COMPUTERS AND BIOMEDICAL RESEARCH 5, 673-683 (1972)

SNAP: A Computer Language for Control of Psychoacoustic Experiments*

DAVID B. MOODY

Kresge Hearing Research Institute, University of Michigan Medical School, Ann Arbor, Michigan 48104

Received June 27, 1972

A real-time programming language is described which contains specialized statements for the control of auditory experiments including routine audiometric tests. The program statements correspond closely to prose descriptions of the experimental contingencies making it an easy language to learn and to use. Programs may be prepared and compiled using standard small computer software, and the only specialized program required is a real-time interpreter. Provisions are made for monitoring the experiment in progress and for making any desired changes in the experimental parameters. The language has been implemented on a 4K PDP8-L.

One of the stumbling blocks in applying digital computers to the control of psychoacoustic experiments or, for that matter, to the control of any real-time process has been the lack of a high-level programming language which is relatively simple to learn and to use. Languages such as Fortran and Basic have been around for many years and are relatively simple, but they are designed to deal with sophisticated mathematical manipulations (number-crunching, as it is known in the trade). What is needed is a Fortran-like language which, instead of adding, subtracting, squaring, and the like, is able to count responses, time intervals, record events, and turn devices on and off. Such languages do exist (*1*, *2*) and are available as components of computer systems sold by several manufacturers (Grason-Stadler: SCAT; Lehigh Valley: INTERACT; Texas Instrument: 960A SYSTEM). The language which will be described in the present paper, SNAP (State Notation in Auditory Psychophysics), is one we have developed which has some of the features of the languages mentioned but which was written for the specific purpose of controlling psychoacoustic experiments. Although our requirements were to control behavioral tests in nonhuman primates (*3*), the language itself is entirely suitable for use with humans.

* This research was supported by grants NS-05077 and NS-05785 from the National Institutes of Health and a grant from the Upjohn Corporation.

Any computer language which controls or responds to external devices must, of necessity, be written for a particular computer system, and will work only on systems identical to the one for which it was written. We believe, however, that the basic idea embodied in SNAP can be easily generalized to systems other than our own. In developing SNAP, we were guided by two main considerations: First, that the language be as simple as possible to learn to use, and that this simplicity should arise from the fact that programming statements be reasonable approximations to prose descriptions of the events taking place in the experimental situation; second, that the programming system should, as much as possible, make use of existing programs already available as supporting software from the computer manufacturer. The result of our efforts, we believe, has achieved both goals: although the grammar is somewhat stilted, it is reasonably prose-like, and the program preparation requires only standard paper tape editor and compiler programs.

In the present system, a paper tape is prepared using the Editor program supplied by the computer manufacturer. This tape, the symbolic or source tape, is punched with codes which represent symbolic characters (alphabetic, numeric, and punctuation) which spell out the words defined in the language. This tape, together with a tape which provides the definitions for the words in the language (a dictionary tape), is fed into one of the compiler programs supplied with the computer. The compiler then generates another tape which is punched with binary coding and is the result of translating the symbolic code from the source tape according to the definitions provided on the dictionary tape. This binary tape is not itself in a form which can be understood by the computer hardware (i.e., it is not machine language); rather it is a code which requires an interpreter program or operating system to carry out the operations specified by the original source program. This operating system program is written in machine language and serves the function of reading the binary program tape into the computer's memory, and then translating the bit patterns from that tape into the operations which they represent.

Experimental procedures can be considered as sequential systems. Such systems have received extensive mathematical treatment (*4*, *5*), and a theory, called the theory of finite automata, has been developed which can be used to describe any sequential system. Snapper et al. (*1*) have provided an excellent example of how part of the theory can be applied to the description of experimental procedures (reinforcement schedules) which have similarities to procedures often encountered in psychoacoustic research.

The basic unit of the theory of finite automata, and of the present language, is called the state. A given state defines the conditions that are in effect at any given moment in the experiment. Usually, one or more of the conditions defined within any particular state is an event or events which will cause a transition from the present state to some other state. These events can be the occurrence of some response, the passage of a specified amount of time, the occurrence of an event elsewhere in the program, or a variety of other things. States are grouped into state sets. Only one

state in a given set may be active, but any number of sets may operate simultaneously. Transitions may occur to or from any state within a given set, but not from one set to another.

An example of a possible state description is as follows:

```
STATE 5                          (Names the state)
TONE 1 ON                        (Turn on a stimulus)
RECORD ON COUNTER 7              (Record the occurrence of this trial)
TIME 3 SEC THEN GO TO STATE 1
COUNT 1 R2 THEN GO TO STATE 10
END                              (Close this state)
```

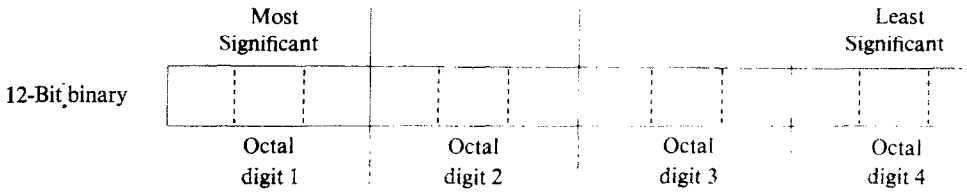
In the preceding example, when State 5 is entered as a result of a transition from some other state, several events occur nearly simultaneously. First, a device called Tone 1 (a tone switch) is turned on. Second, counter 7 is incremented by one count to indicate that Tone 1 was presented. Third, the program sets up internal registers which will note the passage of 3 sec and the occurrence of one of the subject's responses (R2). If 3 sec occurs before the subject makes 1 R2, the program will transfer to State 1; if R2 occurs first, transfer will be to State 10. These states will define another set of conditions which will be in effect until a transition occurs to yet another state.

This example of a particular state illustrates the power of the present system for programming experimental contingencies. Unfortunately, it is somewhat idealized for purposes of clarity. In actual practice, the statement TIME 3 SEC, THEN GO TO STATE 1 is written TIME SEC; 3; STATE 1. Similarly, the "Count" statement is written COUNT R2; 1; STATE 10 and the "Record" statement is RECORD 7. The reasons behind this stilted grammar arise from the use of standard compilers and will become obvious as the internal structure of the language is discussed.

Our version of this language was written for a 12-bit word length computer (PDP8-L). A 12-bit binary word may be represented as a 4-digit octal number between 0000 and 7777, in which each octal digit represents three binary bits. Figure 1 may help to clarify the correspondence between 12-bit binary and 4-digit octal numbers. It is important to understand this correspondence since the presence or absence of given bits is the basis on which the operating program determines which operations it must carry out for each state.

Major Instruction Groups

Program instructions are divided into eight major classes determined by the most significant octal digit. Examples of six of these classes have already been presented: count, time, record, state, on-off, and specials such as END. The remaining two are a stimulus control class including oscillator and attenuator commands and a SYNC class which is used to transmit information between state sets. These are summarized in Table 1. Related to each of these instruction groups is a subgroup of arguments



3-Bit patterns	Octal equivalent
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Example:

101 011 100 010	Binary
5 3 4 2	Octal

FIG. 1. Bit pattern equivalents of the octal digits and the positions of the digits in a 12-bit word.

which serve to modify and limit the effects of a given instruction. Each of the major instruction groups will be discussed below.

TABLE 1

MAJOR INSTRUCTION GROUPS

Octal Equiv.	Instruction class
0000	SPECIALS—see text
1000	STATE
2000	SYNC
3000	COUNT
4000	TIME
5000	ON-OFF
6000	RECORD
7000	OSCILLATOR–ATTENUATOR

The STATE (1000) group serves both to identify and to specify where control is transferred when a given state is terminated. As a state identifier, a 1000-group instruction must be the first instruction in a given state. The form of such an instruction is STATE N, which is translated by the compiler program as 1000 + N, where

N is a state number. To specify where control will be transferred, a State instruction follows the number word in a COUNT or TIME instruction sequence (see below), or it may immediately precede an END statement to specify an unconditional transfer to a new state. A PROB(ability) instruction, which is one of the special instruction class words (g.v.), also requires a state word as the next program statement. The STATE instruction group contains two arguments: SET (= 400) and CLOSE (= 200). These are used to specify the beginning and end of a state set. For example, the statement STATE SET 1 (1000 + 400 + 1 = 1401) must precede any states in that set, and the instruction CLOSE STATE SET 1 = 1601 must follow the states in set 1.

The COUNT (3000) instruction group is used to specify transitions which occur after a specified number of events. These events are specified as arguments in the COUNT word and can be one of three possible responses (R1 = 200, R2 = 100, R3 = 40), or one of 12 SYNC's which originated in some other state set. When a COUNT word occurs in a program, the operating program automatically considers not only the COUNT word, but also the words in the next two sequential memory locations following the COUNT word. The location immediately following the COUNT contains a number which specifies how many times the specified event must occur in order for a transition to take place, and the second location following the COUNT word specifies to which state the transition will transfer control. For example, the statement COUNT R2; 1; STATE 10 would be assembled into three successive locations as

3100 (3000 = COUNT + 100 = R2)
0001 (Number of COUNTs)
1012 (1000 = STATE + 12 = octal equivalent of 10 decimal).

Note that numbers specified in the program are interpreted as being decimal and are automatically converted to their octal equivalents. If SYNC's are used as the arguments, the SYNC definition (2000 + N, where N is the SYNC #) is OR'ed with the COUNT definition (3000), the result of which is 3000 + N. The difference between a COUNT SYNC N and a COUNT R1 (or R2 or R3) is that the COUNT SYNC word does not have any bits set in the 200, 100, or 40 positions. One additional argument is possible with the count words: VAR(iable) = 400. This specifies that the count contained in the second word of the sequence is to be taken as the mean value of a uniform random distribution of count values. Each time the statement containing a COUNT VAR instruction is entered, a new random count value is determined. If the count value is N, this random count will be somewhere between 1 and 2N. The VAR argument is especially useful in increasing the degree of uncertainty in a given procedure.

The TIME (4000) instruction group is very similar to the COUNT group except that it is used to specify transitions which occur with the passage of time. Three time

units may be specified as arguments: MSEC[1] = 200, SEC = 100, and MIN = 40. In addition, the VAR (400) argument also applies to the TIME instruction. TIME, like COUNT, is a three-word sequence with the first being TIME + arguments, the second the number of the specified time units, and the third the state to which the transition will be made. An example of a TIME instruction sequence is

TIME VAR SEC; 5; STATE 3

which is compiled as

4500 (4000 = TIME + 400 = VAR + 100 = SEC)
0005 (# of SEC)
1003 (STATE 3).

Note that the semicolon is used to indicate to the compiler program to terminate the current octal word and begin a new one.

The RECORD (6000) instruction group can use the arguments R1, R2, and R3 used by the COUNT group, or may use no arguments. A RECORD instruction is always followed by a number (0–15) which specifies the counter (actually a location in memory) on which the event will be recorded. If no argument is specified, the operating program will increment the appropriate counter once when the state containing the RECORD word is entered. If an R1, R2, or R3 argument is specified, each occurrence of the specified response will cause the counter to increment. RECORD instructions are in effect only while the state containing the instruction is active.

The device control instruction group (5000) consists of two basic instructions: ON (5000) and OFF (5400). As arguments, it has a total of eight different devices which can be controlled: 3 tone switches, 3 lights, an automatic feeder, and an electrical stimulator. The last two of these are required for our work with primates, and would not be needed for human research. These arguments can be used singly or in any combination in a single ON or OFF instruction. For example, LIGHT1 TONE1 ON is a legal instruction as is LIGHT1 LIGHT2 LIGHT3 TONE1 OFF. Issuing an ON or OFF instruction for a device that is already on or off has no effect on the device. Once a light or tone is turned on, it will remain on until it is specifically turned off. The feeder, however, has a fixed cycle and control cannot be transferred out of a state containing a FEEDER ON instruction until the feeder cycle is terminated. FEEDER OFF is a meaningless instruction. The electrical stimulator remains on only during the time the state containing the SHOCK ON instruction is active.

The 7000 instruction group is also used to control external devices; namely, those concerned with the generation of auditory stimuli (OSC = 7200) and with the control of the intensity of these stimuli (ATTEN = 7400). In our system, stimuli are generated by a battery of nine different pure tone oscillators, one of which can be selected by

[1] It has been proven more useful and efficient to use 10-msec units rather than 1-msec units.

an OSC N instruction, where N is a number 1–9. The OSC instruction can also have as arguments the words UP (= 100) or DOWN (= 40). These arguments cause the operating program to switch to the next higher or lower numbered oscillator. The ATTEN instruction also uses the UP and DOWN arguments, but these are further modified by the arguments FIVE (= 0) and TEN (= 20) which indicate the size in dB of the attenuation step. In our system, 5 dB is the smallest attenuation step, but smaller steps could easily be programmed. The instruction ATTEN UP TEN would be translated by the compiler to 7520 (7400 = ATTEN + 100 = UP + 20 = TEN) and would cause the operating program to increase the attenuation in the system by 10 dB. One of the shortcomings of the present system is that there is no provision for setting the attenuator to a specific value, nor can the state program sense what the current attenuator setting is. Since the details of the psychophysical procedure are not controlled directly by the state program (see below), these two shortcomings have not presented major problems.

The SYNC instruction group (2000) serves the specialized function of enabling one state set to communicate with another state set. Without the possibility of SYNC's, each state set would operate autonomously from every other state set. The SYNC instruction is always followed by a number 0–11. The instruction SYNC 1 in any state of any state set causes the operating program to signal the occurrence of the event SYNC 1 to any COUNT SYNC 1 instruction which might be in effect at the time the instruction is issued. The operating program further records the fact that SYNC 1 is currently in effect. Should a COUNT SYNC 1 instruction be issued by any state set when SYNC 1 is in effect, the operating program will automatically increment the register concerned with the number of counts in the count instruction. If this incrementing causes a transition, control will immediately be transferred to the state specified in the COUNT instruction. The argument CANCEL (400) is used to terminate a given SYNC. Thus, in the above example, the instruction CANCEL SYNC 1 would be required to terminate SYNC 1.

The final instruction group (0000) we have given the general name of specials. These are instructions which carry out functions not readily subsumed under any of the other headings. One example of this instruction class was presented earlier: the state terminator instruction END (0001). The END instruction must be the last instruction in each state. Another instruction in the special class is STOP (0000). This instruction causes the operating system to carry out the operations necessary to terminate the experimental session, including printing out the internal counters controlled by RECORD instructions. A third group of special instructions, HIT (0002), MISS (0003), XHIT (0004), and XMISS (0005), are used by the state program to report the results of a given test trial to the operating program. The operating program uses this information to print and punch the data. HIT and MISS signify that the trial was a true test trial in which the subject did or did not respond. XHIT and XMISS signify the same things except for blank (catch) trials. The operating program also contains machine language subroutines for controlling various

psychophysical procedures, and uses the information transmitted by these instructions to determine which stimulus will be presented on the next trial. Another member of the special instruction group is the PROB (= 200) instruction. This instruction is always followed by a number 0–100 which specifies a probability value. The next sequential instruction following a PROB instruction is always a STATE N instruction. The effect of the PROB instruction is to cause control to be transferred to the specified state with the specified probability. If control is not transferred, the state containing the PROB instruction will remain active. For example, the sequence PROB 33; STATE 7 will be compiled into two successive locations as

0241 (200 = PROB + 41 = 33 in octal)
1007 (State 7)

and will cause control to be transferred 33 % of the time to state 7. This is an extremely useful instruction for psychophysical experiments, and overcomes the fact that computers generally have difficulty introducing randomness into processes they control.

The final instruction currently in the special instruction group is the OPTION instruction. This instruction is always followed by a number 1–7, and always appears as the first instruction in any program. The OPTION instruction serves to name a given program. Our system contains a set of seven push buttons which are sensed by the operating program. When one of these buttons is pressed, the operating system automatically selects the state program having the OPTION number corresponding to the number of the button. Thus, seven completely different procedures are stored in memory and are easily selected with the push of a button. When any option is selected, the first state in each of the state sets is automatically activated.

The operating program is designed so that the special instruction group can be easily supplemented with new instructions to perform tasks which may be required for particular purposes. It is only necessary to write a machine language subroutine which performs the desired task, and then change one location in the operating program which will transfer control to the new subroutine whenever the new special instruction is encountered in the state program.

Sample Program

The integration of these various instructions into a program which will control a very simple testing procedure is shown in Fig. 2. In this procedure, the subject has two responses: R1, an observing response; and R2, a reporting response. In the program, state set 1 sets up conditions which will be in effect for the whole session. These include recording every occurrence of R1 and R2 on counters 1 and 2, respectively, and timing the 60-min session length. At the end of 60 min, control goes to state 2, which stops the session.

```
          /
          /SAMPLE STATE PROGRAM
          /
0000 0401 OPTION 1                /IDENTIFIES THE PROGRAM
          /
0001 1401 STATE SET 1             /OPEN THE FIRST STATE SET
          /
0002 1001 STATE 1
0003 6201 RECORD R1 1             /ALL R1'S WILL BE RECORDED ON COUNTER 1
0004 6102 RECORD R2 2             / AND ALL R2'S ON COUNTER 2
0005 4040 TIME MIN;
0006 0074  60;
0007 1002  STATE 2                /AFTER 60 MIN GO TO STATE 2
                                  /(SESSION TIMER)
0010 0001 END                     /CLOSE THIS STATE
          /
0011 1002 STATE 2
0012 0000 STOP                    /STOP THE SESSION
0013 0001 END
          /
0014 1601 CLOSE STATE SET 1
          /
          /
0015 1402 STATE SET 2             /OPEN THE SECOND STATE SET
          /
0016 1001 STATE 1
0017 4500 TIME VAR SEC;
0020 0005  5;
0021 1002  STATE 2                /TIME A RANDOM INTERVAL BETWEEN 1 AND
                                  /10 SECONDS. THEN GO TO STATE 2
0022 0001 END
          /
0023 1002 STATE 2
0024 3200 COUNT R1;
0025 0001  1;
0026 1003  STATE 3                /1 OCCURRENCE OF R1 WILL TRANSFER
                                  /CONTROL TO STATE 3
0027 3100 COUNT R2;
0030 0001  1;
0031 1001  STATE 1                /IF 1 R2 OCCURS FIRST, CONTROL
                                  /GOES BACK TO STATE 1
0032 0001 END
          /
0033 1003 STATE 3
0034 6003 RECORD 3                /INCREMENT COUNTER 3 (# OF TRIALS)
0035 5004 TONE1 ON                /TURN ON THE TONE
0036 4100 TIME SEC;
0037 0003  3;
0040 1005  STATE 5                /TRIAL DURATION IS 3 SECONDS
0041 3100 COUNT R2;
0042 0001  1;
0043 1004  STATE 4                /IF R2 OCCURS, GO TO STATE 4
0044 0001 END
          /
0045 1004 STATE 4
0046 6004 RECORD 4                /INCREMENT COUNTER 4 (# OF HITS)
0047 5404 TONE1 OFF
0050 5200 FEEDER ON               /OPERATE THE FEEDER
0051 0002 HIT                     /REPORT A CORRECT RESPONSE TO THE TONE
0052 1001 STATE 1                 /ALWAYS GO BACK TO STATE 1
0053 0001 END
          /
0054 1005 STATE 5
0055 6005 RECORD 5                /INCREMENT COUNTER 5 (# OF MISSES)
0056 5404 TONE1 OFF
0057 0003 MISS                    /REPORT A MISSED TONE
0060 1001 STATE 1                 /GO BACK TO STATE 1
0061 0001 END
          /
0062 1602 CLOSE STATE SET 2
          /
```

FIG. 2. Sample program written in the SNAP language.

The various stages of the experiment are controlled by state set 2. The subject responds most of the time with R1. At the end of an interval which averages 5 sec and is timed by state 1, control is transferred to state 2. If one R1 occurs before any R2 occurs, control will transfer to state 3. If R2 occurs first, control goes back to state 1, and a new interval is set up. State 3 turns on the tone for 3 sec and records the occurrence of a trial on counter 3. If R2 occurs before the 3 sec has elapsed, control is transferred to state 4, and if no R2 occurs, control goes to state 5. State 4 turns the tone off, the feeder on, records a hit on counter 4, and reports that the subject has responded correctly (HIT). At the conclusion of the feeder operation, control returns to state 1 to begin another trial. State 5 turns the tone off, records a MISS on counter 5, and reports to the operating program that the subject has failed to report a tone (MISS). Control is then transferred back to state 1.

The above example is probably too simple a procedure for actual use but it serves to illustrate how the instructions in the language can be combined to describe an experimental procedure. Note that the description of the procedure in the paragraphs above closely resembles the programming statements which are required for the computer to run the experiment.

In Fig. 2, there are two columns of numbers at the left margin. The left-hand column contains, in octal, the step number in the program, and the right-hand column contains the instruction codes. The operating program contains an editing feature, with which the operator can type in a step number, and then examine and change any step in the program. Thus, it is possible to change any of the parameters of the experiment while it is running. To change the trial duration, for example, the operator enters the edit mode, and types 37 and a space. The program responds by typing 0003 followed by a space. If the operator types 5 and a space, the next trial will be 5 sec in duration instead of 3.

CONCLUSION

The preceding description has purposely omitted the details of both the operating program and the electronics required to connect the various devices to the computer. Our desire is not to keep these details to ourselves, but rather to introduce SNAP in a general way which makes it applicable to any type of computer. The power of the computer as an experimental control device has been amply demonstrated in many laboratories. The speed and accuracy with which these machines can control complex procedures has made them invaluable to us, and hopefully, the ease with which SNAP can be used to program these procedures will make them attractive to other laboratories engaged in psychoacoustic reasearch.

REFERENCES

1. SNAPPER, A. G., KNAPP, J. Z., AND KUSHNER, H. K. Mathematical descriptions of schedules of reinforcement. *In* "The Theory of Reinforcement Schedules" (W. N. Schoenfeld, Ed.), pp. 247–275. Appleton–Century–Crofts, New York, 1971.

2. MILLENSON, J. R. A programming language for on-line control of psychological experiments. *Behav. Sci.* **16**, 248–256 (1971).
3. STEBBINS, W. C. Studies of hearing and hearing loss in the monkey. *In* "Animal Psychophysics" (W. C. Stebbins, Ed.), pp. 41–66. Appleton–Century–Crofts, New York, 1970.
4. MEALY, G. H. A method for synthesizing sequential circuits. *Bell Systems Tech. J.* **34**, 1045–1079 (1955).
5. MOORE, E. F. Gedanken-experiments on sequential machines. *In* "Automata Studies." Princeton Univ. Press, Princeton, N.J., 1956.