DECUS
PROGRAM LIBRARY

DECUS NO.	8-465
TITLE	THE SKED SOFTWARE SYSTEM
AUTHOR	Dr. A. G. Snapper, Psychology Research Lab. Franklin D. Roosevelt V. A. Hospital, Montrose, New York
COMPANY	Contributions and Submittal by: Andrew Walker Digital Equipment Corporation Maynard, Massachusetts
DATE	October 8, 1971
SOURCE LANGUAGE	

ATTENTION

This is a USER program. Other than requiring that it conform to submittal and review standards, no quality control has been imposed upon this program by DECUS.

The DECUS Program Library is a clearing house only; it does not generate or test programs. No warranty, express or implied, is made by the contributor, Digital Equipment Computer Users Society or Digital Equipment Corporation as to the accuracy or functioning of the program or related material, and no responsibility is assumed by these parties in connection therewith.

ERRATA SHEET

DECUS NO. 8-465

THE SKED SOFTWARE SYSTEM

PAGE	SECTION	CHANGE
2-4	2.2.1	Last sentence should read: The line must end with a comma.
2-4	2.2.2	Last sentence should end: ...by a comma, which terminates the line.
3-2	3.1.1	Paragraph 2 should be replaced with: 2. A SKED source tape must always be terminated with a dollar sign ($).
3-6	3.1.5	Add the following sentence to Step 4: Then type a character (space) to start the read-in sequence. Add the following sentence to Step 7: ...alternatives as in Step 5. Once an error has been detected, punched paper tape output will cease....
3-10 to 3-12	3.3	The following error codes should be changed: 753 to 760 7Ø5 707 624 623 1355 1353 37Ø2 37ØØ 232Ø 2316 2346 2344 255Ø 2546 2652 2654 266Ø 2662 37Ø2 37ØØ

8/30/72

ERRATA

DECUS 8-465
THE SKED SOFTWARE SYSTEM

Page	Section	Change
2-4	2.2.1	Last sentence should read: The line must end with a comma.
2-4	2.2.2	Last sentence should end: ...by a comma, which terminates the line.
3-2	3.1.1	Paragraph 2 should be replaced with: 2. A SKED source tape must always be terminated with a dollar sign ($).
3-6	3.1.5	Step 4 should read: Load the SKED source tape onto the appropriate input device, turn the device on and press the start key. If the low speed reader is being used, type a space. Add the following sentence to Step 7. ...alternatives as in Step 5. Once an error has been detected, punched paper tape output will cease....
3-10 to 3-12	3.3	The following error codes should be changed: 753 to 760 705 707 624 623 1355 1353 3702 3700 2320 2316 2346 2344 2550 2546 2652 2654 2660 2662 3702 3700
5-22	5.2.3.2	Since the UDCB can run on both a PDP-8e and a PDP-12, the first question actually asked is to define which clock is used, the KW-12 or DK8-EP.
Table of Contents	Referenced	Add Appendix C - Implementation NOtes

5.5 DR8E SYSTEM BUILDER

The DR8E System Builder is a modified version of the UDC8 builder (Section 5.2.3). The dialogue to build a system is the same in both cases, except that the DR8E Builder assumes that the DK8-EP clock is to be used. Thus, to decide on the correct answers, refer to the explanation of the dialogue in Section 5.2.3.

Note that each slot in the DR8E (Ø-7) has both inputs and outputs while each slot in the UDCF has either inputs or outputs. Thus, for a system with 36 inputs and outputs, the DR8E slots would be 0,1,2 for both inputs and outputs while with the UDC8 there would be no overlap; i.e., 0,1,2 for inputs and 3,4,5 for outputs.

Appendix C -

Implementation Notes

DECUS 8-465

The SKED Software System

1. Making SKED Modifications

a) Compiler - This comes in two parts and must be assembled together, with part 1 first.

b) Run-Time-System - This comes in two parts, the RTS proper which is used in all cases and the hardware dependent I/O handlers. Handlers are available from DECUS for the UDC8, DR8E and PDP-12 digital I/O. If these are modified, or a new device handler is written (Section 5.1 and 5.2), the RTS and the device handler should be assembled together to produce a new Run-Time-System binary.

2. Writing SKED Programs

Transition Ordering - If there are several possible transitions to exit from a given state, they must be in the order of R transitions first, then Time, then Z. Example:

```
S.S.1,
   S1,
      R1: ON 1 ---> S4
      1": ON 2 ---> S2
     3Z2: OFF N---> S3
```

If this order is not followed, the program will compile correctly, but the RTS will detect a checksum error when trying to load the state table.

APPENDIX A: Notes on the Format and Interpretation of a Compiled State Program

INTRODUCTION

This appendix contains notes explaining many of the inner workings of the SKED R.T.S., particularly the interpretation of the compiled state program. A working knowledge of the SKED state language, of PDP-8 machine language and of octal arithmetic are prerequisite for the understanding of this material.

GENERAL NOTES

A compiled state program consists of lists of calls ("state tables") to subroutines in the R.T.S. proper preceded by a small group of pointers and constants (the pointer table). The subroutines called execute the component functions involved in determining whether inputs from the clock or experiment stations should affect the state program and, if so, how (see TABLE B).

Some notes regarding terminology:

> The terms "pointer", "pointer word", "pointer cell" will all be used in this appendix to describe a cell in memory that contains a core address of another memory cell that has some interesting characeristic. Usually the address will be a starting address of a particular series of commands or subroutine calls.
>
> The meaning of the term "state table" as it is used in this appendix is not consistent with its meaning in other parts of the SKED USER DOCUMENT. Elsewhere, "state table" often means the entire compiled state program, the binary output of the SKED compiler. Here the term refers to a small part of the compiled program, specifically the series of subroutine calls that describe the transitions of any one state. It is the binary form of all of the transition expressions in one state expression.

POINTER TABLE FORMAT

The pointer table at the beginning of the compiled state program consists of a lead word that contains the two's complement of the number of state sets in the program, followed by groups of four cells per state set that act as indicators of the status of each state set.

Within each four-word group, the first three words are pointers and the last word contains the two's complement of the number of the state active in that state set. Each pointer cell indicates the subroutines that will be called if one of the three types of events occurs AND the event is capable of causing a state transition. Each pointer position within the pointer word group is related to one of the event types. Thus the first pointer specifies where the clock contingency processing routine will begin; the second pointer speaks for the response checking routines; the third points to Z checking routines.

Any time a clock, response or Z event occurs, the R.T.S. will examine the appropriate cell in each group of pointers to determine whether the event might cause a transition in a given state set. When the event will definitely not be able to cause a transition, the cell will contain zero. Otherwise, the cell will contain the starting address of the group of subroutine calls (in the state table) that will determine whether the event will cause a transition and, if so, what else will happen.

Figure 1 illustrates the pointer table for a state program with two state sets. In the first state set the currently active state is STATE 1 which can only be left after some sort of response event(s). In the second state set STATE 1 is also active. Either a clock or Z event can cause transition to the next state.

FIGURE 1: EXAMPLE POINTER TABLE FOR A STATE PROGRAM WITH TWO STATE SETS

LOCATION	CONTENTS	FUNCTION
3000	7776	-2; the number of state sets
3001	0000	S.S.1 clock pointer
3002	3020	S.S.1 R pointer
3003	0000	S.S.1 Z pointer
3004	7777	S.S.1 STATE NUMBER (-1)
3005	3063	S.S.2 clock pointer
3006	0000	S.S.2 R pointer
3007	3067	S.S.2 Z pointer
3010	7777	S.S.2 STATE NUMBER (-1)

While state activity in any state set is determined by the contents of the four status words in the pointer table that pertains to the state set, state change is accomplished through modification of the four cells so that the three pointer words point to parts of the state table that describe the state being activated and the last cell in the group contains the two's complement of the number of the new state.

STATE TABLE FORMAT

A state table is a description of the transitions in a given SKED state expression. It is of variable length, consisting of a series of calls to subroutines in the R.T.S. that execute the functions involved with state transition and the various output operations.

The generalized state table has two main sections. They will be described here as the "lead section" and the "transition section". Briefly, the lead section consists of a call to the subroutine CHGSTS and a group of cells called "event counters" which aid in tallying events when a repetition of events is the criterion for triggering a transition. The transition section is comprised completely of subroutine calls and has three subsections: one to execute transitions due to elapsed time, one for R events, and one for Z events. Figure 2, below, is a format diagram of the state table.

FIGURE 2: FORMAT DIAGRAM OF GENERAL STATE TABLE

JMS I CHGER

(ARGUMENTS)

lead section

EVENT CTR

AREA

Subroutine calls

for transition

transition section

execution

NOTES ON THE LEAD SECTION

Event Counter Format

The format of the event counter area of a state table consists of an alternation of cells in which counting takes place, and cells in which are kept the initial values of event counters. Initial values are two's complement expressions of the number of events to be counted. For example, if, for a given transition, five events of a given type must occur, then a pair of event counter cells must be compiled at the beginning of the state table for that state. The first cell will contain the initial value 7773, which is the two's complement of five. The second cell will be initialized with this value whenever this state becomes active and will contain the currently accumulated value during the event counting procedure.

State Change Subroutine Call

At the front of the lead section is a call to the subroutine CHGSTS with the following format:

```
JMS I CHGER (CHGER is a page zero pointer)
CLOCK POINTER
R-POINTERS
Z-POINTER
-STATE NUMBER
NUMBER OF EVENT CTR CELLS:  -2N OR ZERO
EVENT CTR 1 INIT*         STATE TABLE EVENT COUNTER AREA
EVENT CTR 1 TALLY CELL    2N EVENT CTR CELLS
EVENT CTR 2 INIT
EVENT CTR 2 TALLY CELL
```

*"Event ctr init." indicates the cell that is used to initialize the interval counter; as distinguished from the event counter itself, which follows the initialization cell, in which the actual counting takes place, and which is indicated by "event ctr".

The subroutine CHGSTS executes two tasks which accomplish a state transition:

a) The status data in the four cells immediately following the call are transferred to the POINTER TABLE. This means that the next clock or response event will cause examination of transition contingencies described in the state table immediately following the JMS I CHGER.

b) The values in the event counter initialization cells are transferred to the tally cells. Thus, the event counters are "reset". Note that the fifth cell after the call to CHGTS contains a value indicating the number of event counters in the state. CHSTS uses this value to guide it in the update procedure. If the value is zero, there will be no counters and, obviously, no need to update.

State Set Scan Loop

When a clock interrupt occurs, all active boxes are scanned to see if a time contingency transition is performed. In addition, the digital I/O is also scanned to see if any new responses have occurred since the last clock tick. The routine to perform the scan is BOXSCN, which all state sets and states in an active box for possible transitions. When all states in a state set have been checked, control returns to STLOOP (via JMP I LOOPER) to check for more state sets to be scanned.

Because a state table may often be loaded across a PDP-8 page boundary, the possibility exists that a JMP I .+1 could be the last word on a page. The routine PPLUS1 is used to execute a page independent effective JMP I .+1 to avoid the problem.

THE TRANSITION SECTION

This section describes the state table format for each of the functions involved directly in transition execution. Examples are given of schedules requiring each function. At the end of this section a state table containing more functions than are usually required is presented to further illustrate the format.

TIME INPUT

If elapsed time can cause a transition in a particular state set, the first cell of the pointer group pertaining to that state in the pointer table will contain the address of a call to the subroutine KCTR. This subroutine call has the form:

```
JMS I KTGO  (KTGO is a pointer in core page zero)
ADDRESS.
```

KCTR counts clock interrupts (intervals of 10 milliseconds) until enough have been sensed to accumulate to the time designated by the SKED transition expression. The counting of clock intervals takes place in two event counter cells allowing a maximum of $2^{24}-1$ interrupts to be tallied. The argument of the call, represented by "ADDRESS", is the address of the low order interval counter cell allocated for use in this counting operation. Following the argument is the series of subroutine calls that will be executed only when the stated time has elapsed. If the interval has not elapsed after an increment has been executed by KCTR, the subroutine exits to the pointer section scan procedure so that the clock contingency for the next state set can be examined. Figure 3 shows an example pointer table and partial state table for a state program with one state, one transition. Note that for each state there will never be more than one call to the subroutine KCTR.

FIGURE 3: PARTIAL LISTING OF A COMPILED STATE PROGRAM WITH ONE STATE

Part A: The State Program

```
/SAMPLE SKED STATE PROGRAM
S.S.1,
S1,
    .Ø5--> S1
```

Part B: Partial listing of compiled state program. To illustrate the format of the state table with a call to KCTR.

LOCATION	CONTENTS	FUNCTION
3ØØØ	7777	-1; the 2's complement number of state sets.
3ØØ1	3Ø16	S.S.1 pointer to interval counting process.
3ØØ2	ØØØØ	S.S.1 R pointer
3ØØ3	ØØØØ	S.S.1 Z pointer
3ØØ4	7777	S.S.1 current state #.
3ØØ5	4564	JMS I CHGER for state 1.
3ØØ6	3Ø16	pointer to clock subroutine.
3ØØ7	ØØØØ	R pointer
3Ø1Ø	ØØØØ	Z pointer
3Ø11	7777	state number
	7776	-2; number of event ctr. cells
3Ø12	7773	-5; low order interval ctr. init.
3Ø13	7773	-5; low order interval ctr. tally
3Ø14	7777	high order interval counter; init.
3Ø15	7777	high order interval counter; tally
3Ø16	4573	JMS I KTGO; call to clock process.
3Ø17	3Ø13	pointer to event counter.

RESPONSE INPUTS

When the computer senses one or more response events, the RTS sorts the response data into tables that show which responses affect which boxes. The tabular information is expressed in terms of a twelve-bit word for each box in the system. In each word a bit is set for every response channel on which an event has occurred. An example of such a table follows:

Channel #	12	11	10	9	8	7	6	5	4	3	2	1	
	0	0	0	0	0	0	0	0	0	1	1	1	Status Word for Box Ø

0 0 0 0 0 0 1 1 1 0 0 0	Status Word for Box 1
0 0 0 0 1 1 1 0 0 0 0 0	Status Word for Box 2

In this example of a table for a three-box system, the 1's represent channels where events have occurred, Ø's represent momentarily silent channels. Thus, in Box Ø, events have occurred on channels 3, 2, 1; in Box 1, events have occurred on channels 6, 5, 4; etc.

When this table has been updated, the R.T.S. starts the scan procedure in which, for each box, the proper transitions are activated, according to the tabular data.

When a response can cause a transition in a particular state set, the second cell in the pointer group for that state set will contain the starting address of a chain of subroutine calls that describe which responses (e.g., R1 or R2 or R11, etc.) will cause a transition and what functions will operate during the execution of the transition. The first subroutine call in the chain will have the following format:

```
JMS I INCHKR  (INCHKR is a page Ø pointer)
CHANNEL WORD
EVENT CTR PTR
NEXT CALL PTR
```

This is a call to the subroutine INNCHK, which has a three-part operation aimed towards determining whether a response will be acted upon.

The first step is a matching operation. CHANNEL WORD, the first argument in the subroutine call, is a twelve-bit code indicating a response channel that, when active can cause a transition. This code has a format similar to that of the response status tables, but will always have only one bit set and the other eleven bits equal to zero. Thus, each CHANNEL WORD indicates one response contingency. The matching operation compares (with a logical AND) the CHANNEL WORD with an entry in the RTS tables that show which response channels are receiving signals at any given time. When the comparison shows that the response channel indicated by CHANNEL WORD has recently received a signal, the process continues to Step 2. Otherwise the process skips to Step 3.

Step 2 executes event counting when that is necessary. The argument designated EVENT CTR. PTR. is examined. When no counting is necessary, the value will be zero, and the computer will jump to the subroutine call immediately following the last argument. Thus, the contingency checking process will be over and a transition will be executed.

When event counting is necessary (e.g., when the event is something like 3R1 EVENT CTR. PTR. will be the address of the cell in the event counter area in which the tally of the occurrences of the event is being kept. The counter cell will be incremented. If it goes to zero, meaning that the requisite number of responses for a transition has occurred, the computer will jump to the subroutine call immediately following the last argument - the transition will be executed. When the tally cell does not go to zero, Step 3 will occur.

Step 3 of the response checking subroutine is executed when the contingency just checked has not occurred at all or not enough times, so that no transition can occur. Step 3 examines the last argument of the INNCHK subroutine call. When there are no more response contingencies in the state, this argument will be zero in value and the response contingency checking process will end. If there are more response contingencies in the state, there will be occurrences of sets of three arguments further on in the state table. These sets of arguments will serve the same function as the three after the JMS I INCHKR. The last argument will contain the address of the beginning of the next set of arguments when there are more contingencies to be checked. The program will find this set of arguments and start through the three-step process with them, as it did with the set before - as explained above.

Figure 4 illustrates the interweaving of sets of response contingency processing arguments into a state table. Part A of the figure is part of a state program and Part B shows the corresponding state tables.

FIGURE 4: PART A

```
S.S.1,
S1,
     R1:  on 1-->S1
     1" -------->S2
     R2:  on 2-->S1
    3R3 -------->S1
     :
     :
```

FIGURE 4: PART B

Octal listing of compiler output for the above state program:

```
Location   Content   Comment
------------------------------------------------------------------------
3000
3001        7777     Two's complement of number of state sets
   2        3032     Points to call to time contingency check
   3        3022     Points to call to R contingency check
   4        0000
   5        7777     State number - 2's complement
   6        4564     JMS I CHGER
   7        3032     Points to call to time contingency check
3010        3022     Points to call to R contingency check
  11        0000
  12        7777     Two's complement of state number
  13        7775     -3:  three event ctrs to be used
  14        6030     event counter 1: 10-worder
  15        6030     Time ctr for 1" contingency
  16        7777     Even counter 2:  high order time
  17        7777     Ctr for 1" contingency
3020        7775     Event ctr 3:  value = -3  event
  21        7775     Counter for "3R3" contingency
  22        4574     JMS I INCHKR
  23        0001     CHANNEL WRD = 1; this set of arguments  refers to
  24        0000     R1; no events to be counted.
  25        3036     Points to R2 processing arguments.
  26        --       These are transition operation calls - to be
  27        --       discussed later
3030        --
  31        --
  32        4575     JMS I KTGO
  33        3015     Points to low order ctr cell
  34        --
  35        --
3036        0002     Begin set 2 of R contingency arguments; no ctr
3037        0000     needed.
3040        3045     One more contingency
  41        --
  42        --
  43        --
  44        --
  45        0003     Arguments to process "3R3".
  47        3021     Points to counter cell.
3050        0000     No more R contingencies to check.
```

THE Z-PULSE CONTINGENCY

During the process of executing transitions due to elapsed time or sensed responses, Z pulses may be generated. (See C-12). After all time and response contingencies have been checked, the R.T.S. looks at each state set pointer group to determine whether any transitions depend on Z-pulse contingencies.

When a Z pulse can cause a transition in a particular state set, the third cell in the pointer group for that state set will contain the starting address of a chain of subroutine calls and groups of arguments that describe which Z pulses (e.g., Z1, Z2, Z6, etc.) will cause a transition and what functions will operate during the execution of the transition. The first subroutine call in the chain will have the following format:

```
JMS I INCHKR
CHANNEL WORD
EVENT CTR PTR
DEPT CALL PTR
```

Note that this is the same subroutine call that handles the checking of response contingencies. Indeed, the operation for Z pulses is very similar to the procedure described in the previous section. CHANNEL WORD is matched against a status word in the RTS that keeps track of what Z pulses have been generated. If the match shows that there might be a transition, EVENT CTR PTR is examined. If it is zero, a transition occurs. If not, the counting operation occurs. When no transition is allowed NEXT CALL PTR is checked. If it is zero, all Z contingencies for the current state have been checked. If not, there's more checking to be done, and NEXT CALL PTR indicates where the next set of arguments will be found.

CONTINGENCY GATING

When a time, response or Z contingency is gated the subroutine call and argument list for the transition is identical to that for a non-gated contingency. However, following the list is a call to GATE through which the gating contingency will be processed. The format is as follows:

```
JMS I GGO                   /Page Ø pointer to GATE.
Ptr. to Gating Tag          /Points to active state word.
                             in gated state set.
-N                          /Number of states which allow
                             transition.
```

```
State No. 1                     /List of states which allow
                                 a transition.
State No. N
GATE Open Funct.                /= Ø if no blank transit
GATE Closed Funct.
Ptr. to GATE Closed Funct.
```

OUTPUT EXPRESSIONS

Recording Counters

```
JMS I RECPTR                   /Single Precision
N
or
JMS I RECPTR+1                 /Double Precision
N
```

The call is to the single or double precision increment routine. N is the relative location of the counter, i.e. N = 12 increments C12. For a statement like CM, the F1 or F2 function puts the correct value in call +1 for N.

F1 Function

```
JMS I F1GO                     Variable type
VAR
ADDR                           Variable address
INC                            Increment
LIM                            Limit
```

The variable type VAR is defined as:

```
0 = Event counter (negated by routine)
1 = Time expression
2 = Single precision variable, pointer, status word or
    counter
```

Both INC and LIM will be two words long if VAR is 1, else they are single words only.

F2 Function

```
JMS I F2GO                     Variable type
VAR
ADDR                           Variable location
VAL                            Value
```

VAR and ADDR are the same as for the F1 function. VAL is the value to be assigned to the variable and if VAR = 2, VAL is two words long.

F3 Function

```
JMS I F3GO

STADR                         Starting address
ARG 1                         Arguments passed to user
ARG 2                         routine
```

STADR is the starting address of the user subroutine. ARG 1, etc. are the arguments to be passed to it. Note that when control is passed to the user routine, the AC = address of ARG 1 and when the user routine exits, the AC = address of ARGn+1.

Z-PULSE GENERATION

```
JMS I ZOUTGO
MSK
```

MSK is the mask of Z pulses to generate. Thus, for each bit set in MSK the corresponding Z pulse will be turned on. For MSK = 0044 = 000000100100, Z1 and Z6 will be generated.

STIMULUS OFF

```
JMS I STIMOFF
MSK
```

MSK is the same as the Z pulse mask, where each bit set turns off the corresponding stimulus.

STIMULUS ON

```
JMS I STIMON
MSK
```

MSK is the same as the Z pulse mask, where each bit set turns on the corresponding stimulus.

STOP

```
JMS I STOPP
```

This command has no arguments.

Table A - Subroutine Calls from Binary State Tables

Subroutine Name	Call Statement	Octal Value	Reference Page	Function
General				
STLOOP	JMP I LOOPER	5561	C-4	S.S. Scan
PPLUSA	JMS I PPLS1	4577	C-5	JMP I .+1 Page independent
Input				
INNCHK	JMS I INCHKR	4574	C-6	R, Z transition
KCTR	JMS I KTGO	4575	C-5	Time transition
GATE	JMS I GGO	457Ø	C-10	Gated transition
Output				
REC	JMS I RECPTR	4562	C-11	Counter
RECAST	JMS I RECPTR+1	4563	C-11	Counter, two word
FORT1	JMS I F1GO	4565	C-11	F1 function
FORT2	JMS I F2GO	4566	C-11	F2 function
FORT3	JMS I F3GO	4567	C-12	F3 function
ZOUT	JMS I ZOUTGO	4571	C-12	Turn Z on
STMF	JMS I STIMOFF	4572	C-12	Stimulus off
STMN	JMS I STIMON	4573	C-12	Stimulus on
Transition				
CHGSTS	JMS I CHGER	4564	C-4	State Change
STOPR	JMS I STOPP	4576	C-12	Stop

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

1. INTRODUCTION

1.1 Software Overview

SKED is a process control software system that has been developed for use in the behavioral research laboratory. The software system consists of:

A. The Two-Pass SKED Compiler.

This accepts as input "state programs" written in a simple format similar to state graph notation, which has been shown to facilitate concise and accurate descriptions of sequential processes. These programs are punched on paper tape in a format compatible with the PDP-8 SYMBOLIC EDITOR. Output is a binary tape, a "state table," that the Run Time System will use to direct its process controlling activities. The output tape can be loaded by the DEC PDP-8 Binary Loader.

Since the compiler source language is extremely general in scope, the system can be adapted to almost any process control application, though, to date SKED has only been applied to supervision of behavioral experiment systems.

B. The Run Time System (R.T.S.)

This software monitors responses from up to ten stations (each station may have as many as twelve response channels, and twelve stimulus channels) and directs the process of a reinforcement schedule according to state tables that have been loaded into the system. State tables can be loaded, started, aborted and interrogated through the Teletype keyboard. Up to ten state tables may be run simultaneously.

C. The DEBUG System

This is an adaptation of the SKED Run Time System that enables the programmer to find flaws in state tables before they are used with experimental subjects.

D. The System Builder

This is a dialog program designed to assist the user in setting up the Run Time System routines that handle data interchange in a system consisting of a UDC (Universal Digital Controller) and a PDP-8/e with DK8/e-P clock or a PDP-12 with KW12-A clock.

1.2 Hardware Requirements

A. Minimum Requirements

a. Any PDP-8 family computer with type 33 ASR Teletype and 4K of core memory.

b. A real-time 1ØØ cycle clock. If the PDP-8/S is used, this should be slower (program modification will be necessary to accommodate slower time base).

c. The hardware interface between the processor and the experimental stations. (See Section 1.3 d.)

B. Useful Hardware Options

a. High speed paper tape reader and punch (almost indispensible).

b. An extra 4K of memory.

1.3 Provisions for System Adaptability

In designing the SKED system the authors realized that:

a. Since it is impossible to anticipate and incorporate into a fixed system of software the ability to handle all of the peripheral devices that SKED can theoretically control, it will be desirable for users to modify the system to conform to varying I/O configurations.

b. The system should embody enough flexibility so that users can implement such modifications while maintaining a high degree of language compatibilty among installations.

c. In order to offer a viable alternative to laboratory control by relay or solid state circuitry, a computer based control system using SKED must be easy to set up and rearrange.

So that these needs might be fulfilled, the following features were included in the SKED System:

a. A Run-Time System binary tape is generated by the assembly of two symbolic source tapes: that of the R.T.S. proper, which is the main body of the R.T.S. and should rarely need modification; and the source of the device handler section, a set of subroutines and program sections that manipulate incoming and outgoing data, translating formats between the R.T.S. proper and the laboratory environment hardware. The device handler section must be programmed for each particular laboratory interface. Thus, the user modifications mentioned above may be implemented by writing a version of the device handler that fits the user's needs, then assembling the R.T.S. using this device handler as the second section.

b. Functional properties of the device handler routines and instructions for writing them are given in section 5.2 of this document. Using these descriptions as a guide, a programmer should be able to write the most efficient device handler for any given laboratory interface system.

c. A sample device handler has been included in the SKED software and is available as a source tape. It can control a lab interface with twelve response bits and twelve stimulus bit (distributed among four experiment stations) and can be expanded with very little modification. The clock can be a KW12-A or DK8/e or other type if the user wishes to modify the clock routines. Section 5.2.2 is an explanation of the handler in terms of the subroutines contained and the requisite interface structure and data structure.

d. The UDC System Builder has been included in the software system for the benefit of users of the UDC-8 (Universal Digital Controller) This program is a variation of the R.T.S. that requests data through the teletype describing the configuration of the UDC system to be used. The builder then punches a binary version of the R.T.S. with a handler for the configuration. Clocks required are the KW12-A (for use with a PDP-12) or the DK8/e-P (for use with the PDP8/e).

Since this program will automatically write a device handler, it will make easier the set-up of a computer controlled laboratory and facilitate fast rearranging of the relationships between the S/R channels and the experiment stations. The system builder is documented in section 5.2.3.

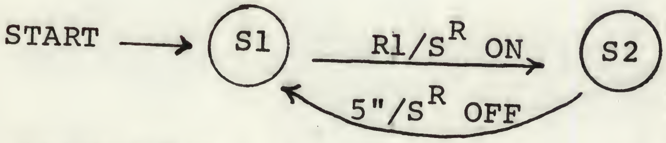
2. THE SKED LANGUAGE

2.1 Some Very General Theory[1]

A SKED program is made up of a sequence or many sequences of numbered logical components called states. Each state is directly associated with a static condition of the output equipment connected to the computer. Also associated with the state are transition data which describe what state in the sequence will be entered when the current state is exited, the new state's hardware condition, and what will cause the transition from one state to another. Transitions are initiated by any of three events: an input signal from the peripheral hardware (called a response), an internal "input" (called a Z input) and a clock counter input.

A very simple example of a sequence of states (or a state set) is seen in the simplest operant conditioning schedule of reinforcement in which the subject is mechanically rewarded for 5 seconds every time it presses a lever. In this situation let there be one system input, the pressing of the lever (abbreviated R1), and one system output, the reward (abbreviated S^R) mechanism.

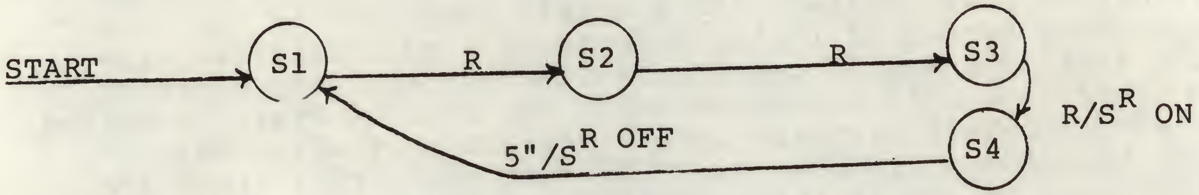
An Illustrative State Graph



The circles represent each of the states in the process and symbols within the circles are state identifiers. The arrows (or transition vectors) indicate the direction and order of state to state transition and the first symbol above each vector indicates the event which causes transition. The expression above the transition vector, following the division mark (/), indicates what output conditions will be active in the state being entered.

The diagram demonstrates that when the procedure starts, STATE 1 is entered with no specific hardware condition active. When the lever is pressed (symbolized by R1), a transition to STATE 2 is activated. The expression "S^R ON" indicates that stimulus hardware, the reward mechanism, is activated on entry into STATE 2. That the next transition, from STATE 2 to STATE 1, occurs after 5 seconds is represented by the lower vector and accompanying notation of '5"'. The additional notation, "S^R OFF", associated with the transition to STATE 1, indicates that the reward mechanism is turned off at the time STATE 1 is reactivated.

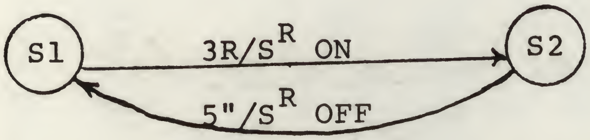
Often processes like the above reinforcement schedule require that more than one response input be recorded before an output is activated. For example, instead of rewarding a subject for every correct response, a schedule of 3 responses to one reward may be desired. One way of expressing this in state graphs is as follows:



Each response after the start of the process causes a transition to the next state until in STATE 4 a reward is administrated for 5 seconds after which STATE 1 is reentered.

A disadvantage of this expression is that the number of states in the expression increases for each response which must be counted before the reinformcement is administered. An easier method of expressing the same process using only two states is to accept multiple responses as the criterion for a state transition.

Example:



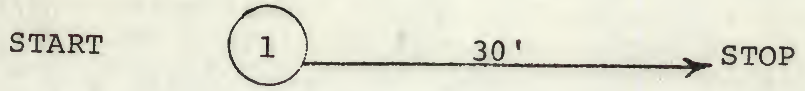
In this state graph, the transition to STATE 2 is triggered not by one response; rather by 3.

In the previous example reinforcement was presented for a duration controlled by a timer associated with STATE 2. Each state may possess a timer that is reset on state entry and will cause a transition when the specific duration elapses. When there is a need for timers that are free running, it is convenient to represent some processes in terms of two or more separate sequences of states or STATE SETS (representing sub-processes), which may communicate with each other through the internal logical input/output scheme called "Z pulses".

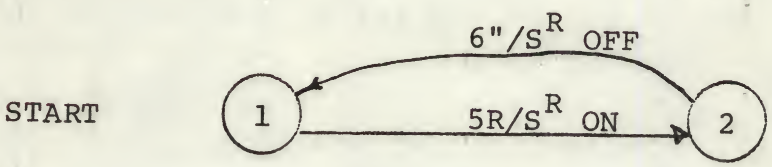
An example of two co-existent but non-interacting sub-processes is the reinforcement schedule which must stop automatically at the end of a certain time interval. One sub-process is the actual schedule of rewards and the second sub-process is the session timer. The following state graph demonstrates a schedule of one reward for every five responses, with a session duration of 30 minutes.

Application Example 1

STATE SET 1



STATE SET 2



Notice that STATE SET 1 represents the session timer while STATE SET 2 represents the same schedule of reinforcement described in the previous example. The sub-process in both state sets will start simultaneously at STATE 1 and will run their courses independently of each other.

While the schedule in STATE SET 2 is shifting back and forth between its two states, STATE SET 1 will have its activity in STATE 1 for 30 minutes. At the end of 30 minutes the transition in STATE SET 1 to the "STOP" state will cause the entire process, in both state sets, to halt completely.

The task of describing a computer controlled process, then, may be accomplished by using the SKFD language to describe the states of the process in terms of the transitions from one state to another. The following general rules are essential to the understanding and usage of transition descriptions. Note that these rules, being theory, will have minor modifications in practice. These problems will be discussed as they arise.

1. In any given state set, only one state describes the condition of status of a process at any given time.

2. A status transition is to be considered instantaneous.

3. Inputs are instantaneous events which trigger state to state transitions. Where multiple transition criteria exist, that input which occurs first will trigger its associated transition.

2.2 The Basic Structure

It is important that the reader of this document understand that there is a distinction in the SKED language format between a zero and the letter "O". In the reproduction of SKED state programs these characters will appear as they are printed on a Teletype: a zero appears as "Ø" and the letter is "O". Also, note that octal integers in the SKED expressions are always led by the letter O which indicates that an octal number follows.

A SKED language program or process description always consists of one or more STATE SETS, the terminating character of the program being a dollar sign ($).

A STATE SET consists of a STATE SET LABEL followed by a list of STATE EXPRESSIONS. Each STATE EXPRESSION, in turn, contains a STATE LABEL followed by a list of TRANSITION EXPRESSIONS. The most common SKED program format then, is as follows:

```
STATE SET LABEL

STATE LABEL
    Transition expression }                                  }
    Transition expression }       A State Expression         }
    Transition expression }                                  }
                                                             } A STATE SET
STATE LABEL             }                                    }
    Transition expression }       A State Expression         }
    Transition expression }                                  }

STATE SET LABEL                   The remaining program:
STATE LABEL                       STATES within STATE SETS.
      :
      :  $ <----------------------- The terminating character
```

2.2.1 STATE SET LABELS

A STATE SET LABEL must always begin with a CARRIAGE RETURN/LINE FEED combination. The first characters in the new line are "S.S." These are followed by a positive decimal integer with up to four digits whose value does not exceed 4095 and that serves as a STATE SET IDENTIFIER. The number is usually followed by a comma, which is often the last character in the line. An example follows:

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

2.2.2 STATE LABELS

A STATE LABEL must always begin with a CARRIAGE RETURN/LINE FEED combination. The first character in the next line is "S". This is followed by a four digit, positive decimal integer, whose value does not exceed 4Ø95 and together they serve as a STATE IDENTIFIER. The number must always be followed by a comma, which is often the last character in the line.

2.2.3 TRANSITION EXPRESSIONS

A TRANSITION EXPRESSION must always begin with a CARRIAGE RETURN/LINE FEED and usually occupies only one printed line but may sometimes be continued over several lines. The TRANSITION EXPRESSION, being the dynamic part of the SKED program, is composed of an INPUT SECTION, an OUTPUT LIST SECTION and a TRANSFER SECTION. The INPUT SECTION consists of an expression designating one of four types of input events that can trigger a transition. The OUTPUT LIST SECTION begins with a colon or semicolon and lists one or several output functions that will be activated if their TRANSITION EXPRESSION is executed. The TRANSFER SECTION consists of one or more hyphens (-) followed by a right angle bracket (>), followed by the STATE IDENTIFIER of the state which will be activated if the transition is executed.

A single STATE SET then would appear schematically as follows:

```
S.S.1,

    S1,
        INPUT SECTION:  OUTPUT LIST SECTION -------------------> S1
        INPUT SECTION:  OUTPUT LIST SECTION -------------------> S2
        INPUT SECTION:  OUTPUT LIST SECTION -------------------> S3
    S2,
        INPUT SECTION:  OUTPUT LIST SECTION -------------------> S10
        INPUT SECTION:  OUTPUT LIST SECTION -------------------> S11
        INPUT SECTION:  OUTPUT LIST SECTION -------------------> S20
```

If this state set were to stand alone as a complete program, the last state, S2, would be terminated by a dollar sign.

2.2.4 COMMENTS

Comments, explanatory aids to program documentation, may be used liberally throughout a SKED program. The COMMENT consists of a division mark (/) followed by a character string, which is ended by a CARRIAGE RETURN/LINE FEED combination. The character string may contain any ANSCII character except the dollar sign, which always signifies "end of program". A COMMENT may appear on a line by itself or on the same line as a STATE SET LABEL, STATE LABEL or TRANSITION EXPRESSION. In each case the comment must follow the expression.

Example:

```
/THIS COMMENT IS ON A LINE BY ITSELF.
S.S.1,          /THIS COMMENT FOLLOWS A STATE SET LABEL
S1,
        INPUT SECTION:  OUTPUT SECTION ---> S1      /YOU CAN'T FIT
                /A VERY LONG COMMENT ON THE SAME LINE WITH A
                /TRANSITION EXPRESSION.
```

2.2.5 Starting and Stopping Programs

The process described by a SKED program will always start at the first state. When there are multiple state sets then multiple sub-processes are started simultaneously at the first state in each state set. Processes are stopped by a "STOP" command in the TRANSFER SECTION of a TRANSITION EXPRESSION.

Example:

```
    S4,
         Rl ----------------> STOP
```

Instead of transferring to another state when Rl is sensed, the process in question will come to a complete stop. All sub-processes will halt -- there will be no more action -- the session has ended.

2.3 THE INPUT SECTION

The INPUT SECTION of a TRANSITION EXPRESSION consists of the expression of one of four types of contingencies that can trigger a transition. These are:

a. The passage of a specified interval of time after state entry.

b. The sensing of one or more response inputs from some peripheral hardware (subject station, for example).

c. The sensing of one or more Z pulses.

d. One of the above conditions gated with the activity of a certain state or states in a certain state set. or with the content of a certain cell in memory.

2.3.1 The Time Contingency

The time of activity of a state must be expressed in terms of minutes and seconds, in that order (unless it is represented by a variable. See section 2.5.4). The number of minutes in the time interval is followed by a single quote mark (') and the number of seconds is followed by a double quote mark ("). Note that the double quote is a single character - not a repetition of the single quote. A TIME EXPRESSION in a transition expression, then, may look like the following:

```
S1,
   1'2Ø" ------------> S2
```

This would indicate "a transition to STATE 2 after one minute and twenty seconds following entry of STATE 1". In a time contingency expression, either the minutes or seconds expressions may be omitted if not needed. It would not be necessary to write

```
Ø' 2Ø" -------------> S7
```

to indicate "transition to STATE 7 after 20 seconds". This could be expressed

```
2Ø" ----------------> S7.
```

Within certain limits, time can be expressed in terms of non-integer decimal numbers. Example:

```
.1Ø" ---------------> S6
```

This is an expression of "transition to STATE 6 after one-tenth of a second".

The limits are:

a. The compiler can accept no decimal value smaller or more precise than .Ø1.

b. No integer value greater than 4Ø96 is acceptable.

c. The currently implemented RUN TIME SYSTEM counts time in intervals of 10 milliseconds. Since it cannot count higher than 2^{24}, only temporal values in the range .01 to 2^{24} x .Ø1 seconds can be counted by the R.T.S.

d. In the decimal expression of a value less than one, two digits must always follow the decimal point. For example, the value "one tenth" must be expressed: ".1Ø" rather than ".1 ". More than two digits following the decimal point is an error condition.

2.3.2 The Response Contingency

When it is desired that one or more occurrences of a given response should be the triggering criterion for a state change, the expression in SKED should take the following form:

```
MRN ----------> S5
```

where M may be a decimal integer or a variable (see section 2.5.4) with value between 1 and 4Ø96, inclusive;

RN is a response channel identifier with N representing a decimal integer between 1 and 12, inclusive.

For a concrete example:

```
5R1 ------------> S6
```

means, "after five responses on response channel 1 change to STATE 6". When only one response is required to trigger a transition the number represented by M may be omitted.

```
R3 -----------> S2
```

signifies: "transition to STATE 2 after the first response on channel 3".

Note that in some laboratory hardware configurations there will be less than 12 response input channels and some response channel identifiers will have no functional meaning. For example, the original lab situation set up by Dr. A. G. Snapper allows for three input channels per box. Only R1, R2, and R3 can cause transitions and R4-R12 have no meanings in a reinforcement schedule.

2.3.3 The Z Contingency

A Z pulse is treated as an instantaneous logical pulse which is generated as an output function at the time of transition from one state to another (see section 2.5.2) and is completely an internal, logical phenomenon. There is no physical condition which it represents and no condition of peripheral equipment is directly affected by it.

One or more repetitions of a certain Z signal can trigger a transition. The expression which allows this is:

```
MZN ----------> S7
```

M is a decimal integer or an alphabetic variable (see section 2.5.4) with value between 1 and 4ø96, inclusive.

ZN is a Z signal identifier with N representing an integer between 1 and 12, inclusive.

As in the response contingency expression the value M may be omitted when only one occurrence of the Z signal must trigger the transition.

Examples:

```
    Z5-------->   S6
```

signifies, "transition to STATE 6 after the first pulse of Z5".

```
    7Z3------->   S1
```

means, "after seven pulses of Z3, change to STATE 1".

Note that there are always 12 different Z pulses available to the SKED user. This is independent of the hardware configuration of the SKED system.

2.3.4 The Gating Expression

The GATING EXPRESSION allows programmed control of the conditions of state transition by causing the status of agiven state set or the content of a given core memory cell to be taken as a transition contingency in connunction with one of the three types of input events (explained above).

THE GATING EXPRESSION format:

```
    E.T(N1,N2,N3...)
```

E is a time, response or Z contingency expression.

T is a GATING TAG or an octal integer expressing which core memory location will be referred to in the contingency checking operation.

N1,N2,N3...is a list of decimal or octal integers (up to ten arguments may be used).

The GATING EXPRESSION must contain a GATING TAG (rather than an octal or decimal core address) when the transition is to be contingent upon the status of a state set (i.e., the state that is active in the state set). The GATING TAG must also appear in the STATE SET LABEL of the state set involved. The tag will precede the the comma in the STATE SET LABEL and will be preceded by a "=". This type of gated transition will be executed when the input event designated in the expression coincides with the activity of one of the states whose numbers are listed within the parentheses in the GATING EXPRESSION. An example of this usage:

```
R1.A(1,3)---> S4
```

According to this expression, a transition to STATE 4 will be executed when there is a coincidence of an R1 and the activity of STATE 1 or STATE 3 in the state set whose label includes the tag "A". An example of the inclusion of a GATING TAG in a STATE SET LABEL follows.

```
      S.S. 3=A
```

Thus, if this expression were in the same program with the expression in the immediately preceding example, the transition would occur only in the event of the concurrance of an R1 with the activity of STATE 1 or STATE 3 of STATE SET 3.

A schematic example of an entire program with this type of gating:

```
S.S.1,

S1,              /A GATING EXPRESSION FOLLOWS:
                  R1.A(2,3)- - -> S2

S2,              TRANSITION EXPRESSION
                 TRANSITION EXPRESSION
                 TRANSITION EXPRESSION

S3,              TRANSITION EXPRESSION

S.S.2 = A,       /NOTE THE GATING TAG HERE

S1,              TRANSITION EXPRESSION
                 TRANSITION EXPRESSION
                 TRANSITION EXPRESSION

S2,              TRANSITION EXPRESSION

S3,              TRANSITION EXPRESSION
                 TRANSITION EXPRESSION
                 TRANSITION EXPRESSION
```

The GATING EXPRESSION in this example will allow a transition only if a response is sensed on response channel 1 while STATE 2 or 3 of STATE SET 2 is active. Sensing of R1 while STATE 1 of STATE SET 2 is active will result in no transition.

When the contingency of a transition is to be defined in terms of an input event and the content of a core memory cell an octal or decimalinteger will take the place of the GATING TAG in the GATING EXPRESSION. When the input event designated in the expression occurs the core location indicated by the address must contain one of the values listed within the parentheses in order for the transition to be executed. The values within the parentheses may be expressed in terms of octal or decimal integrers.

Example:

R1.01111(1,3)--S2

If, when an R1 is sensed and the content of location 1111_8 is 1 or 3 there will be a transition to STATE 2.

Example:

R1.585(1,3)--2

Since 585_{10} is equivalent to 1111_8, this example expression will work the same as the preceding example. If R1 is sensed while the indicated location contains a 1 or 3 the transition will occur.

Rules for GATING EXPRESSIONS:

1. A GATING TAG consists of only one character.

2. Only the letters A, B, C and D may be used as GATING TAGS.

3. A GATING TAG may be used any number of times in GATING EXPRESSIONS.

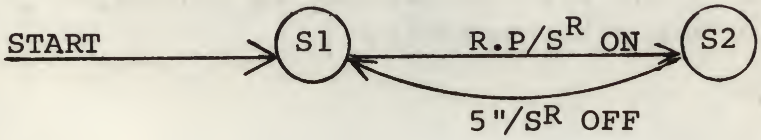
4. A GATING EXPRESSION may execute a comparison with any cell in core memory if the GATING TAG is replaced by the core address of the cell, expressed in terms of a four digit octal or decimal integer. The octal integer must appear in the format "ONNNN" where NNNN is an octal value in the range Ø-777. The decimal integer must, of course, be limited to the range Ø-4Ø95.

Application Example 2: RANDOM RATION REINFORCEMENT

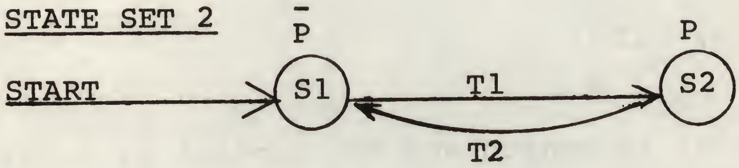
To program a random ration reinforcement schedule a free running probability generator may be used, which can be examined each time a response occurs, to determine whether or not the response should be reinforced. The probability generator takes the form of a separate STATE SET in which activity changes between two states according to elapsed intervals.

Example:

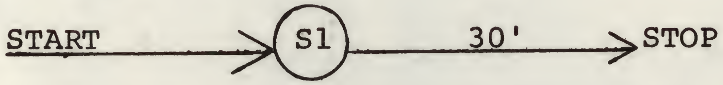
STATE SET 1



STATE SET 2



STATE SET 3



Note that the transition condition for STATE 1 in STATE SET 1 is symbolized:

R.P.

This signifies that only P percent of Responses, R, should trigger a transition. If R can be reinforced only when it occurs while STATE 2 of STATE SET 2 is active, then the probability of its reinforcement is T2/(T1+T2) since this is the percentage of time during which STATE 2 is active. Thus, P=T2/(T1+T2).

The SKED expression of this schedule might look like this:

```
S.S.1
S1,
              R1.A(2):ON 1 --> S2
S2,           5":OFF 1  --> S1


S.S.2 = A,
S1,
              .1Ø"  --> S2

S2,           .Ø5"  --> S1


S.S.3,        /THIS IS THE SESSION TIMER.
S1
              3Ø'  --> STOP

              $
```

In this program the onset of reinforcement stimulus (signified by the expression "ON 1". "OFF 1" signifies offset) is possible only when an R1 pulse (a response) coincides with the activity of STATE 2 in S.S.2. Since, in S.S.2. Since, in S.S.2, state activity is always shifting between STATE 1 and STATE 2 at the constant intervals .1Ø sec. and .Ø5 sec., the probability of this coincidence is .Ø5 sec./.15 sec., or 33%. Thus, there is a 33% probability that any R1 will be reinforced (the empirical outçome is approximately equal to the computed probability).

Caution: This algorithm works only when the average inter-response time is large and variable compared with T1+T2.

2.3.5 The Blank Transition Expression

The above example of a GATING EXPRESSION application demonstrated a situation in which the transition was triggered only when the implicit logical statement "Input X sensed AND STATE Y active" is true. It is sometimes desirable, however, not only to trigger the above type of transition, but also to trigger a transition when STATE Y is not active and the logical statement implicit in the GATING EXPRESSION is false.

In SKED this seocnd possibility may be expressed by following the GATED TRANSITION EXPRESSION with a BLANK TRANSITION EXPRESSION, i.e., a TRANSITION EXPRESSION which has no explicit INPUT SECTION.

Example:

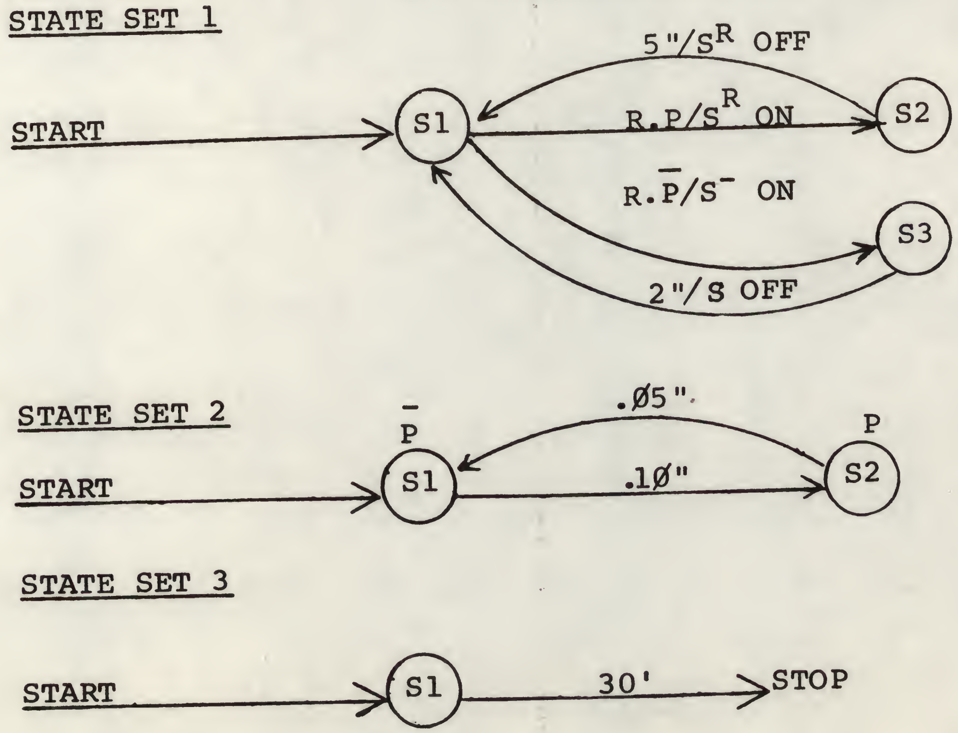
```
S3,
             R1.A(2)      --> S1
                          --> S2

S4,
```

In this example R1 will always trigger a transition: if STATE 2 is the STATE SET tagged by A is active when R1 is sensed, the transition will be to STATE 1; else the transition to STATE 2 will be triggered by R1.

Note that the BLANK TRANSITION expression can only be used directly after a GATING EXPRESSION.

Another example will demonstrate an application of the BLANK transition.

Application Example 3: RANDOM RATION REWARDS AND PUNISHMENTS



R = response at lever
S^R = stimulus: reinforcement (food hopper open)
S- = stimulus: punishment (electric shock)
The session will be thirty minutes long.

Conceptually, the schedule expressed above is similar to the schedule in Application Example 2 with the difference that in STATE SFT1 there are two transitions coming out of STATE 1 contingent upon the coincidence of a response and another STATE's activity.

In STATE SET 1, STATE 2 will be entered from STATE 1 only when the response is sensed while STATE 2 of STATE SET 2 is active. STATE 3 of STATE SET 1 will be entered from STATF 1 when the response is sensed and STATF 2 of STATE SFT is not active. In other words, a rat pressing the lever (R) will have a 66% probability of being rewarded (S^R) and a 33% probability of being punished (S^-).

The SKED expression of this schedule would look like this:

2.4 THE OUTPUT LIST SECTION

The OUTPUT LIST SECTION of a TRANSITION EXPRESSION states what stimuli or conditions will be activated or deactivated on entry of the state indicated in the TRANSFER SECTION of the TRANSITION EXPRESSION. The five types of output functions, which can be used by including their identifying expressions in the OUTPUT LIST SECTION, are:

1. external or stimulus outputs
2. internal Z-signal generation
3. recording counter incrementation
4. variable value assignment
5. assembly language sub-routines

The format of the OUTPUT LIST SECTION consists of a colon followed by a list of output function expressions divided by colons or semicolons.

Schematically, the output list appears like this:

```
      : identifier;  identifier; identifier
```

The end of the list is at first hypen of the TRANSFER SECTION.

The output list can be extended over several lines by typing CARRIAGE RETURN/LINE FEED combinations directly before a colon or semicolon that comes before an identifier.

Example:

```
R1  :identifier; identifier; identifier; identifier
    ;identifier; identifier
    ;identifier; identifier  ----> S2
```

Here we see a schematic TRANSITION EXPRESSION with a very long output list extended over several printed lines.

Caution: When the output section is extended in this manner, no COMMENT should be placed on a line between parts of the output list.

```
Example:   R1   :identifier; identifier
                ;identifier; identifier
                /THIS IS A BAD PLACE TO PUT A COMMENT
                ; identifier  ---> S2
```

Such placement of the COMMENT is illegal because it comes in the middle of the transition expression.

Although, in theory, a transition is an instantaneous occurence and the output functions, being a part of the transition, should also be instantaneous, and therefore, their executions should be simultaneous, there is an order of execution. Output functions are executed sequentially in left to right order of mention in the output list.

2.5 THE OUTPUT FUNCTIONS AND THEIR EXPRESSIONS

2.5.1 The External Stimulus Output

A STIMULUS OUTPUT (e.g., light, shocker, food hopper mechanism, etc.) may be activated or deactivated by the words ON or OFF, respectively, followed by a list of decimal numbers ranging in value from 1 to 12 which serve as channel identifiers, in this format:

```
or      ON 1,2,3;
       OFF 1,2,3;
```

Note that in some cases the list of channel numbers can be replaced by an alphabetic variable or an acted integer. (see sections 2.5.4 and 2.5.5).

Example:

```
          R1: ON 1Ø -----> S3
```

This will turn on the output equipment on Channel 1Ø before transferring to STATE 3 <u>if</u> a response is detected on response Channel 1.

```
     R2:  OFF 11;  ON 1   --->  S3
```

This expression would cause deactivation of equipment on Channel 11, activation of equipment on Channel 1, and a transfer to STATE 3 if response on input Channel 2 is sensed.

```
     R3:  ON 1, 4, 5   --->  S5
```

This will activate the equipment on Channels 1, 4, and 5 before transferring to STATE 5, if a response is sensed on response Channel 3.

(Note: no inherent significance can be attached to any output channel identifier number. The meaning a given channel identifier will have will be a function of the relationship between the R.T.S. proper and the laboratory control interface, as defined by the device handling subroutines. See section 5.2)

2.5.2 Z Pulse Generation

One or more Z pulses will be generated as an output function by the expression

```
    Z  1,  2,  3,  4;
```

1,2,3...is an example of a list of decimal integers with value between 1 and 12, inclusive, that indicate which of 12 pulses will be generated. Note that in some cases this list may be replaced by an alphabetic variable (see sections 2.5.4 and 2.5.5).

Example:

```
        S1,
             10"   :ON 1; Z 1, 3, 4 --->  S2
             R1    :Z2 ----> S1
```

Ten seconds after entry of STATE 1 there will be a transition to STATE 2 during which output channel 1 will be activated and pulses Z1, Z3, and Z4 will be generated. If R1 is sensed before the ten seconds elapse, there will be a generation of the Z2 pulse and a return to STATE 1.

2.5.3 Recording Counter Incrementation

Recording of accurances of parts of a process, or a schedule of reinforcement, is considered by SKED to be an output operation in which a recording counter, a cell in core memory located at the end of the SKED STATE TABLE is incremented during a transition from one state to another.

The accumulated values can be printed on the systems teletype at any time. (see section 4.3)

Recording Counter incrementation occurs when a RECORDING COUNTER EXPRESSION is included in the OUTPUT LIST SECTION of a TRANSITION EXPRESSION that is executed.

The RECORDING COUNTER EXPRESSION is as follows:

```
     CN
or
     CN*
```

where N indicates the position of the Recording Counter to be incremented (relative to the beginning of the reserved Recording Counter area) and may be either a four digit decimal constant ranging in value from 1 to 4Ø95 or a variable (see next section) with the same range of values.

The presence or absence of the asterisk (*) in the RECORDING COUNTER EXPRESSION denotes how many cells will be involved in the recording operation. If no asterisk is present, the Recording Counter consists of one 12 bit cell, the maximum recordable value being 4Ø95 (when more than 4Ø95 events have been recorded the counter will start recording at zero again); the presence of the asterisk causes two cells to be used in event recording giving a 24 bit value (maximum value of 16,777,215).

Note that it is significant that "N" in the RECORDING COUNTER (R.C.) EXPRESSION is a position indicator (or relative address), not an identifier. "C12" designates that the twelfth call in the Recording Counter area will be incremented. It does not indicate that some cell identified as C12 will be incremented. This has several less than obvious implications:

A. The numbers used in R.C. EXPRESSION should be an accurate reflection of the number of reserved cells desired. For example, if there are to be three cells used in a given program, their incrementations should be expressed as C1, C2, C3 in the program.

B. Asterisked R.C.'s use two cells so that misuse of an R.C. EXPRESSION could result in one expression causing counting in the overflow cell of another.

Example:

```
        S1,
             R1:C1* -----> S2
             10":C2 -----> S1
```

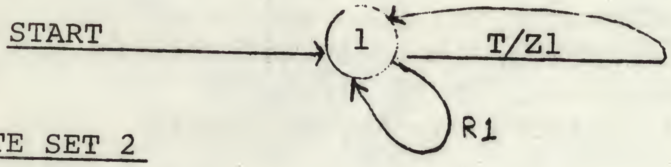
In this STATE EXPRESSION, every transition due to a sensing of R1 will cause incrementation of a double cell R.C. whose low order cell is the first cell in the reserved R.C. area and whose overflow cell is the second cell in the reserved area. But every time the second transfer occurs, a "C2" (or incrementation of the second cell in reserved R.C. area) is executed; i.e., the overflow cell for C1 is modified. This is not good if the user actually wished to count in two separate Recording Counters.

At run time all Recording Counter cells are set to zero when the state program is loaded into the R.T.S. and when a DUMP operation is executed (see section 4.3). Retrieval of the accumulated values in the counters is an operation of the Run-Time System.

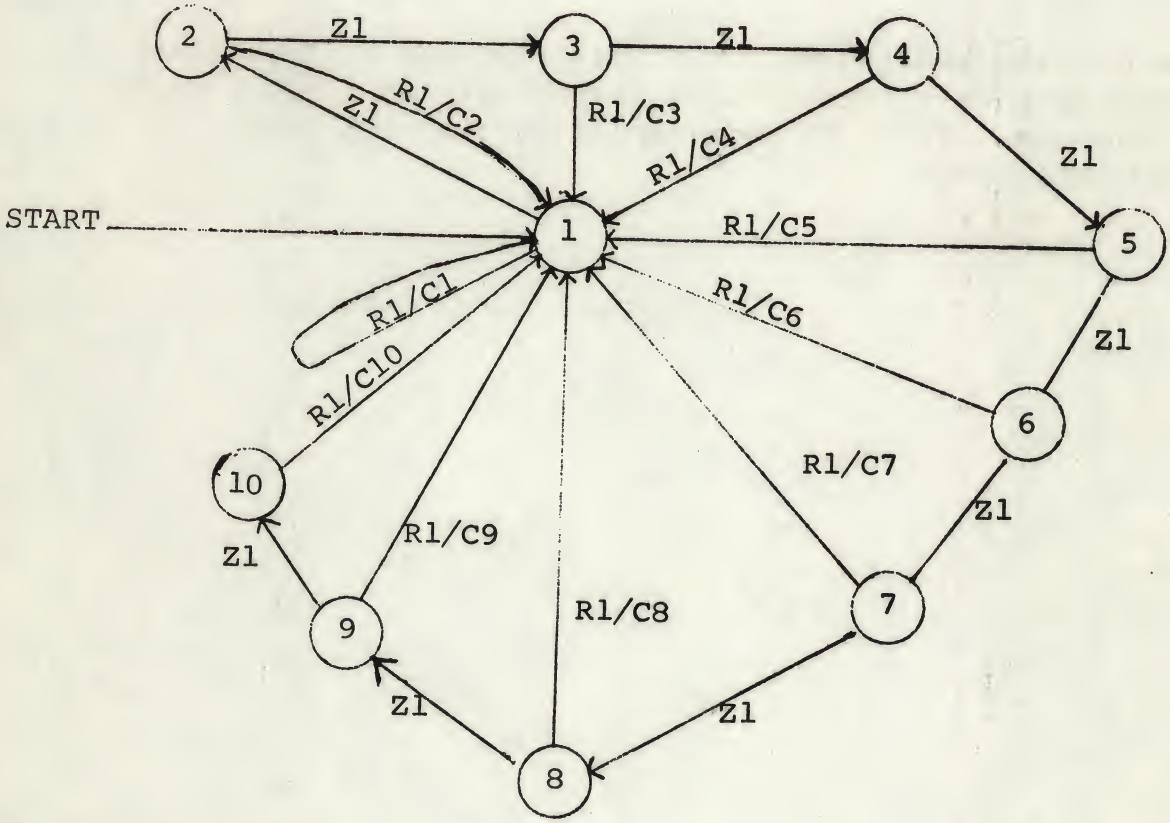
Application Example 4: AN I.R.T. DISTRIBUTOR

The design of an inter-response-time (I.R.T.) distributor is a very good example of the use of both Recording Counters and Z pulses for a much better method (see application example 5). A state graph of a process for accumulating I.R.T. data during any reinforcement schedule follows:

STATE SET 1



STATE SET 2



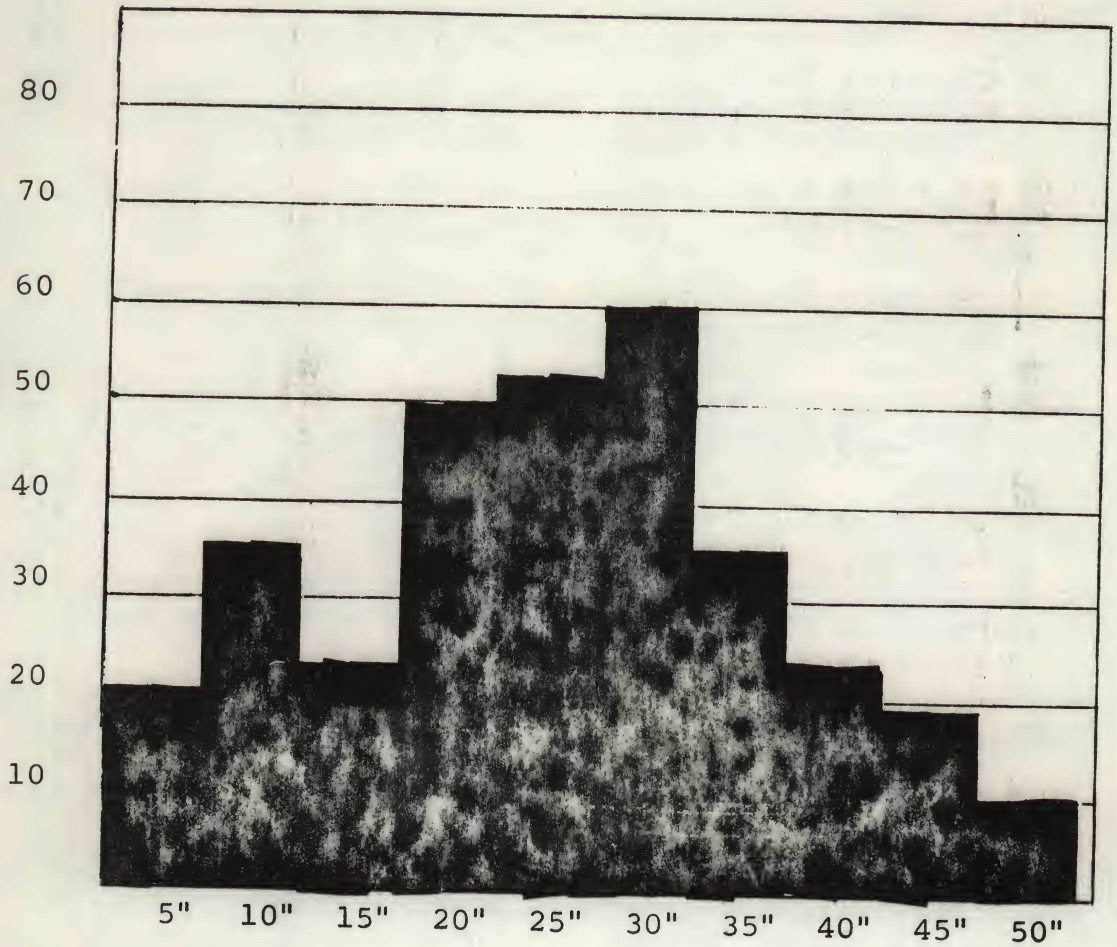
The algorithm of I.R.T. data collection used here is as follows:

There are ten Recording Counters each one of which will contain the number of responses that have been sensed a certain amount of time after the previous response. Counter 1 will contain the number of responses which came within T seconds after the directly previous response; counter 2 will record the number of responses which occurred between T and 2T seconds after the previous response; counter 3, the number of responses between 2T and 3T seconds after, etc.

In the state graph, STATE SET 1 serves as one state clock which generates a Z1 pulse every T seconds with the time count starting after each response because of the response initiated recycling to a position in S.S.1). STATE SET 2 records the number of responses falling into each inter-response time-elapse category. It can be seen that each state is active for a maximum of T seconds after which the Z1 pulse causes transition to the next state. If a response is sensed while STATE N is active, then the response will be recorded in counter N and the process will return to STATE 1. When STATE N is active, at least NxT seconds must have elapsed since the last response. Thus, (N-1) XT is the inter-response time-elapsed category for any Recording Counter.

At the end of the session an I.R.T. distribution histogram may be produced by merely representing graphically what value each counter contains. It might look something like the graph on the following page.

Illustration: An I.R.T. distribution histogram



Vertical axis: Number of responses

Horizontal axis: Time since previous response

The SKED program to collect I.R.T. data using this algorithm is as follows (T will be 5"):

```
S.S.1
S1,
                  5":Z1  ---> S1
                  R1 ---> S1

S.S.2
S1,
                  R1:C1  ---> S1
                     Z1  ---> S2
S2,
                  R1:C2  ---> S1
                     Z1  ---> S3
S3,
                  R1:C3  ---> S1
                     Z1  ---> S4
S4,
                  R1:C4  ---> S1
                     Z1  ---> S5
S5,
                  R1:C5  ---> S1
                     Z1  ---> S6
S6,
                  R1:C6  ---> S1
                     Z1  ---> S7
S7,
                  R1:C7  ---> S1
                     Z1  ---> S8
S8,
                  R1:C8  ---> S1
                     Z1  ---> S9
S9,
                  R1:C9  ---> S1
                     Z1  ---> S10
S10,
                  R1:C10 ---> S1
                  $
```

2.5.4 Variables and the Assignment Functions

There are several types of values in a SKED program that can be either constants (set at compile time and never changing) or variables, which are identified by an alphabetic character in the range E-Z, inclusive.

The allowable uses of variables are:

1. as event counters in R and Z input expressions. For example:

```
NR1:C1; ON 11 ---> S3
MZ2:C13  ---> S5
```

2. as Recording Counter position indicators. For example:

```
R1:CJ; OFF 12 ---> S6
```

3. as identifiers in External output and Z-output expressions. For example:

```
10":ON K; OFF L; Z N ---> S7
```

   This application has special problems which will be covered in section 2.5.5.

4. as a time expression. For example:

```
F:ON 1 ---> S6
```

   F stands for a time value

In the first three of these contexts, the variable can have from 1 to 4 digit positive decimal value not exceeding 4096. When representing time, the variable will have a maximum value of 167,772.15 seconds.

Values are assigned to the variables through the use of two EXTENDED FUNCTIONS, called ASSIGNMENT FUNCTIONS, that operate when included in the output list of a TRANSITION EXPRESSION.

The general format of the EXTENDED FUNCTION is:

```
FN (ARG1, ARG2, ARG3 ...)
```

N is the function identifier. ARG1, ARG2, ARG3 ... is a list of arguments describing the parameters of the function to be executed. Any of these arguments may be:

1. an octal integer in the range of Ø-7777$_8$ expressed in the form "ONNNN" where NNNN is the octal value.
2. a decimal integer in the range of Ø-4Ø95 expressed in terms of 1 to 4 decimal digits.
3. a variable's symbol.
4. a time value expressed in the format illustrated in section 2.3.1.

5. an expression of the form "FA + K" or "FA-K" where K represents a decimal or octal integer as described above; "FA" represents the final address of the compiled state program. The expression as a whole represents an address in core memory referenced relative to the end of the state program and is similar to a variable symbol.

Note that any of these formats (except number three) may be preceded by a "+" or "-" to indicate arithmetic signs.

A value may be assigned to a variable by the F2 function:

```
F2 (T, VALUE)
```

T is the variable's core location or its alphabetic symbol; VALUE is a value expressed in any of the above formats (not including the variable) which will be given to the variable.

Example:

```
S1,
    R1:F2(N,5) ---> S2
S2,
    NZ1:ON 1 --->   S1
```

If STATE 2 is entered from STATE 1, five Z pulses will be required before a transition back to STATE 1 will occur.

The F1 function allows incrementation or decrementation of a variable value. Example:

```
F1 (T, INC, LIM)
```

T is the variable's core location or its alphabetic symbol; INC is the value by which the variable will be incremented or decremented; LIM is the value beyond which the variable may not be incremented or decremented.

Example:

```
2R2:  F1(N,  1,5) ---> S7
```

When two responses on input channel 2 have been sensed, the variable symbolized by "N" will be incremented <u>unless</u> the incrementation will cause the value of N to exceed 5, in which case N will remain unchanged.

Example:

```
Z6:  F1(M  -1,Ø) ---> S4
```

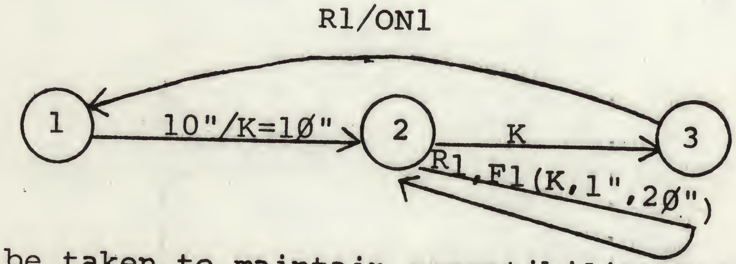
When the next Z6 pulse occurs, M will be decremented unless such decrementation would cause the value of M to fall below zero.

Example:

```
S.S.1,
S1,
      10":  F2(I, 1Ø")   ---> S2
S2,
      R1:  F1(I, 1", 2Ø")   ---> S2
      I  --->S3
S3,
      R1:  ON  1  ---> S1
      $
```

This demonstrates the use of a variable to represent time (second transition in STATE 2) and the manner in which the value of the variable can be set and modified by the Assignment Functions.

The corresponding state graph follows:



Care must be taken to maintain compatibility between the incrementation value and the limiting value in an F1 function. For example, INC = 1, LIM = -1Ø and a starting value of Ø would be a meaningless situation and would be impossible to execute (the Run-Time System would allow no incrementation) yet, the compiler would not catch such an error. If the user of SKED ASSIGNMENT FUNCTIONS observes the following simple rule, operational compatibility between these two arguments will be assured.

Given: VAL = The value of a variable before incrementation
INC = the incrementation value
LIM = the incrementation limit

THEN; if the sign of
LIM - VAL - INC
is the same as the sign of
INC
incrementation will occur; otherwise the value of the variable will remain unchanged.

NOTE: This is a safe general rule for use with time expression incrementation. When variables that represent four digit decimal or octal values are operated on, two additional considerations must be realized:

a. VAL and LIM must both be greater than or equal to zero.

b. INC must be a signed integer with absolute value in the range cf Ø-2047.

Nonconforming values can be compiled but the results thereof at run time are unpredictable.

Other rules of operation with variables:

1. Time Variables must be the characters in the range E-I.

2. Variables used in the first three applications mentioned at the beginning of this function must be characters in the range J-Z. Remember that A-D are GATING TAGS.

3. If, in a SKED program, a variable must be referenced before it has been assigned a value by the F2 function, it will have an automatic starting value assigned according to its use:

   a. when the variable acts as an R or Z event counter, the starting value will be 1.

   b. when the variable represents time, the starting value will be .ø1".

   c. when the variable acts as a Recording Counter identifier, the starting value is ø.

   d. when the variable acts as a Z or External Output Identifier, the value will be assumed to be zero, i.e. no output or Z pulse will be operated.

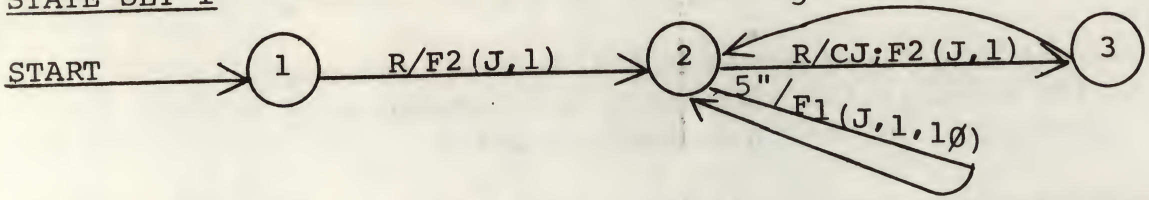
   e. The SKED compiler will not allow compilation of an ASSIGNMENT FUNCTION in which the variable is receiving a value which is incompatible with its size. Specifically, a function in which a time value is to be assigned to an R or Z event counter is illegal and will generate an error diagnostic at compile time.

Note that since any argument of the ASSIGNMENT FUNCTIONS may be a decimal or octal integer, the first argument, usually a variable's symbol, may express the absolute address of a cell to be modified by the function. This enables the user to store a value in some location in a 4K memory field and later test the content of the cell with a GATING EXPRESSION and modify the course of a program according to the result of that test (see section 2.3.4). Such a procedure might be especially useful in coordinating the sessions of several different experimental stations.

Application Example 5: Compact I.R.T. Distributor

The use of variables and ASSIGNMENT FUNCTIONS makes possible marked reduction of the state graph describing the process of I.R.T. data collection described in Application Example 4. A new state graph describing the same process but using the new techniques follows:

STATE SET 1



The process that selects which Recording Counter will be incremented when R1 is sensed is drastically shortened. From the start, the variable J is incremented every T seconds. When R1 is sensed, CJ, the Recording Counter whose position is indicated by the value of "J" will be incremented. Thus, as before, when a response is sensed, a specific Recording Counter will be incremented according to the length of the interval that has elapsed since the last response.

The SKED program corresponding to the above state graph follows:

```
S.S.1,
S1,
          R1:F2(J,1)  ---> S2
S2,
          R1:CJ;F2(J,1)         ---> S2
          5":F1(J,1,10)  ---> S3
S3,
          5":  ---> S1
          $
```

Note that in the first transition of S2 it is important that "CJ" comes before "F2(J,1)" in the output list. If this were not so, if the order were reversed, then the variable, J, would always be set to 1 before CJ could be executed with the result that on execution of CJ, J = 1 and only C1 would be incremented. Note, also, that the I.R.T. distributor starts after the first response of the session and that this response initializes J to 1. If this step had been omitted, the initial value of J would have been zero and the latency of the first response after the start of the session would be entered into Recording Counter Zero.

2.5.5 Problems With Variable Channel Identifiers

If a variable is assigned to an external or Z output identifier, the user must remember that the form an output identifier has in the source language is different from the form it has in the actual SKED run-time program. When an external or Z output expression is compiled, the identifiers are transformed into 12 bit binary values in which the positions of set bits are determined by the values of the identifiers.

For example, "ON 1" will yield a binary value with the first bit on the right set: ØØØ ØØØ ØØØ ØØ1. Similarly, "ON 3" will yield a value with the bit set that is three in from the right. A list of identifiers will be converted to their position oriented binary values, then merged by a logical OR operation. Thus, "OFF 1,2,4,5" will yield the binary value ØØØ ØØØ Ø11 Ø11.

In summary, any external or Z output expression will generate one 12 bit channel identifier word (in addition to the call to the output routine) in which a bit has been set for each identifier in the expression and the positions of the set bits will be governed by the values of the identifiers.

Since the ASSIGNMENT FUNCTIONS must work with post-compilation values, any assignments to a variable output identifier must take the above format into account.

Example:

Given some output expression somewhere in a SKED source program:

```
    OFF N; ON M; ZK
```

If the user wants this to execute

```
    OFF 2; ON 5; Z1Ø
```

he must assign appropriate <u>octal</u> values to the variables in the output list. If "OFF N" is to turn off channel 2, the variable N must be given the binary value ØØØ ØØØ Ø1Ø. In octal this is ØØØ2. Thus the function to properly set "OFF N" will be "F2 (N,0ØØØ2)." Similarly, "ON M" and "Z K" can be given proper values by F2 (M,0ØØ2Ø); F2 (K,01ØØØ). Furthermore, "N" in "OFF N" could be made to indicate channels 5,6,7,8 by function F2 (NOØ34Ø).

An alternative format for stimulus activation or deactivation is "ON OXXXX" and "OFF OXXXX" where XXXX is a four digit octal integer. This format will cause compilation of the call to the "ON" or "OFF" function, followed by the twelve bit binary value represented by XXXX.

2.5.6 Linking Special Purpose Machine Language Programs with SKED Programs

Some users will find it necessary to write special purpose subroutines in assembly language (PAL 3) to handle peripheral devices requiring special output programs, to control analog input devices or other non-digital types of input signals, or to execute various other functions which the SKED R.T.S. does not handle as a matter of course. Such a special purpose subroutine can be accessed by a SKED language program through the use of the "F3" output function whose format is:

```
    F3 (N, ARG1, ARG2, ARG3 ---)
```

This will be compiled as a JMS instruction that will transfer the operating program to the special-purpose subroutine, which should be written after the compilation of the SKED program.

Following the JMS instruction will be a series of arguments. The first will be a bistable value (either ØØØ1 or ØØØØ) indicating whether there are any time expressions used as arguments in the function. This will be explained in greater detail in section 5.3.

The second argument will be the value or expression represented by N that will be interpreted as the starting address of the special purpose subroutine. If N, in the F3 function, is an octal or decimal integer, its binary equivalent will be compiled. N can also be an expression in the form FA + K or FA - K where K is an octal or decimal constant. The compiled value will be the binary equivalent of the final address of the program being compiled plus K (where K is taken to be a signed integer).

The remaining arguments of the JMS will be a list of twelve bit or twenty-four bit values corresponding to "ARG, ARG2, ARG3 ---."

Example:

```
    R1:  F3 (FA+1,-3,3)  ---> S2
```

If this transition is executed, a special subroutine will be called whose starting address will be FA + 1 and which will use the values -3 and 3 as its parameters. See section 5.3 for more format information.

2.6 THE TRANSFER EXPRESSION

The TRANSFER EXPRESSION, the last of 3 sections in a TRANSITION EXPRESSION, indicates what state or condition of the SKED program will be activated next.

The TRANSFER EXPRESSION has the form:

```
    ---> SN
```

N is the number of the next state to be activated (Note that the stop command causes all stimuli associated with the state program to be turned off.); or

```
    ---> STOP
```

when the process is to be halted completely; or

---> SX

to indicate a "pseudo-transition."

It is important to realize that when a state program is being run under the R.T.S., the transitions in each state set are not executed simultaneously. On the contrary, transitions are executed as each state set is examined by the R.T.S. in order of mention (top to bottom) in the state program!

One consequence of this is that if a "STOP" is executed and one or more state sets follow the "STOP" in the state program, the latter will not be scanned and any transitions therein that should theoretically occur simultaneously with the transitor to "STOP" will never be executed. This means that it is wise to place all stop commands near the end of a state program where their execution will not interfere with the final operations of other parts of the state program.

2.7 THE "PSEUDO-TRANSITION"

On occasion it is desirable to execute some output function, contingent upon a given event, while remaining in a state with the state's clock and event counters held constant. For example, in a fixed interval (F.I.) reinforcement schedule it might be useful to record unreinforced responses. To do this, it would be necessary to listen for responses in State 1 while awaiting the end of the interval (see Figure A below). When such a response is heard, a transition could be triggered whose sole purpose is to increment a Recording Counter and return to the waiting state without administering a reinforcement (see Figure B). However, in the act of re-entering the waiting state the timer for that state will be re-set.

In such a case, then, it is desirable to have a "pseudo-transition" in which output functions can be executed when a certain input is sensed yet no counters or clocks are re-set; in actuality, the state from which the pseudo-transition springs is never deactivated. It just is suspended momentarily while counting or some other output operation is executed. The suitably modified F.I. schedule is represented in Figure C.

The following SKED program is written from that state graph.

Figure A: A simple F.I. schedule

T = 15"

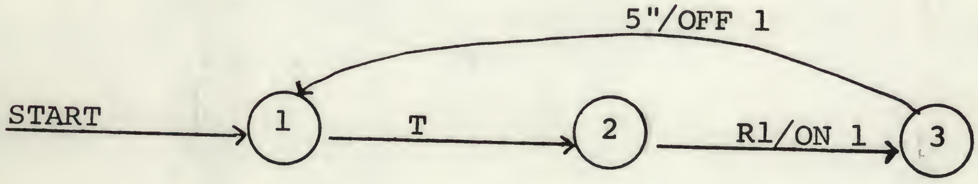
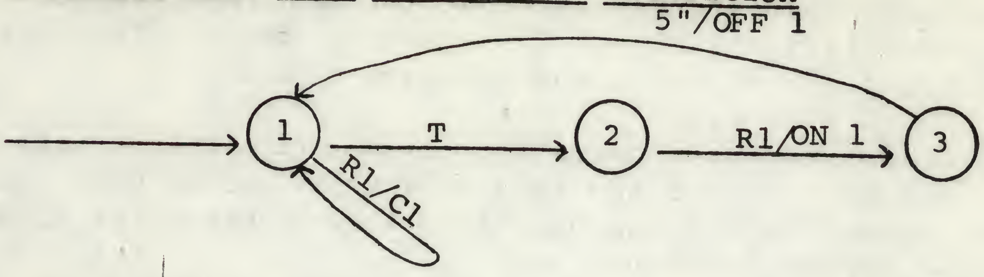
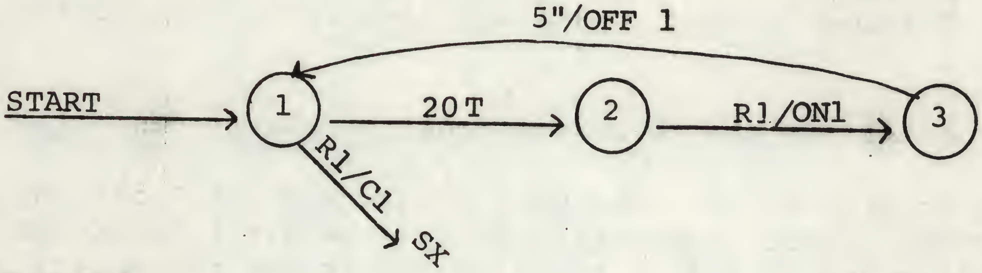


Figure B: A simple F.I. with theoretical transition



Theoretical transition to cause an unreinforced response to be recorded in Cl. Note that the response initiated transition from STATE 1 makes this a DRL schedule rather than E.I.

Figure C: A simple F.I. with "SX"



T = 20"
The "SX" is used to say "do something without changing states."

```
S.S.1,
S1,
              20" ---> S2
              R1:C1 ---> SX

NOTE THAT "SX" IS USED ABOVE RATHER THAN "S1."

S2,
              R1:ON1---> S3

S3,
              5" ---> S1
              $
```

3. THE SKED COMPILER

3.1 GENERAL PROCEDURES

The SKED compiler is a 4K program that loads into locations Ø-4177 of any memory field and uses the remainder of the memory field (up to 7577) to store comoilation tables. The starting address is location Ø2ØØ of the memory field in which the compiler resides.

SKED comoilation is a two oass ooeration in which the first pass is devoted to setting up oointers and tables and the second pass punches the SKED object tape. PASS 1 has no output exceot for error codes, which are orinted on the Teletype.

3.1.1 Program Preparation and Formatting

The compiler is programmed to accept SKED source language expressions through either high or low speed paoer tape readers. Paper tape programs may be prepared either on-line using some editor program or off-line using the Teletyoe reader, punch and keyboard in the editing operation.

The following conditions and allowances must be observed:

1. A SKED program source tape must always have at least one CARRIAGE RETURN/LINE-FEED combination before the first STATE SET LABEL. If this is not included, the first STATE SET LABEL will be skipped and the program will not compile correctly.

2. A SKED source taoe does not need to be terminated by a dollar sign($) when the high speed paper tape reader is the compilation input device. However, while the character is not necessary, its presence will orevent random characters from being stored in the text buffer as a result of a faulty reading at the end of the paper tape.

3. In a SKED orogram spaces have absolutely no significance and are completely ignored by the compiler. For this reason, the user of SKED may feel perfectly free in using spaces for formatting convenience.

4. Although the "TAB" character is not recognized by the SKED comoiler as a legal character, TABS may be placed, for formatting convenience, in a source language program if they are always followed by a "RUBOUT" character. Since this is standard practice within the PDP-8 SYMBOLIC EDITOR and PIP, no problems will arise as a result of using this format with PDP-8 systems. Failure to include a RUBOUT after a TAB in a source tape will cause the TAB to be treated as an illegal character.

3.1.2 Input/Output Selection

The user may, at the beginning of each compilation pass, set Switch Register (on the comouter console) bits 1Ø and 11 to indicate which input and output devices will be used to read source and punch object tapes. The switch settings are as follows:

Bit 1Ø selects the input device -
    Bit 1Ø = Ø indicates Teletype reader will be used
    Bit 1Ø = 1 indicates high speed reader will be used

Bit 11 selects output device -
    Bit 11 = Ø indicates output of SKED object on Teletype paper tape punch
    Bit 11 = 1 indicates output of SKED object on high speed paper tape punch

Error codes are always printed on the Teletype printer and are not affected by the I/O select switches.

3.1.3 Parameter Setting

There are several data relevant to compilation which must be set at compile time through a short, terse interactive dialog with the computer. This dialog is executed after the end of PASS 1, before starting PASS 2. The specific parameters and their significances are explained below.

Parameter 1: Box Number

The SKED run time system can supervise up to ten experimental stations (each having 12 response inputs and 12 stimulus outputs) numbered from Ø to 9 (decimal). The object tape must indicate which of the ten stations the program will control. The compiler requests this datum in the first query of the dialog by printing:

```
Box # =
```

The user must reply by typing the appropriate decimal integer, between Ø-9, inclusive, followed by a RETURN. Errors can be corrected anytime before the RETURN is typed by typing a RUBOUT or any other non-digit character. The compiler will retype:

```
Box # =
```

and wait for correct input. Note that any numeric input besides Ø-9 will be considered incorrect and will automatically evoke the error procedure.

Parameter 2: Starting Address

Within certain limits the arrangement of SKED object programs in the computer memory is completely subject to the judgment of the user. The limits are:

1. The SKED R.T.S. loader will not permit a program to be loaded over itself. R.T.S. resides in locations Ø-2577.

2. The SKED R.T.S. loader will not permit a SKED object program to be loaded over another SKED object program which it (the loader) has already loaded into core.

Since the SKED object program is not relocatable, the program starting address must be included on the object tape as a function of compilation. To allow flexibility of source programs the compiler treats this datum as changeable and requests that it be typed during the parameter setting dialog.

The Teletype will print on a new line:

```
SA =
```

The user must reply by typing a 4 digit octal integer indicating where between locations 26ØØ and 7777, inclusive, the program should start. This number must be followed by a RETURN.

The compiler will only recognize as incorrect input non-digit characters and the digits 8 and 9. Any other inputs will be accepted - the user must be careful to type an address in the correct range. The errors that the compiler can detect will cause the compiler to retype "SA =" and wait for correct input.

Parameter 3: Number of Recording Counters

The SKED compiler's scanning algorithm is unable to determine how many cells must be reserved at the end of the SKED object program for use as Recording Counters. Therefore, the programmer must supply this datum manually.

The number of cells to be reserved is identical to the highest Recording Counter number unless that counter has been extended by the use of an asterisk in the Recording Counter Expression. In that case the number should be increased by one to indicate the true number of cells needed.

The compiler prints on a new line:

```
#CTRS =
```

and the user must reply with a 1 to 4 digit decimal integer, followed by a RETURN. Note that if there are no recording counters in a program, the user may reply to "#CTRS =" by simply typing a RETURN. Error checking follows the same rules as with BOX # except that the input may be in the range Ø - 4Ø95.

Much care should be taken in entering this datum because entering a value which is smaller than the actual number of cells needed for recording will cause a compiler error during the second pass. Entering a value which is too big will cause a waste of valuable core space.

When all of these parameters have been successfully entered the compiler will compute the final address of the program (which has just been read during PASS 1) in the form.

FA = XXXX

Where XXXX is a 4 digit octal integer, the computer will then halt. At this point the user has an opportunity to re-check the validity of the data that were input during the dialog. If there are errors (note that FA = XXXX is a good indicator of the validity of the input), the user may re-start at location 16ØØ (of the field in which the compiler is residing) to re-execute the dialog. If all is well the user may proceed to PASS 2. (See step 7 in the STEP-BY-STEP compilation instructions.)

3.1.4 Error Codes

When the compiler finds an error in a SKED source program, it prints an error message in the form:

ERROR XXXX AT YYYY

XXXX is a 4-digit octal integer that expresses what type of error has been found and at what location in memory the error was discovered.

YYYY is a 4-digit decimal integer that expresses which printed line in the source program contains the text that produced the error.

The following lists describe the significance of each error number (XXXX). Note that any time the SKED compiler program is modified, some if not all of the error numbers will change because they refer to locations in which calls to the error handling routine are stored. Therefore, after each error number, the symbolic tag which refers (in the PAL III symbolic source of the SKED compiler) to the core location will be listed. This will help programmers who change the compiler to update their error lists.

3.1.5 Step-by-Step Compilation Instructions

Step 1) Load compiler binary tape with the DEC BIN Loader. The SKED compiler occupies locations Ø-4177 in any core memory field.

Step 2) Set the starting address of Ø2ØØ (and relevant data field address) in the SWITCH REGISTER and press the LOAD ADDRESS key.

Step 3) Select input and output devices on S.R. bits 1Ø and 11 (see section 3.1.2).

Step 4) Load the SKED source tape onto the appropriate input device (if low speed reader is being used, turn it on and press the START key).

Step 5) The compiler will now begin reading the SKED source tape for the first compilation pass. As the paper tape is read the characters will not be echoed on the Teletype printer.

The only output during this pass will be error codes, which will be printed on the Teletype. After an error code has been printed, the computer will halt permitting the user to decide whether to continue with the compilation or to stop, correct the source tape and restart compilation at STEP 1. Compilation will be continued when the user presses the CONTINUE key on the computer console.

Step 6) At the end of PASS 1, the parameter setting phase of compilation will begin. If the low speed reader was used in PASS 1, the computer halts. The user must then turn off the Teletype reader and press CONTINUE to proceed. (When high speed reader is used, the compiler enters the next phase automatically without halting.) The computer will print:

```
BOX # =
```

If program errors were found during PASS 1, it is best to diagnose (see error list in Section 3.1.4) and correct them, then return with fresh source tape to STEP 1. If no errors were found, the user must complete the parameter setting dialog (see Section 3.1.3) after which the computer will halt.

Step 7) To start PASS 2, the user should reset the I/O select switches as in STEP 3 (if necessary), put the source tape in the appropriate reader (turn it on), turn on the appropriate paper tape punch and press CONTINUE. This time paper tape will be punched on the selected device as the source is read and compiled. Again, error messages may be printed and the user will have the same alternatives as in STEP 5. At the end of PASS 2, the computer will halt.

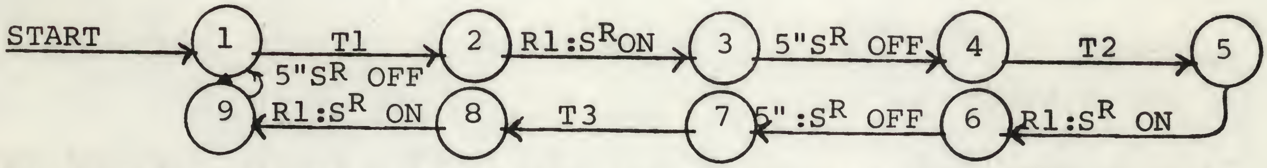
Step 8) The user may now remove his object tape from the output device. To restart the compiler at PASS 1, the user need only select the I/O devices on the SWITCH REGISTER, load the next SKED source tape on the appropriate reader (turn it on), press CONTINUE and go back to STEP 5.

3.2 Space Saving Hints

The basic design of the SKED system involves compilation of state programs that are of variable length; i.e., the amount of core occupied by each program is determined by the number of states, outputs, etc., required by the particular experiment to be conducted. Since there are often a number of alternative ways to construct state diagrams that result in identical performance but differ in the amount of core utilized, it may often be desirable to analyze the state graph for potential reduction in the program length. (These considerations may be of no interest for those possessing machines with more than 4K of memory or who are conducting only two or three simultaneous experiments with 4K.)

Only one general rule applicable to many different experimental situations has been discovered (beyond those that led to the specialized output functions, F1 and F2). This procedure resembles the algebraic process of factoring. It consists of identifying repetitive states within a state set that can be reduced in number by constructing a second state set. For example, consider a variable interval schedule of reinforcement. This procedure consists of a pre-specified string of intervals that change in length from reinforcement to reinforcement.

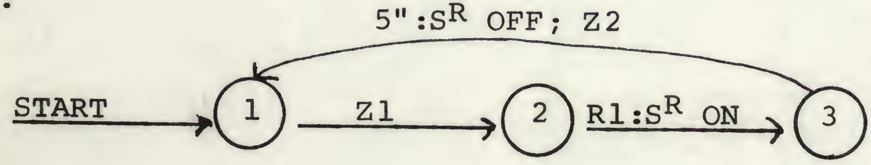
Example:



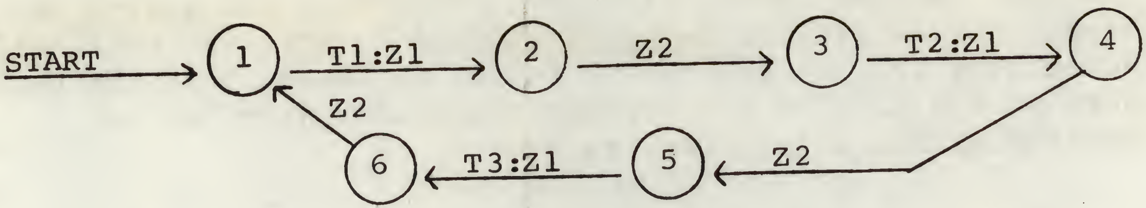
In the example, T1, T2, and T3 are the three variable intervals repeated in sequence during the experiment. It is clear that the state pairs 2,3; 5,6; and 8,9 are identical "set-up" and reinforcement states that differ only as to the next interval to be initiated at the end of reinforcement. If the sequence of intervals is quite long it is evident that reducing the state set into two separate state sets will save considerable core space.

Example:

S.S.1,



S.S.2,



In this example the "waiting state" S1 of S.S.1, the "set-up" state S2, and the reinforcement state S3 have been placed in a separate state set, which is driven by Z1 pulses from the timing state set at a considerable saving of program length. (Note the cross synchronization of state sets, in that the timing state set starts only after the reinforcement has been obtained in State Set 1. This feature may be unimportant if the interval lengths are long compared to the reinforcement duration and latency of the reinforced response, but it is easily programmed if necessary.)

Further space saving may be accomplished in particular state graphs by observing that different input, transfer and output functions require different numbers of core locations for their realization. The following table lists the number of locations for each function so that alternative state programs utilizing different functions can be evaluated in terms of the space required.

TIME AND SPACE REQUIREMENTS OF SKED STATE PROGRAM COMPONENTS

		RUN TIME in Microsecs.	
COMPONENT	CELLS NEEDED	MIN.	MAX.
STATE SET POINTERS (One set per state set):	4 +1 for the first set	-	-
STATE POINTERS (One set per state):	6	-	-
State Transitions (One set per TRANSITION EXPRESSION):	2	73	Add 23 for each event counter in the new state
Pseudo Transitions	1	2	2
Checking of R or Z Events:	4	72 (No transition)	104
R or Z Event Counting (when required):	2	Ø	16

COMPONENT	CELLS NEEDED	RUN TIME IN MICROSECS.	
		MIN.	MAX.
Clock Input (when used):	2 if a time transition is first in a state expression; 3 otherwise.	27	62
Stimulus Output:	2 per list	64	?
Z-Pulse Output:	2 per list	38	40
Recording Counter:	2; +1 for actual counter.	50	88
F1:	6 to change R,Z counters and output identifiers;	225	268
	8 to change time counters	225	298
F2:	4 to change R,Z counters;	83	83
	8 to change time counters	105	105
Event Gating:	5;+1 for each argument after the first	80;+32 for each argument after the first	102;+32 for each additional argument

3.3. COMPILER ERROR CODE LIST

PASS 1 Error Codes

Number	Tag	Significance
Ø512	LABERR	Improper state label or state set label format.
Ø611	INER1	Illegal format in an input expression.
Ø753 1Ø16	INER2 INER3	Illegal character in an input expression.
Ø7Ø5	SAME2	Same input used in more than 1 transition in the current state.
Ø624	SAMTM	A time expression is used in more than 1 transition in the current state.
1Ø22	TAGER1	Illegal variable usage in the input section.
1Ø66	GATER	Illegal TAG for a gating expression.
1Ø76	OUTER1	Illegal format in output list.
1146	FER1	Illegal format associated with assignment function identifier.
1173	STIMER	Bad format in a stimulus expression.
1355	TSETER	Variable used more than once.
3767	ARGERR	Error in argument list (too many arguments).
1425	FORMR1	Bad format for a decimal integer: too many digits or overflow.
1544	FORMR2	Bad format for an octal integer: too many digits, decimal digits or overflow.

PASS 1 Error Codes (Con't.)

Number	Tag	Significance
3233	TIMERR	Illegal format in a time expression
37Ø2	BDTG	Undefined variable in assignment function or bad argument format.
374Ø	CHKER	Illegal argument format. or too many arguments in a function or gating expression

COMPILER
ERROR CODE LIST

PASS 2 Error Codes

Number	Tag	Significance
232Ø	CONMLT-1*	Same as SAMTM.
2346	DOTY+1*	Same as INER1.
255Ø	RECER	Too few Recording Counters specified during dialog.
2652	PSUDR	Illegal use of BLANK TRANSITION.
2633	GATAG+2	Undefined Gating tag.
266Ø	FCMERR*	Same as FER1.
3372	SRCHER	Reference made to undefined state.
3767	ARGERR	Undefined Variable.
Ø512	LABERR	Same as in PASS 1.
1425	FORMR1	
1544	FORMR2	
37Ø2	BDTG	
374Ø	CHKER	
3233	TIMERR	

* These diagnostics refer to errors that will be found in PASS 1 and will generate the codes indicated in the "significance" column.

4. OPERATION OF THE SKED RUN-TIME SYSTEM

4.1 General Description of the R.T.S.

The SKED Run Time System is a 4K program (on binary tape) that loads into locations Ø-2577 (this varies with the size of the device handler) of memory field zero and has a starting address of Ø2ØØ. The remainder of memory field zero is occupied by SKED state tables being run under the system.

The R.T.S. is designed to load, run, and monitor up to ten SKED state tables; controlling one experimental station per state table. System control is a function of Teletype commands that initiate six basic operations. The commands are:

L — LOAD the state table into the system.

S — START the state table that was most recently LOADED for ABORTED.

AN↵ — ABORT operation of the state table controlling BOX #N; clear all stimulus flops associated with that station.

RN↵
1, 2, 3--↵ — Stimulate activation of response RN in the state programs controlling boxes 1, 2, 3.

D
1, 2, 3 ↵ — DUMP the contents of all Recording Counters. 1, 2, 3, etc. are the box or station numbers.

↑ — Terminate operation of all currently running stab tables; clear the stimulus flops at all stations.

While using the R.T.S., papertape I/O devices can be selected by setting switch register switches 10 and 11:

Switch 1Ø designates the input device for LOAD operations -

Bit 1Ø = Ø:	Teletype reader will be the input device
Bit 1Ø = 1:	High speed reader will be the input device

Switch 11 designates the output device for Dump operations-

Bit 11 = Ø:	The teleprinter will be the output device
Bit 11 = 1:	The high speed paper tape punch will be the output device

4.2 Getting on the Air

Getting on the air with the RUN-TIME SYSTEM is a simple matter of loading the R.T.S. binary tape with the DEC BIN LOADER, then starting the computer at location Ø2ØØ. Starting at this address always clears all stimulus flops and leaves all experimental stations ready to receive state tables. The system will type:

```
*↑
*
```

Signifying that it is ready to accept any of the above commands. Hereafter, work with the R.T.S. will be a process of loading, starting, aborting state programs through the control of the Teletype. After each command has been executed, the system will again type a CARRIAGE RETURN/LINE FEED combination followed by an asterisk, then will await its next command. If any character is typed, other than the carriage return or the command characters described above, the system will print a question mark then re-enter wait mode after printing an asterisk.

The next section describes for each keyboard command the calling sequence, possible error conditions, the command's effects and ways by which a command can be aborted. Note that the symbol "↲" signifies a CARRIAGE RETURN typed by the user.

4.3 The Keyboard Commands

LOAD (L)

To load a SKED state table (output of the SKED compiler), place the program's binary tape on the appropriate paper tape reader, indicate which reader will be used by setting Switch Register bit 1Ø (bit 1Ø = 1 indicates high speed reader input; bit 1Ø = Ø indicates Teletype reader input) and type "L". If a high speed reader is being used, the papertape will be read immediately. If the low speed reader is being used, it must be turned ON before the tape can be read.

The R.T.S. loader will check for several types of errors during the loading process. They are:

a. When the state table being loaded will control a station that already has a running state table controlling it.

b. When the state table being loaded partially or completely overlays a state table controlling another station.

c. When the state table being read begins to overlay the R.T.S. program.

d. When a character is read whose value is inconsistent with the expectations of the R.T.S. loader.

e. When the checksum value at the end of the state table is incorrect.

If any of these errors occur, the R.T.S. will immediately stop reading the program tape and print, on the Teletype, an error message in the format ERROR NN where NN is a two digit error identifier value. (Reference to the error code list in section 4.4 will reveal which error has occurred). After the error message has been printed, the R.T.S. will return to the ready state, awaiting further commands.

The user can abort a LOAD operation at any time by typing the letter A.

The R.T.S. will react by ceasing to read the state program, then typing:

```
                            * ERROR 15
                            *
```

Thereafter, the R.T.S. can accept more commands. The status of tables previously running will not have been changed.

Note that a state program that has just been LOADed must be STARTed before any other LOAD or any ABORT is executed. If this is not done, the data required to START a program will be lost and the program will have to be reLOADed before it can be STARTed.

START (S)

After a state table has been LOADed, the experimental session can be started by the command "S." When this command is typed, the system will re-arrange its internal pointers so that the state table most recently LOADed or ABORTed will be put into operation. It is not necessary that this command have an argument to designate the station that will be controlled by the state table being started since the previous LOAD or ABORT command will have supplied this information.

ABORT (AN↲)

To interrupt the operation of a state table, the user must type the command AN↲ when N is the number of the box being controlled by the state program. All stimulus flops associated with that experimental station will be turned off and the state table will stop operating; it will be unable to react to clock or response events. The START command will then cause the state table to restart where it left off. The stimulus status, however, cannot be restored by the START command.

When a typing error is made during the input of an ABORT command, the user can erase the command by pressing the RUBOUT key on the Teletype at any time before the command terminating CARRIAGE RETURN has been typed. The R.T.S. will echo a back slash (\) and return to wait mode after printing an asterisk. The user can then try again to type the correct command.

Note that the ABORT command erases internal system pointers that allow a program to run, and at the same time saves the contents of the pointers in auxiliary cells. Therefore, if two ABORT commands are given, only the run-status pointers from the last ABORT will be saved. The data from the first will be lost and the state program involved will not be restartable. Note, also, that if an ABORT command is given for a box which presently has no state program running, no action will be taken but a question mark will be printed before the system returns to wait mode.

Since the ABORT clears the run-status pointers for the box designated by "N," that box will be available for receiving another state table via the LOAD operation. In other words, an ABORTed state table can be LOADed over.

RESPONSE simulate (RN↲
1, 2, 3 ···↲)

In the past, experimental systems have had response flops from several stations wired together so that the experimenter could, by pressing one button, activate the same response event at each station, and, in this manner, coordinate the starting of the reinforcement schedules at each station.

The keyboard monitor system offers the experimenter this capability without the problems of special circuitry. To simulate simultaneous activation of response event N at any station, type:

RN↲

then type a list of station numbers (Ø-9) in the form

1, 2, 3, 4··· ↲

describing which stations the simulation should affect. Note that only those stations in which state programs have been LOADed and STARTed can be coordinated in this manner.

As with the ABORT command, the user can erase erroneous Teletype input for this command, at any time before the final carriage return is typed, by typing a RUBOUT. The effect will also be the same: the system will print a back slash then return to wait mode after printing the asterisk. The user will then be able to type the command correctly.

DUMP of Recording Counters (D
1, 2, 3 ··· ↲)

To produce a complete listing of values in all Recording Counters from one or more experimental stations, type the command "D." After the R.T.S. automatically prints a RETURN/LINE FEED the user may type a list of station numbers indicating from which experimental stations data should be dumped. The station numbers should be separated by non-digit characters and the list must be terminated by a RETURN. The entire command format will appear as follows:

```
D
  1, 2, 3  --- ↲
```

This command can be issued at any time whether or not the stations listed have actively running state programs controlling them, and its execution will not interfere with the execution of any state programs. Before the command is issued, the desired output device must be selected by setting switch 11 on the Switch Register (bit 11 = 1 indicates high speed punch; bit 11 = Ø indicates low speed punch) and the appropriate output device must be turned on. Output for each station will be in the form:

```
BOX # L
CTRS. (NNNN, MMMM)
(XXXX) (XXXX) (XXXX) (XXXX) (XXXX) (XXXX)  ..........
```

(The characters XXXX represent the specific contents of cells being listed.) When the listing is finished, the R.T.S. will print an asterisk (*) indicating that it is ready to accept further commands.

It is possible that one of the boxes for which data is being dumped will never have had a state table controlling it or the state program will have had no Recording Counters. In these cases only the dump heading for that box will be printed, after which the system will proceed to dump the next box or enter the waiting state if there are no more boxes to be DUMPed.

The user may cause Recording Counters to be cleared as their contents are DUMPed by setting Switch Register switch Ø to the "ON" position.

Errors can be corrected at two stages:

1. As with the A and R commands, the user can erase an erroneous command, at any time before the final CARRIAGE RETURN is typed, by typing a RUBOUT. The effect is the same as with the above command.

2. The actual DUMP operation can be aborted if an "A" is typed. The system will print the ERROR 15 message and return to wait mode.

The General System Clear (↑)

In a panic situation all stimulus flops can be disabled and all state table operation aborted if the user types the up-arrow character (↑). This is typed by pressing the SHIFT key while typing "N."

Functionally, the command is equivalent to stopping the computer and restarting at address Ø2ØØ. The system prints an up-arrow then returns to wait mode after printing an asterisk. None of the state tables that were running before the command was given will be restartable. Everything must be reloaded and STARTed.

This function is good only for use in emergencies or at the end of the day.

4.4 The R.T.S. Error Codes

ØØ Tape not mounted on leader trailer at beginning of load operation or illegal character read.

Ø1 Tape being loaded will control a station that already has a state table running it.

Ø2 The starting address of the state table being loaded is too low.

Ø3 The state table being loaded threatens to overlay a state table already in operation.

Ø4 The state table being loaded exceeds the upper storage limit (location 7777_8).

Ø5 Checksum error.

1Ø Command requested operation on a nonexistent box.

15 Load operation ABORTed from the Teletype keyboard.

4.5 The SKED Debug System

The SKED Debug System is a modified version of the SKED R.T.S. that allows simulation of all response and clock events through the Teletype keyboard and reports, on the Teletype, the status of stimulus channels and state programs when important changes occur. Its commands and operations are the same seen in the R.T.S. but include additional Teletype functions. Since all stimulus/response functions are Teletype oriented, the hardware requirements are the same as for the SKED compiler: a PDP-8 family computer, a Teletype, and optional high speed paper tape reader/punch. Note that, since the clock interrupts are simulated by keyboard commands, no problem is confronted when running the program on a PDP-8/s.

The SKED Debug System loads in locations Ø-2777 of memory field zero and uses the starting procedure described for the SKED R.T.S. After a state program has been loaded and started in the Debug System (see sections 4.2, 4.3 for basic operations), all events must be simulated through two keyboard commands. These may be entered whenever the Debug System is in wait mode and has printed an asterisk at the Teletype's left margin. The commands are:

```
RN
B1,B2,B3 ---
```

and

```
T (time expression)
```

The first command simulates the response event RN at boxes B1,B2,B3 ---. This is the standard R.T.S. command to simulate a response and is described in section 4.3.

The second command simulates passage of an interval of time by imitating clock interrupts and executing the operations indicated by the state tables in operation. The format of this command consists of the letter T followed by a time expression in a format closely similar to that described in section 2.3.1. The terminating character for this command is either a double quote as in

```
T2.Ø5"
```

or a CARRIAGE RETURN, which is necessary only when the time expression ends with some expression of minutes, such as

```
T6.5Ø'.
```

Time event simulation affects all operating experimental stations.

When the simulation command has been entered, the system will simulate the response or time events and print reports whenever there is a change of either status of stimulus equipment or of states in state programs or when a Z pulse is generated.

A change-of-stimulus report will be a printed message in the form

```
ON N1 N2 N3 ---
ACTIVE N1 N2 N3 ---
```

or

```
OFF N1 N2 N3 ---
ACTIVE N1 N2 N3 ---
```

N1 N2 N3 represents a list of decimal integers indicating what stimulus channels are being changed.

The first line of each message indicates which stimuli have been activated or deactivated. The second line indicates those channels still active after the change has occurred.

The Z pulse activation report will be a printed message in the form:

```
ON Z N1 N2 N3 ---
```

N1 N2 N3 --- indicate which Z pulses have been activated.

State transitions will be reported in the format:

```
S.S.N
STATE M
S.S.N+1
STATE M
S.S.N+2
   :
   .
```

N is the state set in which the change is occurring.
M is the number of the state activated in the process of the change. Whenever a state change report is printed, it will include reports on all state sets in the given state table -- no matter whether or not there have been state changes in all state sets of the table.

Whenever a set of stimulus reports status or state reports is about to be printed (i.e., at the beginning of each transition), a box number report is printed in the format

```
#N
```

N is the box number.

This will designate with what box the next state or status change reports should be associated. When an "SX" is executed, the box number report will be printed with none of the other reports following it.

Note that the R.T.S. Debug Program has no error codes aside from those used by the R.T.S. This means that errors will only show up when a state program does not work in a way expected by the user.

For a sample run of a state program simulation see Appendix A.

The SKED Debug System does not have one entire PAL source tape; rather, it can be assembled with the source of the SKED R.T.S. proper and the source tape entitled R.T.S. Debug PATCH. Thus, if a change is ever made to the R.T.S. proper, a new Debug Program may be made by assembling the corrected R.T.S. source and the Debug Patch source (with any corrections made necessary by the correction in the first source tape).

Note: Assembly of the Debug System with PALIII (or any variants thereof) as explained above will generate the assembly error "US STIMO." This is no occasion for concern since the binary output will properly run in spite of the error.

5. VARIED EXPERIMENTAL HARDWARE SYSTEMS AND ADAPTATION TO THEIR REQUIREMENTS

5.1 BASIC INFORMATION ABOUT THE EXPERIMENTAL HARDWARE

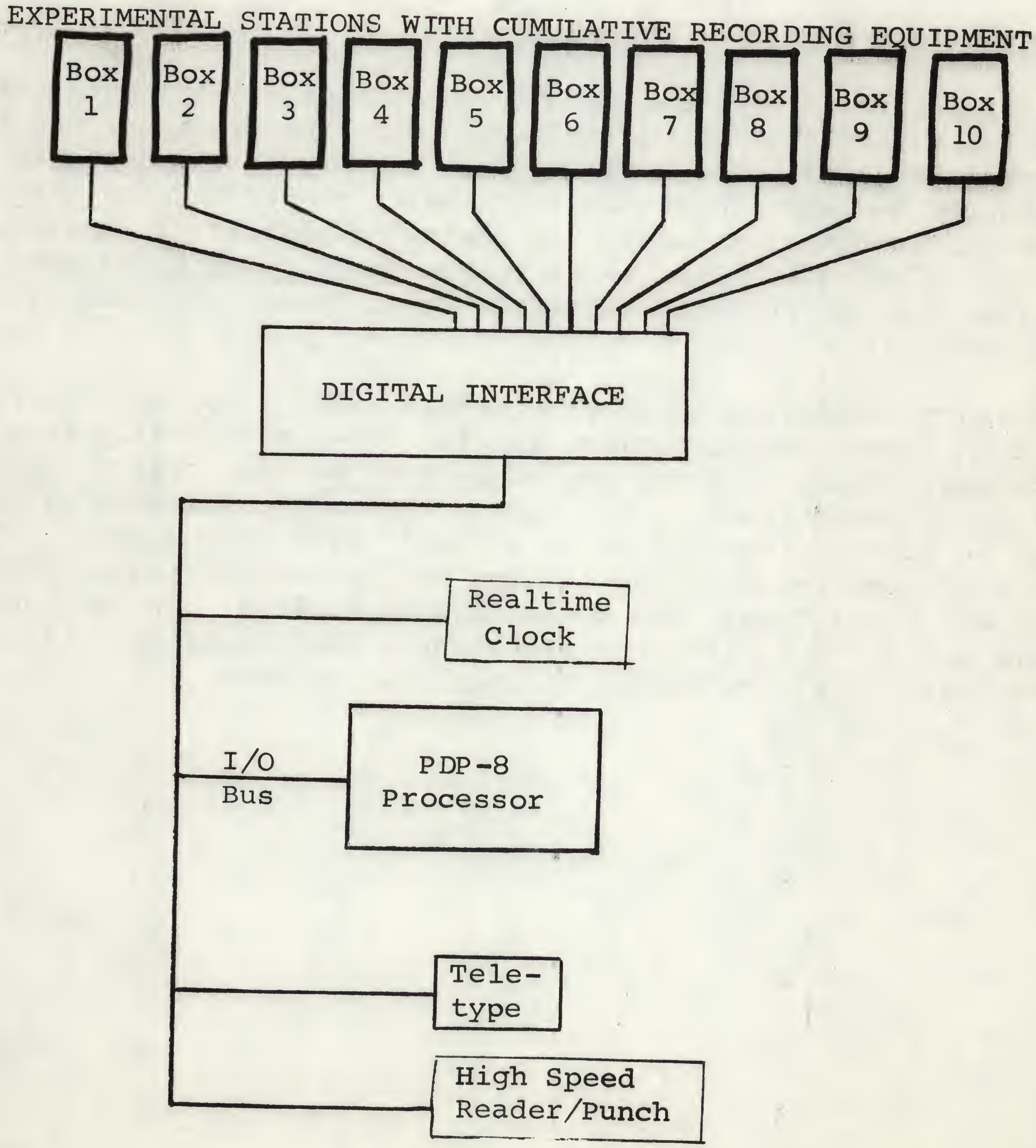
In the list of hardware requirements for the operation of the SKED software system (section 1.1.2) is a category described as "experimantal hardware." Under this category are included:

a. The experimental stations which consist of electro-mechanical systems directly responsible for presentation of stimuli and monitoring of responses. Skinner boxes, monkey chairs, etc., are such systems.

b. Various cumulative recording devices.

c. A system of digital logic that acts as an interface between the above systems and the PDP-8 processor.

These hardware components will ideally be configured in the manner depicted in the following diagram.



Since standard devices have been designed for the systems described in categories a and b, above, it is category c that will be the point of interest of this section.

The digital interface logic must consist of:

a. Arrays of flip flops and logic gates designed to store stimulus and response data going to and from the computer. This logic can be seen as the basic interface system and the design principles pertaining to it are fairly general and simple.

b. Whatever switch filters and output drivers are necessary to achieve signal compatibility between the interface proper and the experimental stations. This portion of the interface will be very variable and dependent upon the requirements of the hardware of the experimental stations.

5.1.1 Notes on the Digital Interface

The digital interface will consist of two sets of digital data buffers plus the logic required for communication with the processor and the experimental stations. The interface can be very easily designed and built with DEC M-series logic modules. Refer to the DEC Logic Handbook[2] for all information about the modules' capabilities, power requirements and prices. Also, refer to the Small Computer Handbook[3] for detailed data on building interface logic to be used with the PDP-8 I/O Bus.

One set of data buffers will store response data; i.e., when a response occurs at an experimental station, it will cause a flip flop in a register to be set. These response registers must be connected to the AC input lines of the computer's I/O bus so that the conditions of the flops may be strobed into the AC as a result of an IOT command. The flip flops in each register should also be inter-connected so that one IOP pulse can clear them. See the following logic diagram and accompanying notes for a clearer explanation. The points of interest to which the notes refer are marked by encircled numbers.

NOTES FOR SIX BIT RESPONSE BUFFER LOGIC DIAGRAM

Note 1: The component enclosed in the box is a pulse generator. One of its input leads is connected to a filtered response switch whose logic level will change from high (+3 volts) to low (ground) when it is registering a response. The first stage in the component will convert the low-going level change to a high pulse. The second stage, a NAND gate, will invert the logic condition of the pulse so that the entire component generates a logic-low pulse every time the response switch changes logic levels from high to low. (one M606 module contains eight set-reset flip flops)

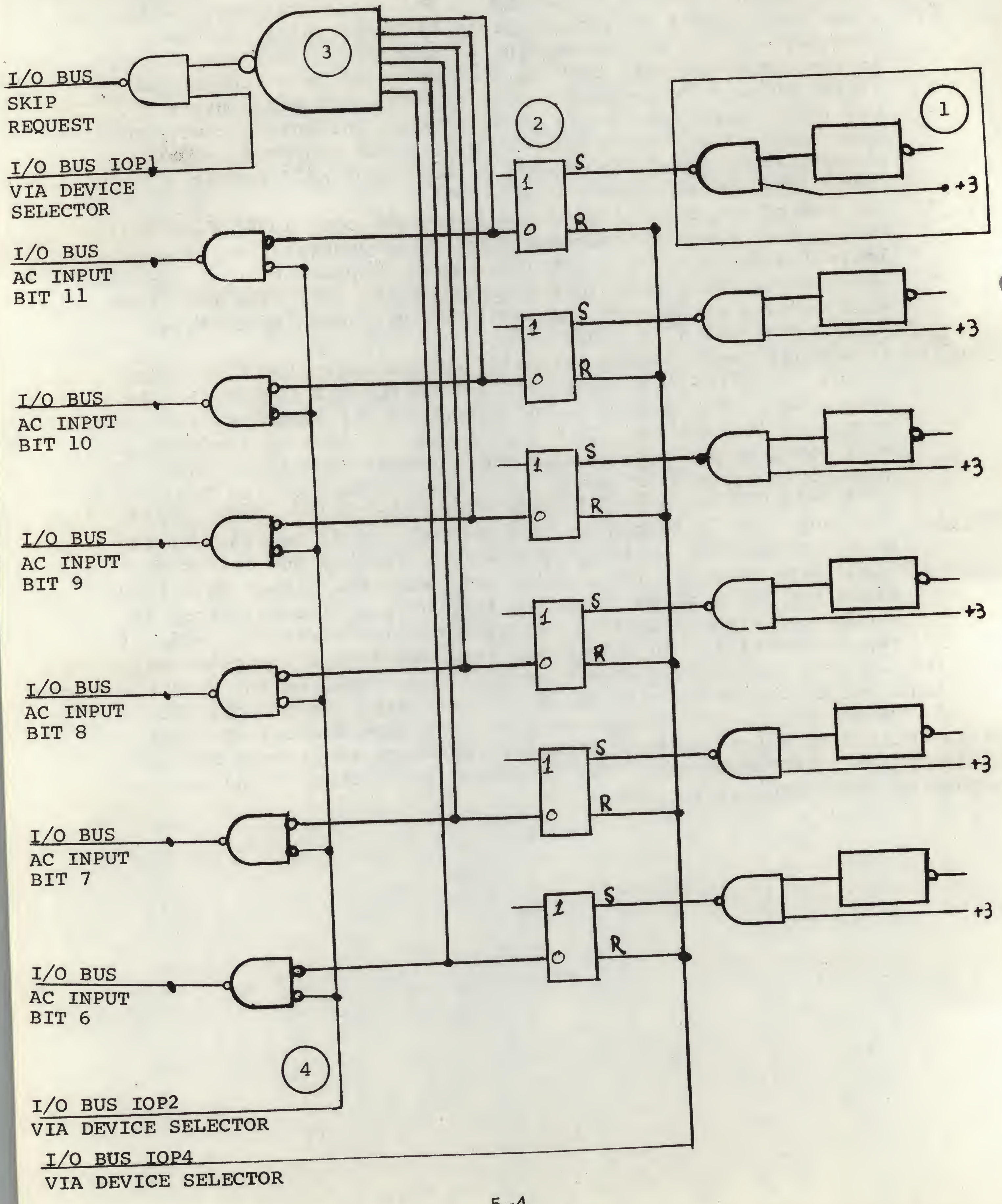
Note 2: The row of set-reset flip flops below the note identifier will be set each time the corresponding pulse generator emits a logic-low pulse. Their settings, then, represent which response lines have been activated since the last time the flops were cleared by the IOP-4 pulse. (one M203 module contains eight set-reset flip flops)

Note 3: This multi-input "NAND" gate will emit a logic-low level when all of the flip flops have been set to the off condition. As soon as any flop comes on, the output of the "NAND" will change to logic-high. This high level, in combination with a high IOP-1 pulse, will enable the adjacent two-input "NAND" gate to activate a processor SKIP request through the I/O Bus. (one M119 module contains three such multi-input "NANP" gates)

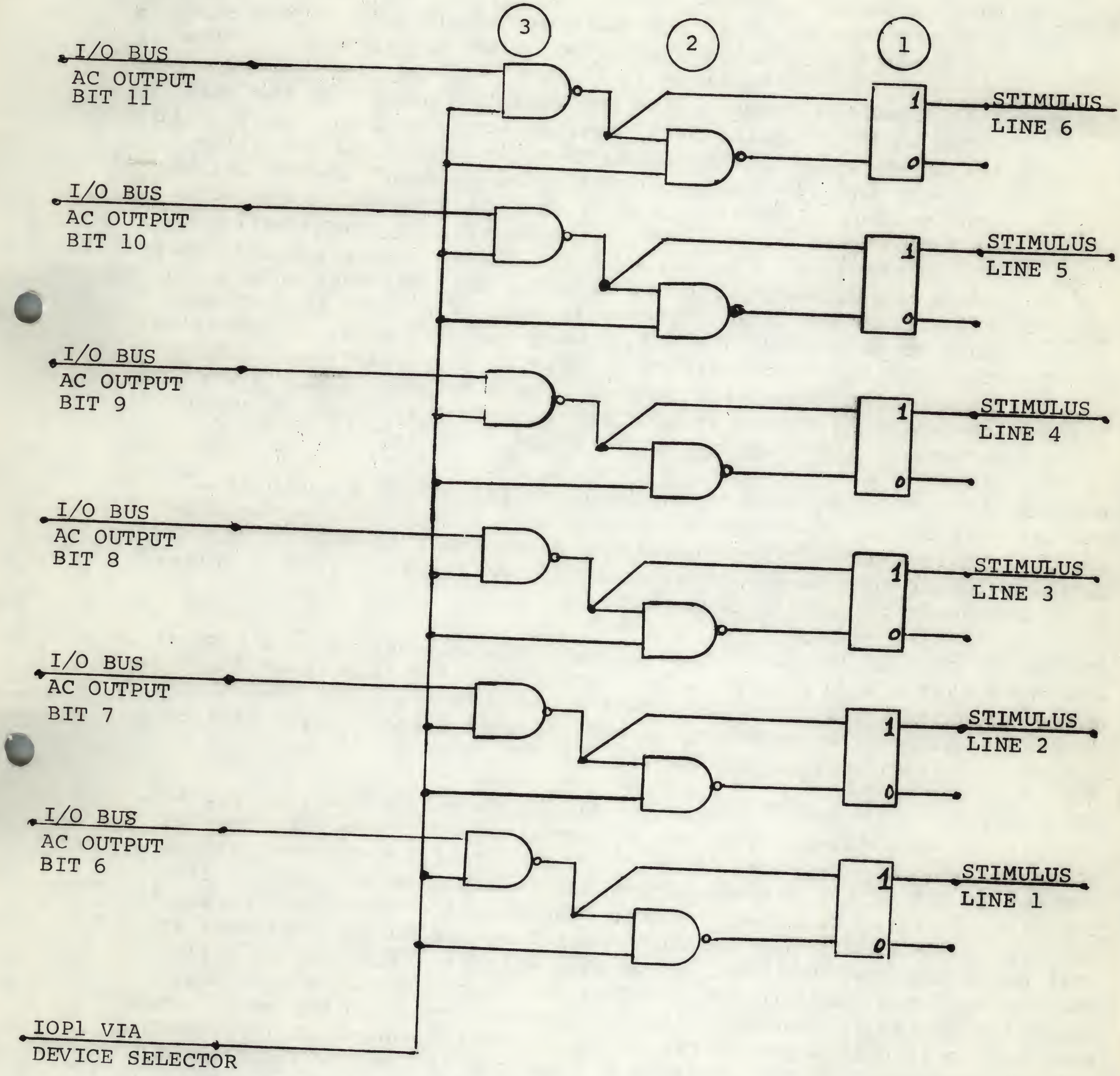
Note 4: When any flop is turned on, its condition will be transferred to a bit position in the ACCUMULATOR through a non-inverting "AND" gate when the IOP-2 pulse activates the gate. Note that since the IOP-2 pulse is coming through a device selector, it can be modified so that its logic-condition meets the needs of the application. In this case, IOP-2 will be a logic-low pulse. (This serves of non-inverting "AND" gates will be found on the M624 bus driver)

The second set of registers will store stimulus data coming from the ACCUMULATOR and hold stimulus equipment, at the experimental stations, active until flip flops in these stimulus registers are turned off by the execution of IOT commands. The following logic diagram and accompanying notes explain further.

LOGIC DIAGRAM OF SIX BIT RESPONSE BUFFER LOGIC



SIX BIT STIMULUS BUFFER LOGIC



NOTES FOR SIX BIT STIMULUS BUFFER LOGIC

Note 1: This column of six set-reset flip flops is the six bit register that stores active/inactive data pertaining to stimulus channels and keeps active devices turned on. (one M203 module contains eight of these flip-flops.)

Note 2: This column of "NAND" gates controls the setting of the flip flops in the stimulus data register (explained in Note 1). If the ACCUMULATOR bit connected to a given gate has the value "1" when the IOP-1 pulse is activated through a device selector (not visible), the flop that is connected to gate's output will be turned on. If the AC bit is off, this gate will not affect the flop. (The entire set of "NAND" gates for the operations explained in both note 2 and 3 will be found on one M101 module)

Note 3: This column of "NAND" gates will clear the flops that correspond to AC bits with Ø values when an IOP-1 pulse is generated through the device selector. This is the operational complement of the gates explained in Note 2 because when AC bits have the value 1, the corresponding gates in this column have no effect on the stimulus buffer flops.

The over-all effect of this circuitry is to set or clear the bits of the stimulus buffer whenever the command 6XX1 is executed in the processor. XX is a six bit device selection code that is decoded by a device selector that decides whether or not to permit the IOP's pulses to be active in the above circuitry.

The number of flip flop registers of each category needed in a digital interface system will depend on how many channels will be used for interchange of data. Where more than twelve channels are used in each direction, it will be necessary to have several response data registers and several stimulus data registers.

An example of this type of situation is the interface built by Dr. A.G. Snapper[1b]. That interface allows communication with ten experimental stations, each of which has three response channels and eleven stimulus channels. All of the response channels feed into three response data buffers; the three response channels emanating from each box represent R1, R2, R3 for that box and each ten bit response buffer indicates at what boxes the response event associated with it (R1,R2,R3) is active. One register has its flip flops set by the R1 channel from each box, another register is changed by the R2 channel from each box, etc. Whenever a flip flop in any register is set a program interrupt is requested and the Response Interrupt Service Routine can find out which R event interrupted by executing a SKIP chain (each buffer will set a different SKIP flag), then read and clear the indicated response buffer, finding out at what station the response occurred.

Stimuli are administered at any station in Dr. Snapper's system by loading the ACCUMULATOR with the eleven bit stimulus pattern that describes what stimuli should be active at the station, then executing an IOT command that causes the content of the AC to be strobed to the stimulus buffer that controls that station.

5.1.2 Switch Filters and Output Drivers

Almost never will a response lever or stimulus device have electronic characteristics that conform to the requirements of M-series logic modules. Response levers will invariably have jagged signals and stimulus devices will usually draw too much power.

DEC K-series Logic Modules[4] can handle the problems of electronic conversion between the real-world and the digital control interface discussed above. These modules include a variety of switch filters and output drivers that are logic-compatible with M-series modules and are pin-compatible within various K-series applications. Note that K-series switch filter outputs can be directly wired to the M606 pulse generator (six pulse generators per card) with no conversion gating. Similarly, input to the K-series output drivers can be the output lines of M-series flip flops.

5.2 ON PROGRAMMING THE SKED DEVICE HANDLER

This section describes the functional units of the device handler section of the SKED R.T.S., how the units are called and what operation each component must perform. In addition, there are:

a. Notes on assembling the SKED R.T.S.
b. Descriptions of several utility routines, included in the R.T.S. proper, that may be called in the service of device handling.
c. Documentation of a sample device handling system.

It is wise for the reader of this section to have a listing of the SKED R.T.S. readily at hand. Without it the reader will become quite confused.

Occasional symbols are used herein. They are:

AC	for	"ACCUMULATOR"
C(N)	for	"the content of the register tagged N"
C(AC)	for	"the content of the accumulator"

5.2.1 General Notes

The SKED R.T.S. device handler is a collection of subroutines and program

sequences that execute the component operations of interpreting time, stimulus, and response data and handling their flow between the digital interface hardware and the SKED R.T.S. proper.and are crucial to the operation of the R.T.S.
The basic functional units are:

a. The Clock Interrupt Service Routine
b. The Response Interrupt Service Routine
c. The subroutine SKEDIO
d. The subroutine RWRDST
e. The subroutine STIMO
f. The subroutine CLRBOX

In order to assemble a working Run-Time System, two source tapes are needed: the tape of R.T.S. proper and a source tape of the device handler wherein the above components are programmed in assembly language.(The R.T.S. listing shows an exemplary device handler)

In addition to the above components, two separate symbol definitions must be included in the device handler:

a. SKPCLK, which is the IOT instruction to "skip when the clock flag is set."

b. SKDND, which should be defined as "the last location +1" of the device handler. Since the device handler should follow the R.T.S. proper in core, SKDND will, effectively, be the lowest unused location of the memory field in which the R.T.S. resides.

In the following pages of this section, each of the components listed above will be described in terms of function and changes that they are allowed to make in the processor. At the beginning of each component description is the tag of the page zero location through which each component must be accessed and the symbolic address that is stored in that cell according to the symbolic source of the R.T.S. proper.

CLOCKIN, CLOCK

The SKED device handler must include a small clock interrupt service routine that restarts and resets the clock (if necessary), then exits to the time contingency scan control routine every time a clock interrupt occurs. The sequence must be tagged CLOCK and its exit must be an indirect jump to the location tagged CLOCKN.

Note that when this sequence is written, the IOT command SKPCLK should be defined. This will be a device dependent definition; the command that executes a skip when the system clock's flag is set.

NCLKIN, NOTCLK

The Response Interrupt Service Routine, which must be tagged NOTCLK, is entered whenever a program interrupt occurs. The sequence must determine:

a. Whether a response caused the interrupt

b. If the interrupt was caused by a response, which response caused it

When the interrupt has not been caused by a response, an indirect jump to the location tagged LOWPRI must be executed so that the low priority devices (Teletype, punch, reader, etc.) may be serviced.

When the interrupt was caused by one or more boxes the Response Interrupt Service Routine must set up an array of status registers showing which response caused the interrupt in a format that is usable by the subroutine RWRDST. A suggested format consists of one memory cell per experimental box into which is deposited twelve bits of status data detailing which response channels have been activated in each box.

Example:

Bit #	Ø	1	2	3	4	5	6	7	8	9	1Ø	11	
							..	.	..	R3	R2	R1	BOX 1
							.	..	.	R3	R2	R1	BOX 2
							.	.	..	R3	R2	R1	BOX 3

This is an example data format for a system with 3 boxes of experimental stations. Each of the three cells represents the conditions of the response channels for a corresponding box and the cells are all reset every time a response interrupt occurs.

In addition to this array, another status word must be arranged that shows with what boxes the interrupting responses are associated. This must be a 12-bit word stored in the location tagged SELWRD that has an ordered bit set for each box at which an interrupting response has occurred. The format must be as follows:

Bit #	Ø	1	2	3	4	5	6	7	8	9	1Ø	11
SELWRD BIT FORMAT	NOT USED	NOT USED	BOX 9	BOX 8	BOX 7	BOX 6	BOX 5	BOX 4	BOX 3	BOX 2	BOX 1	BOX Ø

should be zero

Any bit = 1
signifies that the corresponding box has one or more response channels activated

Note that during this process of interpreting interrupting response channels, all response flags and response flops must be cleared so that they cannot cause false interrupts later on. When the response interpretation is complete, the content of the located tagged Issim must be set to Ø designating that the responses being processed have not been simulated through the teletype (actually this can be done at any time during response interpretation)and an indirect JUMP to the location CLKBK1 must be executed with C (AC) = 1.

SPECIO, SKEDIO

SPECIO must point to a subroutine named SKEDIO that clears any special purpose flags, clears any response flags and response flops, initializes the 1ØØ cycle clock and starts same.

The subroutine makes no restrictions on the content of the ACCUMULATOR on entry , but should be exited with the content of the ACCUMULATOR equal to zero. It is wise to include a series of NOP commands in the subroutine so that flag clearing IOT commands can be easily added after assembly time.

RSETWD, RWRDST

The subroutine tagged RWRDST must analyze the conditions of the response channels assigned to the box whose number is C (BOXNO) and, using these data, set up a twelve bit status word, tagged RNUM, indicating which of twelve channels for that box are active. The bit positions of RNUM correspond to the twelve response events that trigger transitions in the state program. The format is as follows:

Bit #	Ø	1	2	3	4	5	6	7	8	9	1Ø	11
RNUM Bits	R12	R11	R1Ø	R9	R8	R7	R6	R5	R4	R3	R2	R1

When a bit is set, the corresponding response event will be able to trigger a transition in the state table. If CCissim=1, RWRDST should be exited making no changes to C(RNUM).

Note that NOTCLK RWRDST are the subroutines that transform real world response data into an expression of active response events, which the state program understands. Therefore, the manner in which any given response channel affects a state program is defined by these two subroutines.

On entry into RWRDST, RNUM may already contain a value resultant from a keyboard R simulation command. In such a case it will not be necessary and may even be detrimental to try to store other data in this cell. Therefore, RWRDST should do nothing to RNUM if there are data already contained therein.

On entry into RWRDST, the ACCUMULATOR will contain zero but the LINK will be set. The ACCUMULATOR must be cleared before RWRDST is exited.

STIMER, STIMO

The subroutine STIMO is called indirectly through the page zero cell tagged STIMER whenever the status of real-world stimulus flops must be changed.

At the time of call, C(BOXNO) equals the number of the box at which stimuli must be activated and the cell whose address is C(XTNSAV) acts as a control word detailing which specific stimuli must be activated. STIMO must interpret these data to determine what IOT's must be executed to accomplish this activation and to execute them. STIMO is the output counterpart of SELBOX and RWRDST in that it is the medium of transformation of the symbolic data in a format understandable to state programs to a format compatible with hardware in the real world. The format of the stimulus control word (whose address is C(XTNSAV)) is described in the following diagram.

Bit #	Ø	1	2	3	4	5	6	7	8	9	1Ø	11
Stimulus Control Word	Stim 12	Stim 11	Stim 1Ø	Stim 9	Stim 8	Stim 7	Stim 6	Stim 5	Stim 4	Stim 3	Stim 2	Stim 1

A "one" in any of these bit positions should be translated into an active stimulus at the box whose number is C(BOXNO).

Bit(s) are set by execution of a transition in which the expression "ON N" is included. N is one octal or decimal number or a list of same, separated by commas. Hence, execution of the transition described by

```
R1:ON,1,3,5,7 --- S2
```

would result in the stimulus control word being given the following setting:

0	0	0	0	0	1	0	1	0	1	0	1

On entry into STIMO, C(AC)=Ø and before exit the AC must be cleared.

BOXCLR, CLRBOX

The subroutine CLRBOX must clear all stimulus flops connected to the box whose number is C(BOXNUM). When these flops are cleared, the stimulus control word for that box must also be cleared. The address of the stimulus control word may be computed by adding three to C(BOXADD).

If the process of clearing flops is sufficiently complicated, the subroutine STMF may b- called with the following sequence. This will accomplish all that must be done in the most space economical way.

```
/OPERATIONS PREPARATORY TO CALLING STMF:

/C(BOXADD)+3 MUST BE STORED IN XTNSAV
/THE BOX # IS STORED IN BOXNO
JMS I STMOFF
-1       /TURN OFF ALL
         /STIMULUS CHANNELS
```

5.2.2 Example of a Device Handler

This section consists of explanatory comments on the subroutines in the R.T.S. device handler submitted in this software package.

The device handler will work with a system consisting of a KW-12A real-time clock and twelve bits of bi-directional digital control. The two twelve bit data buffers are built in a manner similar to that described in the circuit diagrams in the previous section.

The contents of the stimulus and response buffers are interpreted by the handler so that they represent four experimental stations, each having three stimulus channels and three response channels.

The first part of the device handler consists of the IOT definitions as seen below.

```
SKPCLK= 6131
CLLR=   6132
CLAB=   6133
CLEN=   6134
CLSA=   6135
CLRWRD= 6654
RDWRD=  6652
SKPWRD= 6651
SETSTM= 6662
```

CLLR, CLAB, CLEN, and CLSA are clock operation commands. SKPWRD is "SKIP if a response flip flop has been set." RDWRD will read the contents of the twelve bit response data register into the AC and CLRWRD clears the register and its SKIP flag. SETSTM will load the stimulus data register with the content of the AC.

The subroutine SKEDIO consists of a procedure that initializes and starts the KW-12A clock[5], the commands to clear the response data register and the software stimulus status register, and a string of NOP commands to allow for the addition of flag clearing commands as needed.

```
/CLOCK AND RESPONSE BUFFER INITIALIZATION.
SKEDIO, Ø
        CLA CMA
        CLAB                    /-1 LOADED INTO CLOCK.
        CLA                     /PRESET REGISTER.
        TAD     CONTRL
        CLLR                    /LOAD CONTROL WORD INTO
        CLA                     /CLOCK CONTROL BUFFER.
        TAD     CLNABL
        CLEN                    /LOAD CLOCK ENABLE REGISTER.
        CLSA
        CLA CLL                 /CLEAR CLOCK FLAG.
        CLRWRD
        DCA I   ALLXT           /CLEAR RESPONSE BUFFER.
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        JMP I   SKEDIO
ALLXT,  ALLXTN                  /EXIT
CONTRL, 51ØØ
CLNABL, Ø3ØØ
```

The clock interrupt service routine is very straightforward. The command CLAB clears and restarts the KW-12 clock. Then control transfers to CLOCKN.

```
/CLOCK INTERRUPT SERVICE ROUTINE.
CLOCK,  CLSA
        CLA CLL                 /CLEAR INTERRUPT FLAG.
        JMP I   .+1
        CLOCKN                  /JUMP TO CLOCK SCAN.
```

The response interrupt service routine is divided into two parts. First the routine determines whether any bit in the response data register (the input half of the digital interface - see section 5.1.1) has been set; and, if one has, it reads and clears the register, storing the result of the read in location RSTAT.

```
/NONCLOCK INTERRUPT SERVICE ROUTINE.
NOTCLK, SKPWRD                          /R INTERRUPT?
        JMP I   LOWPR                   /NO; CHECK LOW PRIORITY DEV.
        RDWRD   CLRWRD                  /READ AND CLEAR STATUS.
        DCA     RSTAT
        DCA     ISSIM
/ROUTINE TO SELECT BOXES TO BE
/SCANNED AND TO DISTRIBUTE R-STATUS DATA.
        CLA IAC                         /INITIALIZE BOX
        DCA     RNUM                    /SELECT BIT.
        TAD     RLIST
        DCA     INDEX1
        JMP     .+4
SELOOP, TAD     RNUM
        CLL RAL
        DCA     RNUM
        TAD     RSTAT                   /ANY R'S UNACCOUNTED FOR?
        SZA
        JMP     .+3
        IAC
        JMP I   SCAN                    /NO; EXIT.
        AND     BTMASK
        DCA     BTEM
        TAD     BTEM
        DCA I   INDEX1
        TAD     RSTAT
        CLL RAR
        CLL RAR
        CLL RAR
        DCA     RSTAT
        TAD     BTEM
        SNA CLA
        JMP     SELOOP
        TAD     RNUM
        TAD     SELWRD
        DCA     SELWRD
        JMP     SELOOP
BTMASK, ØØØ7
RSTAT,  Ø
LOWPR,  LOWPRI
SCAN,   CLKBK1
RLIST,  .
        Ø
        Ø
        Ø
        Ø
```

The second part of the procedure is the interpretation of the data read from the response data register. The register consist of twelve bits in the format described below.

Bit #	Ø	1	2	3	4	5	6	7	8	9	1Ø	11
Response Register Format	R3	R2	R1	R3	R2	R1	R3	R2	R1	R3	R2	R1
	Box #3			Box #2			Box #1			Box #Ø		

Any bit set to "1" indicates activity of the corresponding R-event.

The response event bits for each box are separated and stored in a four cell array such that the least significant three bits of each of the cells contains the R-status data for one of the four experimental stations. Concurrently, a bit is set in SELWRD for each box that has an active response channel. When the response channels have been thus interpreted, the routine exits to CLKBK1 with C(AC) = 1.

The subroutine RWRDST adds C(BOXNO)+1 to the address of the first cell of the four cell R-status data array that was compiled by the response interrupt service routine. This sum will be the address of the cell containing the R-status for the box numbered C(BOXNO). Note, below, that if C(issim)≠ Ø (CRNUM) is not changed.

```
/SUBROUTINE TO DEFINE RESPONSE BITS
/FOR BOX TO BE SCANNED NEXT.
RWRDST, Ø
        TAD     ISSIM  TAD   ISSIM
        SZA CLA
        JMP I   RWRDST
        IAC
        TAD     BOXNO
        TAD I   RLISTP
        DCA     XTNTEM
        TAD I   XTNTEM
        DCA     RNUM
        JMP I   RWRDST
RLISTP, RLIST
```

The flip flops in the stimulus data register are set by the subroutine STIMO, which uses a cell in memory, ALLXTN, to keep a running record of what flops are set at any given time. When STIMO is called, it retrieves the data describing which flops must be changed from the cell whose address is C(XTNSAV). Since the three status bits will be in the least significant bit positions in the cell, STIMO must reposition them so that they gain the significance described in the following chart.

Bit #	Ø	1	2	3	4	5	6	7	8	9	1Ø	11
Stimulus Bit Format	Stim 3	Stim 2	Stim 1	Stim 3	Stim 2	Stim 1	Stim 3	Stim 2	Stim 1	Stim 3	Stim 2	Stim 1
	Box 3			Box 2			Box 1			Box Ø		

When the revised stimulus status word has been assembled from C(C(XTNSAV)) and C(ALLXTN), it is re-stored in ALLXTN and then transferred by the SETSTM command from the AC to the stimulus data register.

```
/SUBROUTINE TO SET STIMULUS FLOPS
/IN THE BOX WHOSE NUMBER IS
/C(BOXNO).
STIMO,   Ø
         CLA CLL
         TAD        BTMASK
         DCA        ATEM
         TAD        BOXNO
         CMA
         DCA        BTEM
         TAD I      XTNSAV
         AND        BTMASK
         DCA        CTEM
STMRTL,  ISZ        BTEM
         SKP
         JMP        STLR
         TAD        ATEM
         RTL
         RAL
         DCA        ATEM
         TAD        CTEM
         RTL
         RAL
         DCA        CTEM
         JMP        STMRTL
STLR,    TAD        ATEM
         CMA
         AND        ALLXTN
         TAD        CTEM
         DCA        ALLXTN
         TAD        ALLXTN
         SETSTM
         CLA CLL
         JMP I      STIMO
ALLXTN,  Ø
ATEM,    Ø
BTEM,    Ø
CTEM,    Ø
```

CLRBOX is entered with C(BOXNUM) equal to the number of the box whose stimulus channels are to be cleared. CLRBOX transfers this datum to BOXNO, sets C(XTNSAV) to C(BOXADD)+3, the address of the stimulus data cell associated with the box whose number is C(BOXNUM). The subroutine STMF is called, which uses C(BOXNO) and C(XTNSAV) to do the actual work involved in clearing stimulus register flops.

```
/SUBROUTINE TO EXECUTE GENERAL CLEAR OF THE
/STIMULUS FLOPS OF ONE EXPERIMENTAL STATION.
CLRBOX, Ø
        IOF
        TAD     BOXNUM  /DEFINE NUMBER OF BOX TO CLEAR.
        DCA     BOXNO
        TAD     BOXADD
        TAD     C3
        DCA     XTNSAV  /SELECT STIM. STATUS WORD.
        JMS I   STMOFF  /TURN OFF ALL CHANNELS
        -1              /FOR THIS BOX.
        JMP I   CLRBOX
XTNTEM, Ø
BOXTEM, Ø
        SKDND=.
        $
```

5.2.3 The UDC-8 and the UDC System Builder

The UDC System Builder is a PDP-8 program that will design a Run-Time System to control variety of laboratory configurations using a PDP-8 and a specific digital interface. The program elicits information regarding the number and arrangements of experiment stations then punches a binary tape of the required R.T.S. This tape can be used thereafter to control step programs. The hardware required to use this system is:

a) PDP-8/e or PDP-12
b) real-time programmable clock: DK8-ep for PDP-8/e or KW-12A for PDP-12.
c) DEC's UDC-8 universal and digital controller

This section consists of:

a) notes on the structure of the UDC-8 as needed by the user of the UDC-8 System Builder
b) a step-by-step description of the operation of of the UDC-8 System Builder
c) an example run of the System Builder dialog.

Each note contains a deeper explanation of some point of confusion in the dialog and is referenced in the description of the dialog at the point to which it pertains. The wise user will review the questions, notes and sample dialog before attempting to use the System Builder

5.2.3.1 The UDC-8 Description and Explanatory Notes

The UDC-8 is a highly flexible digital input/output option for process control applications. It can effectively monitor and control the stimulus/response signals for SKED software in the manner described in the previous chapters.

When the UDC-8 is being used as the basic digital interface to the laboratory, connection of lab devices to the computer interface system will be a simple matter of connecting wires from the lab to appropriate screw terminals. The specific data needed for this task will be available in the UDC-8 Maintenance Manual (DEC-08-HZDA-D).

Output control is especially easy because the UDC provides the user with relay cards that attach directly to the UDC minimizing special considerations for output signal conditioning.

Input will be a little bit more complicated. The user must make sure that all signals going into the UDC from the lab are in the form of pulses with a duration of 6-9 miliseconds.

The main unit of the UDC-8 is a series of "card slots" that can receive or send twelve bits of digital data with equal facility. Into each card slot a control module will be plugged that will define which direction data will be going through the slot. If the user wishes that a given slot should be an input register, that slot will have an input control module plugged into it. The input control module will be connected to twelve devices in the laboratory that are capable of generating digital pulses. If a card slot is to be an output register, an output control module is inserted.

Since there are two kinds of input control registers that can be used with the UDC-8, the user must be careful to obtain the correct type. The choice is between "contact interrupt signal conditioners" and "contact sense input conditioners." The R.T.S. that will be produced by the System Builder will only work with "contact sense" cards. Similarly, the user has a choice of several types of output control cards.

Note 1

Each card slot or data register, has an octal numeric address by which it is referenced by the computer. The addresses are arranged ordinarily, starting with zero, and correspond to the physical position of the slot from the UDC-8 control. (see UDC Maintenance Manual for a map of the UDC card slots).

When the experimentor is setting up the lab, he must choose which slots will be used for input and which for output. At this time he will decide that wires from certain experiments stations will go to certain control slots. Note that each slot of twelve bits must be all input or all output. A slot cannot be functionally divided.

During the dialog the user will be asked which slots are input slots and which are output. By referring to the map of slots or counting I/O slots from the controller, the slot number can be determined. This information is used to answer the question:

For Box N
Give Register Used

Example:

Let's say that a user has a UDC-8, which has twelve data registers (or slots). With this, the user wishes to control six experiment stations each of which will have twelve stimulus signals and twelve responses that must be monitored. The user has connected his response channels so that responses from box (or station) Ø are sensed through data register Ø, response channels from BOX 1 are wired to register 1, etc.

When the dialog asks for the arrangement of a response channels it will print:

For BOX Ø
GIVE DATA REGISTER

The correct response from the user will be to type "Ø" followed by a RETURN

For BOX 1
GIVE DATA REGISTER

The user should type "1" followed by a RETURN; etc. Note that response registers do not have to be directly beside response registers. It is perfectly acceptable to have register Ø handle input, register 1 handle output, register 2 handle input, etc.; alternating input and output registers.

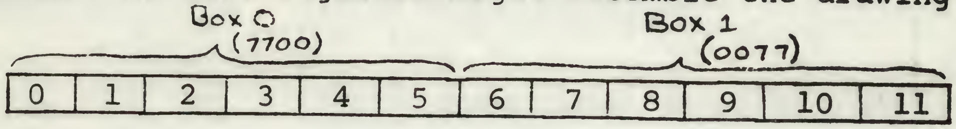
Note 2

The twelve bits within each UDC data register may be divided among several experiment stations. For example, if both station Ø and station 1 will have only six response channels emanating from them, the 12 bits of any data register can be divided between them. The user can indicate to the system builder that such a division has occurred. Each time the dialog asks what register will be used to monitor a given box, it procedes to inquire:

"Give Data Mask"

which means that the user should signify with a four digit octal integer what bits in the given register will be used to monitor the given experiment station. The data mask is formulated by representing the twelve bit positions of the register in question with respect to whether or not each will be used to scan the box in question; then translating the binary number into four octal digits in the same way that the ACCUMULATOR bits can be interpreted as octal digits. If a register bit is to be used, this will be represented by a 1 in the binary position corresponding to that bit. Unused bits are represented by zeros in the corresponding positions. For example, in a situation where all of the bits in a register will be used to monitor one box, the data Mask will be 7777, which is the octal representation of the binary 111 111 111 111. This binary number represents the bits that will be used in the given register.

A further example: Suppose an experimenter has a lab with two experiment stations each having 6 response bits. To monitor both boxes he needs only one input register. This means that the twelve bits will be divided into two groups of six. The wiring arrangement for this register might resemble the drawing below



The bit usage scheme for BOX Ø then, would be represented in binary as 111 111 000 000 or as 7700 in octal. Similarly the binary bit usage representation for BOX L will be 000 000 111 111 and its octal will be 0077.

For BOX Ø
GIVE REGISTER USED: Ø
GIVE DATA MASK: 7700

FOR BOX 1
GIVE REGISTER USED: Ø
TYPE DATA MASK: ØØ77

Note 3

The answering the questions pertaining to stimulus channels are the same as described above in NOTE 2; simply follow the same method and all will be well.

5.2.3.2 Loading, Starting and Running the UDC System Builder

The UDC System Builder is a standard PDP-8 binary program with starting address of 3ØØØ . To run it, the user loads it with the DEC BIN Loader, and starts at 3ØØØ with the teletype on. The program will lead the user through a dialog, set up RTS parameters, and at the end of the print the final dialog core address of the R.T.S. that has been set up. The user can then decide, if the RTS is suitable, whether to punch the R.T.S. on paper tape or to immediately enter the RTS and try it out to see if it works properly.

At some points in the dialog the user may wish to abort and restart the dialog or restart the question section that is currently running. An abort can be executed any time by the typing of the up arrow character (Shift N). When the user wishes to go back to the beginning of the question section currently being worked on he can type the ALTMODE key. All the data entered before that question section will be preserved.

The UDC System Builder Dialog

At the start of the dialog the computer prints:
SKED SYSTEM BUILDER FOR SYSTEM USING UDC
AND PDP-8/e WITH DK8-EP CLOCK OR PDP-12 WITH
KW-12-A CLOCK
The dialog then proceeds as follows:

Question 1: Miscellaneous "clear commands". This is the users opportunity to inform the System Builder of any devices that are attached to the interrupt facility of the computer and have flags that must be cleared during the intializing of the system. The Builder will print:

```
TYPE MISC FLAG CLEAR COMMANDS
```

Now the user can list the octal codes of commands to clear any I/O flags that can initiate program interrupts (not including the flags involved with the UDC and KW12-A or DK8/e-P clock, since these flags are already accounted for in the System Builder). After the colon type the four digit octal code, followed by a CARRIAGE RETURN (generated by pressing the RETURN key on the teletype). If a non-octal character is typed, the program will type a "back-slash" character (\) designating that the input has been rejected and return with another":". After each command has been successfully entered the program will print a new colon on a new line and await further input.

When there are no more commands to be listed the user can continue to the next question by typing a RETURN in reply to the computer's printed colon.

Question 2: The number of Stations

The computer will print:

```
HOW MANY STATIONS?:
```

to which the user must reply with a decimal number in the range 1-10 designating the number of experiment stations to be controlled by the SKED System. The numberic input must be followed by a typed RETURN.

Question 3: The Input Channels For Each Station

This is a two part question through which the user declares the relationships between each experiment station and the input channels that carry response data from the station to the computer.

The computer starts the question by printing

```
FOR EACH STATION TYPE THE NUMBER OF THE
UDC CARD THAT WILL RECEIVE DATA AND AN
OCTAL NUMBER REPRESENTING THE PERTINENT BITS
```

Then the computer prints

```
FOR BOX N
GIVE REGISTER USED:
```

N represents a number in the range 0-9; that is the number of the station for which data channels are being declared. The appropriate reply will be a decimal number, indicating which UDC data register will receive data from experiment station N (see 5.2.3.1 Note 1), followed by a RETURN. Next, the computer will print

```
TYPE DATA MASK:
```

in reply to which the user should type a four digit octal number expressing which bits of the given register will pertain to the given station. (see 5.2.3.1, Note 2) The octal number must be followed by a RETURN.

This pair of questions will be repeated for each station insofar as the number of stations in the system has been defined in question 2. If, for example, the user declares that there will be four experiment stations in the system being built, then the two parts of question 3 will be repeated four times.

Question 4: The Output Channels for Each Station

Question 4 is also a two part question, having the same function for the output channels of the UDC that question 3 fulfilled for the inputs. The computer prints:

```
FOR EACH STATION TYPE THE NUMBER OF
THE UDC CARD THAT WILL CONTROL DATA OUTPUT AND
AN OCTAL NUMBER REPRESENTING THE PERTINENT BITS.
```

Again, for each station the computer prints

```
FOR BOX N
GIVE REGISTER USED:
```

and, after the user has supplied the appropriate data in the form of a decimal number (see note 3) followed by a RETURN, the computer prints

```
TYPE DATA MASK:
```

Again, the proper reply is a four digit octal number followed by a RETURN. This question pair will be repeated as many times as are necessary to supply output control data for each station to be controlled.

Question 5: Punch? or Test?

At this point all data concerning what the system will look like has been supplied to the Builder by the user. The Builder will now indicate how much core will be used by the R.T.S. being built by printing:

```
FA=XXXX
```

where XXX is a four digit octal address, the end of the memory area occupied by the R.T.S. The user may find this number helpful in determining whether the system just described is realistic or if something has gone wrong during the dialog. The computer will continue, printing

```
FINISH OPTION:
```

To this the user must reply by typing "1", "2", or "3" followed by a carriage return according to which of the three following possibilities is to occur.

if "1"	The newly built R.T.S. will be entered immediately
if "2"	The program will enter the proceedure to punch the newly built R.T.S. (see below)
if "3"	The dialog will rerun from the beginning.

The R.T.S. Punch Proceedure:

If the reply to "FINISH OPTION" was "2" the computer will HALT. This will allow the user to indicate what device is to be used to punch the output tape.

If high speed punch is to be used set switch Register Bit 11 to the ON or ONE position. If the teletype punch is to be used set bit II to the off position. After the appropriate punch has been selected, press the CONTINUE switch on the computer to execute the punch operation.

After the R.T.S. has been punched the computer will again HALT, allowing the user time to turn off the punch and remove the tape. This new binary tape can, hereafter, be used to run SKED programs. Its operation is described in section 4.

When the user presses CONTINUE the computer will again print "FINISH OPTION". The options open to the user are the same as were described above.

5.3 Notes on Programming Special Purpose Functions

When the SKED compiler encounters an "F3" function, it compiles a JMS instruction followed by a series of arguments. The sequence has the following format:

```
JMS I F3GO
T
ENTRY
ARGUMENT
ARGUMENT
etc.
```

T will be a value, either 2, 1 or Ø, that indicates whether any of the following arguments are time expressions, output identifiers or event counters.

ENTRY is the starting address of the special purpose subroutine being entered through the F3 function.

ARGUMENTs are one or two cell values that the special purpose subroutine will use as data. If there are any time expressions in the F3 function, all arguments will be two cell values and the datum symbolized by T will be ØØØ1. With no time expressions in the F3 function, T will be ØØØØ or ØØØ2 and the ARGUMENTs will each occupy one cell. T = ØØØ2 indicates that one of the ARGUMENTs is a variable's symbol, where the variable represents "N" in the output expressions "ZN", "CN", "ON N". Two examples follow:

Example 1

The function F3 (5,1,2,01Ø) will be compiled

```
JMS I F3
ØØØØ           Note that these numbers are all
ØØØ5
ØØØ1           representations of twelve bit binary
ØØØ2
ØØ1Ø           values.
```

Example 2

The function "F3(21,1,1")" will be compiled

```
JMS I F3
Ø
ØØ25 -----  This is the octal equivalent of 21,
ØØØØ -----  a number that should be expressed
ØØØ1        in 12 bits.
Ø144 -----  low order time value
ØØØØ -----  high order time value
```

The instruction JMS I F3GO calls a subroutine in the SKED R.T.S. that, in turn, calls the user's subroutine whose starting address is ENTRY with the C(AC) = address of the first argument. Thus, when the user's routine is entered, the only information that is provided indicates where its arguments are located. The routine must be programmed so that it "knows" what type of argument and how many arguments to expect. When the user's subroutine has finished its work, the state program can be re-entered if C(AC) = address of the first cell after the functions argument list and a "JMS I ENTRY" is executed.

Example:

Imagine that, as reinforcements in a certain experiment, the researcher wishes to display on a CRT scope the words "YES" and "NO." Since these reinforcements will not be operable with digital control reinforcement hardware as described in section 5.1, it will be necessary to write a special routine to handle the task. The researcher would like to call the routine with simple expressions like F3(N,1) for "display 'YES'," F3(N2) for "display 'NO'," and F3(N,Ø) for "clear the scope." The state program using these might appear as follows:

```
S.S.1
S1,
            R2 ----> S2
S2,
            R1:F3(FA+1,1,3) ----> S3
S3,
            2":F3(FA+1,3) ----> S2

S.S.2
S1,
            60' ----> STOP
```

This process will start when R2 occurs and thereafter, for a period of one hour, will display the words "YES" or "NO" for two seconds according to the responses during the activity of S2.

The display routine will begin at location FA+1 where FA is the final address of the compiled state tables. The first argument will define what display function will be executed, and the second will define the size of the characters.

The display service routine would appear something like the following:

```
/DISPLAY SERVICE ROUTINE ENTERED HERE, . = FA+1
    ENTRY, DCA     ARGPTR       /STORE THE ADDRESS OF THE ARGUMENT.
           TAD  I  ARGPTR       /RETRIEVE THE DISPLAY
           DCA     FUNC         /FUNCTION INDICATOR.
           ISZ     ARGPTR       /ADVANCE THE ARGUMENT POINTER.
```

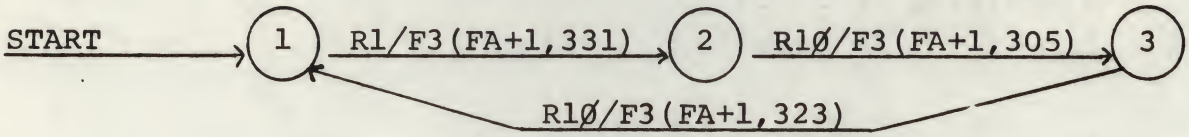
```
        TAD  I  ARGPTR       /RETRIEVE THE CHARACTER
        DCA     CSIZE        /SIZE INDICATOR AND STORE IT.
        JMS     DISPLAY      /EXECUTE THE DISPLAY FUNCTION.
        TAD     ARGPTR       /WITH C(AC) = THE ADDRESS OF
        IAC                  /NEXT INSTRUCTION IN THE
        JMP  I  ENTRY        /STATE TABLE, EXIT BACK TO THE
                             /F3 SERVICE ROUTINE.
```

Note that the function of the last three instructions could just as easily have been accomplished by

```
        ISZ     ARGPTR
        JMP  I  ARGPTR
```

which exits directly to the state table without going through the end of the F3 routine.

It is important to remember that the R.T.S. is operated in the interrupt mode and that the interrupt must not be kept off longer than half the time between clock pulses (i.e., 5 msec). This restriction may require a sequential string of states to handle a slow peripheral device. Furthermore, in these circumstances it may be necessary to rewrite the input section of SKED so that the interrupt generated by the compilation signal of the peripheral is defined as an R event. In the previous example each character of the YES stimulus requires time for accurate registry by the CRT. A state graph of the form



expresses a procedure to display "YES" in three discrete steps. The process is started by the transition from S1 to S2, triggered by R1. The F3 function causes intensifying of one letter (in this case, the second argument of the function is the ANSCII code for the letter to be displayed) and re-enters the state program before the display function is actually finished. When the display is ready for another character, it will set a flag that is interpreted as R1Ø, which can trigger transitions allowing more characters to be displayed.

It will usually be desirable to have any one special function executed by one subroutine that can be called by all state programs running under R.T.S. control. In this case it will also be desirable to make the R.T.S. Loader protect the subroutine from being overlayed by state tables. To do this, the programmer must

a. Program special purpose subroutines so that they follow the R.T.S. in core with very little space between them.

b. Redefine the symbol SKDND, as it was in the device handler, so that it follows the last special subroutine and represents the first unused cell in the core that will contain state programs.

c. Redefine the cell tagged SKEDND as to contain the new value of SKDND. The core location of SKEDND may be ascertained by reference to the SKED R.T.S. listing. These re-definitions might appear as follows:

```
*1717                                /THIS ORIGIN INDICATES THE
SKEDND,    -SKDND                    /CORRECT ADDRESS OF SKEDNE
                                     /AS OF 2/1/71
ENTR,      0/                        /FIRST CELL OF SPECIAL ROUTINE
           JMP I ENTR                /LAST COMMAND OF SPECIAL ROUTINE
           SKDND =                   /SKDND DEFINED AS FIRST CELL
                                     /IN FREE CORE AREA
```

5.4 Large Scale Changes to the System

Some changes to SKED that might be desirable in many cases and that will require more work than just re-writing the device handler are:

1. Change of time base. The SKED compiler assumes that time will be counted in intervals of 1Ø milliseconds. If a clock with different frequency is used, it will probably be necessary to modify the compiler. Fortunately, this modification will probably include changing only the routine TCON, which is responsible for the evaluation of all time expressions.

   TCON's operation includes reducing the time expression to a two cell two's complement number (SEC is low order, MIN is high order), the number of 1Ø millisecond intervals expressed by the time expression. Modification may simply be a task of rewriting the part of the subroutine that computes how many intervals are indicated. If this is so, then no changes will be needed in the R.T.S.

2. Extension of core. With 8K or more of core memory several extensions can be made to the SKED system.

   a). The SKED R.T.S. could, conceivably be modified to use two fields of memory by loading some state tables into the lower field and some into the upper field. This could be accomplished by changing the R.T.S. loader so that state tables for boxes Ø-3 load into the lower field and those for boxes 4-9 load into field 1. In addition, the device handler would need to be a bit more complicated.

b) By modifying the device handler scheme for both the R.T.S. and the compiler, it may be possible to run them simultaneously, each being loaded into separate core memory field.

   Further, it may be desirable to enable the compiler to compile directly into core, thus eliminating the need for the R.T.S. loader and various tape operations.

c) With extra core, it may be possible to add functions to the R.T.S. that will enable the user to change the parameters used to direct the R.T.S. operations.

REFERENCES

1. For a more thorough explanation of the theory of states, see:

   a. Snapper, A. G., Knapp, J. and Kushner, H. "Mathematical Description of schedule of Reinforcement." In W. N. Schoenfeld and J. Farmer (Eds.), Theory of Reinforcement Schedules. New York: Appleton-Century-Croft, 1970.

   b. Snapper, A.G. and Kadden, R.M. "Time Sharing in a Small Computer Based on a Behavioral Notation System." In B. Weiss (Ed.), Digital Computers in the Behavioral Laboratory. New York: Appleton-Century-Crofts, in press.

2. Digital Equipment Corporation, 146 Main Street, Maynard, Mass. 01754. The Small Computer Handbook. See the 1970 edition for information on the PDP-8/I and PDP-8/L. The 1971 edition contains information about the PDP-8/e.

3. The Logic Handbook, 1970 edition.

4. The Control Handbook, 1970, 1971.

5. The Laboratory Handbook, 1971. A detailed description of the programming of the KW-12A real-time clock is contained herein.

6. Ferster, C. B. and Skinner, B. F., Schedules of Reinforcement. New York: Appleton-Century-Crofts, 1957.

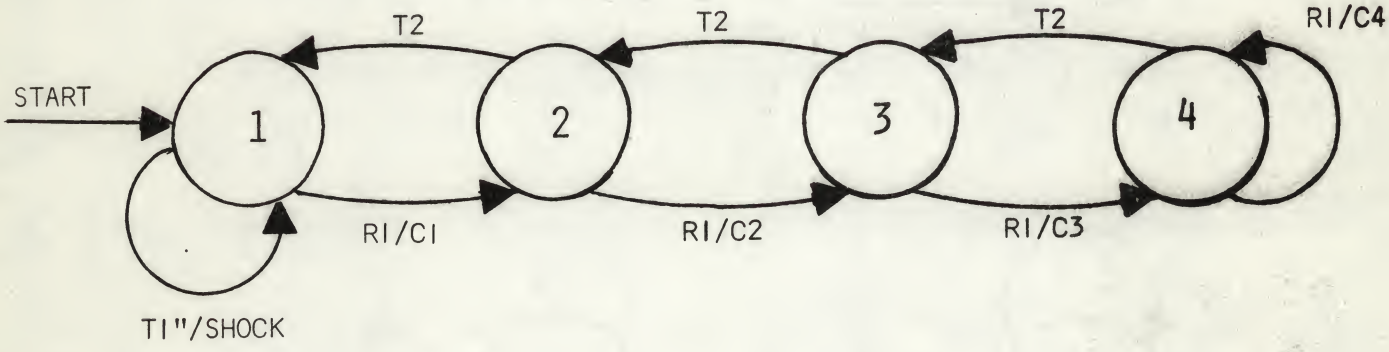
7. Farmer, J."Properties of Behavior Under Random Internal Reinforcement schedules." Journal of the Experimental Analysis of Behavior (JEAB), 1963 Vol. 6 page 607-615

8. Millenson, J. R., Random Interval Schedules of Reinforcement. Journal of the Experimental Analysis of Behavior, 1963, Vol. 6, page 437-443.

Appendix B: Sample Schedules of Reinforcement.

The following set of schedules and associated data recording has been chosen to illustrate the power of the state operating functions FI, F2, z pulses, and gating. The examples all have been thoroughly tested on the compiler, D-Bug and SKED R.T.S. programs and can be used to test the entire system.

Example I: The first example is an adjusting avoidance schedule developed by Field and Boren (1963). In this schedule shocks are programmed every TI seconds if no responses are made by the subject. Each response, when emitted, adds T2 seconds to the delay before the next programmed shock up to a specified maximum. A schematic state diagram with recording counters follows:

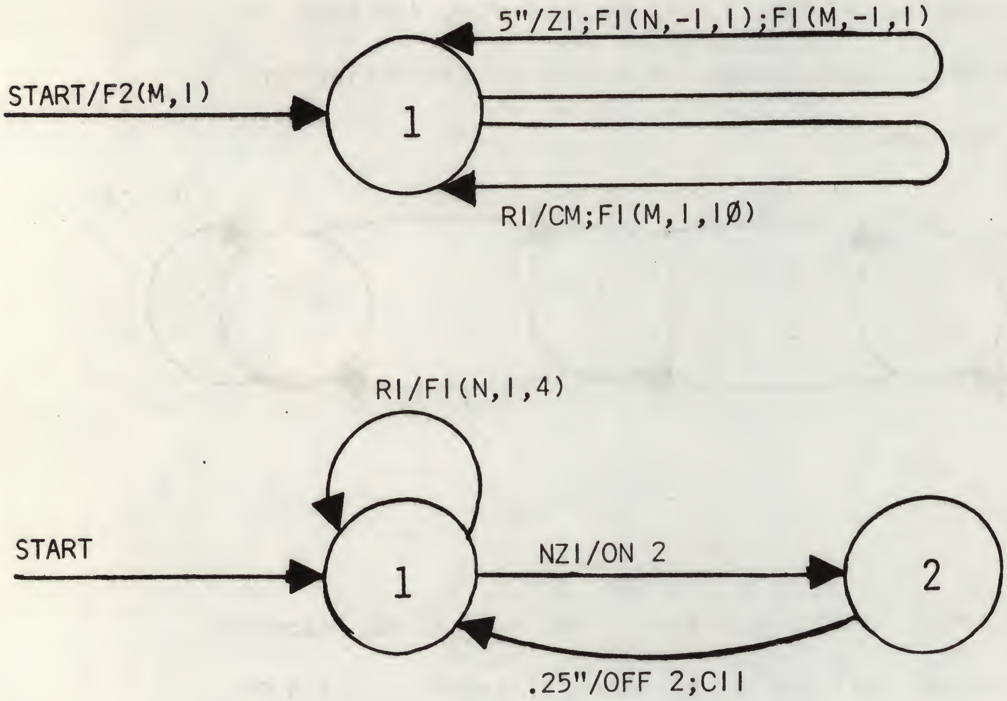


In this diagram the experiment starts in State I. As long as no responses occur shocks (in another state set) are given every TI seconds. If a RI is emitted, State 2 is entered. T2 seconds later State I is re-entered if no further responses occur. After TI seconds the shock train again begins. If a RI occurs in State 2, State 3 is entered adding an additional

T2 seconds to the remaining time to the next shock. Responses in the final state reset its T2 timer adding between 0 and 5 seconds to the interval before the next shock and providing a maximal delay of 3T2 + TI seconds.

Each response 's recorded in a separate counter yielding information about the prevailing delay when the response is emitted.

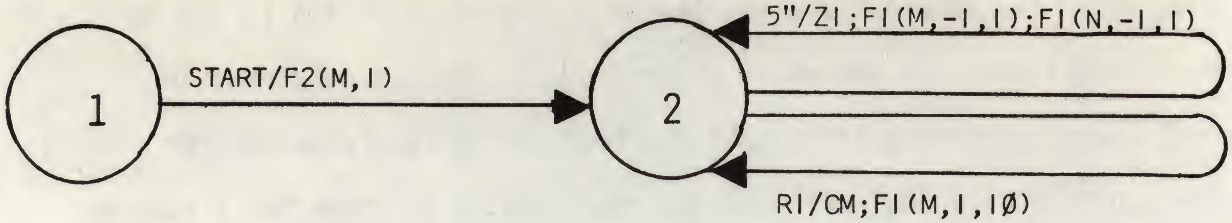
This program can be shortened (if TI=T2) by use of the FI and F2 functions, as is shown in the following diagram:

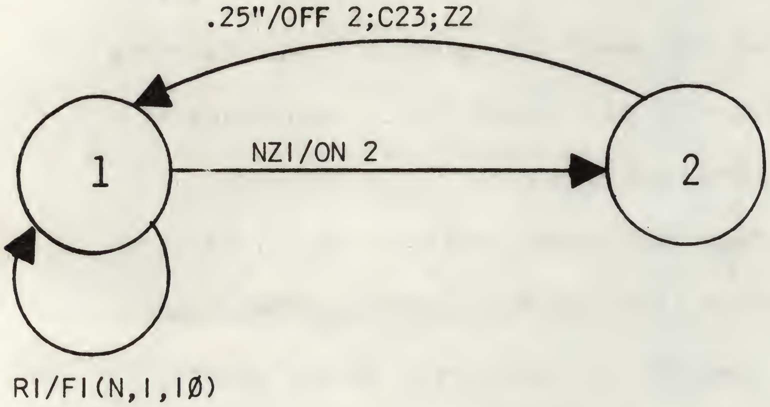


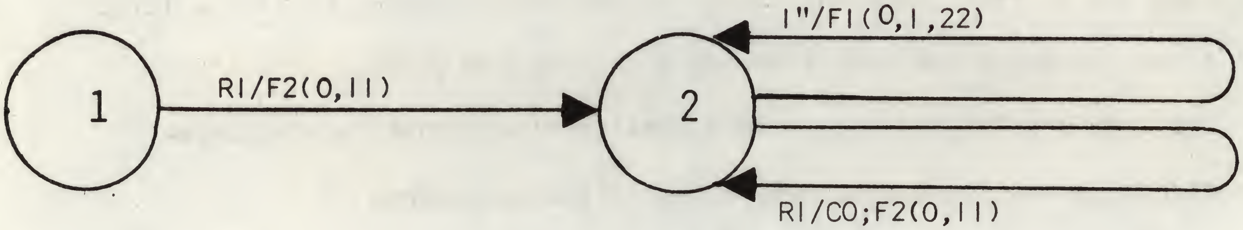
On entry into State I of State Set I the counter address M is set to I to initialize it (the Compiler sets CM to counter zero initially). A zI is generated if no response has occurred in the preceding 5" period. The zI is counted in the second state set (event counter N is initially set to I by the compiler). If the count is complete a shock is generated on channel 2 and terminated after .25" and recorded in recording-counter II. If the count of zI's does not equal N, the 5" pulse in State Set I reduces the value of N. If State I of State Set 2 is entered by a transition, the new value of N will be established as the next interval. Responses will record in recording counter M, next increment the value of M and reset the 5" timer of State Set I. Responses will also cause the current value of N to be the new count of zI's required for shock.

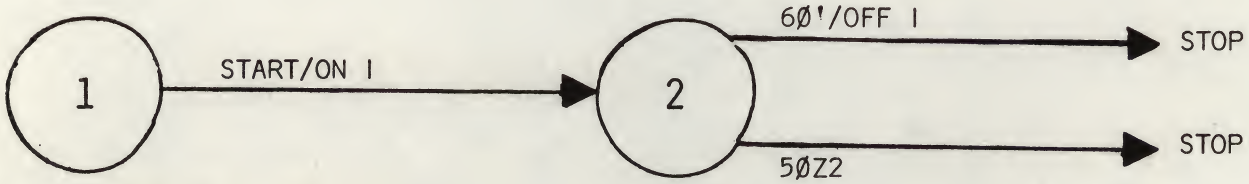
This program depends on the faet that event counters consist of both a counter constant (N) and a location in which the actual counting takes place. The value of the counter constant is transferred to the count location on state entry. Thus, N can be varied for the next count without modifying the current count when a zI is generated. The RI re-initializing will first increment and then transfer N for the new count.

The complete diagram including a session timer and inter-response-time recording histogram is shown in the following diagram:


5"/Z1;F1(M,-1,1);F1(N,-1,1)
START/F2(M,1)
1
2
R1/CM;F1(M,1,1Ø)


.25"/OFF 2;C23;Z2
NZ1/ON 2
1
2
R1/F1(N,1,1Ø)


1"/F1(O,1,22)
R1/F2(O,11)
1
2
R1/CO;F2(O,11)


6Ø'/OFF 1
STOP
START/ON 1
1
2
STOP
5ØZ2

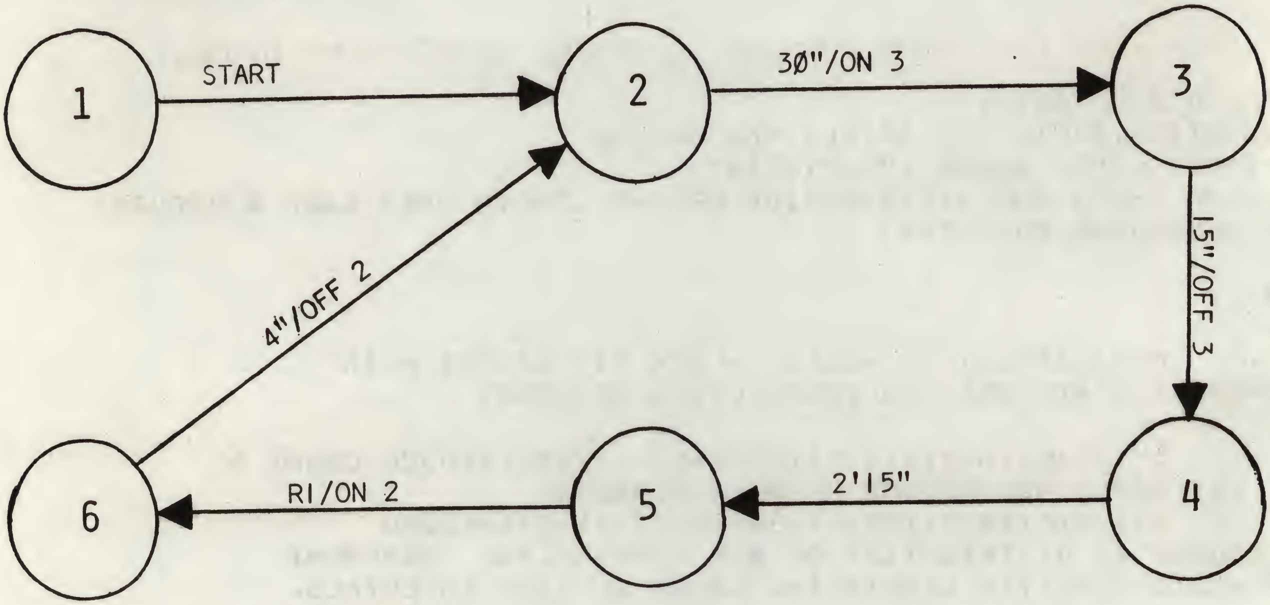
Note that this diagram has added an IRT distributor in State Set 3. The first response of the session (RI) is not recorded but initializes the counter 0 to II, the first counter of the distribution (II to 22). In State 2 a RI records in counter 0 and then sets 0 back to II for the next recording.

State Set 4 controls a session cue in channel I. Two methods of terminating the session are programmed. Stop occurs either at the end of 60' or after 50 shocks (each shock generates z2) whichever comes first. The state table produced by means of the DEC Editor follows:

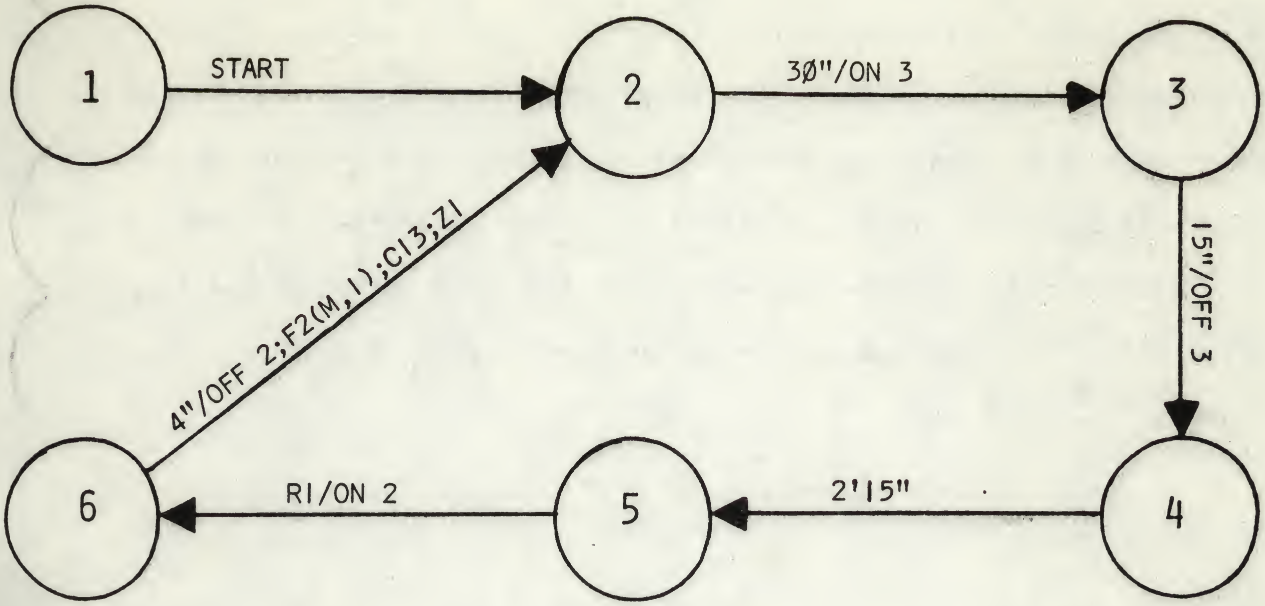
```
/APPENDIX B TEST 1
/ADJUSTING AVOIDANCE (FIELD AND BOREN)
/S-S=5";EACH R EARNS 5" UPTO 5Ø"
/RECORD IRT'S AND DISTRIBUTION OF R-S LENGTH WHEN EACH R OCCURS
/23 RECORDING-COUNTERS.

S.S.1,
S1,
        R12:F2(M,1)---->S2/R12 FROM TTY STARTS MAIN
/SCHEDULE STATE SET AND INITIALIZES COUNTERS
S2,
        5":Z1;F1(N,-1,1);F1(M,-1,1)---->S2/REDUCE COUNT OF
/5" INTERVALS AND REDUCE COUNTER ADDRESS.
        R1:CM;F1(M,1,1Ø)--->S2/RESET TIMER,RECORD
/RESPONSE IN DISTRIBUTION OF R-S LENGTHS,AND INCREMENT
/BOTH DISTRIBUTION LENGTH AND COUNT OF TIME INTERVALS.

S.S.2,
S1,
        NZ1:ON2----->S2/TURN ON SHOCK AFTER VARIABLE INTERVAL
        R1:F1(N,1,1Ø)-------->S1/ESTABLISH NEW COUNT
S2,
        .25":OFF2;C23;Z2--->S1/END BRIEF SHOCK,AND RECORD SHOCK.

S.S.3,
S1,
        R1:F2(O,11)--->S2/START IRT DIST. ON FIRST
/RESPONSE AND INITIALIZE DISTRIBUTION COUNTER.
S2,
        1":F1(Ø,1,22)--->S2/INCREMENT COUNTER
        R1:CØ;F2(Ø,11)---->S2/RECORD AND RESET COUNTER TO FIRST.

S.S.4,
S1,
        R12:ON1---->S2/TURN ON SESSION LIGHT.
S2,
       6Ø':OFF1--->STOP
        5ØZ2------->STOP/STOP AFTER 5Ø SHOCKS OR ONE HOUR.
$
```

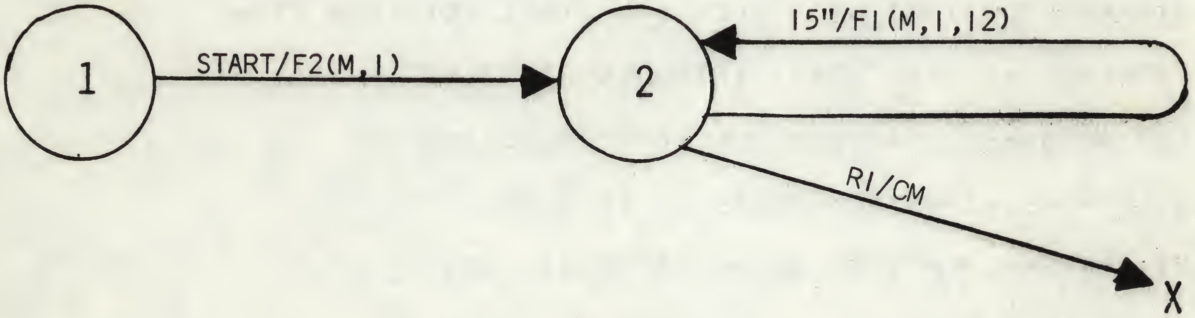
This experiment took 341 octal memory locations, including space for 23 recording counters.

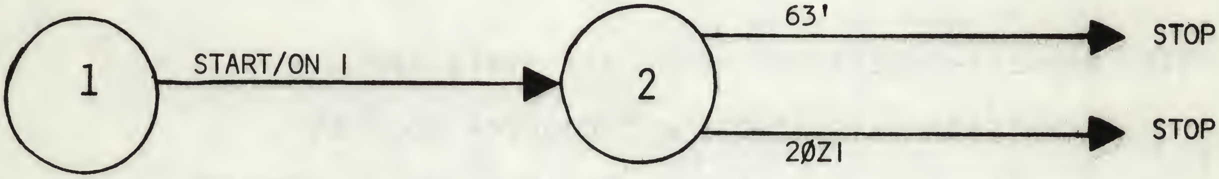
Example 2: This program is of a Fixed Interval schedule of positive reinforcement with an intruded (originally neutral) stimulus (Farmer and Schoenfeld, 1966). Recording includes responses in successive subintervals of the FI averaged over the session. The basic state diagram is straight-forward:



In this graph 30" after starting a stimulus or channel 3 is intruded. After 15" the stimulus is turned off. Finally after 3' from session start, the interval is completed and State 5 entered. Here, the first response (RI) starts reinforcement, after which the cycle is repeated. To add recording a separate state set may be used:


1
START
2
3Ø"/ON 3
3
15"/OFF 3
4
2'15"
5
R1/ON 2
6
4"/OFF 2;F2(M,1);C13;Z1


1
START/F2(M,1)
2
15"/F1(M,1,12)
R1/CM
X


1
START/ON 1
2
63'
STOP
2ØZ1
STOP

The second state set controls recording. When session start occurs counter M is set to 1. Every 15" thereafter it is incremented. (It is reset to 1 by the end of reinforcement in State Set 1.) Responses in State Set 2 are recorded in counter M and do not reset the 15" timer.

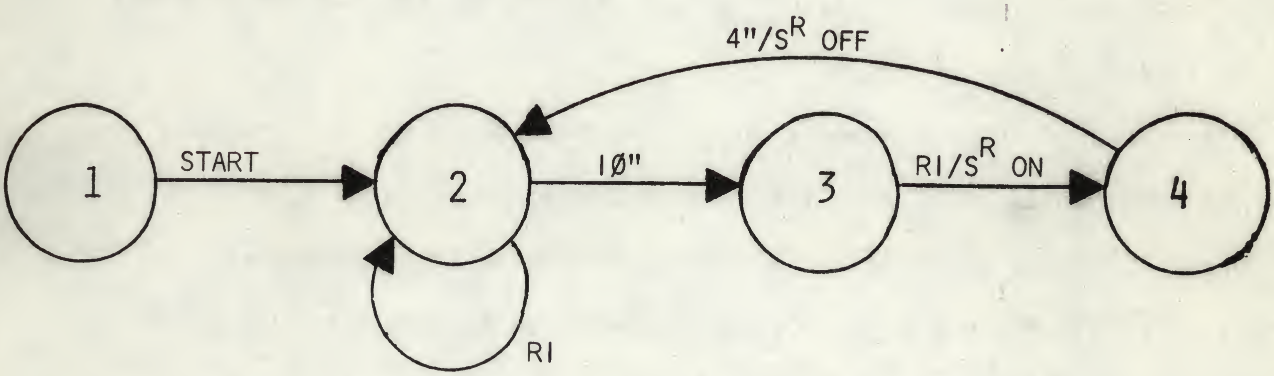
State Set 3 turns on a session cue on start and turns off the session after 63' or 20 reinforcements whichever comes first. The state table follows:

```
/APPENDIX B TEST 2
/FI 3' BY THE RESPONSE WITH A 15" STIMULUS INTRUDED AFTER 30"
/IN THE INTERVAL. RECORD RESPONSES IN SUCCESSIVE 15" SUB-
/INTERVALS.
/13 RECORDING-COUNTERS.

S.S.1,
S1,
        R12---->S2/START MAIN INTERVAL STATE SET FROM TTY.
S2,
        30":ON3------>S3/START INTRUDED STIMULUS.
S3,
        15":OFF3----->S4/TURN OFF INTRUDED STIMULUS.
S4,
        2'15"-------->S5/COMPLETE 3' INTERVAL.
S5,
        R1:ON2---->S6/FIRST RESPONSE STARTS SR.
S6,
        4":OFF2;C13;Z1;F2(M,1)--->S2/END,RECORD AND
/RESET TO FIRST COUNTER OF SUB-INTERVAL DISTRIBUTION.

S.S.2,
S1,
        R12:F2(M,1)--->S2/START RECORDING STATE SET
S2,
        15":F1(M,1,12)--->S2/INCREMENT COUNTER ADDRESS
/EACH 15".
        R1:CM---->SX/RECORD RESPONSES IN APPROPRIATE COUNTER.
/DO NOT RESET TIMER.

S.S.3,
S1,
        R12:ON1---->S2/START SESSION LIGHT.
S2,
        63'--->STOP
        20Z3---->STOP/STOP AFTER 63' OR 20 SRS.
$
```

This program requires 306 octal locations including 13 recording counters.

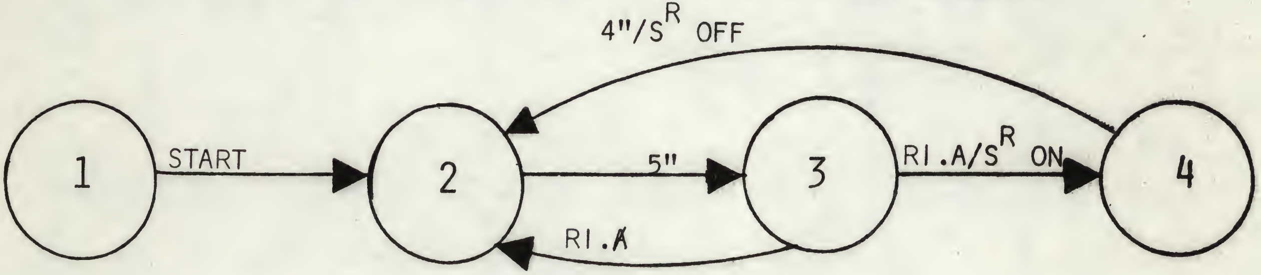
Example 3: This example illustrates the use of multiple schedules and the use of gating to produce Random Interval schedules of reinforcement. Two separate cued components alternate. Either one can be chosen to be first by the experimenter at the start of the session. The two schedules are DRL and RI.

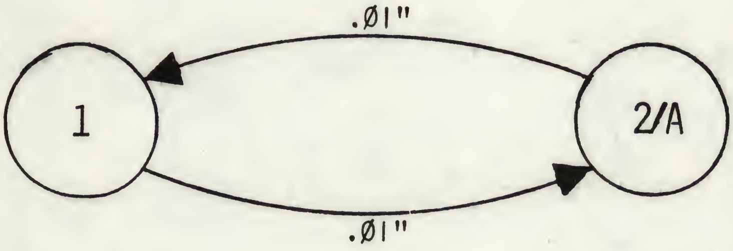
A simple graph of DRL follows:



In this schedule 10" after session start State 3 is entered if no RI is emitted in State 2. If an RI occurs in State 2 the 10" interval is restarted. An RI in State 3 produces reinforcement after which the cycle is restarted.

Random Interval (Farmer, 1963) is shown in the next diagram:



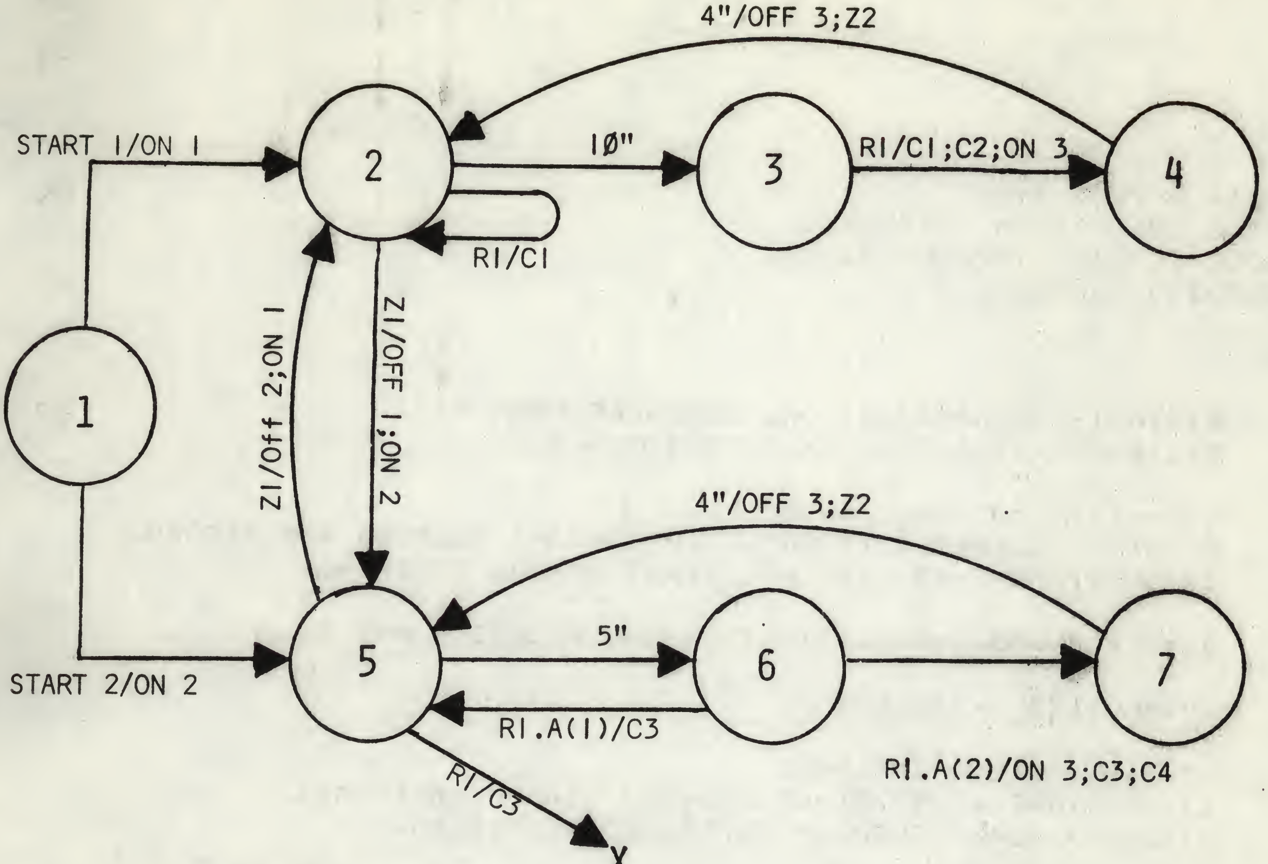


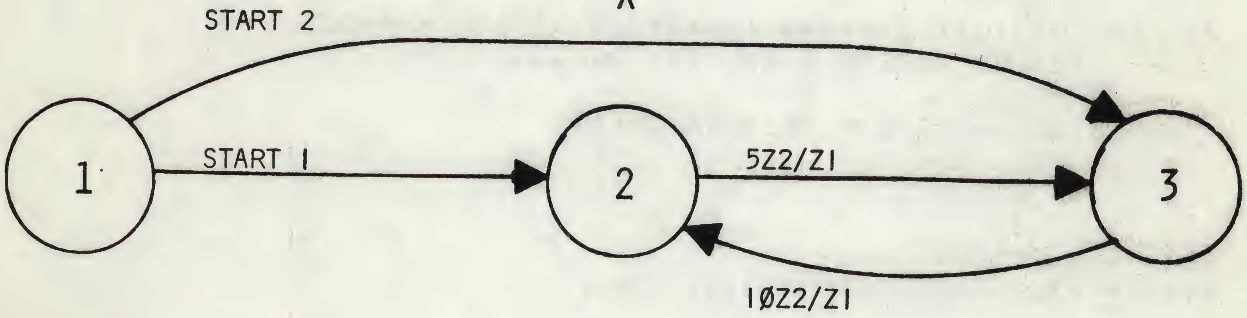
In this schedule 5" after session start State 3 is entered. The first RI is gated with the alternating State Set 2. If State Set 2 is in State 2, reinforcement is initiated. If State Set 2 is in State 1, the next interval is initiated. Since State Set 2 is driven by .01" pulses, the probability of being in State 2 is .5. Therefore, the average interval is 10" (5"/.5).

This program includes recording of all responses and reinforcements for the components. Note that the same recording counters in each component (C1 for DRL and C3 for RI) are incremented in two different states to obtain the sum of both reinforced and unreinforced responses. The second recording counters (C2 and C4) record reinforcements.

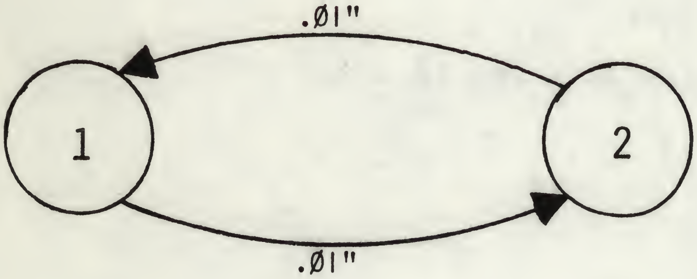
State Set 2 controls the alteration of the two components. The z2 generated at the end of each reinforcement will change components (by a z1) after different numbers of trials. The session machine is of the usual sort.

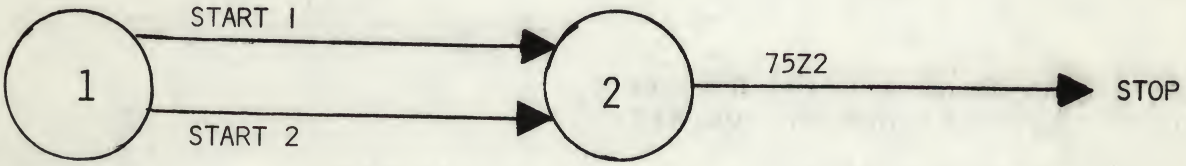
The following graph shows the multiple schedule with separate stimuli marking each component:


4"/OFF 3;Z2
START 1/ON 1
2
1Ø"
3
R1/C1;C2;ON 3
4
R1/C1
1
Z1/Off 2;ON 1
Z1/OFF 1;ON 2
4"/OFF 3;Z2
5
5"
6
7
START 2/ON 2
R1.A(1)/C3
R1.A(2)/ON 3;C3;C4
R1/C3
X


START 2
1
START 1
2
5Z2/Z1
3
1ØZ2/Z1

(A)


.Ø1"
1
2
.Ø1"


START 1
1
2
75Z2
STOP
START 2

```
/APPENDIX B TEST 3
/MULTIPLE DRL-RANDOM INTERVAL
/VARIABLE LENGTH COMPONENTS.
/4 RECORDING-COUNTERS

S.S.1,
S1,
        R12:ON1---->S2/START DRL SCHEDULE FROM TTY.
        R11:ON2---->S5/START RANDOM INTERVAL.
S2,
        10"--->S3/10" DRL
        R1:C1--->S2/RESET TIMER IF 10" HASN'T ELAPSED AND RECORD.
        Z1:OFF1;ON2--->S5/END DRL-START RANDOM INTERVAL.
S3,
        R1:C1;C2;ON3---->S4/START SR AND RECORD R AND SR.
S4,
        4":OFF3;Z2--->S2/END SR.
S5,
        5"----->S6/END INTERVAL.
        Z1:OFF2;ON1--->S2/START DRL-END RANDOM INTERVAL.
        R1:C3---->SX/RECORD UNREINFORCED RESPONSES.
S6,
        R1.A(2):ON3;C3;C4---->S7/START SR,RECORD R AND SR.
        :C3---->S5/OR RECORD R AND TRY AGAIN.
S7,
        4":OFF3;Z2--->S5/END REINFORCEMENT.

S.S.2,
S1,
        R12---->S2/START DRL.
        R11---->S3/START RANDOM INTERVAL.
S2,
        5Z2:Z1---->S3/END DRL ON 5 SRS.
S3,
        10Z2:Z1--->S2/END RANDOM INTERVAL AFTER 10 SRS.

S.S.3=A,
S1,
        .01"---->S2
S2,
        .01"---->S1/P=.5


S.S.4,
S1,
        R12---->S2/START SESSION TIMER.
        R11---->S2/ON EITHER R11 OR R12.
S2,
        75Z2--->STOP/STOP AFTER 75 TOTAL REINFORCEMENTS.
$
```

The program with 4 recording counters requires 445 octal locations. The following examples include testing programs for checking input and output channels and a few basic reinforcement schedules. Documentation is limited to comments on the state tables. It should be noted that sections of these programs can be used as the basis for more complicated procedures by use of the Editor.

```
/APPENDIX B TEST 4
/TEST ALL POSSIBLE INPUT AND OUTPUT CHANNELS
/EACH RN TURNS ON OUTPUT N FOR 2"

S.S.1,
S1,
        R1:ON1--->S2/TURN ON CHANNEL 1.
        R2:ON2--->S2/ON2.
        R3:ON3--->S2/ON3.
        R4:ON4--->S2/ON4.
        R5:ON5--->S2/ON5.
        R6:ON6--->S2/ON6.
        R7:ON7--->S2/ON7.
        R8:ON8--->S2/ON8.
        R9:ON9--->S2/ON9.
        R1Ø:ON1Ø->S2/ON1Ø.
        R11:ON11->S2/ON11.
        R12:ON12->S2/ON12.
S2,
        2":OFF1,2,3,4,5,6,7,8,9,1Ø,11,12-->S1
$
```

```
/APPENDIX B TEST 5
/TEST ALL OUTPUTS AT 2" INTERVALS FROM R1 INPUT

S.S.1,
S1,
        R1:ON1--->S2
S2,
        2":OFF 1;ON2--->S3
S3,
        2":OFF 2;ON3--->S4
S4,
        2":OFF 3;ON4--->S5
S5,
        2":OFF 4;ON5--->S6
S6,
        2":OFF 5;ON6--->S7
S7,
        2":OFF6;ON7--->S8
S8,
        2":OFF7;ON8--->S9
S9,
        2":OFF 8;ON9--->S1Ø
S1Ø,
        2":OFF 9;ON1Ø-->S11
S11,
        2":OFF 1Ø;ON11->S12
S12,
        2":OFF11;ON12->S13
S13,
        2":OFF 12;ON1-->S2
$
```

```
/APPENDIX B TEST 6
/SIDMAN AVOIDANCE WITH RS=1Ø":SS=5"
/2 RECORDING COUNTERS

S.S.1,
S1,
        R12:ON1--->S2/START SESSION CUES
S2,
        1Ø":ON2--->S3/RS INTERVAL
        R1:C1----->S2/RESET TIMER AND RECORD
S3,
        .25":OFF2;C2;Z1--->S4/END SHOCK AND RECORD IT.
S4,
        5":ON2--->S3/SS INTERVAL
        R1:C1---->S2/START RS INTERVAL.

S.S.2,
S1,
        5ØZ1--->STOP
        6Ø'---->STOP
$
```

```
/APPENDIX B TEST 7
/REGULAR REINFORCEMENT (CRF)
/2 RECORDING COUNTERS

S.S.1,
S1,
        R12:ON1--->S2/START SESSION
S2,
        R1:C1*;ON2--->S3
S3,
        2":OFF 2;Z1---->S2

S.S.2,
S1,
        5ØZ1--->STOP
        6Ø'---->STOP
$
```

```
/APPENDIX B TEST 8
/SCHEDULE TO SHAPE RATS ON CRF
/FREE REINFORCEMENT EVERY 6Ø"
/IF 15 R1'S OCCUR CHANGE TO CRF

S.S.1,
S1,
        R12:ON1--->S2/START SESSION
S2,
        6Ø":ON2--->S3/FREE REINFORCEMENT
        R1:ON2;Z1---->S3/EARNED REINFORCEMENT
        Z2---->S4/EARNED 15 THEREFORE CRF STARTS
S3,
        2":OFF 2--->S2/END SR
S4,
        R1:ON2;C1---->S5/START REGULAR REINFORCEMENT
S5,
        2":OFF2-------->S4

S.S.2,
S1,
        15Z1------>S2
S2,
        2":Z2---->S1/CHANGE STATES AT END OF SR.

$
```

```
/APPENDIX B TEST 9
/RANDOM RATIO WITH P=.1Ø

S.S.1,
S1,
        R12:ON1--->S2
S2,
        R1.A(2):ON2--->S3
S3,
        2":OFF2--->S2

S.S.2=A,
S1,
        R12--->S2
S2,
        .1Ø":OFF3--->S3
S3,
        .9Ø":ON3--->S2
$


/APPENDIX B TEST 1Ø
/VARIABLE INTERVAL
/INTERVAL TIMER TURNS OFF WITH SET-UP OF REINFORCEMENT
/1 RECORDING COUNTER

S.S.1,
S1,
        R12:ON1---->S2
S2,
        Z1:Z2----->S3/INTERVAL IS UP.TURN OFF CLOCK.
S3,
        R1:C1;ON2--->S4/START SR
S4,
        2":OFF2;Z3--->S2/END SR AND TURN ON CLOCK.

S.S.2,
S1,
        3Z4:Z1---->S2/FIRST INTERVAL IS 15"
S2,
        Z4:Z1----->S3/SECOND IS 5"
S3,
        5Z4:Z1---->S1/LAST IS 25"

S.S.3,
S1,
        R12---->S2
S2,
        5":Z4--->S2/TIMER
        Z2---->S3/TURN OFF UNTIL SR OBTAINED.
S3,
        Z3---->S2/START TIMER AT END OF SR.
$
```