PROGRAM ABSTRACTS/ALGORITHMS

A software system for real-time laboratory use of TRS-80 microcomputers

MICHAEL PERONE

West Virginia University, Morgantown, West Virginia

Time is a critical variable in most behavioral research. Investigators using microcomputers to control and monitor laboratory events must be able to program temporal contingencies (e.g., variable-interval schedules of reinforcement), specify the duration of stimulus presentation (e.g., response feedback), and collect temporal data (e.g., interresponse times). This report offers a software solution to these problems of real-time experimental control and data collection, for use with TRS-80 microcomputers. An assembly language program, running independently of other programs, provides a set of software clocks that can be set, read, and reset by programs written in BASIC and other high-level languages. Additional routines provide for efficient control of output relays, including timing routines to allow the relays to be pulsed. Time resolution is sufficient for most behavioral research, with units of 33.33 msec on TRS-80 Models III, 4, and 4P and 25 msec on Model I.

Several versions of the software have been in use for over 3 years in human and animal conditioning experiments, some involving complex arrangements of concurrent interval schedules in independent experimental chambers. The programs have been used on Model I (Baron, Menich, & Perone, 1983) and Model III (Galizio, Smaltz, & Spencer, 1984) and are in current use on Model 4 (in Model III mode).

Principles of Operation. Except during disk and cassette read/write functions, the TRS-80's central processor is interrupted at regular intervals. A built-in circuit periodically signals the processor, causing it to stop executing its current program, execute a "background" program residing in a protected area of memory, and, afterwards, resume execution of the original program. The interrupts occur at a rate of 30 Hz in Models III, 4, and 4P (operating in Model III mode), and at 40 Hz in

Model I. A program called an interrupt service routine (ISR), executed as each interrupt occurs, can count the interrupts and thereby keep time. The TRS-80's disk operating system (DOS) automatically loads an ISR that arranges the real-time clock described in TRS-80 documentation (e.g., see Tandy Corporation, 1980b). This is the clock that returns the time and date when the BASIC statement `PRINT TIME$` is executed. Additional clocks can be supplied by another ISR that patches into the DOS's ISR. The clocks can be set, read, and reset independently of one another, even though they have a common time base (the stream of interrupts). Because execution of the ISR (system or user supplied) is "interrupt driven" by the computer's own circuitry, time is kept without an external hardware clock and without programmer intervention. As a background task, the ISR operates independently of other software—such as a BASIC program monitoring responses in experimental chambers—even though the clocks can be accessed (set, etc.) by BASIC programs.

ISR Version 4.3 arranges seven clocks, giving the programmer a total of eight, including the one arranged by the DOS. The ISR also arranges eight pulse timers, to pulse output relays for periods of up to 8.5 sec (10.2 sec on Model I), so that the clocks need not be used for such purposes as response feedback and cumulative recorder operation.

Clocks. Each clock repeatedly updates four time counters representing ticks (33.33 msec, or 25 msec on the Model I), seconds, minutes, and hours. The range of possible values is as follows: ticks, 0-29 (0-39 on Model I); seconds, 0-59; minutes, 0-59; and hours, 0-255. A byte of memory is used for each counter. Accessing the counters from a BASIC program is straightforward: The `PEEK` statement is used to read a counter, and `POKE` is used to set or reset. Two types of clocks are provided: "Count-up" clocks mimic running-time meters (Clocks 1-4), and "count-down" clocks mimic universal timers (Clocks 5-7). The count-up clocks are especially useful for measuring variable intervals (e.g., postreinforcement pauses), and the count-down clocks for controlling specified intervals (e.g., stimulus durations). However, either can serve measurement or control functions.

In most applications, a count-up clock is cleared and later read to determine how much time has elapsed. The clock is cleared by `POKE`ing the ticks, seconds, minutes, and hours counters with 0 and is read by `PEEK`ing at these counters. A count-down clock is not normally cleared; instead, its counters are `POKE`d with specific values representing a predetermined amount of time to be counted off. `PEEK`ing at the hours counter determines whether the specified interval has passed. This counter is set to

Development of the software was supported by Grant RII-8308469 from the National Science Foundation to the University of North Carolina at Wilmington and Grant AG02513 from the National Institute on Aging to the University of Wisconsin-Milwaukee. Thanks are due Stephen R. Menich for assistance in testing the software and Mark Galizio for advice regarding the manuscript. An earlier version of the paper was presented at the Association for Behavior Analysis convention, Nashville, May 1984. Requests for reprints should be sent to Michael Perone, Department of Psychology, West Virginia University, P.O. Box 6040, Morgantown, WV 26506-6040.

255 to "flag" the timed-out condition and holds there until set to a new value.

Output Relay Control. Besides the clocks, ISR Version 4.3 provides for control of output relays with minimal attention from the BASIC programmer. Output control requires a hardware interface to translate the computer's logic signals to relay latches that can operate lights, feeders, and other laboratory equipment.

A port-addressing scheme is used on TRS-80s, such that each of 256 ports can transmit or receive a byte of information via eight parallel data lines. When a byte is output to a port, the 8 bits (binary digits) constituting the byte represent the on/off status of eight output relays (Relays 0-7) assigned to that port. A simple way to decide the value of an output byte is to note that each relay corresponds to a power of 2. To latch Relay 0, the value of the byte is 2 raised to the 0th power; to latch Relay 1, the byte is 2 raised to the 1st power. Thus, the values to latch Relays 0 through 7 are 1, 2, 4, 8, 16, 32, 64, and 128, respectively. Adding these values allows for latching several relays at once; for example, a value of 5 would latch Relays 0 and 2.

ISR Version 4.3 provides subroutines to latch and unlatch relays at Ports 0 and 1 (the port address of a set of eight relays is determined by a switch on the hardware interface). Use of these routines, instead of the BASIC statement OUT, simplifies programming. The routines keep track of the status of all of the relays and block the common errors of attempting to latch a relay that is already latched or to unlatch one that is already unlatched. The routines are accessed from BASIC programs using the standard procedure for calling machine language subroutines, and the relay value is passed as an argument in the call statement. For example, the code to latch Relay 6 at Port 0 would be $X=USR0(64)$, where 64 is the value of Relay 6. The subroutine would latch this relay, store the fact that the relay is latched in an output status byte for future reference, and maintain the status of the other seven relays at the port. A separate routine unlatches relays at Port 0, and another pair of routines latches and unlatches relays at Port 1.

Pulsing the Relays. Eight pulse-timer routines are arranged by ISR Version 4.3. Pulsers 1 through 4 can be used with any of the relays at Port 0, and Pulsers 5 through 8 can be used at Port 1. BASIC programs access each pulser by PEEKing and POKEing three memory locations designated here as Relay, Duration, and Activity. The Relay byte holds the value of the relay to be pulsed. The Duration byte holds the length of the pulse interval in ticks, up to a maximum of 255. At each interrupt, the ISR decrements the value until 0 is reached, at which point the designated relay is unlatched. The Activity byte is used to switch the pulser on and off. The pulser is activated by POKEing its Activity byte with a 1. At the end of the pulse, the routine deactivates itself by clearing this byte.

The following sequence is used to pulse a relay: (1) The availability of the desired pulser is verified by PEEKing its Activity byte; if the value of this byte is 1, the pulser is already active and cannot be used. (2) The appropriate

output control subroutine (see previous section) is called to latch the relay(s). (3) The length of the pulse, in ticks, is set by POKEing the Duration byte. (4) The value of the relay(s) is POKEd to the Relay byte. (5) The pulser is activated by POKEing a 1 to its Activity byte. The exact order of the steps is not critical, but Step 1 must come first and Step 5 last.

BASIC Program Speed. In the real-time environment of the behavioral laboratory, programs should execute rapidly so that events can be monitored accurately and experimental contingencies arranged precisely. Even well-written BASIC programs may be too slow for some procedures, for example, those involving multiple chambers. (Note, however, that the speed of BASIC programs does not affect the precision of the clocks and pulsers, because these are assembly language background tasks that execute at every interrupt, regardless of the BASIC program in the foreground.) One way to increase speed is to program in assembly language, but, unfortunately, this is a laborious chore. A more palatable way is to use a compiler to translate BASIC programs into machine code. Several compilers are available for TRS-80 (see Brothers, 1985, and Cangro, 1984, for reviews). The compiler described by Garipey (1982; "Zbasic 2.2," available from Simutek Computer Products, 4897 E. Speedway, Tucson, Arizona 85712) greatly increases the execution speed of programs written in BASIC, and it is possible to specify the memory location of the compiled program so that the program does not interfere with the ISR that arranges the clocks and pulsers.

System Requirements. ISR Version 4.3 was designed for use on TRS-80 Models I, III, 4 (in Model III mode), and 4P (in Model III mode), equipped with at least one disk drive and 48K of user memory (designated as 64K on Models 4 and 4P). The program occupies about 1K of memory; the starting and ending addresses are about 1.6 and .6K below the top of memory, respectively. The upper .6K of memory is unoccupied because it is used by the TRSDOS 1.3 "DO" function (Tandy Corporation, 1980a). The program can be relocated if the DO function is not needed or if there is less than 48K of user memory available. The output control routines were developed for use with the Interfacer 80 (Alpha Products, 79-04 Jamaica Avenue, Woodhaven, New York 11421), which provides eight relay outputs (and eight inputs); however, the routines should work with any output interface that can be attached to Models I, III, 4, and 4P.

Availability. The following listings are available from the author upon request: the assembled source code for ISR Version 4.3; a BASIC program to load and save the ISR without an assembler; and user instructions, including programming examples.

REFERENCES

- BARON, A., MENICH, S., & PERONE, M. (1983). Reaction times of younger and older men and temporal contingencies of reinforcement. *Journal of the Experimental Analysis of Behavior*, 40, 275-287.
- BROTHERS, H. (1985, January). Basic compiler comparison. *80 Micro*, p. 45.

- CANGRO, R. (1984, July). WIBASIC: Powerful compiler/interpreter. *80 Micro*, pp. 186, 188, 190, 192.
- GALIZIO, M., SMALTZ, S. C., & SPENCER, B. A. (1984). Effects of ethanol and naltrexone on free-operant avoidance behavior in rats. *Pharmacology, Biochemistry and Behavior*, **21**, 423-429.
- GARIEPY, A. (1982). *Zbasic Version 2.2* [Computer program and manual]. Tucson, AZ: Simutek Computer Products.
- TANDY CORPORATION. (1980a). *TRS-80 Model III disk system owner's manual*. Fort Worth, TX: Author.
- TANDY CORPORATION. (1980b) *TRS-80 Model III operation and Basic language reference manual*. Fort Worth, TX: Author.

(Revision accepted for publication January 8, 1985.)