THE SKED SOFTWARE SYSTEM

MASTER

SUPER SKED

ARTHUR G. SNAPPER

GEOFFREY B. INGLIS

IDE '78

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

SKED^{TM} SOFTWARE SYSTEM

(MANUAL 3)

REVISION D

TIME-SHARED SUPERSKED^{TM}

Arthur G. Snapper
Western Michigan University

Geoffrey Inglis
University of Rochester

SKED USERS GROUP
PSYCHOLOGY DEPARTMENT
WESTERN MICHIGAN UNIVERSITY
KALAMAZOO, MICHIGAN 49008

STATE SYSTEMS, INC.
PO Box 2215
KALAMAZOO, MICHIGAN 49001

The word "SKED" is the trademark of State Systems, Incorporated for its brand of computer interfacing and software.

The word "SUPERSKED" is the trademark of State Systems, Incorporated for its brand of software.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

ACKNOWLEDGEMENTS

This work was supported in part by Research Scientist Development Award MH-70483 from the National Institute of Mental Health, to the senior author. The work of the junior author was supported in part by grants ES-01885, ES-01247 and ES-01248 from the National Institute of Environmental Health Sciences; grant DA-00623 from the National Institute on Drug Abuse; grant MH-11752 from the National Institute of Mental Health; and in part by a contract with the U. S. Department of Energy at the University of Rochester and has been assigned report No. UR-3490-1476.

This manual is dedicated to my late father, Arthur Snapper Sr.
Who provided both moral and financial support to all of my
efforts.

TABLE OF CONTENTS

INTRODUCTION

STATE NOTATION	i
SYSTEM CONCEPT	i
DATA ACQUISITION	i
TIME SHARING	ii
FLEXIBILITY	ii
PAPER TAPE VERSION	ii

CHAPTER 1: STATE NOTATION

SECTION 1: CRF	1-2
SECTION 2: FIXED COUNT CONTINGENCIES	1-7
SECTION 3: VARIABLE CONTINGENCIES	1-10
SECTION 4: INTERVAL SCHEDULES	1-14
SECTION 5: RANDOMIZATION	1-19
SECTION 6: COMPLEX SCHEDULES	1-22
SECTION 7: DATA COLLECTION	1-28
SECTION 8: TIME, TYPE, BREAK, CALL	1-41

CHAPTER 2: STATE TABLE PRIORITIES

PRIORITY RULE 1	2-3
PRIORITY RULE 2	2-5
PRIORITY RULE 3	2-7
PRIORITY RULE 4	2-8

CHAPTER 3: TRANSLATION OF STATE DIAGRAMS TO STATE TABLES

CRF SOURCE	3-2
CRF LISTING	3-3
CRF WITH HOUSELIGHT	3-5
CRF - 100 SR'S	3-6
FIXED RATIO 10	3-7

FIXED RATIO USING ADD	3-8
FIXED CONSECUTIVE NUMBER	3-9
FIXED CONSECUTIVE NUMBER USING ADD	3-10
VARIABLE RATIO	3-11
VARIABLE RATIO USING IF	3-12
VARIABLE RATIO USING LIST	3-14
FIXED INTERVAL(RESPONSE)	3-15
DRL	3-16
FIXED INTERVAL (CLOCK)	3-17
VARIABLE INTERVAL	3-18
RANDOM RATIO	3-19
RANDOM INTERVAL (FARMER)	3-20
RANDOM INTERVAL (CATANIA)	3-21
DISCRIMINATION (RAND)	3-22
GENERALIZATION (RAND)	3-23
SIDMAN AVOIDANCE	3-24
INTERLOCK	3-25
PROGRESSIVE RATIO	3-26
CHAINED VI VI	3-27
CONCURRENT VI VI	3-28
CONCURRENT CHAIN	3-30
TIME ALLOCATION	3-32
TIME AND TYPE	3-35
QUARTER LIFE	3-36

CHAPTER 4: SYNTAX RULES FOR STATE TABLES

SECTION 1: STATE SETS	4-2
SECTION 2: STATES	4-3
SECTION 3: INPUTS	4-3
SECTION 4: OUTPUTS	4-9
SECTION 5: TRANSITIONS	4-11
SECTION 6: DATA ACQUISITION	4-11
SECTION 7: DECISION FUNCTIONS	4-14
SECTION 8: VARIABLES	4-18
SECTION 9: OPERATORS ON VARIABLES	4-20
SECTION 10: ANALOG INPUTS AND CALL 0	4-28

CHAPTER 5: THE SKED MONITOR

LOADING STATE TABLES	5-2
AUTOMATIC DATA ACQUISITION	5-5
START, GO, AND RESPONSE SIMULATION	5-6
MODIFICATION OF VARIABLES	5-7
LISTING VARIABLES	5-7
PRINTING COUNTERS	5-8
STORING COUNTERS IN FILE	5-8
ENTERING IDENTIFIERS	5-9
TIME OF DAY COMMANDS	5-10
SAFETY SWITCH	5-10
TERMINATING BATCH	5-10
PRINTING ACTIVE STATIONS	5-11
ABORTING STATIONS	5-11
ABORTING TTY PRINTING	5-11
BATCH CONTROL OF EXPERIMENTS	5-11
STOPPING AND RESTARTING BATCH	5-15
EXITING THE SKED MONITOR	5-15
RECOVERING BACKGROUND OS/B	5-16
SUMMARY OF MON COMMANDS	5-17

CHAPTER 6: OS/B OPERATING SYSTEM

INTRODUCTION	6-2
BUILD	6-4
OVERVIEW	6-5
SYSTEM FILES	6-6
KEYBOARD MONITOR COMMANDS	6-7
EDITORS	6-9
DIRECTORY FUNCTIONS	6-10
COPY	6-14
PIP	6-14
ABSOLUTE LOADER	6-14
SYSTEMS DEVICE	6-15

PAL8	6-16
OS/8 ANALYSIS LANGUAGES FOR SKED DATA	6-16
BASIC	6-16
FORTRAN II	6-19
FORTRAN IV	6-20
FOCAL	6-24
BATCH	6-32
CCL	6-32
COMPILING STATE TABLES	6-33

CHAPTER 7: ASSEMBLING TIME-SHARE SUPERSKED

NECESSARY HARDWARE	7-2
NECESSARY SOFTWARE	7-2
EDITING PARAM. PA	7-3
SUMMARY FOR NEW USERS	7-8

CHAPTER 8: DIALOGUE FOR CONFIGURING INTERFACE

PROCEDURES	8-2
SAMPLE BATCH JOB	8-7

CHAPTER 9: REFERENCES

APPENDIX 1: SYNTAX SUMMARY

APPENDIX 2: SKED MONITOR SUMMARY

STATE NOTATION - THE LANGUAGE OF PROCESS CONTROL

State Notation was developed to aid the design of sequential switching circuits (Moore, 1955; Mealy, 1956). It is used to describe sequential processes in which outputs are completely determined by (1) the preceding set of inputs and (2) the current input. For procedures of this sort, State Notation can be shown to provide the rules for process control (including the procedures of Experimental Psychology) involving the minimal amount of control logic while completely eliminating indeterminate states (McClusky, 1963), i.e. "race" conditions.

A modified version of State Notation was developed for both hardware and software design of laboratory procedures involving control sequences that vary as a function of the behavior of the experimental subjects. The major advantages of State Notation as a process control language include: (A) generality across different scientific and industrial process control problems, (B) simplicity of the State Notation makes it easy to learn, (C) State Notation is powerful enough to describe the most complex sequential processes, and (D) since State Notation was designed for process control it is very efficient in terms of memory and CPU utilization, permitting as many as 12 concurrent procedures to be controlled by a PDP8.

SYSTEM CONCEPT

The SKED program originally developed in 1966 for a 4K PDP8 paper tape system, has been refined and integrated into the OS/8 and time-share systems developed by DEC, to take advantage of the development of mass-storage devices and the sophistication of the operating system. The current system consists of a two-pass compiler with an optional third pass listing, a set of programs for identifying and merging data acquisition files and a sophisticated system monitor that can be used to start, stop, load and modify as many as 12 independent process control and data acquisition procedures. CCL has been modified to accept standard DEC Execute, Compile and Load commands with special switch options to control the compiling of procedures designed in state notation.

DATA ACQUISITION

Three different methods are available to acquire data from as many as 12 independent stations. Each station can acquire every event time of interest and can record these

Introduction

sequentially on mass-storage devices for later processing using standard Fortran, Basic or Focal programs. Analog information may be digitized and recorded along with codes indicating conditions under which the data was acquired. Time of day and the current status of variables may also be obtained and recorded automatically.

TIME SHARING AND SKED

The system has been developed to operate under RTS-8 V2B to permit the use of the computer for OS/8 while the 12 process control stations are collecting data and being controlled by their independent programs. A minimum of 16K of memory is required for the system, but to run 12 stations and to permit the use of BATCH in the OS/8 background, 32K of memory is desirable. RTS-8 has been improved to include a method for rewriting the system area in the background and restarting OS/8 without interfering with the experimental control system.

FLEXIBILITY AND GENERALITY

The system includes a parameter file and dialogue that permits the user to configure his system. Four different clocks, the UDC controller, EAE units, line-printers and other specialized devices may be included in the system. The available memory may be partitioned between the process control section and the OS/8 background. Finally, the number of digital input and output interface lines may be assigned to 12 stations in a flexible manner, by means of an interactive dialogue. The system can be configured on any 8K OS/8 machine or in the background.

PAPER-TAPE VERSION

A paper-tape version of the system can run in 8K of memory on any PDP8 family machine, except for the PDP8-S. The only difference between the paper-tape and the time-shared versions is in the acquisition of data files. However, OS/8 may be used to create and compile procedures for the paper tape system on another machine. Alternatively, an 8K paper-tape 2 pass compiler can be used to produce programs on the paper-tape machine.

CHAPTER 1

STATE NOTATION

The following description of state notation is an introductory unit designed to teach the most common functions of the notation, using the most frequently referenced reinforcement schedules. The unit is divided into eight sections, dealing with similar types of schedules. Following each section, there will be a list of new concepts introduced in that section, including a short description of each. More formal definitions of each expression or function will be included in Chapter 3.

SECTION 1: CRF

The first, and most basic procedure of operant psychology is regular or continuous reinforcement (CRF). This procedure consists of two states, only one of which can be in effect at any moment. In the first state, reinforcement is not present but is a potential condition. State two is the reinforcement state, usually of brief duration, during which reinforcement is presented to the subject by means of a reinforcement mechanism.

FIGURE 1
CRF

Notice that CRF is completely described by two mutually exclusive states. Either State 1 is present (represented by the leftmost circle identified by the numeral 1), or State 2 is present with the onset of reinforcement (labelled as ON SR). In this procedure, the different states are associated with different stimuli. We shall see that in more complex procedures, the correlation between states and stimuli is not always perfect.

Inputs in state diagrams are written above the arrow leading from one state to the next. In Figure 1, R1 (a response of type 1) causes the transition from State 1 to State 2. The input can also produce one or more outputs which are listed following a

colon after the input. In Figure 1, R1 produces the reinforcement indicated by R1:ON SR, if the response occurs when State 1 is present. Four seconds after State 2 is entered, the reinforcement is terminated as is shown by 4":OFF SR. In this case, the end of the 4 second period is the input and OFF SR is the output.

States may represent not only stimulus conditions but also contingencies or a combination of stimuli and contingencies. In CRF, responses (identified by the symbol R1) in State 1 produce reinforcement. When State 2 is present, responses have no effect and therefore there is no response transition indicated in State 2. Transitions are symbolized by arrows leading from one state to the next. They are considered to be instantaneous as are the response or time inputs that cause transition.

Transitions from one state to the next can be initiated by elapsed time periods as is shown by the 4 second (symbolized 4") reinforcement duration of State 2 in CRF. The 4" timer is started on entry into State 2. When the 4" period has elapsed, transition back to State 1 occurs.

In the laboratory (and alas in real life) reinforcement is not always present or available. For the purposes of an experiment, it is useful to define some discrete event which starts the session, and thus presents opportunity to earn reinforcement. This situation is diagrammed in Figure 2, representing CRF as conducted in the experimental laboratory.

FIGURE 2
CRF With House Light

The state diagram of Figure 2 consists of 3 states. State 1 represents the conditions present before the experiment begins. When the start transition occurs (usually produced by an event generated by the experimenter), a stimulus (in this case the houselight of the experimental chamber) is turned on. In the

diagram of Figure 2, the houselight is not affected by responses or reinforcement and therefore is not turned on or off by the response or 4" transitions. Since no :OFF HOUSELIGHT is included in this diagram, once the experiment begins it continues forever!

Unfortunately, both in the laboratory and in real life, reinforcement opportunities have a limited duration given that they occur at all. Figure 3, then, is a complete and rigorous description of a typical laboratory CRF experiment in which only 100 reinforcements are available every session.

FIGURE 3
CRF - 100 SR'S

In Figure 3, a variable (represented by the letter A) is set to zero when the experiment starts. After each reinforcement, A is incremented by 1 as is shown by the statement SET A=A+1 on the transition line from State 3 to State 2.

The SET function, introduced in Figure 3 as an output is borrowed from computer languages, for example FORTRAN and BASIC. It is an assignment function that causes the variable to the left of the equal sign to be replaced by the value or expression to the right of the equal sign. SET A=A+1 is equivalent to saying "replace the current value of A by the old value of A plus 1", or in other words, add 1 to A. Then a decision is made (as part of the 4" transition) concerning the next state. At the end of each of the first 99 reinforcements, the transition continues to State 2, permitting more reinforcements. This occurs because A is not yet equal to 100. Therefore the "ELSE" transition to State 2 will occur. At the end of the 100th reinforcement, transition to State 1 occurs and the houselight is turned off, signalling the end of reinforcement availability.

Multiplication, division, and subtraction operations may also be expressed in the SET function. For instance, SET B=C*D

multiplies the value of two variables and deposits the result in the variable B. For calculation of rate measures, frequency is divided by time, as in $SET\ R=F/T$. Subtraction is illustrated by $SET\ A=B-1$.

Decision functions (including the IF statement of Figure 3) are graphically represented by a diamond with the decision criterion written inside the diamond. The decision function is initiated by an input, and so it is a type of output function. The decision function differs from other output functions in that it has two possible output paths. Of course, only one of these paths will be taken. The actual path taken following a decision function will depend in any instance upon the validity of the condition specified in the decision function. For example, in Figure 3 if $A=100$, then the light is turned off and State 1 is entered. If A is not currently 100 but some other number (0 to 99 in the example of Figure 3) then the transition to State 2 occurs.

In state notation, the comment "THEN" can be drawn on the transition arrow indicating the path to be taken if the decision criterion is currently valid. The comment "ELSE" indicates the transition that occurs if the conditions are not valid. All decision functions in state notation are binary. That is, either the condition is valid or it is not valid when the decision function is tested. The decision is assumed to be instantaneous and the decision function should not be confused with a state, since it has no duration.

Figure 3 uses all of the major functions of state notation. Complex reinforcement schedules consist of complex sets of contingencies. However, each of these complex procedures may be reduced to a series of states, describing the momentary contingencies and stimuli present when the state is in effect. Note that the state diagram is only a description of procedures, but not of the actual outcome of an experiment. For example, in Figure 3 if the subject never responds, reinforcement is never presented. Furthermore, if less than 100 reinforcements are earned by the subject, then the houselight will continue to shine (Consumers Power willing).

SECTION 1 CONCEPTS

=====

STATE: A portion of a sequential procedure. A state consists of the contingencies present at the time the state is in effect including the stimuli and the designation of the possible state transitions that will result when an input occurs.

STATE SET: The collection of states that make up a procedure including possible state transitions.

INPUT: A response or the passage of time after state entry. (Also, "Z pulses" see Section 4)

OUTPUT: Any stimulus change or mathematical operation which may occur when an input requirement is completed.

TRANSITION: The instantaneous change between two states when an input occurs. The previously active state is immediately deactivated and any outputs are immediately implemented.

DECISION FUNCTION: Specifies what occurs if an algebraic statement is valid at the time of state transition. May also specify what happens if the statement is not valid, and can include outputs which occur regardless of validity. See "IF", "THEN", "ELSE", "ALWAYS", "WITH", in Chapter 3 on syntax.

VARIABLE: A quantity whose value changes as a function of algebraic relationships between other quantities. See "SET" in Section 3.

SECTION 2: FIXED COUNT CONTINGENCIES

A procedure that is only slightly more complex than CRF is fixed ratio (FR). In fixed ratio, a prespecified number of responses (greater than one) is required for reinforcement. Figure 4 is a state diagram of fixed ratio 10.

FIGURE 4
Fixed Ratio 10

For simplicity, the event terminating the session is not included in this drawing. Notice that the only difference between Figure 4 and Figure 2 is the 10 preceding the transition from State 2 to State 3. The numerical prefix of a response input signifies that the 10th response to occur after entry to State 2 will produce reinforcement and go to State 3. The first 9 responses are counted but do not produce transition.

Figure 5 illustrates the fact that the FR procedure may be diagrammed using variables (e.g. B) and the IF statement.

FIGURE 5
Fixed Ratio Using Variable

In this example, each response increases (i.e. increments) B by 1 and then the computer checks (in the decision function) to see if B is 10. If B is not equal to 10, the transition arrow returns to State 2. If B is 10, then B is set back to 0 for the next trial and reinforcement is presented in State 3. This method of notating FR is cumbersome compared to the method of Figure 4, but it serves to illustrate the logic involved in the numerical counting of response inputs.

Figure 6 illustrates the fixed-consecutive-number procedure (FCN) first used by Mechner (1958).

FIGURE 6
Fixed Consecutive Number

In this procedure, 10 responses of type 1 must be emitted consecutively before a response of type 2 (R2) is reinforced. Responses of type 2 reset the accumulated count of R1's, and a new set of 10 consecutive R1's must again be emitted without an intervening R2 for transition to State 3. The recycling transition of R2 in State 2 resets the accumulated count of R1's. (Recycling refers to transitions originating and terminating in the same state). The rule is that the counter of response inputs is reset to zero on state entry. An equivalent explicit diagram using variables is shown in Figure 7.

FIGURE 7
Fixed Consecutive Number

Here the reset of the accumulated counts of R1 in State 2 is shown explicitly by the expression SET D=0 caused by the reset transition of R2. Another rule illustrated by the FCN procedure shown in both Figure 6 and Figure 7 is that multiple transitions can occur in one state. In State 2, R1 has one effect while R2 has a second effect. In order to avoid conflict among the various possible transitions, there is a basic rule that states that only one input can occur at any one time. Whichever input occurs first will cause its associated transition first. Of course, in Figures 6 and 7, only the 10th R1 in State 2 will cause a transition to State 3.

SECTION 2 CONCEPTS

RESPONSE COUNTING: The completion of a required number of responses is considered to be the input which can cause transition.

RESET INPUT COUNTER: When a state is entered, all input counter requirements are reset to a specified number.

INPUT: Since inputs are instantaneous, only one input is considered to occur at a time.

SECTION 3: VARIABLE CONTINGENCIES

Variable ratio or VR (Ferster and Skinner, 1957) is a set of schedules in which the ratio requirement varies after each reinforcement according to some rule. For example, a simple variable ratio with two different ratio values that alternate throughout the session is diagrammed in Figure 8.

FIGURE 8
Variable Ratio

For each ratio in the sequence, it is necessary to diagram two successive states, i.e. the state containing the ratio value and the reinforcement state. To simplify the diagramming of variable ratio schedules, it is possible to list the sequential values of the variable ratios and to then set a second variable to a value from the list. Figure 9 shows a VR consisting of an arithmetic series from 1 to 10 with a mean of 5.5 responses.

LIST A=3, 7, 9, 1, 10, 5, 6, 2, 8, 4

FIGURE 9
Variable Ratio Using LIST

Figure 9 introduces four new concepts: the list A; an indexed variable, A(K); SX; and the comment "ALWAYS". The list statement LIST A=3,7... is a shorthand expression equivalent to setting the variable A(0)=3, A(1)=7, etc. It assumes the assignment of 10 variables, each identified by the letter A and a number from 0 to 9. Each element of the A array is assumed to be set to a value from the list in sequential order.

The subscripted variable A(K) can then be used to select values from the list. The subscript K is another variable. In the present example, it can have a value from 0 to 9. When K is 0, then A(K) is A(0), so that the expression SET V=A(K) means SET V=3, the value of A(0). When K is 1, SET V=A(K) is equivalent to SET V=A(1), resulting in V becoming 7, the value of A(1). By controlling the value of K, we then can control which value from the list A(0) to A(9) we will assign to the variable V.

SX in Figure 9 is a null transition. Since each decision function has two possible exit transitions, one which will occur if the decision criterion is valid and one which will occur if it is not valid, we need a method for indicating no transition. In the example of Figure 9, if V is not zero, then we merely wish to wait for another R1. We do not need to change states or to cause a transition. This condition can be diagrammed by a transition to SX.

In Figure 9, we established a list of sequential ratios in which A(0)=3, A(1)=7, etc. by a simple LIST statement. On starting the session, the variable V is set to A(K) where K=0. This means we set V to A(0) or the ratio 3 in this example. Each R1 in State 2 then decreases (decrements) V by 1. The first R1 sets V to 2, the second to 1 and the 3rd to 0. For the first 2 responses, V is not 0, and therefore the transition is to the null state SX.

Following the 3rd response, V does equal 0, then the value of K is incremented and then checked. If K is 10, it is then set to 0. If K is not 10 or if it has just been reset to 0, then V is set to A(K), the next item of the list. Following completion of the first ratio, K is incremented to 1, and V is set to A(1) or the value 7 for the next ratio. By checking the value K after each ratio, the index of the list can be set back to the first item after the last ratio A(9) or 4, has been used.

"ALWAYS" is used as a comment in Figure 9 to simplify the diagram. We only want to set K to 0 if K is currently at 10. However, we wish to set V=A(K) and turn on the SR whether or not K=10. "ALWAYS" indicates the common path that follows the second decision function, after the decision has been made whether to

set $K=0$.

To further simplify the notation of variable schedules, two new conventions have been established. The first of these is the equivalence between panel A and B of Figure 10.

FIGURE 10

Preceding $R1$ by a variable is equivalent to decrementing the variable and then checking the variable for 0. If the variable is 0, then the transition proceeds. Otherwise, the same state remains in effect (the meaning of the null transition to SX).

The second convention is the use of a new output function, $LIST$. Thus, the equivalence of the statements in Figure 11 is defined.

FIGURE 11

The LIST function is equivalent to the SET and IF functions of Figure 11.

Using these two conventions, variable ratio reduces to the diagram of Figure 12.

LIST A=3, 7, 9, 1, 10, 5, 6, 2, 8, 4

FIGURE 12
Variable Ratio

Of course, any variable letter could have been used in place of V, A, or K. The sequence of numbers in the list A then specifies the particular type of variable ratio to be used.

SECTION 3 CONCEPTS

LIST: The list is a sequence of input requirements used in variable ratio or interval schedules. When the last element of the list is used, the list starts again from the beginning.

SX: SX is used to terminate "null" transitions that do not terminate states or reset input requirements. Outputs may be generated by null transitions, however.

SECTION 4: INTERVAL SCHEDULES

Interval schedules may be programmed in a manner analogous to ratio schedules. For example, consider Figure 13 which contains the diagram of a 30 second fixed interval (FI 30") "programmed by the response" (Ferster and Skinner, 1957).

FIGURE 13
Fixed Interval (Response)

FI 30 second requires the subject to wait 30 seconds after a reinforcement before another reinforcement is available. Responses occurring in State 2 before the 30 second period has elapsed have no effect. A variant of the FI schedule called DRL or Differential Reinforcement Low Rate (Ferster and Skinner, 1957) is diagrammed in Figure 14.

FIGURE 14
DRL

In DRL, responses that occur in State 2 before the 30 seconds have elapsed reset the 30 second timer, as is shown by the recycling R1 transition in State 2. The reset transition requires the subject to wait at least 30 seconds after reinforcement or after an unreinforced response before

reinforcement can be obtained.

In some procedures, it is useful to identify separate processes and to diagram these as parallel state sets. Since parallel state sets are assumed to operate independently of one another, a convention must be added at this point: the Z pulse. This is an instantaneous logical condition that synchronizes parallel state sets, by being generated as an output in one state set and serving as an input in another state set. A good example of this is contained in Figure 15, which illustrates a second variant of the fixed interval schedule, FI "by the clock" (Schoenfeld, Cumming and Hearst, 1956).

FIGURE 15
Fixed Interval (Clock)

Figure 15 shows the essential nature of FI "by the clock" in which the 30 second interval begins at the start of the session. Thirty seconds later and every 30 seconds thereafter, a Z1 pulse, or internal synchronizing pulse, is emitted.

If State Set 1 (S. S. 1) is in State 2 when the Z1 occurs, then State 3 is entered, making reinforcement available for the next response. This schedule differs from FI "by the response" in that the duration of State 2 of S. S. 1 depends upon the behavior of the subject as well as the length of the fixed interval. If the subject waits for an appreciable amount of time after entry to State 3 to make the response, then the duration of State 2, following the reinforcement, could be much shorter than the time between Z pulses.

Variable intervals are easily diagrammed using a notation similar to that used for VR. In the arithmetic VI of Figure 16,

the list output function is used to establish the value of V.

LIST A=10", 5", 20", 15"

FIGURE 16
Variable Interval

V T then is used to indicate the timing out of V by a basic clock pulse. Every T seconds, V is decremented until it reaches 0. Thus V T works in a way analagous to the V R1 of Figure 11.

Another method of programming variable ratio or interval schedules involves selection of each ratio or interval on a probabilistic rather than sequential basis. For example, random ratio (RR) (Brandauer, 1957) assigns an equal probability of reinforcement to each response.

Figure 17 uses a new decision function, incorporating a probability decision.

FIGURE 17
Random Ratio

WITH P=.02 means that the transition line leading to State 3 will occur randomly with a probability of 1/50 or 2%. Otherwise

with probability of 98%, the ELSE transition to SX will occur. All decision functions have one input transition and two output transitions. Figure 17 illustrates an RR schedule with a ratio of 50 responses per reinforcement. Each response in State 2 has one chance in 50 of being reinforced and producing State 3.

Random Interval (RI) schedules have also been investigated (Farmer, 1963; Catania and Reynolds, 1968). The method used by Farmer is illustrated in Figure 18.

FIGURE 18
Random Interval-Type 1

In this schedule, once every 5 seconds there is an opportunity for reinforcement with probability of 2%. The 5 second interval is timed by the clock, as is shown in State Set 2. If a response in State 3 is not reinforced, State 2 is reentered. Thus, if a response is emitted at least once every 5 seconds then, on the average, reinforcement will be delivered every 250 seconds by the formula T/P where T is 5 seconds and P is .02.

An alternative method for programming RI (Catania and Reynolds, 1968) is shown in Figure 19.

FIGURE 19
Random Interval-Type 2

In this variant, every 5 seconds there is a probability equal to 2% of entering State 3 in which the first response will be reinforced. Once again the average inter-reinforcement interval will be 250 seconds. However, there may be some behavioral differences between these two schedules. The Catania-Reynolds method may generate lower rates, since longer inter-response-times have a higher probability of reinforcement.

SECTION 4 CONCEPTS

PARALLEL STATE SETS: Subprocesses which are essentially independent, but which may interact, can be diagrammed as separate state sets operating in parallel.

Z PULSE: An internal pulse generated as an output in one state set, and used as an input by another state set, thus synchronizing them.

WITH P: A decision function based on a random probability.

SECTION 5: RANDOMIZATION

Several methods for producing variable interval and variable ratio schedules have already been discussed. Another method for randomly setting a variable to a quantity taken from a list has been developed for the computer system. The important feature of this function is that it selects from the list without replacement. That is, each item must be selected from the list, before it can be selected again. After each item has been selected once, the function is reset and selection can begin again. An example of the use of the RAND function is illustrated in Fig. 20.

LIST A=10", 20", 30", 40", 50", 60", 70"

FIGURE 20
Discrimination (RAND)

The procedure depicts a discrimination in which an SD is presented after a variable inter-trial-interval with a mean of 40 seconds. On START and after reinforcement, the inter-trial-interval begins in State 2 of State Set 1. The length of the inter-trial-interval is controlled by the value of the variable timer VT. On entry into State 2 of State Set 1, a Z1 is produced. The Z1 then randomly selects the next value of V from the list A by means of the RAND V=A of State 2 in State Set 2.

The RAND function will randomly select one of the seven values from the list A and it will then set V to this value. In this way, successive sets of 7 trials will each have inter-trial-intervals ranging from 10 to 70 seconds, in random order.

The RAND function is particularly valuable in selecting stimuli in random order for generalization tests. Figure 21 illustrates a procedure in which one of 6 stimuli are presented in random order.

LIST A=1, 1, 1, 1, 1, 2, 3, 4, 5, 6

LIST B=10", 20", 30", 40", 50", 60", 70"

FIGURE 21
Generalization (RAND)

If stimulus 1 is presented, then the first R1 will be reinforced. If one of the other stimuli (2 through 6) are presented, responses are not reinforced, but they start a new trial. State Set 2 randomly selects inter-trial-intervals from list B, and the next stimulus from list A. Since the stimulus value 1 occurs 5 times in list A, the number of items in the list is 10. Thus, each stimulus has a probability of 1/10 of being selected in each set of 10 trials, and stimulus 1 will occur on 5 of those trials. The order of presentation of the stimuli is randomly selected and will differ across each set of 10 trials.

The computer code for the RAND function is quite complex, although the function has been optimized for speedy operation. However, the user of the RAND function must be aware of one of its properties which may not be obvious from the format of this function.

Each RAND function has an internal list in which it keeps track of each element in the list. When an element is selected, then the internal list location corresponding to this location is

incremented. Then, when all elements have been used, the internal list is reset to zero. This feature means that only one RAND call may be used to select variables in the sampling without replacement. It is legal to have identical RAND $V=A$ calls in different places in the state table. However, each call has its own internal list which is totally independent of the other. In Fig. 20, both START in State Set 1 and the transition from State 2 to 3 need to call RAND to set V from list A. Two separate RAND calls (one after START and one after the VT transition) would cause the element selected on START to be repeated in the first set of stimuli selected by the VT transition. This problem is avoided by the use of Z pulses and a separate state set for the single RAND of Fig. 20.

SECTION 5 CONCEPTS

RAND: The RAND function selects elements at random from a list of elements without replacement until each element of the list has been selected.

SECTION 6: OTHER COMPLEX SCHEDULES

The introduction of parallel state sets simplifies diagramming the contingencies controlling session length. For example, consider Sidman Avoidance (Sidman, 1953) in which shocks are presented every 2 seconds unless a response occurs. Each response produces a 10 second delay before the next shock. Assume the session is to be terminated following either the delivery of 500 shocks, or the emission of 800 responses, or after one hour following the beginning of the session. Figure 22 illustrates the procedure and uses a parallel state set to determine the length of the session.

FIGURE 22
Sidman Avoidance

In State Set 1, the program alternates between State 1 with no shock and State 2 with a brief inescapable shock every 2 seconds. If a response occurs while in State 2, State 4 is entered and maintained by further responses occurring before 10 seconds elapse. Note that since R1's have no effect in State 3, the shock is avoidable but inescapable once it is initiated. After 10 seconds in State 4 without a response, shock is delivered. Every R1 in State Set 1 (except any that occur during shock) produces a Z1 and every shock offset produces a Z2. In State Set 2, the 800th Z1, or the 500th Z2, or 60 minutes after the start of the session will all stop the session. The plus sign between the inputs in State 2 of State Set 2 indicates the "logical OR" statement, and does not mean the "logical AND".

More complex reinforcement schedules may involve modifications of contingencies according to local response rates. For example, the interlock schedule (Rider, 1977; Berryman and

Nevin, 1962) involves a schedule in which a ratio requirement decreases as time from the preceding reinforcement increases. Figure 23 illustrates an interlock in which 50 responses are required for reinforcement at the beginning of each trial.

FIGURE 23
Interlock

When a response occurs, or after every 2 seconds, the response requirement is decreased by one. At least one response is required for reinforcement as is shown by the final R1 in State 3.

Other schedules having changing requirements for reinforcement have also been investigated. Progressive ratio (Hodos, 1961) is illustrated in Figure 24.

FIGURE 24
Progressive Ratio

In this schedule, the ratio V is incremented by a fixed amount 5 after each reinforcement. In the example of Figure 24, the first ratio is 5, the second 10 etc., and the session terminates when 15 minutes elapse without a response.

The schedule needs two variables: V and I . The variable V is set to the current ratio requirement on entry to State 2. Initially it will be 5, then 10, etc. The variable is then decremented by 1 by each response through the use of the V R1 statement. The fixed increment, 5, is added to the current ratio requirement for the next trial. The variable I "remembers" the last ratio value initially contained in V . Thus, after the first reinforcement, V is set to 10 ($5+I$ is $5+5$), and I is then set to 10. After the second reinforcement, V is set to 15 ($5+I$ is $5+10$) and then I is set to 15, etc. Remember that V R1 destroys the initial value of V , requiring that we remember it as a second variable.

Chained schedules (Ferster and Skinner, 1957) are easy to diagram. Figure 25 is an illustration of chained VI VI.

LIST A=6", 12. 67", 20. 17", 28. 74", 38. 74", 50. 74", 65. 74", 80. 74",
115. 74", 175. 74"
LIST B=12", 25. 33", 40. 33", 57. 48", 77. 48", 101. 48", 131. 48",
171. 48", 231. 48", 351. 48"

FIGURE 25
Chained VI VI

The first variable interval is terminated by a response in State 3. Instead of producing primary reinforcement, the response changes stimulus conditions and initiates a second

variable interval. When a response occurs in State 5, the reinforcement is delivered and stimulus conditions revert to their initial values.

Concurrent schedules (Ferster and Skinner, 1957) involve two or more different contingencies available for different responses. Figure 26 illustrates concurrent VI VI in which two responses are each reinforced on independent VI schedules.

LIST A=6", 12. 67", 20. 17", 28. 74", 38. 74", 50. 74", 65. 74", 80. 74", 115. 74", 175. 74"

LIST B=12", 25. 33", 40. 33", 57. 48", 77. 48", 101. 48", 131. 48", 171. 48", 231. 48", 351. 48"

FIGURE 26
Concurrent VI VI

To reduce the amount of reinforcement for switching from one key to the other, a changeover delay (COD) is often programmed (Catania, 1965). One way to program this delay is to prevent reinforcement for the first response on one key following responses on the other key. In Figure 26, this is accomplished in State 3 (S.S.3), which reflects the most recent response. If an R1 occurred most recently, then State Set 3 will be in State 2. If an R2 occurred last, S.S.3 will be in State 3. Responses in State 3 of State Sets 1 and 2 check the current State of S.S.3. "IF S.S.3=2" is equivalent to saying "if the current state of State Set 3 is State 2". If this condition is valid,

transition will occur on the transition line labelled "THEN". If State Set 3 currently has another state present, then transition will occur on the line labelled "ELSE". If S.S.3 indicates the last response was different from the current response, then the COD state, State 5, will be entered for 3 seconds. This example also illustrates the fact that there is a temporal sequence to state diagrams. The response in State Set 1 can check the state of State Set 3 and find that the preceding response was different, only if State Set 3 has not yet reacted to the current response. The rule is that the response goes to S.S.1 first, then S.S.2 and so forth in sequential order. This implies that S.S.3 must follow S.S.1 for the IF statement to work. If State Set 1 and 3 were reversed, then the "IF" statement would never discover the previous response type.

A frequently used schedule is a concurrent chain, incorporating features of both concurrent and chain procedures. Figure 27 illustrates concurrent VI 1' VI 1' chain FI 10" FI 10", in which two different responses each produce a different stimulus condition on independent V I schedules.

LIST A=6", 12. 67", 20. 17", 28. 74", 38. 74", 30. 74", 65. 74", 80. 74",
115. 74", 175. 74"
LIST B=12", 25. 33", 40. 33", 57. 48", 77. 48", 101. 48", 131. 48",
171. 48", 231. 48", 351. 48"

FIGURE 27
Concurrent VI VI
Chain FI-FI

If stimulus 2 is presented, then an FI 10" is in effect until the reinforcement is delivered. In the presence of stimulus 3, a 30" FI is in effect. If either VI is completed and its stimulus condition presented, then the other V I is suspended until after reinforcement. This is accomplished by producing a Z1 when the stimulus condition of the second component is presented. The Z1 then suspends action on the other schedule while retaining the status of its interval timer, until a Z2 is generated after reinforcement.

SECTION 6 CONCEPTS

LOGICAL OR: Inputs can be "OR'd" , meaning that the completion of either input requirement will have the same effect, producing the same transition and the same outputs. The OR symbol is +.

SECTION 7: DATA COLLECTION

Data can be collected in several different ways by SUPERSKED. The different types of data collection are not mutually exclusive and the procedures include:

1. Simple counts of events or time periods in different states.
2. Calculated data, sorted and treated by means of state table instructions.
3. Sequential data consisting of interevent times of all events of interest, coded according to the type of event recorded. Data of this sort usually requires extensive data analysis after it has been collected by data analysis programs written in FORTRAN, FOCAL or BASIC.
4. Acquisition and online analysis of analog or other sorts of data, that may be acquired by means of a special function named CALL O. Data acquired in this fashion may be manipulated in any one of the three ways mentioned above.

Data acquired by methods 1 and 2 can be examined in 3 different ways:

- A. The data can be printed any time during the session.
- B. The data may be printed at the end of the session and saved as a named output file on a mass-storage device by means of keyboard commands at the terminal.
- C. The data may be saved during or at the end of the session under state table control for later analysis.

Sequential inter-event times quickly accumulate, and are usually almost impossible to analyse without the use of data analysis programs. For this reason, data of this sort are automatically stored on the mass storage device under state table control. Methods for printing and storing data are described in the chapter concerning monitor commands (Chapter 5). The current section deals with data acquisition.

A. RECORDING OF COUNTED DATA OR ELAPSED TIME

This subsection describes the method of collecting simple counts of data or elapsed times for printing during the session and/or for later analysis. This procedure is the easiest method to use and often provides the most meaningful and useful information concerning performance of the subject. The method is based on the use of the variable C as a special type of variable that can be printed by simple commands on the user's terminal. Although all variables except C may be printed by use of the X command, the P command for the C variables permits examining individual counters without the necessity of printing all of them. Furthermore, it is possible to store the current contents of the C array of variables as a named data file on mass-storage devices, through the use of a few simple commands at the terminal.

In all other aspects the C array is equivalent to any other variable. Each counter in the C array is double precision (i. e. can range between 0 and 16,777,215), and each counter can be incremented, decremented, or operated upon by the same mathematical operators available for changing the current value of other variables.

An example of data acquisition of the simplest sort is shown in Figure 28.

LIST A=10"
LIST B=5"

FIGURE 28
Time Allocation

Figure 28 represents a "time allocation" procedure in which the subject can choose between two shock distributions. When the green light is on in the second state of S.S.1, then if the variable timer VT of S.S.2 times out, a shock is delivered. If the red light is on, produced by an R2, then shocks can be presented by State Set 3. A changeover delay (COD) signalled by a white light, permits shocks from both distributions in this hypothetical example. The new function of this diagram is the ADD function following responses and times, shown in State Set 1. ADD 1 is a faster version of and completely equivalent to SET $C(1)=C(1)+1$. Thus, ADD 1 following each R1 in State 2 of S.S.1 will increment C1.

Similarly, ADD 6 following each .01" spent in State 2 of S.S.1 will increment C(6), keeping a record of the total time spent in that state. The following table describes the count

obtained at the end of the session in each counter.

COUNTER	MEANING
1	Total R1's in green
2	Total R2's in red
3	Total R1's in COD (State 5)
4	Total R2's in COD (State 3)
5	Session duration in .01" steps
6	Total time in green (.01")
7	Total time in red (.01")
8	Total shocks in green (or white)
9	Total shocks in red (or white)

Thus, this simple-minded recording scheme permits the experimenter to present time and response allocation data for the session as a whole. Even if the experimenter were also collecting inter-event-times, he or she might want simpler summary data of this sort to describe daily performance. The essential feature of this figure is that data is merely being collected and summed over the entire session. On-line data analysis, here, merely consists of collecting the number of responses, shocks, and times emitted or elapsed in different stimulus conditions. There is no attempt to look at the fine-grain structure of the behavior or to obtain data that could permit an analysis of the sequential structure of the responding.

SUBSECTION A CONCEPTS

ADD 1: ADD 1s equivalent to SET $C(1)=C(1)+1$

C ARRAY: The variable C is unique in that it can be printed or recorded. Otherwise it is similar to any other variable.

B. ON-LINE DATA ANALYSIS

A more elaborate analysis of data from the experiment of Figure 26 is easily accomplished. For example, histograms of the latencies between shock and the next changeover response might provide evidence of temporal discriminations of the shock distributions in each component of the schedule.

Figure 29 shows the two additional state sets that collect this information.

FIGURE 29

State Set 5 records each latency from the delivery of shock by State Set 2 of Figure 26 to the next occurrence of the R1 response that leads to State 4 of State Set 1. (In State 4 of State Set 1, the red condition, shocks will not be delivered by State Set 2.)

In State 2 of State Set 5, a Z1 produced by the end of a shock generated in State Set 2 will set the variable Z to 10. If an R1 occurs in the next second, C(Z), which in this case is C10, will be incremented and State 2 will be re-entered. If, however, an R1 is not emitted in the first second of State 3, then if Z is less than 20, Z will be incremented to 11. If an R1 occurs in the next second, then C11 will be incremented. In this manner, the total number of latencies occurring in 1 second categories up to 10 seconds will be accumulated in C10 to C19. Latencies greater than 10 seconds will be stored in C20. The IF statement that checks for Z being less than 20 will prevent Z from being incremented beyond this value.

State Set 6 will collect the latencies between shocks generated by State Set 3 and the changeover response leading back to the green condition, in C21 to C31. This latency data may be examined during, or at the end of the session by keyboard commands of the experimenter.

SUBSECTION B CONCEPTS

LATENCY DISTRIBUTIONS: may be collected on-line by incrementing a variable (N) after the appropriate length of time has elapsed without a response. The variable (N) is then used as an index for an ADD C(N) statement that will increment the appropriate recording counter in the C array. The initial value of N must be

set at the beginning of the latency period to the first recording counter of the distribution. The final counter of the distribution is controlled by an IF statement that occurs before the variable index N is incremented. Distributions may be of any length and may have categories of any desired width.

C. STATE TABLE CONTROL OF DATA STORAGE

The preceding examples demonstrated data collection in which the data was summed over an entire session, although it can be printed during the session by means of manual intervention by the experimenter.

Alternatively, the summed data can be stored as a permanent record on the mass-storage device at the end of the session. However, it is often useful to be able to store data automatically by means of state table commands. For example, with very long sessions or overnight experiments it is convenient to have the state table store the data at the end of the session. Furthermore, it is well known that behavior changes during the course of sessions. It may be important to record data periodically, to permit assessment of systematic changes in performance over time. Some experimenters conduct sessions periodically over each 24-hour period on the same subject. For this purpose, it is useful to automatically start and stop sessions and to automatically store data under state table control.

To be able to use automatic data collection procedures, it is necessary to understand the basic principles involved in transferring data from computer memory to the mass-storage device. Mass storage devices operating under OS/8 each have an area which is reserved for a directory describing each file or program stored on the device. This directory contains a name for each file and the location of the file on the device and the amount of space used to store the file on the device, as well as information concerning file type and other necessary data. All of the system programs that create new files search the directory to find an unused portion of the device for the new file. If the search discovers that there is no more room on the device, the program will print this error condition on the user's terminal.

In Time-Shared SUPERSKED (TSSS), the computer can be used simultaneously to control and acquire data from 12 independent experiments and at the same time be used to run standard OS/8 programs. Since the OS/8 programs may create new files, and since each of 12 different experiments may need to store data on the mass-storage device, a strategy is required to reserve space on the device for the state table data files, so that OS/8 will

be directed not to use the specified areas for other files.

The procedure used for this purpose is to create an empty file of adequate length before starting each session. A set of keyboard commands creates the file, zeroes the first location of each block of the file, and enters the name of the file and its length into the directory. Then, when the experiment is started and starts sending data to the file, OS/8 can be used to create other files on the device in any locations that remain open. Mass storage devices operate quickly enough so that it is possible to use the device for several different purposes. It is only necessary to have a program that stores requests to transfer data between the device and computer memory, and to perform the requests sequentially. The Run Time System for SKED state tables is assigned the highest priority, so that the state table transfer will take place before OS/8 requests, given that the device is not in immediate operation when the request is made. When a state table requests recording, it usually will be able to obtain its desired transfer to the device in less than 100 msec., depending on the speed of the storage device and whether it is busy with another transfer request when the new request is generated. Most OS/8 mass-storage devices transfer data directly between memory and the device by means of a data-break interface. Conventionally, 256 12-bit words of information are transferred in one operation to or from the device. This means that an area of memory at least 256 locations long must be reserved for each experimental station requiring state table recordings.

However, since the transfer is not immediate and it may take some time after the transfer request before the recording actually takes place, more data may be acquired before the recording of the previous data has been accomplished. For this reason, it is necessary to allocate two storage spaces in computer memory for each experimental station, one for holding data currently being transferred to the storage device, and other for holding data acquired after the transfer request but before the transfer has been completed.

Since memory space is a limited resource, a special strategy has been developed to use the minimum of computer space for this purpose. The trick is to divide the minimum 256 locations needed for transfer into two logical buffers of 124 locations. When one of these buffers is full, a data recording request is generated, and new data is placed in the other buffer. This results in an unusual format for the data file on the mass-storage device. Each block of 256 locations contains only 124 valid recordings, with the other half being a mixture of valid new data, and old meaningless numbers collected earlier. A special OS/8 program has been developed to modify this storage format and to concatenate the file of data after it has been collected. The important feature is that every time 124 locations (or 62 successive double-precision data points) have been collected, it will take up to several seconds before another data transfer from

a station may be safely generated, to assure that the previous transfer has taken place. During this period, it is possible, however, to collect 61 counters in computer memory.

In summary, the experimenter must:

1. Create an individual file with a unique name for each experiment before starting the session. (see Chapter 5)
2. The experimenter must not attempt to record more than 62 counters on the mass storage device before enough time has elapsed for the recording to take place.
3. The experimenter must process the data file at the end of the session to combine the data blocks on the storage device.

To obtain automatic recording of data, the state table must contain an OPEN statement. The OPEN function checks to see whether the experimenter has created a storage file on the device for the data to be collected from the current station. If such a file exists, the OPEN function initializes the recording routine. If not, the terminal will print the message "OPEN FAILED". The experiment may proceed in this case but the data can not automatically be transferred to the storage device, since the experimenter has not reserved the file for this data. At the end of the session, the state table must also contain a CLOSE statement to ensure that any data remaining in the buffers is written onto the recording device. Otherwise, up to 61 data points may be lost from the end of the session, since they may still be in the computer memory and not on the storage device.

To record counters from the C array onto the storage device, it is necessary to transfer the data from the C array to the computer buffer. When 62 data points have been placed in the buffer, then the transfer to the storage device will automatically be requested by the Run Time System.

Figure 30 contains a state set that will record the counters of the procedure diagrammed in Figures 26 and 27 on the storage device.

FIGURE 30

If the experiment has reserved the storage file for this data on the storage device, the OPEN statement on START will initialize the recording routine. Five minutes later, and every 5 minutes thereafter until the end of the session, a FOR statement will cause the 31 counters of the experiment to be transferred to the data buffer. Since every 10 minutes 62 counters have been collected, a data transfer to the storage device will be requested at this point. At the end of the session an end-of-session code is written, the file is closed and the session stops.

First the FOR function must be described. The FOR statement specifies a variable, its initial value, an increment and a maximum value. All operations occurring after the FOR statement and before the NEXT statement will occur repetitively with the variable successively set at the initial value, then at the initial value plus the specified increment and so forth until the variable has reached the final value. This means that on the 5' clock tick, the FOR loop will write C1 to C31 in succession to the computer buffer, and then set C1 to C31 to zero to again accumulate data for the next 45 minutes.

The WRITE statement transfers the contents of the variable specified in the statement (in this case C(L)) to the memory buffer reserved for the station. The 16 included in the WRITE statement is a code value that is also stored in the double-precision location used for each number in the buffer. The code may be used by data analysis programs to identify special events, but if the code is specified to be greater than 15 as in the present case then this specification is equivalent to no code.

The transfer of the 31 counters of the C array to the memory buffer is accomplished by a new function, the FOR loop. The FOR L=1,1,31 statement means do the following output statements (up

to the NEXT statement) 31 times, starting with L set to 1. Each time the NEXT statement is encountered, L is incremented by 1 until it equals 31. After L=31, the FOR loop conditions are satisfied and the statement is completed until the next 5 minutes elapse. In this way, counter 1 is first transferred to the buffer (WRITE C(L), 16, where L=1) and then it is set to 0 (SET C(L)=0 where L=1). Next, C1 is transferred and zeroed, etc., until all 31 counters have been transferred and zeroed. Remember that all of this occurs on the one clock tick that ends the 5 minute period. FOR statements are dangerous in that they can take an inordinate amount of time to complete if the number of repetitions is large or the functions being repeated are time-consuming. The syntax of the FOR statement is:

FOR Variable = Initial value, Increment, Last value.

The 100th shock generates a WRITE 0,15 and goes to State 3. WRITE 0,15 causes a 0 with a code 15 to be transferred to the buffer. Data analysis programs can detect the code 15 to determine that this is the end of the session. Next, an attempt to close the file (transfer any numbers remaining in the memory buffer, followed by zeroes) is made on every clock tick. If the device is busy, (for example, transferring the preceding block of data or transferring data for OS/8 or another station) then the CLOSE command fails and transition is to SX. The next clock tick will again attempt to CLOSE and this process will continue until the CLOSE is successful in requesting a data transfer, after which the transition to STOP occurs.

SUBSECTION C CONCEPTS

OPEN initializes automatic data recording if a file has been prespecified on the device to reserve room for data. If the file has not been created the message "OPEN FAILED" will be typed.

WRITE C(L),N will transfer counter L to the memory buffer and assign it code N. Codes 0 to 15 are valid and codes greater than 15 are equivalent to no code. The Run Time System will automatically generate a transfer request when 62 counters have been transferred.

CLOSE will attempt to transfer the buffer (filled with zeros after the last valid entry). If a request is already pending but not complete for this station, CLOSE defaults to SX.

FOR Variable=Initial Value, Increment, Final Value will cause all functions between the FOR statement and a NEXT command to be conducted repetitively with the variable starting at the initial value, then with the variable set to the initial value plus the increment, etc. until the final value has been reached. All of

this occurs on one clock tick, so that timing becomes critical.

D. AUTOMATIC RECORDING OF SEQUENTIAL INTEREVENT TIMES

Often, questions concerning performance in experiments occur after data analysis has begun. It is not uncommon for the experimenter to wish that additional data had been collected after initial data analysis has been completed.

Microanalysis of sequences of responses and stimuli often has been more informative than grosser measures such as response rate (Weiss, 1970, 1973; Anger, 1956). It is possible to collect the time at which every event of interest in the session occurs, by means of a simple set of state table commands. Of course, the data is so extensive that data analysis programs are necessary to summarize and treat data acquired in this fashion. However, some general data analysis programs are available and specific programs can be written when required in FORTRAN or FOCAL.

In any case, collection of the sequences and times between events can easily be obtained with the highest precision of which SKED is capable, the .01 second tick.

For example, an alternative method of recording for the procedure diagrammed in

Figure 28 is shown in Figure 31.

FIGURE 31

In this diagram, the time at which each response of type R1 and type R2 as well as each type of shock Z1 and Z2 are recorded. Each of these events produces the statement RECORD N, where N is a code number used by the data analysis program to recognize the event producing the recording. Each RECORD statement transfers the time in .01 second clock ticks to the buffer. The time is the number of .01 second ticks that have occurred since the last RECORD event. The first RECORD event of the session is the time since the OPEN statement occurred at session start. The end-of-session code 15 is added to the end of the buffer, and the CLOSE command is given as in the preceding example.

Other events may be recorded if of interest to the experimenter and each may be identified uniquely. If more than 15 codes are required, an additional number of codes can be obtained by reserving one code value for a WRITE statement and then WRITEing the desired code before each RECORD statement. For example if 50 codes were required, we would reserve code 0 for the WRITE event and then use a RECORD 16 for each RECORD statement. To indicate that an event type 48 had occurred and to RECORD the elapsed time since the last event, we would write the following statement:

R1: WRITE 48,0; RECORD 16

This would transfer two double precision numbers to the memory buffer. The first would be the CODE 48 and the second would be the elapsed time (with a CODE 0).

SUBSECTION D CONCEPTS

RECORD N transfers the elapsed time since the OPEN statement to the memory buffer for the first recorded event of the session. Thereafter, each RECORD statement will transfer the time elapsed since the last event. The N is a code from 0 to 15 that can be used by data analysis programs to identify the event. If more codes are necessary, they can be obtained from the WRITE statement.

SECTION 8: TIME, TYPE, BREAK, and CALL

Three special functions have been added to the notation for convenience and additional power.

A. TIME: The first statement obtains the time of day and stores it into three consecutive variable locations, first as the full time of day (in hours and minutes), next, as the hours alone, and third, as the minutes alone. These values may then be used to turn on stimuli by means of IF statements or may be stored in the counter array or written in the data file by means of the WRITE statement. For example, the statement TIME A: at 10:05 will provide the full time of day in A0 (1005), the hours in A1 (10), and the minutes in A2 (5). To start a session at 1300, Figure 32 would work well.

FIGURE 32
Time and Type

B. TYPE: It is possible to type text on the terminal from state table control. For example,

R1:TYPE " !THIS IS R1!"

will type (c.r.) THIS IS R1 (c.r.) where (c.r.) stands for carriage return. The symbol ! causes the carriage return/line feed combination to occur. The typing will interrupt any OS/8 typing that is underway when the TYPE command occurs. Typing will then revert to OS/8 when the message from SKED finishes, and no characters will be lost from the OS/8 printout.

C. BREAK: This command may be used in conjunction with the FOR loop and IF statement to interrupt the FOR loop when a condition has been met.

For example, one might wish to reinforce a subject on an FI schedule if his quarter-life exceeded 50% of the interval (50 seconds). The quarter-life is that amount of time necessary for 25% of the responses in the interval to occur. Figure 33 is an example of this schedule.

FIGURE 33
Quarter Life

During the interval, responses are recorded in counters 1 to 10 representing successive 10 second portions of the 100 second interval. At the end of the interval, X is set to one-fourth the number of responses emitted in the interval which were recorded in C11. Next, the FOR K=1,1,10 statement starts summing the successive counters in variable S. When S is greater than one-fourth the total responses of the interval, then the summation of the FOR loop is halted by the BREAK command. At this point, the counters and S can be zeroed for the next trial. Then, the index of the FOR loop, K, is compared to 5. If one-fourth of the responses took longer than 50" to be emitted, then K will be bigger than 5. Otherwise, no reinforcement will be delivered and a new trial can begin.

D. CALL and CALL O. The CALL statement is used to pass arguments to subroutines which may be added by the user to accomplish special operations not included in the notation. As an example, CALL O has been written to permit the operation of external devices or to acquire data.

The format of CALL statement includes a number from 0 to 8, which may be used by the programmer for his programs. CALL 0 will transfer a 12-bit number from the low order variable and then operate an input/output command specified by the user. Next, it will store the contents of the accumulator back in the low order variable. Thus, with an IOT specifying A to D conversion, the converted value will be placed in the variable. For D to A conversion, the contents of the variable will be passed to the analog device. The format of CALL 0 is : CALL 0 [variable, ?IOT] where the ? indicates that the next number is Octal. The variable is either the source argument for the CALL or the recipient, depending upon the IOT.

CHAPTER 2

STATE TABLE PRIORITIES

State notation, as originally designed (Mealy, 1955; Moore, 1956) required that only one input can occur at a time and that all inputs, transitions, and outputs are instantaneous. These two basic rules guarantee that no conflicts or indeterminate paths ("race" problems in engineering jargon) can occur. The first valid input to occur within a state will cause its associated inputs and transitions. Thus it is possible to predict the outputs and transitions that will occur when a particular input occurs in a particular state.

In the theory of sequential machines (McClusky, 1965) only one state set is considered at one time. Its relationships to the world outside of the state set is limited to receiving inputs from the world and sending outputs to the external environment. Although all sequential procedures, including reinforcement schedules can be diagrammed by means of a single state set, Chapter 1 suggested that for clarity, simplicity, and often for reduction in the number of states, it is useful to introduce parallel state sets. The concept of parallel state sets is implicit in the theory of sequential machines, since any input to a sequential machine can be generated as an output by another sequential machine which is also described by state notation.

Snapper, Knapp and Kushner (1970) proposed explicit construction of parallel state sets that communicate by a set of internal Z pulses that can be produced as an output by one state set and which can serve as an input for another state set. The explicit consideration of parallel state sets introduces a new set of problems (dealing with priorities) that must be considered in a formal logical system, even though these problems are rarely encountered in programming reinforcement schedules. The problems arise from two rules of state notation. The first rule is that only one input may occur at one moment in time. The second rule is that outputs are instantaneous and simultaneous with inputs. Since Z pulses are both outputs that can be generated by response, time, or Z inputs, and also can be inputs, the Z pulses could logically be simultaneous with the inputs that produce them. Therefore, the basic rule of only one input at a time would be violated. This problem can be clearly illustrated in a single state set.

FIGURE 36

In Figure 36 in State 2, an R1 both produces a Z1 and causes transition to State 3. The Z1, if it occurs in State 2, should cause transition to State 4. To prevent conflict in diagrams of this sort and to retain the rule that only one input can occur at a time, we must introduce a rule that states: "Z pulses operate after the input that produces them has operated on each state set, but before an input of any other type can occur." For instance, a Z1 produced by an R1 will occur before an R2 is operated on. This is Priority Rule 1. This Rule then clarifies Figure 36. The Z1 is slightly later than and non-simultaneous with the R1, and therefore the Z1 will occur in State 3 and will cause transition to State 5. The sequence will be as follows: First, R1 occurs in State 2, producing a Z1 but causing transition to State 3; next, the Z1 will be an immediate input, causing transition from State 3 to State 5. Although this type of conflict rarely arises in state diagrams, there must be a well-defined rule that tells us what will happen in this rare case.

The same type of conflict may occur more frequently in cases involving parallel state sets. The rule previously introduced will then solve the conflict. For example, in Figure 37, both State Sets 1 and 2 will be in State 2 after a Start pulse.

FIGURE 37

When an R1 occurs, the following sequence of events will be produced: The R1 will produce Z1 and produce State 3 in State Set 1. Next, the R1 will cause transition from State 2 to State 3 in State Set 2. The Z1 will then cause transition from State 3 to State 5 in State Set 1, and in State Set 2 it will also produce a Z2 and transition to State 5. Next, the Z2 will cause transition from State 5 to State 6 in State Set 1 and will have no effect in State Set 2. Thus, the R1 will produce State 6 in State Set 1 and State 5 in State Set 2.

In general, it is never necessary and seldom useful to use the Priority Rule in programming reinforcement schedules. The use of the Priority Rule will, in general, reduce the clarity of the state diagram and make it difficult for others to read.

The use of Z pulses also may lead to infinite loops (the worst problem in a time-sharing computer system). For example in Figure 38, the start event produces a Z1 and goes to State 2.

FIGURE 38

The Z1 will then produce a Z2 and go to State 3. The Z2 will produce a Z1 and go to State 2. The cycle between States 2 and 3 will continue indefinitely. After .2" the Run-Time System will terminate the session in the station containing the infinite loop. It will also print SKED ERROR on the user's terminal.

Another priority conflict can occur with Z pulses. Consider the diagram of Figure 39, in which an R1 in State 2 produces both Z1 and Z2.

FIGURE 39

The Z pulses will follow the R1, but which Z will be effective in State 3? The rule that was created to handle conflicts of this sort is: "When a state has two or more transitions produced by separate Z pulses, and when the two or more Z pulses occur on the same input, then the transition line in the higher position in the state diagram (occurring as the first line in the state table) will occur first." This is Priority Rule 2. In Priority Rule 2, by higher position we mean the uppermost transition line in the state having the multiple transitions. In Figure 39, the Z1 transition will take place because the Z1 transition is drawn above the Z2 transition.

FIGURE 40

In Figure 40, involving parallel state sets, the R1 in State Set 1 produces Z1 and Z2. In State Set 2, the Z2 transition will occur producing State 4 because it is the upper transition line.

This Priority Rule must sometimes be invoked when different Z pulses are produced in different state sets by the same input. For example, in Figure 41, the R1 will produce Z2 in State Set 1 and Z3 in State Set 2.

FIGURE 41

The result, of course, will be to produce State 4 in State Set 3. The careless programmer can easily construct a state table of this sort that does not do what he/she wanted. Be sure to check for the occurrence of this sort of conflict when you have a state with two or more Z pulses. Although Priority Rule 2 permits some programming "tricks" to be used on purpose, it probably should be avoided, since it results in obscure diagrams.

Another priority problem arises from decision functions that check the current state of another (or the same) state set. Consider Figure 42.

FIGURE 42

In Figure 42, an R1 in State 2 of State Set 1 checks the status of State Set 2, to choose between transition to State 3 or to State 2. State Set 3 advances to State 3 on an R1. If the R1 simultaneously causes transition in the parallel state sets, the decision function of State Set 1 is indeterminate since R1 causes State Set 2 to change states while simultaneously checking the current state of State Set 2 in the decision function of State Set 1.

Priority Rule 3 defines the outcome for state diagrams involving this sort of conflict: "When an input occurs, it drives state sets sequentially (State Set 1 first, State Set 2 next, etc.). Priority Rule 3, then, specifies that R1 will cause transition to State 3 in State Set 1 when State Set 2 is in State 2, because the R1 will drive State Set 1 before State Set 2. If the State Sets of Figure 42 were reversed, as is shown in Figure 43, then the outcome would be reversed.

FIGURE 43

The R1 in Figure 43 would never cause transition to State 3 of State Set2, since State Set 1 would move from State 2 to State 3, before the decision function of State Set 2 would be entered.

A fourth Priority Rule is needed to define what will happen with "IF" decision functions that check a variable that is modified on the same transition line, in which the decision function is used. For example, Figure 44 illustrates this potential conflict.

FIGURE 44

Each R1 in State 2 SETs A to 3 and checks the current status of A. Priority Rule 4 states: "The output and decision functions caused by an input in a state occur in sequential order from left to right". In Figure 44, then, transition will always occur from State 2 to 3 when an R1 occurs in State 2, because A is always set to 3 first. Although this is a trivial and non-useful example, Priority Rule 4 defines the outcome of

any similar conflicts.

Although the Priority Rules are arbitrary, to some degree, in that other equally useable rules could have been chosen to handle conflicts arising with parallel state sets, the current rules have been used successfully in several versions of the SKED computer system, which is based on state notation. No major problems have been reported to arise from the four Priority Rules in the ten years which SKED has been available.

PRIORITY RULES

PRIORITY RULE 1: Z pulses operate after the input that produces them has operated on each state set, but before an input of any other type can occur.

PRIORITY RULE 2: When a state has two or more transitions produced by separate Z pulses, and when two or more Z pulses occur on the same input, then the transition line in the higher position in the state diagram (occurring as the first line in the state table) will occur first.

PRIORITY RULE 3: When an input occurs, it drives state sets sequentially (State Set 1 first, State Set 2 next, etc.).

PRIORITY RULE 4: The output and decision functions caused by an input in a state occur in sequential order from left to right.

CHAPTER 3

TRANSLATION OF STATE DIAGRAMS TO STATE TABLES

State diagrams must be entered on a computer terminal to produce the desired binary state table. Since most terminals resemble typewriters in format, the state diagrams of Chapter 1 must be converted to a linear form called a state table. This chapter includes state tables for all of the complete procedures described in Chapter 1. The source code or state table initially typed into the computer for Figure 1 of Chapter 1 is shown below:

```
/CHAP 1
/FIGURE 1
/REGULAR REINFORCEMENT OR CRF
/R1=RESPONSE
/ON3=REINFORCEMENT
/REINFORCEMENT DURATION=4"
```

```
S. S. 1,
S1,      R1: ON 3--->S2/THIS IS A COMMENT
S2,
      4": OFF 3--->S1
$
```

Time share SUPERSKED has a 3 pass compiler that will provide a listing of the source code of the state table along with the number of each line of text. This chapter will use the listings of the state tables since the line numbers simplify referencing the tables. Remember when typing these state tables you must not include the line numbers or the heading produced by the compiler. The listing of State Table 1 follows:

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001          ///////////////////////////////////////////////////
0002          /
0003          TABLE 1
0004          CRF
0005          /
0006          //////////////////////////////////////
0007
0008
0009          /CHAP 1
0010          /FIGURE 1
0011          /REGULAR REINFORCEMENT OR CRF
0012          /R1=RESPONSE
0013          /ON3=REINFORCEMENT
0014          /REINFORCEMENT DURATION=4"
0015
0016          S. S. 1,
0017          S1,
0018          R1: ON 3--->S2/THIS IS A COMMENT
0019          S2,
0020          4": OFF3--->S1
0021          $

```

The first point of interest is that all lines starting with a slash (/) are comments and are ignored by the compiler. The comments are included to clarify the operation of the state table for future reference. In Figure 1, lines 1 to 15 are either comments or simply carriage returns resulting in blank lines, e.g. lines 7, 8, and 15.

Line 16 containing S. S. 1, is the first significant entry of Figure 1. This notation specifies the beginning of State Set 1, and it must be included in every state table. Similarly line 17 containing S1, specifies the beginning of State 1. Both the periods and the commas of lines 16 and 17 must be included.

Line 18 begins with a Tab (causing 8 spaces) just for formatting purposes. The line then contains an R1 which signifies an input of a transition line. The R1 in this case results when a lever is pressed, and the lever must be electrically connected to input channel 1 for each station using this state table. The

colon (:) following R1 specifies the beginning of the output section of the transition. ON 3 signifies activation of output channel 3. The meaning of this activation depends upon the initial dialogue and upon the unique wiring connections of each laboratory. The space between ON and 3 is unnecessary and is only used for formatting purposes.

The dashes (-) and arrow heads (>) indicate the transition arrow. At least one dash and the arrow head are needed. Additional dashes are included for clarity. S2 following the > indicates transition will occur to State 2 when an R1 occurs.

Line 19 indicates the beginning of State 2, since State 1 has only one transition. Line 20 specifies a 4" reinforcement duration with transition back to State 1. Line 21 is a dollar sign (\$) indicating the end of the state table. It is unnecessary with the Time Sharing compiler OSCOMP, but is required by the paper-tape compiler, COMP16.

Table 2 is similar to Table 1, but includes two different outputs, ON 1 in line 19 and ON 3 and OFF 3 in lines 22 and 24. The word START signifies an R1200 initiated by the S command on the terminal.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////
0002      /
0003      TABLE 2
0004      CRF WITH HOUSE LIGHT
0005      \
0006      //////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 2
0011      /CRF
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=4"
0016
0017      S. S. 1,
0018      S1,
0019      START: ON 1--->S2/START TURNS ON HOUSE LIGHT
0020
0021      S2,
0022      R1: ON 3--->S3
0023      S3,
0024      4": OFF 3--->S2
0025      $

```

Table 3 line 24 illustrates multiple outputs on the same transition line. We recommend the practise of separating the input from the output section by a colon and then using semi-colons to separate any successive outputs on the same transition line. However, colons and semi-colons are considered to be interchangeable by the compilers.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001
0002
0003
0004
0005
0006
0007
0008
0009    /CHAP 1
0010    /FIG 3
0011    /REGULAR REINFORCEMENT OR CRF
0012    /R1=RESPONSE
0013    /ON 1=HOUSE LIGHT ON
0014    /ON 3=REINFORCEMENT ON
0015    /REINFORCEMENT DURATION=4"
0016    /SESSION TERMINATES AFTER 100 REINFORCEMENTS
0017
0018    S. S. 1,
0019    S1,
0020        START: ON 1--->S2
0021    S2,
0022        R1: ON 3--->S3
0023    S3,
0024        4": ADD A; OFF 3; IF A=100; THEN; OFF 1--->S1
0025        : ELSE----->S2
0026    $

```

Note that the statement ADD A is used in Table 3 in place of the SET A=A+1 of Figure 3. Furthermore, line 24 has a decision function IF in the output section. Everything following IF A=100; only occurs after the 100th reinforcement. Otherwise line 25, which must begin with a colon, will occur.

Table 4 is a straight-forward translation of Figure 4.

DS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////
0002      /
0003      /
0004      /
0005      /
0006      /
0007      /
0008      /
0009      /CHAP 1
0010      /FIG 4
0011      /FIXED RATIO 10
0012      /ON 1=HOUSE LIGHT ON
0013      /ON 3=REINFORCEMENT ON
0014      /R1=RESPONSE
0015      /REINFORCEMENT DURATION=4"
0016
0017      S. S. 1,
0018      S1,
0019      START: ON 1--->S2
0020      S2,
0021      1OR1: ON 3--->S3
0022      S3,
0023      4": OFF 3--->S2
0024      $

```


However, Table 5 uses the ADD function instead of SET $B=B+1$ for its speed. Also the ELSE transition of Figure 5 has been omitted. If the transition for the false condition of a decision function leads to the same state, it is usually unnecessary, since the IF condition will automatically go to SX if no default line is specified. It would be necessary if the reset property of recycling transitions were needed to reset timers or counters in State 2.

Although the use of explicit ELSE lines and the use of SET probably clarifies state tables, the computer user is confronted with limits on the speed of operation of the procedure and on the amount of computer memory available. Therefore, the practise of this chapter will be to use the most efficient state tables.

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024
0025

/CHAP 1
/FIG 5
/FIXED RATIO 10
/EQUIVALENT SCHEDULE WITH SET
/R1=RESPONSE
/ON 1=HOUSE LIGHT ON
/ON 3=REINFORCEMENT ON
/REINFORCEMENT DURATION=4"

S. S. 1,
S1,

START: ON 1----->S2

S2,

R1: ADD B; IF B=10; THEN; ON 3; SET B=0--->S3

S3,

4": OFF 3--->S2

\$

TABLE 5
FIXED RATIO USING ADD

Table 6 shows in lines 22 and 23 the notation for two (or more) inputs to a state. The second input merely occurs on the line following the first input.

DS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001
0002
0003
0004
0005
0006
0007
0008
0009    /CHAP. 1
0010    /FIG 6
0011    /FIXED-CONSECUTIVE-NUMBER
0012    /R1=RESPONSE OF TYPE 1
0013    /R2=RESPONSE OF TYPE 2
0014    /ON 1=HOUSE LIGHT ON
0015    /ON 3=REINFORCEMENT ON
0016    /REINFORCEMENT DURATION=4"
0017
0018    S. S. 1,
0019    S1,
0020          START: ON 1--->S2
0021    S2,
0022          1OR1--->S3
0023          R2--->S2
0024    S3,
0025          R2: ON 3--->S4
0026    S4,
0027          4": OFF 3--->S2
0028    $

```

TABLE 6
FIXED-CONSEC. -NO.

Tabl omits SET D=0 in line 20, that appears in Figure 7. All vars not used in a list are initially set to 0. Therefore, output was omitted to save space. Moreover, the ELSE functis omitted following line 22, and the ADD function was used. 2 of line 23 goes to SX since this is both faster and requires space than the unneeded transition to State 2 of Figure 7

OS/8 SUPER E D 3.20

JUN 15 1978

PAGE ONE

0001

0002

0003

0004

0005

0006

0007

0008

0009 /CH

0010 /FI

0011 /FIONSECUTIVE-NUMBER OR FCN

0012 /R1ONSE OF TYPE 1

0013 /R2ONSE OF TYPE 2

0014 /ONUSE LIGHT ON

0015 /ONINFORCEMENT ON

0016 /RECEMENT DURATION=4"

0017

0018 S. S

0019 S1,

0020 START: ON 1---->S2

0021 S2,

0022 R1: ADD D; IF D=10; THEN---->S3

0023 R2: SET D=0--->SX

0024 S3,

0025 R2: ON 3--->S4

0026 S4,

0027 4": OFF 3; SET D=0--->S2

0028 \$

TABLE 7

FIXED-CONSEC.-NO.

Table 8 is a straight-forward translation of Figure 8.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001
0002
0003
0004
0005
0006
0007
0008
0009      /CHAP 1
0010      /FIG 8
0011      /VARIABLE RATIO OR VR
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=4"
0016
0017      S. S. 1,
0018      S1,
0019          START: ON 1--->S2
0020      S2,
0021          6R1: ON 3--->S3
0022      S3,
0023          4": OFF 3--->S4
0024      S4,
0025          3R1: ON 3--->S5
0026      S5,
0027          4": OFF 3--->S2
0028      $

```

Table 9 includes a specified list of variables in line 17. This list declaration must precede S.S.1, of line 19

Additionally in line 23 the faster SUB V output is used instead of SET V=V-1.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      ///////////////////////////////////////////////////
0002      //
0003      //          TABLE 9
0004      //          VARIABLE RATIO
0005      //
0006      ///////////////////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 9
0011      /VARIABLE RATIO OR VR
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=4"
0016      /LIST A=VR VALUES
0017      LIST A=3, 7, 9, 1, 10, 5, 6, 2, 8, 4
0018
0019      S. S. 1,
0020      S1,
0021          START: ON 1; SET V=A(K)--->S2
0022      S2,
0023          R1: SUB V;
0024          IF V=0; THEN; ADD K;
0025          IF K=10[THEN; SET K=0]ALWAYS; SET V=A(K); ON 3-->S3
0026          ; ELSE-->SX
0027      S3,
0028          4": OFF 3--->S2
0029      $

```

The largest discrepancy between state diagrams and state tables is apparent when comparing Figure 9 with Table 9, line 24, 25 and 26. The state diagram makes it easier to associate both output transitions (i.e., the transition that occurs if the decision criterion is valid and the transition desired if the criterion is not met) with the appropriate decision function. Lines 24, 25 and 26 of Table 9 illustrate the problem that arises when there are two or more decision functions on the same transition line. With which IF statement should line 26 be associated? To handle this difficulty two different methods for specifying the valid and invalid transitions have been developed.

The first method, introduced in Table 3 is used for the decision function IF $V=0$ at the beginning of line 24. If V is not 0, then the default line shown on line 26 will occur. The second method is shown when V is 0. K will be incremented and then the second decision will occur, i.e., IF $K=10$. The left bracket, [, indicates the beginning of the outputs required if the IF conditions are true. The right bracket,], terminates the "true" transition. The rest of the transition starting with the comment ALWAYS will occur whether or not $K=10$, but only if $V=0$. Line 26 is optional, since the transition required is to SX.

Figures 10 and 11 are not complete procedures. Figure 12 is represented in Table 10. The transformation from Figure 12 to Table 10 is straight-forward, as is the translation of Figure 13 to Table 11.

DS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      ///////////////////////////////////////////////////
0002      /
0003      TABLE 10
0004      VARIABLE RATIO WITH LIST /
0005      \
0006      ///////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 12
0011      /VARIABLE RATIO OR VR
0012      /R1=RESPONSE
0013
0014      /ON 1=HOUSE LIGHT ON
0015      /ON 3=REINFORCEMENT ON
0016      /REINFORCEMENT DURATION=3"
0017      /LIST A=VR VALUES
0018      LIST A=3, 7, 9, 1, 10, 5, 6, 2, 8, 4
0019
0020      S. S. 1,
0021      S1,
0022      START: ON 1; LIST V=A(K)--->S2
0023      S2,
0024      V R1: LIST V=A(K); ON 3--->S3
0025      S3,
0026      3": OFF 3--->S2
0027      $

```

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```
0001      //////////////////////////////////////
0002      /
0003      /
0004      /          TABLE 11
0005      /          FIXED INTERVAL (RESPONSE)
0006      /          \
0007      //////////////////////////////////////
0008
0009      /CHAP 1
0010      /FIG 13
0011      /FIXED INTERVAL 30" BY THE RESPONSE
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=4"
0016
0017      S. S. 1,
0018      S1,
0019      START: ON 1--->S2
0020      S2,
0021      30"--->S3
0022      S3,
0023      R1: ON 3--->S4
0024      S4,
0025      4": OFF 3--->S2
0026      $
```


In Table 12, showing the DRL procedure of Figure 14 line 22 represents the recycling transition to State 2 that occurs following a response. Re-entering State 2 resets the 30" timer of line 21.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////
0002      /
0003      TABLE 12
0004      DRL
0005      \
0006      //////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 14
0011      /DIFFERENTIAL REINFORCEMENT OF LOW RATE OR DRL 30"
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=4"
0016
0017      S. S. 1,
0018      S1,
0019      START: ON 1--->S2
0020      S2,
0021      30"--->S3
0022      R1--->S2
0023      S3,
0024      R1: ON 3--->S4
0025      S4,
0026      4": OFF 3--->S2
0027      $

```

Table 13 representing Figure 15 shows how parallel state sets are typed. Line 27 indicates the beginning of State Set 2 (and the end of State Set 1) by S. S. 2,.

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001
0002
0003
0004
0005
0006
0007
0008
0009    /CHAP 1
0010    /FIG 15
0011    /FIXED INTERVAL 30" BY THE CLOCK
0012    /R1=RESPONSE
0013    /ON 1=HOUSE LIGHT ON
0014    /ON 3=REINFORCEMENT ON
0015    /REINFORCEMENT DURATION=4"
0016
0017    S. S. 1,
0018    S1,
0019        START: ON 1--->S2
0020    S2,
0021        Z1--->S3
0022    S3,
0023        R1: ON 3--->S4
0024    S4,
0025        4": OFF 3--->S2
0026
0027    S. S. 2,
0028    S1,
0029        START--->S2
0030    S2,
0031        30": Z1--->SX
0032    $

```

Table 14 representing Figure 16 is a straight-forward translation.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024
0025
0026
0027
0028

```

////////////////////
//
//          TABLE 14
//          VARIABLE INTERVAL
//
////////////////////

```

```

/CHAP 1
/FIG 16
/VARIABLE INTERVAL OR VI
/R1=RESPONSE
/ON 1=HOUSE LIGHT ON
/ON 3=REINFORCEMENT ON
/REINFORCEMENT DURATION=4"
/LIST A=VI VALUES
LIST A=10", 5", 20", 15"

S. S. 1,
S1,
    START: ON 1; LIST V=A(K)---->S2
S2,
    V T: LIST V=A(K)---->S3
S3,
    R1: ON 3---->S4
S4,
    4": OFF 3---->S2
$

```

Table 15 illustrates Figure 17. Line 21 demonstrates that WITH P requires a number between 001 and 999. Probabilities must be expressed in terms of chances out of 1000, and therefore the .02 of Figure 17 is entered as 020 in line 21. The ELSE transition to SX of Figure 17 happens by default, and therefore has been omitted in Table 15.

DS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024

TABLE 15
RANDOM RATIO

```

/CHAP 1
/FIG 17
/RANDOM RATIO
/R1=RESPONSE
/ON 1=HOUSE LIGHT ON
/ON 3=REINFORCEMENT ON
/REINFORCEMENT DURATION=4"

S. S. 1,
S1,
START: ON 1--->S2
S2,
R1: WITH P=020; THEN; ON 3--->S3
S3,
4": OFF 3--->S2
$

```

Table 16 is a straight-forward representation of Figure 18.

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

0001

0002

0003

0004

0005

0006

0007

0008

0009

/CHAP 1

0010

/FIG 18

0011

/RANDOM INTERVAL (FARMER, 1963)

0012

/R1=RESPONSE

0013

/ON 1=HOUSE LIGHT ON

0014

/ON 3=REINFORCEMENT ON

0015

/REINFORCEMENT DURATION=4"

0016

0017

S. S. 1,

0018

S1,

0019

START: ON 1--->S2

0020

S2,

0021

Z1--->S3

0022

S3,

0023

R1: WITH P=020; THEN; ON 3--->S4

0024

: ELSE--->S2

0025

S4,

0026

4": OFF 3--->S2

0027

0028

S. S. 2,

0029

S1,

0030

START--->S2

0031

S2,

0032

5": Z1--->SX

0033

\$

```

////////////////////
//
//          TABLE 16
//          RANDOM INTERVAL-TYPE 1
//
////////////////////

```

In Table 17 the ELSE transition of Figure 15 has been omitted, since transition to SX is the default transition of decision functions.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////
0002      /
0003      TABLE 17
0004      RANDOM INTERVAL-TYPE 2
0005      \
0006      //////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 19
0011      /RANDOM INTERVAL (CATANIA AND REYNOLDS, 1968)
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=4"
0016
0017      S. S. 1,
0018      S1,
0019      START: ON 1--->S2
0020      S2,
0021      5": WITH P=020; THEN--->S3
0022      S3,
0023      R1: ON 3--->S4
0024      S4,
0025      4": OFF 3--->S2
0026      $

```

Table 18 line 35 demonstrates that the RAND function of Figure 20 translates in a straight forward manner.

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////////////////
0002      /
0003      TABLE 18
0004      DISCRIMINATION (RAND)
0005      \
0006      //////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 20
0011      /DISCRIMINATION PROCEDURE
0012      /MEAN INTER-TRIAL-INTERVAL=40"
0013      /R1=RESPONSE
0014      /ON 1=HOUSE LIGHT ON
0015      /ON 2=DISCRIMINATIVE STIMULUS
0016      /ON 3=REINFORCEMENT ON
0017      /REINFORCEMENT DURATION=3"
0018      /LIST A=INTER-TRIAL-INTERVAL VALUES
0019      LIST A=10", 20", 30", 40", 50", 60", 70"
0020
0021      S. S. 1,
0022      S1,
0023          START: ON 1; Z1--->S2
0024      S2,
0025          V T: ON 2--->S3
0026      S3,
0027          R1: OFF 2; ON 3--->S4
0028      S4,
0029          3": OFF 3; Z1--->S2
0030
0031      S. S. 2,
0032      S1,
0033          START--->S2
0034      S2,
0035          Z1: RAND V=A--->SX
0036      $

```

Table 19 demonstrates the use of two variable lists in lines 20 and 21. Lines 29 and 30 are somewhat different from Figure 21. First, the OFF D that occurs after both true and false transitions following IF D=1 has been placed immediately following R1. Therefore, OFF D does not have to be repeated on line 30, thus saving some state table space. Otherwise the state table resembles the state diagram.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001
0002
0003
0004
0005
0006
0007
0008
0009      /CHAP 1
0010      /FIG 21
0011      /RAND FUNCTION
0012      /STIMULI IN RANDOM ORDER FOR GENERALIZATION TESTS
0013      /ON D=GENERALIZATION STIMULI 1-7
0014      /ON 3=REINFORCEMENT ON
0015      /ON 8=HOUSE LIGHT ON
0016      /R1=RESPONSE
0017      /REINFORCEMENT DURATION=3"
0018      /LIST A=STIMULI
0019      /LIST B=INTER-TRIAL-INTERVAL VALUES
0020      LIST A=1, 1, 1, 1, 1, 2, 7, 4, 5, 6
0021      LIST B=10", 20", 30", 40", 50", 60", 70"
0022
0023      S. S. 1,
0024      S1,
0025          START: ON 8; Z1--->S2
0026      S2,
0027          V T: ON D--->S3
0028      S3,
0029          R1: OFF D; IF D=1; THEN; ON 3; Z1--->S4
0030          ; ELSE; Z1----->S2
0031      S4,
0032          3": OFF 3--->S2
0033
0034      S. S. 2,
0035      S1,
0036          START--->S2
0037      S2,
0038          Z1: RAND V=B; RAND D=A--->SX
0039      $

```


Table 20 is a straight-forward representation of Figure 22.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001
0002
0003
0004
0005
0006
0007
0008
0009      /CHAP 1
0010      /FIG 22
0011      /SIDMAN AVOIDANCE
0012      /RS=10"
0013      /SS=2"
0014      /SHOCK DURATION=. 5"
0015      /R1=RESPONSE
0016      /ON 1=HOUSE LIGHT ON
0017      /ON 2=SHOCK ON
0018
0019
0020      S. S. 1,
0021      S1,
0022          START: ON 1---->S2
0023      S2,
0024          2": ON 2---->S3
0025          R1: Z1---->S4
0026      S3,
0027          . 5": OFF 2; Z2---->S2
0028      S4,
0029          10": ON 2---->S3
0030          R1: Z1---->S4
0031
0032      S. S. 2,
0033      S1,
0034          START---->S2
0035      S2,
0036          800Z1+500Z2+60'---->STOP
0037      $

```

Table 21 differs from Figure 23 in line 21, where SUBZGO V has replaced the equivalent statements SET V=V-1; IF V=0; THEN; of Figure 23. The SUBZGO function is faster and requires less state table space. However, it can only be used when the false transition (V is not yet 0) is to SX, as in Figure 23.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      ////////////////////////////////////////////////////
0002      /
0003      TABLE 21
0004      INTERLOCK
0005      \
0006      //////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 23
0011      /INTERLOCK SCHEDULE OF REINFORCEMENT
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=3"
0016
0017      S. S. 1,
0018      S1,
0019      START: ON 1; SET V=49--->S2
0020      S2,
0021      R1+2": SUBZGO V; THEN; SET V=49--->S3
0022      S3,
0023      R1: ON 3--->S4
0024      S4,
0025      3": OFF 3--->S2
0026      $

```

Table 22 is a traight forward representation of Figure 24, and the same is true of Table 23 and Figure 25.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      ////////////////////////////////////////////////////
0002      /
0003      TABLE 22
0004      PROGRESSIVE RATIO
0005      \
0006      ////////////////////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 24
0011      /PROGRESSIVE RATIO (HODOS, 1961)
0012      /R1=RESPONSE
0013      /ON 1=HOUSE LIGHT ON
0014      /ON 3=REINFORCEMENT ON
0015      /REINFORCEMENT DURATION=3"
0016      /SESSION TERMINATES ATER 15' WITHOUT A RESPONSE
0017
0018      S. S. 1,
0019      S1,
0020          START: SET V=5, I=5--->S2
0021      S2,
0022          V R1: SET V=5+I, I=V; ON 3--->S3
0023      S3,
0024          3": OFF 3--->S2
0025
0026      S. S. 2,
0027      S1,
0028          START: ON 1--->S2
0029      S2,
0030          15'--->STOP
0031          R1--->S2
0032      $

```

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////
0002      /
0003      TABLE 23
0004      CHAINED VI VI
0005      \
0006      //////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 25
0011      /CHAINED VI VI
0012      /ON 1=STIMULUS CONDITION 1
0013      /ON 2=STIMULUS CONDITION 2
0014      /ON 3=REINFORCEMENT ON
0015      /R1=RESPONSE
0016      /LIST A=VI VALUES FOR STIMULUS CONDITION 1
0017      /LIST B=VI VALUES FOR STIMULUS CONDITION 2
0018      LIST A=80.74",28.74",175.74",38.74",115.74",12.67",
0019      65.74",50.74",20.17",6"
0020      LIST B=12",231.48",77.48",57.48",40.33",25.33",
0021      171.48",131.48",351.48",101.48"
0022
0023      S. S. 1,
0024      S1,
0025      START: ON 1; LIST V=A(K); LIST W=B(J)--->S2
0026      S2,
0027      V T: LIST V=A(K)--->S3
0028      S3,
0029      R1: OFF 1; ON 2--->S4
0030      S4,
0031      W T: LIST W=B(J)--->S5
0032      S5,
0033      R1: ON 3--->S6
0034      S6,
0035      3": OFF 2,3; ON 1--->S2
0036      $

```

Table 24 in lines 16, 17, 18 and 19 illustrate how lists longer than one line can be entered. On both lines 16 and 18 the final character before carriage return is and must be a comma. Of course, lists longer than two lines may also be continued by means of commas.

DS/8 SUPER S K E D 3.20

JUN 15 1978.

PAGE ONE

```

0001          ///////////////////////////////////////////////////
0002          /
0003          TABLE 24
0004          CONCURRENT VI VI
0005          \
0006          ///////////////////////////////////
0007
0008
0009          /CHAP 1
0010          /FIG 26
0011          /CONCURRENT VI-VI WITH COD
0012          /R1=RESPONSE OF TYPE 1
0013          /R2=RESPONSE OF TYPE 2
0014          /ON 3=REINFORCEMENT
0015          /REINFORCEMENT DURATION=3"
0016          LIST A=80.74",12.67",175.74",38.74",115.74",
0017          28.74",65.74",50.74",20.17",6"
0018          LIST B=12",101.48",77.48",57.48",40.33",
0019          351.48",171.48",131.48,25.33",231.48"
0020
0021          S. S. 1,
0022          S1,
0023          START:LIST V=A(K)--->S2
0024          S2,
0025          V T:LIST V=A(K)--->S3
0026          S3,
0027          R1: IF S. S. 3=2; THEN; ON 3--->S4
0028          : ELSE--->S5
0029          S4,
0030          3": OFF 3--->S2
0031          S5,
0032          3"--->S3
0033
0034          S. S. 2,
0035          S1,
0036          START:LIST W=B(J)--->S2
0037          S2,
0038          W T:LIST W=B(J)--->S3
0039          S3,
0040          R2: IF S. S. 3=3; THEN; ON 3--->S4
0041          : ELSE--->S5
0042          S4,
0043          3": OFF 3--->S2

```

0044	S5,	
0045		3"--->S3
0046		
0047	S. S. 3,	
0048	S1,	
0049		R1--->S2
0050		R2--->S3
0051	S2,	
0052		R2--->S3
0053	S3,	
0054		R1--->S2
0055	\$	

Although relatively long, Table 25 is a straight-forward representation of Figure 27.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////
0002      /
0003      /
0004      /
0005      /
0006      /
0007      /
0008      /
0009      /CHAP 1
0010      /FIG 27
0011      /CONCURRENT VI 1'-VI 1'--CHAIN FI 10"-FI 30"
0012      /ON1=STIMULUS CONDITION1
0013      /ON2=STIMULUS CONDITION2
0014      /ON3=STIMULUS CONDITION3
0015      /R1=RESPONSE OF TYPE 1
0016      /R2=RESPONSE OF TYPE 2
0017      /ON3=REINFORCEMENT ON
0018      /REINFORCEMENT DURATION=3"
0019      /LIST A=VI VALUES FOR STIMULUS CONDITION1
0020      /LIST B=VI VALUES FOR STIMULUS CONDITION2
0021      LIST A=80.74", 12.67", 175.74", 38.74", 115.74",
0022      28.74", 65.74", 50.74", 20.17", 6"
0023      LIST B=12", 101.48", 77.48", 57.48", 40.33",
0024      351.48", 171.48", 131.48", 25.33", 231.48"
0025
0026      S. S. 1,
0027      S1,
0028      START: LIST V=A(J); ON1---->S2
0029      S2,
0030      V T: LIST V=A(J)---->S4
0031      Z1---->S3
0032      S3,
0033      Z2---->S2
0034      S4,
0035      R1: IF S. S. 3=2; THEN; OFF1; ON2; Z1---->S7
0036      : ELSE---->S6
0037      Z1---->S5
0038      S5,
0039      Z2---->S4
0040      S6,
0041      3"---->S4
0042      S7,
0043      10"---->S8
0044      S8,
0045      R1: ON 3---->S9
0046      S9,

```

0047 3": ON1; OFF 2, 3; Z2--->S2
0048
0049 S. S. 2,
0050 S1,
0051 START: LIST W=B(K)--->S2
0052 S2,
0053 W T: LIST W=B(K)--->S4
0054 Z1--->S3
0055 S3,

OS/B SUPER S K E D 3.20

JUN 15 1978

PAGE TWO

0056 Z2--->S2
0057 S4,
0058 R2: IF S. S. 3=3; THEN; OFF1; ON8; Z1--->S7
0059 : ELSE----->S6
0060 Z1--->S5
0061 S5,
0062 Z2----->S4
0063 S6,
0064 3"--->S4
0065 S7,
0066 30"--->S8
0067 S8,
0068 R2: ON 3--->S9
0069 S9,
0070 3": ON1; OFF 3, 8; Z2--->S2
0071
0072 S. S. 3,
0073 S1,
0074 START--->S2
0075 S2,
0076 R2--->S3
0077 S3,
0078 R1--->S2
0079 \$

Table 26 is even longer, and it represents the combined Figures 28, 29, 30 and 31. Except for the combination of Figures 30 and 31 into the same State Set 7, and except for the omission of unneeded default transitions to SX, Table 26 is quite similar to the corresponding Figures.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001          //////////////////////////////////////////////////
0002          /
0003          TABLE 26
0004          TIME ALLOCATION
0005          \
0006          //////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 28
0011      /S. S. 5, AND S. S. 6, =FIG 29
0012      /S. S. 7, =FIG 30, FIG 31
0013
0014      /TIME ALLOCATION PROCEDURE
0015      /CHOICE BETWEEN TWO SHOCK DISTRIBUTIONS
0016      /COD=2"
0017      /R1=RESPONSE OF TYPE 1
0018      /R2=RESPONSE OF TYPE 2
0019      /ON 1=WHITE LIGHT
0020      /ON 2=RED LIGHT
0021      /ON 9=GREEN LIGHT
0022      /ON 4=SHOCK ON
0023      /LIST A=SHOCK DISTRIBUTION 1
0024      /LIST B=SHOCK DISTRIBUTION 2
0025      LIST A=3", 7", 9", 1", 10", 5", 6", 2", 8", 4"
0026      LIST B=3", 7", 1", 10", 16", 9", 4", 8", 14", 11"
0027      CTRS=40
0028
0029      S. S. 1,
0030      S1,
0031          START: ON 9--->S2
0032      S2,
0033          R1: OFF 9; ON 1; ADD 1--->S3
0034          . 01": ADD 6--->SX
0035      S3,
0036          2": OFF 1; ON 2--->S4
0037          R2: ADD 4--->S5
0038      S4,
0039          R2: OFF 2; ON 1; ADD 2--->S5
0040          . 01": ADD 7--->SX
0041      S5,
0042          2": OFF 1; ON 9--->S2
0043          R1: ADD 3--->S3

```

```

0044
0045     S. S. 2,
0046     S1,
0047         START: LIST V=A(K)---->S2
0048     S2,
0049         V T: LIST V=A(K); IF S. S. 1=2; THEN; ON 4---->S3
0050         : ELSE: IF S. S. 1=3; THEN; ON 4---->S3
0051         : ELSE: IF S. S. 1=5; THEN; ON 4---->S3
0052     S3,
0053         . 5": OFF 4; ADD 8; Z1---->S2
0054
0055     S. S. 3,

```

DS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE TWO

```

0056     S1,
0057         START: LIST W=B(J)---->S2
0058     S2,
0059         W T: LIST W=B(J); IF S. S. 1=3; ON 4---->S3
0060         : ELSE: IF S. S. 1=4; THEN; ON 4---->S3
0061         : ELSE: IF S. S. 1=5; THEN; ON 4---->S3
0062     S3,
0063         . 5": OFF 4; ADD 9; Z2---->S2
0064
0065     S. S. 4,
0066     S1,
0067         START---->S2
0068     S2,
0069         . 01": ADD 5---->SX
0070
0071
0072     S. S. 5,
0073     S1,
0074         START---->S2
0075     S2,
0076         Z1: SET Z=10---->S3
0077     S3,
0078         1": IF Z<20; THEN; ADD Z---->SX
0079         R1: ADD C(Z)---->S2
0080
0081     S. S. 6,
0082     S1,
0083         START---->S2
0084     S2,
0085         Z2: SET Y=21---->S3
0086     S3,
0087         1": IF Y<31; THEN; ADD Y---->SX
0088         R2: ADD C(Y)---->S2
0089
0090     S. S. 7,
0091     S1,
0092         START: OPEN---->S2
0093     S2,

```

```
0094          R1: RECORD 0---->SX
0095          R2: RECORD 1---->SX
0096          Z1: RECORD 2---->SX
0097          Z2: RECORD 3---->SX
0098          5': FOR L=1, 1, 31; WRITE C(L), 16; SET
C(L)=0; NEXT--->SX
0099          100Z1+100Z2: WRITE 0, 15--->S3
0100      S3,
0101          . 01": CLOSE: THEN--->STOP
0102      $
```

Table 27 represents Figure 32 and also shows a TYPE statement in line 20.

OS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      ////////////////////////////////////////////////////
0002      /
0003      TABLE 27
0004      TIME AND TYPE
0005      \
0006      //////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 32
0011      /ILLUSTRATES USE OF TIME
0012      /SESSION STARTS AT 13:00
0013      /TYPE OUT START AT 1300
0014      /R1=RESPONSE
0015      /ON 3=REINFORCEMENT ON
0016
0017      S. S. 1,
0018      S1,
0019          1': TIME A; IF A=1300; THEN;
0020          TYPE "!START. IT IS NOW 13:00!"---->S2
0021      S2,
0022          R1: ON 3---->S3
0023      S3,
0024          3": OFF 3-->S2
0025      $

```

Table 28 has a number of changes from the structure of Figure 33. Since there are a number of IF statements in a row, it is necessary to rearrange the decisions to clearly specify which ELSE transitions belong to which decisions as in Table 9. For example, line 22 has rearranged the IF A=10 of Figure 33 to IF A<11. This permits completing the SX transition if A is <11 on line 22. Next, on line 23, the SET statements A=1, X=c(11)/4, and S=0 are all completed before the FOR loop of line 24. Remember that FOR loops take a great deal of time to be completed. Again on line 25 the decision function of Figure 33 IF S>X is reversed, and separately checked for S<X and S=X on line 25 to continue the FOR loop until S>X. The remainder of the state table is straight-forward.

DS/8 SUPER S K E D 3.20

JUN 15 1978

PAGE ONE

```

0001      //////////////////////////////////////
0002      /
0003      TABLE 28
0004      QUARTER LIFE
0005      \
0006      //////////////////////////////////////
0007
0008
0009      /CHAP 1
0010      /FIG 33
0011      /FIXED INTERVAL 100"
0012      /REINFORCE ONLY IF QUARTER LIFE IS > .50
0013      /ON 1=BRIEF FLASH OF LIGHT FOR QL .50
0014      /ON 3=REINFORCEMENT ON
0015      CTRS=11
0016
0017      S. S. 1,
0018      S1,
0019      START: SET A=1--->S2
0020      S2,
0021      R1: ADD C(A).11--->SX
0022      10": ADD A; IF A<11; THEN---->SX
0023      ; ELSE; SET A=1, X=C(11)/4, S=0;
0024      FOR K=1, 1, 10; SET S=S+C(K);
0025      IF S<X[THEN; NEXT; ]; IF S=X[THEN; NEXT; ];
0026      ELSE; BREAK;
0027      ALWAYS; FOR L=1, 1, 11; SET C(L)=0; NEXT;
0028      IF K>5[THEN---->S3];
0029      ELSE; ON 1--->S5
0030      S3,
0031      R1: ON 3--->S4
0032      S4,
0033      3": OFF 3--->S2
0034      S5,
0035      .5": OFF 1--->S2

```

CHAPTER 4

SYNTAX RULES FOR STATE TABLES

The following chapter is a complete description of the components of the state notation language. There are ten major components, some of which are further subdivided into specific functions:

1. State Sets
2. States
3. Input Functions
4. Outputs - External outputs (ON, OFF), internal outputs (Z pulses), TYPE statements
5. Transitions
6. Data Acquisition
7. Decision operators IF, WITH, FOR, SUBZGO
8. Variables
9. Operators on Variables
10. Analog to digital and digital to analog conversion (CALLO)

Each of these major components and specific functions will be described under 3 headings:

- A. Description of component
- B. Syntax of the function
- C. Example of use of the component and its syntax

SECTION 1: STATE SETS

A. DEFINITION: A state set consists of a set of states, only one of which may be present at any point in time. One or more state sets are necessary to describe the procedure, and they must be numbered sequentially.

B. SYNTAX: State sets are identified by the letters S.S.n, where n stands for a number from 1 to 4095. The period must follow each S and the comma serves as a terminator. An example of the state set identifier follows:

S.S.2,

There has to be at least one state set and the first state set number must be 1. Each succeeding state set must be numbered consecutively. When deciding how to arrange the state sets, remember that inputs operate sequentially on state sets beginning with State Set 1. This temporal sequence may be important when using decision functions or Z pulses to synchronize parallel state sets (see Chapter 1, Section 1 on decision functions, and chapter 2 on priorities).

C. EXAMPLES: Theoretically, any procedure could be described using a single state set. However, the use of parallel state sets often clarifies complex procedures. For example, concurrent schedules are best diagrammed in parallel state sets, one for each schedule. Then the programmer can look at each schedule separately. Where schedules interact (as is the case with a change-over-delay) the interaction can be clearly notated using a

combination of Z pulses and with IF statements using state set numbers. For an example of a concurrent schedule, see Figure 24 in Chapter 1.

Under some circumstances, parallel state sets can result in fewer states than would result from a diagram incorporating all functions in a single state set. In this respect, a state set resembles the subroutines used by other programming languages. Furthermore, a frequently used subroutine or state set can be established and saved for use with other programs in the future.

SECTION 2: STATES

A. DEFINITION: A state describes the contingencies in effect while the state is present. Contingencies are rules that define what outputs will occur given that a particular input occurs, and also contingencies define what state will be produced by an input (see Section 1 of Chapter 1).

B. SYNTAX: States are numbered sequentially beginning with State 1, and are represented by the letter S followed by the state number and terminated by a comma. When a state table is loaded into the computer, all state sets are in State 1. An example state identifier is S3,(CR/LF). CR/LF stands for carriage return line feed. It is illegal to write anything on the same line with the state number.

C. EXAMPLES: States are one of the basic elements of state notation. Examples of states will occur throughout this chapter.

SECTION 3: INPUTS

There are 4 major types of inputs:

1. Response (digital)
2. Experimenter initiated event signals
3. Z pulses
4. Time

1. RESPONSE INPUTS

A. DEFINITION: Digital response inputs are specified by the letter R followed by a number ranging from 1 to 49152. The number of digital inputs assigned to a station is specified by the initial dialogue.

Digital response inputs may be simulated from the terminal (see Chapter 5). The external digital response input lines are examined once every clock tick (.01") and only the transition from OFF to ON is recorded as an input. The input line must be off for at least one clock tick before a new digital input can occur. This on-off feature limits the digital input rate to 50 inputs per second on any one input channel.

Response durations can be recorded by assigning one input channel to the onset of a response and a second channel to the end of the response. This assignment may be made electrically or a program parameter can be set to automatically assign a response channel to the absence of a signal on an input line.

For FR and some other reinforcement schedules it is often useful to require a fixed number of responses for transition. For this purpose, it is possible to associate a counter with response inputs. Writing 5 R1's, then, specifies that the fifth response of type 1 will produce the transition. The first 4 R1's merely are counted. Introducing a counter also requires introducing a method for re-establishing the count requirement. The rule that has been most useful has been to reset all fixed counters associated with a state on entry to the state. Furthermore, to permit a null transition after a count has been completed, the counter causing transition is also reset. Thus, 5 R1:Z1--->SX will permit the production of a Z1 for every 5 R1's as long as the state is in effect.

Sometimes, it is useful to remember how many responses occurred when a state was last in effect. Since the fixed response count is reset when the state is re-entered, a second method of counting permits remembering that some responses did occur when the state was last present. Variable counters permit this usage by requiring the programmer to specify when the counter is to be reset.

B. SYNTAX: With the exception of an OR'd input, all inputs must follow either a carriage return/line feed combination, or a space or spaces following a CR/LF, or a tab (CTRL I). A single response of type 5 would be written in State 1 of State Set 1 as follows:

S. S. 1,
 S1,
     R5:----->S2

SINGLE RESPONSE

The number identifier of a response input (5 in the preceding example) can range from 1 to 49152, and may not be replaced by a variable.

FIXED RESPONSE COUNT

Responses with fixed count requirements must also follow CR/LF, CR/LF and spaces, or CR/LF and tab, except in OR'd statements. If 5 responses of type 3 were required for transition from State 1 to State 2 of State Set 1, the fixed count would be written:

S. S. 1,
 S1,
     5R3----->S2

VARIABLE RESPONSE COUNT

Variable response requirements must also follow CR/LF, CR/LF space(s), or CR/LF and tab. The variable may be subscripted as in the following example:

S. S. 1,
 S1,
     V(K) R1----->S2

There must be a space between the last portion of the variable and the letter R; i.e. between ")" and "R" in the preceding example.

OR'd Response

Sometimes inputs of different types all have the same outputs and transitions. To avoid the necessity of repeating the same outputs and transition lines, inputs may be OR'd together, separated by a + (plus). The first input of the OR'd set must follow CR/LF, CR/LF space(s), or CR/LF tab. All other OR'd

inputs must follow a +. Fixed and variable response counts (as well as Z and time inputs) may be included in the OR'd statement. An example follows:

S. S. 2,

S1,

2 R1+R2+V R3: ON1----->S2

C. EXAMPLES: Response inputs occur in almost all state tables. Special techniques worth noting include the possibility of using several different fixed or variable response counts of the same response type in the same state, if each of the smaller counts has an SX transition. An example follows:

S. S. 1,

S1,

2 R1: ON1----->SX

4 R1: ON2----->SX

6 R1: ON3----->S2

S2,

1": OFF1, 2, 3----->S1

In the preceding example, the second R1 in State 1 turns on 1, the 4th R1 turns on 2 and the 6th R1 turns on 3 and goes to State 2. Since the transition lines for 2 R1 and 4 R1 both go to SX, State 1 is still present when the 6th R1 occurs.

2. EXPERIMENTER INPUTS

A. DEFINITION: Inputs can be generated by the experimenter to start a session or to change conditions during a session. The experimenter can press a push-button to simulate a response of the subject. Alternatively, three methods for simulating responses from the terminal are available.

I. Any response from R1 to R49152 can be typed on the terminal with MON running even if the response is not specified in the initial dialogue. If the appropriate response is included in the state table, then the simulation of that response will cause the specified transition to occur.

II. TYPING G on the terminal in MON will cause an R12 to be simulated for all stations that are active. R12 therefore can be used in the state table to produce the desired effect.

III. TYPING S in MON will generate an R1200 to any specified stations. The word START as an input in a state table is equivalent to R1200. Therefore, START can be reserved for the session start and may be produced by typing S.

B. SYNTAX: Simulated R's have the same syntax as digital R's, except for the START input. START may follow CR/LF, CR/LF space, or CR/LF tab.

C. EXAMPLE: The following example shows how start may be used.

S. S. 1,

S1,

START: ON1----->S2

3. Z PULSES

A. DEFINITION: Z pulses are inputs for one or more transitions that are produced as outputs by other transitions. They may be used to synchronize parallel state sets. Although Z pulses may be produced by other Z pulses, there must be either a response or time input at the beginning of a chain of Z's for the Z's to occur. Therefore, Z pulses can be thought of as indicating that a response (or time) has occurred in a specific state of a specific state set.

B. SYNTAX: A Z input resembles a response input but is limited to the range from Z1 to Z12. As an input, the Z must be preceded by CR/LF, CR/LF tab, or CR/LF space(s). A Z may be produced by one or more transitions as an output and may occur in one or more transitions as an input. Z's may be counted by fixed counters (e.g. 5 Z10) or by variable counters (e.g. V(K) Z1) as is true of responses. Z's may be OR'd with responses, times or other Z's.

C. EXAMPLE: Z's are most useful for permitting several different inputs all to generate the same transition. For example, if an R1 in State 2 or 15" in State 3 are both to cause a sequence of events to occur in time consisting of turning on 2 stimuli in succession, then it is most convenient to have the 15" time and the R1 both generate a Z1 as an output and use the following state set to control the sequential stimuli:

S. S. 2,

S1,

Z1: ON1----->S2/10" IN S3 or an R1 in S2 of S. S. 1

S2,

5": OFF1: ON2----->S3

S3,

5": OFF2----->S1

In this example, the Z1 serves as a call to a common subroutine. The subroutine needs only to be written once, and may be triggered by a number of different inputs in different states.

4. TIME INPUTS

A. DEFINITION. Time after entry to a state can cause a transition. Time expressions are equivalent to a fixed count of .01" clock pulses. Therefore, an expression such as 5" is equivalent to a count of 500 basic clock ticks. Since time expressions are equivalent to fixed counts of a basic clock tick, the same rules apply to time expressions that apply to counted responses or Z's. The time expression is reset whenever a state is entered by any transition. Also, time expressions reset themselves when the time specified in the expression has elapsed. Of course, times will be counted only when the state containing the time expression is in effect. Variable timers may be used, but the variable must be re-initialized explicitly, when the timer is to be reset. When a variable is set to a time expression, then the counters of the variable are adjusted by the compiler to reflect the appropriate count of .01" ticks. Thus, SET A=3" will result in A containing the number 300.

B. SYNTAX: Time expressions may be in minutes, seconds, or 1/100ths of a second. Thus the following time expressions are all valid:

5'
 5' 5"
 5' 5.5"
 .02"
 .01"
 3' 2. "

At the time this document is being prepared, only one time expression may be used in a state. Thus the following is illegal:

S. S. 1,
 S1,
     5": ON1----->SX
     10"----->S2

C. EXAMPLES: The following example is an autoshaping procedure that is completely controlled by temporal contingencies. Responses are merely recorded:

S. S. 1,
 S1,
     30": ON1----->S2/ START SD AFTER FIXED ITI
 S2,

```

5":OFF1; ON2----->S3/ END SD PRESENT SR
R1:ADD 0----->SX/ RECORD R'S IN CTR 0
S3,
3":OFF2----->S1/ END SR AND START ITI

```

Other examples of temporal contingencies can be found in Chapter 1.

SECTION 4: OUTPUTS

Output functions are of two types. The external output functions operate output channels sending stimuli to the subject, or they operate special devices attached to the computer. Internal outputs include the generation of Z pulses, and the typing of messages on the terminal.

1. EXTERNAL OUTPUTS

i. ON and OFF

A. DEFINITION: The ON and OFF outputs turn on or off output channels connected to the interface. Each channel is independent; and the number of channels and correspondence between specific interface output lines and channel numbers must be specified for each station in the initial dialogue (see section). Up to 49152 separate output channels may be specified for a single station, and each channel is identified by a unique number.

B. SYNTAX: ON and OFF must be preceded by a colon or semi-colon and must be contained in a transition beginning with an input. ON and OFF must be followed by one or more digits, specifying the channel to be controlled. If more than one channel is to be turned ON or OFF, the channel numbers must be separated by a comma. Thus, ON 1,2,15 means turn on channels 1,2 and 15. Channel numbers may be replaced by variables that contain the appropriate number. Thus ON 1,A,3 is a valid command.

C. EXAMPLE: The following example demonstrates turning on stimulus 1, for 5 seconds whenever a response occurs. The output channel is then turned off.

```

S. S. 1,
S1,
START----->S2
S2,

```

```

R1: ON1----->S3
S3,
  5": OFF1----->S2

```

2. INTERNAL OUTPUTS

i. Z pulses:

A. DEFINITION: The Z pulse is generated as an output in one or more transitions and is used as an input by other transitions. There are only 12 Z pulses available in the system.

B. SYNTAX: As an output function, the Z must be preceded by a colon or semi-colon in a transition line, beginning with a valid input. The Z is followed by the appropriate Z number. Thus, :Z3 means output a Z type 3. Multiple Z's may be produced on the same transition when separated by commas. Thus, :Z1,3 means generate Z1 and Z3.

C. EXAMPLE: In the following example an R1 will produce a Z3, that serves to turn on stimulus 5.

```

S. S. 1,
S1,
  START----->S2
S2,
  R1: Z3----->SX          /Z3 as Output
  Z3: ON5----->S3          /Z3 as Input
S3,
  5": OFF5----->S2

```

ii. TYPE

A. DEFINITION: The type statement will print a specified message on the user's OS/8 terminal. It will interrupt any other OS/8 typing that is underway.

B. SYNTAX: The type statement is an output function and must follow a colon or semi-colon in a transition expression. The message to be typed must be enclosed by " which are not legal inside of the message. The character ! inside the quotes will generate one carriage return line feed combination.

C. EXAMPLE: The following example tests the current value of variable A, used in a FR schedule. If A is not set to a number before start, then it is set to 10 and a message typed. If A is not zero when start occurs, a second message is typed.

S. S. 1,

S1,

START: IF A=0; THEN SET A=10, B=A; TYPE"!FR10!"-->S2
: ELSE SET B=A; TYPE"! FR NOT 10!"-->S2

S2,

B R1: SET B=A; ON1----->S3

S3,

4": OFF1----->S2

SECTION 5: TRANSITIONS

A. DEFINITION: Transitions are composed of an input section and may include one or more decision sections, one (or more) output sections and must include one or more transfer sections. Multiple output sections and transfer sections can occur when decision functions are included as part of the transition expression. The input section consists of an expression designating one of four types (START, R, Z or T) of events that can cause a transition. The decision section (if one is present) includes one of three types of decision functions (WITH, IF, SUBZQO) each of which may have 2 output and transfer sections. The output section can precede or follow, or both precede and follow decision sections, but an output section is not mandatory. The output section begins with a colon or semi-colon and lists one or more output functions also separated by colons or semi-colons. The output functions (including data acquisition and variable operators) will be activated if their transition section is executed. The transfer section consists of one or more hyphens (-) followed by a right angle bracket (>), followed by the state identifier of the next state to be active if the transfer is executed. The transfer section may also consist of the null transition SX or the STOP statement.

B. SYNTAX: A transition statement must begin with a new line, that is either following CR/lf, or CR/lf TAB or SPACES. It must begin with an input expression and end with a transfer statement to a new state, SX, or STOP. It may include output and decision functions.

C. EXAMPLES: All of the valid examples of state tables contain one or more transition expressions.

SECTION 6: DATA ACQUISITION

There are three methods for acquiring data. The first is to use a special routine called ADD, or the SET FUNCTIONS to store numbers in the C array. Methods for printing this array or creating files containing the C array are discussed in the chapter on the Monitor. The second method is to write the data (usually stored in the C array or other variable arrays) into a file by means of an internal memory buffer. The third method is to store sequential inter-event times and codes in a file by means of the internal buffer. All three methods may be used simultaneously.

1. THE C ARRAY

A. DEFINITION: The set of variables in the C array are reserved for recording data. Room for the C array must be reserved at the beginning of the state table before the first occurrence of S.S.1, in a statement of the form:

C=500 (CR/LF)

which reserves 501 recording counters from C(0) to C(500). If no counters are reserved, the compiler assigns C(0) automatically providing one counter. The maximum number of counters reserved depends on the length of the state table with the restriction that the maximum length of the state table including counters and other variables must be less than 3072 (decimal) locations. Each recording counter requires two memory locations since each element of the C array consists of 2 12-bit numbers (double precision) that can contain values ranging from 0 to 16,777,215.

Numbers can be placed in the C array by means of the variable operators, SET or ADD. ADD followed by a number (i.e., ADD 10) results in incrementing a recording counter (i.e., C10) by one. It is equivalent to saying add 1 to the current contents of recording counter 10. SET can be used to manipulate recording counters in a number of different ways. For example, SET C(3)=45 will unconditionally set recording counter 3 to the number 45. SET C(3)=C(3)+45 will add 45 to the current contents of recording counter 3. (For other methods of altering the contents of variables see section on variable operators.)

B. SYNTAX: To reserve a number of recording counters, the dimension for the C array must be established at the beginning of the state table (usually following initial comments) before the first state set is established by S.S.1.. The statement reserving space is of the form:

(CR/LF)

C=50 (CR/LF)

S. S. 1,

The syntax of the variable operators is discussed in section 7. However, the function ADD followed by a number is reserved for incrementing recording counters in the C array. ADD is located in the output section of a transition and therefore must be preceded by a colon or semi-colon. The number following ADD specifies the recording counter which is to be incremented by one. The number must not be larger than the number of counters reserved for the C array.

C. EXAMPLE: Figure 26 of Chapter 1 and its associated state table demonstrate the use of the ADD function and the C array to collect the number of responses and other events in a session, and Figure 27 demonstrates the collection of latency distributions with these functions.

2. THE WRITE COMMAND

A. DEFINITION. The WRITE command can be used to store any number from 0 to 16,777,215 in two successive 12-bit locations of a data file. Optionally the WRITE command can use a code from 0 to 15 that limits the precision of the written number from 0 to 1,048,575 with larger numbers reserved for the codes. To use the WRITE command, MON must be used to reserve a data file on the mass-storage device.

The state table must then contain one OPEN statement, which must occur before a WRITE statement is operative. There also may be one or more CLOSE statements which will add one block of zeros to the data file. The CLOSE function does not make it impossible to continue to WRITE in the data file, but it merely fills out a block in the file with zero's, and transfers the file to the mass-storage device. If there still is room in the data file, WRITE will continue to function until either the file is completely filled or the session stops.

B. SYNTAX: OPEN may occur only once in a state table. It is in the output section of a transition expression, and therefore must be preceded by a colon or semi-colon. WRITE also occurs in the output section and therefore also must be preceded by a colon or semi-colon. WRITE must have two arguments. The first argument will be the value to be written in the output file or a variable containing this value. The first argument must be followed by a comma, and then followed by the second argument. The second argument is the code value or a variable containing

the code. If the value ranges between 0 and 15, then the value of the first argument is truncated, so that its maximum is 1,048,575. If the second argument is 16 or greater, then the first argument may be as large as 16,777,215. No more than 61 WRITE statements can occur before the mass-storage device has accepted the data block. The time necessary for the data transfer depends upon the type of mass-storage device.

CLOSE also appears in the output section and must follow a colon or semi-colon. More than one CLOSE is permitted to occur. CLOSE often can occur following the last WRITE of a session, at which time the data transfer from the data buffer to the mass-storage device may still be in progress. For this reason, CLOSE checks to see if the mass-storage device has already received a command to transfer a buffer from this stations. If this is the case, the CLOSE will default to SX. If further attempts to CLOSE are made, the transfer will finally take place when the preceding data transfer has been completed.

C. EXAMPLE: An example of the OPEN, WRITE and CLOSE statements can be found in Figure 30 of Chapter 1 and its associated state table.

3. RECORD

A. DEFINITION. The RECORD statement transfers the time since the last RECORD statement in .01 second steps into a data file. RECORD also requires one OPEN and one or more CLOSE statements, as does the WRITE statement. The first RECORD statement of the session after the OPEN statement, records the time since the OPEN statement occurred.

B. SYNTAX: The RECORD statement is in the output section and must be preceded by a colon or semi-colon. RECORD has one argument specifying the code value. If the argument is between 0 and 15 the accuracy of the time recorded is limited to 10,485.76 seconds. If the argument is 16 or greater, then the full precision time of 167,772.15 seconds will be recorded.

C. EXAMPLE: Figure 29 of Chapter 1 and its associated state tables demonstrates the use of the RECORD statement. See section on data analysis for a discussion of recovering data generated by WRITE and RECORD statements.

SECTION 7: DECISION FUNCTIONS

Decision functions include IF and WITH. These functions have two possible outcomes and can therefore have two possible transfer sections following the decision. A few other functions including FOR, CLOSE, SUBZQO and variable requirements also

resemble decision functions in that they can have two possible outcomes. However, the latter functions have been discussed under different sections, since they have other characteristics.

I. IF

A. DEFINITION

The IF function has three modes of comparison of two variables (or one variable and a number). The three comparison operators are the equal sign (=), the greater than sign (>) and the less than sign (<). In all comparisons the result can be either true or false. The IF function will exit on one branch, which can include outputs and transfer sections, if true, and a second branch with a different set of outputs and transfers if the result of the comparison is false.

B. SYNTAX: Although IF is a decision function it resembles an output function and always occurs in the output section of a transition expression. Therefore, IF is always preceded by a colon or semi-colon. IF always has three arguments and one or more output or transfer sections. Argument one must be a variable or subscripted variable. Argument 2 is the comparison operator, either =, >, or <. Argument three is either a number or variable or subscripted variable. The following are legal IF statements:

```
: IF A<B  
: IF B>A  
: IF B=3
```

The IF function can be used to test the current state of a state set and to make a decision based upon this number. In this case the syntax is identical to the other IF statements except for argument one which identifies the State Set to be evaluated. The State Set is indicated by S. S. N where N is the number of the State Set. An example is:

```
: IF S. S. 2=3
```

which translates to the statement "IF State Set 2 is currently in state 3". The periods following each S are mandatory.

There are three comments that may be used to clarify the operation of IF or other decision functions. These comments are :THEN, :ELSE and :ALWAYS. Since these comments are in the output section they must be preceded by a colon semi-colon, or left bracket.

Each decision function has two possible output and transfer sections, one for the true comparison and one for the

false. A typical IF statement is:

```
R1: IF A=3: THEN: ON1----->S2
      : ELSE: ON2----->S3
```

In the preceding example, when an R1 is emitted the output ON1 and transfer to state 2 will occur if and only if A=3. Otherwise, the next line, written without an input section will occur. If A=3 then ON2 will occur and transition will be to state 3.

An alternative format for the IF statement includes brackets([,]) for separating the true condition from the false condition. Transitions inside the brackets will occur if the IF condition is true. For example, an alternative method for typing the previous example is:

```
S. S. 1,
S1,
R1: IF A=3[THEN: ON1---->S2]: ELSE: ON2---->S3
```

In some instances, the user may want a null transition to occur if the condition of the IF statement is false. In this case, if an :ELSE line is omitted and a new input or a new state occurs on the line following an IF statement, the false transition automatically will default to SX. Thus the following two examples are equivalent:

```
R1: IF A>3: THEN: ON1----->S2
      : ELSE----->SX
S2,
and
R1: IF A>3: THEN: ON1----->S2
S2,
```

In the two preceding examples IF A is greater than 3 then ON1 will occur with transition to S2. If A is 0, 1, 2 or 3, then nothing happens, and transition is to SX. The second version is slightly faster and requires one less state table location than the first, but the first version may be clearer and easier to comprehend because of the explicit ELSE transition.

Sometimes, it is useful for both the false and the true transitions to go to the same state. For example, it may be that IF A<B then you wish to RECORD B, and go to state 3, while if A>B then you wish to go to state 3.

In both cases transition is to the same state, but the RECORD statement is only desired for the true condition. For cases of this sort the true condition can follow the IF statement

inside brackets ([ ]), and then a single transition line can follow. For example:

```
R1: IF A<B [ : THEN: RECORD B]: ALWAYS: ON1-->S3
```

In this example, if $A < B$, then when R1 occurs RECORD B, turn ON 1 and go to state 3. If A is not less than B, turn ON 1 and go to state 3. This version would be equivalent to but requires less state table length than:

```
R1: IF A<B: THEN: RECORD B: ON1-->S3
      : ELSE: ON1---->S3
```

The comments THEN, ELSE and ALWAYS do not require state table space and are highly recommended for clarity.

Sometimes it is necessary for a number of conditions to be true for transition. At the time of this writing there only can be one condition specified in an IF statement. However, IF statements can be strung together with a common false condition which will occur if any of the IF statements is false. For example:

```
R1: IF A>B: THEN: IF D<3: THEN: IF X=W: ON1-->S2
      : ELSE: ON2----->S3
```

In this example, IF $A > B$ and $D < 3$ and $X = W$, then and only then will ON1 occur with transition to S2. If any of the conditions are false then ON2 and will occur with transition to state 3.

C. EXAMPLES: Examples of the IF statement occur in Figures 3, 5, 7, 9, 10, 21, 27 and 32. Examples of IF statements evaluating state sets are to be found in Figures 24, 25 and 26.

11. WITH

A. DEFINITION: SKED contains a random number generator that produces a number between 001 and 1000 on a psuedo-random basis. This number can be used to randomly select a variable or transition by means of the WITH statement.

B. SYNTAX: WITH must always be preceded by a colon or semi-colon. WITH has three arguments. The first must be the letter P, the second must be the equal sign (=), and the third is either a number between 0 and 1000 or a variable containing the number. WITH statements are similar in format to IF statements in that the probability condition will randomly be true or false with the specified probability. There are two possible transitions, one a true condition and one a false condition. WITH statements may need a separate transition for the false condition. For example,

```
R1: WITH P=500: THEN: ON1----->S2
      : ELSE: ON2----->S3
```

In this example, one half of the R1's will produce ON1. The rest will produce ON2. If no transition is required for the false condition, default is to SX. Thus the following examples are equivalent:

```
R1: WITH P=020: ON1----->S2
      : ELSE----->SX
```

```
1"----->S2
```

or

```
R1: WITH P=020: ON1----->S2
```

```
1"----->S2
```

In both cases 2% of the R1's will produce ON1, and 98% of the time the R1 will produce nothing

WITH statements may be strung together as long as their total probabilities do not exceed 1 (that is 1000). For example:

```
R1: WITH P=500----->S1
      : ELSE: WITH P=250----->S2
      : ELSE----->S3
```

In this case R1 will go to S1 50% of the time to State 2 25% of the time and to State 3 25% of the time.

C. EXAMPLES: Example of the WITH statement can be found in Figures 17, 18 and 19 of Chapter 1. Brackets may be used with the WITH statement.

SECTION 8: VARIABLES

A. DEFINITION: Variables can be used as arguments by output functions, decision functions, data acquisition functions, and for counting variable events. The C array has the special property of being printed and stored as a file by means of the appropriate MON commands. All other variables are printed by the X command or modified by the Q command.

All variables begin with a letter from A to Z and may be given a number from 0 to 1023 defining an array. Note that each variable element takes 2 locations. The maximum number of variables possible is much less than 1023 in practice, since the entire state table including variables must not exceed 3072 locations.

Each variable element may contain a number between 0 and 16,777,215. The meaning of this number, of course, depends upon the state table.

Variables may be defined in several ways. First, the variable may be assigned an initial value that may be different from zero by defining a list. Secondly, an array may be reserved by explicitly dimensioning a variable at the beginning of the state table. Thirdly, a statement like A(33) will automatically reserve 33 elements for the A array. The compiler will locate the maximum element in an array that is specified in the state table and then dimension the array to the size of the maximum. However, for statements involving subscripted variables (i.e. A(K)), the compiler cannot determine the maximum value of the inner variable, which will determine the size of the array that is needed. Therefore, the user should specify the size of the array in a dimension statement for the outer variables in a subscripted variable. If a variable letter is used without an index, then the index is assumed to be zero.

B. SYNTAX: To explicitly dimension a variable array a dimension statement should be typed at the beginning of the state table, before the S.S.1. line. For example, to reserve 50 elements for the A array from A0 to A49, type:

```
(CR/LF)
DIM A=50 (CR/LF)
```

The same format will work for the C array but in this instance the word DIM may be omitted.

To assign initial values to the array a list statement can be used, again before the S.S.1. line. The elements of the list may be assigned temporal values in minutes, seconds or 1/100th's of a second. The compiler translates all temporal values into multiples of .01 seconds. Thus, if you set A(0) to 10", the value of A(0) will be 1000, equivalent to 1000 .01 second clock ticks. The format for a list specification is:

```
(CR/LF)
LIST A=1,4,3 (CR/LF)
or
LIST B=1", 3'10", .02" (CR/LF)
```

In the first example A(0) is set to 1, A(1) is set to 4 and A(2) is set to 3. In the second example, A(0) will be 100, A(1) will be 1900 and A(2) will be 2.

Variables may be subscripted, with as many as 10 levels. Parentheses may be used in defining the levels or they may be omitted. Thus, A(0)=A0 or ABA=A(B(A)). However, if parentheses are used for clarity, then the number of left and right parenthesis must match.

The meaning of subscripted variables is a little difficult to comprehend. With a single level of nesting and a number as the inner level, the concept is clearer. Thus, $A(3)$ means the 4th element of the A array, i.e. A_3 . The A array then has at least 4 elements, $A(0)$, $A(1)$, $A(2)$ and $A(3)$. When the inner level of the nest is another variable, i.e. $A(B)$ then the concept is more difficult. Remember the rule that all variables have an index. If none is written, then the index 0 is assumed. Therefore $A(B)=A(B(0))$. This expression is equivalent to saying "find the value of $B(0)$. Use this value for the index of A." for example, if $B(0)$ is 3, then $A(B)$ is equivalent to $A(3)$. Even more difficult is this expression $A(A)$. This will translate to: find the value of $A(0)$. Use this value as the index for A. If $A(0)=5$, then $A(A)=A(5)$. Try $A(A(A))$. If you can figure the result for this expression, then perhaps you will be able to find a use for it. The maximum size of a subscript of a variable is 1023. Any greater number found as a subscript will cause a SKED ERROR to occur.

C. EXAMPLES: Examples of variables are to be found throughout Chapter 1.

SECTION 9: OPERATORS ON VARIABLES

There are 9 operators that modify the contents of a variable. These include: SET, ADD, SUB, SUBZQO, variable input requirements, LIST, RAND, TIME and FOR. Some of these (SUBZQO, variable input requirements and FOR) also incorporate properties of decision functions, in which two possible output and transfer sections can follow the decision function, depending upon the current value of a variable. Variables themselves serve in one (or more) of five capacities.

1. In ON and OFF statements, variable output channels can be selected by the passage of time or the appropriate set of input events.
2. Variables that can be altered by the appropriate contingencies can be used to make decisions about the next set of contingencies as in decision functions.
3. Variables can be used for temporary data storage, either in the C array or in WRITE statements.
4. Variables can be used as the index for another variable, as in subscripted variable arrays.

5. Arithmetic calculations can be performed on variables with the result being used for one or more of the other four functions.

I. The SET Statement

A. DEFINITION: SET is used both as a replacement function and as the function for mathematical computations. In the simplest form, SET can be used to specify a new value for a variable as in the expression SET A=3. The contents of the left hand variable are replaced by the number on the right of the equality sign. Slightly more complex is the case in which the replacement number is the contents of another variable, as in the expression: SET A=B. The latter expression is equivalent to saying replace the current value of A(O) by the contents of B(O). A further degree of complexity (and power) is afforded by subscripted variables in which the index of the variable array is a third variable as in: SET A(B)=D which is equivalent to saying "find the value of B(O) and use this value for selecting the element of the A array; then replace the contents of this element of the A array with the current contents of D(O)".

Limited mathematical operations are also possible with the SET statement. For example, SET A=A+2 which means add 2 to the current value of A(O) and then replace the current contents of A(O) with the new value. Another mathematical operation might be SET A=A*B, which means multiply the current value of A(O) by the contents of B(O) and then put the product into A(O). Subtraction and division are indicated by the minus (-) and the slash (/) respectively as in SET A=D-2 or SET A=D/E.

B. SYNTAX: SET always occurs in an output section of a transition expression and therefore must be preceded by a colon or semi-colon. SET may have 3 or 5 arguments, depending on whether it is a simple replacement function or whether it contains a mathematical operator. The first argument must always be a single variable or a subscripted variable. It may not be a digit. The second argument must be equal sign (=). The third argument may be either a number or a variable (or subscripted variable) in a three argument replacement function. The third argument may be a variable or a number in a 5 argument mathematical operation. If there is a 4th argument it must be one of the 4 basic math operators (+, -, *, /) representing addition, subtraction, multiplication and division, respectively. The 5th argument may be either a variable or a number. Thus, there are two separate formats for SET, one for replacement and one for mathematical operators. SET functions may be listed in a string separated by commas with only one SET printed at the

beginning of the string. Thus the following expression is legal:

```
:SET A=3, B=4*2, D=B/A, E(3)=D:
```

Note, that the first SET in this string is a replacement, the second is a simple multiplication, etc.

In summary, the format of the two types of SET is:

(A) Replacement :SET VARIABLE=DIGIT OR VARIABLE

(B) Math :SET VARIABLE=DIGIT OR VARIABLE-MATH OPERATOR-DIGIT OR VARIABLE

The addition operation can add two 24-bit numbers. A 25-bit result however, will result in a SKED ERROR.

The subtraction operation can subtract two unsigned 24-bit numbers. It assumes that both the third and the fifth argument are positive numbers. If the result of a subtraction is negative, then a SKED ERROR will result. Multiplication of two 24 bit numbers is permitted but a product greater than 16,777,215 will also produce a SKED ERROR. Division of 2 24 bit numbers is permitted, but for a non-zero result the dividend must be equal to or greater than the divisor. Dividing by zero will produce a quotient of zero.

C. EXAMPLES: Examples of SET statements are found in Figures 23, 28, and 33

ii. The ADD Statement

A. DEFINITION: The ADD statement is a shorthand function used to increment by one a recording counter in the C array or other variables. This function is required often, and it actually represents two separate routines. The first routine is designed to increment recording counters specified by numbers. Thus, ADD 5 is equivalent, but much faster than SET C(5)=C(5)+1. ADD followed by a digit always increments the appropriate recording counter.

ADD followed by a variable or subscripted variable is a faster routine than the equivalent SET statement. Thus, ADD C(5) is faster than SET C(5)=C(5)+1 but slower than ADD 5.

D. SYNTAX: ADD always occurs in the output section of a transition function and therefore must be preceded by a colon or semi-colon. ADD is followed by one argument, a number or a variable (or subscripted variable). When followed by a number, ADD assumes the number to be the element of the C array. When followed by a variable, ADD will increment the variable by 1. If

the result of the incrementation is 16,777,216, ADD will cause a SKED ERROR.

C. EXAMPLES: ADD is to be found in Figures 28 and 29 of Chapter 1.

III. SUB

A. DEFINITION: SUB is a shorthand function to decrease the contents of a variable by one. It differs from ADD in that it may not have a digit as an argument. The argument must always be a variable. SUB B is somewhat faster than its equivalent expression SET B=B-1.

B. SYNTAX: SUB always occurs in the output section of a transition expression, and therefore must be preceded by a colon or semi-colon. SUB has one argument which must be a variable (or subscripted variable).

C. EXAMPLES: The SUB function is found in Table 9 of Chapter 3. (Note, the SUB routine is a subroutine!)

iv. SUBZQO

A. DEFINITION: The SUBZQO routine is a complex shorthand function, incorporating both variable manipulation and decision functions. The SUBZQO routine stands for subtract 1 (from a variable) and if the variable is zero, go. SUBZQO B is equivalent to the statement:

```
SET B=B-1: IF B=0: THEN---->
      : ELSE----->SX
```

Thus, the routine first subtracts 1 from a variable and then checks to see if the variable is now zero. If not, transition will default to SX. If the variable is now zero, then the output and transfer statements following the SUBZQO statement will be activated.

B. SYNTAX: SUBZQO always occurs in the output section of a transition expression and therefore is preceded by a colon or semi-colon. SUBZQO has one argument which must be a variable or subscripted variable.

C. EXAMPLE: SUBZQO is particularly useful when an input must produce one output every time it occurs, but produces a second output only after a fixed number of inputs occur. For example if every R1 is to be recorded in counter 1, but only every 5th response is to be reinforced, the SUBZQO routine can be used to advantage as in the following state table:

```
S. S. 1,
S1,
```

```

START: SET V=5----->S2/ INITIALIZE V
S2,
R1: ADD1: SUBZQO V: THEN: SET V=5: ON1---->S3
S3,
3": OFF1----->S2

```

For further examples of SUBZQO see Table 21 of Chapter 3.

V. Variable Input Requirements

A. DEFINITION: Variable requirements are very similar in operation to SUBZQO and in fact use the same subroutine in SKED. However, they serve to redefine an input, requiring a specified number of repetitions of the input before transition occurs. Thus if V is set to 5, V R1 as an input means that transition will only occur with the 5th R1. The statement is equivalent to:

```

R1: SET V=V-1: IF V=0: THEN----->
: ELSE----->SX

```

It is also equivalent to:

```

R1: SUBZQO V----->

```

B. SYNTAX: Variable response, time and Z requirements occur in the input section of a transition expression and therefore are preceded by CR/LF, CR/LF tab, or CR/LF space(s). There must be a space between the variable (or subscripted variable) and the T, R or Z that follows it. Remember to reset the value of the variable after transition, since it will have been set to 0 by the process of being decremented.

C. EXAMPLES: The following example is functionally equivalent to the example for the SUBZQO statement:

```

S. S. 1,
S1,
START: SET V=5----->S2
S2,
R1: ADD 1----->SX
V R1: SET V=5: ON1----->S3
S3,
3": OFF1----->S2

```

VI LIST

A. DEFINITION: The LIST function selects items sequentially from a list defined as described in section on dimensioning lists. The LIST function is both a replacement and decision function. It will select items sequentially until it has taken the last item from the list. Then, the next item selected will be the first item of the list. In other words, the LIST function recycles, using elements of the list sequentially and then

repeating. LIST uses a variable index to point to the LIST item to be selected next. The variable index may be modified by separate SET or other variable operators.

B. SYNTAX: LIST always occurs in the output section of a transition expression, and therefore must be preceded by a colon or semi-colon. LIST has four arguments. The first argument must be a variable (or subscripted variable) which is to be the recipient of the next element of the variable list array. The next argument must be an equal sign (=). The third argument is the letter variable of the pre-specified array of list values. The fourth argument is the index variable, used by LIST to point to the next element of the array to be selected.

C. EXAMPLE: LIST A of 10 variables has been established, and the list contains successive variable intervals to be used in a variable time requirement B, and if the index variable is K, then the following state table will accomplish this set of operations. LIST A=1", 5", 2", 9", 8", 3", 4", 6", 10", 7"

```
S. S. 1,
S1,
    START: LIST B=A(K)----->S2
S2,
    B T: LIST B=A(K): ON1----->S3
S3,
    1": OFF1----->S2
```

The LIST function of the above state table is equivalent to:
LIST A=1", 5", 2", 9", 8", 3", 4", 6", 10", 7"

```
S. S. 1,
S1,
    START: SET B=A(K)----->S2
S2,
    B T: IF K 10: THEN: SET K=K+1; SET B=A(K): ON1---->S3
        : ELSE SET K=0, B=A(K): ON1----->S3
S3,
    1": OFF1----->S2
```

VII. RAND

A. DEFINITION: RAND is a function used to select a value from a list and store this value in a variable. The selection process is random, without replacement until all list values have been selected once. When all list values have been selected once, the function recycles, and will again select items from the list. The major consideration in using RAND is that two separate RAND calls for the same list and variable will not be synchronized, and therefore will not operate without replacement.

B. SYNTAX: RAND is always located in the output section of a transition expression and therefore must be preceded by a colon or semi-colon. RAND has three arguments. The first argument is the target variable, that is, the variable which will be set to a

new value sel from the list. The target variable may be subscripted. second argument is an equal sign. The third argument is a \$letter which must refer to a list (or array) of prespecified, established by a dimension statement coming before \$

:RAND (TARGET) ILE=(LIST) VARIABLE

C. EXAMPLES: Uses of the use of RAND are to be found in Figures 30 and Chapter 1 and accompanying state tables and text.

VIII. TIME

A. DEFINITION: Time of day is requested on entry into time-share in as and hours. After this point, a clock has been established and is automatically incremented every minute until midnight(00) at which time the clock is reset to 0:0. The time of day is examined by the MON command V or modified by the MON C W. The TIME function in SKED permits transferring the time-of-day into 3 variables. The first variable will be the decimal hour and minute value, the second variable and the third variable, minutes.

B. SYNTAX: TIME in the output section of a transition section and the preceded by colon or semi-colon. TIME has one argument which be a variable or subscripted variable, but TIME will also set the next two variables in the array of the first variable. if the current time is 13:05 :TIME A will set A(0)=1305, A(1)=13, and A(2)=5. Also :TIME V(3) will set V(3)=1305, V(4)=V(5)=5.

C. EXAMPLE: Use of TIME is found in Figure 32 and accompanying text.

IX. FOR, NEXT

A. DEFINITION: FOR loop function and auxiliary NEXT and BREAK are used for repetitive operations on the same clock tick. On each repetition the variable is incremented from an initial value to a maximum. This variable is used as an index for manipulating an array of variables. Watch out! FOR loops take time.

B. SYNTAX: FOR loops located in the output section of a transition section and therefore must be preceded by a colon or semi-colon. 5 arguments. The first argument (or index variable) is a variable or subscripted variable that will be incremented each operation of the loop. The second argument must be a sign (=). The third argument (either a variable or number) is the initial value of the index variable on first operation of the loop. The fourth argument is a number or variable specifying the size of the increment for each loop. The

fifth argument is a number or variable that contains the maximum value of the index variable. This maximum value will be the final value of the index variable. When the index variable is greater than the maximum value, the loop will be terminated. Thus:

```
:FOR (INDEX) VARIABLE=INITIAL VALUE, INCREMENT, FINAL VALUE
```

The loop will activate all functions following the FOR statement until the statement NEXT, or BREAK or the next transition arrow, whichever is encountered first.

The function :NEXT which must always be preceded by a colon or semi-colon, will increment the index variable of the FOR loop and repeat the loop until the final value is exceeded.

The function :BREAK must also be preceded by a colon or semi-colon. This function will automatically terminate a FOR loop. It is usually used in conjunction with an IF statement.

C. EXAMPLE: See Figure 33 for examples of FOR, NEXT and BREAK functions.

SECTION 10: ANALOG INPUTS AND CALLO

A. DEFINITION: Analog events may be obtained by operating an analog-to-digital (AD) converter connected to the computer. Since the analog process is continuous, it is necessary to establish a rule for when to sample and measure the analog signal. This requires operating a special subroutine to convert the analog signal to a digital event, and to store the resulting digital number in a variable. When the digital representation of the analog signal is stored as the contents of a variable, the number may be compared to a fixed criterion for generating contingent transitions, or it may simply be recorded. The function that initiates the A to D conversion is CALL O. The CALL functions, which are explained in a later section, pass arguments from the state table to special purpose, machine language subroutines. CALL O requires the specification of a variable that will receive the digitized number, and the specification of the Octal IOT pulse that will operate the A to D converter.

B. SYNTAX: CALL O is both an input and an output function. Logically, it is an input function because it provides data from the experimental subject. In fact, it must be an output function since it must be triggered by an input. Therefore, CALL O must always be preceded by an input (R, Z, T or START) function, and separated from the input by a colon, or semi-colon.

CALL 0 must indicate the variable that is to receive the digitized data, and it must transmit the appropriate Octal IOT to the A-D converter. Arguments of CALL functions are enclosed in brackets, and Octal numbers are preceded by a ?. The CALL 0 functions, therefore would be followed by Variable, ?IOT to pass the two arguments to the subroutine.

C. EXAMPLE: In the following example, CALL 0 is used every 10 seconds to digitize an analog signal, to store the result in variable A, and to write the result with code 0 in an input file. The mythical IOT 6355 operates the A-D converter.

S. S. 1,

S1,

START: OPEN----->S2

S2,

10": CALL 0 [A, ?6355]; WRITE A, 0----->SX

CHAPTER 5

THE SKED MONITOR

THE SKED MONITOR (version 14)

Experiments can be started, stopped, changed, and data can be examined in a variety of ways. This section considers direct control of on-going experiments by means of a background program called MON.SV. Some of the functions can be produced directly with CCL commands, but these will be considered in the next section. This section first describes the MON keyboard commands in terms of function. The function of each keyboard command is summarized at the end of this chapter.

The MON program is special in that it must be run under Time-Share. It is an OS/8 program that can only run in the background of the Time-Share SKED system. However, when run, MON can recognize whether it is in stand-alone OS/8 or in the background. If it discovers that it is in stand-alone OS/8, it will then cause the Time-Sharing SKED system (SHARE.SV) to run, if it exists on the system device. In this case the message: "Congratulations Time-Share SKED is up" will be printed along with a request for the time of day. After this information is entered by the user, MON will now be running as indicated by its prompt character %. If MON was loaded in the background it will omit the initial request for time of day.

MON can be called directly by the CCL command MON, or can be called by the command R MON. Execute, LOAD, COMPILE and DATE commands will also call MON under some circumstances (see CCL commands in the next chapter).

1. LOADING STATE TABLES

Once a state table has been successfully edited and compiled, then the binary (.ST) or merged (.SM) version must be loaded into memory from the mass-storage device. One way to load the state table is by means of the SKED monitor program MON. When MON is loaded and running as indicated by the %, the command "I" will cause MON to print the prompt:

SPECIFY INPUT FILE

followed by an asterisk (*) which is a request for the user to type the device containing the desired file followed by the file name and extension plus any desired options. For example, assume the experimenter has created a state table called TEST.SK and successfully compiled the program. The output of the compiler was named TEST.ST (see section on compiling) and this file was stored on the system device SYS (see section on mass storage device names). The following reply will start the loading process:

*SYS: TEST. ST (C.R.)

after the carriage return (C.R.) MON will state:

LOAD BOXES:

If it finds the requested file on the requested device. If it cannot find the requested file name on the requested device, MON will reply

*FILE NOT FOUND (C.R.)

*

Assuming the file exists on the appropriate device, the user is now expected to type the number(s) of the station(s) into which the state tables should be loaded. If the user wanted to load TEST. ST into boxes 0, 3, and 11, he would type 0, 3, 11 (C.R.) to specify these stations. MON will now attempt to load the state tables into the desired stations. If successful, MON will print:

NO IDENTIFIERS. LOAD OK

ACTIVE: 0, 3, 11

This message means that a .ST rather than a .SM file has been loaded and that it is now running in boxes 0, 3, and 11. If a .SM file had been requested with an identifier section, then the message NO IDENTIFIERS would be omitted.

A number of possible errors can occur on the loading process. The most common of these are:

1. STATION ALREADY RUNNING. This message will occur if any of the stations requested by the user has a state table still running. The state table will not be loaded into any of the requested stations. The user should terminate the active stations with the ABORT or KILL commands, before attempting to reload.

2. STATION NOT SPECIFIED IN DIALOGUE. One or more of the requested stations was not specified in the initial dialogue (see section on "Getting on the Air"). It is not possible to load a station not requested in the dialogue. The SKED system will have to be re-assembled and reloaded for this to be possible. No stations (even those specified in the dialogue) will be loaded with this error condition. However, by retyping "I", the user may load the stations specified in the dialogue.

3. STATE TABLE TOO LONG FOR REMAINING SPACE IN SKED. This message means that the total amount of space required for loading the state table (summed across 3 stations in the example) is

greater than the amount of space available. Perhaps 1 or 2 stations can be loaded in the available space.

4. CHECKSUM ERROR ON .SM FILE. This error report usually occurs when an error in the state table was not detected by the compiler. However, in some cases a reading error can occur in the hardware controlling the mass-storage device. The user should try to recompile and reload the state table. If the error persists, the user should examine the state tables closely for syntax errors, which can be corrected. If the error cannot be detected, contact your local SKED dealer.

DEFAULT specifications:

A. DEVICE TYPE

Most OS/8 programs will assume the user meant the device DSK if no device is specified in the command request. Thus the command:

*TEST.SK

is equivalent to the command:

*DSK:TEST.SK

If the output of the compiler is stored on the mass-storage unit DSK, then the storage device does not need to be specified. This is most convenient because the default device name for OSCOMP, IDENT and MERGE, is also DSK. (See section on compiling state tables.)

B. FILE EXTENSION

If no extension is specified to MON it will assume the extension .SM for merged state table. Thus the command request *TEST is equivalent to *DSK:TEST.SM

C. OPTIONS

The experimenter can bypass the request for specifying station numbers for the load request by including these numbers in the initial file specification inside parentheses. Thus the command request: *TEST (013) will load TEST.SM into stations 0, 1 and 3 without further specification required.

Two additional options may be requested along with the station numbers, when the input file is specified. If automatic state table data collection is planned, the M option must be used to request output files. Slash M (/M) will store the station numbers of the load request. Later, when M is typed, the user need not specify stations, since this information is already

available to MON. Of course, if MON is exited after the load and before the M command is typed, then when MON is recalled, it will have no recollection of the appropriate station numbers. The station numbers will then have to be entered after M is typed. The syntax of this option would be:

```
*TEST (013)/M (C.R.).
```

The second option that may be specified along with /M is /C. This option sets a protection mode switch in MON. Further commands after the switch is set will warn the next person who types a MON command to check with the user before proceeding. This caution message is often useful when one experimenter must leave the computer room, and there is some possibility that another user may interfere with an experiment in progress. The syntax of this command is:

```
*TEST (013)/M/C (C.R.)
```

2. AUTOMATIC DATA ACQUISITION

If the state table contains OPEN, WRITE, RECORD, and CLOSE commands to generate automatic data collection, a file must be reserved on a mass-storage device for each station collecting data. The MON command M begins this process by requesting the user to specify the stations for which data is to be collected. When the user types M, MON will print:

```
% SPECIFY BOXES
```

and the user will type

```
0,1,3 (C.R.)
```

specifying a file request for three stations. (This step is bypassed with the /M option). Next, if these stations are loaded, MON will prompt:

```
*OUTFIL.NN [MM]<
```

to remind the user to enter the name and length of an output file for this station. The user must specify a length for this file in terms of the number of data blocks to be collected. Each block contains 124 double-precision counters plus the initial 8 identifiers.

Remember that two blocks are required to collect one block worth of valid information. If the user expected 4000 "record" events to occur in a session, he/she would need $4000/62=65$ blocks of room in the output file. Assuming the user wanted to collect the data from station 0 in FILE0 on the system device and that 65 blocks were to be reserved, then the statement

*SYS:FILE0 [65]< (C.R.)

would be typed. The < is required to specify that we are creating an output file. After this command is entered, a brief period will elapse during which the mass-storage device writes zero in the first location of each of the 65 blocks. The maximum file length that can be specified is 255 blocks.

The request for file specification will be made for each station specified in the M command (or by the /M option).

3. START, GO AND RESPONSE SIMULATION

There are two different methods for starting experiments. The first of these is to start at the experimenter's convenience. This permits the user to load the state tables and then to open output files, to set the values of variables and to put the subject into the experimental chamber. At this point the user would start the session by means of commands at the terminal keyboard.

In the second method the state table starts at a particular time of day, regardless of the time at which the state table was loaded into memory. Essentially the first method involves manual intervention to start the session, while the second method is automatic, once the state table has been successfully loaded.

The method of starting the session manually involves simulating a response input from the terminal. The simulation routine permits entering responses of any magnitude (not necessarily defined in the dialogue). Thus, it is possible to enter R10000 from the terminal to a station which only has 3 responses defined by the dialogue. The experimenter, then, specifies that a special response starts the session. One response type, R12, is generated for all stations by typing G in MON. The state table then has an R12 as the event that initiates the session. There is one danger incumbent in this method of starting the session. G generates R12 for all active stations. If a station uses R12 for some purpose other than starting, it will receive an R12 whenever G is pressed in MON, for starting a session in a different station. Reserve R12 for start only!

A slightly safer method is provided by typing S in MON. S generates R12 but also requires the user to specify the stations to receive the R1200. The compiler also translates the word START as being equivalent to R1200.

The experimenter also may choose any response value to start the experiment, merely by including this value as the input that turns on the session stimuli. The specified response input can

then be obtained either by operating a valid input or response channel defined in the dialogue, or alternatively by simulating a response on the terminal. Typing the letter R in MON will be recognized as a request to simulate a response. MON will then request the response number and the stations to which the response should be sent.

4. MODIFICATION of VARIABLES

Variables can be modified at any time after a state table is loaded and before it has stopped. Modification of variables permits the experimenter to write one state table for a parametric study. For example, to program a Fixed Ratio experiment in which the ratio value is to be varied during the course of an experiment, would merely require the user to set the ratio value into a variable at the beginning of the session.

To modify a variable, type Q in MON. The request to specify one station will follow, and MON will request the variable letter and the numerical index of the variable. When these quantities have been entered, MON will print the current contents of the variable followed by a "/", indicating that a new value may be assigned to the specified variable. If no new value is required, type "return" to leave the variable intact. For example, to set the variable A3 to 5 in box 3, the following sequence would be used:

% Q

MODIFY 1 VARIABLE

SPECIFY BOXES 3 (C.R.)

SPECIFY VARIABLE LETTER A

SPECIFY VARIABLE NUMBER 3 (C.R.)

A=7/5 (C.R.) (original contents are 7)

5. LISTING THE CURRENT CONTENTS OF ALL VARIABLES

MON will list the current contents of all variables (except for the C array) for a specific station. Assuming variables A0 to A3 and B0 to B6 in box 3 the command X will cause MON to print as follows:

% X

LIST VARIABLES

SPECIFY BOXES 3 (C.R.)

VARIABLE	0	1	2	3	4
A	4	2	70	5	
B	14	3	22	6	7
	10	9			

%

6. PRINTING THE COUNTERS

The C array is not included in the X printout of the variables. The printing routine in MON for counters permits the user to print a subset of the counters or the entire counter array. The P command can be used to print counters 0 to 5 (assuming at least 5 counters were reserved) as follows:

% P

SPECIFY BOXES 3 (C.R.)

1ST CTR 0 (C.R.)
 LAST CTR 5 (C.R.)
       3 4 2 1 6
       2

7. SAVING THE COUNTERS AS A NAMED FILE

At any time after a state table has been loaded the current contents of the C array may be stored as a named output file on a mass-storage device. Usually this is done at the end of the session, with automatic recording methods used for data acquisition during the session. The MON command O is used for this procedure as in the following example:

% O

SPECIFY BOXES FOR DUMP

0, 3 (C.R.)

SPECIFY OUTPUT FILE

*SYS:FILE03< (C.R.)

%

At this point, MON has created FILE03.DA on the system device and has transferred the identifiers and the C array from stations 0 and 3 to this file. The default extension is .DA. It can be set to any desired extension. The default device is DSK:.

8. ENTERING IDENTIFIERS WITH MON

Output data files created automatically or created by the O command both contain a set of 8 identifying numbers at the beginning of the file. The data produced by the automatic methods with the M command repeats the identifiers on every block of 124 counters. Each station has its own set of identifiers for the O command. The eight numbers are:

1. The number of counters in the file.
2. The station or box number for the following data.
3. The month.
4. The day.
5. The year.
6. The experiment number.
7. The subject number.
8. The group number.

Of course, counters 6, 7 and 8 are only arbitrarily defined as experiment, subject, and group numbers. The user may redefine these for her own purposes. These identifiers must be numeric and no larger than 4095.

The last 3 numbers are usually created in the compiling process (see section on compiling) but they may be modified by the user with the help of MON. The Z command will produce the following dialogue to modify the identifiers in boxes 0 and 3:

% Z

SPECIFY BOXES 0, 3 (C.R.)

EXP: 2 (C.R.)

SUB: 1 (C.R.)

GROUP: 4 (C.R.)

EXP: 2 (C.R.)

SUB: 2 (C.R.)

GROUP: 4 (C.R.)

%

At this point, box 0 has identifiers specifying experiment 2, subject 1 and group 4. Box 3 has experiment 2, subject 2 and group 4. Of course, it is necessary to load the state tables before changing the identifiers since the loading of the state table will either set the identifiers to those merged with the state table or will set them to zero, if none had been specified during the compilation process.

9. TYPING AND MODIFYING TIME OF DAY

On entering the time share mode, MON will request the time of day. For example:

CONGRATULATIONS! TIME SHARE SKED IS UP

ENTER TIME OF DAY:

HR: 15

MIN: 32

%

Sometimes the user may wish to change the time of day. The W command will permit this backward move in time.

To obtain the current time of day, type V in MON, or type DA in OS/8 monitor.

10. SETTING THE SAFETY SWITCH

Since TIME-SHARE SKED may be shared to run several experiments simultaneously by several users, it is sometimes necessary to warn others not to use MON at critical times (especially when BATCH is used). There is a special option that can be used when loading state tables to set a switch that warns the user if he tries to use critical MON commands. The switch can be set by adding /C to the input file specification. Thereafter, MON will print:

CAUTION!!! GET OK FROM USER BEFORE PROCEEDING!

TO CONFIRM TYPE Y

before any critical MON command can be used. Typing / on the terminal will set this switch at any time. You must exit MON.sv and reenter to turn off the caution switch.

11. TERMINATING BATCH and STORING BATCH POINTERS

When BATCH is used to control loading state tables, the BATCH job can be aborted within MON by typing U. The U command

also saves the current pointers of batch so that batch may be restarted at some other time using the F command.

12. PRINTING ACTIVE STATIONS

The command B will print the numbers of the stations that have state tables running.

13. ABORTING ONE OR MORE ACTIVE STATIONS

The A command will immediately stop one specified station, turning off all of its outputs and ending the session. The station cannot be restarted except by reloading the state table. However, only until any new station is loaded, it is possible to print or use the O command to store the C array from the station.

The K command, which always asks for confirmation, can be used to abort all 12 stations. LOOK OUT! See getting on the air to enable!

14. CHANGING YOUR MIND IN MON

Any print out in MON can be stopped, eventually, by typing CTRL O. If you begin another command and change your mind or hit the wrong key, typing CTRL O will abort the command.

15. BATCH CONTROL OF EXPERIMENTS USING MON

MON has been designed to facilitate running experiments under BATCH. Experiments can be loaded, output files established, variables modified, experiments started, and data analysis completed all under control of a BATCH file. Although BATCH control files are somewhat confusing to produce, once they are created and debugged, the BATCH file can accomplish the otherwise tedious job normally done by typing the same commands every time the experiment is to be run. BATCH can thus eliminate frequent typographic errors.

BATCH will load experiments into MON with the LOAD FILE.EX(0123...Z) command to CCL and thus start MON with the desired stations active. The addition of /B to the load request (e.g. LOAD FILE.EX (0123)/B) will request MON to read TTY commands from the BATCH file, located immediately after the load command. All TTY commands except the / option that sets the confirm switch are legal.

Each MON command should be preceded by a % from the BATCH file. Since MON has been constructed so that single letter TTY commands do not require a carriage return or any other terminator, commands without arguments such as B,G or C are simply written:

%B

%G

%C

MON commands with numerical arguments do require a carriage return to terminate the numerical argument or arguments. For example, to ABORT a station requires A without a carriage return followed by the Box No. which must be specified, and terminated by a carriage return. For formatting purposes, carriage returns, spaces, tabs, (,), and line-feeds are ignored in the BATCH file. The character ! is used to terminate numerical arguments in BATCH. Thus to ABORT station 3 from BATCH, after LOADING FILE1 in stations 0,1,2 and 3 you would use the following batch job.

\$JOB

.LOAD FILE1 (0123)/B

%A   %3 !

%C

\$END

There are three ways to end the BATCH stream of commands. A \$ which will leave you in MON and allow typing of keyboard commands. In this case typing C or CTRL C will allow continuation of the input batch stream provided the next command is an OS/8 command. A C which will cause MON to go to the OS/8 monitor and therefore read the next command in the BATCH stream (which must be a command preceded by a "."), and third the "U" command which will terminate batch.

ADVANCED BATCH

Two methods have been added to permit experiments to force MON to return to the OS/8 monitor and therefore read new instructions from the BATCH file.

A. The first method will cause MON to go to OS/8 when a specified set of stations stop. This method is useful for LOADING new experiments automatically after other experiments are completed. To use this method you must specify the stations by means of the D command which will request the specific stations.

Of course the D command may be included in the BATCH job. For example, to run EXP1 in boxes 0, 1 and 2, and then to load EXP2 in the same stations when all of the subjects of EXP1 have finished, use the following BATCH job:

\$JOB TO RUN EXP1 AND THEN EXP2

.LOAD EXP1(012)/B

%B

/THIS WILL PRINT ACTIVE STATIONS ON THE BATCH LOG

%M 0, 1, 2!

*FILE0[10]<

*FILE1[10]<

*FILE2[10]<

/THIS CREATES OUTPUT FILE FOR THE 3 STATIONS.

%D 0, 1, 2!

/THIS SPECIFIES RETURN TO THE MONITOR AT THE END OF THE SESSION

%S 0, 1, 2!

/THIS STARTS THE SESSION

\$/THIS ENDS THE BATCH INPUT TO MON AND WAITS FOR 0, 1, 2 TO END.

.PIPI RXA1:<SYS:FILE0,FILE1,FILE2/A

/THIS BACKS UP THE DATA FILES

.LOAD EXP2 (012)/B

/THIS LOADS THE SECOND STATE TABLE AND CALLS MON

%B

%M 0, 1, 2!

*FILE0[20]<

*FILE1[20]<

*FILE2[20]<

%D 0, 1, 2!

%S 0, 1, 2!

\$SKED INPUT END

/THIS ENDS INPUT FROM BATCH TO SKED MONITOR.

.PIPI RXA1:<SYS:FILE0,FILE1,FILE2/A

\$END BATCH

B. The second method for state table control of the BATCH stream uses CALL 2 in the state table and the H command in MON to specify the appropriate stations. This method would permit BATCH to load and start an experiment in one set of stations, and then later load and start an experiment at a specified time of night.

CALL 2 requires one argument. If CALL 2 [0] is included in the state table, then only one bit corresponding to the calling station is set in the switch referenced by MON. If the argument is non-zero then all bits of the switch are set, so that MON will proceed to BATCH regardless of the stations specified in the H command in the latter case.

Assuming that the state table of EXP1 contains a CALL 2 generated at Midnight, then the following BATCH job could be used to load EXP2 in another station and start it at this time:

\$JOB TO LOAD AND START EXP2 AT MIDNIGHT

.LOAD EXP1(012)/B

%M 0, 1, 2!

*FILE0[20]C

*FILE1[5]C

*FILE2[100]C

%S 0, 1, 2!

%H 0, 1, 2!

/THE PROGRAM STAYS HERE IN MON UNTIL MIDNIGHT

\$WAIT HERE

.LOAD EXP2(345)

%M

%3, 4, 5!

*FILE3[10]<

*FILE4[10]<

*FILE5[10]<

%S   3, 4, 5!

\$SKED INPUT END

\$END BATCH

16. STOPPING AND RESTARTING BATCH

BATCH jobs can conveniently control the flow of experiments. However, it is sometimes necessary to interrupt an ongoing BATCH job that is waiting in MON for an experiment to end before continuing. For example if one user has loaded an experiment that waits for a CALL 2 to occur, before processing data, but another user needs to use OS/8 for a few minutes, then it would be convenient to stop the BATCH job and use OS/8 before resuming the batch job in MON at the place it previously was waiting. The U command aborts BATCH and saves the necessary information for restarting BATCH so the job can be resumed by means of the F command in MON.

Do not type F after a system has been squished following the U command, since the BATCH file may now be in a different place. Use this option with extreme care.

17. EXITING MON

Since MON is a background OS/8 program, the most common way to exit is to return to the OS/8 monitor. Typing either C or CTRL C will result in the print out:

OS/8 MONITOR

At which time you may type any legal OS/8 commands. The stations will still be running under state table control.

To leave the time-shared mode and return to stand-alone OS/8

Type E in MON. MON will print a request to confirm by typing a space. It also will not let you exit if stations are still running. You will have to end the session before returning to stand-alone OS/8.

18. RECOVERING BACKGROUND OS/8

Some programs will not return to the OS/8 monitor when CTRL C is typed in the background due to errors. Also it is possible to enter into endless loops in some data analysis languages. If this occurs in stand-alone OS/8 one might stop the computer and boot up the system. In the background, of course, this may be impossible if experiments are currently running. For the purpose of recovering the OS/8 monitor the CTRL B command has been added to the system. Typing CTRL B in the background will reboot the OS/8 system. Remember to check the date and the assignment of devices after typing CTRL B.

Never type CTRL B during a COPY or squish command or during any program that will modify the directory.

SUMMARY OF MON COMMANDSCOMMAND FUNCTION

A	ABORT ONE STATION
B	PRINT ACTIVE STATIONS
C	RETURN TO BACKGROUND OS/8
D	ADVANCE BATCH ON END OF SESSION
E	EXIT TO STAND ALONE OS/8
F	FINISH BATCH
G	R12 TO ALL STATIONS
H	ADVANCE BATCH ON CALL 2 FOR BOXES SPECIFIED
I	LOAD SPECIFIED STATE TABLE
J	UNDEFINED
K	KILL ALL STATIONS
L	UNDEFINED
M	CREATE OUTPUT FILES FOR WRITE AND RECORD
N	UNDEFINED
O	OPEN AND DUMP COUNTERS FROM SPECIFIED STATIONS
P	PRINT COUNTERS
Q	MODIFY ONE VARIABLE
R	SIMULATED RESPONSE
S	R1200 TO SPECIFIED STATION (START)
T	UNDEFINED
U	UNBATCH AND SAVE POSITION IN BATCH
V	PRINT TIME OF DAY
W	CHANGE TIME OF DAY
X	PRINT ALL VARIABLES
Y	UNDEFINED
Z	CHANGE IDENTIFIERS (EXP, SUB, GROUP)
CTRL C	GO TO BACKGROUND OS/8
/	SET SAFETY SWITCH

CHAPTER 6

OPERATING SYSTEM

This section presents a general overview of the OS/8 operating system. For a more complete discussion of the intricacies of OS/8 see the OS/8 Handbook, Digital Equipment Corp.

Historically paper tape served as the permanent storage media for both programs and data files. For many purposes, paper tape is still quite adequate for SKED laboratories. However, a paper-tape punch and reader tends to be quite slow, and fast versions of these devices are quite expensive. If the experimenter wishes to conduct micro-analysis of his/her data, then paper tape may be much too slow and cumbersome for this purpose.

The development of relatively inexpensive and fast external storage devices including magnetic tape, and flexible disks has made it possible to acquire and analyse large quantities of data in a convenient format. Furthermore, it has led to the development of time-sharing in which case the computer can be used simultaneously for conducting experiments (with automatic data collection), and for data analysis or program preparation.

The efficient use of a fast external storage media, however, requires a number of programs, which organize a library of data and program files and which simplify the use of the computer. The whole set of programs must be organized according to a protocol which enables programs to communicate with each other in a standard manner. For example, in the SKED system, five separate programs are used to compile, identify, merge and load state tables, after they have been created by an editor. The five programs are too big to occupy internal memory of the computer at the same time. Each program operates on input files or data from the terminal and produces an output file for use by the next program. More importantly, each program needs to communicate with the next program in sequence to pass along the system name of the state table used for the various files. Only the extension of this name varies among the programs.

The following example illustrates the need for a method of passing input and output information from one program to the next. Furthermore it illustrates an important point about all computer programs. Programs translate input information into output information. The Editor translates inputs from the terminal to an output file, stored on the mass-storage device. The compiler takes the output of the editor as its input and it produces an output file in a different code. The run-time system uses the output of the compiler as a set of instructions for translating inputs from behavioral subjects into stimulus outputs and into data files stored on the mass-storage device.

Thus, each program needs to read one or more input files stored on the mass-storage device and needs to create new output files on the device. In computer programming, identification of

a common function needed by many programs, immediately generates the notion of one common set of instructions (or subroutine) that can fulfill the needs of all these different programs. The subroutine can be developed and written only once (and in less than 3 years DEC has been able to repair most of the "bugs" in these routines). The code for the routine then takes a limited amount of room on the storage device and it may be called by any program that needs it.

One of the common functions is the set of instructions needed to translate terminal commands concerning input files into the code necessary to find the file on the mass-storage device (or to discover that it is missing) and to specify the device on which to look for this file. The actual operations of searching the file directory and obtaining the correct address of a file then can be contained in one subroutine, if the subroutine passes this information to other programs in a standard manner. Similarly, to create an output file, one must enter the name and location into the directory, and the length of the file must either be prespecified or added into the directory after the output file has been written onto the storage device.

Even more importantly the set of instructions used to read and write on the mass storage devices is used by all programs that are part of the system. Even the code used to direct communication with the terminal can be standardized. However, before the scheme used to organize these subroutines is discussed further, we should note that there is one large source of variability between PDP8 installations that must be considered, and that is the large number of different storage devices available for running OS/8. Some users have one of two types of DECtape. Others have flexible disks. Still others have one of at least three different types of cartridge disk systems. Still others have fixed disk systems. Finally, many have a combination of devices, including one or more disks and one or more tapes. In addition, more than one kind of terminal is in common use. Some users have teletypes, others DECwriters, others "glass teletypes", etc. Also, some lucky users have line-printers and/or graphic plotters as extra peripherals.

All of these devices require several pages (up to 256 locations) to read or write information between them and computer memory. It is necessary to provide for use of subroutines for some set of these devices, so that room must be arranged for this code on the storage devices. To provide for "device independence" so that programs written for OS/8 can be used with any conceivable combination of these devices, a very simple scheme was developed, which can be comprehended by reading the OS/8 manual 10 or 20 times.

Briefly, the scheme depends on keeping one set of code in the computer at all times. This set of codes includes commands to read from or write to one device, called the system device.

There is a method for assigning your device as the system device which will be reviewed later. In any case, even if you have 20 devices associated with your computer, the only set of instructions that will be in the computer at all times is the code to read and write the system device. This permanently resident program (which is located in the top 128 locations of Fields 0 and 1 in stand-alone OS/8) also contains commands to load a program called the keyboard monitor (KBM) from your system device and to then run this program. If you start your computer at location 07605 in field 0, the KBM will then run, as is indicated by a . on your terminal. (Starting at 07600 in field 0, first stores the code located in 0 to 2000 of field 0 in a special storage area of the system device. This permits the ODT commands of the KBM to examine, modify and save this portion of memory despite the fact that the KBM runs in this area of memory.)

The KBM is used to run system programs, and may be used to accomplish many successive operations in one command by the use of another program called CCL. (See Section K of this chapter and the OS/8 handbook on KBM commands.)

The KBM can be used to load a program from the system device, or from any other standard storage device that may be attached to the computer. If it is asked to load a program from another device, it must load the code needed to read from that device into the computer. This code, called device handler code, is stored in a special area of the system device. It is not always in the computer. It will be loaded only when it is needed. Only the handler for the system device is always resident in main memory.

BUILD: OS/8 assigns a finite amount of room on mass storage device for the device handlers needed for any single system. There are more than 32 device handlers listed by DEC and there is only room for 15 device handlers on the system device. However, no individual user will need more than a few of these device handlers for his system. A program has been provided to select the appropriate device handlers from the larger set of potential handlers and to store the necessary handlers in the space reserved for this purpose by OS/8. The program BUILD.SV has commands for loading, inserting, deleting and unloading specified device handlers. Before reading about the specific details of BUILD in Chapter 2 of the OS/8 handbook, the user should be aware of several concepts. First if a device handler is active in BUILD (as indicated by an asterisk * by the print command) this does not mean that the handler is correctly stored on your system device. The only way in which you can be positive that your system has the handlers indicated in BUILD is to use the BUILD command BOOT. The printout of active handlers only tells you what will be active after you issue the BOOT command.

The second concept concerns the specification of the DSK device. Since it is possible to have several different mass-storage devices, OS/8 permits the user to assign a special name to each device. The name SYS is reserved for the system device (which can also be called DTAO, RKA0, etc.). You always have a device called SYS. This is important since several save format (.SV) programs are expected by OS/8 programs to reside on your system device, including the SKED system programs, FORTRAN and BASIC.

Another device common to all OS/8 installations is called DSK. In some cases with only a single mass-storage device, DSK=SYS. The same device is called by two different names. However, if the user has two or more devices, then it is often convenient to assign a non-SYS device the name DSK. The name DSK is the default specification for data files for a large number of OS/8 programs. For example, in creating a SKED state table, it is necessary to use the EDITOR or TECO to open an output file with the name of the desired state table. The complete specification of the field must include the name of the device on which the file is to be stored. If no device is specified, OS/8 will assume the name DSK for that device. The following list of programs assume DSK for the default device:

1. EDITOR
2. TECO
3. OSCOMP (the SKED compiler)
4. IDENT
5. MERGE
6. PIPI
7. TRANIA
8. PAL8
9. CREF
10. DIRECT

In any case, there are two methods for assigning a name to a device. SYS must be specified in BUILD. DSK may be specified in BUILD to be equivalent to SYS or to some other device. However, by use of the ASSIGN command, the name of the device to be called DSK can be altered by the user (see ASSIGN in OS/8 handbook). Furthermore, DSK can be assigned for the time-share background system in a parameter file when the system is built (see Chapter 7).

OVERVIEW OF THE DEC OS/8 SOFTWARE

OS/8 (VERSION 3D) is the trade name of an operating system marketed by Digital Equipment Corporation which is

designed to run on the PDP-8* family of minicomputers. It consists of a library of system programs which must be stored on some type of mass storage peripheral device, such as DECTape, disk, or floppy disk. Certain OS/8 software (Keyboard Monitor, Command Decoder, User Service Routine, and various device handlers) used to call programs into core and run them, are of interest only to the experienced programmer, and a knowledge of these programs is not necessary for the beginning SKED user to successfully interact with the OS/8 system. The best source of documentation for Version 3D of OS/8 and its component programs is the OS/8 Handbook (DEC-S8-OSHBA-A-D from Digital Equipment Corporation, Communications Services, Parker Street, Maynard, Massachusetts 01754). The OS/8 Handbook will be referenced frequently in the present overview, which is intended to acquaint the OS/8 SKED user with the editor programs, loaders, file-transfer programs, directory functions, and data analysis packages which might be used in connection with OS/8 SKED.

A. SYSTEM FILES

OS/8 is a file-oriented system, which means that programs stored on the system device are assigned a file name. Each OS/8 device is serviced by a directory block, which contains all file names and their physical addresses on the device, as well as their creation dates and the number of blocks they occupy. Each file name is of the following format:

FILENM.EX

where FILENM is the name of the file, six letters or less, and EX is a two-letter extension. The file name may be preceded by a device designation. If none is given, the default is assumed to be DSK: for most programs except R command where SYS: (the system device) is assumed. Any device, such as floppy disk or DECTape, can be assigned as DSK. For example, to use DTA1 as DSK, the command is, ".ASSIGN DTA1: DSK". Other commonly used device designations are DTAO: (for DECTape drive 0), PTR: for papertape reader (never followed by a file name), PTP: for papertape punch, and TTY: when input to a file will be entered from Teletype or output will print out on the Teletype without being written out to a systems file. (See OS/8 Handbook Table 1-12, Pg. 1-31). The file name itself may be up to six letters long with an optional two-letter extension. This extension may be used to indicate the nature of the file's format; for instance, T1000.PA will be recognizable as an ASCII format file while T1000.BN would be an OS/8 binary file. T1000.SV would be a "save" format file which could actually be run on the system (see Section B.4 on saving programs). The extensions can also be used as organizational tags, since the user might choose to call the eighth parametric change to a state table T1000.8. In most cases, choice of an extension is arbitrary, but some OS/8 programs will assume certain extensions as "default" if none is given. (See OS/8 Handbook Table 1-13, Pg. 1-32).

Editor programs and file-transfer programs which manipulate files use a standard format to specify the file to use as input (read) and to specify how the file will be output (written). This format is:

*DEV: FILE1. EX< DEV: FILE2. EX

where File 2 is the input (to the right of the backarrow) and File 1 is the output (to the left of the backarrow). For example, if the EDIT program has been called into core, the command string

*T1000. 1< DTA1: T2

would specify that file T2 (note that this file name has no extension) on DECTape drive 1 will be read into core and under control of EDIT, and when the file has been edited (see Section C on editors), the output will be written onto the DSK: device (DSK: is default) as T1000.1.

B. KEYBOARD MONITOR (KBM) COMMANDS

All that is necessary for the beginning OS/8 SKED user to know about the KBM is that when OS/8 types a period at the left-margin of the Teletype or other terminal, it is ready to accept one of the monitor commands, followed by RETURN or ALTMODE (ESCAPE on some terminals). The OS/8 Handbook covers KBM commands from p. 1-35 to p. 1-44, but the following commands are frequently used by OS/8 SKED users: DATE, R, RUN, SAVE, GET, and ODT.

B.1 DATE

To assign a date to the system, the user may type

.DA 1-FEB-78.

Later, to check the date to make certain it is correct, the user can type .DA. The computer will answer by printing:

WEDNESDAY FEBRUARY 1, 1978.

B.2 RU

To load a program which is in the SAVE format on DECTape drive 0 and start it, the user types:

.RU DTAO PROGRAM. SV

B.3 R

While the RU or RUN command calls in a program from a non-systems device, a program on the systems device would be run by:

.R PROGRAM

Both R and RU will assume the .SV extension if none is specified, but R will also assume SYS as the device, the above example is equivalent to:

.RU SYS PROGRAM

B.4 LOAD and SAVE

Programs that run on the OS/8 system must be saved on the system in a different way than ASCII or binary files. The binary file which is produced by an assembler or compiler (PAL8) is first loaded into core by the Absolute Loader (see Chapter 2-F of this manual), using the LOAD command. The next command will be a SAVE command with arguments indicating sections of core to be saved, a Job Status Word having this information in code (optional), and a starting address. For example, if A.BN produced by PAL8 is to be saved, the following commands might be appropriate:

.LOAD A.BN

.SA SYS A 0-7577, 10000-17577;200

This would make a new file, A.SV, that would save all but the last page on each of the first two fields of memory, and set 200 as the starting address. .R would run this program. (See Chapter 2-F) for more information on LOAD operations.)

B.5 ODT

At times, it is useful to make selected changes in the octal code for programs that have been saved. This can be done by typing a GET command, and then making changes through the Octal Debugging Technique (ODT). When the changes are complete,

the program can be saved again, without specifying arguments unless the program is to be different. For example, to change location 7011 on field 1 from its present contents of 5031 to 5131, in A. SV:

.GET SYS A

.ODT

17011/5031 5131

^ C

.SA SYS A

After the user types the "/", the current content, 5031, is automatically printed. At this point, the user could have left this unchanged by typing, "RETURN", or he could have examined the next location, 17012, by typing LINE FEED, or he could have made the change, as in the above example. Since no further changes were to be made, "RETURN" was typed, followed by Control C (^ C) to return to Monitor control (a period was printed).

ODT can also be used to examine blocks of locations, either searching for occurrences of a certain type word, or simply listing the contents of all locations between two given limits.

C. EDITOR PROGRAMS

C.1 EDIT

For the purposes of creating new files, or making additions, deletions, or other changes to existing files, there are several resident editor programs. The most commonly used of these is the OS/8 version of DEC's Symbolic Editor, called EDIT. This program is described in the OS/8 Handbook, from 1-78 through 1-97. As an example of the use of EDIT to make changes in an existing program, the following commands will open a file on the system called TABLE1.PA for editing, and open TABLE2.PA for the output.

.EDIT TABLE2.PA < TABLE1.PA

While TABLE1 is not itself changed, TABLE2 is the result of the editing. To make changes in TABLE1, the command

.EDIT TABLE1.PA

will use the same file for both input and output. To make a new file using EDIT, the CREATE command can be used.

.CREATE FILE.1

will open FILE.1 for output and will expect input from the keyboard. In cases where an existing file is to be used as input, the first EDITOR command to be used is

#R

to read the file into the text buffer. In creating a new file, the first command will be

#A

to enter text mode. Other Editor commands are summarized in Table 1-23 of OS/8 Handbook, beginning on page 1-93. To close an output file after desired changes have been made, the command

#E

will exit to the monitor after it closes the file. This must be done to save any changes made with EDIT.

C.2 TECO

Another editor program in the OS/8 system is TECO, which is a programmable text editor that offers greater power and speed to the experienced programmer. TECO is described between 2-132 and 2-184.

To open a new file with TECO, use MAKE:

.MAKE TABLE.PA

To change an existing file,

.TECO TABLE.PA

TECO is more difficult to learn, but offers considerable advantages once it is in the behavioral repertoire of the user.

D. DIRECTORY FUNCTIONS

As each device maintains its own directory of files that it has stored, it is sometimes necessary to delete a file, call a file a different name, to remove empty spaces in the directory, to print out the contents of a file, and to list information on all files of a certain description.

D.1 DIRECT

The directory listing functions are controlled by an OS/8 program, DIRECT. (See pages 2-77 to 2-82 in the OS/8 Handbook.) Directory commands are of the form:

.DIR DEV:FILENM.EX/switch options

D.2 Wild Card Characters

In the specification of a file name, it is useful to be able to list file name structures with a certain minimal configuration, regardless what else is included in the name. This can be done by the inclusion of the wild card characters, ? and *. The ? Character will match any character in a corresponding position in a file name or extension. For example, the command:

.DIR FILE??.PA

will cause directory listings to be printed of DSK:FILE1.PA, DSK:FILENM.PA, DSK:FILE.PA, and any other files on the dsk device that match the description.

.DIR DTAO:FILENM.?

Will list directory information for any file on the DECTape mounted on drive 0 which have FILENM and a one letter extension or no extension. Note that FILENM.PA would not match the specified description.

The * wild card will match a whole file name or a whole extension, so

.DIR *.BN

would list all system files having the .BN extension, and

.DIR FILE.*

would list all files having FILE as the name, regardless of their extensions.

.DIR *.*

will give a complete listing of programs on the DSK device, but this could be easily be accomplished by typing .DIR. To obtain a

directory of all files on DEC tape Drive 2, type

.DIR DTA2:

There is a superior version of direct available from DECUS, that has more formatting capability than is included in the OS/8 version.

D.3 Switch Options

Many OS/8 programs contain options for the manner in which an operation will be carried out. This is specified by a "/", followed by a letter designation of the option, usually at the end of the command line. A list of DIRECT switch options is in Table 2-16 on page 2-78 of the OS/8 Handbook. An example of their use,

.DIR/C

would give listings of all DSK files created on the current system date.

.DIR *.BN/F=5

would cause all files on the dsk device with the .BN extension to be listed in their short form (name and extension only) in five columns across the page.

D.4 DELETE and ZERO

Certain other directory functions are controlled indirectly by the FOTP programs. However, it is not necessary to call in the FOTP program formally, by

.R FOTP

*(commands)

Instead, simple commands of the Concise Command Language can be given directly to the monitor.

To delete a file from the device, the DEL command can be used along with the files to be deleted. For instance,

.DEL DTA0: *.BK

would delete from the DECTape mounted on drive 0, all .BK files (backup files created by TECO). To delete all the files from a device, that is not used as a systems device and which therefore does not need to have a system area, the ZERO command can be used:

.ZE:

it usemmand if the device is to be used as the sysice. use the delete command.

D. 5E

call a different name, RENAME will alter the dirwithoing the file in any way. If the user desio call A instead of B, this command will be effi

.REI

D. 6 H

en file deleted and sometimes when files are editaps an the factory which can cause problems. The on to blem is type

.SQU: /O

whiche samening PIP program with its /S and /O opti WithouO theputer will ask ARE YOU SURE? To whicuser mu YES. RNING !! DO NOT SQUISH THE DEVI WHICH IS RDING OR WRITING OS/8 FILES. The systests th to b the locations current when the M cocreated

D. 7

) examincomptents of a file without using an eprogram, user mply request to

.TYPE. PA

Thisistively rOTP wi/L option. If the user does not to contexamirE. PA, he can type CONTROL-O (O) tinate unded ovr CONTROL-C (^c), which alwa:turns col tognitor for almost all OS/8 prog

E. TRANSFER

E. 1 and FOTP

It is frequently necessary to transfer the contents of a file from one device to another, to combine two or more files, and to delete files. The OS/8 system contains several programs which perform these functions, as well as perform some of the directory functions discussed in the previous section. The most basic in format and the easiest to learn is the File Oriented Transfer Program (FOTP). Another program for transfers which are more complicated is the Peripheral Interchange Program (PIP).

E.2 COPY

The easiest transfer program to use is FOTP, which is indirectly run by the COPY command with input and output file specification and any switch options (See OS/8 Handbook, Table 2-20, pages 2-104 to 2-106). For instance,

```
.COPY DSK: < DTA0;Z
```

will cause, by default, a new file (Z.) to be created on the DSK: With the same name as on DTA0.

FOTP always copies files word for word, character for character, without any changes (image mode), unlike PIP which requires transfer options to be specified for any files not in ASCII format.

E.3 PIP Command Format

The other OS/8 file transfer program is PIP. Any commands to PIP must be preceded by .R PIP. The command line format is always:

```
*DEV: FILENM. EX< DEV: FILENM. EX/options.
```

PIP is described in detail from 1-97 to 1-108 in the OS/8 Handbook.

F. THE ABSOLUTE LOADER (ABSLDR)

The Absolute Binary Loader loads absolute binary files produced by the PALB machine language assembler, (as opposed to the relocatable binary produced by the SKED compiler, OSCOMP). However, it may also load programs in the "save" format, if the /I option is used. If a single binary file is to be loaded, the LOAD command can be used, as in the following example:

```
.LOAD A.BN.
```

In cases where the preceding monitor operation included a COMPILE, PAL, or EXECUTE command and a binary file was produced, just .LOAD will be sufficient to use this binary output. For

example, an ASCII file written in the PAL8 language, FILE.PA, is to be assembled by PAL8, creating FILE.BN, which is to be loaded, saved, and given a starting address of 0200:

```
.COMPILE FILE.PA (automatically runs PAL8)
```

```
.LOAD
```

```
.SA SYS FILE 0-4000;0200
```

This will create FILE.SV which can be started at this point by typing .ST. Some programs need to be started when they are loaded, but not saved. The /G option in the following example accomplishes this:

```
.LOAD FILE/G
```

In some cases, there may be multiple files to be loaded, some of which may be in save format. Then, the LOAD command is inadequate and a more formal format is required. The user must call in the ABSLDR program and give a list of files to be loaded. When lines of input are followed by a RETURN, this is interpreted as meaning more files will be input on later lines. If ABSLDR finds an ALTMODE, it assumes that input is finished, and returns to the monitor. The following example first loads the core image save format file (using /I) and then overlays this with four binary files. The save format file must be loaded first, and only one of these may be used (/I on first line only). Following the binary files, a binary tape is read from the papertape reader, and the combined program starts itself (/G option) without being saved on the system.

```
.R ABSLDR
*SAVE.SV/I
*BIN1,BIN2,BIN3,BIN4
*PTR:/G $
```

G. THE SYSTEMS DEVICE

The OS/8 system requires at least one mass-storage peripheral device to hold the systems programs. Initially, an octal code program known as a bootstrap must be toggled in from the switch register to set up a device as SYS:.

(The PDP8A computer has bootstrap routines for most common devices, resident in ROM memory.

However, when another device is needed as the system device, the BOOT command may be used. For example, if the disk is currently SYS:, the command

.BOOT/TD8E

will establish the DECTape as SYS:.

H. PAL8

PAL8 is the name of the assembler on the OS/8 system which reads as input in ASCII format that consists of PAL instruction statements, and produces absolute binary output and ASCII output suitable for source listing. After PAL8 has been called in with the .R PAL8 command, a command string is entered of the following format:

```
*DEV: BINARY, DEV: LISTING < DEV: INPUT/options
```

it expects input with the .PA extension unless another is specified. The binary output will be given .BN extension, unless otherwise specified; the listing will be assigned the .LS extension.

For example, to assemble the file PAL.PA, and make a binary and list file only:

```
.R PAL8
```

```
*PAL, PAL < PAL
```

will create PAL.BN and PAL.LS, the latter of which may now be queued to a lineprinter. Chapter 3 of the OS/8 Handbook gives complete information on the PAL language and its assembler.

I. OS/8 MATH LANGUAGE

I.1 DEC Data Analysis Software

DEC supplies data analysis software with the OS/8 system at extra cost. The languages that are available are BASIC, FORTRAN II, and FORTRAN IV. In addition, a streamlined version of PS/8 FOCAL called U/W FOCAL is available through:

Software Development Group
Oregon Museum of Science and Industry
4015 S. W. Canyon Road
Portland, Oregon 97221

I.2 BASIC

Chapter 6 of the OS/8 Handbook deals exclusively with OS/8 BASIC. The description given here will provide only the

essential instructions for running an existing program written in the BASIC language, or to begin writing a new program. It assumes that all components of the OS/8 BASIC system exist on the device, namely the BASIC editor, the BASIC compiler, the BASIC loader, the BASIC runtime system, and various runtime system overlays. (See OS/8 Handbook, pages 6-70).

A. The following command to the Monitor brings in BASIC:

```
.R BASIC
```

Once BASIC is in core and starts, it asks the following:

NEW OR OLD -

B. If the user wishes to create a new program in the BASIC language, he responds "NEW" followed by the file name he wishes to call his program. A typical reply of this sort might look like:

```
.R BASIC  
NEW OR OLD - NEW ANALYZ.BA
```

The extension used by most BASIC programs is BA.

After this is typed, BASIC responds with READY. Then the user can start typing in a new program.

C. If the user already has a program he wants to run he types in:

```
OLD FILE.BA
```

When BASIC types READY, he may type RUN (which prints a header line for the program) or RUNNH (no header). Or, if he wants to make changes in FILE.BA, and keep both the original and corrected files, he may type his changes, and finish by typing:

```
.SAVE FILENW.BA
```

D. To exit from BASIC and return to the keyboard monitor, the user types "CTRL-C" (^C) while in the READY mode, or he types "BYE".

E. A sample BASIC program, BDUMP.BA accepts FILE.FD as input, and prints the headers for each box, followed by all the counters for each station. The program is listed below, followed by a sample run.

.R BASIC
NEW OR OLD - OLD
FILE NAME - BDUMP
READY
LIST

```
BDUMP  BA 3.0  09-MAR-78
50 DIM X(256)
100 LET V9=20
200 FILE 1: "FILE. AD"
600 INPUT 1: B
610 IF B=0 THEN 2900
700 FOR K=1 TO 7
710 INPUT 1: X(K)
715 NEXT K
800 B=(B/2)
810 FOR K=8 TO B+7
820 INPUT 1: X(K)
825 INPUT 1: U
830 X(K)=X(K)+4096*U
835 NEXT K
1050 PRINT
1055 PRINT
1100 PRINT "DATE  "; X(2); "/" ; X(3); "/" ; X(4)
1300 PRINT "EXP NO. ="; X(5); "GROUP NO. "; X(7); "SUB"; X(6); "BOX"; X(1)
1350 PRINT
1400 C=12
1405 S=15
1500 FOR K=8 TO B+7
1600 GOSUB 2300
1700 NEXT K
1800 LET V9=V9-1
1900 IF V9=0 THEN 2100
1905 PRINT
1907 PRINT
2000 GOTO 600
2100 PRINT
2105 PRINT "DONE.... FINALLY"
2200 STOP
2300 IF (K-C)=0 THEN 2600
2305 PRINT (K-7); "=";
2400 PRINT X(K); TAB (S);
2405 S=S+15
2500 RETURN
2600 PRINT (K-7); "=";
2650 PRINT X(K)
2700 C=C+5
2705 S=15
2800 RETURN
2900 END
```

.R BASIC
NEW OR OLD - OLD

FILE NAME - BDUMP
READY
RUN

BDUMP BA 3.0 09-MAR-78

DATE 3/7/1976

EXP. NO. = 3101 GROUP NO. 1 SUB 632 BOX 1

0	0	5	1	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	10	10	5	88	83	98
8	1	0	1	60	1	0	1	1670	

READY
BYE

F. To create a "batchable" Save format version of the program BDUMP.BA which assembles more quickly and loads using the R or RUN commands, use the following procedure:

```
.L R BCOMP
*BDUMP.BA/K/B
.SA SYS BDUMP
.R BDUMP
```

I.3 FORTRAN II

OS/8 FORTRAN II offers standard FORTRAN features such as explicit and implied do loops, Hollerith statements, program chaining subroutine calls, and device-independent I/O. Users who are familiar with FORTRAN IV (as implemented on large computers) will find FORTRAN II fast and easy to use, but lacking sophistication in the handling of implied statements.

FORTRAN II consists of three components: the FORTRAN compiler (FORT), an assembler (SABR), and the LINKING LOADER. Programs are prepared using standard OS/8 editors or punch cards. These programs, written in ASCII, are submitted to the FORTRAN compiler via the OS/8 command decoder. The compiler creates a new file, which is suitable for input to the SABR assembler. The FORTRAN compiler automatically chains to the SABR assembler, which produces a relocatable binary file. The user has available

listings, bitmaps, and a variety of switch settings (to control processing and select various options). The binary file produced by the SABR assembler is loaded via the LINKING LOADER. The loader may be called via a switch option to the FORTRAN compiler or SABR assembler, or called directly (as when previously compiled routines are to be loaded). The purpose of the LINKING LOADER is to provide all subroutines with information about the locations in core of various other subroutines and where data arrays are to be found in core. After specifying to the command decoder (called by the loader) user-supplied subroutines, the LINKING LOADER searches its library of subroutines (LIBB) for any OS/8 supplied subroutines which are called (such as EXIT or CHAIN) and creates a program which is ready for use, or may be saved for reuse. Syntax and logical errors are checked for in all three programs and comprehensive error print-outs allow easy debugging of programs. FORTRAN II may be used with the interrupt on, however, users must supply their own interrupt handler and may have to re-write some OS/8 LIBB I/O routines as they are not written to be interrupted (see OS/8 Handbook, Section 7.6 for further information).

I.4 Extended ANSI FORTRAN IV System

The most powerful data analysis tool in OS/8 is an extended ANSI FORTRAN IV system. Through chaining and overlays, OS/8 FORTRAN is able to run large programs in 8K of core. A disk and 12K are very desirable, although not essential. I/O is device independent, making use of the OS/8 handlers, and internal interrupt driven handlers for the Teletype and line printer.

The FORTRAN source is compiled by F4 which chains to the necessary files to produce a relocatable binary file (Ralf module). These "RL" files may be linked with other RALF modules by LOAD to produce a loader image ("LD" file) or stored by LIBRA in the FORTRAN library which is searched automatically by LOAD. The FORTRAN Runtime System (FRTS) executes the loader image, after checking to see if the current hardware has an FPP or EAE (KEB-E only). Thus, FORTRAN software may be run on different hardware configurations with no revisions.

FORTTRAN runs roughly twice as fast with EAE and from ten to twenty times as fast with an FPP. FORTRAN should run well on the new 8/A with FPP (Super 8). FORTRAN does not use the FPP hardware while running in background under RTS-8.

Presently, DEC does not supply FORTRAN routines to open or close files and/or devices within the FORTRAN program or to read integer data files. Several subroutines can be supplied by the SKED Users Group to facilitate reduction of SKED data. These modules may then be included in your FORTRAN library (FORLIB.RL).

A brief description of the available subroutines and an example follow:

A. SUBROUTINE USR (UNIT, NAME, FUNCT, ERROR). This routine allows runtime declaration of files.

UNIT: Logical unit number. Only numbers 5-9 are allowed. Fewer logical units may be allowed depending on core availability (see programming note below).

NAME: DEV:FILE.EX

Stored in format 3A6 or equivalent. Device assumed to be DSK: if not otherwise stated. Null characters ('@') and spaces are ignored in this field.

FUNCT: Function:

- 2 - Open file for input
- 3 - open file for output
- 4 - close output file

ERROR: Return error condition:

- 0 - No errors
- 1 - Illegal device
- 2 - Illegal file name
- 3 - Illegal unit number (core exceeded!?)
- 4 - Illegal function code.

User errors may terminate execution unless the /E option was specified to FRTS. The following user errors from USR are defined:

0002 - The user has defined a non-resident device handler external to USR.

B. READB WRITB. Allows reading/writing binary files in FORTRAN. The calling sequence is:

```
DIMENSION BUFR (256)
CALL READB (UNIT,BUFR)
```

or

```
CALL WRITB (UNIT,BUFR)
```

This reads/writes the next block of 256 12-bit binary words from/to logical unit number UNIT into/from the 256 element array BUFR. The values read/written are between 0 and 4096. The program will not read or write past an End Of File. To check for an End Of File, code the following:

```
11  EXTERNAL EOF
    LOGICAL EOF
    IF EOF GO TO 900
    (continue source code)
```

900 (process End Of File)

If SKED data files are 248 counters and 8 for header (256 counters, or one OS/8 block), then the next two routines (READ and WRITE) are not needed, greatly reducing analysis time!

C. READ. Calls READB to allow reading less or more than one OS/8 block at a time. Calling sequence:

CALL READ (UNIT,A,I)

Read I elements into array A from the next I locations of the opened binary file. New records will be read from unit UNIT. To read from a new file, skip the remaining words on the current record by calling the subroutine with I=0.

D. WRITE. Calls WRITB to allow writing more or less than one OS/8 block. Calling sequence:

CALL WRITE (UNIT,A,I)

Write I elements from array A into the next I locations of the opened binary file. Records will be written on unit UNIT. To write out the last record before closing the file, call the subroutine with I=0.

E. UREAD. Allows reading basic numeric files with FORTRAN unformatted READ or reading more or less than one OS/8 basic numeric WRITES compatible with FORTRAN READS. Calling sequence:

CALL UREAD (UNIT,A,I)

Read unformatted files 85 numbers to a block. Reads I elements into array A from any full unformatted file. Enter with I=0 to skip rest of block. UNIT is FORTRAN unit number.

F. UWRITE. Allows FORTRAN unformatted writes to be read with BASIC or more or less than one FORTRAN unformatted record to be written at one time and still have one record per block. Calling sequence:

CALL UWRITE (UNIT,A,I)

Write I elements from array A into a file on unit UNIT. Enter with I=0 to fill block with zeros ready for closing.

G. IOT. Allows 8-mode IOT's to be issued. Calling sequence: CALL IOT (INST,AC) INST is the 8-mode IOT instruction in decimal. AC is the value of the accumulator when the instruction is

issued. If the IOT changes this value, the new value is returned. This feature may be used to read the switch register if the instruction LAS is issued in place of an IOT. The following program IDUMP.FT demonstrates the use of some of these subroutines.

```

C              IDUMP.FT
C
C  READS SKED DATA FILES MADE WITH ANY INTEGER SKED RTS.
C  PRINTS SKED HEADER FOR EACH DUMP, THEN THE COUNTERS.
C  DOUBLE COUNTERS ONLY. CHANGE FOR SINGLE COUNTERS!!
C  CTRL O TO STOP LISTING AT END OF CURRENT DUMP.
C
      DIMENSION DVFILE(3), HEAD(8), CONTER(400), FMAT(6)
      EXTERNAL EOF
      DATA FMAT/36H(8(14, '=', 18, 2X)) (5(14, '=', 18, 2X)) /
      DATA YES/'YES'//, NEW/'NEW'//, UNIT/4//, IFMTPT/4/
      LOGICAL EOF, ERR
1      WRITE (4, 5)
5      FORMAT('output TO LINE PRINTER? (YES OR NO) '$)
      READ (4, 20)ANS
      IF(ANS.EQ. YES)UNIT=3
      IF(ANS.EQ. YES)IFMTPT=1
7      WRITE (4, 10)
10     FORMAT ('OENTER INPUT DEV: FILE. EX '$)
      READ (4, 20)DVFILE
20     FORMAT (3A6)
30     CALL USR(9, DVFILE, 2, ERR)
      IF (.NOT. err) GO TO 50
      WRITE (4, 60)ERR
60     FORMAT ('USR ERROR', 12, ' TRY AGAIN')
      GO TO 7
50     WRITE (4, 100)
100    FORMAT('SINGLE SUBJECT SEARCH (YES, NO OR NEW) ', $)
      READ (4, 20)ANSWER
      IF(ANSWER.EQ. NEW)GO TO 7
      IF(ANSWER.NE. YES)GO TO 200
      WRITE (4, 120)
120    FORMAT('ENTER SUBJECT NUMBER ', $)
      READ (4, 150)SUBJNO
150    FORMAT(15)

200    CALL READ (9, HEAD, 8)
      IF (EOF)GO TO 454
      IF (HEAD(1).GE. 1)GO TO 456
      WRITE (UNIT, 452)
452    FORMAT ('OEND OF SKED DATA FILE. FIRST COUNTER =0. ')
454    CALL READ (9, DUMMY, 0)
      REWIND 9
      GO TO 50
456    CALL READ )9, CONTER, HEAD(1))
      IF (ANSWER.EQ. YES. AND. HEAD(7). NE. SUBJNO)GO TO 200
      DO 500 I=1, HEAD(1), 2

```

```

J=J+1
500  CONTER(J)=)4096*CONTER(I+1))+CONTER(1)
    C WRITE ALL THE JUNK OUT

    WRITE (UNIT,510) HEAD(7),HEAD (3),HEAD(4),HEAD(5)
510  FORMAT ('1SUBJECT NO. =',14,' ',14,'/',12,'/',14)
    WRITE(UNIT,520)
520  FORMAT ('0 BOX NO.  EXP.NO.  GROUP NO. ')
    WRITE(UNIT,530)HEAD(2),HEAD(6),HEAD(8)
530  FORMAT ((1H,' ',12,' ',14,' ',14)///)
    WRITE(UNIT,FMAT(IFMTPT))(I,CONTER(I),I=1,HEAD(1)/2)
    CALL IOT(3102,CTRL0)
    IF(CTRL0.EQ. 143)GO TO 454
    GO TO 200
    END

```

To implement this program on your system without the subroutines in FORLIB.RL, do the following:

```

.COM IDUMP.FT
.R LOAD
*IDUMP(IDUMP,READ,READB,USR,IOT/G
$ (altmode)

```

To run the program a second time, type:

```
*IDUMP.LD $
```

The program will respond with:

```
OUTPUT TO LINE PRINTER? (YES OR NO)
```

Once all the routines are inserted in FORLIB.RL with LIBRA, the only command that is necessary to compile, load, and execute is:

```
.EXEC IDUMP
```

This produces a loader image file and executes it the first time. To run the loader image you must do:

```
.R FRTS
```

```
*IDUMP $
```

I.5 U/W FOCAL

A. U/W FOCAL (DECUS No. FOCAL 8-301) is an improved and expanded version of PS/8 FOCAL (DECUS FOCAL 8-177) with more commands, more functions, and other improvements, while fitting in the same core space. Documentation on both these programs is available from Oregon Museum of Science and Industry. As this is

not a standard OS/8 language, the user must first obtain and load either FOCAL '69 or FOCAL 8, and then overlay it with a binary program FCMOD, and save the programs as UFOCAL.SV, or he may obtain the complete U/W FOCAL from DECUS. A version of U/W FOCAL is available that can be run in the background while Time Share SKED runs in the foreground.

B. When UFOCAL is on the system, it may be called in and run by .R UFOCAL. It will respond with an *, to which the user may start typing in a new program (U/W FOCAL has its own editor). when he has finished entering it, he can type,

```
*L S PROGRM
```

(library save) to save it as PROGRM.FC. Or he may want to start it before or after the proceeding step by ust typing *G.

C. If he already has a program that he wants to run, he can type, "*L R LATNCY", (library run) assuming the program LATNCY.FC exists on the system.

D. If there is a program which needs modification, the user can type,

```
*L C PROG (library call)
```

```
(changes)
```

```
*L S PROG
```

E. The "break character" for U/W FOCAL is CTRL-F (F). This will terminate any FOCAL operations and set the program counter back to the beginning of the program currently in core. CTRL-C (C) will return the user to the Monitor, leaving UFOCAL. This can also be done under program control by command

```
*L E.
```

```
.SA SYS UFOCAL
```

F. One FOCAL program of general interest is FDUMP.FC. This program types all of the counters, 5 per line.

```
C U/W-FOCAL:  FDUMP  06/24/70
```

```
01.01 ZERO
```

```
01.06 L GOSUB OPEN
```

```
01.07 S Y=FRA(-1,W)
```

```
01.10 S B=FRA(0+W)
```

```
01.11 I (B) 1.12,9.01,1.12
```

```
01.12 F K=1,1,7;S X(K)=FRA(K+W)
```

```
01.30 T %4,!, "DATE" X(2),"/",X(3),"/",X(4),!!
```

```
01.35 T %4,"EXP=",X(5),"GROUP NO.",X(7),"SUB ",X(6),"BOX",X(1)!
```

```
01.37 T %3
```

```
01.38 S C=18
```

```

01.40 F K=8,2,B+7;S X=FRA(K+W);S Y5=FRA(K+W+1);S X=X+Y5*4096;D 2
01.45 S LEN=LEN-1
01.46 I (LEN) 1.65,1.90,1.65
01.65 I (B-248) 3.01,1.67,3.01
01.67 S B=FRA(B+8)
01.70 I (B) 1.71,1.90,1.71
01.71 S W=W+256
01.72 GOTO 1.12
01.90 T "DONE",!
01.95 Q

```

```

02.01 I (K-C) 2.05,2.10,2.15
02.05 T (K/2-3), "=",X;R
02.10 T !,%3,(K/2-3), "=",X;S C=C+10;R
02.15 T X;R

```

```

03.01 F K=B+8,1,257;S X=FRA(K)
03.02 S B=X
03.77 GOTO 1.70

```

```

09.01 T "NO DATA IN INPUT FILE YET",!
09.80 L GOSUB OPEN 9.40
09.85 GOTO 1.07

```

C U/W-FOCAL: OPEN 06/20/70

```

09.01 O I TTY:,E
09.10 T "INPUT FILE NAME? ";F I=14,-1,1;S C(I)=-192
09.20 S C=FIN();I (C-255),9.3;I (C-141),9.4;S I=I+1,C(I)=-C;G 9.2
09.30 S C(I)=-192,I=I-FSGN(I);I (C)9.2;T "";G 9.2
09.40 O I (C(1))(C(2))(C(3))(C(4))(C(5))(C(6))(C(7))(C(8))
(C(9))(C(10))(C(11))(C(12))(C(13))(C(14))
09.50 O I,E;I (C(1)+192)9.6,,9.6;T !;R
09.60 T %4 :30 F?BLK?()" "F?LEN?(1) !
09.65 S LEN=FLEN(1)
09.70 R

```

```

*L R FDUMP
INPUT FILE NAME? DATA.DA

```

BLK 1151 LEN 1

DATE 6/ 24/ 1978

```

EXP NO. = 2221 GROUP NO. 1 SUB 350 BOX 6
1= 0 2= 27 3= 23 4= 4 5= 0
6= 179 7= 133 8= 27 9= 23 10= 27
11= 23 12= 3 13= 2 14= 48 15= 821
16= 50 17= 12DONE

```

Although the above program is useful for listing the counters of data files obtained with the WRITE statement (see Chapter 1), other programs are better suited for listing the

counters of files obtained with the RECORD statement. Since a RECORD data file contains double precision counters which include the code of the RECORDeD event as well as the time since the last event occurrence, it is to the experimenter's convenience to have a program which separates out the code from the time since the event occurrence before listing the counters. The following program, listed in Focal, Basic, and Fortran, first separates out the code from the time, and then prints them out in the following format.

Counter No.=Code   Time since last RECORD event

C U/W-FOCAL: FCODE 04/27/78

01.01 ZERO

01.02 A "IF SINGLE SUBJECT SEARCH TYPE NO. ",N

01.06 L GOSUB OPEN

01.07 S Y=FRA(-1,W)

01.10 S B=FRA(0+W)

01.11 I (B) 1.12,9.01,1.12

01.12 F K=1,1,7; S X(K)=FRA(K+W)

01.15 IF (N) 1.16,1.30,1.16

01.16 IF (N-X(6)) 1.45,1.30,1.45

01.17

01.30 T %4,!, "DATE" X(2),"/",X(3),"/",X(4),!!

01.35 T %4, "EXP=",X(5), "GROUP NO. ",X(7), "SUB",X(6), "BOX",X(1)!

01.37 T %6

01.38 S C=16

01.40 F K=8,2,B+7; S X=FRA(K+W); S Y5=FRA(K+W+1); D 4; D 2

01.45 S LEN=LEN-1

01.46 I (LEN) 1.65,1.90,1.65

01.65 I (B-248) 3.01,1.67,3.01

01.67 S W=W+256

01.72 GOTO 1.12

01.90 T "DONE",!

01.95 L E

02.01 I (K-C) 2.05,2.10,2.15

02.05 T %3, (K/2-3), "=", I, %6, X; R

02.10 T !, %3, (K/2-3), "=", I, %6, X; S C=C+B; R

02.15 T X; R

03.01 F K=B+8,1,257; S X=FRA(K)

03.02 S B=X

03.99 GOTO 1.70

04.10 F U=0,256,256*15; IF (Y5-(U+255)) 4.50,4.50,NEXT U

04.50 B

04.51 S I=U/256

04.60 S Y5=(Y5-U)*4096

04.70 S X=X+Y5

09.01 T "NO DATA IN INPUT FILE YET",!

09.80 L GOSUB OPEN 9.40

09.85 GOTO 1.07

BCODE BA 3.0 10-JUL-78

```
50 DIM X(256)
75 DIM P(256)
80 PRINT "IF SINGLE SUBJECT SEARCH TYPE SUBJECT NO. "
85 INPUT N
100 FILE 1: "FILE. AD"
600 INPUT 1: B
610 IF END 1 THEN 2900
620 IF END 1 THEN 2900
700 FOR K=1 TO 7
710 INPUT 1: X(K)
715 NEXT K
800 B=B/2
810 FOR K=8 TO B+7
820 INPUT 1: X(K)
821 IF END 1 THEN 2900
825 INPUT 1: U
826 P(K)=INT(U/256)
827 U=U-P(K)*256
830 X(K)=X(K)+4096*U
835 NEXT K
900 IF N=0 THEN 1050
905 IF N=X(6) THEN 1050
910 GOTO 600
1050 PRINT
1055 PRINT
1100 PRINT "DATE "; X(2); "/" ; X(3); "/" ; X(4)
1300 PRINT "EXP NO. "; X(5); "GROUP NO. "; X(7); "SUB"; X(6); "BOX"; X(1)
1350 PRINT
1400 C=11
1405 S=15
1500 FOR K=8 TO B+7
1600 GOSUB 2300
1700 NEXT K
1705 PRINT
1907 PRINT
2000 GOTO 600
2300 IF (K-C)=0 THEN 2600
2305 PRINT (K-7); "=" ; P(K);
2400 PRINT X(K); TAB (S);
2405 S=S+15
2500 RETURN
2600 PRINT (K-7); "=" ;
2625 PRINT P(K);
2650 PRINT X(K)
2700 C=C+4
2705 S=15
2800 RETURN
2900 STOP
3000 END
```

```

C
C                                     SSCODE. FT
C
C READS SKED DATA FILES MADE WITH ANY INTEGER SKED RTS.
C PRINTS SKED HEADER FOR EACH DUMP, THEN THE COUNTERS.
C DOUBLE COUNTERS ONLY.  CHANGE FOR SINGLE COUNTERS!!
C CTRL Q TO STOP LISTING AT END OF CURRENT DUMP.
C FOR SKED DATA MADE WITH F3CODE
C
  DIMENSION DVFILE(3), CONTER(256), FMAT(8), ICODE(256)
  EXTERNAL EOF
  DATA FMAT/36H(8(14, '=', 18, 2X)) (5(14, '=', 18, 2X)) /
  DATA YES/'YES'//, NEW/'NEW'//, UNIT/4//, IFMTPT/4/
  LOGICAL EOF, ERR
1  WRITE (4, 5)
5  FORMAT(' OUTPUT TO LINE PRINTER? (YES OR NO) '$)
  READ (4, 20)ANS
  IF(ANS.EQ.YES)UNIT=3
  IF(ANS.EQ.YES)IFMTPT=1
7  WRITE (4, 10)
10  FORMAT(' OENTER INPUT DEV: FILE. EX '$)
  READ (4, 20)DVFILE
20  FORMAT (3A6)
30  CALL USR(9, DVFILE, 2, ERR)
  IF (.NOT.ERR) GO TO 50
  WRITE (4, 60)ERR
60  FORMAT(' USR ERROR', I2, '. TRY AGAIN')
  GO TO 7
50  WRITE (4, 100)
100  FORMAT(' SINGLE SUBJECT SEARCH (YES, NO OR NEW) ', $)
  READ (4, 20)ANSWER
  IF(ANSWER.EQ.NEW)GO TO 7
  IF (ANSWER.NE.YES)GO TO 200
  WRITE (4, 120)
120  FORMAT(' ENTER SUBJECT NUMBER ', $)
  READ (4, 150)SUBJNO
150  FORMAT(I5)
200  CALL READB (9, CONTER(1))
  HEAD=CONTER(1)
  IF (EOF)GO TO 50
  IF (ANSWER.EQ.YES.AND. CONTER(7).NE. SUBJNO)GO TO 200
  J=8
  DO 500 I=9, CONTER(1)+8, 2
  J=J+1
  ICODE(J)=CONTER(I+1)/256
500  CONTER(J)=(4096*(CONTER(I+1)-ICODE(J)*256))+CONTER(I)
  C WRITE ALL THE JUNK OUT
  WRITE(UNIT, 510) CONTER(7), CONTER(3), CONTER(4), CONTER(5)
510  FORMAT ('1SUBJECT NO. =', I4, ' ', I4, '/', I2, '/', I4)
  WRITE(UNIT, 520)
520  FORMAT ('O BOX NO. EXP NO. GROUP NO. ')
  WRITE(UNIT, 530)CONTER(2), CONTER(6), CONTER(8)
530  FORMAT ((1H , ' ', I2, ' ', I4, ' ', I4)///)

```



```
      WRITE(UNIT,FMAT(IFMTPT))(I,ICODE(I+8),  
1  CONTER(I+8),I=1,HEAD/2)  
      CALL IOT(3102,CTRLD)  
      IF(CTRLD.EQ.143)GO TO 50  
      GO TO 200  
      END
```

J. BATCH PROCESSING

J.1 OS/8 offers the capabilities of BATCH processing (see OS/8 Handbook, p. 2-1 to 2-25). This means that a series of OS/8 instructions can be included in a control file, along with a few BATCH commands. When this control file is submitted, the commands are executed sequentially.

J.2 The control file can be created with EDIT or TECO and should be assigned a .BI extension. The first command line should start with the \$JOB command. The \$MSG command can be used to print a message to the user and pause until any key on the keyboard is struck, at which time it will continue with the batch stream.

The \$END command should be included as the last entry in the control file. Keyboard Monitor Commands and other command strings entered to the control file must contain their own left-margin periods or asterisks that are normally supplied by the Monitor and the programs it calls in.

J.3 Once a control file, JOB.BI, has been created, it can be executed by typing, "SUBMIT JOB.BI", or just, ".SU JOB" (.BI is assumed by Monitor).

J.4 BATCH, as distributed by DEC, has a number of problems in it. These bugs have been eliminated by Geoff Inglis, and the new version may be obtained by any bona fide OS/8 licensee through the SKED Users Group.

K. C.C.L.

Besides the formal KBM commands (ASSIGN, DEASSIGN, GET, SAVE, ODT, RUN, R, START, and DATE), the user is provided with an extended set of KBM commands through the use of the Concise Command Language (CCL). It is a type of shorthand that allows the user to combine the calling sequence of a program and its command decoder input string into one indirect calling sequence. Several CCL commands have been included in the preceding summaries of OS/8 programs, e.g. EDIT, CREATE, MAKE, TECO, DIRECT, DELETE, ZERO, RENAME, SQUISH, TYPE, COPY, COMPILE, LOAD, BDOT and SUBMIT. These commands take the place of longer calling sequences. For instance,

EDIT T100.SK

is equivalent to:

.R EDIT *T100.SK<T100.SK

Five new CCL commands have been added to the DEC version.

DEC10
FUTIL
PIPI
MON
SYSTAT

Also several commands including EXECUTE, COMPILE and LOAD incorporate state table extensions as described in section on compiling state tables. The command VER will list the CCL recollections of EDIT, TECO, EX-COM-LOAD and UA, UB and UC commands. Other standard DEC CCL commands are described in the OS/8 Handbook, pages 1-52 to 1-75.

COMPILING STATE TABLES

The SKED compiler is a program designed to transform ASCII format state tables produced by EDIT or TECO into relocatable binary format object files, ready to be loaded by MON into the desired stations. The compilation process is a two pass system with an optional third pass to produce listings with line numbers. State table errors detected by OSCOMP will produce an error report and will print the line containing the error. Unfortunately OSCOMP cannot be restarted after an error is detected so that errors must be eliminated in sequential fashion, one error at a time. Although OSCOMP can be run directly by the R OSCOMP command to the Keyboard Monitor of OS/8 (KBM), a number of CCL commands are available for more powerful control over the compiling process.

The basic strategy of the compiling system is to use standard extensions and the same state table name for the 5 files comprising the state table system. The five files and their extensions are:

<u>FORMAT</u>	<u>CREATED BY</u>	<u>FUNCTION</u>	<u>EXTENSIONS</u>
ASCII	EDIT	Typed state table	.SK<
BINARY	OSCOMP	Relocatable Binary	.ST<.SK
BINARY	IDENT	Identifiers	.ID<
BINARY	MERGE	Merged	.SM<.ST+.ID
ASCII	OSCOMP	Line Numbered listings	.LS<.SK

1. ASCII State table = .SK

THE typed state table is given the extension .SK standing for SKED. this table is in ASCII format and can be produced by

EDIT or TECO. This file usually is created on the DSK device and is saved for archival purposes and for modifications for similar procedures. Assuming the name of the experiment is to be EXP1, then first test for the presence of any other files on the DSK with any extensions with the CCL command:

```
.DIR EXP1.*
```

The .* is a "wild card" option that will cause the computer to type all of the programs on DSK (the default since no device was specified) with the name EXP1. If none are found, it is safe to proceed. If files with the EXP1 name are found, it may be dangerous to proceed. Those with extensions given in the above table will be replaced by files with the same name. Files with the same name but different extensions may interfere with compiling (especially those with the extensions .BA, .FT, .PA, .RA or .SB) since the COMPILE command recognizes these extensions and assigns them a higher priority than the .SK extension (see OS/8 P 1-59). Furthermore, the extensions .BN and .RL will interfere with the LOAD command.

Assuming that no other files called EXP1 exist on DSK, then the command:

```
.CREATE EXP1.SK
```

will call EDIT and permit typing the state table. Similarly:

```
.MAKE EXP1.SK
```

will call TECO for this job. After EXP1.SK has been typed and the file has been closed, then the compiler can be used to check for errors. The command:

```
.COMPILE EXP1
```

will start OSCOMP, which will search EXP1.SK for errors. If none are found, OSCOMP will print the final address of the state table in octal, and will chain to DSC002 to produce the relocatable binary file EXP1.ST. If an error is found by OSCOMP, then an error number is printed along with the line containing the error upto the point at which the error was detected. Furthermore, an error message is typed, explaining the cause for the error. It is often useful at this point to have OSCOMP produce a listing of the state table along with the error report printed below the line containing the error. This is especially useful for correcting errors with EDIT or TECO, since line numbers are printed for each line of the state table. To get this listing on your terminal type:

```
.COMPILE EXP1-T
```

If you want to store a listing on the DSK device, typing

```
.COMPILE EXP1-LS
```

will produce EXP1.LS on DSK.

If no error was discovered by OSCOMP, or if the error has been corrected, EXP1.ST now exists on your DSK device. The next step is to load and test the program.

```
.LOAD EXP1 (01)
```

will call MON to load EXP.ST into stations 0 and 1 of time-shared SKED. If the editing and compiling was conducted in stand-alone OS/8, MON will ask for the time-of-day when it is called. After the time has been entered, stations 0 and 1 will be loaded. If the system already is in the time-shared mode, then the loading of the state table will take place directly.

Sometimes, errors are not detected by OSCOMP and the .ST file will have problems. These are usually discovered when the .ST file refuses to be loaded. MON will type:

CHECKSUM ERROR ON .ST FILE

If this occurs, try recompiling and reloading. If the problem continues, study your state table for errors. If you can't find any obvious errors, mail the listing of your state table to SKUG along with an explanation. We will try to correct the problem and will improve OSCOMP to detect this type of error in the future.

Even if your state table loads successfully, it is necessary to test whether it actually works the way you had planned. This requires simulating responses from the terminal or push-buttons to see if the contingencies operate properly.

When your state table works the way you want it to, it is time to create an identifier file, containing the experiment subject and group numbers for each subject to be exposed to this state table. Although the identifiers which are used to identify your data files, can be typed in MON by the Z command, it is usually more convenient to add these identifiers to your state table. To create the identifiers and to merge them with your .ST file to create a .SM file, can be done in a number of ways. The command:

```
EXECUTE EXP1 (01)/I
```

will recompile EXP1.SK to make EXP1.ST and then chain to IDENT.SV to ask you for the EXP, SUB and GROUP numbers for stations 0 and 1. When you have answered these questions, IDENT will chain to MERGE.SV to produce EXP1.SM. Then MERGE will chain to CCL to

call MON and load EXP1.SM into stations 0 and 1. In the future you can merely type:

```
LOAD EXP1 (01)
```

which will load your .SM file. At this time you can delete EXP1.ST and EXP1.ID to save room on your DSK device.

Under some circumstances, however, you may want to save these intermediate files. For example, if you want to use the same state table for different groups of subjects to be run at different times you will need to create another .SM file for the second group. This file should contain different identifiers, appropriate to the second group. However, the basic .ST file is the same. To create EXP2.SM, the use the following commands:

```
.R IDENT
```

```
*EXP2.<(35)
```

and then answer the questions. The file EXP2.ID will now exist. To merge this with file EXP1.ST to produce EXP2.SM requires the following command:

```
.R MERGE
```

```
* EXP2< EXP1.ST, EXP2.ID
```

You now can load EXP2 into stations 3 and 5.

Note on switch options:

To load stations 10 and 11 you have to type (YZ). Sorry about that, but it is one of those problems created by DEC OS/8 conventions.

CHAPTER 7

ASSEMBLING TIME SHARE SUPERSKED

To assemble your time-sharing system you need:

- A. A license for OS/8 from Digital Equipment Corp. and the associated programs.
- B. A license for RTS/8 from DEC and associated source programs.
- C. A PDP8 family computer with at least 8K of memory and OS/8 for assembling the system and a PDP8 family computer with at least 16K of memory for operating time-shared SUPER SKED.
- D. At least one mass storage device supported by OS/8 and RTS8 handlers including:
  - I. RKB
  - II. RK8E
  - III. RX01
  - IV. TC08
  - V. TD8E*
  - VI. RL8A

*Note section concerning restrictions on TD8E operating system

E. The following programs from SKUG:

	NAME	VERSION
I.	PARAM. PA *	
II.	SKED. PA	V. 9
III.	EXEC. PA *	
IV.	OS8. PA*	
V.	OSCOMP. SV	3. 30
VI.	OSC002. SV	3. 30
VII.	OSC003. SV	3. 30
VIII.	SYSTAT. PA	
IX.	CHAINS. PA	
X.	CCL. SV *	V2B

XI.	BATCH. SV **, *	BEH78 V3
XII.	PIPI. SV	V 12
XIII.	TRANIA. SV	V 6
XIV.	MERGE. SV	V 4
XXV.	IDENT. SV	V 5
XVI.	SKED. BI	
XVII.	MON. SV**	V 11 OR LATER
XVIII.	HELP. HL*	

* Improved versions of DEC software

**Sources for these are available from SKUG

Step 1.

The .file PARAM.PA contains a number of parameters which must be adjusted for your system. These parameters include:

- A. Allocation of memory for the background OS/8 system and selection of the background OS/8 device handlers.
- B. Adjustments of special options in the run-time system.
- C. Addition of special time-sharing tasks including additional device handlers for additional OS/8 devices or special OS/8 terminals, etc.
- D. Selection of interface hardware. (CLOCK and I/O)
- E. Selection of system task for OS/8 and RTS8.

Each of these options will be considered below. To modify parameters a working knowledge of a DEC editor, Edit or Teco is assumed.

The parameter file PARAM.PA consists of two sections. The first section contains parameters SKED users may want to modify. The second section which starts with the comment

STOP HERE!!!!

Should not be altered in any way. The first step in adjusting your PARAM file is to type it. If you have CCL on your system, the command

TYPE DEV:PARAM.PA

will print the entire file for your inspection.

A. Memory allocation

1. HGFHLD=70 The first memory allocation is related to the amount of core available on the computer which will execute SUPER-sked (not the computer on which the assembly is conducted). The parameter to be adjusted is HGFHLD which should be equal to 10 times the number of the highest memory field. Thus, if you are assembling SKED for a 16K machine HGFHLD=30, which is 10 times FIELD 3, the largest field. The following table shows HGFHLD for 16 to 32K computers:

<u>MEMORY</u>	<u>HGFHLD</u>
16	30
20	40
24	50
28	60
32	70

The easy way to adjust HGFHLD is to search for HGFHLD= and then replace the number you find with the number you wish. The system is initially distributed with HGFHLD=70 for 32K machines.

2. OSFLDS=3 The second parameter to adjust is the number of fields allocated for background OS/8. The minimum number is 2 for an 8K OS/8, and the maximum would be 6 for a 24K OS/8. Remember, SKED state tables also need a storage area when they are being executed. The amount needed will vary according to the number of parallel stations and the complexity of the programs. The amount of memory assigned to OS/8 and state tables is determined by the requirements of the laboratory using SUPER SKED.

For a machine with only 16k, 8k must be used for background OS/8 (OSFLDS=2). SKED and RTS/8 use one field. This leaves one field for state tables and data buffers. (Two pages of core for each data buffer for each station requested at DIALOGUE time.) If only one or two stations are used a small amount of space will be left for state tables.

In any case, OSFLDS is the parameter to be adjusted to the number of OS/8 fields. Remember you need at least 12K of background OS/8 to be able to use BATCH.

3. OSFILL=0 Some terminals require fill characters after CRLF print commands. Most terminals do not. If you have an unusual

terminal set OSFILL= to the number of fill characters required.

4. DURPAT=0 The DURPAT switch may be used to record the durations of input signals. If and only if your interface is configured so that odd input numbers (i.e. R1, R3, R5 etc.) signify the onset of a response, and if the even input lines (R2, R4, R6 etc.) are not used, then DURPAT can be set to 1. In this case, when the signal is removed from the odd input lines, the next higher even input response will be generated. That is, when the R1 signal is removed (the response switch is released) then an R2 will automatically be generated. The R2 then signals the end of an R1, and R2 can be used in the state table to signal the end of the behavior started with an R1. If you wish to use this duration feature all even responses must be reserved as the end of the response. None can be used for other purposes. To use DURPAT, set DURPAT=1.

5. BIGOUT=1 BIGOUT must be set to 1 if you have any stations with more than 12 outputs. If you have all stations with 12 or less outputs, setting BIGOUT=0 will speed the processing of outputs by SKED.

6. FAST=0 If you have an EAE, hardware multiply and divide unit, then setting FAST=1 will enable the EAE and speed up the runtime system in a number of places. If you do not have an EAE, then FAST=1, is a mistake. Your system will not work.

7. GEOF=0 Everybody except GEOFF INGLIS should have GEOF=0

8. TU10=0 If you have the TU10 magnetic tape set TU10=1. Otherwise TU10=0.

9. HIPUN=0 If you have a high speed paper tape punch, you can use it in time-share background if you disable the interrupt by the SPEC10 command 6020 and by setting HIPUN=1.

10. HIREAD=0 If you have a DEC high speed reader, it can be used in the background by setting HIREAD=1. If you have a State Systems reader, call State Systems before setting this parameter. A simple hardware modification is necessary to use the reader in the background.

11. DIGCLK=0 If you have a special hardware time-of-day-clock, similar to one originally created by Tim Elsmore, set DIGCLK=1.

12. IPB=0 If you have an inter-processor buffer set IPB=1.

You need one of 5 possible hardware clocks for this system. One and only one of the parameters 13 to 17 must be set to 1 to correspond to your clock.

13. STATSY=1 Set this to 1 if you have a State Systems clock. Set to zero otherwise.

14. CLOCK8E=0 Set this to 1 for a DK8EC crystal clock.
15. CLOCK8A=0 Set this to 1 for a PDP8A with a DKCC-AA option board.
16. DK8E=0 Set this to 1 for a DK8E programmable clock.
17. KW12=0 Set this to 1 for KW12 clock on PDP12.
18. UDC=0 If you have a UDC digital controller set this to 1.
19. TRAPDF=6677 This is a special invention to permit background programs to communicate with SKED. If you have a device that uses this IOT, you must select a new IOT for TRAPDF. Do not use 6000 or any IOT you use for other purposes. Most SKED users will not need to change this code.
20. PDP8E=0 Set this to one if running on PDP8E or PDP8A.
21. PDP12=0 Set this to 1 for PDP12
22. PDP8I=0 Set to 1 if PDP8I or PDP8L.

TASK PRIORITY

The following parameters are critical for establishing priorities of tasks in time share. They usually should not be changed.

STATE=1 Sked task

RESTOR=2 This task permits recovery of your system in the background by reading the OS/8 system handler when a CTRL B is typed.

OSSYSD=3 The system device must be priority 3.

NT=4 This parameter is for assembly purposes only. Leave it alone!!

RTSERR=NT This task reports SKED errors.

/DTAE=NT;NT=NT+1 This is a comment line since it begins with /. If the slash is removed, then the TD8E DECTape handler is inserted as a task in the system. (The TD8E file must also be assembled and loaded with the basic assembly of SHARE). To add mass-storage devices other than your SYS and DSK, you must insert them as a task here, and add the handler to the assembly of SHARE. Contact the SKED Users Group for further information.

///LPT=NT The LPT runs in the background without a special task. Do not enable this.

OSB=NT

The O/B task is usually the lowest priority and the final task.

NTASKS=NT-1 This establishes the number of tasks.

The next set of parameters defines your SYS and DSK. Your device should be set =OSSYSD. The other 3 devices should begin with a slash (/) to remove them from assembly.

DSKDEV=OSSYSD Should be left alone except under special conditions! (Call SKUG if you think you need to change this one.)

DSKUNIT=0 This defines DSK=SYS. To change this set DSKUNIT to the appropriate number included in the table in the parameter file.

Do not change anything below STOP HERE!!!! in PARAM.PA.

SUMMARY FOR THE NEW USER

1. ENABLE TIME SHARE OPTION!
  - A.) 8E - JUMPER
  - B.) 8A - SWITCH
  - C.) 12 - SEE HARDWARE PRINTS PAGE 166!
2. EDIT PARAM.PA FOR:
  - A.) HGHFLD
  - B.) OSFLDS
  - C.) PROPER SYSTEM DEVICE
  - D.) PROPER CLOCK
3. CHOSE PROPER SYSTEM TASK FROM YOUR RTS/8 SOURCE TAPE SUPPLIED BY DEC.
  - A.) EDIT IT SO THAT
    - 1.) CUR=10
    - 2.) ORIGN=200 (*200)
    - 3.) RENAME IT SYS.PA & COPY IT TO THE DISK WITH SKED.PA, EXEC.PA, AND OSB.PA.
4. THE SYSTEM DEVICE SHOULD CONTAIN AT LEAST THE FOLLOWING:
(*SUPPLIED BY SKUG.)
  - A.) EDIT.SV OR TECO.SV
  - B.) CCL.SV*
  - C.) BATCH.SV*
  - D.) ABSLDR.SV
  - E.) PALB.SV
  - F.) FOTP.SV
  - G.) DIRECT.SV
  - H.) PIP.SV
5. COPY THE FOLLOWING FILES SUPPLIED BY SKUG TO SYS:
  - A.) OSCOMP.SV, OSC002.SV, OSC003.SV
  - B.) MON.SV
  - C.) IDENT.SV, MERGE.SV
  - D.) PIPI.SV
  - E.) BATCH.SV
  - F.) CCL.SV (TYPE R CCL AFTER PUTTING A NEW CCL ON SYS)
6. SOURCE FILES (.PA) SHOULD RESIDE ON THE DEFAULT DEVICE(DSK:)
PARAM
SKED
OSB
EXEC
SYS
CHAINS
SYSTAT
7. COPY SKED.BI TO SYS:
  - A.) BE SURE THE SYSTEM HAS ABOUT 100 FREE BLOCKS
(DIR SYS:/M WILL TELL YOU.)

8. TYPE :

. SUBMIT SYS: SKED OR . SU SYS:SKED/L (FOR LPT LISTING)
A.) IF THIS IS YOUR FIRST TIME ANSWER Y TO THE QUESTION
TAKE ANSWERS FROM BATCH, AND N TO SUPPRESS BATCH ANSWERS?

B.) CHECK TO BE SURE ALL FILES
COMPILED WITH ZERO ERRORS. (DON'T WORRY ABOUT LINKS.)

9. TYPE MON

A.) THIS WILL BRING YOU INTO TIME SHARE.

1.) ENTER TIME OF DAY

B.) PLAY WITH MON COMMANDS TO CONVINCE YOURSELF
THAT EVERYTHING WORKS.

10. WHEN EVERYTHING WORKS TYPE "E" IN THE SKED MONITOR
TO RETURN YOU TO STAND ALONE OS/8.
THEN DO THE DIALOGUE CORRECTLY FOR YOUR INTERFACE.

GOOD LUCK AND HAPPY SKED!

P. S.

THE "K" COMMAND IN THE SKED MONITOR IS NORMALLY
DISABLED. TO ENABLE YOU MUST ODT LOCATION 1000 IN MON. SV
YOU MAY CHOSE TO HAVE "K" WORK ON ONLY CERTAIN BOXES
BY PLACING A ONE (1) IN THE BIT POSITION FOR THOSE
BOXES.

FOR EXAMPLE TO ENABLE "K" TO KILL ONLY THE FIRST
EIGHT (8) BOXES (BOXES 0-7)

DO:

```
GET SYS MON. SV
ODT
1000/XXXX 377
^C
SAVE SYS MON
```

IF USING MON. SV ON A PDP12 ODT LOCATION
624 FROM 6007 TO 7000 (NOP).

CHAPTER 8

DIALOGUE FOR CONFIGURING YOUR INTERFACE

Since it is impossible to match each individual user's interface specifications in a fixed system of software, the SKED runtime systems must be flexible enough to permit modifications which will adapt to any particular interface configuration, while maintaining a high degree of language compatibility among different installations. This is accomplished through the system builder Dialogue, in which the user interacts with the program to assign input and output buffers to stations as they best fit his requirements. The Dialogue, which starts at 10200, asks the user for the IOT (input/output transfer) instructions of the input and output interface cards. The user carefully proceeds through the Dialogue, and when it is completed, the configured runtime system is saved and the process need never be repeated, until the user changes his interface or updates his SKED software.

The command `LOAD SKED=10200/G` will initiate the Dialogue. The command may be given directly to the OS/8 Monitor, or it may be contained in a BATCH job for automatic loading and assembly of the system. The answers to the questions of the Dialogue may be included sequentially in a BATCH job (see Chapter 8). They are preceded by a `/` character in the file, indicating their status as comments, but if the following questions so direct, they may be read directly by the Dialogue, saving time and the monotony of answering the questions manually. An example of a BATCH job and its execution is included as subsection H.

A. When the Dialogue starts, it first asks

INPUT FROM BATCH? (Y OR N)

to which the user must reply according to whether or not he uses the BATCH assembly option just described. If the user wishes to assemble the system using the standard answers included in the BATCH file, type Y. If Y is typed for the first question, then SKED will ask:

SUPPRESS BATCH ANSWERS? (Y OR N)

If Y is typed the Dialogue will be entirely invisible and the next print out should be `/END OF DIALOGUE`. If N is typed then the entire Dialogue is printed.

If the user wishes to give answers to the Dialogue different from those in BATCH, then typing N to the first question will result in SKED printing each question and waiting for a reply from the terminal. With this option, the batched answers will be printed as comments following the end of the Dialogue. After this, the version number of SUPERSKED is typed out and the main body of the Dialogue is begun. At this point, BATCH will begin inserting its answers automatically if the previous two answers were Y and N. If not, the user manually answers the question, remembering to give regular decimal answers to all questions except those that ask for SPECIO commands, masks, and buffer

IOT's, which must be in Octal

B. SPECIO. SPECIO commands are special IOT instructions to manipulate certain nonstandard devices. When SPECIO commands are specified, they will be executed whenever the background is entered. A user who uses the instruction 6501 to send a pulse type 1 to device 50 will respond:

HOW MANY SPECIO COMMANDS? 1

SPECIO: 6501

C. Input Buffers and Stations. The user has by this point decided how he wants to make use of the input buffers in his interface. Each buffer contains 12 input lines, and multiple buffers are possible to give sufficient inputs from as many as 12 stations. There are several ways of assigning input buffers to stations. First, there could be one buffer per station, and this station could control one or more than one experimental chambers, depending on how many responses are to be monitored in each box. Second, each buffer can be split up between two or more stations. Third, in cases where twelve inputs are not enough, more than one buffer can be assigned to each station by the R MULT option. Fourth, stations can be yoked, that is, all the inputs in a buffer will be shared by two stations. The only restriction on allocation of buffer resources is that no station can be serviced by fragments of two buffers, if either of the buffers also services another station. If ten of the 12 inputs in a buffer are used in a station, the remaining two cannot be used in combination with inputs from another buffer to make up a station. Each input card from State Systems consists of three buffers. Each buffer responds to a particular IOT instruction, partly made up of the IOT type (1, 3 or 5), and partly determined by the device code wired into the input card. All IOT's start with 6, so the instruction 6101 will be the "read" IOT for one of the buffers on input card I10 (with device code 10). The following example shows how the Dialogue acquires this information.

HOW MANY INPUT BUFFERS? 6

HOW MANY STATIONS? 11

BUF. NO.	READ	STATIONS	R MULT	
0	6101	:0	0	/The user types "space"
1	6103	:0	1	/Between entries on
2	6105	:1	0	/each line.
3	6111	:2,3,4	0	
4	6113	:5,6,7	0	
5	6115	:8,9,10	0	

In the example above, there are 2 input cards containing 6 buffers. These are to be divided between 11 stations. Station 0 controls a monkey experiment which uses 22 pushbuttons as the manipulanda to be monitored. This requires two buffers, so the multiplier, R MULT, was set to 1 for the second buffer in Station

O. This causes Buffer 0 to contain R1-R12 for this station while the next Buffer contains R13-R24. If another buffer were to be added for this station, the R MULT would be set to 2, and so on. Station 1 requires 10 inputs for an experiment on response variability. There are 2 inputs unused in this buffer, since no other station requires only 2 inputs. The remaining 9 stations require 4 inputs apiece for a matching-to-sample experiment with pigeons. Therefore they can be assigned to the three buffers of an input card, giving each station R1 through R4.

D. Masking Inputs. Interface inputs in each buffer are assigned to stations by masking. A four-digit octal mask is made, so that its binary equivalent corresponds to the relative position of the assigned inputs for that station, using the rightmost bit as the first input in the buffer. For example, a buffer which is split between two stations assigns the 6 rightmost bits to the first station, so the binary representation of the buffer for that station would look like:

000 000 111 111

And its octal mask would be

0 0 7 7

In the example from Section C, the user would answer the dialogue this way:

ENTER OCTAL MASK FOR EACH STATION

STATION NO. 0 7777

STATION NO. 1 1777

STATION NO. 2 0017

STATION NO. 3 0360

STATION NO. 4 7400

STATION NO. 5 0017

STATION NO. 6 0360

STATION NO. 7 7400

STATION NO. 8 0017

STATION NO. 9 0360

STATION NO. 10 7400

Station 0 masks the entire first buffer, and since it requested a second buffer with R MULT, the whole second buffer is automatically masked. Since station 1 uses only ten inputs, the rightmost ten bits are masked, yielding the octal word 1777. Each of the next three buffers is divided like this:

000 000 001 111 = 0017

000 011 110 000 = 0360

111 100 000 000 = 7400

E. Output buffers and IOT's. The process of allocating buffer space to stations and specifying IOT instructions must also be completed for output buffers, similar to the way it was done for inputs. The Dialogue format is slightly different, however. The first question asked is the number of buffers, which the user must answer, remembering that there are three buffers per State Systems output card. The user in our example answers the question:

HOW MANY OUTPUT BUFFERS? 8

since he has three output cards, one of which has an unused buffer, making 8 buffers. The IOT's for the buffers must be given, and as before, these instructions are partly made up of the "6" (to identify it as an I/O instruction), partly of the device code for the card on which the buffer resides, and partly of the type of I/O pulse to be used (1, 2, or 4). Our mythical user checks the device codes of his output cards and answers the Dialogue this way:

LIST IOTS FOR EACH BUFFER

BUFFER NO. 0 IOT = 6121
 BUFFER NO. 1 IOT = 6122
 BUFFER NO. 2 IOT = 6124
 BUFFER NO. 3 IOT = 6131
 BUFFER NO. 4 IOT = 6132
 BUFFER NO. 5 IOT = 6134
 BUFFER NO. 6 IOT = 6141
 BUFFER NO. 7 IOT = 617/6142

Reminder: A single rubout erases the entire answer previous to its occurrence. Typing CTRL C at any point in the Dialogue restarts the Dialogue at the beginning.

F. Masking Outputs. The user in our example needs 28 outputs for his monkey experiment in Station 0, 3 outputs for the response variability study, and 5 outputs for each of the matching-to-sample stations. This requires three buffers for Station 0, since multiple outputs may only be assigned in whole buffer units. Stations 1 and 2 can share one buffer, and the user connects his box wiring into the relay card in such a way that either of these stations can be assigned an extra two outputs should the need arise. All the rest of the buffers must be shared by two stations. Once again, the user plans ahead in his wiring so that each of these stations has a spare output. He answers the rest of the Dialogue in this way, giving the buffer number, mask, number of buffers, and maximum number of outputs for that station.

LIST BUFFER NO. AND MASK

STATION 0 IN BUFFER: 0

MASK: 7777

NUMBER OF BUFFERS FOR THIS BOX = 3 /this assigns buffers 0, 1, and 2 to Station 0.

MAXIMUM NO. OF ON CHANNELS: 28
STATION 1   IN BUFFER: 3
MASK: 0007
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 3
STATION 2   IN BUFFER: 3
MASK: 1740
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 3   IN BUFFER: 4
MASK: 0037
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 4   IN BUFFER: 4
MASK: 3700
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 5   IN BUFFER: 5
MASK: 0037
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 6   IN BUFFER: 5
MASK: 3700
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 7   IN BUFFER: 6
MASK: 0037
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 8   IN BUFFER: 6
MASK: 3700
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 9   IN BUFFER: 7
MASK: 0037
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5
STATION 10   IN BUFFER: 7
MASK: 3700
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 5

G. Saving SKED. After the Dialogue is completed, the next command the user should enter or the next command in the BATCH job should be

.SA SYS SKED 0-6177

This creates SKED.SV which is then loaded with EXEC, OS/8, and SYS to make SHARE.SV.

H. BATCH JOB. The following file, SKED.BI, is a BATCH control file with Dialogue answers for 2 stations:

```
$JOB FOR ASSEMBLING SHARE
.COM SYS:SKED<PARAM. PA, SKED
/NO COMMENTS ALLOWED BETWEEN .LOAD SKED AND NEXT $ SIGN
/EXCEPT ANSWERS TO DIALOGUE!
.LOAD SKED=10200/G
```

```
/1
```

```
/6117
```

```
/1
```

```
/2
```

```
/6101 0.1 0
```

```
/0077
```

```
/7700
```

```
/2
```

```
/6111
```

```
/6112
```

```
/0
```

```
/7777
```

```
/1
```

```
/12
```

```
/1
```

```
/7777
```

```
/1
```

```
/12
```

```
$
```

```
/END OF DIALOGUE
```

```
.SA SYS SKED 0-6177
```

```
.COM .SYS:EXEC<PARAM. PA, EXEC
```

```
.COM SYS:OS8<PARAM. PA, OS8
```

```
.COM SYS:SYS<PARAM, SYS
```

```
.R ABSLDR
```

```
*SKED. SV/I
```

```
*EXEC, OS8, SYS$
```

```
.SA SYS SHARE ; 177
```

```
.DEL SYS:SKED. SV
```

```
.COM SYS:SYSTAT<PARAM, SYSTAT, SKED
```

```
.LOAD SYSTAT=12000
```

```
.SA SYS:SYSTAT=6403
```

```
$END
```

The command

```
.SU SKED.BI
```

will cause the following on the user's terminal:

```
$ JOB FOR ASSEMBLING SHARE
```

```
.COM SYS:SKED<PARAM, SKED
```

```
ERRORS DETECTED: 0
```

```
LINKS GENERATED: 128
```

```
/NO COMMENTS .ALLOWED BETWEEN .LOAD SKED AND NEXT $ SIGN
```

```
/EXCEPT ANSWERS TO DIALOGUE
```

```
.LOAD SKED=10200/G
```

```
INPUT FROM BATCH? (Y OR N) [Y]
```

```

SUPPRESS BATCH ANSWERS? (Y OR N) [N] SUPER SKED VERSION: 5
HOW MANY SPECIO COMMANDS? 1
SPECIO: 6117
HOW MANY INPUT BUFFERS? 1
HOW MANY STATIONS? 2
BUF NO.  READ  STATION  R MULT
      0   :6101   :    0,1   :0
ENTER OCTAL MASK FOR EACH STATION
STATION NO. 0    0077
STATION NO. 1    7700
HOW MANY OUTPUT BUFFERS? 2
LIST IOTS FOR EACH BUFFER.
BUFFER NO. 0  IOT=6111
BUFFER NO. 1  IOT=6112
LIST BUFFER NO. AND MASK
STATION 0  IN BUFFER: 0
MASK: 7777
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO OF ON CHANNELS: 12
STATION 1  IN BUFFER: 1
MASK: 7777
NUMBER OF BUFFERS FOR THIS BOX = 1
MAXIMUM NO. OF ON CHANNELS: 12
/END OF DIALOGUE
.SA SYS SKED 0-6177
.COM SYS: EXEC<PARAM, EXEC
ERRORS DETECTED: 0
LINKS GENERATED: 0
.COM SYS: OSB<PARAM, OSB
ERRORS DETECTED: 0
LINKS GENERATED: 13
.COM SYS: SYS<PARAM, SYS
ERRORS DETECTED: 0
LINKS GENERATED: 0
.R ABSLDR
*SKED. SV/I
*EXEC, OSB, SYS$
.SA SYS SHARE ;177
.DEL SYS: SKED. SV
FILES DELETED
SKED. SV
.COMPILE SYS: SYSTAT<PARAM-NB, SYSTAT, SKED
ERRORS DETECTED: 0
LINKS GENERATED: 195
.LOAD SYSTAT=12000
.SA SYS SYSTAT=6403
$END

```

REFERENCES

- Anger, D. The dependence of interresponse times upon the relative reinforcement of different interresponse times. The Journal of Experimental Psychology, 1956, 52, 145-161.
- Berryman, R. and Nevin, J.A. Interlocking schedules of reinforcement. The Journal of the Experimental Analysis of Behavior 1962, 5, 213-223.
- Brandauer, C.M. The effects of uniform probabilities of reinforcement upon the response rate of the pigeon. Unpublished Doctoral Dissertation, Columbia University, 1958.
- Catania, A.C. and Reynolds, G.S. A quantitative analysis of the responding maintained by interval schedules of reinforcement. The Journal of the Experimental Analysis of Behavior, 1968, 11, 327-383.
- Farmer, J. Properties of behavior under random interval reinforcement schedules. The Journal of the Experimental Analysis of Behavior, 1963, 6, 607-616.
- Ferster, C.B. and Skinner, B.F. Schedules of reinforcement. Englewood Cliffs, N.J., Prentice Hall Inc., 1957.
- Hodos, W. Progressive ratio as a measure of reward strength. Science, 1961, 134, 943-944.

References

McCluskey, E. J., Jr., Introduction To The Theory Of Switching Circuits. New York: McGraw-Hill Book Company, 1965.

Mealy, G. H., A method for synthesizing sequential circuits. Bell System Tech. J., 1955, 34, 1045-1079.

Mechner, F. Sequential dependencies of the lengths of consecutive response runs. The Journal of the Experimental Analysis of Behavior, 1958, 1, 229-233.

Rider, D. P. Interlocking schedules: the relationship between response and time requirements. The Journal of the Experimental Analysis of Behavior, 1977, 28, 41-46.

Moore, E. F., Gedanken-Experiments on sequential machines. Automata Studies, Annals of Mathematical Studies, 1956, 34, 129-153. Schoenfeld, W. N., Cumming, W. W. and Hearst, E. On the classification of reinforcement schedules. Proceedings of the National Academy of Sciences, 1956, 42, 563-570.

Sidman, M. Two temporal parameters in the maintenance of avoidance behavior by the white rat. The Journal of Comparative and Physiological Psychology, 1953, 46, 253-261.

Snapper, A. G., Knapp, J. Z., & Kushner, H. K. Mathematical description of schedules of reinforcement. In W. N. Schoenfeld (Ed.) The Theory of Reinforcement Schedules. New York: Appleton-Century-Crofts, 1970, 247-275.

Weiss, B. Amphetamine and the temporal structure of behavior. In E. Costa & S. Garattini, (Eds.) Amphetamines and Related Compounds. New York: Raven Press, 1970.

Weiss, B. (Ed.) Digital Computers in the Behavior Laboratory. New York, Appleton-Century-Crofts, New York: 1973.

SUPER S K E D Statement Summary

c=Constant only
v=Variable only
n=Constant or variable

STATEMENT	SYNTAX	EFFECT

NONEXECUTABLE FUNCTIONS

STATE SET LABEL	S. S. c,	Define start of State Set.
STATE LABEL	Sc,	Define start of State.
COUNTERS	C=c	Allocate space for the C array where c represents the number of counters needed.
DIM(ension)	DIM v=c	Allocate space for the A-Z array (but not C) where c is a constant representing the size of the array.
LIST	LIST v=c1, c2, c3	Define initial value of v(0), v(1), v(2) and dimension v to size of list.

INPUT FUNCTIONS

INPUTS(responses)	n Rc	Sense n responses on input line c and execute output functions when n has been satisfied.
START	START	Allow the state table to be started from the monitor with the "s" command. This is the same as R1200.
VARIABLE TIME	v T	Reduce v every .01" (wait n X .01"), when v equals zero execute output functions.
Z (as input)	n Zc	Sense n Zc pulses and execute output functions when n is satisfied. c must be in the range of 1 - 12.

OUTPUT FUNCTIONS

ON	: ONn : ONn1, n2, n3	Turn on output bit n.
OFF	: OFFn : OFFn1, n2, n3	Turn off output bit n.
Z(as output)	: Zc	Issue a Zc which may be recieved as an input function in a parallel state set or in the resulting state. c must be a constant in the range of 1 to 12.
SET	: SET v=n	Replace the value of v with n.
ADD	: ADD v	Add one (1) to value of v. Short form of SET v=v+1.
SUB	: SUB v	Subtract one (1) from the value of v. Short form of SET v=v-1.
RAND	: RAND v=v1	Replace v with a value from array v1. The values are selected randomly without replacement from array v1.
LIST	: LIST v=v1(v2)	Replace v with a value from list v1 according to the index variable v2, then add one to v2. When v2 reaches the last number in the list it is reset to 0.
TIME	: TIME v	Return time of day to v(0),v(1),v(2).
OPEN	: OPEN	Set up internal parameters necessary for writing to a mass storage device. Open the device file specified in SKED Monitor with the "m" command.
CLOSE	: CLOSE	Write out to data file all numbers currently held in an unfilled data buffer.
WRITE	: WRITE n, n1	Write n with a code of n1 to the open data file. n1 must be in the range of 0-15, n1 outside this range will allow n to be as large as 16777215.
RECORD	: RECORD n	Write with a code n the time since the last RECORD to the open data file. n must be in the range of 0-15, or no code will be written. If n is greater than 15 then RECORD n will write a full 24 bit inter-event time, rather than 20 bits.
CALL	: CALL c[n1,n1]	Execute machine language subroutine c with arguments n1 & n2.

DECISION FUNCTIONS

IF	: IF v=n; output functions	Execute output functions if expression is true, otherwise goes to default line or to beginning of state if no default.
THEN	: THEN	Comment only! No effect.
ELSE	: ELSE	Comment only! No effect.
ALWAYS	: ALWAYS	Comment only! No effect.
FOR	: FOR v=n1, n2, n3	Repeat statements following the FOR up to NEXT or a transition, starting at v=n1, incrementing v by n2, and terminating when v>n3.
BREAK	: BREAK	Terminate execution of FOR loop.
NEXT	: NEXT	Specify end of FOR loop.
SUBZGO	: SUBZGO v	Subtract one (1) from the value of v, if v equals zero then execute the output functions that follow. Short form of SET v=v-1; IF v=0.
WITH	: WITH P=n	Test the internal probability generator and with a probability of n out of 1000 execute the output functions. n may be a constant or variable in the range of 1-999.

TRANSFER FUNCTIONS

STOP	--->STOP	Stop execution of the state table and allows a new program to be loaded in to that station.
STATE transfer	--->SX	Transfer control to start of current state. Input functions are not reset.
STATE transfer	--->Sc	Transfer control to state c. Input functions which are constant (i.e. 10", 5R1, 10Z4) are set to their stated value. This is true even if the state transfers to itself!

MON. SV

CALLING COMMANDS:

.MON   /ALL! SWITCHES ALLOWED
.LOAD DEV:STATETBL.ST OR STATETBL.SM,.....

SWITCHES:

/B   TAKE ALL INPUT FOR MON FROM BATCH FILE UNTIL "\$"
/C   TURN ON CONFIRM OPTION (NOT USEFUL WITH /B)
/G   LOAD FILE CURRENTLY IN COMMAND DECODER (PASSED BY EXEC & LOAD)
/L   REQUEST INPUT OF TIME OF DAY (PASSED WHEN TIME SHARE LOADED)
/M   PASS BOXES LOADED TO "M" COMMAND (EXECUTE "M" IF IN BATCH)
/S   FORCE BACKGROUND DATE TO BE SYSTEM DATE
/1/2/3.../Y/Z   LOAD BOXES 1-9 (Y=10, Z=11)

SKED MONITOR KEYBOARD COMMANDS   (SKED PROMPT IS %)

CHAR	FUNCTION
A	ABORT ONE BOX
B	BOXES RUNNING
C	RETURN TO OS/8
D	ADVANCE BATCH ON END OF SESSION FOR BOXES SPECIFIED
E	EXIT TO STAND ALONE OS/8
F	FINISH BATCH
G	R12 TO ALL STATIONS
H	ADVANCE BATCH ON CALL 2 FOR BOXES SPECIFIED
I	LOAD SKED STATE TABLE
J	NOT DEFINED
K	KILL ALL STATIONS
L	NOT DEFINED
M	OPEN OUTPUT FILES
N	NOT DEFINED
O	OPEN AND DUMP COUNTERS FROM SPECIFIED STATIONS
P	PRINT COUNTERS FROM SPECIFIED STATIONS
Q	MODIFY VARIABLE
R	SIMULATE RESPONSE FROM KEYBOARD
S	START SPECIFIED STATIONS (R1200)
T	NOT DEFINED
U	TERMINATE BATCH AND STORE POINTERS
V	GET TIME OF DAY
W	SET TIME OF DAY
X	DISPLAY ALL VARIABLES FOR SPECIFIED STATION (EXCEPT "C")
Y	NOT DEFINED
Z	MODIFY HEADERS
/	SET CONFIRM OPTION
CTRL C	RETURN TO OS/8