTS. The next three instructions allocate the following variables: (1) the six VI root constants for expansion to a Fleshler-Hoffman series (only six values in order to save listing space) and the mean VI value; (2) the time remaining in the interreinforcement interval for the "current" VI1 and VI2, and the time remaining in the "current" COD; and (3) "semaphores" noting which key had been pecked last and which VI had been reinforced. Next, the VI root constants and the mean VI value (30 sec) are specified.

Markers indicating the start of each VI are put into the data stream, and the first two interreinforcement intervals of the session are then generated. They are random elements from the VI constants multiplied by the mean VI value and rounded. The two input instructions specify several things; that the hardware inputs 1 and 2 are the first and second input (1,1 and 2,2), should be configured as normally closed (+), all occurrences should be temporally tagged and passed into the event stream (report), the names of the subroutines to be executed whenever one of the inputs occurs (KEY1S AND KEY2S), and the debounce time for each input (30 msec).

The experiment proper is then begun. While there are reinforcers yet to be given, the loop executes. Reinforcement on (RNFON) (i.e., a reinforcement has been scheduled), COD time remaining (CODTIM), and the last key to be pecked (LASTKY) are each initialized to zero.

The current interreinforcement intervals are begun decrementing toward zero (i.e., the condition that allows the next peck to be reinforced) (CALL STICK instructions). The keylights are turned on, and the main WHILE loop is entered. The program waits in this empty BEGIN/END loop for a peck, unless a reinforcer is scheduled. Suppose a peck on Key 1 occurs. As specified in the INPUT instruction, the subroutine for a keypeck on Key 1 (SUBROUTINE KEY1S) begins executing. If the prior peck had been to Key 2 (IF LASTKY.EQ.2), then LASTKY is updated to indicate that Key 1 is now the LASTKY pecked. The COD time is reset to 2 sec, because the procedure is to prohibit pecks within 2 sec of a change from being reinforced. The COD is begun decrementing to zero (the condition that permits the next peck to be reinforced) (CALL STICK instruction), and the program returns to the main WHILE loop to wait for further pecks. If the prior peck had not been on Key 2, then LASTKY is updated (for the case where LASTKY had been zero) (LASTKY = 1 instruction), then it is determined whether the COD and the VI have elapsed. If both have elapsed, then a reinforcer is scheduled (RNFON = 1), a marker indicates that event, and the program returns to the main wait loop. This time back to the main wait loop a reinforcer is scheduled, therefore program flow falls through (the empty BEGIN/END loop), the VI timers are stopped (HTICK instructions), the keylights are turned off, and food is presented for 4 sec (CALL PULSON instruction). The program then determines which schedule paid off (IF(RNFON.EQ.1)THEN and IF(RNFON.EQ.2)THEN); this information is sent to the data stream (with appropriate markers), and a new interreinforcement interval is generated for that VI (VI1TIM= or VI2TIM= instructions), the reinforcement counter is decremented (REINF=REINF-1), and the program goes back to the beginning of the "session is running" loop. The schedule continues unless the specified number of reinforcers had been delivered (when REINF is not greater than zero), in which case the program stops (CALL STOP).

Availability

Both the hardware and software are in the public domain for "not-for-profit" users. A system may be obtained in two ways. Researchers may purchase preassembled,

working systems or may build the systems themselves. The boards are solder masked to minimize the possibility of solder shorts, and they are silk screened to designate the parts and their orientation. We have experienced very few assembly problems ourselves. Unfortunately, our experiences with our previous controller were not universally successful. Some labs that assembled their own boards experienced intermittent failures on some boards, whereas other labs experienced no problems whatsoever with any of their boards. Our solution has evolved to recommend that prebuilt boards be purchased unless cost is of an overriding concern or if the lab technician is known to be sufficiently competent.

REFERENCES

COOPER, L. D., GARCIA, R., & GIBBON, J. (1988). The labtop Macintosh: An interface and communications software for experiment control of animal learning research. *Behavior Research Methods, Instruments, & Computers*, **20**, 88-92.

D'ANDREA, J. A., & KNEPTON, J. (1988). Construction and implementation of a low-cost electronic experiment control interface. *Behavior Research Methods, Instruments, & Computers*, **20**, 97-99.

PALYA, W. L. (1988). An introduction to the Walter/Palya Controller and ECBASIC. *Behavior Research Methods, Instruments, & Computers*, **20**, 81-87.

PALYA, W. L., & WALTER, D. E. (1993). *Document Set for the High Performance Experiment Controller*. Unpublished manuscript, Jacksonville State University, Department of Psychology, Jacksonville, AL.

PEVEY, M. E. (1988). Using an IBM PC to network Walter/Palya experiment controllers. *Behavior Research Methods, Instruments, & Computers*, **20**, 100-103.

WALTER, D. E. (1988). The Walter/Palya experiment controller users' group meeting and help session. *Behavior Research Methods, Instruments, & Computers*, **20**, 104-105.

WALTER, D. E., & PALYA, W. L. (1984). An inexpensive experiment controller for stand-alone applications or distributed processing networks. *Behavior Research Methods, Instruments, & Computers*, **16**, 125-134.

WEISMAN, R., & PALYA, W. L. (1988). Development and operating environments for a network of Walter/Palya experiment controllers on the Macintosh computer. *Behavior Research Methods, Instruments, & Computers*, **20**, 93-96.

APPENDIX A
Operations Currently Available with ECBASIC

Program Mode Commands			
ALLOC	FOR/NEXT	ON GOSUB	REM
BLANK	FREEZE	ON GOTO	REPORT
CALL	GOSUB	ONKEY	RESTORE
CLEAR	GOTO	ONTICK	RETURN
CLICK	HTICK	OUTPB	SEND
CLOCK	IF THEN	OUTPW	SLEEP
CONFIG	INPUT	POKE	STICK
COUNT	LINK	PRINT	STOP
DATA	LPRINT	PULSOFF	THAW
DIM	MARKER	PULSON	TURNOFF
END	MONIT	READ	TURNON
Functions			
ADDR	INCH	MOD	RANGE
ABS	INPB	PEEK	SELECT
DRAND	INPW	PROB	SGN
FREE	LINCH	RAND	TIME
System Control Commands			
ECHO	LOAD	OFF	RUN
LIST	NEW	OFFRUN	SAVE
LLIST	NOECHO	QUIT	

APPENDIX B
Concurrent VI with COD in ECBASIC

```
10 GOTO 1000
100 IF LASTKY <> 2 THEN GOTO 150
110 LASTKY=1
120 CODTIM=2000
130 STICK 3,CODTIM
140 RETURN
150 LASTKY=1
160 IF CODTIM > 0 THEN RETURN
170 IF VI1TIM > 0 THEN RETURN
180 RNFON=1
190 HTICK 1
200 HTICK 2
210 BLANK
220 TURNON 2,3
230 RNSTRT=TIME(0)
240 RETURN
300 IF LASTKY <> 1 THEN GOTO 350
310 LASTKY=2
320 CODTIM=2000
330 STICK 3,CODTIM
340 RETURN
350 LASTKY=2
360 IF CODTIM > 0 THEN RETURN
370 IF VI2TIM > 0 THEN RETURN
380 RNFON=2
390 GOTO 190
400 DATA 884,2897,5424,8822,14055,27918
1000 DIM VIVALS(6),VICON(6)
1010 REINF=50
1020 VIAVG=30
1030 FOR I=1 TO 6
1040 READ VICON(I)
1050 VIVALS(I)=((VICON(I)*VIAVG)+5)/10
1060 NEXT I
1070 REPORT INPUT,1
1080 REPORT INPUT,2
1090 MARKER 100
1100 MARKER 101
1110 VI1TIM=VIVALS(RANGE(1,6))
1120 VI2TIM=VIVALS(RANGE(1,6))
1130 ONKEY 1,100
1140 ONKEY 2,300
2000 RNFON=0
2010 CODTIM=0
2020 LASTKY=0
2030 STICK 1,VI1TIM
2040 STICK 2,VI2TIM
2050 TURNON 4,9,21
3000 IF RNFON=0 THEN GOTO 3000
4000 IF RNFON <> 1 THEN GOTO 4030
4010 MARKER 200
4020 GOTO 4040
4030 MARKER 201
4040 SLEEP 4000-TIME(RNSTRT)
4050 BLANK
4060 REINF=REINF-1
4070 IF REINF < 1 THEN GOTO 5000
4080 IF RNFON <> 1 THEN GOTO 4120
4090 VI1TIM=VIVALS(RANGE(1,6))
4100 MARKER 100
4110 GOTO 2000
4120 VI2TIM=VIVALS(RANGE(1,6))
4130 MARKER 101
4140 GOTO 2000
5000 PRINT "*DONE*"
9999 END
```

APPENDIX C
Operations Currently Available with ECL

Program Flow Commands

BEGIN	CONTINUE	IDLE	STOP
BREAK	END	IF THEN ELSE	SUBROUTINE
CALL	FOR	PRIORITY	WAITIN
CONC	GOTO	RETURN	WHILE

I/O Commands

BLANK	FREEZE	PULSON	THAW
CLICK	INPUT	PULSOFF	TURNON
COMBINE	MARKER	SEND	TURNOFF

Time/Probability Commands

HTICK	RANGE	STICK	XPROB
PROB	SEED	TIME	XRAND
RAND	SLEEP		

Functions, Variables, and Defined Variables

ABS	MOD	SGN	VARIABLE
DEFINE	REINF	TICKS	WORD

APPENDIX D
Concurrent VI with COD in ECL

```
COMBINE 2,3;
DEFINE FOODUP=2;
COMBINE 4,9,21;
DEFINE KEYLTS=4;

VARIABLE VICON[6], VIAVG;
VARIABLE VI1TIM,VI2TIM,CODTIM;
VARIABLE LASTKY, RNFON;

VICON[1] = 884;
VICON[2] = 2897;
VICON[3] = 5424;
VICON[4] = 8822;
VICON[5] = 14055;
VICON[6] = 27918;

VIAVG=30000;

CALL MARKER(100);
CALL MARKER(101);
VI1TIM = ((VICON[RANGE(1,6)] * VIAVG) + 500) / 1000;
VI2TIM = ((VICON[RANGE(1,6)] * VIAVG) + 500) / 1000;

INPUT 1,1,+,,RECORD,KEY1S,30;
INPUT 2,2,+,,RECORD,KEY2S,30;

WHILE(REINF .GT. 0)        /* "session is running" loop */
BEGIN;
  RNFON=0;
  CODTIM=0;
  LASTKY=0;
  CALL STICK(1,VI1TIM);
  CALL STICK(2,VI2TIM);
  CALL TURNON(KEYLTS,RECORD);
  WHILE (RNFON .EQ. 0)      /* "main wait" loop*/
   BEGIN;
   END;
  CALL HTICK(1);
  CALL HTICK(2);
  CALL TURNOFF(KEYLTS,RECORD);
  CALL PULSON(FOODUP,4000,RECORD);
  IF (RNFON .EQ. 1) THEN    /* Key 1 paid off */
   BEGIN;
    CALL MARKER(100);
    VI1TIM = ((VICON[RANGE(1,6)] * VIAVG) + 500) / 1000;
   END;
  IF (RNFON .EQ. 2) THEN    /* Key 2 paid off */
   BEGIN;
    CALL MARKER(101);
    VI2TIM = ((VICON[RANGE(1,6)] * VIAVG) + 500) / 1000;
   END;
  REINF=REINF-1;
END;
CALL STOP;

SUBROUTINE KEY1S(1);        /* when Key 1 peck */
BEGIN;
  IF (LASTKY .EQ. 2) THEN   /*  key switch so restart COD */
   BEGIN;
    LASTKY=1;
    CODTIM=2000;
    CALL STICK(3,CODTIM);
    RETURN;
   END;
  LASTKY=1;
  IF (CODTIM .GT. 0) THEN RETURN;
  IF (VI1TIM .GT. 0) THEN RETURN;
  RNFON=1;
  CALL MARKER(200);
  RETURN;
END;

SUBROUTINE KEY2S(1);        /* when Key 2 peck */
BEGIN;
  IF (LASTKY .EQ. 1) THEN   /*  key switch so restart COD */
   BEGIN;
    LASTKY=2;
    CODTIM=2000;
    CALL STICK(3,CODTIM);
    RETURN;
   END;
  LASTKY=2;
  IF (CODTIM .GT. 0) THEN RETURN;
  IF (VI2TIM .GT. 0) THEN RETURN;
  RNFON=2;
  CALL MARKER(201);
  RETURN;
END;
```