AD-A036 203 COMPUTER SCIENCES CORP FALLS CHURCH VA F/G 9/2
CSEL INTERFACE UPDATE: K-BAND MICROPROCESSOR DEMONSTRATION.(U)
DEC 76 D O ALWINE, R A GLICKSMAN, R G MURRAY F33615-75-C-1229
UNCLASSIFIED CSC-R-3328-1 AFAL-TR-76-169 NL

1 OF 2
AD
A036203

AFAL-TR-76-169

CSEL INTERFACE UPDATE: K-BAND MICROPROCESSOR DEMONSTRATION

COMPUTER SCIENCES CORPORATION
6565 ARLINGTON BOULEVARD
FALLS CHURCH, VIRGINIA 22046

DECEMBER 1976

TECHNICAL REPORT AFAL-TR-76-169
FINAL REPORT FOR PERIOD MARCH 1975 - JUNE 1976

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

This technical report has been reviewed and is approved for publication.

PAUL F. HUMEL, 1LT, USAF
Project Engineer

FOR THE COMMANDER

JAMES D. EVERETT, Col, USAF
Chief, System Avionics Division
Air Force Avionics Laboratory

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFAL-TR-76-169
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): CSEL INTERFACE UPDATE: K-Band Microprocessor Demonstration
5. TYPE OF REPORT & PERIOD COVERED: Final Report, Mar[redacted]75 - Jun[redacted]76
6. PERFORMING ORG. REPORT NUMBER: CSC-R-3328-1
7. AUTHOR(s): Douglas O. Alwine, Robert A. Glicksman, Richard G. Murray, Charles F. Pavey
8. CONTRACT OR GRANT NUMBER(s): F33615-75-C-1229
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Sciences Corporation, 6565 Arlington Blvd., Falls Church, VA 22046
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 12270105
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Avionics Laboratory (AFAL/AAD), Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433
12. REPORT DATE: Dec[redacted]76
13. NUMBER OF PAGES: 169
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited.
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
19. KEY WORDS (Continue on reverse side if necessary and identify by block number): Microprocessor, K-Band Terminal Simulator, Simulation, Software, Frequency Synthesizer, Doppler, Frequency HOP, MFSK, Random Access Memory
20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

The objective of this demonstration/study is to provide the Air Force with basic data and techniques for expanding the capabilities of the K-Band Terminal Simulator. The current simulator consists of three communication channels, each with several real-time effects (Doppler shift, frequency hop, signal fading, and MFSK) driven by a minicomputer which updates all of these real-time effects every 5 milliseconds. Expansion of the K-Band Terminal Simulator along present lines would increase the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) burden on the minicomputer and therefore increase the basic 5 millisecond time step, which may already be too long for some applications.

The results of this study show that an alternative, vastly superior approach to expansion of the simulator would be to use a separate microprocessor to drive each individual channel. This would relieve the main minicomputer of the fast real-time chores and leave it with only non-time critical system control, operator interface, and housekeeping tasks. This approach would give almost unlimited expansion capability and would allow reducing the basic time step somewhat.

To further assess the feasibility of this approach, a "brassboard" demonstration unit was constructed. This unit uses an 8080 based microprocessor system to drive a frequency synthesizer (the heart of a K-Band Terminal Simulator channel) in real-time from tables of hop, Doppler and MFSK effects. These tables are pre-computed by the CSEL minicomputer from operator input data in a manner similar to that of the existing K-Band Terminal Simulator.

The brassboard demonstration unit ran successfully with an observed time step of 2.45 milliseconds, clearly proving the feasibility of this approach. Complete results, along with all hardware and software documentation are contained in this final report.

PREFACE

This Technical Report was prepared by Computer Sciences Corporation, 6565 Arlington Boulevard, Falls Church, Virginia 22046, for the Air Force Avionics Laboratory under Contract F33615-75-C-1129, job order 12273205. The work on this effort was carried out by Messrs. Douglas O. Alwine, Robert A. Glicksman, Richard G. Murray, and Charles F. Pavey. Lt. Paul F. Humel, USAF, AFAL/AAD, was the project engineer for the Air Force Avionics Laboratory.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

Figure

SECTION I
INTRODUCTION

The objective of this demonstration/study is to provide the Air Force with basic data and techniques for expanding the K-Band Terminal Simulator. The current K-Band Terminal Simulator consists of three signal sources, each capable of a wide variety of modulation types, to simulate a wide variety of transmitters. All three simulated transmitters are driven in real-time by a single minicomputer to simulate the effects of Doppler shift, frequency hop, signal fade, and MFSK modulation. In real life, functions such as Doppler shift and fade are continuous. In order to simulate these functions with a system driven by a digital controller, it is necessary to quantize the changes into discrete steps. In order to have a valid simulation, it is necessary to make these steps very small and hence, they must occur rapidly. The present K-Band Terminal Simulator is capable of outputting updated values for the real-time effects every 5 milliseconds. However, because a single processor is used to drive all of the real-time effects, any expansion of the system, either in the number of carriers or the number of effects per carrier, will necessitate increasing the size of this basic time step.

Since it appears that the Air Force may have the need to increase the number of simulated transmitters and to reduce the basic time step from 5 milliseconds to 1 or 2 milliseconds, it becomes necessary to examine alternative means of controlling the K-Band Simulator. The concept examined here is to control each transmitter simulator in real-time with its own microprocessor, leaving the main minicomputer with only non-time critical system control, operator interface, and housekeeping tasks. This would allow an almost unlimited, modular, expansion capability for the K-Band Terminal Simulator, and expansion could be accomplished without lengthening the time step.

The brassboard demonstration system, described herein, can calculate new values for Doppler shift, frequency hop, and MFSK, add these to the nominal carrier frequency and output an updated frequency to a frequency synthesizer approximately every 2.45 milliseconds.

SECTION II
DESCRIPTION OF SYSTEM

The brassboard demonstration unit consists of a commercial microprocessor development system, programmed to accept standard files from the CSEL minicomputer and drive a frequency synthesizer from these files to simulate Doppler shift, frequency hop, and MFSK. The CSEL PDP-11/20 was programmed to accept operator inputs from the console, build the files, and transfer them to the microprocessor memory.

This demonstration system consists of an INTELLEC 8/MOD 80 microcomputer, a Rockland 5100 frequency synthesizer, and a frequency discriminator to aid in observing the real-time effects on an oscilloscope. Detailed descriptions of this hardware can be found in Section IV.

Values for Doppler shift are taken from the table directly. Values for hop and MFSK are picked pseudorandomly from their respective tables by pseudorandom codes generated in software. These three effects are added to the nominal carrier frequency, converted to the proper format to drive the frequency synthesizer, and outputted, demonstrating clearly the feasibility of controlling the K-Band Terminal Simulator hardware with microprocessors.

SECTION III
RESULTS

The microprocessor was interfaced with the Rockland 5100 frequency synthesizer and the DEC PDP-11/20. It successfully received tables of Doppler shift, hop, and MFSK from the PDP-11/20, and was able to drive the frequency synthesizer in real-time, outputting an updated frequency approximately every 2.45 milliseconds. That is, a new value was calculated for each of the parameters of Doppler shift, MFSK, and frequency hop; that all three new values were added to the nominal carrier frequency and the resulting frequency was formatted and sent to the frequency synthesizer every 2.45 milliseconds. Delay loops were used as necessary to get all output cycles as nearly equal in time as possible. These delay loops would be unnecessary if a real-time clock were incorporated into the microprocessor system. The cost of this clock was not considered justified in this demonstration set-up. Such a clock would be required however, if it is decided to upgrade the K-Band Terminal Simulator with microprocessors, as it would permit the required precise control of the output rate.

The 2.45 millisecond output rate is faster than the present 5 millisecond rate currently available in the K-Band Terminal Simulator, but not nearly as fast as could be attained with some of the newer microprocessors. Many of the newer units offer shorter cycle time and other improvements, such as indirect addressing or an improved instruction set. At least one unit has a 16-bit word length, which could increase speed both by requiring fewer memory cycles to execute a single instruction, (the 8080 fetches as many as three bytes per instruction) and by making it simpler to generate the three 10-bit binary words needed to drive the Rockland synthesizer.

It is not unlikely that a microprocessor could, at this time, be found that would be capable of reducing the observed 2.45 millisecond step time to less than 1 millisecond. New and upgraded microprocessor products are currently being introduced on an almost daily basis and these units offer further improvements in speed, instruction set and I/O flexibility over the original 8080 used in the brassboard demonstration unit. Amid all of this, it is also worthy to note that the prices of microprocessor and related semiconductor products are currently dropping very rapidly. Computer Sciences

Corporation (CSC) feels, therefore, that making a specific recommendation as to the "best" choice of microprocessor at this time would be unjustified. Any information obtained now would soon be too obsolete for the Air Force to use. We, therefore, recommend that the Air Force wait until a microprocessor per access expansion of the CSEL facility is required before attempting to select the "best" microprocessor to use.

A secondary fallout of the rapid drop in semiconductor (specifically memory) prices is a simplification in the recommended configuration for a microprocessor controlled CSEL access. When the processor-per-access technique was first proposed, CSC felt that a practical, operational simulator would have a central core memory which would be maintained by the CSEL minicomputer and shared by each microprocessor controlled access for obtaining hop, Doppler, MFSK, and fade parameters. This configuration required expensive DMA interfaces and attention to critical timing situations. We now feel that the reduced price of semiconductor memory makes it economically feasible for each microprocessor to be provided with enough local RAM memory to store a significant amount of real-time data. The data would be stored in a "double buffer" format, wherein half of the local memory would at any time be filled with current parameters while the other half would be receiving a "downline load" from the CSEL minicomputer. This technique will greatly reduce the complexity of the hardware microprocessor to minicomputer interface and be free of the critical timing burden inherent in a single shared memory system.

The interface between the minicomputer and microprocessor can be either serial or parallel. In the brassboard demonstration system, a 2400 baud serial interface was provided. Approximately 2 minutes was required to transfer the entire set of tables to the microprocessor over this interface. This could be reduced to about 30 seconds by increasing the speed to 9600 baud. For real-time operation on a 1-millisecond time step, and on non-repetitive tables a parallel interface would be required. A serial interface will certainly suffice for repetitive tables, as demonstrated with this brassboard system.

Serial interfaces can be purchased, often complete with software support, at reasonably low cost. Parallel, non-DMA interfaces are generally more expensive and require custom software drivers to be written for them. Parallel DMA interfaces are by far the most expensive in both hardware and software and generally produce the greatest number of integration problems. Therefore, the particular operational requirements of a microprocessor-per-access simulator will have to be considered when attempting to configure such a system in the most cost-effective manner.

SECTION IV
DESCRIPTION OF HARDWARE

The hardware block diagram is shown in Figure 1. The major blocks of this diagram are described below.

CONTROL PROCESSOR

The PDP-11/20 is the processor that is normally used to drive the K-Band Terminal Simulator. The software prepared for the PDP-11 (see Section 5) allows the operator to build tables of values for frequency hop, MFSK, and Doppler, and transfer these tables to the microcomputer memory. The interface to the PDP-11 is through a DL11 interface module normally used to drive a second VTO5 console.

MICROCOMPUTER

The microcomputer in the block diagram is an INTELLEC 8/MOD 80 development system. While a number of microprocessor chips were available at the start of the contract, only the Intel 8080 was available off-the-shelf, with the support hardware and software needed to develop this system.

The microcomputer supplied for this demonstration has the following modules:

- 1 - IMM8-83 CPU Module
- 2 - IMM8-61 I/O Cards
- 1 - IMM8-63 Output Card
- 2 - IMM6-28 Random Access Memory Cards, 4K each
- 3 - IMM6-26 PROM Cards, 4K bytes each
- 1 - IMM6-76 PROM Programmer
- 1 - Front Panel Control Module

The microcomputer is normally supplied with one IMM8-61 I/O card and a second was added to provide a serial interface to the DEC DL11-C I/O module. An IMM8-63 output card was added to provide output ports to drive the synthesizer.

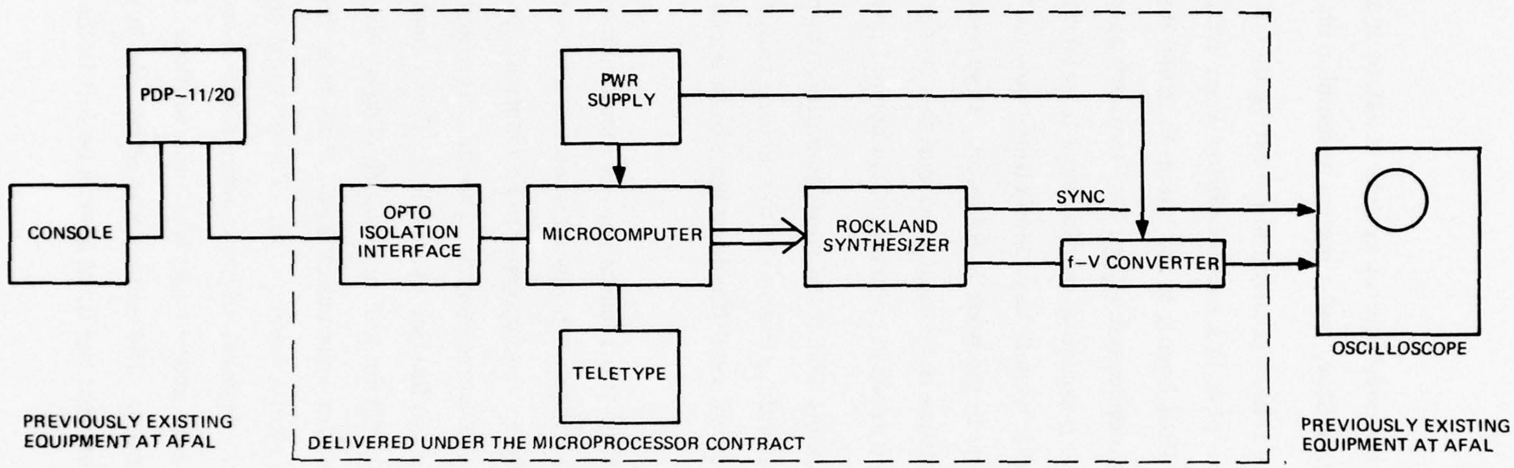


Figure 1. Block Diagram of the Demonstration System

The software included with the development system consists of a system monitor that resides in 2K of PROM, a text editor, and a macro assembler on paper tape.

CSC has found several peculiarities involved in operating this device that the Air Force should be cognizant of. The first is a race condition when bringing up the System Monitor which can be avoided if, after keying in the "jump to 3800" instruction in locations 0, 1, and 2, the reset key is depressed before the "memory access" key is returned to its normal position. The instruction manual says to return the "mem access" key to normal before depressing the "reset" key. Sometimes several tries are required to start the machine if the sequence in the book is followed. Discussions with the factory have verified that a race condition does exist, and that the "reset" key should be depressed first, then the "memory access" returned to its normal position.

The second peculiarity is simply that if the room is chilly, the machine needs a few minutes to warm up. Discussions with personnel at the factory indicate that this is normal. After 5-10 minutes of operation, no difficulties have been encountered, even when the room was quite cool.

The third peculiarity is that the two random access memory modules must be installed in the same slots in which they were placed when the microprocessor was delivered to AFAL. The two modules are supposed to be identical, and indeed they will usually function normally if they are interchanged. The only time any memory problem has been apparent is when running the Intellec assembler. If the assembler is run with the memory cards reversed, the program will not stop reading at the end of the source tape, but will continue until the reader runs out of tape. This is a "hard" error, occurring every time an attempt is made to read a source tape. The problem goes away when the two RAM boards are interchanged. Several attempts were made to isolate the error by writing various patterns into memory and reading what was written. No error was ever found, and the attempt was abandoned. The factory was of no help in explaining this situation, other than recommending that the RAM boards be left in the position wherein no problems were encountered.

FREQUENCY SYNTHESIZER

The frequency synthesizer is a Rockland model 5100. This unit was chosen for its fast switching speed and its ability to switch frequency without any phase discontinuity in the output. A synthesizer with these features was required in order to obtain small Doppler steps at a fast rate.

A description of this device can be found in the manufacturer's manual.

OPTO-ISOLATOR INTERFACE

Both the DEC DL11 and the Intel IMM8-61 I/O modules utilize a serial, current-loop output, and were designed to drive a console such as a CRT or a teletype. These modules were interfaced with opto-isolators (see Figure 2). An opto-isolator consists of a light emitting diode (LED) and a phototransistor, mounted in a common package. Forward current through the LED causes it to emit light which turns on the transistor. This arrangement provides a high degree of electrical isolation, avoiding the problems with ground loops that might occur if the grounds of the two machines were connected together.

FREQUENCY TO VOLTAGE CONVERTER

The frequency-to-voltage converter was constructed as an aid in determining if the output of the frequency synthesizer is being varied correctly. A schematic diagram of this assembly is shown in Figure 3. The CA3012 limiter amp provides a constant level to the input of the monostable flip flop (74121). A pulse train is produced at the one shot output with all pulses being the same width, but with a pulse repetition rate equal to the frequency of the synthesizer output. R_2 and C_2 average the output of the one shot. Since the pulse width is constant, the dc voltage across C_2 is proportional to the frequency. Hence, the output voltage is proportional to the synthesizer output frequency.

CONNECTING CABLE

The values outputted to the synthesizer are transmitted via an IMM8-63 output module. A list of the wiring, from the rear panel of the microcomputer to the synthesizer input is shown in Figure 4. An additional, individual wire coming from the

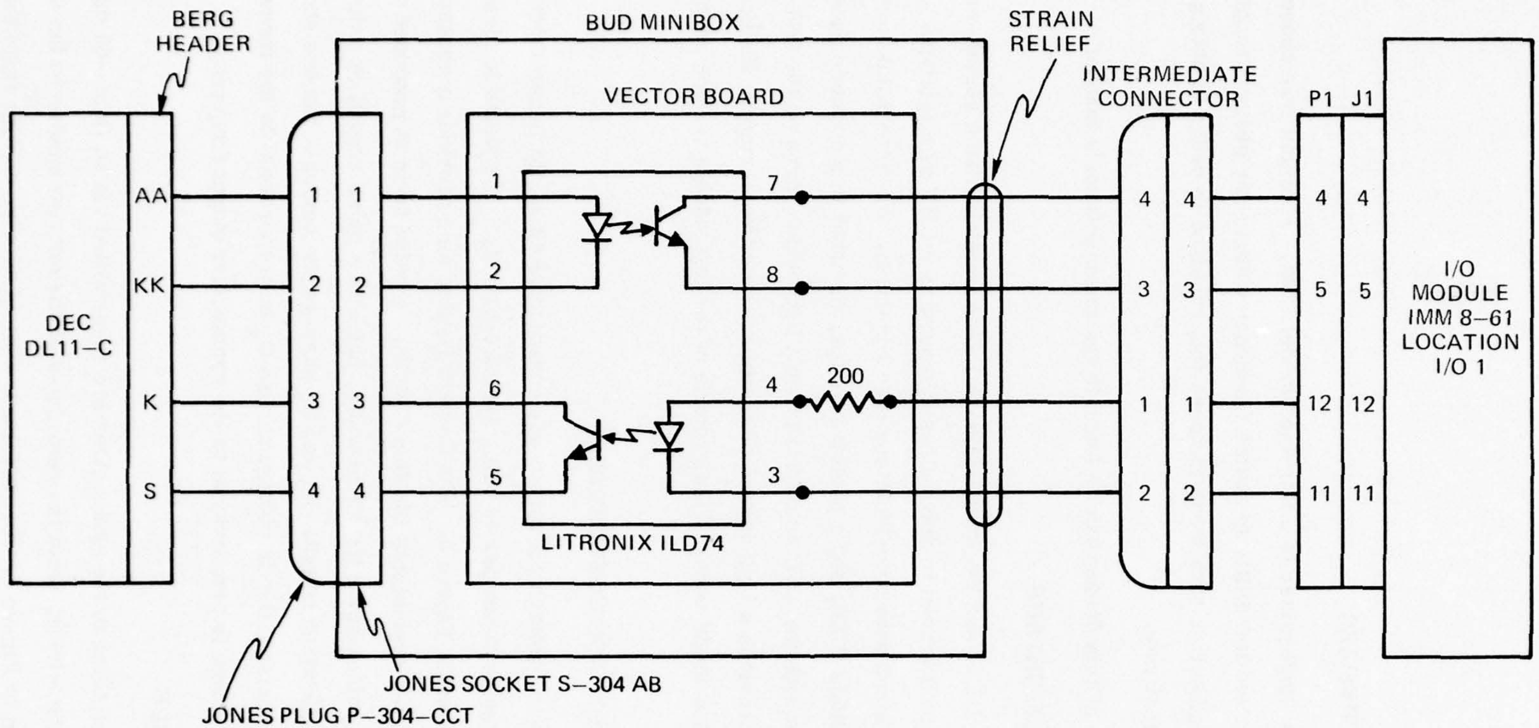


Figure 2. Opto-Isolator Interface Unit - Schematic Diagram

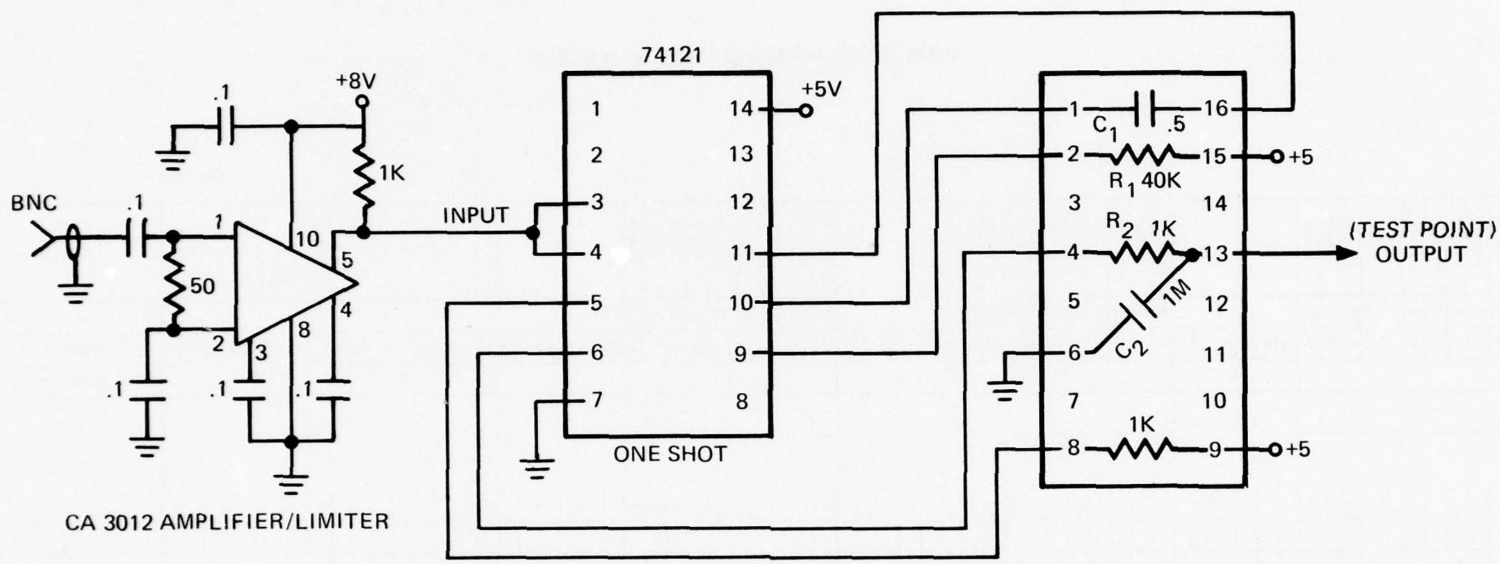


Figure 3. Frequency to Voltage Converter - Schematic Diagram

IMM 8–63 PORT	8							
ADDRESS	15							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	X	X	X	X	X	X	X	LOAD
INTELLEC CONNECTOR	6							
INTELLEC PIN #	36	35	17	16	34	33	15	14
ROCKLAND PIN #								12
WIRE COLOR								PINK

IMM 8–63 PORT	7							
ADDRESS	14							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	X	X	X	X	X	X	X	MHz
INTELLEC CONNECTOR	6							
INTELLEC PIN #	32	31	13	12	30	29	11	10
ROCKLAND PIN #								10
WIRE COLOR								TAN

IMM 8–63 PORT	6							
ADDRESS	13							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	X	X	X	X	X	X	512	256
							← KHz →	
INTELLEC CONNECTOR	6						6	
INTELLEC PIN #	28	27	9	8	26	25	7	6
ROCKLAND PIN #							22	21
WIRE COLOR							WHITE	GREY

IMM 8–63 PORT	5							
ADDRESS	12							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	128	64	32	16	8	4	2	1
	← KHz →							
INTELLEC CONNECTOR	6							
INTELLEC PIN #	24	23	5	4	22	21	3	2
ROCKLAND PIN #	20	19	18	17	16	15	14	13
WIRE COLOR	PURPLE	BLUE	GREEN	YELLOW	ORANGE	RED	BROWN	BLACK

IMM 8–63 PORT	4							
ADDRESS	11							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	X	X	X	X	X	X	512	256
							← Hz →	
INTELLEC CONNECTOR	8						8	
INTELLEC PIN #	36	35	17	16	34	33	15	14
ROCKLAND PIN #							47	46
WIRE COLOR							TAN	WH/BLK

IMM 8–63 PORT	3							
ADDRESS	10							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	128	64	32	16	8	4	2	1
	← Hz →							
INTELLEC CONNECTOR	8							
INTELLEC PIN #	32	31	13	12	30	29	11	10
ROCKLAND PIN #	45	44	43	42	41	40	39	38
WIRE COLOR	WH/BLU	WH/GN	WH/YEL	WH/RED	PINK	RED/BLK	RED/GN	RED/YEL

IMM 8–63 PORT	2							
ADDRESS	9							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	X	X	X	X	X	X	512	256
							← mHz →	
INTELLEC CONNECTOR	8						8	
INTELLEC PIN #	28	27	8	9	26	25	7	6
ROCKLAND PIN #							35	34
WIRE COLOR							WHITE	GRAY

IMM 8–63 PORT	1							
ADDRESS	8							
BIT #	7	6	5	4	3	2	1	0
WEIGHT OF BIT	128	64	32	16	8	4	2	1
	← mHz →							
INTELLEC CONNECTOR	8							
INTELLEC PIN #	24	23	5	4	22	21	3	2
ROCKLAND PIN #	33	32	31	30	29	28	27	26
WIRE COLOR	PURPLE	BLUE	GREEN	YELLOW	ORANGE	RED	BROWN	BLACK

X = NOT USED

Figure 4. Synthesizer Interface

synthesizer connector carries a software generated load pulse, pin 12, which is used for troubleshooting, for synchronizing an oscilloscope, or for measuring the rate at which the synthesizer frequency is being updated on a counter.

POWER SUPPLY CHASSIS

This chassis contains six assemblies, A1 through A6 (see Figure 5). The functions of these assemblies are as follows.

Connector J48 on the rear of the Intellec is an AC connection for a teletype. An AC card from assembly A1 is plugged into this outlet (see schematic, Figure 6). When the microcomputer is turned on, it energizes relay K1, which energizes the teletype and the three power supplies, A4, A5, and A6. A6 is a -10 volt supply and A5 is a +5 volt supply. These two supplies are needed to supplement the ones inside the Intellec when the microprocessor system consists of more than a CPU, 2 RAM boards, 1 ROM board, and 2 I/O boards. If the output board driving the synthesizer is removed, and the extra ROM board in ROM slot 2 is removed, the internal power supplies are sufficient, and the external ones can be removed. The external power supplies are adequate to power the computer with <u>all</u> card slots filled.

Assemblies A2 and A3 are the opto-isolator interface and the frequency to voltage converter, respectively. Both units have already been discussed.

A4 is a +8 volt power supply which is used to power the CA3012 limiter/amplifier in the frequency to voltage converter. A schematic diagram of this unit is shown in Figure 7.

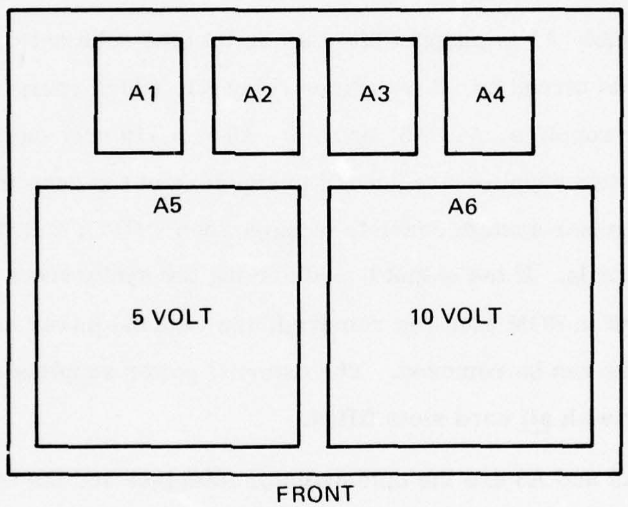


Figure 5. Power Supply Chassis

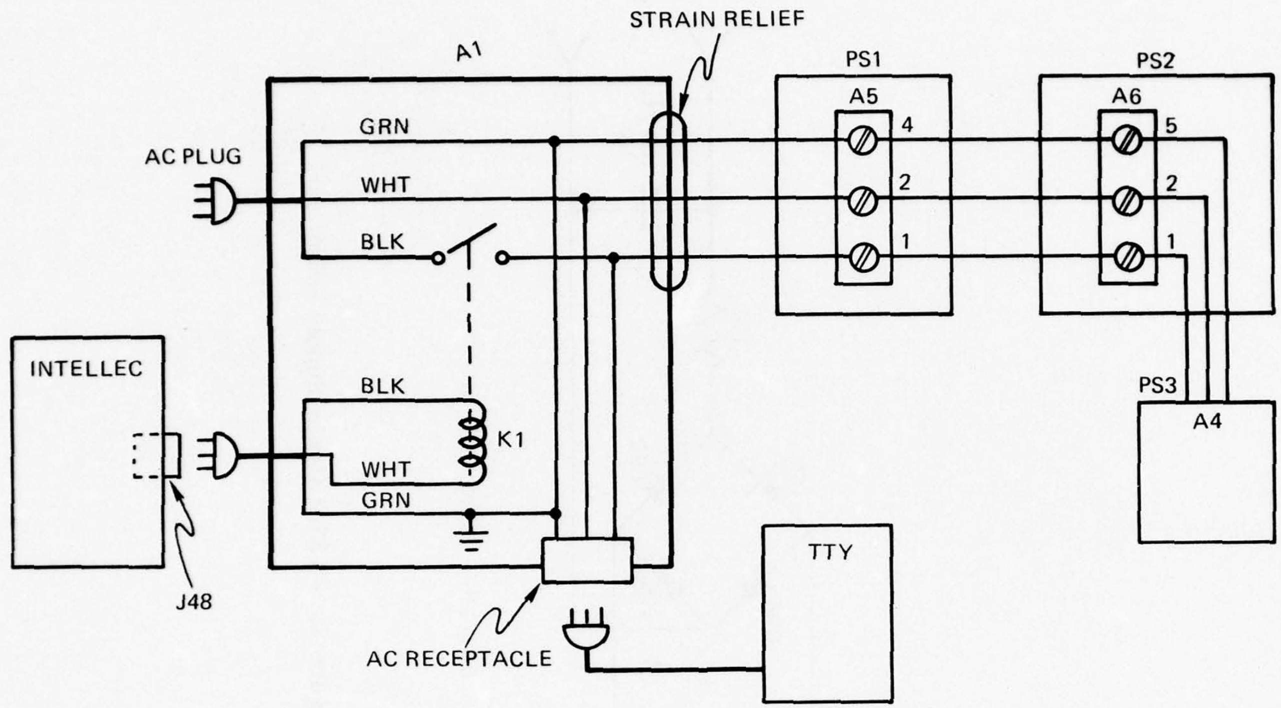


Figure 6. A.C. Power Control Unit

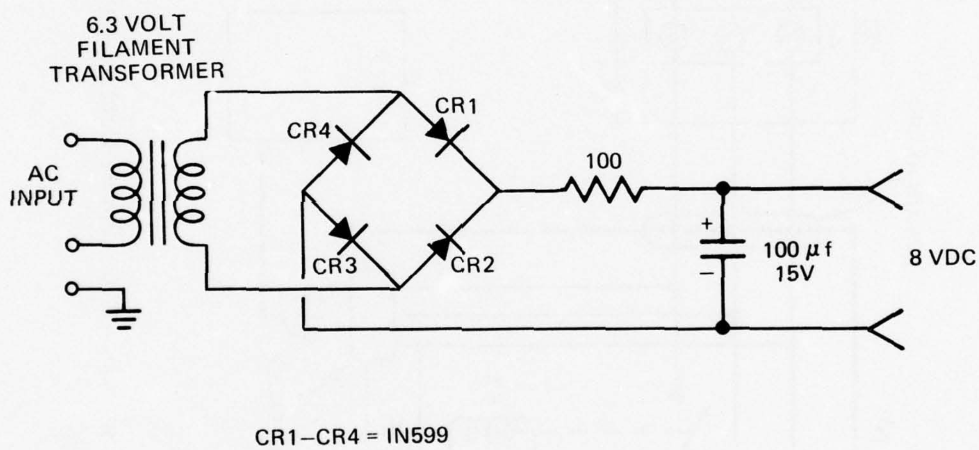


Figure 7. +8 Volt Power Supply Schematic

SECTION V
SOFTWARE DESCRIPTION

INTRODUCTION

The software for the brassboard demonstration system is comprised of two modules — a static mode module and a real-time module. The static mode module is responsible for building and transmitting the system parameter tables and pattern value tables to the Intellec microprocessor. The real-time module is responsible for the execution of the real-time mode.

Detailed documentation on each main program and subroutine in the demonstration software system follows. This documentation includes a description of each module, program, and subroutine, a flowchart, and a computer listing.

STATIC MODE MODULE

The static mode module is responsible for building and transmitting the system parameter and pattern value tables used in the real-time mode. This module is comprised of a mainline program and many functional subroutines.

All variable elements in the system parameter table are input via the system console using keywords to specify a particular variable. Each keyword has an associated subroutine that is responsible for managing the input of its associated parameter value. A listing of the keywords and associated subroutines appears in Table 1.

Prior to conversion to real-time form, each effect parameter value established through console input is stored in labeled common as a four byte BCD value. During conversion to real-time form, each BCD value is converted to double precision and then to the binary form of the real-time mode. This process could have been simplified using the DECODE system routine, but would have defeated our design objective of attempting to keep the module as system independent as possible.

HOP and MFSK table values are input by the module through the console. The table size for each must be a power of two with a maximum size of 64 for MFSK and 128 for HOP. The Doppler values are built by successive additions of an operator supplied offset to the base frequency. The MFSK and HOP values are converted to BCD form and then to their packed decimal storage form. The DOP offset is also converted to BCD form, but is then converted to double precision for the addition process used in determining each successive value. Each value so generated must be converted from double precision to packed decimal. Subroutines internal to the module are used for the conversions from BCD to double precision and from double precision to packed decimal.

Transmission of system parameter and table values is performed via the subroutines designed to transfer data according to an expected sequencing of the real-time mode reader program. Error checking is performed in these subroutines to guarantee the successful transfer of the data to the Intellec microprocessor. An assembly language routine is used to perform the actual byte transfers to the Intellec using the DL11 interface. This is the only system and machine dependent element of the module.

Besides the transmission error checking, the module also performs extensive error checking of operator input for illegal input strings and values out of range. A final consistency check is also performed on the value tables to ensure that only valid frequencies will be output in real-time mode by the frequency synthesizer. Error messages are output to the system console if errors are detected.

Labeled common errors are used extensively throughout the module for communication between the subroutines in the module. All areas are intialized in a BLOCK DATA subprogram. A description of each area, purpose, and initial values is contained in the BLOCK DATA subprogram description.

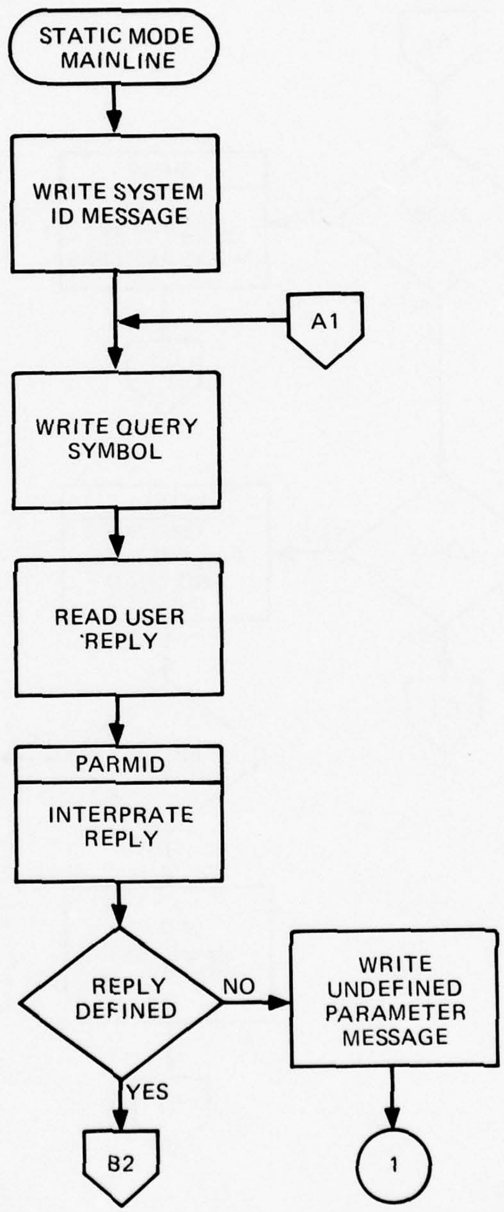
TABLE 1. SYSTEM KEYWORDS

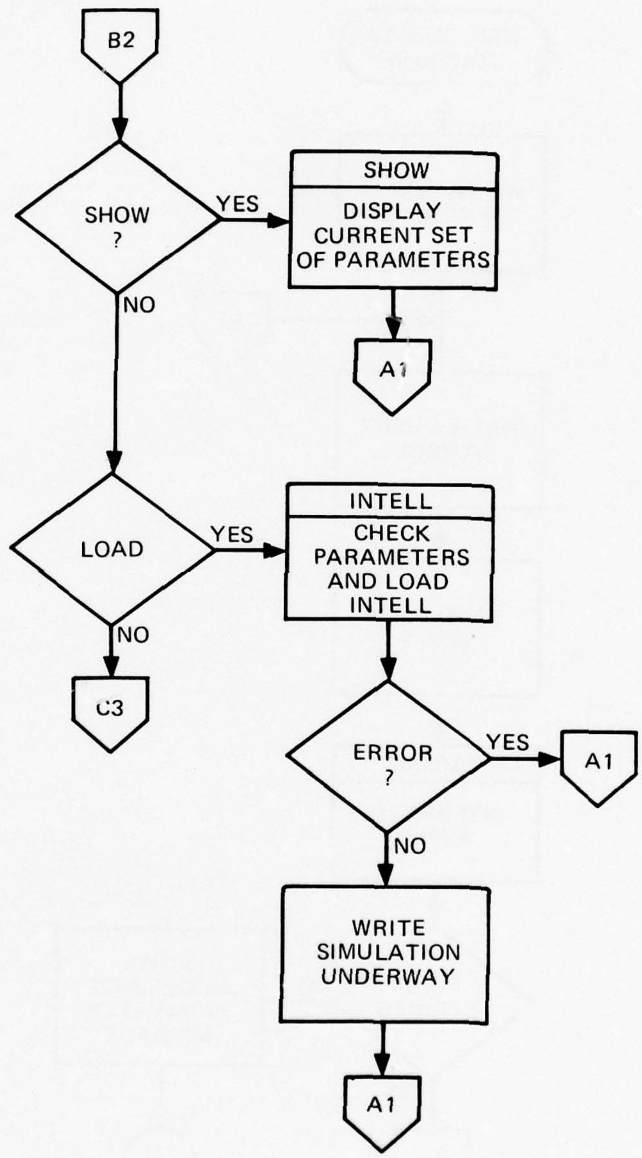
KEYWORD	DESCRIPTION	SUBROUTINE
SHOW	Display current values of effect start, end, cycle, base frequency, and simulation end.	SHOW
LOAD	Perform system parameter and pattern value transfer	INTELL
FREQ	Input base frequency value	FREQ
HOPL	Input HOP table values	HOPLD
MFSL	Input MFSK table values	MFSKLD
DOPL	Build DOP table	DOPLD
HSTR	Input HOP start time	IBIE
MSTR	Input MFSK start time	IBIE
DSTR	Input DOP start time	IBIE
HEND	Input HOP end time	IBIE
MEND	Input MFSK end time	IBIE
DEND	Input DOP end time	IBIE
HCYC	Input HOP cycle value	STEPS
MCYC	Input MFSK cycle value	STEPS
DCYC	Input DOP cycle value	STEPS
SEND	Input simulation end value	ENDSIM

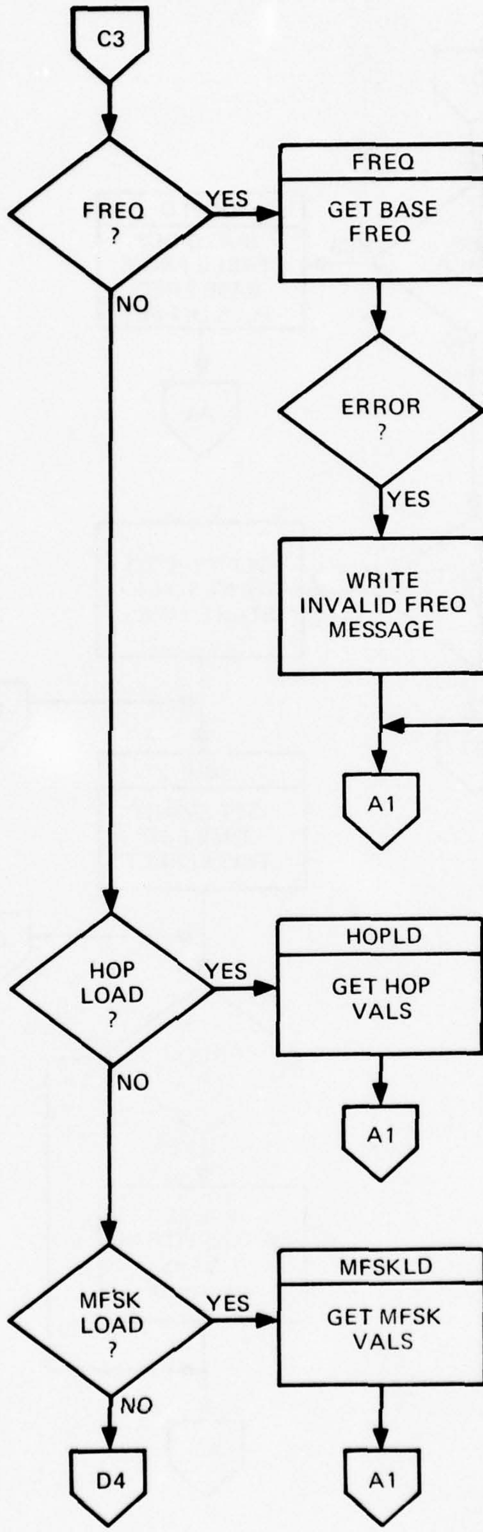
STATIC MODE MAINLINE

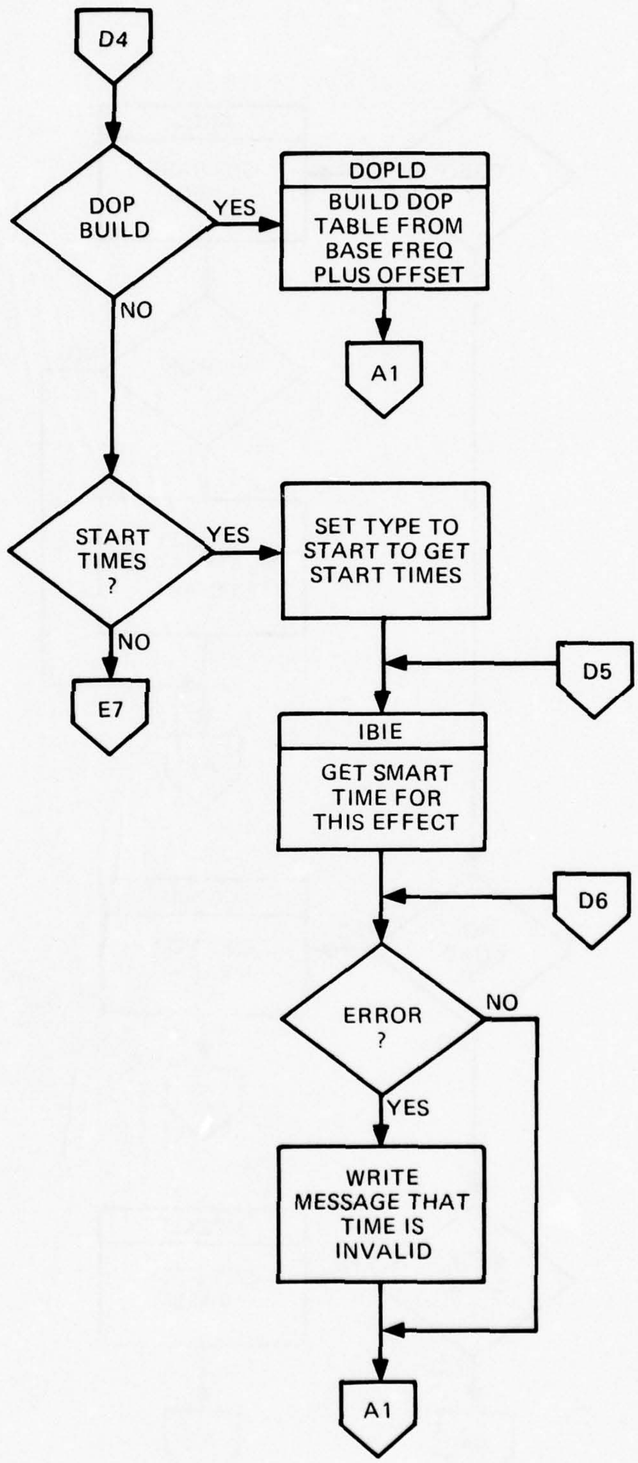
This program is used to perform input of the function keywords that govern system setup and execution and to call the appropriate subroutine that manages the execution of the keyword specified.

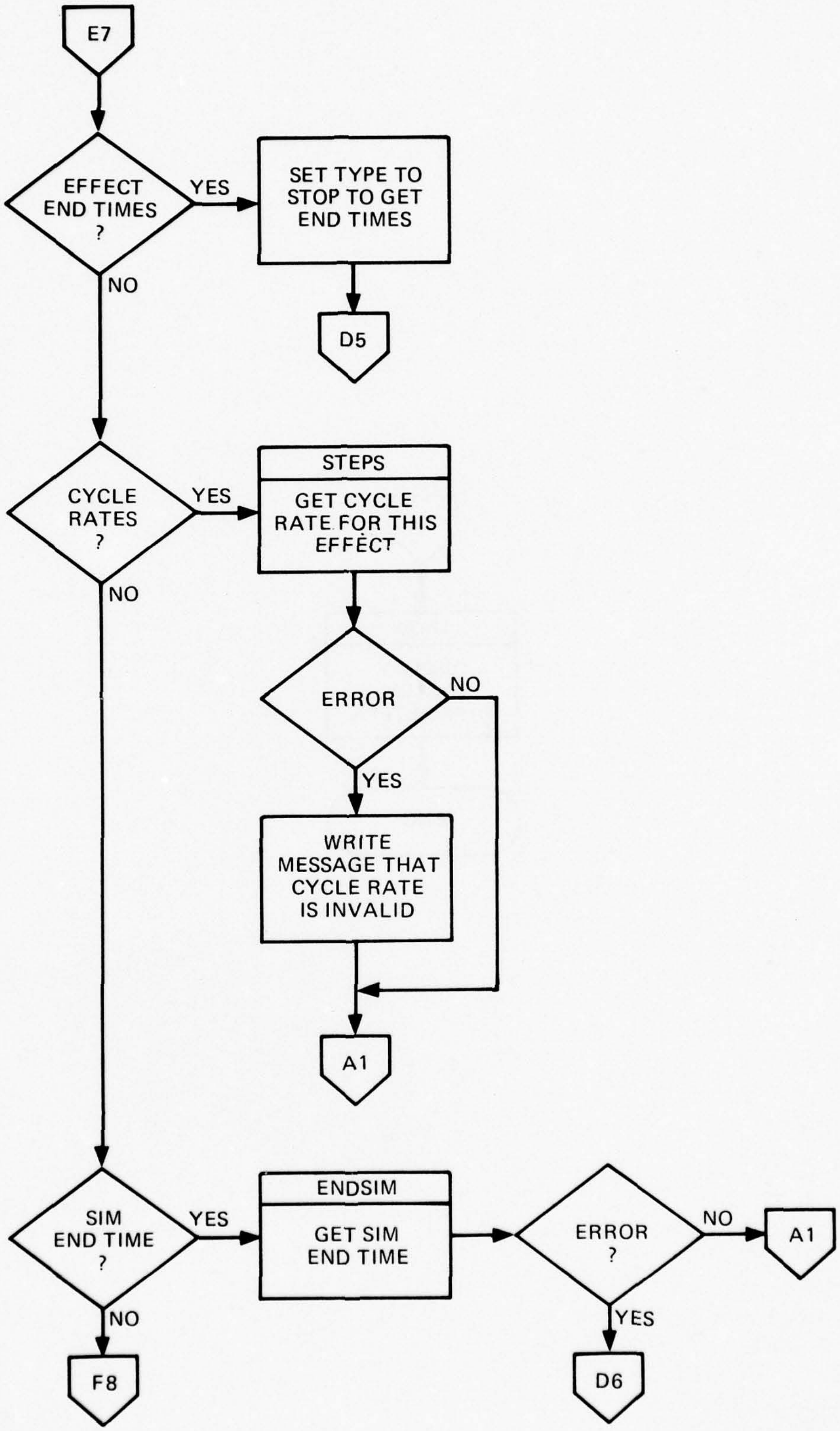
The "PARMID" routine is used to determine the identity of the function keyword supplied. This program successively examines the value returned from the routine to determine the appropriate subroutine to invoke to complete the keyword processing.

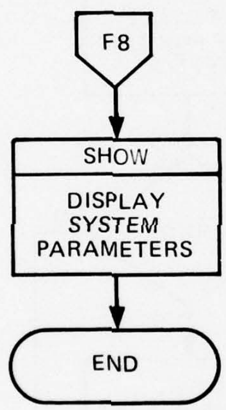

STATIC MODE MAINLINE
WRITE SYSTEM ID MESSAGE
A1
WRITE QUERY SYMBOL
READ USER REPLY
PARMID
INTERPRATE REPLY
REPLY DEFINED
NO
YES
WRITE UNDEFINED PARAMETER MESSAGE
B2
1


B2
SHOW ?
YES
SHOW
DISPLAY CURRENT SET OF PARAMETERS
A1
NO
LOAD
YES
INTELL
CHECK PARAMETERS AND LOAD INTELL
NO
C3
ERROR ?
YES
A1
NO
WRITE SIMULATION UNDERWAY
A1


C3
FREQ ?
YES
NO
FREQ
GET BASE FREQ
ERROR ?
NO
YES
WRITE INVALID FREQ MESSAGE
A1
HOP LOAD ?
YES
NO
HOPLD
GET HOP VALS
A1
MFSK LOAD ?
YES
NO
MFSKLD
GET MFSK VALS
D4
A1


D4
DOP BUILD
YES
NO
DOPLD
BUILD DOP TABLE FROM BASE FREQ PLUS OFFSET
A1
START TIMES ?
YES
NO
E7
SET TYPE TO START TO GET START TIMES
D5
IBIE
GET SMART TIME FOR THIS EFFECT
D6
ERROR ?
NO
YES
WRITE MESSAGE THAT TIME IS INVALID
A1


E7
EFFECT END TIMES ?
YES
SET TYPE TO STOP TO GET END TIMES
D5
NO
CYCLE RATES ?
YES
STEPS
GET CYCLE RATE FOR THIS EFFECT
ERROR
NO
YES
WRITE MESSAGE THAT CYCLE RATE IS INVALID
A1
NO
SIM END TIME ?
YES
ENDSIM
GET SIM END TIME
ERROR ?
NO
A1
YES
D6
NO
F8


F8
SHOW
DISPLAY SYSTEM PARAMETERS
END

```
C***********************************************************************
C
C       KBAND DEMONSTRATION STATIC MODE PROGRAM
C
C**** THIS IS THE MAINLINE PROGRAM FOR THE KBAND MICROPROCESSOR
C**** DEMONSTRATION SYSTEM, ITS MAIN FUNCTION IS TO QUERY THE
C**** USER FOR A FUNCTION KEYWORD AND CALL THE APPROPRIATE
C**** ROUTINE THAT WILL PERFORM THAT FUNCTION, THE PARMID
C**** THAT IDENTIFIES THE FUNCTION TO BE PERFORMED. THE KEYWORDS
C**** AND INDICES ARE AS FOLLOWS,
C       SHOW=1      LOAD=2      FREQ=3      HOPL=4      MFSL=5
C       DUPL=6      HSTR=7      MSTR=8      DSTR=9      HEND=10
C       MEND=11     DEND=12     HCYC=13     MCYC=14     DCYC=15
C       SEND=16
C
C***********************************************************************
        COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
        COMMON/TABLES/HOP,DOP,MFSK
        COMMON/INBUF/INLINE
        COMMON/SPDEC/CHARS,ICAR
        BYTE CHARS(20)
        BYTE INLINE(20),HOP(640),DOP(5120),MFSK(320),PBASE(12)
        REAL PCYCLE(3)
        REAL PSTART(3),PSTOP(3),PSMEND,TYPE,R
        INTEGER MASK(3)
C**** WRITE SYSTEM IDENTIFICATION MESSAGE
        WRITE(6,900)
  900   FORMAT(1H ,'****KBAND MICROPROCESSOR DEMO SYSTEM')
C**** WRITE QUERY SYMBOL
  10    WRITE(6,901)
  901   FORMAT(1H ,'$',/)
C**** READ USER REPLY INTO INLINE
        READ(6,902)INLINE
  902   FORMAT(20A1)
C**** INTERPRATE REPLY-RETURN NUMERIC IDENTIFIER
        CALL PARMID(IND)
        IF(IND.GT.0)GO TO 20
        WRITE(6,904)
  904   FORMAT(1H ,'***UNDEFINED PARAMETER')
        GO TO 10
  20    IF(IND.GT.1)GO TO 40
C**** DISPLAY CURRENT SET OF SYSTEM PARAMETERS
        CALL SHOW
        GO TO 10
  40    IF(IND.GT.2)GO TO 60
C**** CHECK SYSTEM PARAMETERS,LOAD INTELL,AND GO
        CALL INTELL(IERR)
        IF(IERR.GT.0)GO TO 10
        WRITE(6,920)
  920   FORMAT(1H ,'****SIMULATION UNDERWAY')
        GO TO 1000
  60    IF(IND.GT.3)GO TO 80
C**** GET BASE FREQUENCY FROM USER
        CALL FREQ(IERR)
        IF(IERR.EQ.0)GO TO 10
```

```
        WRITE(6,906)
  906   FORMAT(1H ,'***INVALID FREQUENCY')
        GO TO 10
  80    IF(IND.GT.4)GO TO 100
C****   BUILD MOP TABLE FROM USER FREQS
        CALL MOPLD
        GO TO 10
  100   IF(IND.GT.5)GO TO 120
C****   BUILD MFSK TABLE FROM USER FREQS
        CALL MFSKLD
        GO TO 10
  120   IF(IND.GT.6)GO TO 140
C****   GENERATE DOPPLER TABLE FROM BASE AND USER OFFSET
        CALL DOPLD
        GO TO 10
  140   IF(IND.GT.9)GO TO 160
C****   GET EFFECT START TIMES IN MINUTES FROM USER
        TYPE='STAR'
  150   CONTINUE
        CALL IBIE(IND,TYPE,IERR)
  155   IF(IERR.EQ.0)GO TO 10
        WRITE(6,908)
  908   FORMAT(1H ,'***TIME IS INVALID')
        GO TO 10
  160   IF(IND.GT.12)GO TO 180
C****   GET EFFECT END TIMES IN MINS FROM USER
        TYPE='STOP'
        GO TO 150
  180   IF(IND.GT.15)GO TO 200
C****   GET EFFECT CYCLE RATES FROM USER IN #CYCLES
        CALL STEPS(IND,IERR)
        IF(IERR.GT.0)WRITE(6,910)
  910   FORMAT(1H ,'***CYCLE RATE IS INVALID')
        GO TO 10
  200   IF(IND.GT.16)GO TO 1000
C****   GET SIM DURATION OR END IN MINUTES
        CALL ENDSIM(IERR)
        IF(IERR.GT.0)GO TO 155
        GO TO 10
C****   DISPLAY SYSTEM PARAMETERS BEFORE EXITING
 1000   CALL SHOW
        END

ROUTINES CALLED:
PARMID, SHOW  , INTELL, FREQ  , MOPLD , MFSKLD. DOPLD
IBIE  , STEPS , ENDSIM

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
MAIN.   489     (001722)*
SYSTEM  29      (000072)
TABLES  3040    (013700)
INBUF   10      (000024)
SPCDEC  11      (000026)
```

```
**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES 00827 02139
EXECUTABLES  01023 01943
ASSEMBLY     01485 06121
```

BLOCK DATA

The purpose of this subprogram is to initialize the labeled common areas of the static mode module prior to its execution. A listing and description of each area follows:

1. RTMTB - a 31 word integer array that initializes the system parameter table used in the real-time mode. The real-time pattern masks, random number seeds, and pattern table addresses are non zero.

2. MAXMIN - a 6 element double precision array used to store the HOP, MFSK, and DOP maximum and minimum values.

3. TABLES - a 6080 byte area used for pattern value storage in packed decimal form. This area is initialized to zero.

4. SYSTEM - a 29 word integer area used for the BCD intermediate storage of effect start, stop, cycle values, base frequency value, and simulation end. This area is initialized to zero.

5. SPCDEC - a 22 byte area used to store the ASCII character representations for those characters used in operator input processing.

```
FORTRAN V05.13                00:26:45     21-JAN-76    PAGE    1

      BLOCK DATA
C***********************************************************************
C
C     SYSTEM TABLES,PATTERN TABLES,STATIC MODE WORK AREAS AND CONSTANTS
C
C     RTMTAB IS A 62 BYTE AREA CONTAINING THE REAL TIME PARAMETER
C     TABLES USED BY THE INTELL,THE CONSTANT VALUES SUCH AS TABLE
C     ADDRESSES,RANDOM NUMBER SEEDS,AND PATTERN MASKS ARE INITIALIZED.
C     SYSTEM IS A 29 WORD ARRAY USED TO STORE THE REAL TIME PARAMETERS
C     AS THEY ARE INPUT FROM THE USER AND CONVERTED TO BCD FORM.
C     MAXMIN IS A 12 WORD AREA USED TO STORE EFFECT MAX AND MIN VALS
C     IN DOUBLE PRECISION FORM.
C     TABLES IS A 6080 BYTE AREA USED TO STORE THE RT PATTERN VALUES
C     IN THE COMPRESSED BCD FORM USED BY THE INTELL.
C     SPCDEC IS AN ELEVEN WORD AREA USED TO STORE CONSTANTS USED
C     THROUGHOUT THE PROGRAM.
C
C***********************************************************************
      COMMON/RTMTAB/TAB4
      INTEGER TAB4(31)
      COMMON/SYSTEM/SYS
      COMMON/MAXMIN/RMAX,RMIN
      INTEGER SYS(29)
      DOUBLE PRECISION RMAX(3),RMIN(3)
      COMMON/TABLES/HOP,DOP,MFSK
      BYTE CHARS(20)
      REAL ICHARS(5)
      BYTE HOP(640),DOP(5120),MFSK(320)
      EQUIVALENCE (CHARS(1),ICHARS(1))
      COMMON/SPCDEC/ICHARS,ICAR
      DATA RMAX/0.0D0,0.0D0,0.0D0/
      DATA RMIN/0.0D0,0.0D0,0.0D0/
      DATA ICHARS/'09+-','.,  ','0123','4567','8900'/
      DATA SYS/29*0/
      DATA ICAR/013/
      DATA TAB4/8*0,2*0,Z00FF,Z00FF,Z1A04,4*0,ZFF00,0,Z04FF,Z001D,
     1 6*0,Z0604,3*0/
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
DATA.    0       (000000)*
RTMTAB  31       (000076)
SYSTEM  29       (000072)
MAXMIN  24       (000060)
TABLES  3040     (013700)
SPCDEC  11       (000026)

**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES  00622 02344
EXECUTABLES   00542 02124
ASSEMBLY      00870 06736
```

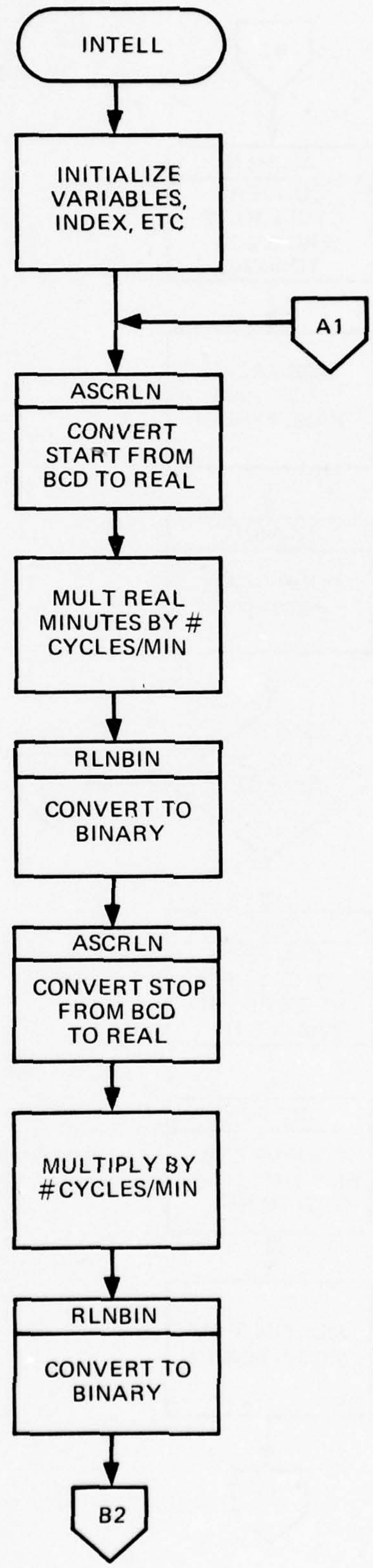
SUBROUTINE INTELL

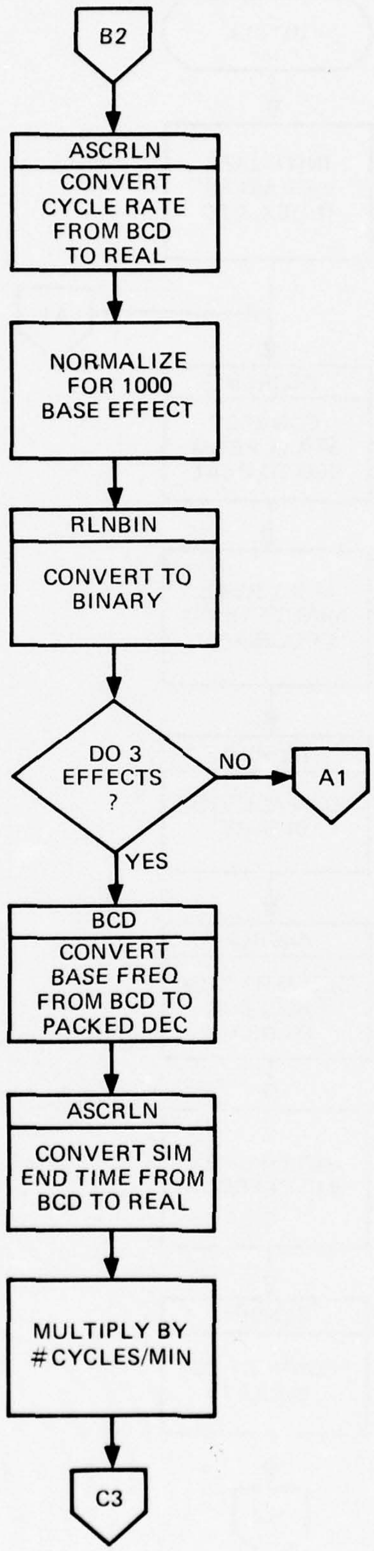
The primary purpose of this routine is to manage the conversion and transmission of the system parameter values and pattern value tables to the Intellec microprocessor. This routine also checks the consistency of the pattern value tables to ensure that a frequency less than 0 or greater than 2 megahertz cannot be generated.

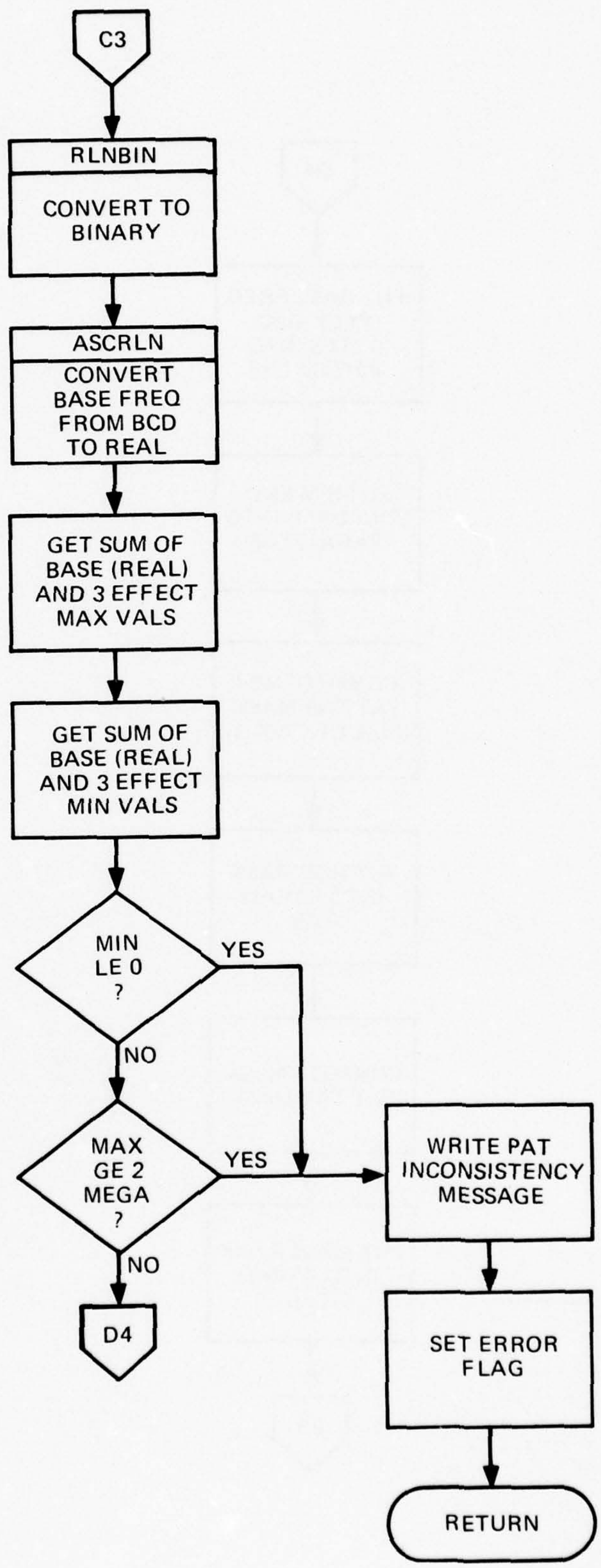
The simulation end and the three effect start, stop and cycle values are first converted from BCD to real number values using the ASCRLN subroutine. Simulation end and effect start/stop values which are specified in minutes are then converted to number of cycles per minute. The values are then converted to the binary form of the real-time mode with the RLNBIN subroutine.

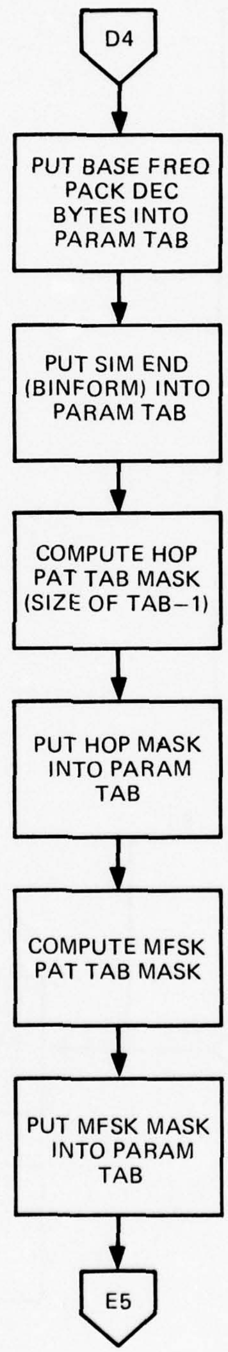
After conversion, the simulation end, effect start, stop, cycle values, and base frequency value in packed decimal form are inserted into the "TAB" array. The "TAB" array is a 61 byte table that is initialized in BLOCK DATA with the system parameter elements that are constant; i.e., pattern mask, running sum, addresses etc. Data transmission to the Intellec is done in the following order using the COMMO subroutine:

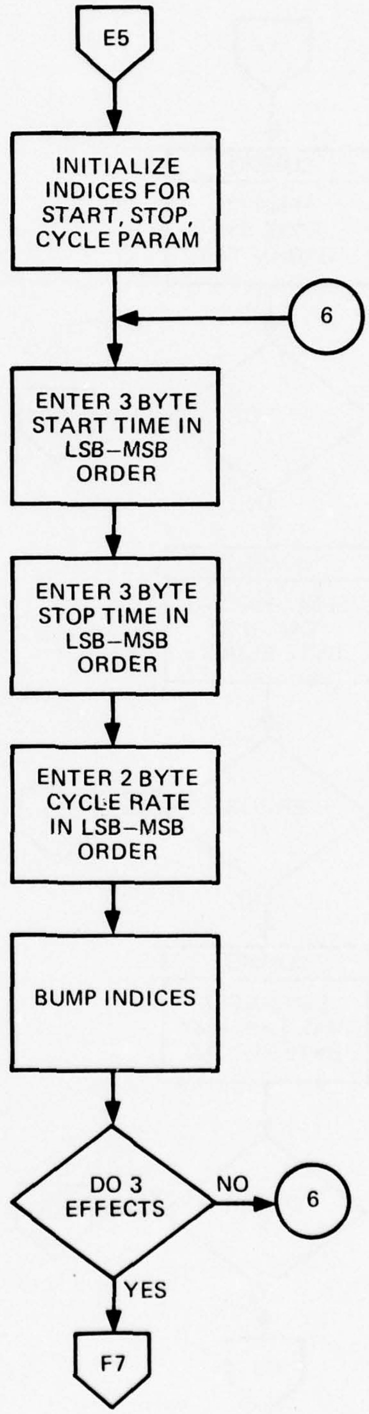
1. System parameter table - 61 bytes starting at location 20 HEX.
2. HOP pattern value table - 640 bytes transmitted in 32 byte blocks beginning at location 1A00HEX.
3. MFSK pattern value table - 320 bytes transmitted in 32 byte blocks beginning at location 1D00HEX.
4. DOP pattern value table - 5120 bytes transmitted in 32 byte blocks beginning at location 600 HEX.

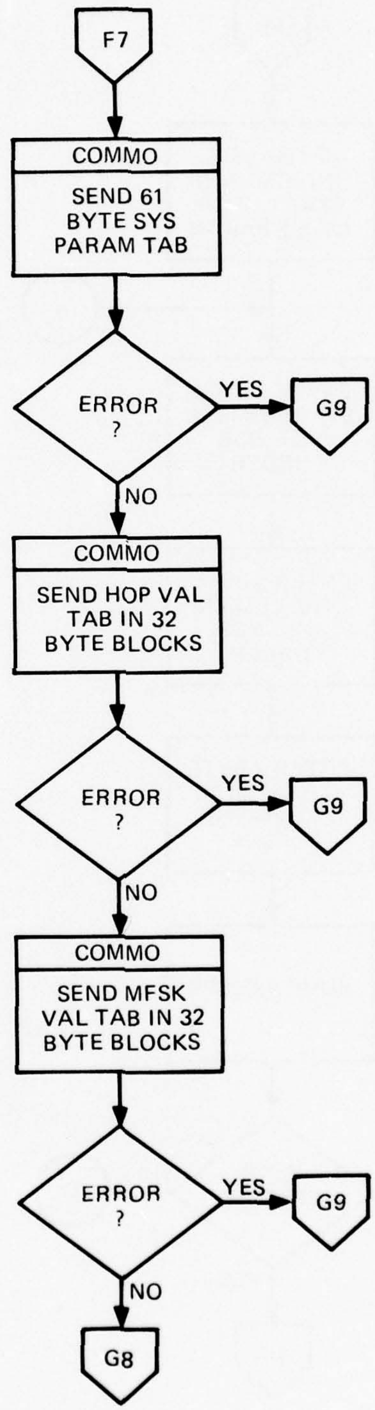

INTELL
INITIALIZE VARIABLES, INDEX, ETC
A1
ASCRLN
CONVERT START FROM BCD TO REAL
MULT REAL MINUTES BY # CYCLES/MIN
RLNBIN
CONVERT TO BINARY
ASCRLN
CONVERT STOP FROM BCD TO REAL
MULTIPLY BY #CYCLES/MIN
RLNBIN
CONVERT TO BINARY
B2

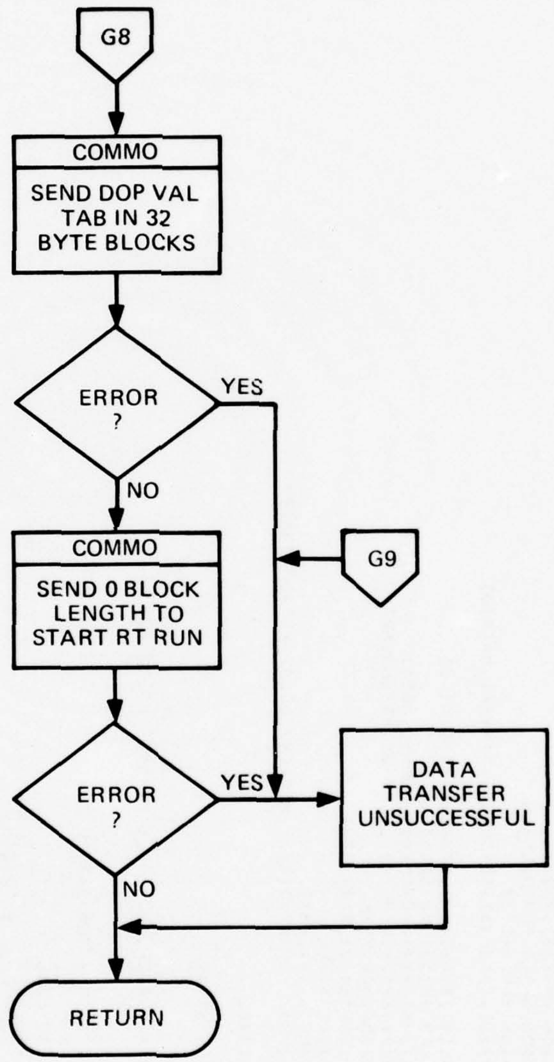

B2
ASCRLN
CONVERT CYCLE RATE FROM BCD TO REAL
NORMALIZE FOR 1000 BASE EFFECT
RLNBIN
CONVERT TO BINARY
DO 3 EFFECTS ?
NO
A1
YES
BCD
CONVERT BASE FREQ FROM BCD TO PACKED DEC
ASCRLN
CONVERT SIM END TIME FROM BCD TO REAL
MULTIPLY BY #CYCLES/MIN
C3


C3
RLNBIN
CONVERT TO BINARY
ASCRLN
CONVERT BASE FREQ FROM BCD TO REAL
GET SUM OF BASE (REAL) AND 3 EFFECT MAX VALS
GET SUM OF BASE (REAL) AND 3 EFFECT MIN VALS
MIN LE 0 ?
YES
NO
MAX GE 2 MEGA ?
YES
NO
WRITE PAT INCONSISTENCY MESSAGE
SET ERROR FLAG
RETURN
D4


D4
PUT BASE FREQ PACK DEC BYTES INTO PARAM TAB
PUT SIM END (BINFORM) INTO PARAM TAB
COMPUTE HOP PAT TAB MASK (SIZE OF TAB−1)
PUT HOP MASK INTO PARAM TAB
COMPUTE MFSK PAT TAB MASK
PUT MFSK MASK INTO PARAM TAB
E5


E5
INITIALIZE INDICES FOR START, STOP, CYCLE PARAM
6
ENTER 3 BYTE START TIME IN LSB–MSB ORDER
ENTER 3 BYTE STOP TIME IN LSB–MSB ORDER
ENTER 2 BYTE CYCLE RATE IN LSB–MSB ORDER
BUMP INDICES
DO 3 EFFECTS
NO
6
YES
F7


F7
COMMO
SEND 61 BYTE SYS PARAM TAB
ERROR ?
YES
G9
NO
COMMO
SEND HOP VAL TAB IN 32 BYTE BLOCKS
ERROR ?
YES
G9
NO
COMMO
SEND MFSK VAL TAB IN 32 BYTE BLOCKS
ERROR ?
YES
G9
NO
G8


G8
COMMO
SEND DOP VAL TAB IN 32 BYTE BLOCKS
ERROR ?
YES
NO
COMMO
SEND 0 BLOCK LENGTH TO START RT RUN
G9
ERROR ?
YES
NO
DATA TRANSFER UNSUCCESSFUL
RETURN

```
      SUBROUTINE INTELL(IERR)
C*************************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO CONVERT THE REAL TIME
C**** PARAMETERS INTO THE BINARY INTELL FORM,CHECK THE CONSISTENCY
C**** OF THE PATTERN TABLES,AND TO LOAD THE INTELL.
C
C*************************************************************************
      INTEGER SYSADR,HOPADR,MFSADR,DOPADR,ADR
      BYTE DAT(32)
      COMMON/RTSTAB/TAB
      COMMON/MAXMIN/RMAX,RMIN
      DOUBLE PRECISION RMAX(3),RMIN(3)
      DOUBLE PRECISION SFREQ,HFREQ,LFREQ
      DOUBLE PRECISION RLN,FACTOR
      INTEGER ISTART(6),ISTOP(6),ICYCLE(6),IEND(2),MASK(3)
      BYTE TAB(32)
      INTEGER SIMEND(4),START(12),STOP(12),CYCLE(12)
      BYTE IVAL(12),PBASE(12),IBASE(5)
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND,ASCII
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
      COMMON/TABLES/HOP,DOP,MFSK
      BYTE HOP(540),DOP(5120),MFSK(320)
      DOUBLE PRECISION  SCALE
      EQUIVALENCE (ASCII,IVAL(5))
      DATA SYSADR/Z0020/,HOPADR/Z1A00/,MFSADR/Z1D00/,DOPADR/Z0600/
      SCALE=26666.
      FACTOR=SCALE/1000.
      DO 5 I=1,12
C**** ZERO BCD STRING ARRAY
      IVAL(I)=0
 5    CONTINUE
C**** CONVERT THE START,STOP,AND CYCLE PARAMETERS TO BIN
      J=1
      DO 10 I=1,3
C**** PUT START INFO INTO BYTES 5-8 OF IVAL
      ASCII=PSTART(I)
C**** CONVERT START TO REAL
      CALL ASCRLN(IVAL,RLN)
C**** CONVERT START FROM REAL TO BINARY
      RLN=FACTOR*RLN
      CALL RLNBIN(RLN,START(J))
C**** PUT STOP INTO BYTES 5-8 OF IVAL
      ASCII=PSTOP(I)
C**** CONVERT STOP TO REAL NUMBER
      CALL ASCRLN(IVAL,RLN)
C**** CONVERT STOP FROM REAL TO BINARY
      RLN=FACTOR*RLN
      CALL RLNBIN(RLN,STOP(J))
C**** PUT CYCLE INTO BYTES 5-8 OF IVAL
      ASCII=PCYCLE(I)
C**** CONVERT CYCLE RATE FROM BCD TO REAL
      CALL ASCRLN(IVAL,RLN)
C**** CONVERT CYCLE FROM REAL TO BINARY
      RLN=RLN/1000.
```

```
      CALL RLNBIN(RLN,CYCLE(J))
C**** BUMP TO NEXT SET OF 4 CYCLE,STOP,START WORDS
      J=J+4
 10   CONTINUE
C**** CONVERT BASE FREQ FROM BCD STRING TO INTELL FIRM
      CALL BCD(PBASE,IBASE)
C**** PUT SIM END TIME INTO BYTES 5-8 OF IVAL
      ASCII=PSMEND
C**** CONVERT SIM STOP TIME TO REAL NUMBER
      CALL ASCRLN(IVAL,RLN)
C**** CONVERT IT TO BINARY
      RLN=FACTOR*RLN
      CALL RLNBIN(RLN,SIMEND)
C**** CONVERT BASE FREQ TO RLN FOR PATTERN LIMIT CHECK
      CALL ASCRLN(PBASE,SFREQ)
      HFREQ=SFREQ
      LFREQ=SFREQ
C**** CHECK IF A PATTERN CAN BE GENERATED IN INTELL THAT IS
C**** OUT OF LIMITS(0-2000000000)
      DO 50 I=1,3
      HFREQ=HFREQ+RMAX(I)
      LFREQ=LFREQ+RMIN(I)
 50   CONTINUE
      IF(LFREQ.LT.0)GO TO 800
      IF(HFREQ.GE.2000000000.)GO TO 800
C**** BUILD TABLE TO BE SENT TO INTELL
      DO 20 I=1,5
C**** BASE FREQUENCY
      TAB(3+I)=IBASE(I)
 20   CONTINUE
C**** DO SIMULATION END
      DO 30 I=2,4
      TAB(18-I)=SIMEND(I)
 30    CONTINUE
C**** DETERMINE MASK VALUES FOR RAN NUMB GENERATOR
      KB=MASK(1)-1
      TAB(20)=KB
      KB=MASK(2)-1
      TAB(35)=KB
C**** POINTER TO SET OF 3 START BYTES PER EFFECT
      J=17
C**** POINTER TO SET OF 3 STOP BYTES PER EFFECT
      K=29
C**** POINTER TO SET OF TWO CYCLE BYTES PER EFFECT
      L=27
C**** ENTER EFFECT START,STOP, AND CYCLE VALUES INTO SYS ARRAY
      DO 40 I=1,9,4
C**** EFFECT START TIMES
      TAB(J)=START(I+3)
      TAB(J+1)=START(I+2)
      TAB(J+2)=START(I+1)
C**** EFFECT END TIMES
      TAB(K)=STOP(I+3)
      TAB(K+1)=STOP(I+2)
      TAB(K+2)=STOP(I+1)
C**** EFFECT CYCLE RATES
```

```
      TAB(L)=CYCLE(I+3)
      TAB(L+1)=CYCLE(I+2)
C**** BUMP TO NEXT EFFECT START,STOP,CYCLE VALUE INPUT
      J=J+15
      K=K+15
      L=L+15
   40 CONTINUE
C**** NOW USE SUB COMMO TO SEND DATA TO INTELL
C**** FIRST SEND SYSTEM PARAMETER TABLE
      CALL COMMO(SYSADR,61,TAB,IERR)
      IF(IERR.GT.0)GO TO 650
C**** NOW SEND HOP TABLE
      ADR=HOPADR
C**** GET ADR OF LOCATION FOR 32 BYTE BLOCK
      DO 120 I=1,609,32
C**** LOAD UP 32 BYTE BLOCK
      DO 110 J=1,32
      DAT(J)=HOP(I+J-1)
  110 CONTINUE
C**** SEND BLOCK OF 32 BYTES
      CALL COMMO(ADR,32,DAT,IERR)
      IF(IERR.GT.0)GO TO 650
C**** BUMP TO NEXT BLOCK ADR
      ADR=ADR+32
  120 CONTINUE
C**** SEND MFSK TABLE TO INTELL
      ADR=MFSADR
      DO 140 I=1,289,32
C**** LOAD UP 32 BYTE BLOCK
      DO 130 J=1,32
      DAT(J)=MFSK(I+J-1)
  130 CONTINUE
C**** SEND BLOCK OF 32 MFSK BYTES TO INTELL
      CALL COMMO(ADR,32,DAT,IERR)
      IF(IERR.GT.0)GO TO 650
C**** BUMP TO NEXT MFSK TABLE ADDRESS
      ADR=ADR+32
  140 CONTINUE
C**** SEND DOPPLER TABLE TO INTELL
      ADR=DOPADR
      DO 160 I=1,5089,32
C**** LOAD UP 32 BYTE BLOCK
      DO 150 J=1,32
      DAT(J)=DOP(I+J-1)
  150 CONTINUE
C**** SEND BLOCK OF 32 BYTES
      CALL COMMO(ADR,32,DAT,IERR)
      IF(IERR.GT.0)GO TO 650
C**** BUMP TO NEXT DOP ADR IN INTELL
      ADR=ADR+32
  160 CONTINUE
C**** SEND 0 BYTE COUNT TO SIGNIFY LOADING COMPLETE
      CALL COMMO(ADR,0,DAT(1),IERR)
      IF(IERR.EQ.0)RETURN
C**** FREQ GENERATED NOT IN 0-2 MEGA RANGE
  650 WRITE(6,1020)IERR
```

```
FORTRAN V06.15                    00:07:00     21-JAN-76    PAGE    4

        1020 FORMAT(1H ,'DATA TRANSFER TO INTELL NOT SUCCESSFUL-ERROR=',I5)
             RETURN
        900  CONTINUE
        800  WRITE(6,802)
        802  FORMAT(1H ,'PATTERN TABLE INCONSISTENCY')
             IERR=1
             RETURN
             END

       ROUTINES CALLED:
       ASCHLN, RLNBIN, BCD    , COMMO

       OPTIONS =/ON,/SU,/OP:1

       BLOCK        LENGTH
       INTELL  854     (003254)*
       RTMTAB  31      (000076)
       MAXMIN  24      (000060)
       SYSTEM  29      (000072)
       TABLES  3040    (013700)

       **COMPILER ----- CORE**
          PHASE        USED  FREE
       DECLARATIVES 00522 02344
       EXECUTABLES  01397 01569
       ASSEMBLY     01721 05885
```

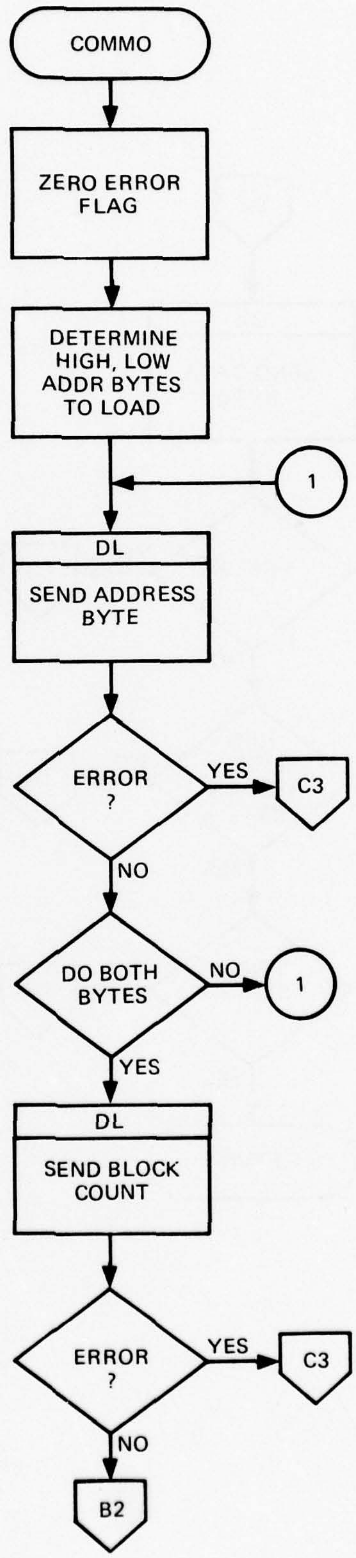
SUBROUTINE COMMO

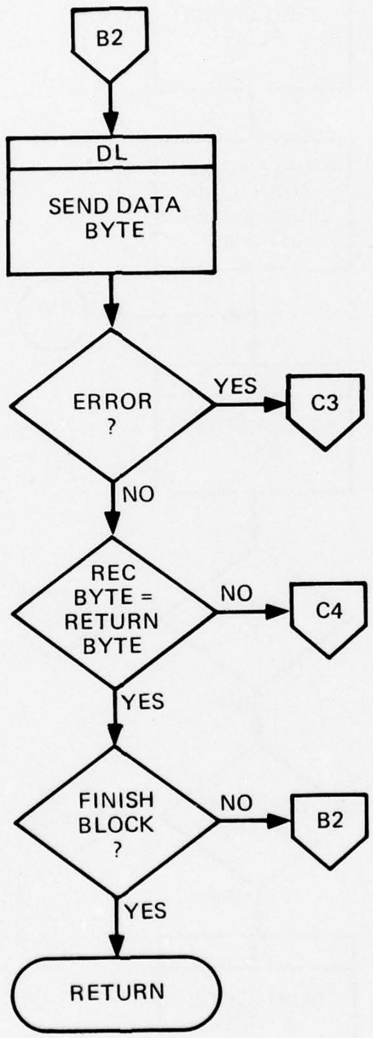
The purpose of this routine is to transmit a block of data via the DL11 interface and the DL subroutine to the Intellec microprocessor.

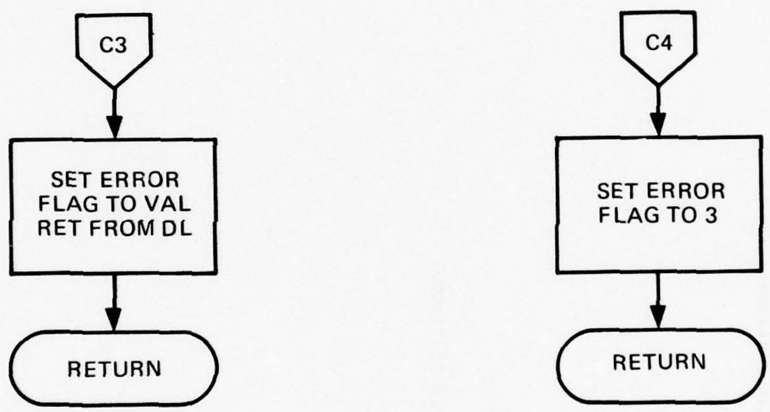
The address of the memory location to load, the number of bytes in the block, and the data block are passed to the routine. The routine then passes this information via the DL subroutine to the Intellec in the following order.

1. High byte address.
2. Low byte address.
3. Number of bytes in the data block.
4. The data bytes one at a time.

After each transfer, the error flag returned from the DL subroutine is checked for possible errors. If an error occurs, the routine returns the value of the DL subroutines error flag to the caller. The byte passed to the DL subroutine is also compared with the byte echoed back from the Intellec after each byte transfer to verify correct reception. An error value of three is returned from this routine if a verification error occurs.


COMMO
ZERO ERROR FLAG
DETERMINE HIGH, LOW ADDR BYTES TO LOAD
1
DL
SEND ADDRESS BYTE
ERROR ?
YES
C3
NO
DO BOTH BYTES
NO
1
YES
DL
SEND BLOCK COUNT
ERROR ?
YES
C3
NO
B2


B2
DL
SEND DATA BYTE
ERROR ?
YES
C3
NO
REC BYTE = RETURN BYTE
NO
C4
YES
FINISH BLOCK ?
NO
B2
YES
RETURN


C3
SET ERROR FLAG TO VAL RET FROM DL
RETURN
C4
SET ERROR FLAG TO 3
RETURN

```
FORTRAN V06.13                 00:07:57     21-JAN-76    PAGE    1

      SUBROUTINE COMMO(IADR,NBYTS,DAT,IERR)
C***********************************************************************
C
C     THE PURPOSE OF THIS ROUTINE IS TO TRANSMIT A BLOCK OF DATA TO
C     THE INTELL VIA THE DL11 INTERFACE.
C     IADR IS THE INTELL ADR,NBYTS IS THE NUMBER OF BYTES IN THE
C     BLOCK TO LOAD,AND DAT IS AN ARRAY CONTAINING THE DATA TO SEND.
C
C***********************************************************************
      BYTE ERRATA(1)
      BYTE ADR(2),DAT(5200)
      INTEGER SEND,REC,NUMB
C**** ZERO ERROR FLAG
      IERR=0
C**** DETERMINE ADDRESS TO LOAD IN INTELL
      I=IADR/255
      ADR(1)=I
      I=IADR-(I*256)
      ADR(2)=I
C**** TELL THE INTELL WHAT ADR DATA WILL GO TO
      DO 10 I=1,2
      ERRATA(1)=ADR(I)
      SEND=ERRATA(1)
      REC=0
      CALL DL(SEND,REC,JERR)
C**** CHECK FOR ERROR-CHARACTER NOT RECEIVED PROPERLY OR TIMEOUT
      IF(JERR.GT.0)GO TO 60
      IF(SEND.NE.REC)GO TO 70
 10   CONTINUE
C**** TELL INTELL HOW MANY BYTES WILL BE TRANFERED
      REC=0
      CALL DL(NBYTS,REC,JERR)
      IF(JERR.GT.0)GO TO 60
      IF(NBYTS.NE.REC)GO TO 70
C**** SEND BLOCK OF DATA TO INTELL
      DO 50 I=1,NBYTS
      ERRATA(1)=DAT(I)
      SEND=ERRATA(1)
      REC=0
      CALL DL(SEND,REC,JERR)
      IF(JERR.GT.0)GO TO 60
      IF(SEND.NE.REC)GO TO 70
 50   CONTINUE
      RETURN
C**** TIMEOUT EROR TRYING TO TRANSMIT OR RECEIVE
 60   IERR=JERR
      RETURN
C**** CHAR REC BACK FROM INTELL NOT WHAT SENT
 70   IERR=3
      RETURN
      END

ROUTINES CALLED:
DL

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
```

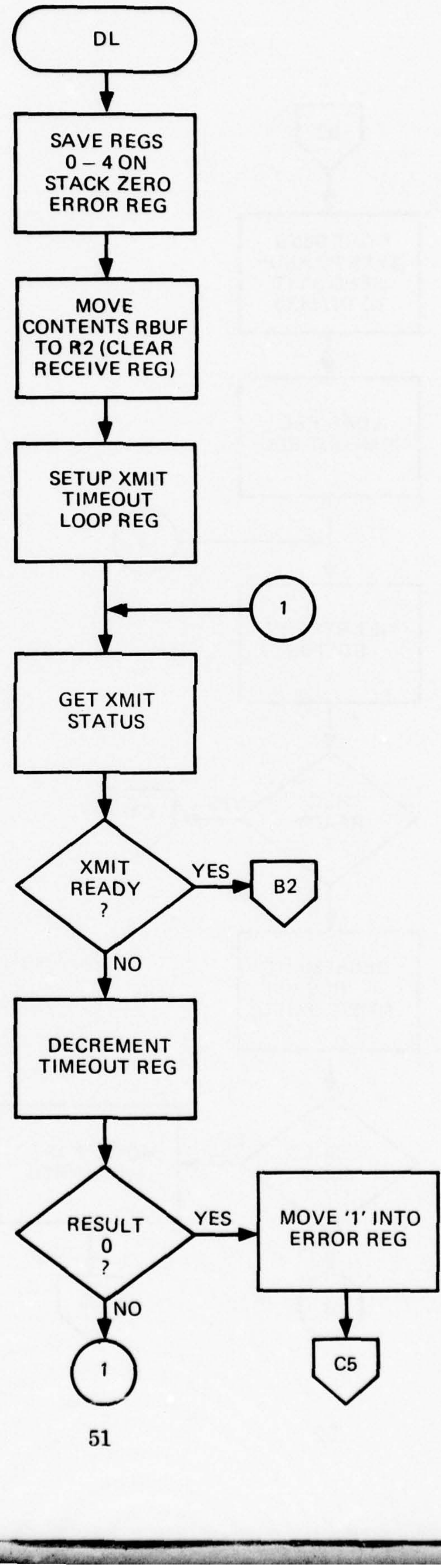
```
FORTRAN V06.13                 00:07:57    21-JAN-78   PAGE    2

        COMMO   215     (000656)*

        **COMPILER ----- CORE**
           PHASE        USED  FREE
        DECLARATIVES 00622 02344
        EXECUTABLES  00783 02183
        ASSEMBLY     01481 06525
```

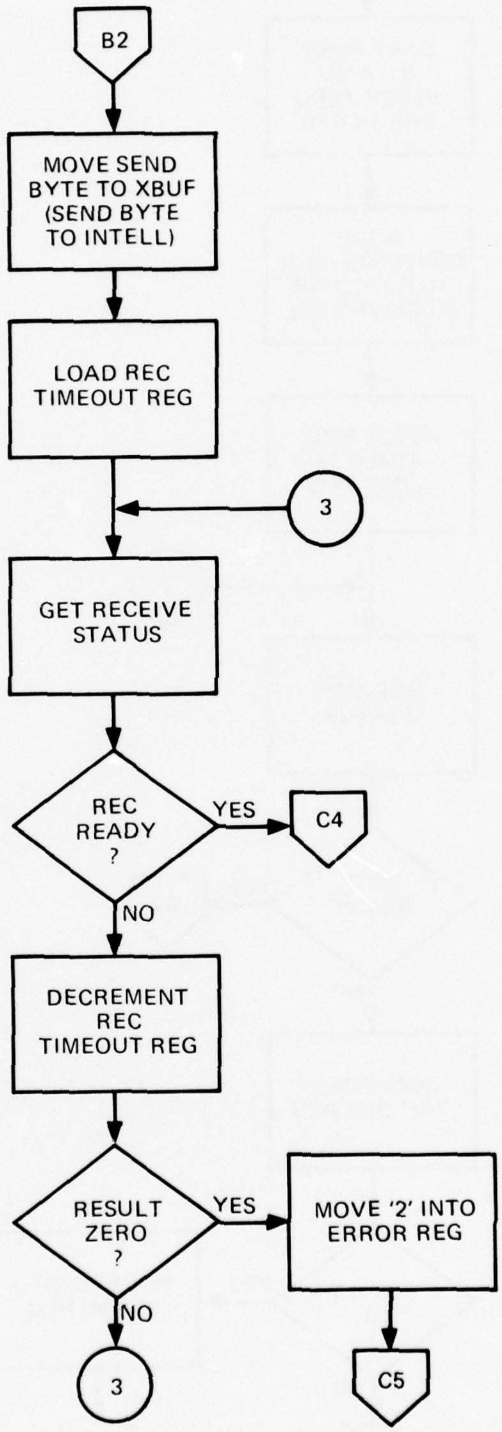
SUBROUTINE DL

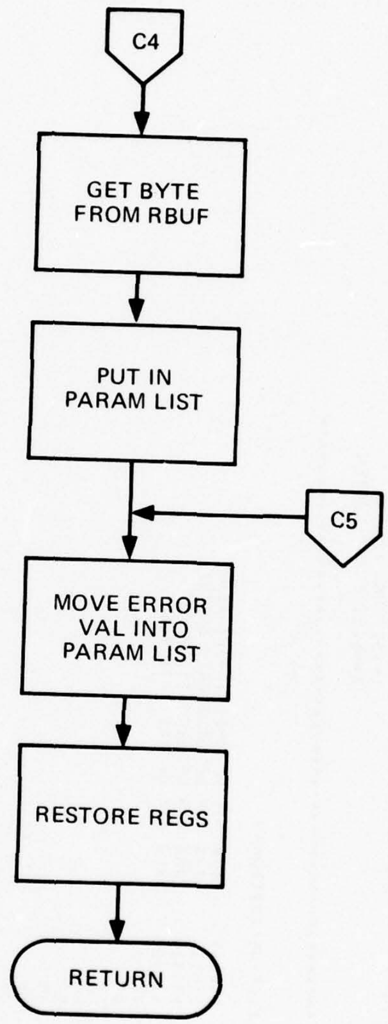
This routine is used to transmit and receive byte data to and from the Intellec via the DL11 interface. A byte to be transmitted is passed to the routine in its parameter list. After transmission of the byte to the Intellec, the routine loops checking the receive status for indication that the byte sent has been echoed back. Once the receive indicator in RCSR becomes true, the echoed byte is moved from RBUF to the callers argument list.

The routine also returns an error flag to the caller that is set as follows:

1. 0 = no errors
2. 1 = XMIT ready not received
3. 2 = REC ready not received


DL
SAVE REGS 0 – 4 ON STACK ZERO ERROR REG
MOVE CONTENTS RBUF TO R2 (CLEAR RECEIVE REG)
SETUP XMIT TIMEOUT LOOP REG
1
GET XMIT STATUS
XMIT READY ?
YES
B2
NO
DECREMENT TIMEOUT REG
RESULT 0 ?
YES
MOVE '1' INTO ERROR REG
NO
1
C5


B2
MOVE SEND BYTE TO XBUF (SEND BYTE TO INTELL)
LOAD REC TIMEOUT REG
3
GET RECEIVE STATUS
REC READY ?
YES
C4
NO
DECREMENT REC TIMEOUT REG
RESULT ZERO ?
YES
MOVE '2' INTO ERROR REG
NO
3
C5


C4
GET BYTE FROM RBUF
PUT IN PARAM LIST
C5
MOVE ERROR VAL INTO PARAM LIST
RESTORE REGS
RETURN

```
DL                          MACRO V06-3M 16-JAN-76 00:13 PAGE 1

                                    .TITLE    DL
                        ;*******************************************************************
                                    .GLOBL    DL
                        ;
                        ; DL11 OUTPUT SUBROUTINE
                        ;
                        ; THE PURPOSE OF THIS ROUTINE IS TO TRANSMIT A BYTE OF DATA VIA THE
                        ; DL 11 INTERFACE TO THE INTELL AND TO RECEIVE THE BYTE AFTER IT IS
                        ; ECHOED BACK BY THE INTELL.
                        ;
                        ; CALLING SEQUENCE:
                        ;           JSR       DL11
                        ;           BR        CONT
                        ;           .WORD     OUT         ;BYTE OUTPUT TO INTELL
                        ;           .WORD     BACK        ;BYTE RETURNED TO 11 FROM INTELL
                        ;           .WORD     FLAG        ;COMPLETION FLAG: 0=NO ERRORS
                        ;                                                   1=XMIT NOT READY
                        ;                                                   2=RECEIVE NOT GOTTEN
                        ;
                        ;*******************************************************************
                        ;
                        ;
                        ;**** EQUATES REGISTER DEFINITIONS
                        ;
              176514                XCSR=     176514      ;ABS ADR OF XMIT STATUS REG
              176516                XBUF=     176516      ;ABS ADR OF XMIT DATA BUFFER
              176510                RCSR=     176510      ;ABS ADR OF REC STATUS REG
              176512                RBUF=     176512      ;ABS ADR OF REC DATA REG
              000000                R0=       %0
              000001                R1=       %1
              000002                R2=       %2
              000003                R3=       %3
              000004                R4=       %4
              000005                R5=       %5
              000006                SP=       %6
              000007                PC=       %7
                        ;
                        ;
                        ;**** SAVE USER REGS
                        ;
        000000 010446   DL:         MOV       R4,-(SP)
        000002 010346               MOV       R3,-(SP)
        000004 010246               MOV       R2,-(SP)
        000006 010146               MOV       R1,-(SP)
        000010 010046               MOV       R0,-(SP)
        000012 012703               MOV       #0,R3       ;ZERO ERROR FLAG
               000000
                        ;
                        ;**** CLEAR REC BUFFER
                        ;
        000016 013702               MOV       @#RBUF,R2
               176512
                        ;
                        ;**** GET TRANSMIT OK
                        ;
        000022 012701               MOV       #777,R1     ;XMIT TIMEOUT REG
               000777
```

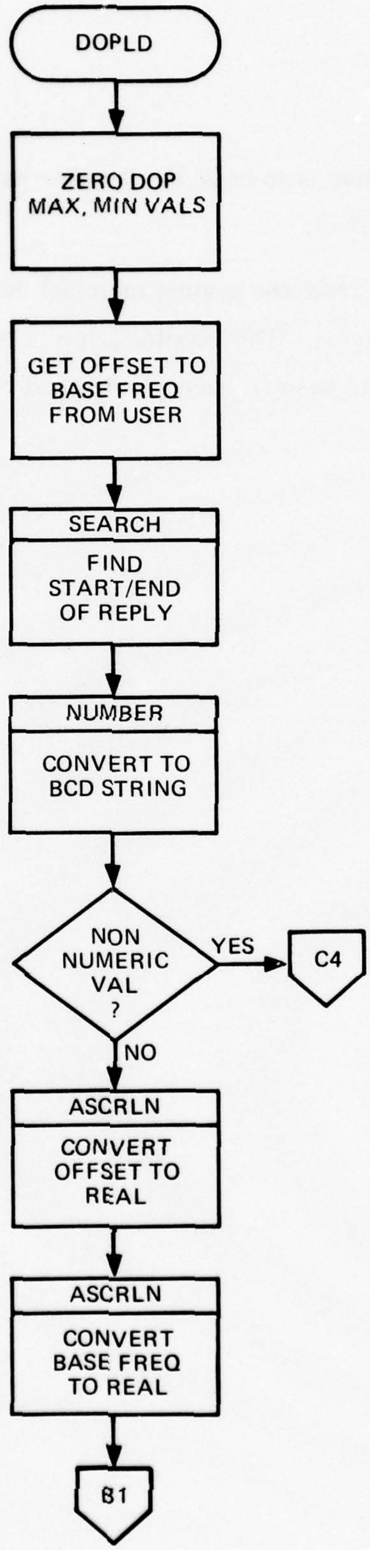
```
00026 013702 XSTAT:  MOV     @#XCSR,R2 ;GET XMIT STATUS
      176514
00032 032702         BIT     #200,R2   ;TEST BIT 7 FOR XMIT READY
      000200
00036 001006         BNE     XMIT      ;BIT 7=0,THEN OK TO SEND
00040 005301         DEC     R1
00042 001371         BNE     XSTAT     ; TRY AGAIN TO GET XMIT OK
00044 012703         MOV     #1,R3     ; XMIT NOT READY ERROR
      000001
00050 000167         JMP     REST
      000050
                     ;
                     ;**** SEND THE BYTE TO THE INTELL
                     ;
00054 117537 XMIT:   MOVB    @2(R5),@#XBUF
      000002
      176516
                     ;
                     ;**** GET BYTE RECEIVED INDICATION
                     ;
00062 012701         MOV     #7777,R1  ; ECHO TMEOUT REG
      007777
00066 013702 RSTAT:  MOV     @#RCSR,R2 ;GET REC STATUS
      176510
00072 032702         BIT     #200,R2   ;TEST BIT 7 FOR REC READY
      000200
00076 001006         BNE     REC       ; BIT 7=0,THEN GOT BYTE BACK
00100 005301         DEC     R1
00102 001371         BNE     RSTAT     ; AGAIN CHECK FOR BYTE FROM INTELL
00104 012703         MOV     #2,R3     ;REC TIMEOUT
      000002
00110 000167         JMP     REST
      000010
                     ;
                     ;**** GET BYTE ECHOED FROM INTELL
00114 113702 REC:    MOVB    @#RBUF,R2        ;SIGN EXTEND THE RECEIVED BYTE
      176512
00120 010275         MOV      R2 ,@4(R5)      ;PUT WORD IN PARAMETER LIST
      000004
                     ;
                     ;**** SET COMPLETION FLAG AND RESTORE REGS
                     ;
00124 010375 REST:   MOV     R3,@6(R5)
      000006
00130 012600         MOV     (SP)+,R0
00132 012601         MOV     (SP)+,R1
00134 012602         MOV     (SP)+,R2
00136 012603         MOV     (SP)+,R3
00140 012604         MOV     (SP)+,R4
00142 000205         RTS     R5        ; RETURN TO CALLER
      000001         .END
```

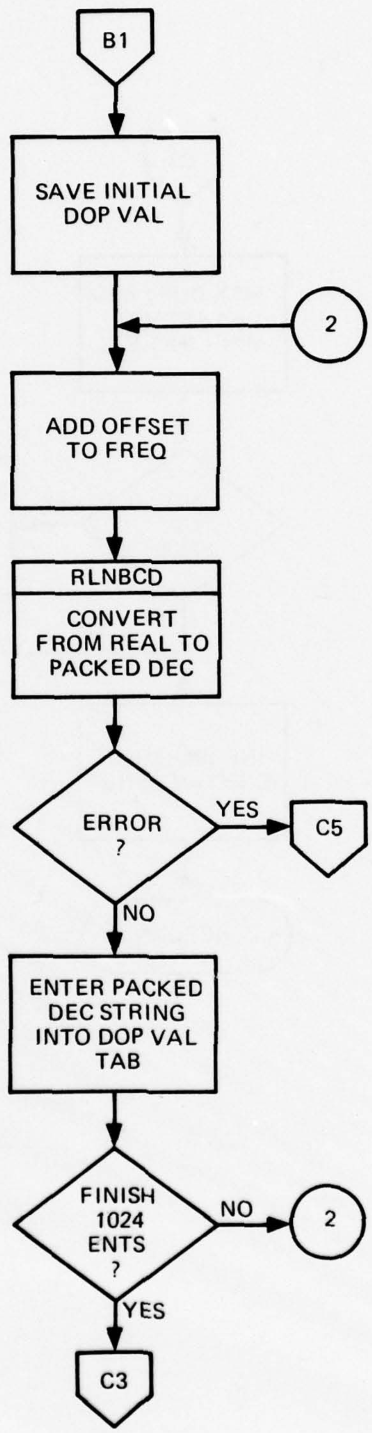
```
DL                                   MACRO V06-3M 16-JAN-76 00:13 PAGE 1-2
SYMBOL TABLE

DL          000000RG          RBUF   = 176512          RCSR   = 176510
REC         000114R           REST     000124R         RSTAT    000066R
XBUF   = 176516               XCSR   = 176514          XMIT     000054R
XSTAT       000026R
. ABS.      000000     000
            000144     001
ERRORS DETECTED:  0
FREE CORE:  4523. WORDS
PART4,OBJ,LP:<PART4,MAC
```

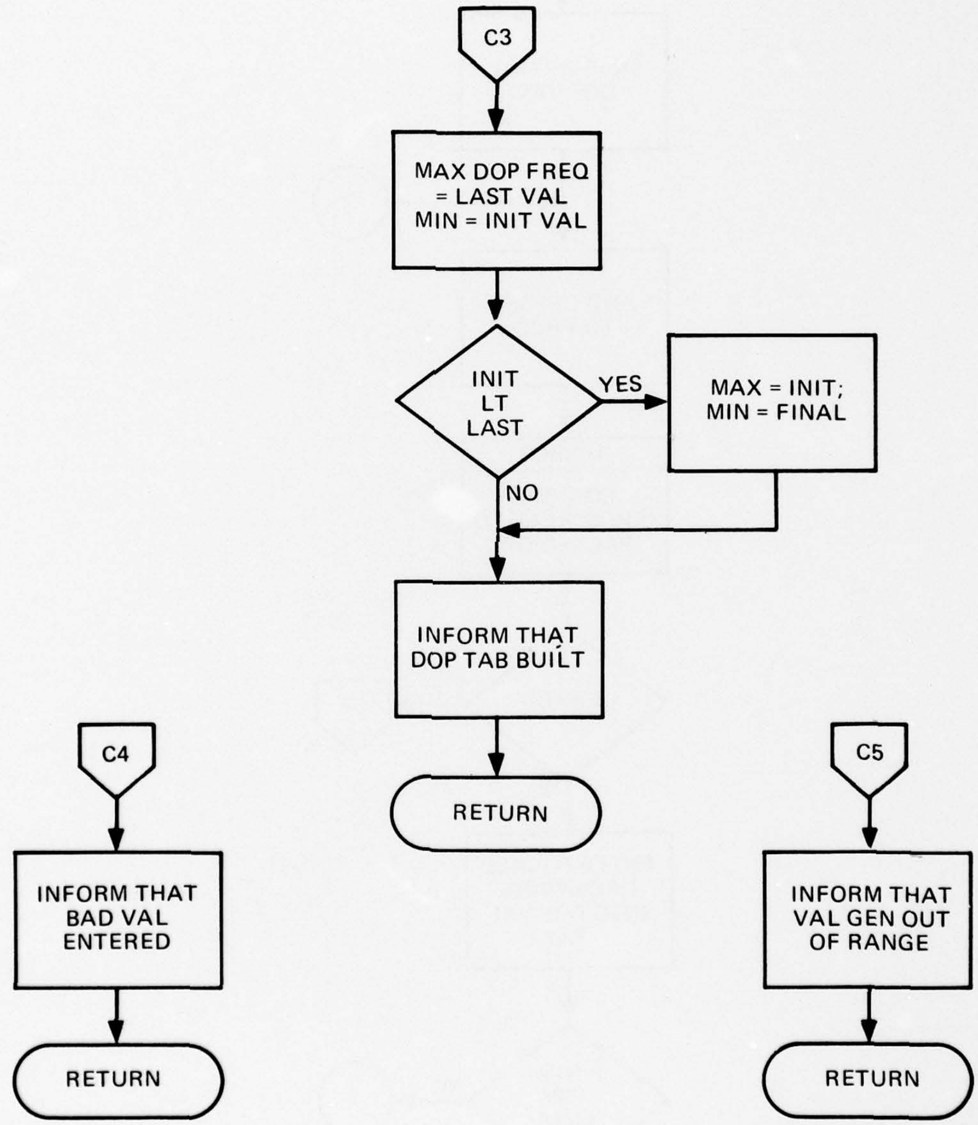
SUBROUTINE DOPLD

The purpose of this routine is to build the Doppler pattern value table from the base frequency value and an offset.

The offset value is read from the system terminal device, converted to BCD, and then converted to double precision. The Doppler table is then built by successively adding the offset value to be the base frequency value and converting the result to packed decimal form.


DOPLD
ZERO DOP MAX, MIN VALS
GET OFFSET TO BASE FREQ FROM USER
SEARCH
FIND START/END OF REPLY
NUMBER
CONVERT TO BCD STRING
NON NUMERIC VAL ?
YES
C4
NO
ASCRLN
CONVERT OFFSET TO REAL
ASCRLN
CONVERT BASE FREQ TO REAL
B1


B1
SAVE INITIAL DOP VAL
2
ADD OFFSET TO FREQ
RLNBCD
CONVERT FROM REAL TO PACKED DEC
ERROR ?
YES
C5
NO
ENTER PACKED DEC STRING INTO DOP VAL TAB
FINISH 1024 ENTS ?
NO
2
YES
C3


C3
MAX DOP FREQ = LAST VAL MIN = INIT VAL
INIT LT LAST
YES
MAX = INIT; MIN = FINAL
NO
INFORM THAT DOP TAB BUILT
RETURN
C4
INFORM THAT BAD VAL ENTERED
RETURN
C5
INFORM THAT VAL GEN OUT OF RANGE
RETURN

```
      SUBROUTINE DOPLD
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO BUILD THE DOPPLER
C**** PATTERN TABLES USED BY THE INTELL.THE USER INPUTS AN OFFSET
C**** TO THE BASE FREQ AND THE ROUTINE BUILDS A TABLE OF VALUES BY
C**** SUCCESSIVE ADDITIONS TO THE BASE FREQ.
C
C***********************************************************************
      BYTE WORK(5)
      COMMON/INBUF/INLINE
      COMMON/MAXMIN/RMAX,RMIN
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
      COMMON/TABLES/HOP,DOP,MFSK
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND
      INTEGER MASK(3)
      DOUBLE PRECISION FREQ,OFFSET
      DOUBLE PRECISION RMAX(3),RMIN(3)
      DOUBLE PRECISION SFREQ
      BYTE INLINE(20),IVAL(12),PBASE(12)
      BYTE HOP(540),DOP(5120),MFSK(320)
      RMIN(3)=0.0
      RMAX(3)=0.0
C**** QUERY USER FOR OFFSET FOR TABLE GENERATION
      WRITE(6,900)
900   FORMAT(1H ,'DOP OFFSET TO BASE FREQ = ',/)
C**** READ OPERATOR REPLY
      READ(6,902)INLINE
902   FORMAT(20A1)
C**** FIND REPLY START AND END
      CALL SEARCH(ISTART,IEND)
      IF(ISTART.EQ.0)RETURN
C**** CONVERT OFFSET TO BCD STRING
      CALL NUMBER(ISTART,IEND,IVAL,IERR)
      IF(IERR.GT.0)GO TO 200
C**** CONVERT BCD STRING TO REAL NUMBER
      CALL ASCRLN(IVAL,OFFSET)
C**** CONVERT BASE FREQ TO REAL NUMBER
      CALL ASCRLN(PBASE,FREQ)
      SFREQ=FREQ+OFFSET
      J=1
C**** GENERATE DOP TABLE BY SUCCESSIVE ADDITIONS OF OFFSET TO BASE
      DO 20 I=1,1024
      FREQ=FREQ+OFFSET
C**** CONVERT REAL NUMBER VAL OF FREQ TO 5 BYTE COMPRESSED FORM
      CALL RLNBCD(FREQ,WORK,IERR)
      IF(IERR.GT.0)GO TO 300
C**** PUT FREQ 5 BYTES INTO DOP TABLE
      DO 10 K=1,5
      DOP(J)=WORK(K)
      J=J+1
10    CONTINUE
20    CONTINUE
C**** CHECK FOR MAX/MIN INPUT FREQ THIS PATTERN
      RMAX(3)=FREQ
```

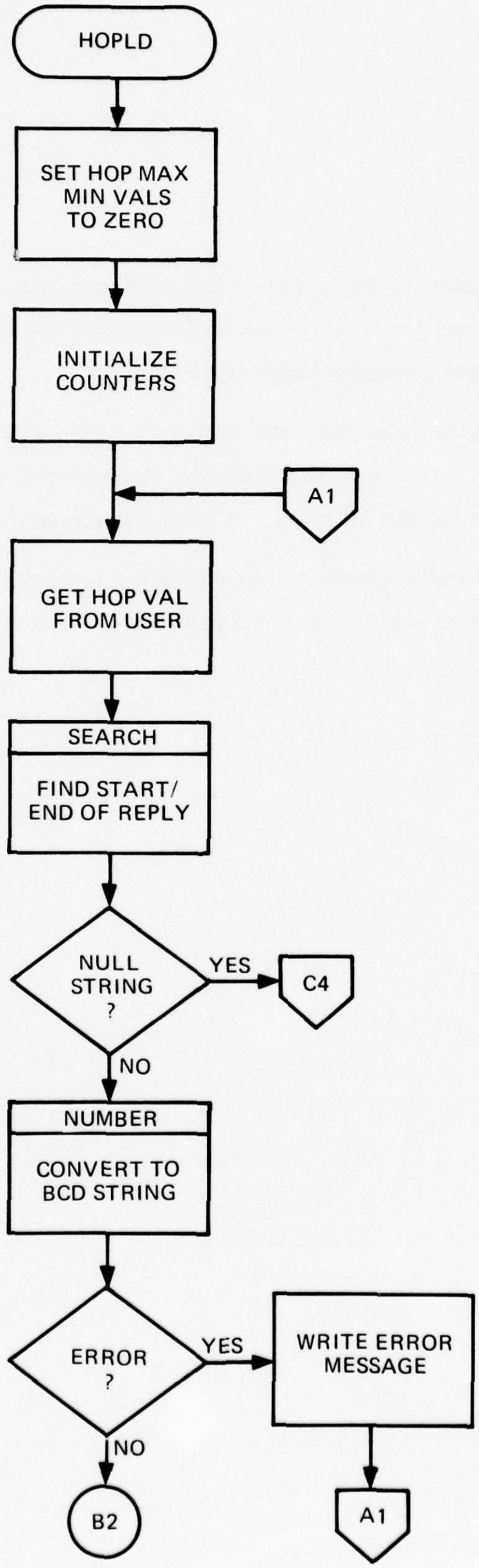
```
        RMIN(3)=SFREQ
        IF(SFREQ.LT.FREQ)GO TO 30
        RMAX(3)=SFREQ
        RMIN(3)=FREQ
  30    CONTINUE
C****   INFORM USER THAT DOP TABLE IS BUILT
        WRITE(6,904)
  904   FORMAT(1H ,'***DOP TABLE BUILT-SIZE=1024')
        RETURN
C****   OFFSET SPECIFIED CONTAINED INVALID CHARACTER
  200   WRITE(6,906)
  906   FORMAT(1H,'***INVALID FREQ')
        RETURN
C****   VALUE NOT IN 0-2 MEGA RANGE GENERATED
  300   WRITE(6,908)
  908   FORMAT(1H ,'***FREQ OUT OF RANGE')
        RETURN
        END

ROUTINES CALLED:
SEARCH, NUMBER, ASCRLN, RLNBCD

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
DOPLD   329     (001222)*
INBUF   10      (000024)
MAXMIN  24      (000060)
SYSTEM  29      (000072)
TABLES  3040    (013700)

**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES  00622 02344
EXECUTABLES   01467 01899
ASSEMBLY      01373 06233
```

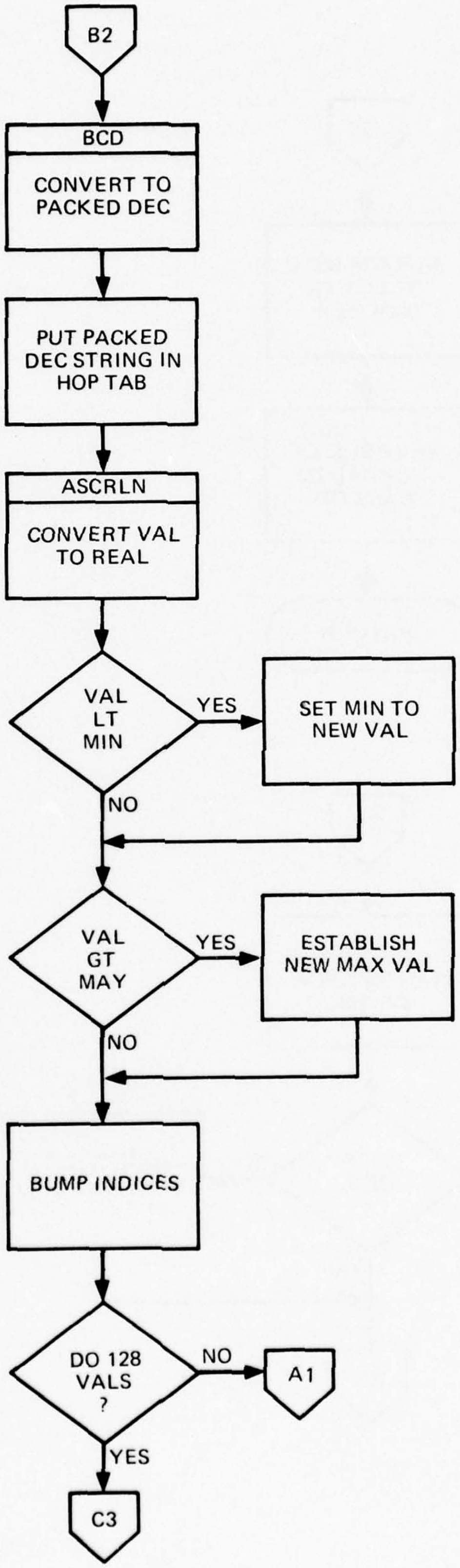
SUBROUTINE HOPLD

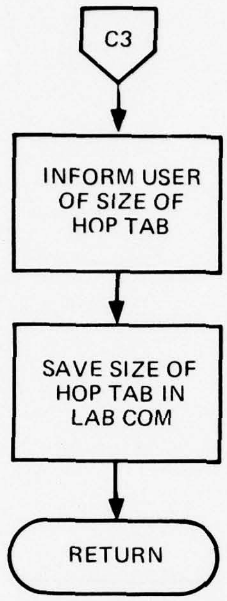
This routine is used to build the HOP table from user input. Each value input is converted to BCD, checked for error, converted to packed decimal form, and then entered into the HOP table contained in labeled common.

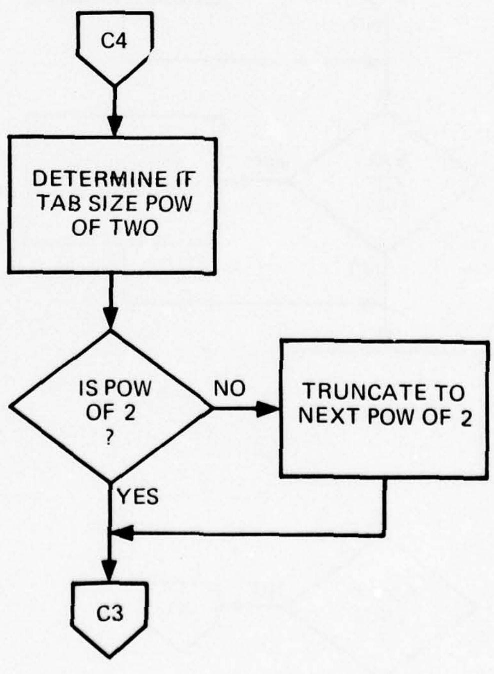
Besides managing the value input and conversions, the routine also determines the maximum and minimum HOP values in the table. These two values, maximum and minimum, are stored in the MAXMIN area of labeled common in double precision form.

The table size, initially determined by bumping a counter after each value is input, is checked before return to verify that it is a power of two. If it is not, the size is truncated to the next closest power of two.


HOPLD
SET HOP MAX MIN VALS TO ZERO
INITIALIZE COUNTERS
A1
GET HOP VAL FROM USER
SEARCH
FIND START/ END OF REPLY
NULL STRING ?
YES
C4
NO
NUMBER
CONVERT TO BCD STRING
ERROR ?
YES
WRITE ERROR MESSAGE
NO
B2
A1


B2
BCD
CONVERT TO PACKED DEC
PUT PACKED DEC STRING IN HOP TAB
ASCRLN
CONVERT VAL TO REAL
VAL LT MIN
YES
SET MIN TO NEW VAL
NO
VAL GT MAY
YES
ESTABLISH NEW MAX VAL
NO
BUMP INDICES
DO 128 VALS ?
NO
A1
YES
C3


C3
INFORM USER OF SIZE OF HOP TAB
SAVE SIZE OF HOP TAB IN LAB COM
RETURN


C4
DETERMINE IF TAB SIZE POW OF TWO
IS POW OF 2 ?
NO
TRUNCATE TO NEXT POW OF 2
YES
C3

```
      SUBROUTINE HOPLD
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO BUILD THE HOP TABLE
C**** USED BY THE INTELL. PATTERNS ARE ALL INPUT FROM THE USER.
C**** TABLE SIZE MUST BE POWER OF TWO OR SIZE IS TRUNCATED.
C
C***********************************************************************
      BYTE WORK(5)
      COMMON/INBUF/INLINE
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
      COMMON/TABLES/HOP,DOP,MFSK
      COMMON/MAXMIN/RMAX,RMIN
      DOUBLE PRECISION FREQ,RMAX(3),RMIN(3)
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND
      INTEGER MASK(3)
      BYTE INLINE(20),IVAL(12),PBASE(12)
      BYTE HOP(640),DOP(5120),MFSK(320)
      RMAX(1)=0.0
      RMIN(1)=0.0
      ISAVE=128
      J=1
      I=1
C**** QUERY FOR PATTTERN I
 10   WRITE(6,900)I
 900  FORMAT(1H ,'HOP FREQ(',I3,') =',/)
C**** READ USER REPLY
      READ(6,902)INLINE
 902  FORMAT(20A1)
C**** FIND START AND END END OF REPLY
      CALL SEARCH(ISTART,IEND)
      IF(ISTART.EQ.0)GO TO 200
C**** CONVERT TO BCD STRING
      CALL NUMBER(ISTART,IEND,IVAL,IERR)
      IF(IERR.EQ.0)GO TO 20
C**** FREQ INPUT CONTAINS BAD CHARACTER
      WRITE(6,904)
 904  FORMAT(1H ,'***INVALID FREQ')
      GO TO 10
C**** CONVERT TO 5 BYTE INTELL FORM
 20   CALL BCD(IVAL,WORK)
C**** PUT IN HOP TABLE
      DO 23 K=1,5
      HOP(J)=WORK(K)
      J=J+1
 23   CONTINUE
      I=I+1
C**** CHECK FOR MAX/MIN INPUT FREQ THIS PATTERN
      CALL ASCRLN(IVAL,FREQ)
      IF(FREQ.LT.RMIN(1))RMIN(1)=FREQ
      IF(FREQ.GT.RMAX(1))RMAX(1)=FREQ
C**** HOP TABLE LESS THAN EGUAL TO 128
      IF(I.LE.128)GO TO 10
C**** INFORM USER OF SIZE OF HOP TABLE
 25   WRITE(6,906)ISAVE
```

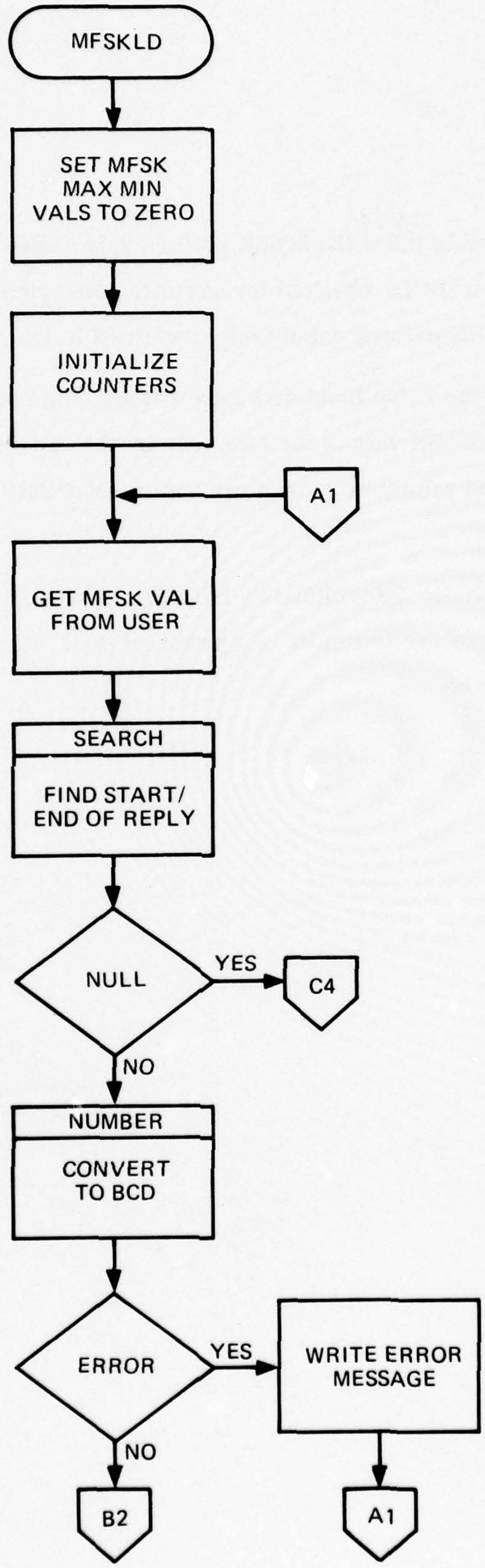
```
  906  FORMAT(1H ,'***HOP TABLE BUILT-SIZE=',I5)
C****  SAVE TABLE SIZE IN COMMON FOR PAT MASK GENERATION
       MASK(1)=ISAVE
       RETURN
C****  DETERMINE IF SIZE IS POWER OF TWO
  200  I=I-1
       DO 230 K=1,8
       M=2**K
       IF(M-I)210,210,230
  210  ISAVE=M
  230  CONTINUE
       IF(I.EQ.0)ISAVE=1
       GO TO 25
       END

ROUTINES CALLED:
SEARCH, NUMBER, BCD     , ASCRLN

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
HOPLD    334     (001234)*
INBUF    10      (000024)
SYSTEM   29      (000072)
TABLES   3040    (013700)
MAXMIN   24      (000060)

**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES 00622 02344
EXECUTABLES  01067 01899
ASSEMBLY     01429 06177
```

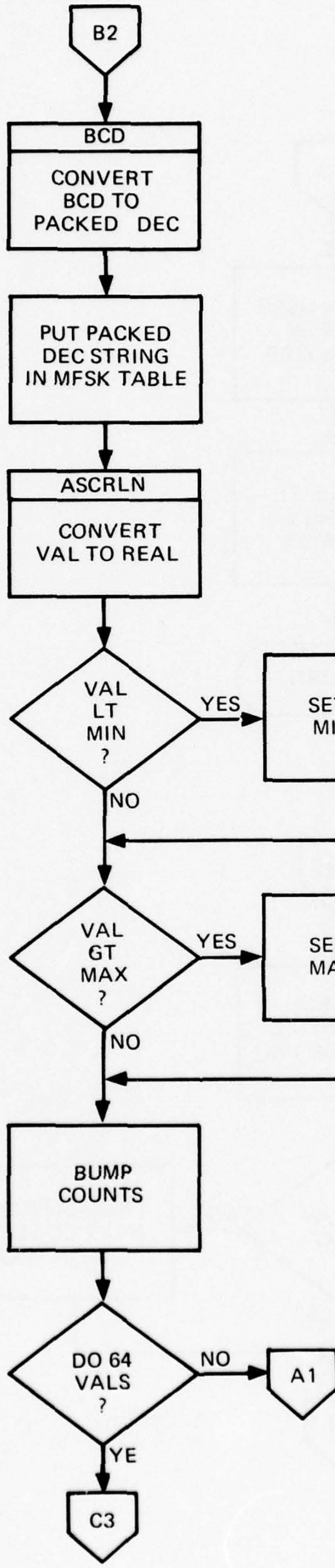
SUBROUTINE MFSKLD

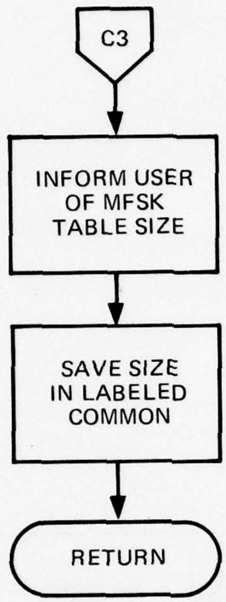
This routine is used to build the MFSK pattern value table from user input. Each value input is converted to BCD, checked for errors, converted to packed decimal, and then entered into the MFSK pattern value table contained in labeled common.

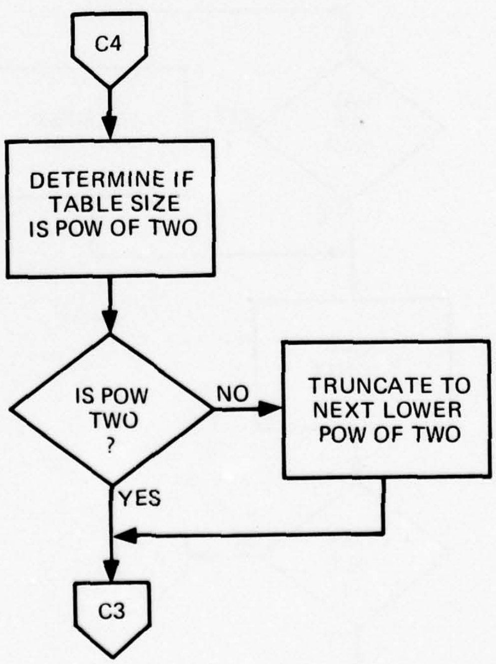
Besides managing the value input and conversions, the routine also determines the maximum and minimum MFSK values for later use by the "INTELL" routine. These two values, maximum and minimum, are stored in the MAXMIN area of labeled common in double precision form.

The table size, initially determined by bumping a counter after each value is input, is checked before return to verify that it is a power of two. If not, the size is truncated to the next closest power of two.


MFSKLD
SET MFSK MAX MIN VALS TO ZERO
INITIALIZE COUNTERS
A1
GET MFSK VAL FROM USER
SEARCH
FIND START/ END OF REPLY
NULL
YES
C4
NO
NUMBER
CONVERT TO BCD
ERROR
YES
WRITE ERROR MESSAGE
NO
B2
A1


B2
BCD
CONVERT BCD TO PACKED DEC
PUT PACKED DEC STRING IN MFSK TABLE
ASCRLN
CONVERT VAL TO REAL
VAL LT MIN ?
YES
SET MFSK MIN VAL
NO
VAL GT MAX ?
YES
SET MFSK MAX VAL
NO
BUMP COUNTS
DO 64 VALS ?
NO
A1
YE
C3


C3
INFORM USER OF MFSK TABLE SIZE
SAVE SIZE IN LABELED COMMON
RETURN


C4
DETERMINE IF TABLE SIZE IS POW OF TWO
IS POW TWO ?
NO
TRUNCATE TO NEXT LOWER POW OF TWO
YES
C3

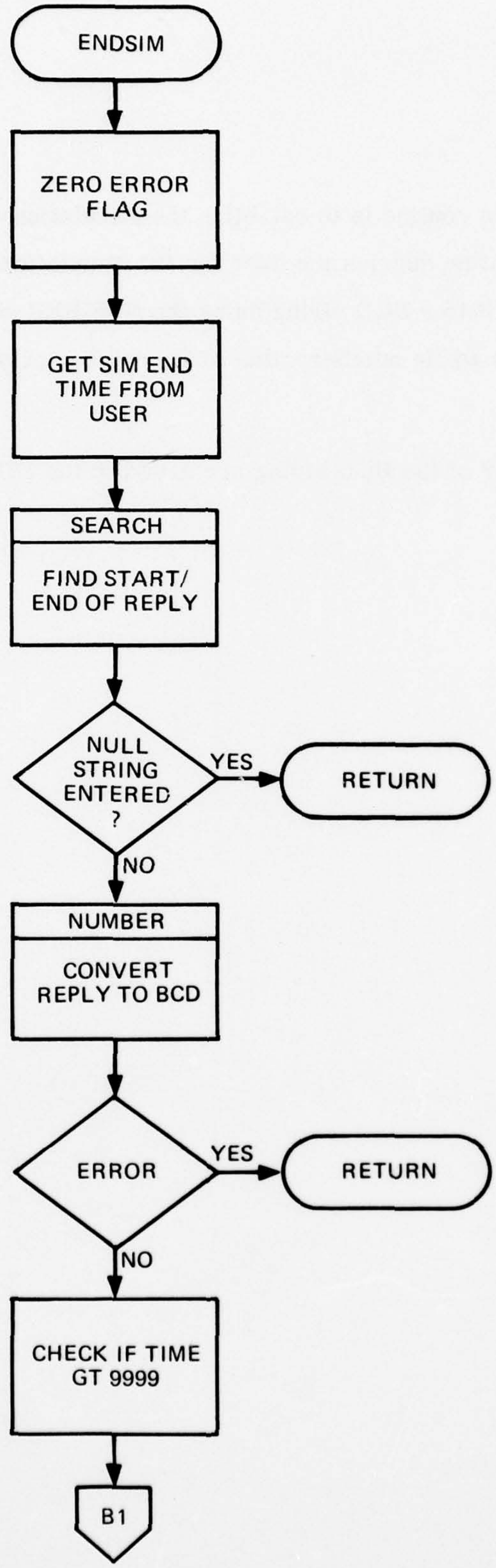
```
      SUBROUTINE MFSKLD
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO BUILD THE MFSK TABLE
C**** USED BY THE INTELL. ALL FREQS ARE INPUT BY OPERATOR.
C**** TABLE SIZE MUST BE POWER OF TWO OR SIZE WILL BE TRUNCATED.
C**** MAX TABLE SIZE IS 64.
C
C***********************************************************************
      BYTE WORK(5)
      COMMON/INBUF/INLINE
      BYTE INLINE(20),IVAL(12)
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
      COMMON/TABLES/HOP,DOP,MFSK
      COMMON/MAXMIN/RMAX,RMIN
      DOUBLE PRECISION RMAX(3),RMIN(3),FREQ
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND
      BYTE HOP(640),DOP(5120),MFSK(320),PBASE(12)
      INTEGER MASK(3)
      RMAX(2)=0.0
      RMIN(2)=0.0
      ISAVE=64
      J=1
      I=1
C**** ASK FOR FREQ I FROM USER
 10   WRITE(6,900)I
 900  FORMAT(1H ,'MFSK FREQ(',I3,') =',/)
C**** READ USER REPLY
      READ(6,902)INLINE
 902  FORMAT(20A1)
C**** DETERMINE START AND END OF REPLY
      CALL SEARCH(ISTART,IEND)
      IF(ISTART.EQ.0)GO TO 200
C**** CONVERT REPLY TO BCD STRING
      CALL NUMBER(ISTART,IEND,IVAL,IERR)
      IF(IERR.EQ.0)GO TO 20
C**** BAD CHARACTER INPUT
      WRITE(6,904)
 904  FORMAT(1H ,'***INVALID FREQ')
      GO TO 10
C**** CONVERT BCD STRING TO 5 BYTE INTELL FORM
 20   CALL BCD(IVAL,WORK)
C**** SAVE FREQ IN MFSK ARRAY
      DO 23 K=1,5
      MFSK(J)=WORK(K)
      J=J+1
 23   CONTINUE
      I=I+1
C**** CHECK FOR MAX/MIN INPUT FREQ THIS PATTERN
      CALL ASCBIN(IVAL,FREQ)
      IF(FREQ.LT.RMIN(2))RMIN(2)=FREQ
      IF(FREQ.GT.RMAX(2))RMAX(2)=FREQ
C**** CHECK FOR TABLE FULL
      IF(I.LE.64)GO TO 10
C**** INFORM USER THAT TABLE IS BUILT
```

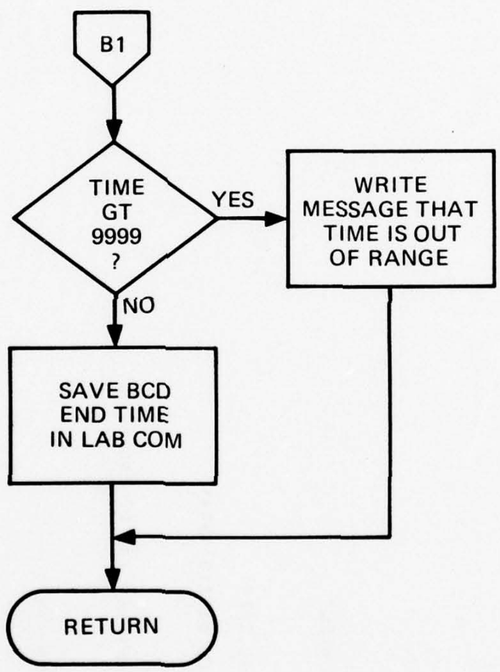
```
 25   WRITE(6,906)ISAVE
 906  FORMAT(1H ,'*****MFSK TABLE BUILT-SIZE=',I5)
C**** SAVE TABLE SIZE IN COMMON FOR PAT NASK GENERATION
      MASK(2)=ISAVE
      RETURN
 200  I=I-1
C**** DETERMINE IF SIZE IS POWER OF TWO ADJUST IF NOT
      DO 230 K=1,7
      M=2**K
      IF(M-I)210,210,230
 210  ISAVE=M
 230  CONTINUE
      IF(I.EQ.0)ISAVE=1
      GO TO 25
      END

ROUTINES CALLED:
SEARCH, NUMBER, BCD   , ASCRLN

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
MFSKLD  336     (001240)*
INBUF   10      (000024)
SYSTEM  29      (000072)
TABLES  3040    (013700)
MAXMIN  24      (000060)

**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES 00522 02344
EXECUTABLES  01067 01899
ASSEMBLY     01437 06169
```

SUBROUTINE ENDSIM

The purpose of this routine is to establish the simulation duration or end time via operator input. The routine queries the user for the simulation end time value, reads the reply, and converts it to a BCD string using the NUMBER subroutine. The converted value is interpreted as a whole number value of the number of minutes to run the simulation.

Bytes 5, 6, 7 and 8 of the BCD string are saved in the SYSTEM labeled common area.


ENDSIM
ZERO ERROR FLAG
GET SIM END TIME FROM USER
SEARCH
FIND START/ END OF REPLY
NULL STRING ENTERED ?
YES
RETURN
NO
NUMBER
CONVERT REPLY TO BCD
ERROR
YES
RETURN
NO
CHECK IF TIME GT 9999
B1


B1
TIME GT 9999 ?
YES
NO
WRITE MESSAGE THAT TIME IS OUT OF RANGE
SAVE BCD END TIME IN LAB COM
RETURN

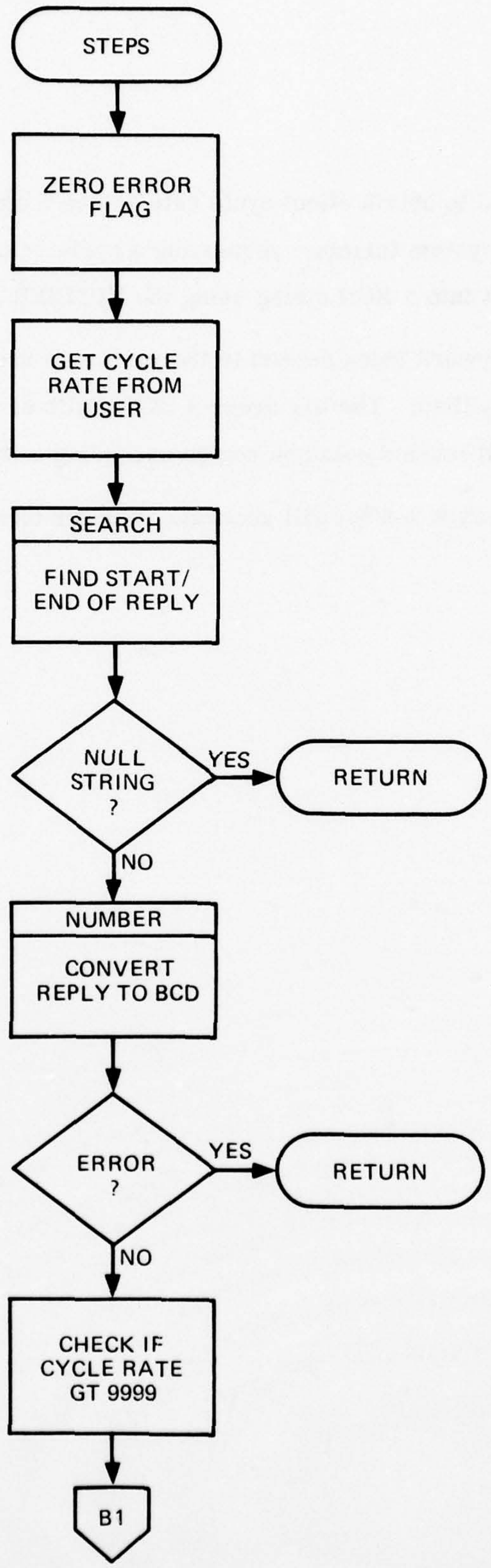
```
FORTRAN V06.13                 00:10:30    21-JAN-76    PAGE    1

      SUBROUTINE ENDSIM(IERR)
C*************************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO ESTABLISH THE SIMULATION
C**** END TIME FROM OPERATOR INPUT. END TIME IS SPECIFIED IN MINUTES.
C**** THE VALUE INPUT AFTER CONVERSION TO BCD IS SAVED IN COMMON.
C
C*************************************************************************
      COMMON/INBUF/INLINE
      BYTE INLINE(20)
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND
      BYTE PBASE(12),IVAL(12)
      EQUIVALENCE (ASCII,IVAL(5))
C**** WRITE QUERY MESSAGE
      WRITE(6,800)
 800  FORMAT(1H ,'END TIME =',/)
C**** GET REPLY
      READ(5,802)INLINE
 802  FORMAT(20A1)
      IERR=0
C**** DETERMINE START AND END OF REPLY
      CALL SEARCH(ISTART,IEND)
      IF(ISTART.EQ.0)RETURN
C**** CONVERT TO BCD STRING
      CALL NUMBER(ISTART,IEND,IVAL,IERR)
      IF(IERR.GT.0)RETURN
C**** VALUE SHOULD NOT BE GT 4 BCD DIGITS(9999)
      DO 20 I=2,4
      IF(IVAL(I).NE.0)GO TO 50
 20   CONTINUE
C**** ASCII EQUIV TO IVAL(5) RETURN BCD VAL OF END
      PSMEND=ASCII
      RETURN
 50   WRITE(6,900)
 900  FORMAT(1H0,'***TIME OUT OF RANGE')
      RETURN
      END

ROUTINES CALLED:
SEARCH, NUMBER

OPTIONS =/ON,/SU,/OP:1

BLOCK       LENGTH
ENDSIM  161     (000502)*
INBUF   10      (000024)
SYSTEM  26      (000064)

**COMPILER ----- CORE**
   PHASE       USED  FREE
DECLARATIVES 00315 02151
EXECUTABLES  00463 02103
ASSEMBLY     01137 06419
```

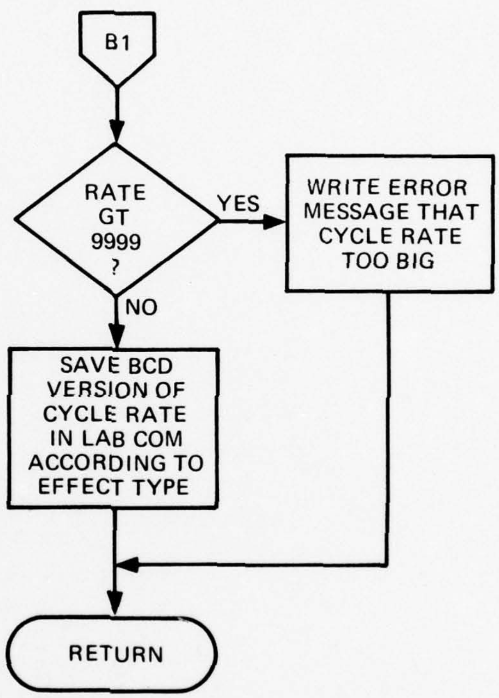
SUBROUTINE STEPS

This routine is used to obtain effect cycle rate values from the user. The routine writes a message to the system terminal requesting a cycle value be provided, reads the reply, and converts it into a BCD string using the NUMBER routine.

The value of the keyword index passed to the routine is used to determine the particular effect type specified. The low order 4 BCD digits of the converted string are stored in the SYSTEM labeled common region according to the effect type determined.

A value not in the range 0-9999 will generate an error message.


STEPS
ZERO ERROR FLAG
GET CYCLE RATE FROM USER
SEARCH
FIND START/ END OF REPLY
NULL STRING ?
YES
RETURN
NO
NUMBER
CONVERT REPLY TO BCD
ERROR ?
YES
RETURN
NO
CHECK IF CYCLE RATE GT 9999
B1


B1
RATE GT 9999 ?
YES
WRITE ERROR MESSAGE THAT CYCLE RATE TOO BIG
NO
SAVE BCD VERSION OF CYCLE RATE IN LAB COM ACCORDING TO EFFECT TYPE
RETURN

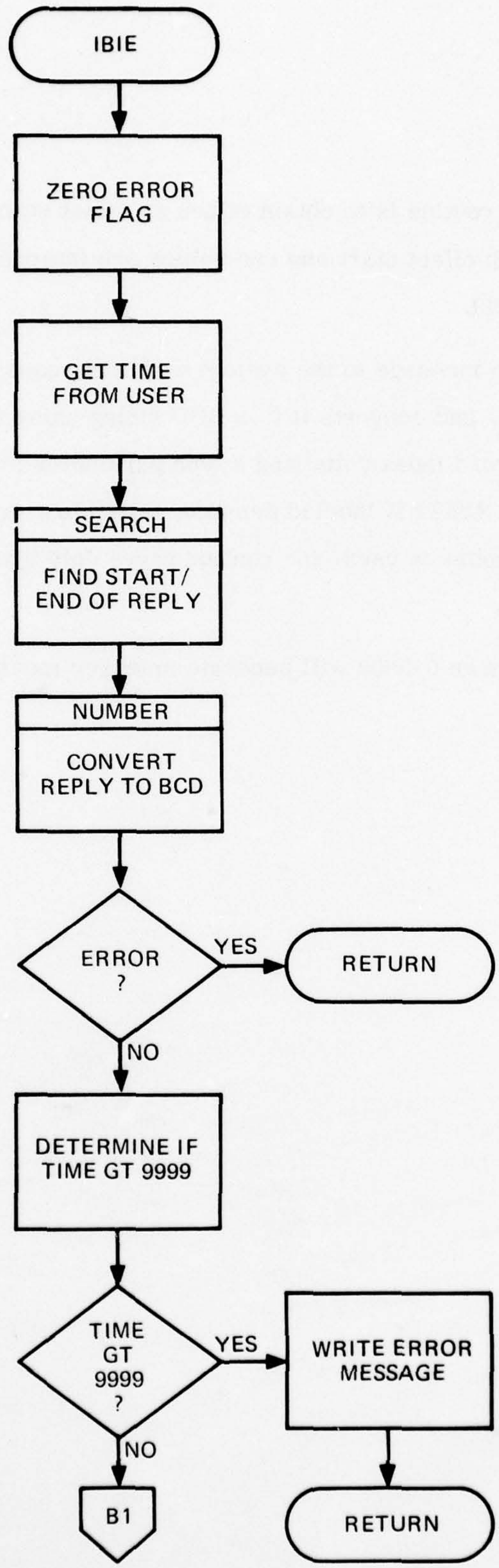
```
      SUBROUTINE STEPS(IND,IERR)
C*********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO ESTABLISH THE CYCLE
C**** RATE THROUGH OPERATOR INPUT. THE VALUE IND DETERMINES WHICH
C**** CYCLE RATE IS INPUT(13=HOP,14=MFSK,15=OOP).THE VALUE AFTER BCD
C**** CONVERSION IS SAVED IN THE PCYCLE AREA OF COMMON.
C
C*********************************************************************
      COMMON/INBUF/INLINE
      BYTE INLINE(20)
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND,ASCII
      BYTE PBASE(12),IVAL(12)
      INTEGER MASK(3)
      EQUIVALENCE (ASCII,IVAL(5))
      IERR=0
C**** ASK FOR CYCLE RATE
      WRITE(6,800)
 800  FORMAT(1H ,'CYCLE RATE =',/)
C**** READ REPLY
      READ(6,802)INLINE
 802  FORMAT(20A1)
C**** GET REPLY START AND END
      CALL SEARCH(ISTART,IEND)
      IF(ISTART.EQ.0)RETURN
C**** CONVERT REPLY TO BCD STRING
      CALL NUMBER(ISTART,IEND,IVAL,IERR)
      IF(IERR.GT.0)RETURN
C**** RATE MUST NOT EXCEED 9999
      DO 20 I=2,4
      IF(IVAL(I).NE.0)GO TO 50
 20   CONTINUE
C**** INSERT CYCLE RATE ACCORDING TO IND VAL
      PCYCLE(IND-12)=ASCII
      RETURN
 50   WRITE(6,900)
 900  FORMAT(1H0,'***CYCLE RATE TOO LARGE')
      RETURN
      END

ROUTINES CALLED:
SEARCH, NUMBER

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
STEPS   170    (000524)*
INBUF   10     (000024)
SYSTEM  29     (000072)

**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES 00522 02344
EXECUTABLES  00496 02070
ASSEMBLY     01215 06391
```

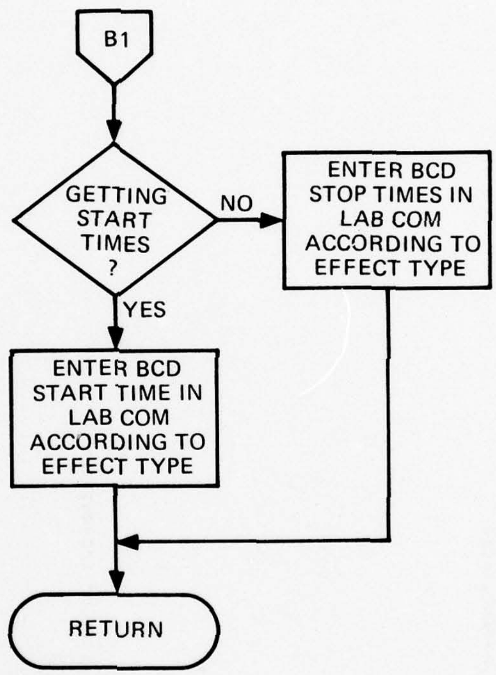
SUBROUTINE IBIE

The purpose of this routine is to obtain either an effect start or effect end time value from the user. Both effect start and end values are interpreted as whole number minutes in the range 0-9999.

The routine writes a message to the system terminal requesting a time value be provided, reads the reply, and converts it to a BCD string using the NUMBER subroutine. The routine uses the keyword index value and a type parameter (either STOP or START) to determine where in the SYSTEM labeled common region to store the value. Since only a four digit whole number is used, the routine saves only bytes 5, 6, 7, and 8 of the converted BCD string.

A value not in the range 0-9999 will generate an error message.


IBIE
ZERO ERROR FLAG
GET TIME FROM USER
SEARCH
FIND START/ END OF REPLY
NUMBER
CONVERT REPLY TO BCD
ERROR ?
YES
RETURN
NO
DETERMINE IF TIME GT 9999
TIME GT 9999 ?
YES
WRITE ERROR MESSAGE
RETURN
NO
B1


B1
GETTING START TIMES ?
NO
YES
ENTER BCD STOP TIMES IN LAB COM ACCORDING TO EFFECT TYPE
ENTER BCD START TIME IN LAB COM ACCORDING TO EFFECT TYPE
RETURN

```
FORTRAN V06.13                 00:11:08    21-JAN-76    PAGE    1

      SUBROUTINE IBIE(IND,TYPE,IERR)
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO ESTABLISH EITHER THE
C**** EFFECT START OR END TIMES THROUGH OPERATOR INPUT. THE
C**** TYPE PARAMETER DETERMINES STOP OR START. IND DETERMINES
C**** WHICH EFFECT. IND=7,10 IS MOP,=8,11 IS MFSK,=9,12 IS DOP. THE
C**** VALUE AFTER BCD CONVERSION IS SAVED IN EITHER THE PSTART OR
C**** PSTOP AREAS OF LABELED COMMON.
C
C***********************************************************************
      COMMON/INBUF/INLINE
      BYTE INLINE(20)
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND,ASCII
      BYTE PBASE(12),IVAL(12)
      INTEGER MASK(3)
      EQUIVALENCE (IVAL(5),ASCII)
      IERR=0
C**** QUERY FOR TIME
      WRITE(6,800)
 800  FORMAT(1H ,'TIME(MINS) =',/)
C**** READ USER REPLY
      READ(6,802)INLINE
 802  FORMAT(20A1)
C**** GET START AND END OF REPLY
      CALL SEARCH(ISTART,IEND)
      IF(ISTART.EQ.0)RETURN
C**** CONVERT REPLY TO BCD STRING
      CALL NUMBER(ISTART,IEND,IVAL,IERR)
      IF(IERR.GT.0)GO TO 200
C**** VALUE MUST BE LE 9999
      DO 20 I=2,4
      IF(IVAL(I).NE.0)GO TO 100
 20   CONTINUE
C**** DETERMINE IF START OR STOP AND SAVE VALUE ACCORDINGLY
      IF(TYPE.EQ.'STOP')GO TO 50
      INDEX=IND-6
C**** ASCII IS BYTES 5-8 OF IVAL
      PSTART(INDEX)=ASCII
      RETURN
 50   INDEX=IND-9
C**** ASCII IS BYTES 5-8 OF IVAL
      PSTOP(INDEX)=ASCII
      RETURN
 100  WRITE(6,900)
 900  FORMAT(1H0,'***TIME OUT OF LIMITS')
      RETURN
 200  IERR=1
      RETURN
      END

ROUTINES CALLED:
SEARCH, NUMBER

OPTIONS =/ON,/SU,/OP:1

BLOCK      LENGTH
```

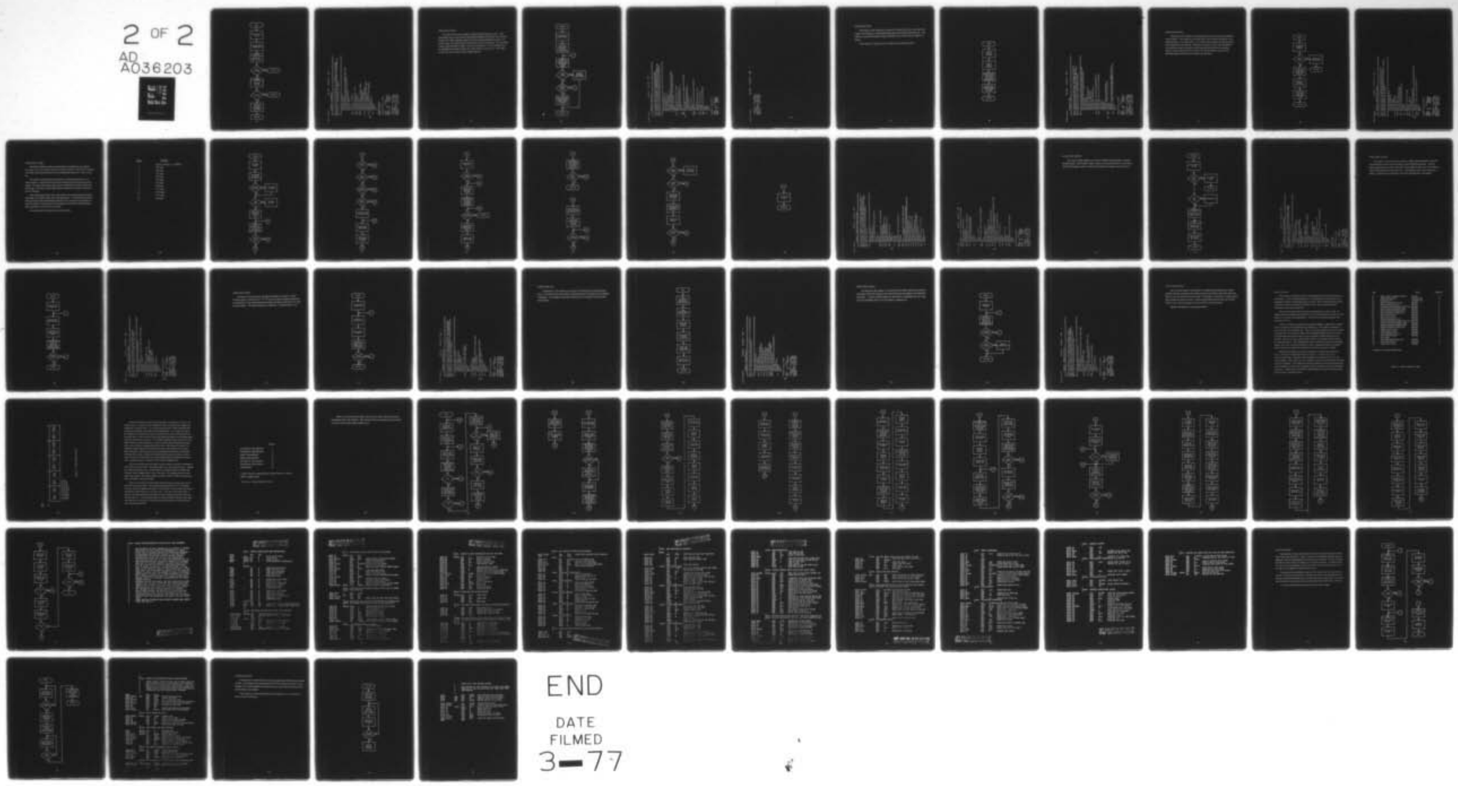
```
FORTRAN V06.13                  00:11:08    21-JAN-76    PAGE    2

        IRIE     005    (000632)*
        INBUF    10     (000024)
        SYSTEM   29     (000072)

        **COMPILER ----- CORE**
           PHASE        USED  FREE
        DECLARATIVES 00622 02344
        EXECUTABLES  00896 02070
        ASSEMBLY     01247 06359
```

SUBROUTINE FREQ

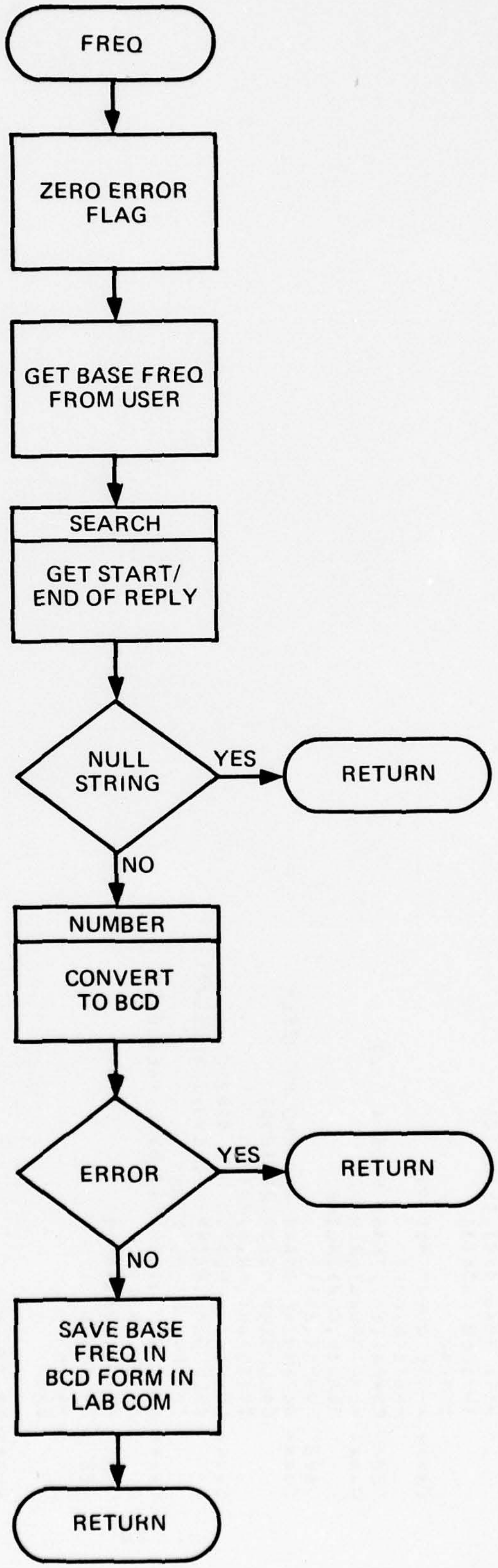
This routine is used to obtain the base frequency value from the user. The routine writes a message to the system terminal requesting a base frequency value be provided, reads the reply, and converts it to a BCD string using the NUMBER subroutine. The BCD value of the base frequency is then stored in labeled common.

AD-A036 203 COMPUTER SCIENCES CORP FALLS CHURCH VA F/G 9/2
CSEL INTERFACE UPDATE: K-BAND MICROPROCESSOR DEMONSTRATION.(U)
DEC 76 D O ALWINE, R A GLICKSMAN, R G MURRAY F33615-75-C-1229
UNCLASSIFIED CSC-R-3328-1 AFAL-TR-76-169 NL

2 OF 2
AD
A036203



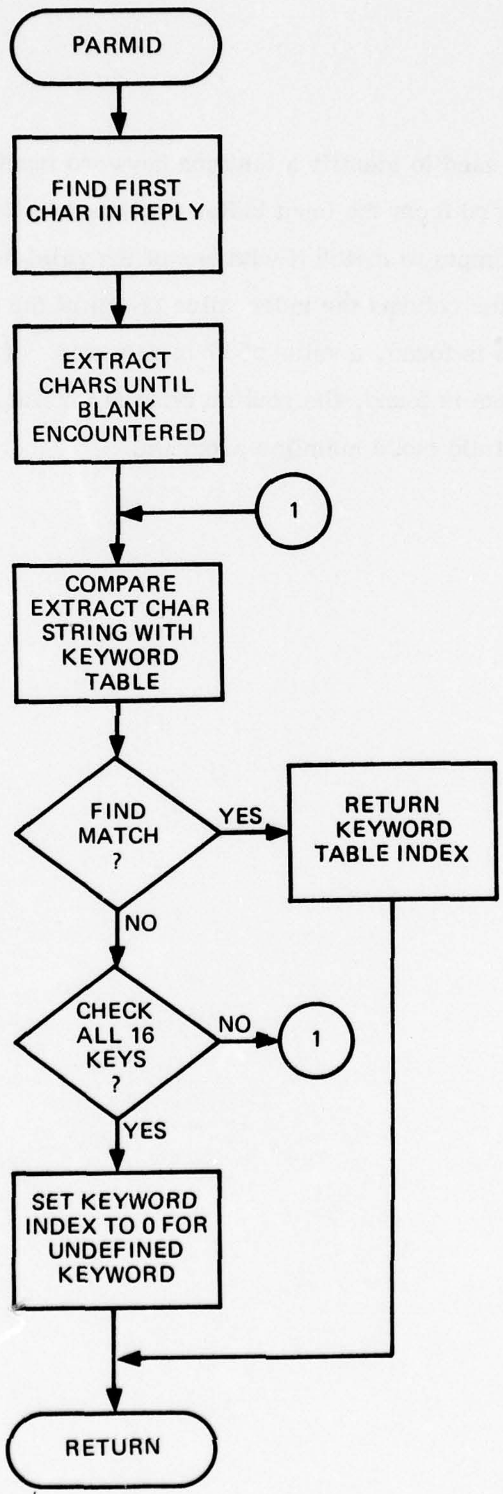
END
DATE
FILMED
3-77


FREQ
ZERO ERROR FLAG
GET BASE FREQ FROM USER
SEARCH
GET START/ END OF REPLY
NULL STRING
YES
RETURN
NO
NUMBER
CONVERT TO BCD
ERROR
YES
RETURN
NO
SAVE BASE FREQ IN BCD FORM IN LAB COM
RETURN

```
      SUBROUTINE FREQ(IERR)
C**********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO ESTABLISH THE BASE FREQ
C**** THROUGH OPERATOR INPUT. THE FREQ VALUE AFTER BCD CONVERSION IS
C**** SAVED IN THE PBASE AREA OF COMMON.
C
C**********************************************************************
      COMMON/INBUF/INLINE
      BYTE INLINE(20)
      COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
      REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND
      BYTE PBASE(12),IVAL(12)
      INTEGER MASK(3)
      IERR=0
C**** WRITE QUERY MESSAGE
      WRITE(6,800)
 800  FORMAT(1H ,'BASE FREQ = ',/)
C**** READ OPERATOR REPLY
      READ(6,802)INLINE
 802  FORMAT(20A1)
C**** DETERMINE START AND END OF REPLY
      CALL SEARCH(ISTART,IEND)
      IF(ISTART.EQ.0)RETURN
C**** CONVERT REPLY TO BCD STRING
      CALL NUMBER(ISTART,IEND,IVAL,IERR)
      IF(IERR.GT.0)RETURN
C**** SAVE BCD STRING AS BASE FREQUENCY
      DO 20 I=1,12
      PBASE(I)=IVAL(I)
 20   CONTINUE
      RETURN
      END

ROUTINES CALLED:
SEARCH, NUMBER

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
FREQ    127     (000376)*
INBUF   10      (000024)
SYSTEM  29      (000072)

**COMPILER ----- CORE**
   PHASE       USED  FREE
DECLARATIVES 00522 02344
EXECUTABLES  00886 02080
ASSEMBLY     01127 06479
```

SUBROUTINE PARMID

This subroutine is used to identify a function keyword input by the user. After extracting the user keyword from the input buffer contained in the INBUF area of labeled common, the routine attempts to match it with one of the valid function keywords. If a match is found, the routine returns the index value (1-16) of the keyword. If a null string, carriage return response is found, a value of 17 is returned. If a non-carriage return and unidentifiable response is found, the routine returns a value of zero. The routine is called only from the static mode mainline program.


PARMID
FIND FIRST CHAR IN REPLY
EXTRACT CHARS UNTIL BLANK ENCOUNTERED
1
COMPARE EXTRACT CHAR STRING WITH KEYWORD TABLE
FIND MATCH ?
YES
RETURN KEYWORD TABLE INDEX
NO
CHECK ALL 16 KEYS ?
NO
1
YES
SET KEYWORD INDEX TO 0 FOR UNDEFINED KEYWORD
RETURN

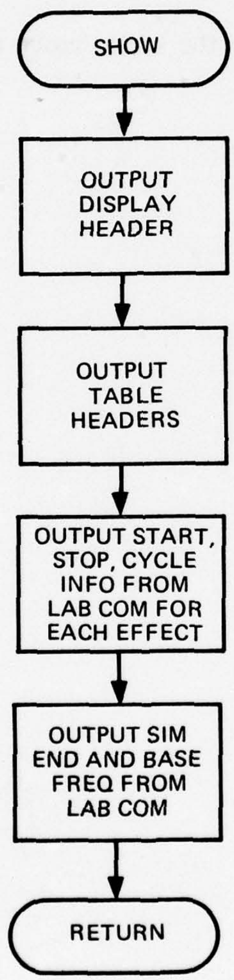
```
FORTRAN V06.13               30:12:59    21-JAN-76    PAGE    1
      SUBROUTINE PARMID(IND)
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO IDENTIFY THE USER
C**** SPECIFIED KEYWORD AND RETURN A NUMERIC IDENTIFIER TO
C**** THE CALLER. THE IDENTIFIER IND CORRESPONDS TO THE INDEX OF THE
C**** KEYWORD IN THE PARM ARRAY. IND IS 0 IF THE KEYWORD CAN NOT BE
C**** FOUND AND IS NOT A CAR RETN. IND=17 WHEN EQUAL TO CAR RETN.
C
C***********************************************************************
      COMMON/INBUF/INLINE
      COMMON/SPCDEC/CHARS,ICAR
      BYTE CHARS(20)
      BYTE INLINE(20),ABYT(4)
      REAL PARMS(16),ASCII
      EQUIVALENCE (ASCII,ABYT(1))
      DATA PARMS/'SHOW','LOAD','FREQ','HOPL','MFSL','DOPL',
     1 'HSTR','MSTR','DSTR','HEND','MEND','DEND','HCYC',
     2'MCYC','DCYC','SEND'/
      ACSII='    '
C**** SCAN FOR FIRST CHARACTER IN REPLY
      DO 10 I=1,20
      IF(INLINE(I).EQ.ICAR)GO TO 70
      IF(INLINE(I).NE.CHARS(7))GO TO 15
 10   CONTINUE
 15   K=1
C**** FILL THE ABYT ARRAY WITH CHARS UNTIL END STRING
      DO 20 J=I,20
      IF(INLINE(J).EQ.ICAR)GO TO 30
      IF(INLINE(J).EQ.CHARS(7))GO TO 30
      ABYT(K)=INLINE(J)
      K=K+1
      IF(K.GT.4)GO TO 30
 20   CONTINUE
C**** ABYT EQUIV TO ASCII-MATCH ASCII WITH KEYS
 30   IND=0
      DO 40 I=1,16
      IF(PARMS(I).EQ.ASCII)GO TO 60
 40   CONTINUE
C**** NO MATCH THAN IND IS ZERO
      RETURN
C**** FOUND MATCH SET IND ACCORDINGLY
 60   IND=I
      RETURN
 70   IND=17
      RETURN
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK      LENGTH
PARMID  206    (000634)*
INBUF   10     (000024)
SPCDEC  11     (000026)
```

```
FORTRAN V06.15          00:12:59     21-JAN-76    PAGE    2

        **COMPILER ----- CORE**
           PHASE       USED  FREE
        DECLARATIVES 00522 02344
        EXECUTABLES  00492 02074
        ASSEMBLY     01107 06499
```

SUBROUTINE SHOW

The purpose of this routine is to display the current effect start, end, and cycle values, base frequency, and simulation end value on the system terminal device. The values are stored in the labeled common SYSTEM area in BCD form and output in "I" format.

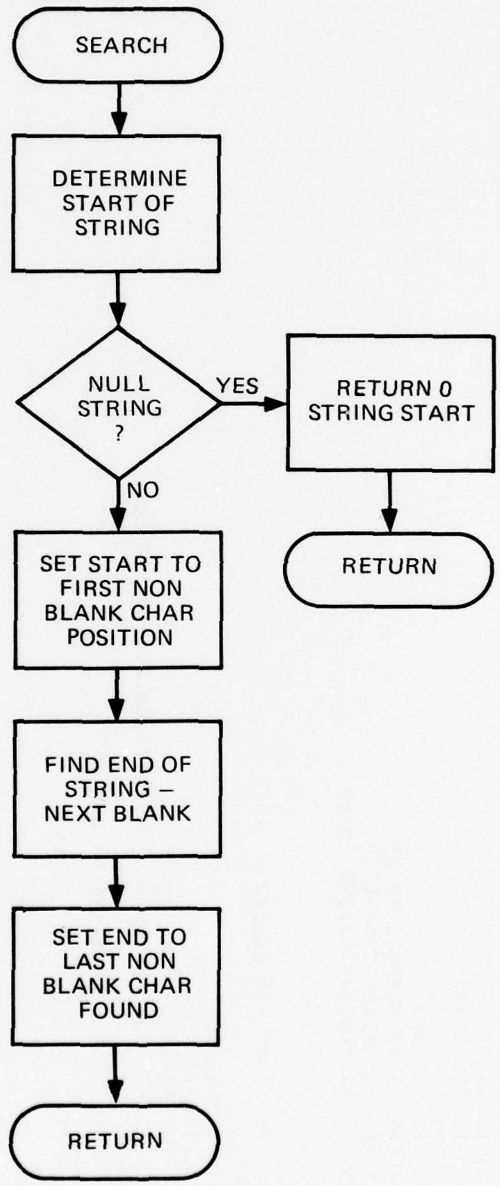
This routine is called only from the static mode mainline program.


SHOW
OUTPUT DISPLAY HEADER
OUTPUT TABLE HEADERS
OUTPUT START, STOP, CYCLE INFO FROM LAB COM FOR EACH EFFECT
OUTPUT SIM END AND BASE FREQ FROM LAB COM
RETURN

```
FORTRAN V06.13                02:12:42     21-JAN-76    PAGE    1

           SUBROUTINE SHOW
      C***********************************************************************
      C
      C**** THE PRUPOSE OF THIS ROUTINE IS TO DISPLAY THE CURRENT
      C**** SYSTEM PARAMETERS TO THE USER. THE PARAMETERS ARE
      C**** INITIALIZED IN BLOCK DATA AND SET BY THE USER, THIER
      C**** CURRENT VALUE ARE IN LABELED COMMON(SYSTEM).
      C
      C***********************************************************************
            COMMON/SYSTEM/PSTART,PSTOP,PCYCLE,PBASE,PSMEND,MASK
            REAL PSTART(3),PSTOP(3),PCYCLE(3),PSMEND,ID(3)
            BYTE PBASE(12)
            INTEGER MASK(3)
            BYTE ABYT(4),BBYT(4),CBYT(4)
            EQUIVALENCE (ABYT(1),A)
            EQUIVALENCE (B,BBYT(1))
            EQUIVALENCE (C,CBYT(1))
            DATA ID/'HOP ','MFSK','DOP '/
            WRITE(6,800)
       800  FORMAT(1H ,2X,'****SYSTEM PARAMETERS****')
            WRITE(6,902)
       902  FORMAT(1H ,8X,'START',5X,'END ',4X,'CYCLE')
            DO 20 I=1,3
            A=PSTART(I)
            B=PSTOP(I)
            C=PCYCLE(I)
            WRITE(6,904)ID(I),ABYT,BBYT,CBYT
       904  FORMAT(1H ,A4,4X,4I1,6X,4I1,5X,4I1)
       20   CONTINUE
            A=PSMEND
            WRITE(6,906)ABYT,(PBASE(J),J=2,8),(PBASE(K),K=9,11)
       906  FORMAT(1H ,'SIM END= ',4I1,2X,'BASE FREQ=',7I1,'.',3I1)
            RETURN
            END

      OPTIONS =/ON,/SU,/OP:1

      BLOCK        LENGTH
      SHOW    244    (000750)*
      SYSTEM  29     (000072)

      **COMPILER ----- CORE**
         PHASE       USED  FREE
      DECLARATIVES 00827 02139
      EXECUTABLES  00863 02103
      ASSEMBLY     01222 06384
```

SUBROUTINE SEARCH

The purpose of this routine is to determine the start and end character positions of a user response. The routine first scans the INBUF area of labeled common for a non-blank character or a carriage return. If a carriage return is found, the routine sets the start parameter to zero and exits. Otherwise, the routine sets the start parameter to the character position of the first non-blank character and continues its scan until a blank or carriage return character is found. The end parameter is then set to the character position of the blank or carriage return character.



SEARCH
DETERMINE START OF STRING
NULL STRING ?
YES
RETURN 0 STRING START
RETURN
NO
SET START TO FIRST NON BLANK CHAR POSITION
FIND END OF STRING – NEXT BLANK
SET END TO LAST NON BLANK CHAR FOUND
RETURN

```
FORTRAN V06.13                  00:12:27     21-JAN-76    PAGE    1

      SUBROUTINE SEARCH(ISTART,IEND)
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO DETERMINE THE START
C**** AND END POSITION IN THE INLINE ARRAY OF THE USERS
C**** RESPONSE. ISTART IS THE BEGINNING,IEND IS THE FINISH. ISTART=0
C**** WILL INDICATE A CAR RETN REPLY.
C
C***********************************************************************
      COMMON/SPCDEC/CHARS,ICAR
      COMMON/INBUF/INLINE
      BYTE CHARS(20)
      BYTE INLINE(20)
C**** DETERMINE THE START
      DO 20 I=1,20
C**** CHECK FOR CAR RETN
      IF(INLINE(I).EQ.ICAR)GO TO 25
C**** CHECK FOR NIN BLANK CHARACTER
      IF(INLINE(I).NE.CHARS(7))GO TO 30
 20   CONTINUE
 25   ISTART=0
      RETURN
C**** ESTABLISH START FOR CALLER
 30   ISTART=I
C**** FIND END POSITION
      DO 50 J=I,20
C**** CHECK FOR BLANK OR CAR RETN
      IF(INLINE(J).EQ.CHARS(7).OR.INLINE(J).EQ.ICAR)GO TO 80
 50   CONTINUE
C**** END IS ONE LESS THAN BLANK OR CAR RETN
 80   IEND=J-1
      RETURN
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK      LENGTH
SEARCH  99     (000306)*
SPCDEC  11     (000026)
INBUF   10     (000024)

**COMPILER ----- CORE**
  PHASE       USED  FREE
DECLARATIVES 00402 02164
EXECUTABLES  00783 02183
ASSEMBLY     00995 06611
```

SUBROUTINE NUMBER

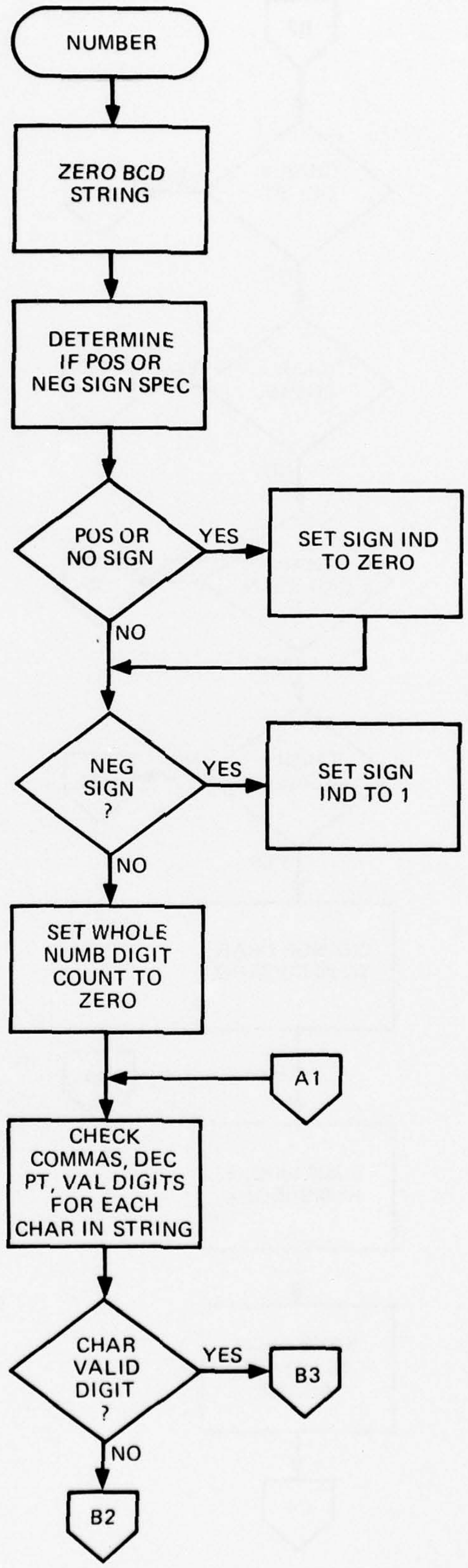
Subroutine NUMBER is called by all subroutines performing operator dialogue. Its purpose is to scan a response string to verify that it contains a valid numeric response and to build a BCD equivalent array for use in determining frequencies, values, times, etc.

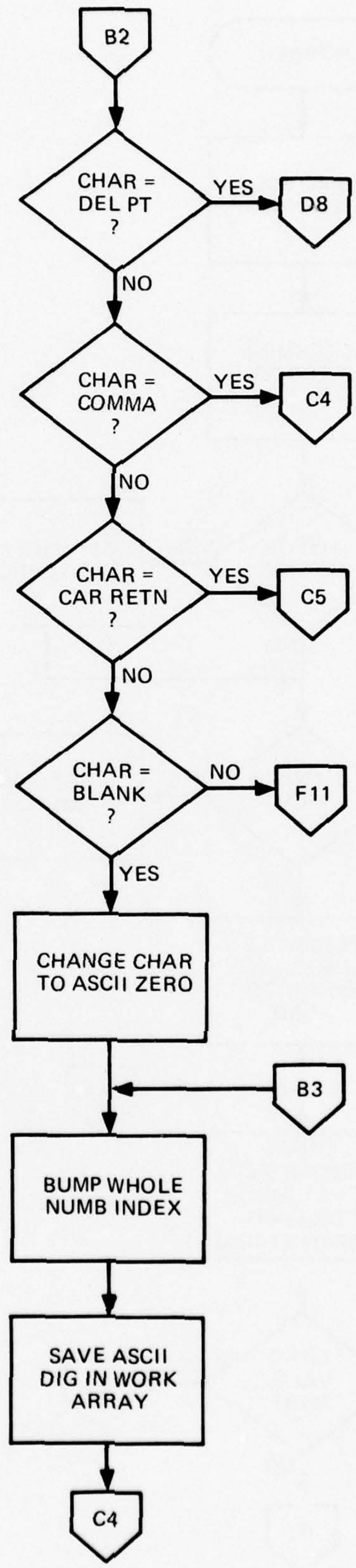
The response string scanned by the routine is contained in the INBUF area of labeled common. The start and end character positions in the string are passed to the routine. The routine will accept a sign, comma, decimal point, or blank as part of the input string. Any other non ASCII digit character contained in the string is considered an error condition.

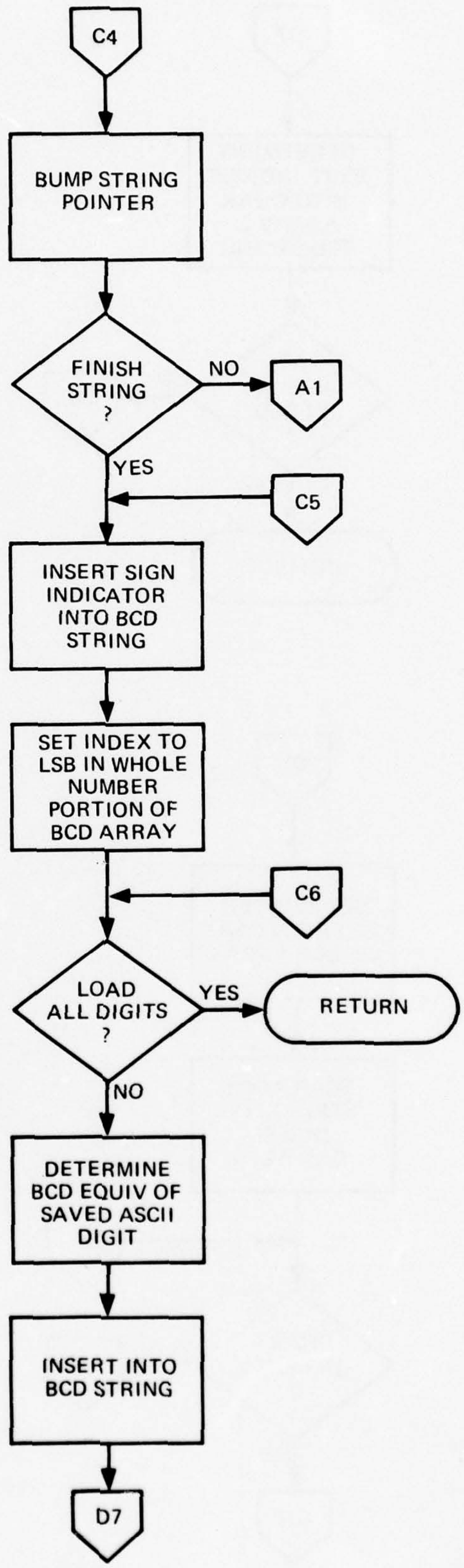
After determining the sign value of the response, the routine begins extracting the ASCII digits of the whole number value of the input string. If a decimal point is encountered, the routine begins extracting the fractional part of the string and converting it to its BCD equivalent. Conversion of the whole number ASCII digits to their BCD equivalents is performed prior to return to the caller.

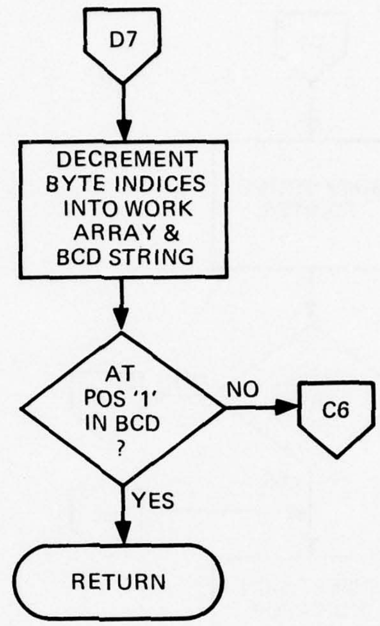
The format of the BCD returned array is shown below.

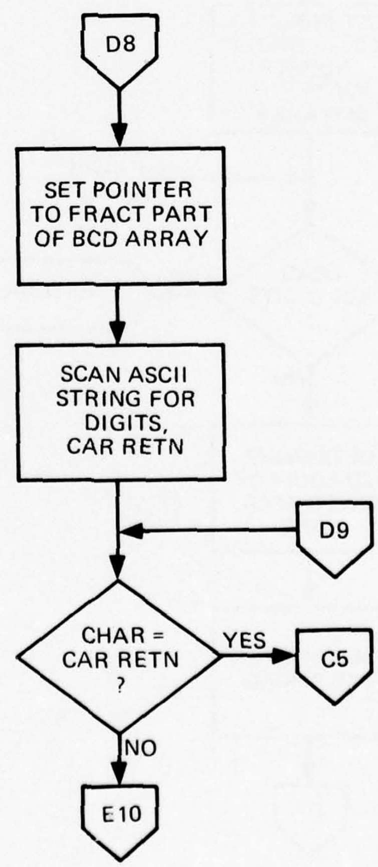
Byte #	Contents
1	Sign (0 = positive, 1 = negative)
2	10^6 value
3	10^5 value
4	10^4 value
5	10^3 value
6	10^2 value
7	10^1 value
8	10^0 value
9	10^{-1} value
10	10^{-2} value
11	10^{-3} value
12	Not used

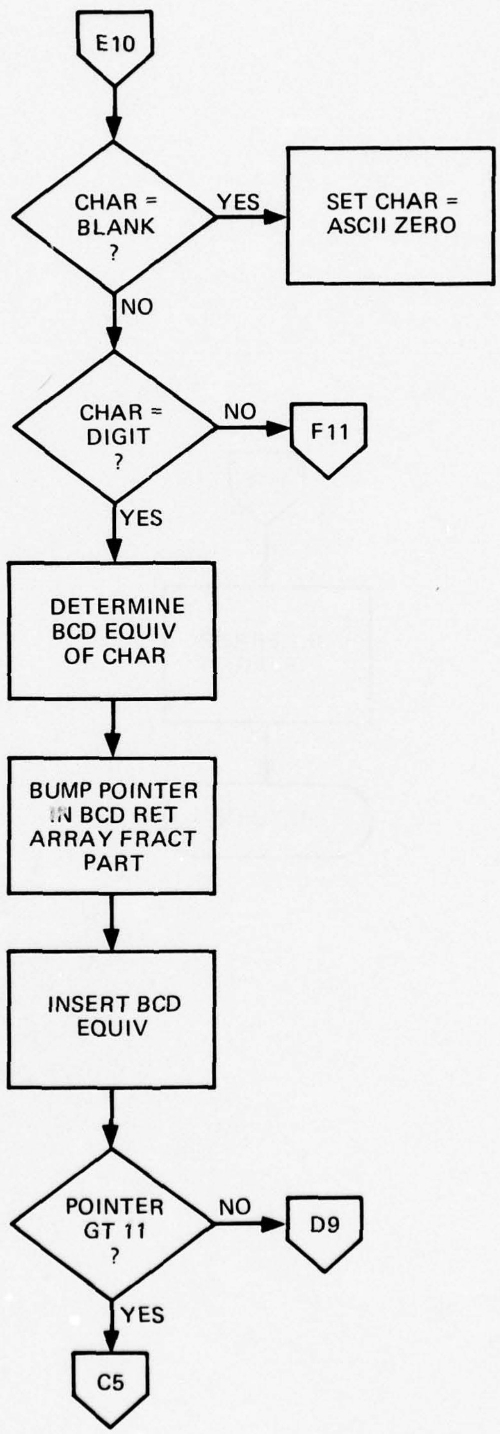

NUMBER
ZERO BCD STRING
DETERMINE IF POS OR NEG SIGN SPEC
POS OR NO SIGN
YES
SET SIGN IND TO ZERO
NO
NEG SIGN ?
YES
SET SIGN IND TO 1
NO
SET WHOLE NUMB DIGIT COUNT TO ZERO
A1
CHECK COMMAS, DEC PT, VAL DIGITS FOR EACH CHAR IN STRING
CHAR VALID DIGIT ?
YES
B3
NO
B2

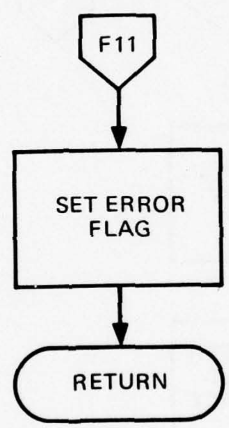

B2
CHAR = DEL PT ?
YES
D8
NO
CHAR = COMMA ?
YES
C4
NO
CHAR = CAR RETN ?
YES
C5
NO
CHAR = BLANK ?
NO
F11
YES
CHANGE CHAR TO ASCII ZERO
B3
BUMP WHOLE NUMB INDEX
SAVE ASCII DIG IN WORK ARRAY
C4


C4
BUMP STRING POINTER
FINISH STRING ?
NO
A1
YES
C5
INSERT SIGN INDICATOR INTO BCD STRING
SET INDEX TO LSB IN WHOLE NUMBER PORTION OF BCD ARRAY
C6
LOAD ALL DIGITS ?
YES
RETURN
NO
DETERMINE BCD EQUIV OF SAVED ASCII DIGIT
INSERT INTO BCD STRING
D7


D7
DECREMENT BYTE INDICES INTO WORK ARRAY & BCD STRING
AT POS '1' IN BCD ?
NO
C6
YES
RETURN


D8
SET POINTER TO FRACT PART OF BCD ARRAY
SCAN ASCII STRING FOR DIGITS, CAR RETN
D9
CHAR = CAR RETN ?
YES
C5
NO
E10


E10
CHAR = BLANK ?
YES
SET CHAR = ASCII ZERO
NO
CHAR = DIGIT ?
NO
F11
YES
DETERMINE BCD EQUIV OF CHAR
BUMP POINTER IN BCD RET ARRAY FRACT PART
INSERT BCD EQUIV
POINTER GT 11 ?
NO
D9
YES
C5

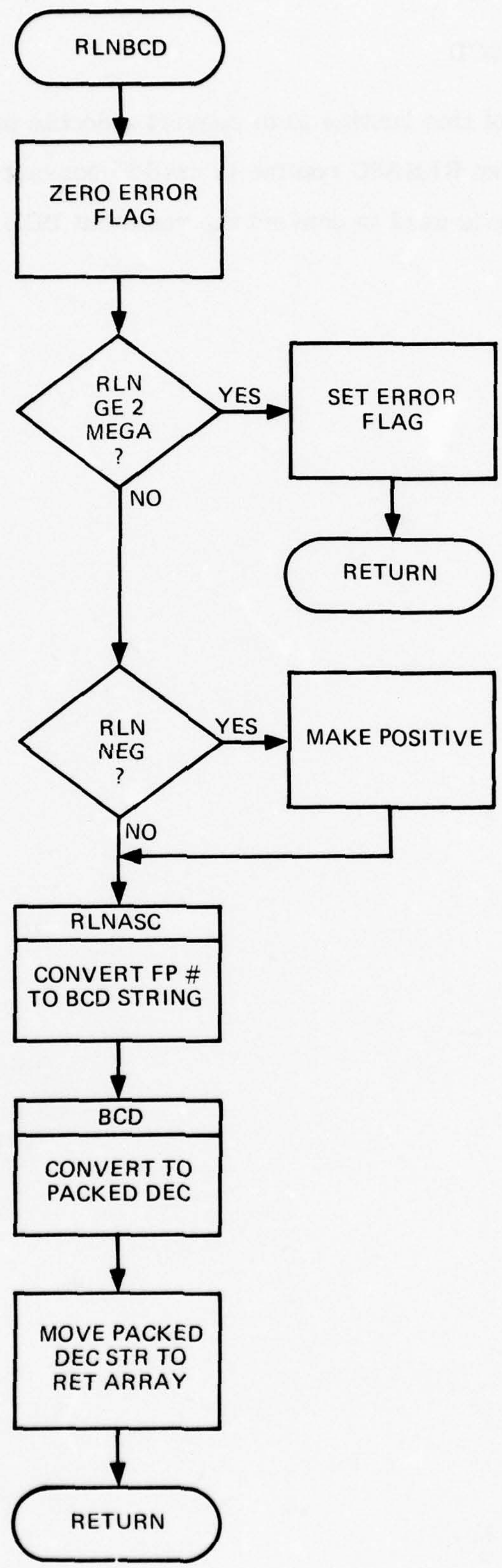

F11
SET ERROR FLAG
RETURN

```
      SUBROUTINE NUMBER(ISTART,IEND,IVAL,IERR)
C***********************************************************************
C
C**** THE PRIMARY PURPOSE OF THIS ROUTINE IS TO BUILD AN 11 BYTE BCD
C**** ARRAY FROM ASCII INPUT. THE ROUTINE ALSO VERIFIES THAT NO
C**** CHARACTERS OTHER THAN SIGN,DECIMAL POINT,COMMA,OR DIGITS ARE
C**** FOUND IN THE INPUT STRING. THE BEGINNING AND END OF THE STRING
C**** ARE PROVIDED BY THE ISTART AND IEND PARAMETERS.
C**** ZERO RETURNED ARRAY
C
C***********************************************************************
      COMMON/INBUF/INLINE
      BYTE CHARS(20)
      COMMON/SPCDEC/CHARS,ICAR
      BYTE INLINE(20),IVAL(12),WORK(20),SIGNA(1)
      IERR=0
      DO 10 I=1,12
      IVAL(I)=0
 10   CONTINUE
C**** DETERMINE IF AND WHERE A SIGN IS SPECIFIED
      DO 20 I=ISTART,IEND
C**** CHECK FOR POSITIVE SIGN
      IF(INLINE(I).EQ.CHARS(3))GO TO 25
C**** CHECK FOR NEGATIVE SIGN
      IF(INLINE(I).EQ.CHARS(4))GO TO 30
 20   CONTINUE
C**** NO SIGN-ASSUME POSITIVE
      SIGNA(1)=0
      NUMST=ISTART
      GO TO 40
 25   SIGNA(1)=0
      NUMST=I+1
      GO TO 40
C**** PUT NEGATIVE SIGN INTO RETURNED ARRAY
 30   SIGNA(1)=1
      NUMST=I+1
 40   K=0
C**** VERIFY NUMERICS,CAR RETN,COMMAS,OR DECIMAL POINT
      DO 100 I=NUMST,IEND
C**** CHECK CHARACTER IS NUMERIC(CHAR,LE,9,AND,GE,0)
      IF(INLINE(I).GE.CHARS(1).AND.INLINE(I).LE.CHARS(2))GO TO 90
C**** CHECK IF DECIMAL POINT THERE
      IF(INLINE(I).EQ.CHARS(5))GO TO 200
C**** CHECK IF COMMA IN STRING
      IF(INLINE(I).EQ.CHARS(6))GO TO 100
C**** CHECK FOR CAR RETN
      IF(INLINE(I).EQ.ICAR)GO TO 110
C**** IF CHAR NOT A BLANK THAN BAD CHARACTER
      IF(INLINE(I).NE.CHARS(7))GO TO 300
C**** BLANK BECOMES A ZERO
      INLINE(I)=0
C**** SAVE NUMERIC FOUND IN WORK ARRAY
 90   K=K+1
      WORK(K)=INLINE(I)
 100  CONTINUE
```

```
C**** INSERT THE WHOLE NUMBER PORTION OF THE NUMBER
 110  L=8
C**** INSERT SIGN INTO RETURNED ARRAY
      IVAL(1)=SIGNA(1)
 120  IF(K.EQ.2)RETURN
C**** DETERMINE BINARY VALUE OF ASCII DIGIT
      DO 57 KK=1,10
C**** ASCII DIGITS CONTAINED IN COMMON
      IF(WORK(K).NE.CHARS(KK+6))GO TO 57
      KB=KK-1
      IVAL(L)=KB
 57   CONTINUE
      K=K-1
      L=L-1
      IF(L.GT.1)GO TO 120
      RETURN
C**** INSERT FRACTIONAL PART OF NUMBER IF SPECIFIED
 200  L=9
      I=I+1
      IF(I.GT.IEND)GO TO 110
      DO 250 J=I,IEND
C**** CHECK FOR CAR RETN
      IF(INLINE(J).EQ.ICAR)GO TO 110
C**** BLANK BECOMES A ZERO
      IF(INLINE(J).EQ.CHARS(7))INLINE(J)=CHARS(1)
C**** CHECK FOR NON NUMERIC
      IF(INLINE(J).LT.CHARS(1).OR.INLINE(J).GT.CHARS(2))GO TO 300
C**** DETERMINE BINARY VALUE OF ASCII DIGIT
      DO 223 KK=1,10
      IF(INLINE(J).NE.CHARS(KK+8))GO TO 223
      KB=KK-1
C**** INSERT FRACTIONAL DIGIT
      IVAL(L)=KB
 223  CONTINUE
      L=L+1
      IF(L.GT.12)GO TO 110
 250  CONTINUE
      GO TO 110
 300  CONTINUE
C**** BAD CHARACTER FOUND IN INPUT STRING
      IERR=1
      RETURN
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
NUMBER  437     (001552)*
INBUF   10      (000024)
SPCDEC  11      (000026)

**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES  00622 02344
EXECUTABLES   00762 02074
ASSEMBLY      01267 06339
```

SUBROUTINE RLNBCD

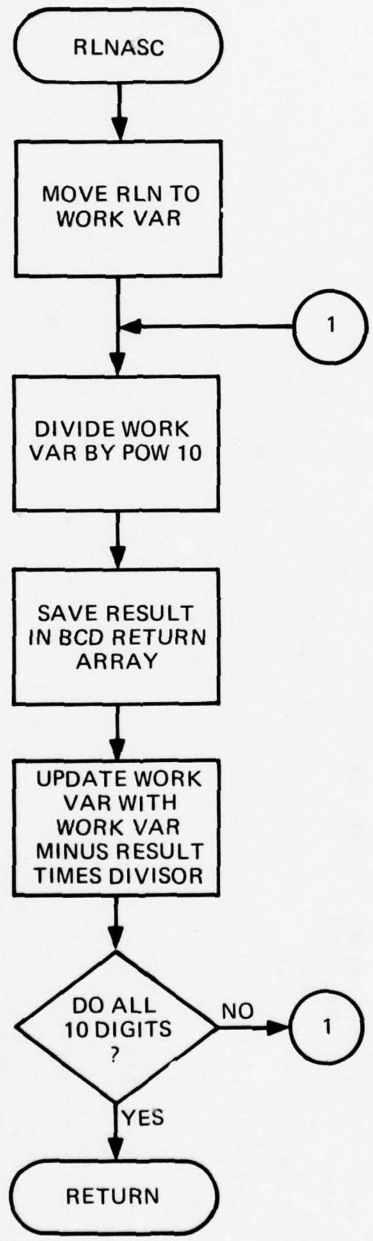
The purpose of this routine is to convert a double precision number to packed decimal format. The RLNASC routine is used to convert the number to a BCD string and the BCD routine is used to convert the resultant BCD string to packed decimal.


RLNBCD
ZERO ERROR FLAG
RLN GE 2 MEGA ?
YES
SET ERROR FLAG
RETURN
NO
RLN NEG ?
YES
MAKE POSITIVE
NO
RLNASC
CONVERT FP # TO BCD STRING
BCD
CONVERT TO PACKED DEC
MOVE PACKED DEC STR TO RET ARRAY
RETURN

```
FORTRAN V06.13                  00:14:23     21-JAN-76    PAGE     1

        SUBROUTINE RLNBCD(RLN,BCDNUM,IERR)
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO CONVERT A DOUBLE
C**** PRECISION REAL NUMBER TO THE FIVE BYTE BCD FORMAT
C**** C**** USED BY THE INTELL,
C
C***********************************************************************
        BYTE WORK(5)
        DOUBLE PRECISION RLN,HOLD
        BYTE MIGN(1),BCDNUM(5),IVAL(12)
        IERR=0
C**** CHECK FOR NUMBER OUT OF RANGE
        IF(RLN.GE.2000000000.)GO TO 200
C**** MAKE SIGN POSITIVE
        MIGN(1)=0
        HOLD=RLN
C**** CHECK FOR NEGATIVE NUMBER
        IF(HOLD.GE.0.)GO TO 10
C**** NEG NUM-MAKE SIGN NEG AND NUM POSITIVE
        MIGN(1)=1
        HOLD=-HOLD
 10     CONTINUE
C**** CONVERT FP NUM TO BCD STRING
        CALL RLNASC(HOLD,IVAL)
C**** PUT SIGN IN BCD STRING
        IVAL(1)=MIGN(1)
C**** CONVERT BCD STRING TO INTELL FORM
        CALL BCD(IVAL,WORK)
C**** RETURN THE FIVE BYTE STRING TO THE USER ARRAY
        DO 20 K=1,5
        BCDNUM(K)=WORK(K)
 20     CONTINUE
        RETURN
C**** FP NUMBER IS OUT OF RANGE
 200    IERR=1
        RETURN
        END

ROUTINES CALLED:
RLNASC, BCD

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
RLNBCD  136    (000420)*

**COMPILER ----- CORE**
   PHASE        USED  FREE
DECLARATIVES 00622 02344
EXECUTABLES  00783 02183
ASSEMBLY     01029 06577
```

SUBROUTINE RLNASC

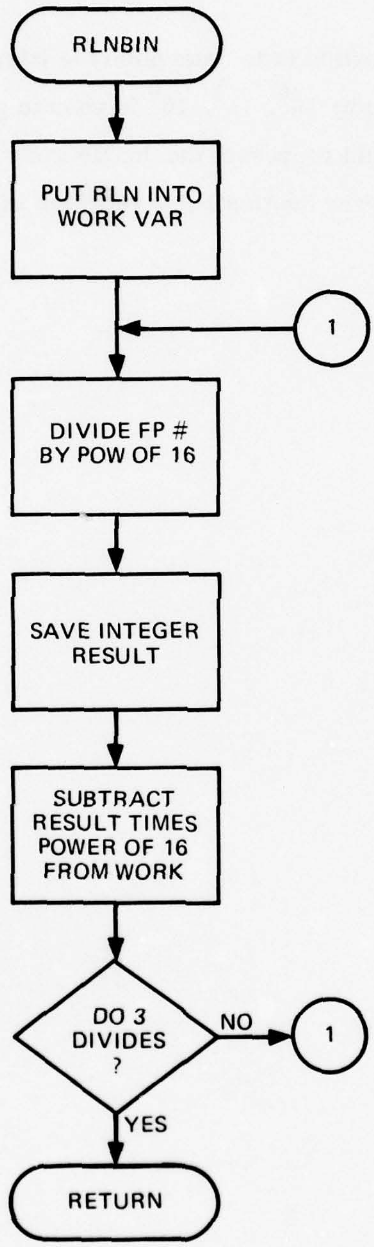
The purpose of this routine is to convert a double precision number to the BCD representation described in the description of the NUMBER subroutine. The BCD string is generated by a series of divisions of the number by powers of 10 and subtractions of the quotient times the power of 10. Each quotient value, when converted to integer, represents one of the digits of the BCD representation of the number.


RLNASC
MOVE RLN TO WORK VAR
1
DIVIDE WORK VAR BY POW 10
SAVE RESULT IN BCD RETURN ARRAY
UPDATE WORK VAR WITH WORK VAR MINUS RESULT TIMES DIVISOR
DO ALL 10 DIGITS ?
NO
1
YES
RETURN

```
FORTRAN V05.15                    00:14:11     21-JAN-76    PAGE    1

      SUBROUTINE RLNASC(DPRLN,ASCII)
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO CONVERT A DOUBLE
C**** PRECISION FLOATING POINT NUMBER TO A BCD STRING.
C
C***********************************************************************
      DOUBLE PRECISION DPRLN,HOLD,DIVIS(10)
      BYTE ASCII(12)
      DATA DIVIS/1.0D9,1.0D8,1.0D7,1.0D6,1.0D5,1.0D4,1.0D3,
     11.0D2,1.0D1,1.0D0/
      HOLD=DPRLN
      DO 10 I=1,10
C**** DIVIDE FP NUM BY POWER OF TEN
      ITEMP=HOLD/DIVIS(I)
C**** SUTRACT OUT QUOTIENT TIMES POW OF TEN
      HOLD=HOLD-(DIVIS(I)*ITEMP)
C**** SAVE BCD DIGIT FOR CALLER
      ASCII(I+1)=ITEMP
 10   CONTINUE
C**** MAKE SIGN POSITIVE
      ASCII(1)=0
      RETURN
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK       LENGTH
RLNASC  119    (000356)*

**COMPILER ----- CORE**
   PHASE       USED  FREE
DECLARATIVES 00622 02344
EXECUTABLES  00783 02183
ASSEMBLY     00965 06641
```

SUBROUTINE RLNBIN

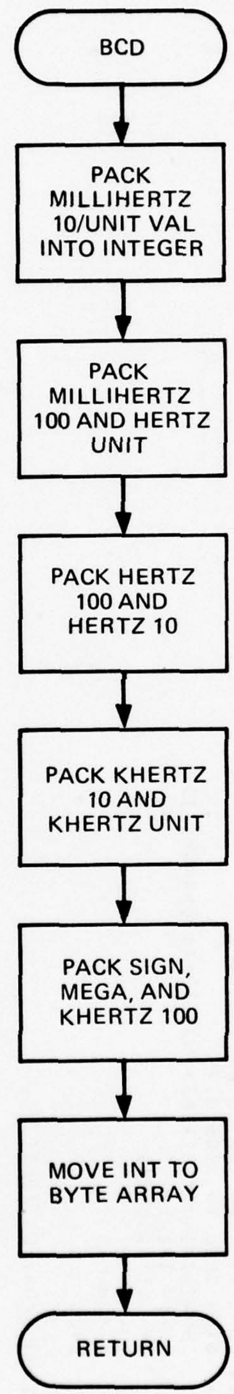
The purpose of this routine is to determine the integer equivalent of a double precision number. Division by 16^6, 16^4, 16^0 is used to generate integer quotients that if concatenated as bytes would represent the double word integer equivalent of the double precision number. The integer quotients are returned in a 4 element integer array.


RLNBIN
PUT RLN INTO WORK VAR
1
DIVIDE FP # BY POW OF 16
SAVE INTEGER RESULT
SUBTRACT RESULT TIMES POWER OF 16 FROM WORK
DO 3 DIVIDES ?
NO
1
YES
RETURN

```
FORTRAN V06.13                 00:14:41     21-JAN-76    PAGE    1

      SUBROUTINE RLNBIN(RLN,BIN)
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO CONVERT A DOUBLE PRECISION
C**** NUMBER TO BINARY. THE ROUTINE RETURNS A 4 WORD INTEGER ARRAY
C**** WHOSE LOW BYTE VALUES WHEN CONCATENATED TOGETHER WILL GIVE THE
C**** BINARY EQUIVALENT OF THE FP NUMBER.
C
C***********************************************************************
      DOUBLE PRECISION RLN,HOLD
      INTEGER BIN(4)
      DOUBLE PRECISION DIVIS(4)
      DOUBLE PRECISION HEX
      INTEGER ITEMP(2)
      BYTE BYT(4)
      EQUIVALENCE (BYT(1),ITEMP(1))
C**** PUT FP NUMBER INTO WORK VARIABLE
      HOLD=RLN
      HEX=16.
      J=6
      DO 20 I=1,4
C**** GENERATE DIVISOR
      DIVIS(I)=HEX**J
C**** DIVIDE TO GENERATE QUOTIENT IN 0-255 RANGE
      JTEMP=HOLD/DIVIS(I)
      J=J-2
C**** SUBTRACT INTEGER RESULT TIMES DIVISOR
      HOLD=HOLD-(JTEMP*DIVIS(I))
C**** SAVE REMAINDER
      BIN(I)=JTEMP
 20   CONTINUE
      RETURN
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK      LENGTH
RLNBIN  130    (000404)*

**COMPILER ----- CORE**
   PHASE       USED  FREE
DECLARATIVES 00522 02344
EXECUTABLES  00793 02173
ASSEMBLY     01421 06585
```

SUBROUTINE BCD

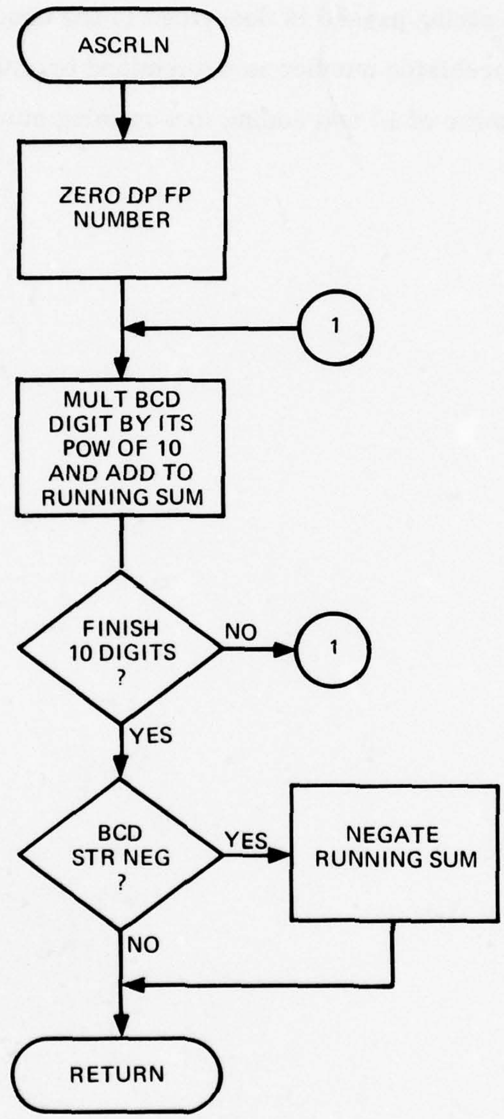
The purpose of this routine is to convert a BCD string to five packed decimal bytes. The format of the BCD string is described with the description of the NUMBER subroutine. A description of the packed decimal form is contained in the real-time mode section.


BCD
PACK MILLIHERTZ 10/UNIT VAL INTO INTEGER
PACK MILLIHERTZ 100 AND HERTZ UNIT
PACK HERTZ 100 AND HERTZ 10
PACK KHERTZ 10 AND KHERTZ UNIT
PACK SIGN, MEGA, AND KHERTZ 100
MOVE INT TO BYTE ARRAY
RETURN

```
FORTRAN V06.13                 00:14:54     21-JAN-76    PAGE    1

      SUBROUTINE BCD(ANUM,BVAL)
C***********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO CONVERT AN 11 BYTE BCD STRING
C**** INTO A COMPRESSED 5 BYTE BCD STRING.
C
C***********************************************************************
      BYTE ANUM(12),BVAL(6)
      INTEGER INUM(6)
C**** CONVERT MILLIHERTZ
      INUM(1)=(ANUM(10)*16)+ANUM(11)
C**** CONVERT MILLI 100 AND HERTZ UNITS
      INUM(2)=(ANUM(8)*15)+ANUM(9)
C**** CONVERT HERTZ 100 AND 10
      INUM(3)=(ANUM(6)*16)+ANUM(7)
C**** CONVERT KHERTZ 10 AND UNITS
      INUM(4)=(ANUM(4)*16)+ANUM(5)
C**** CONVERT KHERTZ 100 TO RIGHT NIBBLE
C**** CONVERT MEGA VAL TO LEFT NIBBLE
C**** SIGN GOES IN SIGN BIT
      INUM(5)=(ANUM(2)*16)+ANUM(3)+(128*ANUM(1))
      DO 10 I=1,5
      BVAL(I)=INUM(I)
 10   CONTINUE
      RETURN
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK        LENGTH
BCD     154    (000464)*

**COMPILER ----- CORE**
   PHASE       USED  FREE
DECLARATIVES 00622 02344
EXECUTABLES  00863 02103
ASSEMBLY     00977 06629
```

SUBROUTINE ASCRLN

The purpose of this routine is to perform BCD to double precision conversion. The format of the BCD string passed is described in the description of the NUMBER subroutine. A double precision number is determined by multiplying each BCD digit by its corresponding power of 10 and adding to a running sum.


ASCRLN
ZERO DP FP NUMBER
1
MULT BCD DIGIT BY ITS POW OF 10 AND ADD TO RUNNING SUM
FINISH 10 DIGITS ?
NO
1
YES
BCD STR NEG ?
YES
NEGATE RUNNING SUM
NO
RETURN

```
FORTRAN V06.15                    00:13:58     21-JAN-76    PAGE    1

      SUBROUTINE ASCRLN(ASCII,RLN)
C**********************************************************************
C
C**** THE PURPOSE OF THIS ROUTINE IS TO CONVERY AN BCD
C**** NUMERIC STRING TO A FLOATING POINT DOUBLE PRECISION
C**** NUMBER, SUCCESSIVE MULTIPLICATIONS BY THE APPROPRIATE
C**** POWER OF TEN AND ADDITIONS TO A RUNNING SUM WILL
C**** YIELD THE DESIRED FLOATING POINT NUMBER.
C
C**********************************************************************
      DOUBLE PRECISION RMULT(10),RLN
      BYTE ASCII(12)
      DATA RMULT/1.0D9,1.0D8,1.0D7,1.0D6,1.0D5,1.0D4,1.0D3,
     11.0D2,1.0D1,1.0D0/
C**** ZERO FLOATING POINT RESULT
      RLN=0.
      DO 10 I=1,10
C**** ADD TO SUM BCD DIGIT TIMES ITS POWER OF TEN
      RLN=RLN+(RMULT(I)*ASCII(I+1))
 10   CONTINUE
C**** CHECK IF BCD STRING IS NEGATIVE
      J=ASCII(1)
      IF(J.EQ.1)RLN=-RLN
      RETURN
      END

OPTIONS =/ON,/SU,/OP:1

BLOCK       LENGTH
ASCRLN  131    (000406)*

**COMPILER ----- CORE**
   PHASE      USED  FREE
DECLARATIVES 00622 02344
EXECUTABLES  00783 02183
ASSEMBLY     00989 06617
```

REAL-TIME MODULE

The real-time module is responsible for loading the main program and reader routine into RAM, performing the system parameter and pattern value table input from PDP-11, and executing the real-time mode. The module is composed of a main program that executes the real-time mode, a reader program that performs the input from the PDP-11, and a loader program to load the main and reader programs.

Detailed descriptions of each program follow.

REAL-TIME MAIN

This program is designed to output a random frequency to the Rockland synthesizer in real-time. A cycle rate of approximately 2.45 milliseconds has been obtained and is maintained in software by delay loops when necessary. Since no internal hardware clock accessible to software is present in the Intellec, this is the only feasible method of maintaining a fairly uniform cycle rate.

The system parameter tables and pattern value tables for the HOP, MFSK, and Doppler effects are obtained from the PDP-11 via a down line load process using the DL11 interface and Intellec serial input/outputs. The format of the system parameter table is shown in Figure 8.

In order to simulate current time, the program maintains a counter that is updated after each output to the Rockland frequency synthesizer. This counter is initialized to zero at the start of the real-time run. There are also three counters for each of the three effect types that are initialized in static mode and updated by their cycle rate each time a corresponding effect pattern value is used in computing an output frequency for the Rockland. During each cycle, all three effect counters are compared with the value of the simulation counter. If the effect counter value is less than or equal to the simulation counter value, than a value for that effect is used in computing the Rockland frequency. When no effect types satisfy this requirement, only the base frequency is output.

Both the base frequency and pattern value tables are stored in packed decimal form. The format of this storage is given in Figure 9. A running sum, also in packed decimal form, is initialized at the beginning of each cycle period to that of the base frequency. Effect values are added or subtracted from the current running sum value using BCD arithmetic. The addition and subtraction routines used have been programmed using straightline repetitive code to minimize the time required for these time consuming operations. A delay loop is used with the addition routine to account for the longer time required for the subtraction process.

ADR		TYPE	LENGTH
20	SIMULATION CURRENT CYCLE	BINARY	3
23	BASE FREQUENCY	PACKED DEC	5
28	RUNNING SUM	PACKED DEC	5
2D	SIMULATION STOP VALUE	BINARY	3
30	HOP CURRENT CYCLE	BINARY	3
33	RANDOM NUMBER INDICATOR (FF)	BINARY	1
34	HOP RANDOM NUMBER MASK	BINARY	1
35	HOP RANDOM NUMBER SEED	BINARY	3
38	HOP PATTERN TABLE LOC	BINARY	2
3A	HOP CYCLE RATE	BINARY	2
3C	HOP STOP VALUE	BINARY	3
3F	MFSK CURRENT CYCLE	BINARY	3
42	RANDOM NUMBER INDICATOR (FF)	BINARY	1
43	MFSK RANDOM NUMBER MASK	BINARY	1
44	MFSK RANDOM NUMBER SEED	BINARY	3
47	MFSK PATTERN TABLE LOC	BINARY	2
49	MFSK CYCLE RATE	BINARY	2
4B	MFSK STOP VALUE	BINARY	3
4E	DOP CURRENT CYCLE	BINARY	3
52	DOP TABLE INDEX	BINARY	2
55	NOT USED		
58	DOP PATTERN TABLE LOC	BINARY	2
5A	DOP CYCLE RATE	BINARY	2
5C	DOP STOP VALUE	BINARY	3

*All fields are in LSB to MSB order

Figure 8. System Parameter Table

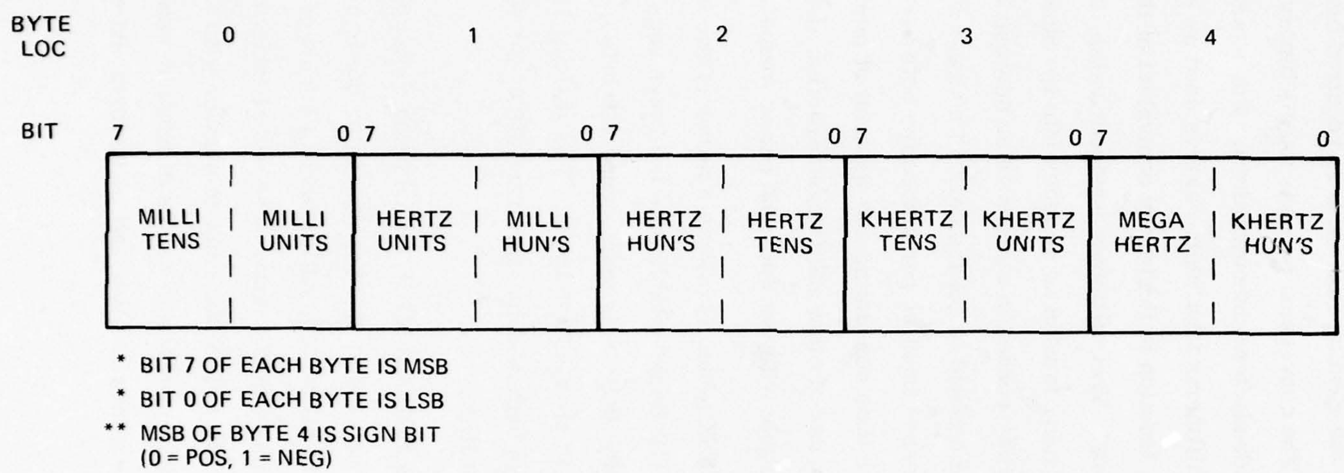


Figure 9. Pattern Storage Format

Since the output format required by the Rockland, see Figure 10, is binary, not packed decimal, a conversion of the running sum value is required before output to the synthesizer can be performed. The conversion process uses a binary equivalent table in tens/units, hundreds, and hundreds/tens denominations. For example, byte one of the running sum containing the millihertz tens/units value is used as an index into the tens/units tables. Stored at this location is the binary equivalent of the tens/units packed decimal value specified in byte one. The millihertz hundreds value, stored in the lower nibble of byte two of the running sum, is used as an index into the hundreds equivalent table. Since two bytes are generally needed to store a value between 0 and 900, the millihertz hundreds digit must be doubled to get the correct address of the two byte binary equivalent. Similar processes must be performed for both hertz and kilohertz. The conversion process, although time consuming and wasteful of memory, has been found to be more economical than use of other addition/subtraction algorithms that generate a running sum more compatible with the Rockland output format.

The size of the HOP and MFSK tables is always a power of two up to a maximum of 128 for HOP and 64 for MFSK. The Doppler table size is always equal to 1024. Allowing for a fixed five bytes per value, the HOP table may occupy 640 bytes, the MFSK table 320 bytes, and the Doppler table is always 5120 bytes. The starting location for each table is not variable. HOP always begins at location 1A00 HEX, MFSK at location 1000 HEX, and Doppler at location 600 HEX.

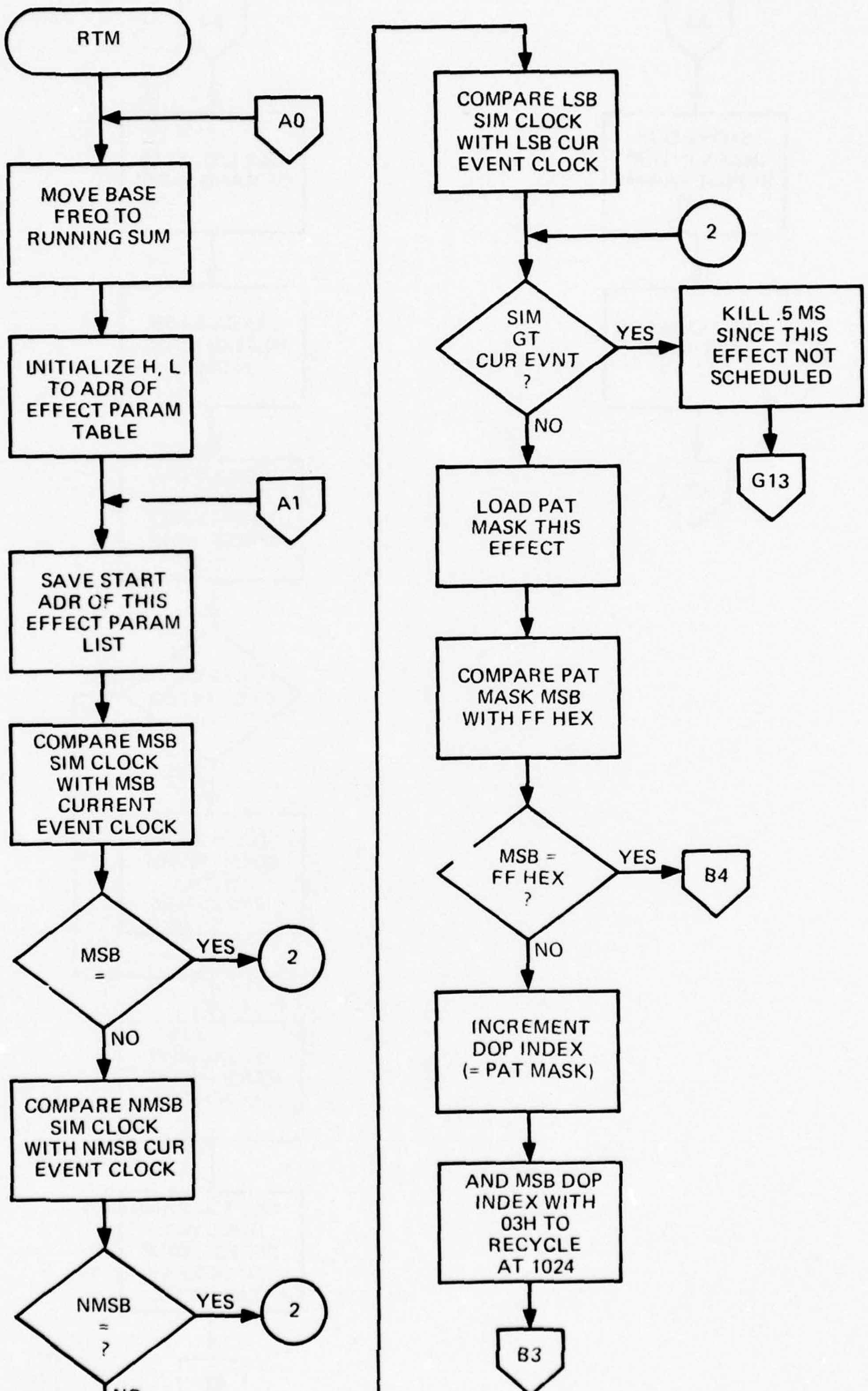
Values are chosen from the HOP and MFSK tables using a pseudorandom number generator for index selection. The random number generator used, a P-N sequence based on a primitive polynomial of degree 24, will provide a series of three byte random numbers that will not repeat for more than 1 hour. The actual table index chosen is gotten by a masking operation on the LSB of the random number with a mask value one less than table size. This technique produces a series of random numbers whose values will range from zero to table size minus one that will be uniform and will repeat each index with equal probability.

	PORT #
MILLIHERTZ LSB (BINARY)	8
MILLIHERTZ MSB (BINARY)	9
HERTZ LSB (BINARY)	10
HERTZ MSB (BINARY)	11
KILOHERTZ LSB (BINARY)	12
KILOHERTZ MSB (BINARY)	13
MEGAHERTZ	14

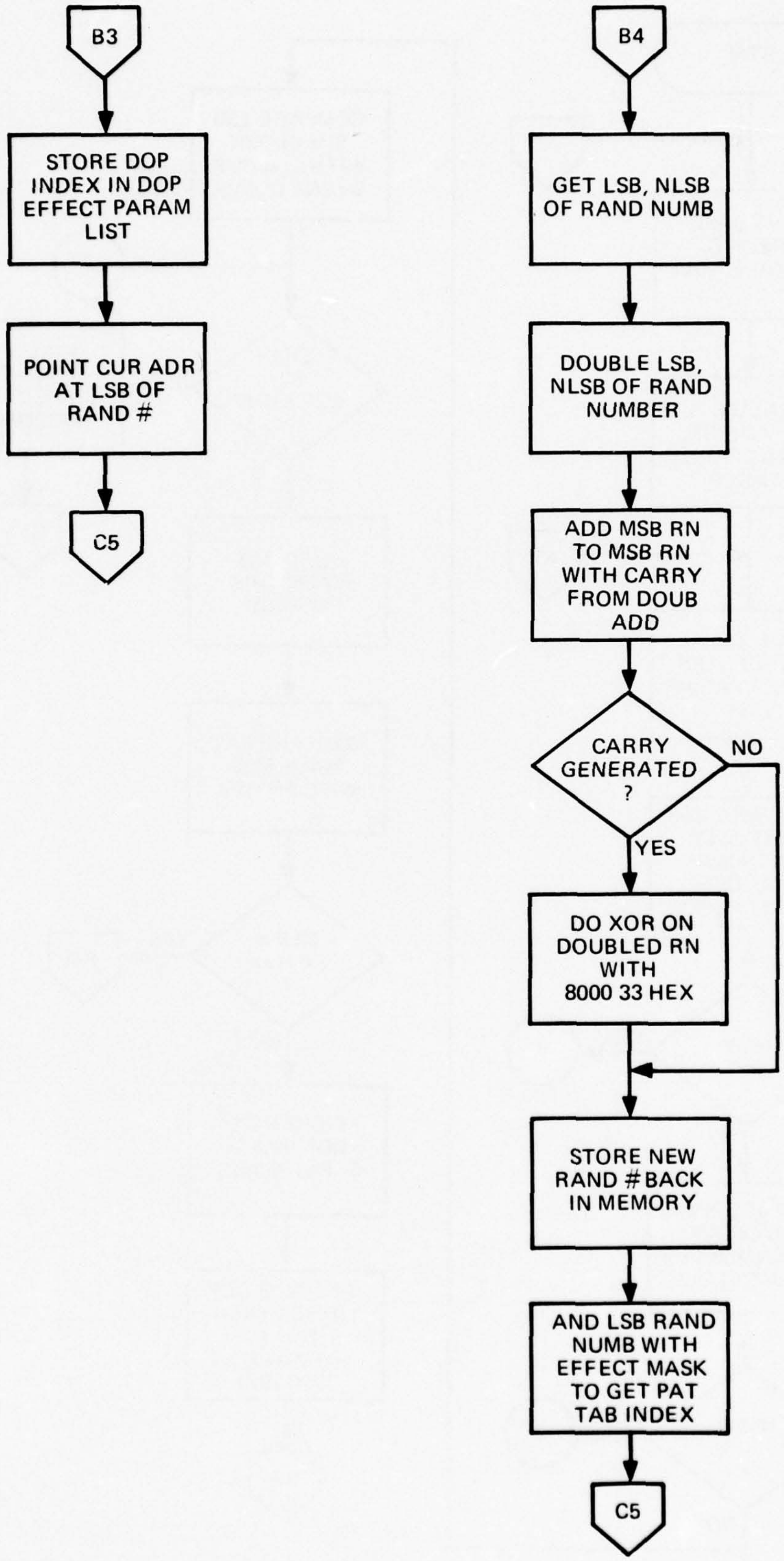
**Binary value of 0 output followed by binary value of 1, both to PORT 15 signals output

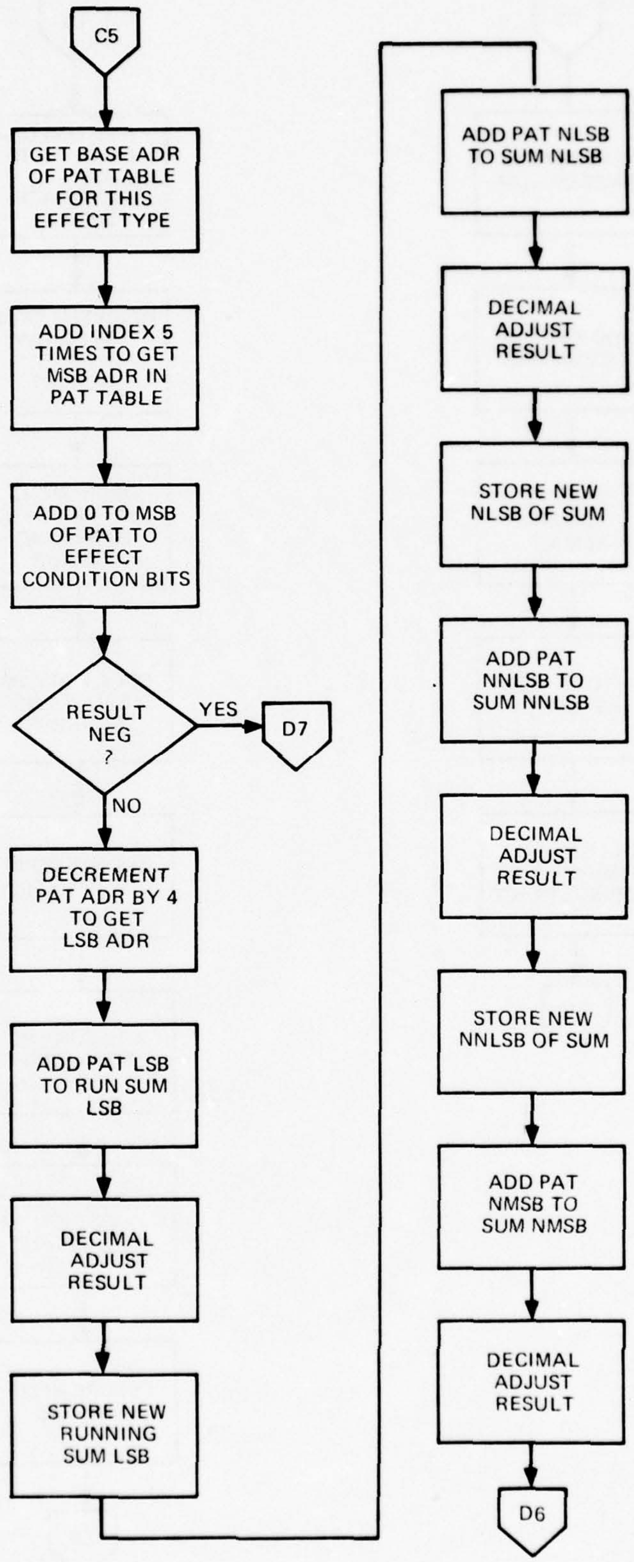
Figure 10. Rockland Interface Format

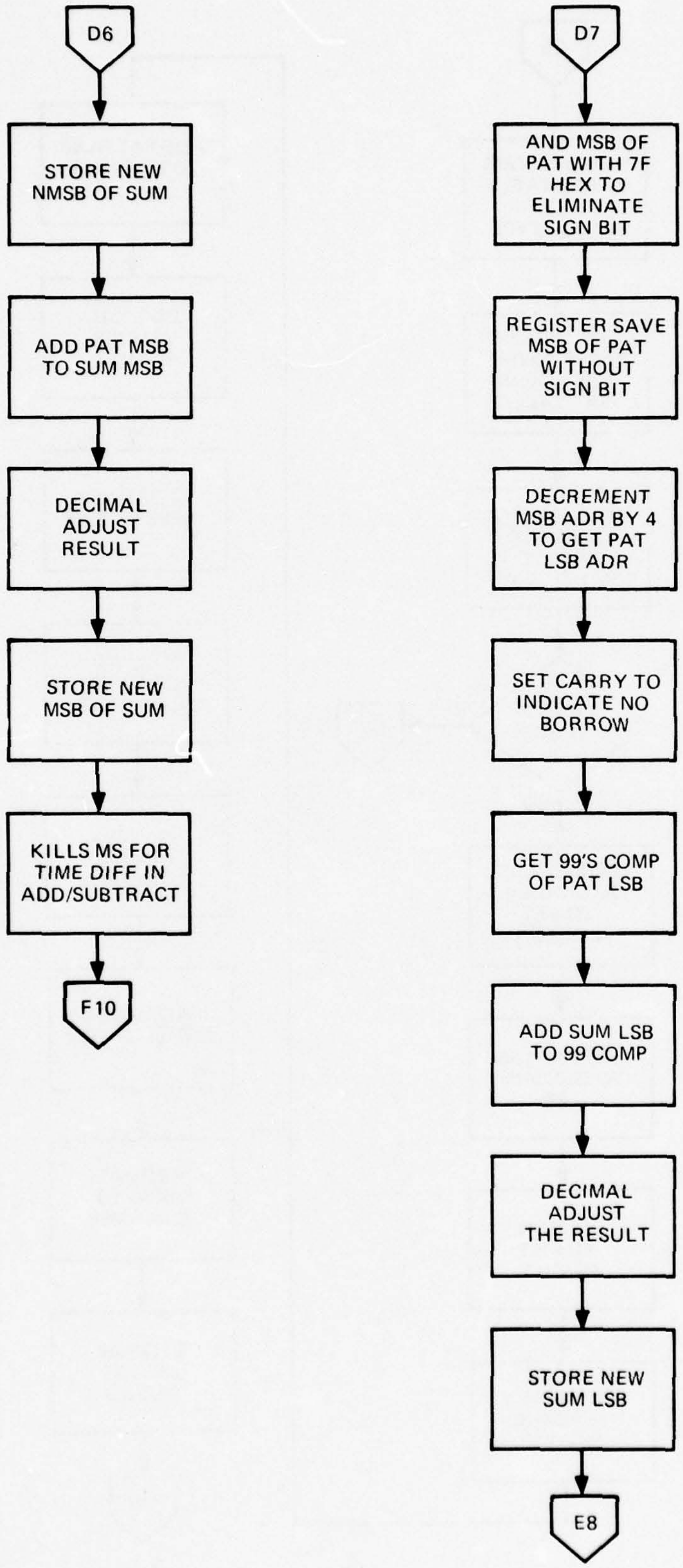
Values are chosen from the Doppler table using an index value that is merely incremented after each selection. When the index value exceeds table size, the index is reset to zero to begin another Doppler cycle.

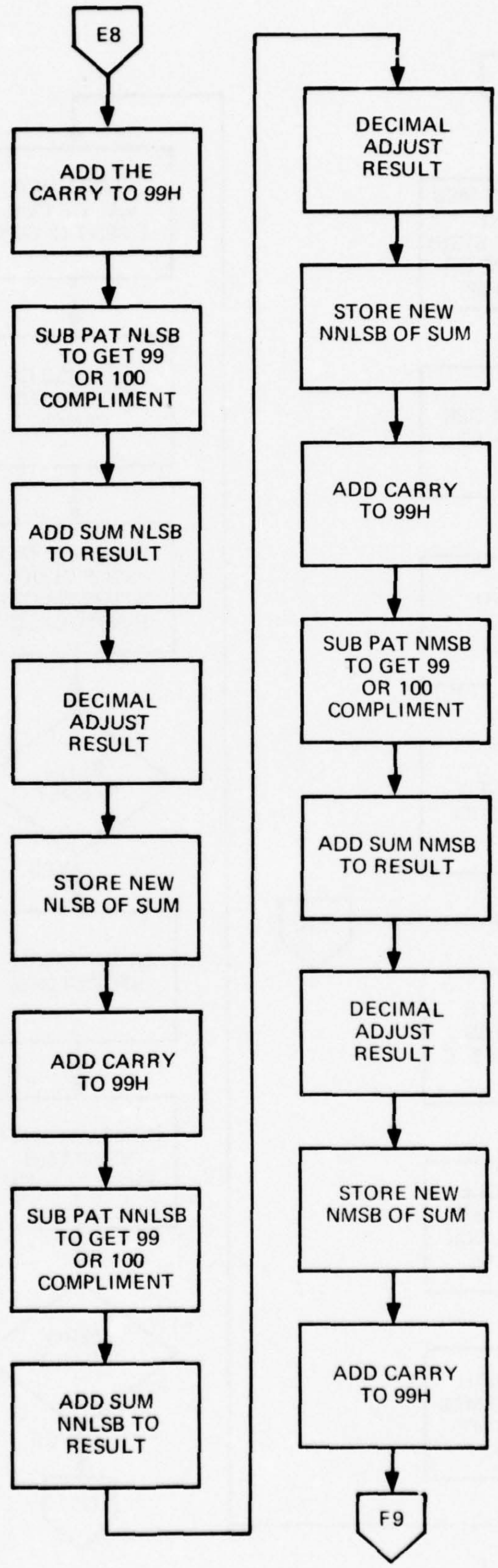


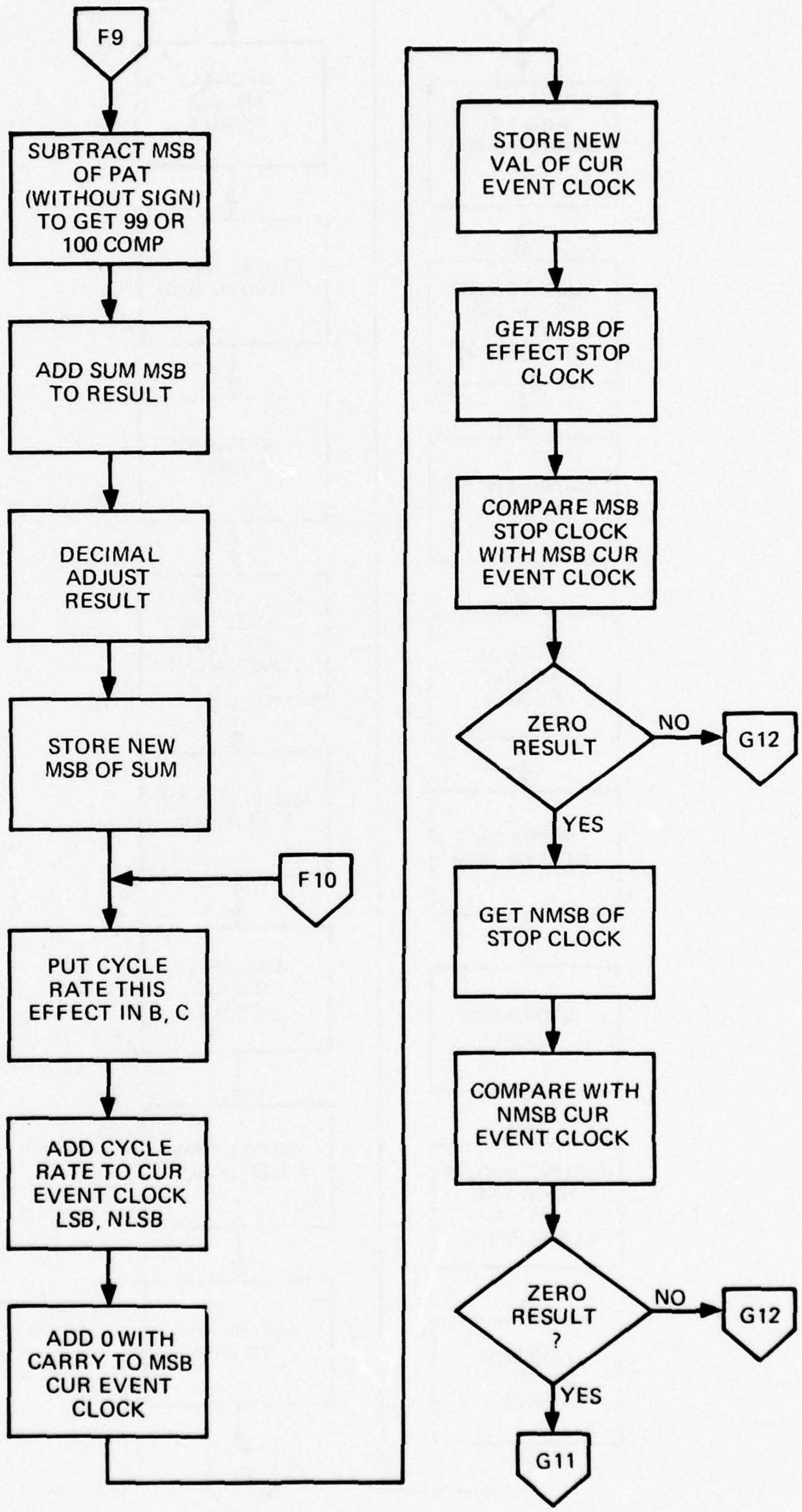
RTM
A0
MOVE BASE FREQ TO RUNNING SUM
INITIALIZE H, L TO ADR OF EFFECT PARAM TABLE
A1
SAVE START ADR OF THIS EFFECT PARAM LIST
COMPARE MSB SIM CLOCK WITH MSB CURRENT EVENT CLOCK
MSB =
YES
2
NO
COMPARE NMSB SIM CLOCK WITH NMSB CUR EVENT CLOCK
NMSB = ?
YES
2
NO
COMPARE LSB SIM CLOCK WITH LSB CUR EVENT CLOCK
2
SIM GT CUR EVNT ?
YES
KILL .5 MS SINCE THIS EFFECT NOT SCHEDULED
G13
NO
LOAD PAT MASK THIS EFFECT
COMPARE PAT MASK MSB WITH FF HEX
MSB = FF HEX ?
YES
B4
NO
INCREMENT DOP INDEX (= PAT MASK)
AND MSB DOP INDEX WITH 03H TO RECYCLE AT 1024
B3

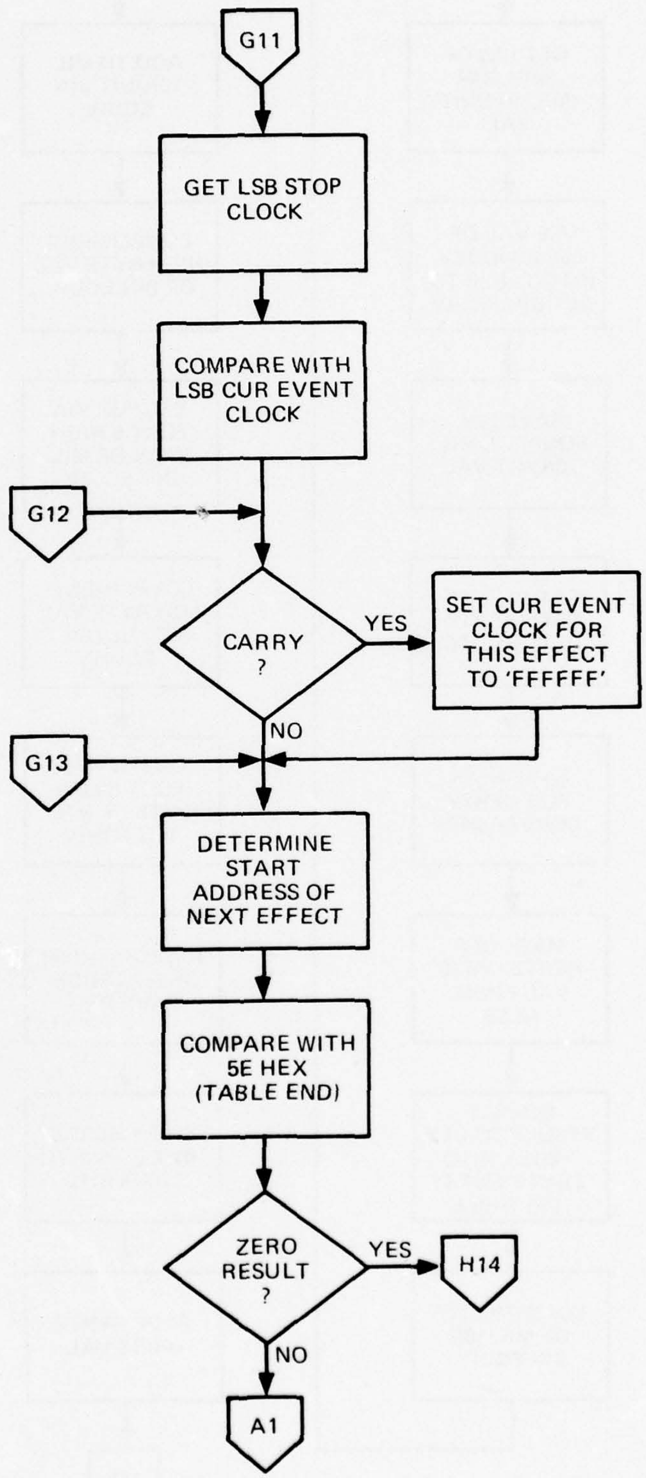

B3
STORE DOP INDEX IN DOP EFFECT PARAM LIST
POINT CUR ADR AT LSB OF RAND #
C5
B4
GET LSB, NLSB OF RAND NUMB
DOUBLE LSB, NLSB OF RAND NUMBER
ADD MSB RN TO MSB RN WITH CARRY FROM DOUB ADD
CARRY GENERATED ?
NO
YES
DO XOR ON DOUBLED RN WITH 8000 33 HEX
STORE NEW RAND # BACK IN MEMORY
AND LSB RAND NUMB WITH EFFECT MASK TO GET PAT TAB INDEX
C5

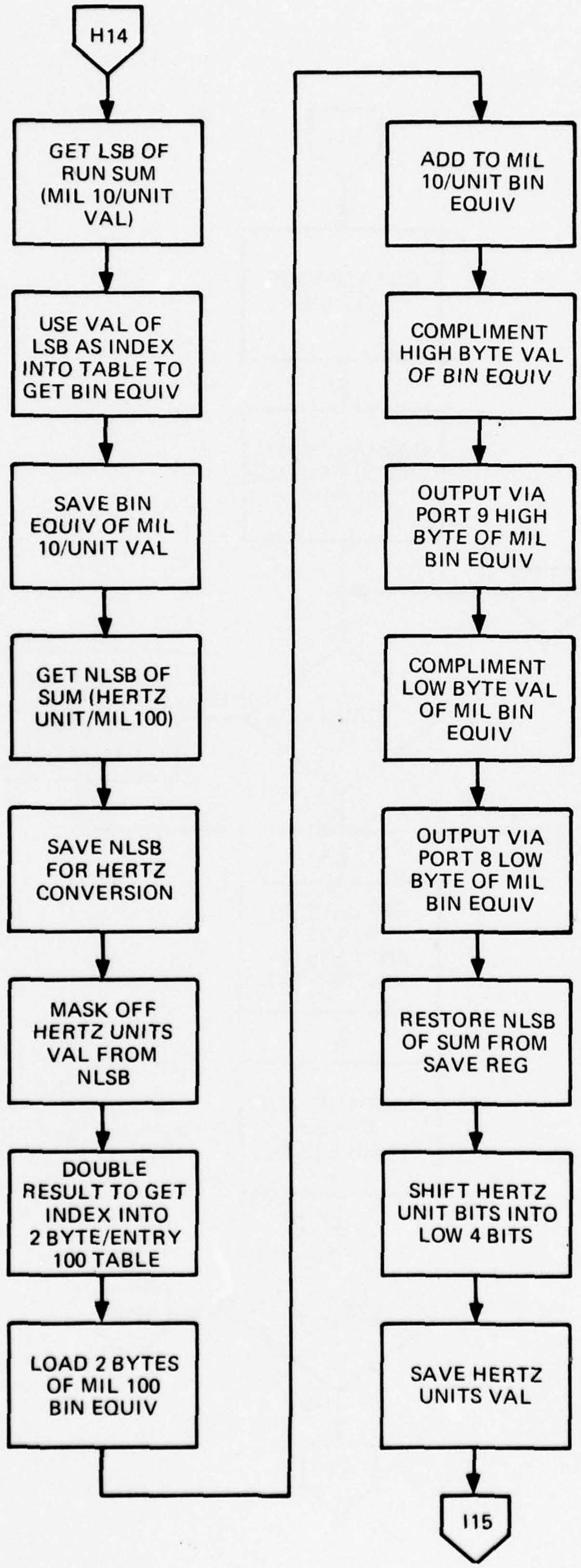

C5
GET BASE ADR OF PAT TABLE FOR THIS EFFECT TYPE
ADD INDEX 5 TIMES TO GET MSB ADR IN PAT TABLE
ADD 0 TO MSB OF PAT TO EFFECT CONDITION BITS
RESULT NEG ?
YES
D7
NO
DECREMENT PAT ADR BY 4 TO GET LSB ADR
ADD PAT LSB TO RUN SUM LSB
DECIMAL ADJUST RESULT
STORE NEW RUNNING SUM LSB
ADD PAT NLSB TO SUM NLSB
DECIMAL ADJUST RESULT
STORE NEW NLSB OF SUM
ADD PAT NNLSB TO SUM NNLSB
DECIMAL ADJUST RESULT
STORE NEW NNLSB OF SUM
ADD PAT NMSB TO SUM NMSB
DECIMAL ADJUST RESULT
D6

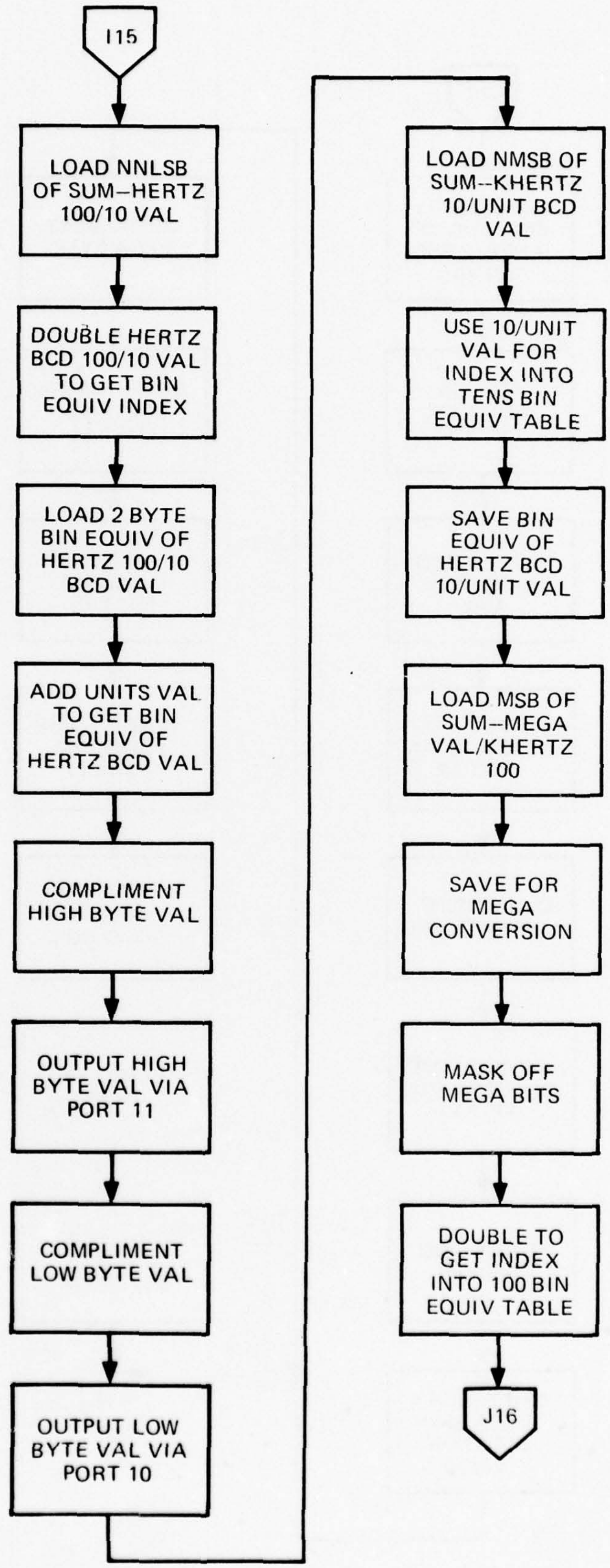

D6
STORE NEW NMSB OF SUM
ADD PAT MSB TO SUM MSB
DECIMAL ADJUST RESULT
STORE NEW MSB OF SUM
KILLS MS FOR TIME DIFF IN ADD/SUBTRACT
F10
D7
AND MSB OF PAT WITH 7F HEX TO ELIMINATE SIGN BIT
REGISTER SAVE MSB OF PAT WITHOUT SIGN BIT
DECREMENT MSB ADR BY 4 TO GET PAT LSB ADR
SET CARRY TO INDICATE NO BORROW
GET 99'S COMP OF PAT LSB
ADD SUM LSB TO 99 COMP
DECIMAL ADJUST THE RESULT
STORE NEW SUM LSB
E8

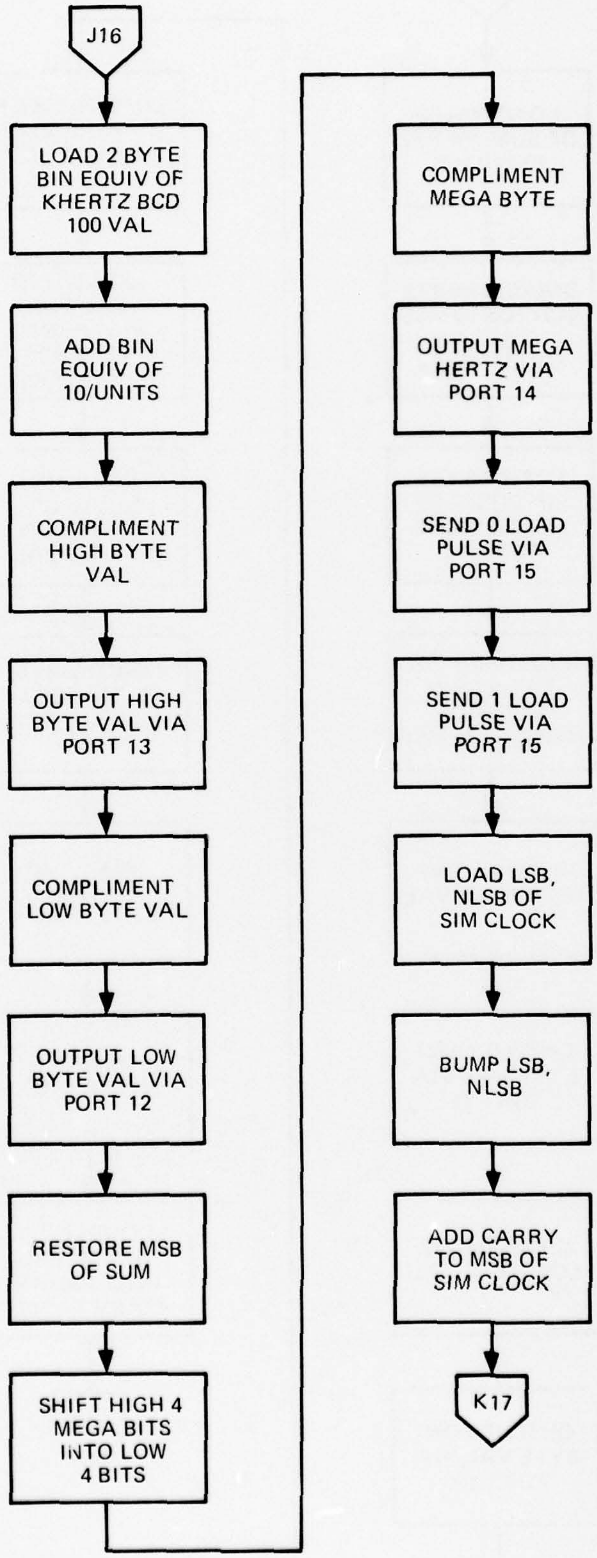

E8
ADD THE CARRY TO 99H
SUB PAT NLSB TO GET 99 OR 100 COMPLIMENT
ADD SUM NLSB TO RESULT
DECIMAL ADJUST RESULT
STORE NEW NLSB OF SUM
ADD CARRY TO 99H
SUB PAT NNLSB TO GET 99 OR 100 COMPLIMENT
ADD SUM NNLSB TO RESULT
DECIMAL ADJUST RESULT
STORE NEW NNLSB OF SUM
ADD CARRY TO 99H
SUB PAT NMSB TO GET 99 OR 100 COMPLIMENT
ADD SUM NMSB TO RESULT
DECIMAL ADJUST RESULT
STORE NEW NMSB OF SUM
ADD CARRY TO 99H
F9

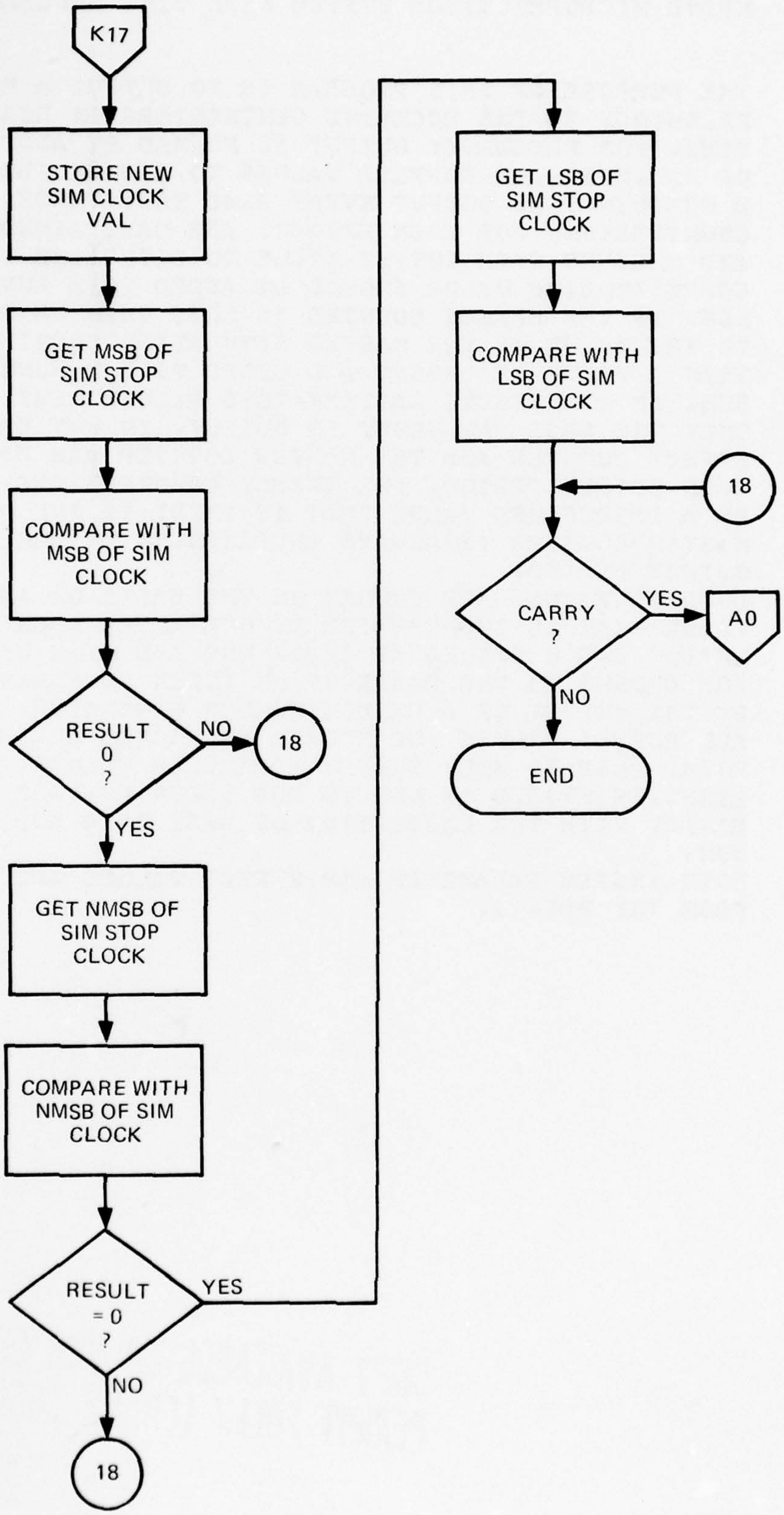

F9
SUBTRACT MSB OF PAT (WITHOUT SIGN) TO GET 99 OR 100 COMP
ADD SUM MSB TO RESULT
DECIMAL ADJUST RESULT
STORE NEW MSB OF SUM
F10
PUT CYCLE RATE THIS EFFECT IN B, C
ADD CYCLE RATE TO CUR EVENT CLOCK LSB, NLSB
ADD 0 WITH CARRY TO MSB CUR EVENT CLOCK
STORE NEW VAL OF CUR EVENT CLOCK
GET MSB OF EFFECT STOP CLOCK
COMPARE MSB STOP CLOCK WITH MSB CUR EVENT CLOCK
ZERO RESULT
NO
G12
YES
GET NMSB OF STOP CLOCK
COMPARE WITH NMSB CUR EVENT CLOCK
ZERO RESULT ?
NO
G12
YES
G11


G11
GET LSB STOP CLOCK
COMPARE WITH LSB CUR EVENT CLOCK
G12
CARRY ?
YES
SET CUR EVENT CLOCK FOR THIS EFFECT TO 'FFFFFF'
NO
G13
DETERMINE START ADDRESS OF NEXT EFFECT
COMPARE WITH 5E HEX (TABLE END)
ZERO RESULT ?
YES
H14
NO
A1


H14
GET LSB OF RUN SUM (MIL 10/UNIT VAL)
USE VAL OF LSB AS INDEX INTO TABLE TO GET BIN EQUIV
SAVE BIN EQUIV OF MIL 10/UNIT VAL
GET NLSB OF SUM (HERTZ UNIT/MIL100)
SAVE NLSB FOR HERTZ CONVERSION
MASK OFF HERTZ UNITS VAL FROM NLSB
DOUBLE RESULT TO GET INDEX INTO 2 BYTE/ENTRY 100 TABLE
LOAD 2 BYTES OF MIL 100 BIN EQUIV
ADD TO MIL 10/UNIT BIN EQUIV
COMPLIMENT HIGH BYTE VAL OF BIN EQUIV
OUTPUT VIA PORT 9 HIGH BYTE OF MIL BIN EQUIV
COMPLIMENT LOW BYTE VAL OF MIL BIN EQUIV
OUTPUT VIA PORT 8 LOW BYTE OF MIL BIN EQUIV
RESTORE NLSB OF SUM FROM SAVE REG
SHIFT HERTZ UNIT BITS INTO LOW 4 BITS
SAVE HERTZ UNITS VAL
I15


I15
LOAD NNLSB OF SUM–HERTZ 100/10 VAL
DOUBLE HERTZ BCD 100/10 VAL TO GET BIN EQUIV INDEX
LOAD 2 BYTE BIN EQUIV OF HERTZ 100/10 BCD VAL
ADD UNITS VAL TO GET BIN EQUIV OF HERTZ BCD VAL
COMPLIMENT HIGH BYTE VAL
OUTPUT HIGH BYTE VAL VIA PORT 11
COMPLIMENT LOW BYTE VAL
OUTPUT LOW BYTE VAL VIA PORT 10
LOAD NMSB OF SUM--KHERTZ 10/UNIT BCD VAL
USE 10/UNIT VAL FOR INDEX INTO TENS BIN EQUIV TABLE
SAVE BIN EQUIV OF HERTZ BCD 10/UNIT VAL
LOAD MSB OF SUM–MEGA VAL/KHERTZ 100
SAVE FOR MEGA CONVERSION
MASK OFF MEGA BITS
DOUBLE TO GET INDEX INTO 100 BIN EQUIV TABLE
J16


J16
LOAD 2 BYTE BIN EQUIV OF KHERTZ BCD 100 VAL
ADD BIN EQUIV OF 10/UNITS
COMPLIMENT HIGH BYTE VAL
OUTPUT HIGH BYTE VAL VIA PORT 13
COMPLIMENT LOW BYTE VAL
OUTPUT LOW BYTE VAL VIA PORT 12
RESTORE MSB OF SUM
SHIFT HIGH 4 MEGA BITS INTO LOW 4 BITS
COMPLIMENT MEGA BYTE
OUTPUT MEGA HERTZ VIA PORT 14
SEND 0 LOAD PULSE VIA PORT 15
SEND 1 LOAD PULSE VIA PORT 15
LOAD LSB, NLSB OF SIM CLOCK
BUMP LSB, NLSB
ADD CARRY TO MSB OF SIM CLOCK
K17


K17
STORE NEW SIM CLOCK VAL
GET MSB OF SIM STOP CLOCK
COMPARE WITH MSB OF SIM CLOCK
RESULT 0 ?
NO
18
YES
GET NMSB OF SIM STOP CLOCK
COMPARE WITH NMSB OF SIM CLOCK
RESULT = 0 ?
YES
NO
18
GET LSB OF SIM STOP CLOCK
COMPARE WITH LSB OF SIM CLOCK
18
CARRY ?
YES
A0
NO
END

```
;****  KBAND MICROPROCESSOR SYSYEM REAL TIME PROGRAM
;
;
;      THE PURPOSE OF THIS PROGRAM IS TO OUTPUT A RANDOM
;      FREQUENCY TO THE ROCKLAND SYNTHESIZER IN REAL
;      TIME. THE FREQUENCY OUTPUT IS FORMED BY ADDITION
;      OF HOP,MFSK,AND DOPPLER VALUES TO A BASE FREQ.
;      A FREQUENCY IS OUTPUT EVERY 2.45 MLS. THREE
;      COUNTERS(ONE FOR EACH EFFECT) ARE MAINTAINED
;      AND CHECKED EACH OUTPUT CYCLE TO DETERMINE IF ITS
;      CORRESPONDING VALUE SHOULD BE ADDED TO A RUNNING
;      SUM. IF THE EFFECT COUNTER IS LESS THAN OR EQUAL
;      TO THE VALUE OF THE MASTER SIMULATION COUNTER
;      THAN A VALUE IS CHOSEN AND ADDED TO THE RUNNING
;      SUM. IF NO EFFECTS SATISFY THIS REQUIREMENT, THAN
;      ONLY THE BASE FREQUENCY IS OUTPUT. IN ANY CASE EA
;      EFFECT COUNTER AND THE MASTER COUNTER ARE UPDATED
;      EACH OUTPUT PERIOD. THE EFFECT COUNTERS ARE UPDAT
;      BY A PREDEFINED VALUE THAT IS INPUT BY THE USER.
;      MASTER COUNTER IS ALWAYS INCREMENTED BY ONE EACH
;      OUTPUT PERIOD.
;      DOPPLER VALUES ARE CHOSEN ON THE BASIS OF AN INDE
;      VALUE THAT IS INCREMENTED BY ONE AFTER EACH SEL-
;      ECTION AND RECYCLED AT 1024. HOP AND MFSK VALUES
;      ARE CHOSEN ON THE BASIS OF AN INDEX DETERMINED
;      BY THE OUTPUT OF A RANDOM NUMBER GENERATOR.
;      ALL EFFECT VALUES ARE STORED IN PACKED DECIMAL
;      FORM, MLSB TO MSB. SYSTEM PARAMETER VALUES ARE
;      LIKEWISE STORED IN LSB TO MSB ORDER BUT ARE ALL I
;      BINARY WITH THE EXECEPTION OF BASE FREQ AND RUNNI
;      SUM.
;      BOTH SYSTEM PARAMETER AND EFFECT VALUES ARE INPUT
;      FROM THE PDP-11.
```

```
                ;****  SYSTEM TABLES-DATA AREA DEFINITIONS
                ;
0000                   ORG     20H
0020            SIM:   DS      3         ;SIMULATION CLOCK
0023            BFREQ: DS      5         ;BASE FREQ
0028            SFREQ: DS      5         ;RUNNING SUM
002D            STOP:  DS      3         ;STOP CLOCK FOR SIMULATION
                ;
                ;
                SYSTB:
0030                   DS      3         ;HOP CUR EVENT CLOCK
0033                   DS      2         ;HOP PATTERN MASK
0035                   DS      3         ;HOP RAND NUMB SEED
0038                   DS      2         ;HOP PAT ADR(1A00)
003A                   DS      2         ;HOP CYCLE RATE
003C                   DS      3         ;HOP STOP TIME
                ;
003F                   DS      3         ;MFSK CUR EVENT CLOCK
0042                   DS      2         ;MFSK PAT MASK
0044                   DS      3         ;MKSK RAND NUMB SEED
0047                   DS      2         ;MFSK PAT ADR(1D00)
0049                   DS      2         ;MKSK CYCLE RATE
004B                   DS      3         ;MKSK STOP TIME
                ;
004E                   DS      3         ;DOP CUR EVENT CLOCK
0051                   DS      2         ;DOP PAT INDEX
0053                   DS      3         ;NOT USED FOR DOP
0056                   DS      2         ;DOP PAT LOCATION(600)
0058                   DS      2         ;DOP CYCLE RATE
005A                   DS      3         ;DOP STOP TIME
005D            ENDAD: DS      1
005E            SAV:   DS      2
0003            TEN    EQU     03        ;HIGH BYTE ADR OF 10/UNIT BIN TAB
0004            HUN    EQU     04        ;HIGH BYTE ADR OF 100/10 BIN TABL
                ;
                ;
                ;****  ESTABLISH RUNNING SUM-SETUP ADDRESSES ETC
                ;
0060 012300     STEP0: LXI     B,BFREQ   ;LOAD ADR OF BASE FREQ
0063 212800            LXI     H,SFREQ   ;LOAD ADR OF RUNNING SUM
0066 1604              MVI     D,4
0068 0A         MVE:   LDAX    B
0069 77                MOV     M,A       ;MOVE BASE BYTE TO SUM
006A 0C                INR     C         ;ADR NEXT BASE BYTE
006B 2C                INR     L         ;ADR NEXT SUM BYTE
006C 15                DCR     D
006D F26800            JP      MVE       ;MOVE ANOTHER BYTE
0070 213000            LXI     H,SYSTB   ;START ADR OF SYS TABLE
```

```
                 ;
                 ;****  DETERMINE IF THIS EFFECT WILL BE INCLUDED
                 ;
0073 54          STEP1: MOV     D,H
0074 5D                 MOV     E,L
0075 225E00             SHLD    SAV       ;SAVE START ADR OF THIS EFFECT
0078 2C                 INR     L         ;ADR NMSB CUR EVENT
0079 2C                 INR     L         ;ADR MSB CUR EVENT
007A 012200             LXI     B,SIM+2   ;ADR MSB OF SIM CLOCK
007D 0A                 LDAX    B         ;LOAD MSB SIM CLOCK
007E BE                 CMP     M         ;COMPARE WITH MSB CUR EVENT CLOCK
007F C28D00             JNZ     CTME
0082 0D                 DCR     C         ;ADR NMSB SIM CLOCK
0083 2D                 DCR     L         ;ADR NMSB CUR CLOCK
0084 0A                 LDAX    B         ;LOAD NMSB SIM CLOCK
0085 BE                 CMP     M         ;COMPARE WITH NMSB CUR EVENT CLOC
0086 C28D00             JNZ     CTME
0089 0D                 DCR     C         ;ADR LSB OF SIM CLOCK
008A 2D                 DCR     L         ;ADR LSB CUR EVENT CLOCK
008B 0A                 LDAX    B         ;LOAD LSB SIM CLOCK
008C BE                 CMP     M         ;COMPARE WITH LSB CUR EVENT CLOCK
008D D29900      CTME:  JNC     STEP2     ;NO CAR-SIM GT = CUR EVENT
                 ;
                 ;****  THIS EFFECT NOT INCLUDED-KILL TIME TAKEN IF EFFEC
                 ;****  WAS INCLUDED
                 ;
0090 3E3C               MVI     A,60
0092 3D          NO:    DCR     A
0093 F29200             JP      NO
0096 C3A901             JMP     OK        ;NOW CHECK IF THIS WAS LAST EFFEC
                 ;
                 ;****  DETERMINE IF DOPPLER EFFECT-IF YES GET DOPPLER IN
                 ;****  OTHERWISE GO GENERATE RANDOM # FOR INDEX VAL
                 ;
0099 EB          STEP2: XCHG              ;START THIS EFFECT WAS IN D,E
009A 2C                 INR     L         ;ADR NMSB CUR EVENT
009B 2C                 INR     L         ;ADR MSB CUR EVENT
009C 2C                 INR     L         ;ADR LSB PAT MASK
009D 4E                 MOV     C,M       ;LOAD LSB PAT MASK
009E 2C                 INR     L         ;ADR MSB PAT MASK
009F 46                 MOV     B,M       ;LOAD MSB PAT MASK
00A0 78                 MOV     A,B
00A1 EEFF               XRI     0FFH      ;CHECK FOR DOP EFFECT-NOT FFH
00A3 CAB200             JZ      STEP3     ;NOT FFH THAN GET RANDOM # INDEX
                 ;
                 ;****  GET DOPLER INDEX AND UPDATE
00A6 03                 INX     B         ;NEXT DOP INDEX
00A7 78                 MOV     A,B
00A8 E603               ANI     03H       ;RECYCLE DOP WHEN IT HITS 1024
00AA 77                 MOV     M,A       ;STORE NEW MSB DOP INDEX
00AB 2D                 DCR     L         ;ADR LSB DOP INDEX
00AC 71                 MOV     M,C       ;STORE LSB DOP INDEX
00AD 2C                 INR     L         ;ADR MSB OF DOP IND
00AE 2C                 INR     L         ;ADR OF LSB OF SEED
[illegible]             JMP     [illegible] ;[illegible]
```

```
                ;
                ;****   RANDOM # INDEX GENERATION FOR HOP AND MFSK
                ;
00B2 2C         STEP3:  INR     L       ;ADR OF LSB OF SEED
00B3 5E                 MOV     E,M     ;LOAD SEED LSB
00B4 2C                 INR     L       ;ADR NLSB SEED
00B5 56                 MOV     D,M     ;LOAD NMSB OF SEED
00B6 2C                 INR     L       ;ADR MSB OF SEED
00B7 7E                 MOV     A,M
00B8 EB                 XCHG            ;ADR TO D,E-VAL TO H,L
00B9 29                 DAD     H       ;DOUBLE LSB AND NLSB
00BA 8F                 ADC     A       ;DOUBLE MSB WITH CARRY FROM DAD
00BB EB                 XCHG            ;VAL TO D,E-ADR BACK TO H,L
00BC D2CA00             JNC     STORE   ;NO CARRY THAN NO XOR NEEDED
00BF EE80               XRI     80H     ;MSB XOR
00C1 47                 MOV     B,A     ;SAVE MSB IN B
00C2 AF                 XRA     A       ;NMSB MASK=0
00C3 AA                 XRA     D       ;NMSB XOR
00C4 57                 MOV     D,A     ;SAVE NMSB IN D
00C5 3E33               MVI     A,33H   ;LSB MASK
00C7 AB                 XRA     E       ;LSB XOR
00C8 5F                 MOV     E,A     ;SAVE LSB IN E
00C9 78                 MOV     A,B     ;MSB BACK TO A
                ;
                ;****   STORE OFF NEW RANDOM NUMBER
                ;
00CA 77         STORE:  MOV     M,A     ;SAVE MSB
00CB 2D                 DCR     L
00CC 72                 MOV     M,D     ;NEW NMSB
00CD 2D                 DCR     L       ;
00CE 73                 MOV     M,E     ;NEW LSB
                ;
                ;****   GET INDEX FROM LSB OF RANDOM # USING MASK FROM TH
                ;
00CF 7B                 MOV     A,E     ;LSB STILL IN E
00D0 A1                 ANA     C       ;MASK WAS IN C FROM STEP 2
00D1 4F                 MOV     C,A     ;SETUP B,C WITH INDEX
00D2 AF                 XRA     A       ;HIGH INDEX VAL IS ZERO
00D3 47                 MOV     B,A
                ;
                ;****   GET BYTE LOC OF FREQ BY MULT OF INDEX BY 5 AND AD
                ;****   TO THE BASE ADRESS OF THE PARTICULAR PATTERN TABL
                ;
00D4 2C         STEP4:  INR     L       ;ADR OF NMSB OF SEED
00D5 2C                 INR     L       ;ADR OF MSB OF SEED
00D6 2C                 INR     L       ;ADR OF LSB PAT LOC
00D7 5E                 MOV     E,M     ;LSB OF PAT BASE ADR
00D8 2C                 INR     L       ;ADR OF MSB OF ADR
00D9 56                 MOV     D,M     ;MSB OF PAT BASE ADR
00DA EB                 XCHG            ;BASE ADR TO H,L-CUR ADR TO D,E
00DB 09                 DAD     B       ;ADD INDEX 5 TIMES TO GET ACT ADR
00DC 09                 DAD     B
00DD 09                 DAD     B
00DE 09                 DAD     B
00DF 09                 DAD     B       ;ADR OF FREQ NOW IN H,L
```

```
                    ;****   BCD ADDITION ROUTINE-SUM+PATTERN
                    ;
00EC C30001                 JMP     CONT      ;JUMP OVER MONITOR STACK AREA(F0-
00E3                        ORG     100H
                    CONT:
0100 1E28                   MVI     E,SFREQ   ;ADR OF RUNNING SUM
0102 7E                     MOV     A,M       ;LOAD MSB OF PATTERN
0103 C600                   ADI     0         ;AFFECT SIGN CONDITION
0105 FA3101                 JM      DSUB      ;SIGN BIT SET-HAVE NEG FREQ
0108 2B                     DCX     H         ;GET LSB OF PATTERN
0109 2B                     DCX     H
010A 2B                     DCX     H
010B 2B                     DCX     H
                    ;****   ADDITION OF LSB BYTES
010C 1A                     LDAX    D         ;LOAD LSB OF SUM
010D 8E                     ADC     M         ;ADD LSB OF PAT TO IT
010E 27                     DAA               ;BCD ADJUST
010F 12                     STAX    D         ;STORE NEW LSB OF SUM
                    ;****   NLSB ADDITION
0110 13                     INX     D         ;ADR OF NLSB OF SUM
0111 23                     INX     H         ;ADR NLSB OF PAT
0112 1A                     LDAX    D         ;LOAD NLSB OF SUM
0113 8E                     ADC     M         ;ADD TO IT PAT NLSB
0114 27                     DAA               ;BCD ADJUST
0115 12                     STAX    D         ;STORE NEW NLSB OF SUM
                    ;****   NNLSB ADDITION
0116 13                     INX     D         ;ADR NNLSB BYTE
0117 23                     INX     H         ;ADR NNLSB OF PAT
0118 1A                     LDAX    D         ;LOAD NNLSB
0119 8E                     ADC     M         ;ADD IN NNLSB OF PAT
011A 27                     DAA               ;BCD ADJUST
011B 12                     STAX    D         ;STORE NEW NNLSB OF SUM
                    ;****   NNNLSB ADDITION
011C 13                     INX     D         ;ADR OF NNNLSB OF SUM
011D 23                     INX     H         ;ADR OF NNNLSB OF PAT
011E 1A                     LDAX    D         ;LOAD NNNLSB OF SUM
011F 8E                     ADC     M         ;ADD TO PAT
0120 27                     DAA               ;BCD ADJUST
0121 12                     STAX    D         ;STORE NEW NNLSB OF SUM
                    ;****   MSB ADDITION
0122 13                     INX     D         ;ADR OF SUM BYTE
0123 23                     INX     H         ;ADR OF PAT BYTE
0124 1A                     LDAX    D         ;LOAD SUM BYTE
0125 8E                     ADC     M         ;ADD TO PAT
0126 27                     DAA               ;BCD ADJUST
0127 12                     STAX    D         ;NEW MSB OF SUM
                    ;
                    ;****   KILL TIME FOR DIFFERENCE IN AD SUB ROUTINES
                    ;
0128 3E06                   MVI     A,6
012A 3D                     DCR     A
012B F22A01                 JP      $-1
012E C37601                 JMP     STEPS
```

```
                    ;
                    ;****  BCD SUBTRACTION ROUTINE
                    ;
                    DSUB:
0131 E67F                  ANI     07FH      ;GET RID OF SIGN BIT FROM PAT

0133 4F                    MOV     C,A       ;SAVE MSB OF PAT
0134 2B                    DCX     H         ;GET ADR OF LSB OF PAT
0135 2B                    DCX     H
0136 2B                    DCX     H
0137 2B                    DCX     H         ;LSB ADR IN H,L
                    ;****  LSB SUBTRACTION
0138 0699                  MVI     B,99H     ;USED IN GETTING 99 OR 100 COMPL
013A 37                    STC               ;INDICATE NO BORROW
013B 78                    MOV     A,B
013C CE00                  ACI     0         ;SETUP FOR 99 OR 100 COMP
013E 96                    SUB     M         ;99 OR 100 COMP OF PAT
013F EB                    XCHG              ;PAT ADR TO D,E-SUM ADR TO H,L
0140 86                    ADD     M         ;ADD SUM TO RESULT
0141 27                    DAA               ;BCD ADJUST RESULT
0142 EB                    XCHG              ;SUM ADR TO D,E-PAT ADR TO H,L
0143 12                    STAX    D         ;NEW LSB SUM BYTE
                    ;****  NLSB SUBTRACTION
0144 13                    INX     D         ;ADR NLSB OF PAT
0145 23                    INX     H         ;ADR OF NLSB OF PAT
0146 78                    MOV     A,B       ;99H TO A
0147 CE00                  ACI     0         ;SETUP FOR 99 OR 100 COMPL
0149 96                    SUB     M         ;99 OR 100 COMP OF PAT
014A EB                    XCHG              ;PAT ADR TO D,E-SUM ADR TO H,L
014B 86                    ADD     M         ;ADD SUM TO RESULT
014C 27                    DAA               ;BCD ADJUST
014D EB                    XCHG              ;SUM ADR TO D,E-PAT TO H,L
014E 12                    STAX    D         ;NEW NLSB OF SUM
                    ;****  NNLSB SUBTRACTION
014F 13                    INX     D         ;ADR OF NNLSB OF SUM
0150 23                    INX     H         ;ADR OF NNLSB OF PAT
0151 78                    MOV     A,B       ;LOAD A WITH 99H
0152 CE00                  ACI     0
0154 96                    SUB     M         ;99 OR 100 COMP OF PAT
0155 EB                    XCHG              ;PAT ADR TO D,E-SUM ADR TO H,L
0156 86                    ADD     M         ;ADD SUM BYTE
0157 27                    DAA               ;BCD ADJUST
0158 EB                    XCHG              ;SUM ADR TO D,E-PAT ADR TO H,L
0159 12                    STAX    D         ;NEW NNLSB OF SUM
                    ;****  NNSB SUBTRACTION
015A 13                    INX     D         ;ADR OF SUM NNSB
015B 23                    INX     H         ;ADR OF PAT NNSB
015C 78                    MOV     A,B       ;LOAD A WITH 99H
015D CE00                  ACI     0
015F 96                    SUB     M         ;99 OR 100 COMPL OF PAT
0160 EB                    XCHG              ;PAT ADR TO D,E-SUM ADR TO H,L
0161 86                    ADD     M         ;ADD SUM NNSB TO RESULT
0162 27                    DAA               ;BCD ADJUST
0163 EB                    XCHG              ;SUM ADR TO D,E-PAT ADR TO H,L
0164 12                    STAX    D         ;NEW NNSB OF SUM
```

```
                ;****   MSB SUBTRACTION
0165 13                 INX     D         ;ADR MSB OF SUM
0166 23                 INX     H         ;ADR MSB OF PAT
0167 78                 MOV     A,B       ;LOAD A WITH 99H
0168 CE00               ACI     0
016A 91                 SUB     C         ;PAT MSB WITHOUT SIGN SAVED IN C
016B EB                 XCHG              ;PAT ADR TO D,E-SUM ADR TO H,L
016C 86                 ADD     M         ;ADD IN SUM BYTE
016D 27                 DAA               ;BCD ADJUST
016E EB                 XCHG              ;SUM ADR TO D,E-PAT ADR TO H,L
016F 12                 STAX    D         ;NEW MSB OF SUM
                ;
                ;****   UPDATE CURRENT EVENT CLOCK WITH CYCLE VALUE
                ;
0170 2A5E00     STEP5:  LHLD    SAV       ;PUT ADR OF START THIS EFFECT IN
0173 55                 MOV     D,L       ;SAVE LOW ADR IN D
0174 7D                 MOV     A,L       ;
0175 C60A               ADI     10        ;ADDING 8 GETS ADR OF CYCLE RATE
0177 6F                 MOV     L,A       ;POINT H,L AT CYCLE RATE
0178 4E                 MOV     C,M       ;LOAD LSB OF CYCLE RATE
0179 2C                 INR     L         ;ADR OF MSB OF CYCLE RATE
017A 46                 MOV     B,M       ;LOAD B WITH MSB OF CYCLE RATE
017B 6A                 MOV     L,D       ;POINT H,L AT CURRENT EVENT CLOCK
017C 5E                 MOV     E,M       ;LOAD LSB OF CUR EVENT
017D 2C                 INR     L         ;ADR OF NMSB OF CUR CLOCK
017E 56                 MOV     D,M       ;LOAD NMSB OF CUR EVENT CLOCK
017F 2C                 INR     L         ;ADR OF MSB OF CUR EVENT
0180 AF                 XRA     A         ;ZERO A
0181 EB                 XCHG              ;ADR TO D,E-LSB,NMSB OF CUR EV TO
0182 09                 DAD     B         ;UPDATE LSB AND NMSB OF CUR EVENT
0183 EB                 XCHG              ;ADR MSB TO H,L-VAL TO D,E
0184 8E                 ADC     M         ;ADD 0+MSB CUR EVENT+CARRY
0185 77                 MOV     M,A       ;UPDATE MSB OF CUR EVENT CLOCK
0186 2D                 DCR     L         ;ADR NMSB OF CUR EVENT
0187 72                 MOV     M,D       ;UPDATE NMSB CUR EVENT
0188 2D                 DCR     L         ;ADR OF LSB
0189 73                 MOV     M,E       ;UPDATE LSB CUR EVENT CLOCK
018A 47                 MOV     B,A       ;SAVE MSB CUR EVENT
018B 4D                 MOV     C,L       ;SAVE ADR LSB CUR EVENT
                ;
                ;****   COMPARE STOP CLOCK WITH CURRENT EVENT CLOCK-IF TI
                ;****   TO END THIS EFFECT SET CURRENT EVENT CLOCK TO MAX
                ;
018C 3E0E               MVI     A,14      ;OFFSET TO EFFECT STOP
018E 85                 ADD     L         ;L HAS START THIS EFFECT
018F 6F                 MOV     L,A       ;POINT H,L AT EFFECT STOPMSB
0190 78                 MOV     A,B       ;MSB CUR CLOCK SAVED IN B
0191 BE                 CMP     M         ;COMPARE MSB CUR TO STOP
0192 C29E01             JNZ     CLC
0195 7B                 MOV     A,E       ;E HAS NMSB OF CUR EVENT
0196 2D                 DCR     L         ;ADR NMSB OF STOP CLOCK
0197 BE                 CMP     M         ;COMPARE NMSB CUR TO STOP
0198 C29E01             JNZ     CLC
019B 7B                 MOV     A,E       ;LOAD LSB CUR EVENT CLOCK
019C 2D                 DCR     L         ;ADR LSB OF STOP CLOCK
019D BE                 CMP     M         ;COMPARE LSB OF CUR TO STOP
[illegible]     CLC:    JC      [illegible] ;[illegible] CUR LT STOP
```

```
                ;
                ;****  SET CUR EVENT CLOCK FOR THIS EFFECT TO MAX
01A1 69                MOV     L,C      ;ADR LSB CUR EVENT CLOCK TO L
01A2 3EFF              MVI     A,0FFH
01A4 77                MOV     M,A      ;MAX LSB BYTE
01A5 2C                INR     L        ;ADR NMSB CUR EVENT
01A6 77                MOV     M,A      ;MAX NMSB
01A7 2C                INR     L        ;ADR MSB OF CUR EVENT
01A8 77                MOV     M,A      ;MAX MSB
                ;
                ;****  CHECK FOR MORE EFFECTS
                ;
01A9 3A5E00     OK:    LDA     5EH      ;LOAD LOW BYTE OF START ADR THIS
01AC C60F              ADI     15       ;GET ADR OF START NEXT EFFECT
01AE FE5D              CPI     5DH      ;5DH IS OUT OF EFFECT TABLE
01B0 6F                MOV     L,A
01B1 C27300            JNZ     STEP1    ;NOT EQUAL TO 5DH THAN HAVE ANOTH
                ;
                ;****  HAVE CONSIDERED ALL THREE EFFECTS-CONVERT VALUE I
                ;****  TO BINARY TWO BYTE FORM OF INTELL
                ;
01B4 112800            LXI     D,SFREQ  ;ADR OF LSB OF SUM
01B7 1A                LDAX    D        ;LOAD LSB OF SUM
01B8 2603              MVI     H,TEN    ;ADR OF TEN/UNIT CONVERSION TAB
01BA 6F                MOV     L,A      ;VAL OF MILL 10/UNIT VAL IS INDEX
01BB 4E                MOV     C,M      ;SAVE BIN EQUIV OF MILL 10/UNITS
01BC 1C                INR     E        ;ADR OF NLSB OF SUM
01BD 1A                LDAX    D        ;LOAD HERTZ UNIT/MILLI 100 BYTE
01BE 47                MOV     B,A      ;SAVE NLSB
01BF E60F              ANI     0FH      ;MASK OFF HERTZ UNITS VAL
01C1 07                RLC              ;DOUBLE 100 VAL TO GET TAB OFFSET
01C2 C69A              ADI     154      ;ADD IN BASE OF 100 BIN TABLE
01C4 6F                MOV     L,A      ;POINT H,L AT 100 BIN EQUIV
01C5 5E                MOV     E,M      ;LOAD LOW BYTE OF MILI 100 BIN EQ
01C6 23                INX     H
01C7 56                MOV     D,M      ;HIGH VAL OF MILLI 100 BIN EQUIV
01C8 69                MOV     L,C      ;LOAD MILLI 10/UNITS VAL
01C9 2600              MVI     H,0
01CB 19                DAD     D        ;MILLI BIN EQUIV IN H,L
                ;****  MILLIHERTZ OUTPUT
                ;
01CC 7C                MOV     A,H      ;HIGH BYTE VAL
01CD 2F                CMA
01CE D309              OUT     9        ;OUTPUT HIGH BYTE VAL
01D0 7D                MOV     A,L      ;LOAD LOW BYTE VAL
01D1 2F                CMA
01D2 D308              OUT     8        ;OUTPUT LOW BYTE VAL
```

```
                    ;****   HERTZ CONVERSION
                    ;
01D4 78                     MOV     A,B         ;NLSB OF SUM SAVED IN B
01D5 1F                     RAR                 ;SHIFT HIGH 4 BITS TO LOW 4 BITS
01D6 1F                     RAR
01D7 1F                     RAR
01D8 1F                     RAR
01D9 E60F                   ANI     0FH         ;MASK OUT HIGH 4 BITS
01DB 4F                     MOV     C,A         ;SAVE HERTZ UNITS VAL
01DC 112A00                 LXI     D,SFREQ+2 ;ADR OF HERTZ 100/10 BYTE
01DF 1A                     LDAX    D           ;LOAD HERTZ BCD 100/10 BYTE
01E0 6F                     MOV     L,A
01E1 2600                   MVI     H,0
01E3 29                     DAD     H           ;DOUBLE 100/10 VAL TO GET BIN TAB
01E4 110004                 LXI     D,0400H      ;ADR OF 100/10 BIN EQUIV TABLE
01E7 19                     DAD     D           ;ACTUAL ADR OF HERTZ 100/10 BIN V
01E8 5E                     MOV     E,M         ;LOAD LOW BIN EQUIV
01E9 23                     INX     H           ;ADR OF HIGH BIN EQUIV
01EA 56                     MOV     D,M         ;LOAD HIGH BINARY EQUIV
01EB 69                     MOV     L,C         ;LOAD HERTZ UNITS
01EC 2600                   MVI     H,0
01EE 19                     DAD     D           ;HERTZ BIN EQUIV IN H,L
                    ;****   HERTZ OUTPUT
                    ;
01EF 7C                     MOV     A,H         ;HIGH BYTE VAL
01F0 2F                     CMA
01F1 D30B                   OUT     11          ;OUTPUT HIGH BYTE VAL
01F3 7D                     MOV     A,L         ;LOW BYTE VAL
01F4 2F                     CMA
01F5 D30A                   OUT     10          ;OUTPUT LOW BYTE VAL
                    ;****   KILLIHERTZ CONVERSION
                    ;
01F7 112B00                 LXI     D,SFREQ+3 ;ADR OF NMSB OF SUM
01FA 1A                     LDAX    D           ;LOAD KHERTZ 10/UNIT BCD BYTE
01FB 2603                   MVI     H,TEN       ;ADR TEN/UNIT BIN EQUIV TABLE
01FD 6F                     MOV     L,A         ;BCD 10/UNIT VAL IS TABLE INDEX
01FE 4E                     MOV     C,M         ;SAVE BIN EQUIV IN C
01FF 1C                     INR     E           ;ADR OF MSB OF SUM
0200 1A                     LDAX    D           ;LOAD KHERTZ 100,MEGA BYTE
0201 47                     MOV     B,A         ;SAVE FOR MEGA CONVERSION
0202 E60F                   ANI     0FH         ;MASK OFF MEGA BITS
0204 07                     RLC                 ;DOUBLE TO GET 100 TAB INDEX
0205 C69A                   ADI     154         ;ADD TABLE BASE
0207 6F                     MOV     L,A
0208 5E                     MOV     E,M         ;LOW BIN EQUIV OF KHERTZ 100
0209 23                     INX     H
020A 56                     MOV     D,M         ;HIGH BIN EQUIV
020B 69                     MOV     L,C         ;LOAD 10/UNITS BIN EQUIV
020C 2600                   MVI     H,0
020E 19                     DAD     D           ;KHERTZ BIN EQUIV
```

```
                ;****   KHERTZ OUTPUT
                ;
020F 7C                 MOV     A,H
0210 2F                 CMA
0211 D30D               OUT     13      ;OUTPUT HIGH BYTE VAL
0213 7D                 MOV     A,L     ;LOAD LOW BYTE VAL
0214 2F                 CMA
0215 D30C               OUT     12      ;OUTPUT LOW BYTE VAL
                ;****   MEGA HERTZ CONVERSION AND OUTPUT
                ;
0217 78                 MOV     A,B     ;MEGA BYTE SAVED IN B
0218 1F                 RAR             ;SHIFT TO LOW 4 BITS
0219 1F                 RAR
021A 1F                 RAR
021B 1F                 RAR
021C E60F               ANI     0FH     ;MASK OFF HIGH 4 BITS
021E 2F                 CMA
021F D30E               OUT     14      ;OUTPUT MEGA HERTZ
                ;****   SEND LOAD PULSE
                ;
0221 3EFF               MVI     A,0FFH  ;LOW PULSE VAL
0223 D30F               OUT     15
0225 3EFE               MVI     A,0FEH  ;HIGH PULSE VAL=BIN 1
0227 D30F               OUT     15
                ;
                ;****   UPDATE SIMULATION CLOCK
                ;
0229 212000             LXI     H,SIM   ;ADR OF SIMULATION CLOCK
022C 5E                 MOV     E,M     ;LSB OF SIM CLOCK
022D 2C                 INR     L       ;ADR OF NMSB
022E 56                 MOV     D,M     ;LOAD NMSB
022F 2C                 INR     L       ;ADR OF MSB
0230 AF                 XRA     A       ;ZERO A
0231 EB                 XCHG            ;ADR TO D,E-VAL TO H,L
0232 010100             LXI     B,1     ;PUT A 1 IN B,C PAIR
0235 09                 DAD     B       ;BUMP SIM LSB AND NMSB
0236 EB                 XCHG            ;ADR TO H,L-VAL TO D,E
0237 8E                 ADC     M       ;SIM MSB+CARRY+ZERO
0238 77                 MOV     M,A     ;UPDATE SIM MSB
0239 2D                 DCR     L       ;ADR OF NMSB
023A 72                 MOV     M,D     ;UPDATE NMSB OF SIM CLOCK
023B 2D                 DCR     L       ;ADR OF LSB
023C 73                 MOV     M,E     ;UPDATE SIM LSB
```

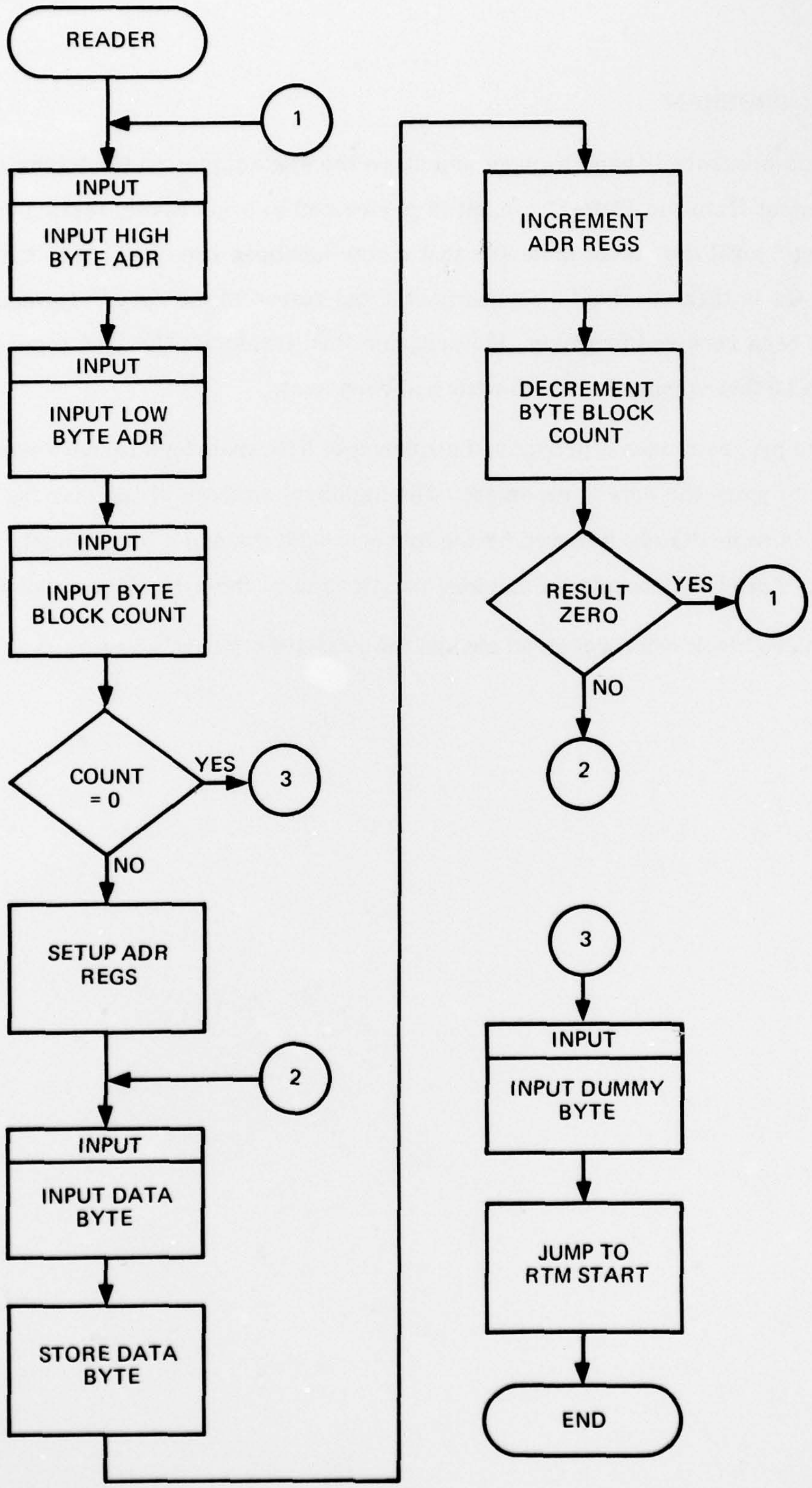
```
                   ;
                   ;****  COMPARE SIM CLOCK WITH SIM STOP FOR END SIMULATIO
                   ;
023D 2E2F                MVI     L,STOP+2   ;LOAD ADR OF STOP CLOCK
023F BE                  CMP     M          ;COMPARE MSB OF SIM CLOCK TO STOP
0240 C24C02              JNZ     CHSTP
0243 2D                  DCR     L          ;ADR OF NMSB OF STOP CLOCK
0244 7A                  MOV     A,D        ;LOAD NMSB OF SIM CLOCK
0245 BE                  CMP     M          ;COMPARE WITH NMSB OF STOP CLOCK
0246 C24C02              JNZ     CHSTP
0249 2D                  DCR     L          ;ADR LSB OF STOP CLOCK
024A 7B                  MOV     A,E        ;LOAD LSB OF SIM CLOCK
024B BE                  CMP     M          ;COMPARE WITH LSB STOP CLOCK
024C DA6000        CHSTP: JC     STEP0      ;DO ANOTHER CYCLE
024F C30000              JMP     00H        ;RETURN TO THE MONITOR
```

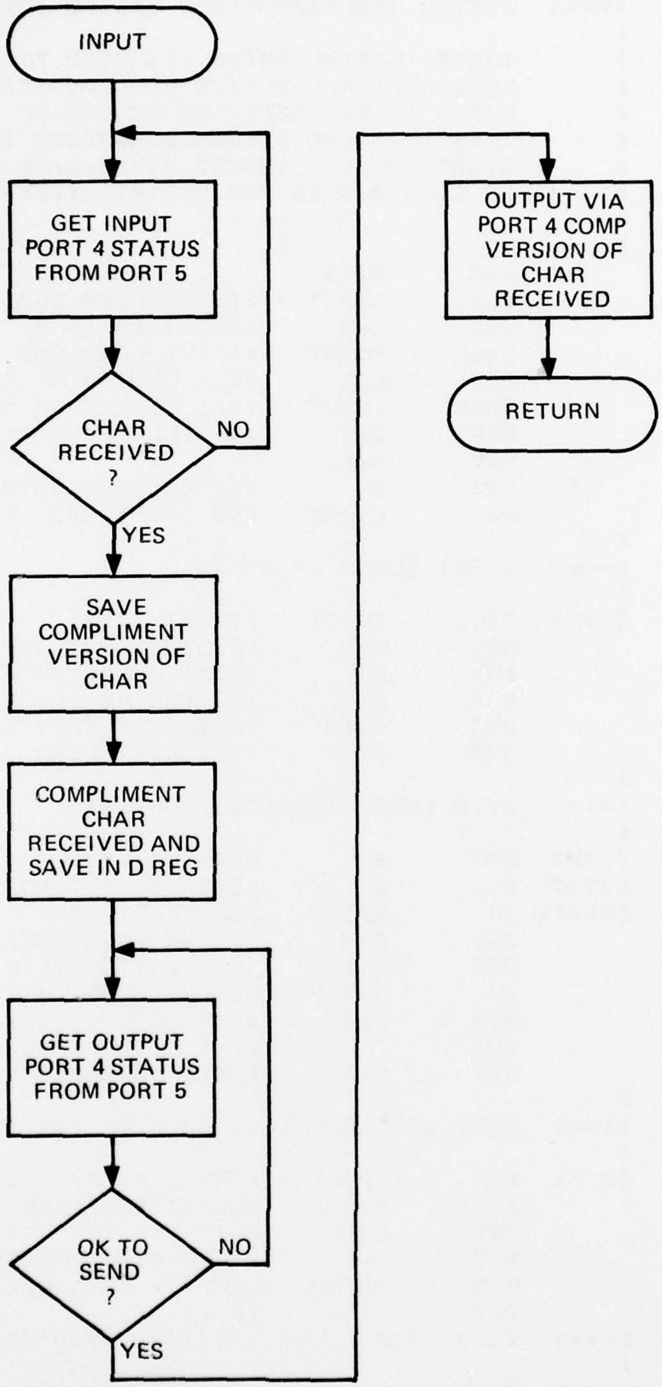
READER PROGRAM

This program is used to input and store the system parameter tables and pattern tables output from the PDP-11. Input is performed by repetitively testing the status of input port 5 until the status indicates that a byte has been received. The byte sent from the PDP-11 is then received via input port 4 and stored in memory. To insure that the byte has been received properly, the program then transmits the byte received back to the PDP-11 that compares it with what had been sent.

The program uses a predefined sequence of byte transfers to determine where in memory to store the data it receives. The high byte address of the starting location is sent and is immediately followed by the low byte address and a block count. Data is then input and stored at consecutive memory locations until the byte block count is satisfied.

A zero block count received causes the real-time mode to begin.


READER
1
INPUT
INPUT HIGH BYTE ADR
INPUT
INPUT LOW BYTE ADR
INPUT
INPUT BYTE BLOCK COUNT
COUNT = 0
YES
3
NO
SETUP ADR REGS
2
INPUT
INPUT DATA BYTE
STORE DATA BYTE
INCREMENT ADR REGS
DECREMENT BYTE BLOCK COUNT
RESULT ZERO
YES
1
NO
2
3
INPUT
INPUT DUMMY BYTE
JUMP TO RTM START
END


INPUT
GET INPUT PORT 4 STATUS FROM PORT 5
CHAR RECEIVED ?
NO
YES
SAVE COMPLIMENT VERSION OF CHAR
COMPLIMENT CHAR RECEIVED AND SAVE IN D REG
GET OUTPUT PORT 4 STATUS FROM PORT 5
OK TO SEND ?
NO
YES
OUTPUT VIA PORT 4 COMP VERSION OF CHAR RECEIVED
RETURN

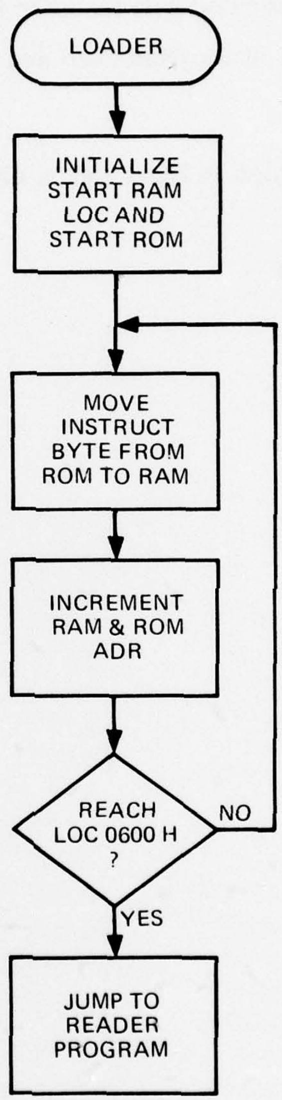
```
                ;
                ;
                ;****   SYSTEM TABLES/PATTERN TABLES INPUT ROUTINE
                ;
                ;       THREE INITIAL READS ARE USED TO IDENTIFY HIGH BYT
                ;       LOAD ADDRESS,LOW BYTE LOAD ADDRESS,AND NUMBER OF
                ;       BYTES IN TRANSFER.THE NUMBER OF BYTES SPECIFIED A
                ;       THEN READ AND STORED ACCORDING TO BYTE COUNT AND
                ;       START MEMORY ADDRESS GIVEN.EACH BYTE READ FROM TH
                ;       IS ECHOED BACK FOR VERIFICATION PURPOSES.
                ;
                ;
0252                    ORG     270H
0270 CD8E02     ST:     CALL    INPUT   ;INPUT HIGH LOAD ADR
0273 62                 MOV     H,D     ;INPUT IS IN D
0274 CD8E02             CALL    INPUT   ;INPUT LOW LOAD ADR
0277 6A                 MOV     L,D     ;H,L NOW POINT TO START ADR FOR B
0278 CD8E02             CALL    INPUT   ;GET NUMBER OF BYTES IN BLOCK
027B 42                 MOV     B,D     ;B WILL BE LOOP REG
027C 78                 MOV     A,B
027D FE00               CPI     0       ;CHECK FOR END(0 BLOCK COUNT)
027F CAA502             JZ      LOADX   ;GO START REAL TIME PROGRAM
                ;
                ;****   INPUT BLOCK OF BYTES
                ;
0282 CD8E02     CHIN:   CALL    INPUT   ;INPUT BYTE
0285 72                 MOV     M,D     ;STORE IN MEMORY
0286 23                 INX     H       ;NEXT MEMORY ADR TO LOAD
0287 05                 DCR     B       ;DECREMENT BLOCK COUNT
0288 C28202             JNZ     CHIN    ;NOT ZERO THEN HAVE ANOTHER CHAR
028B C37002             JMP     ST      ;GET NEXT LOAD ADR,ETC
                ;
                ;****   BYTE INPUT AND ECHO ROUTINE
                ;
0005            SPORT   EQU     5       ;STATUS PORT
0004            DATAP   EQU     4       ;DATA IN/OUT PORT
028E DB05       INPUT:  IN      SPORT   ;INPUT STATUS
0290 E601               ANI     01H     ;CHECK INPUT READY
0292 C28E02             JNZ     INPUT   ;BIT 0=1 THEN INPUT NOT THERE
0295 DB04               IN      DATAP   ;INPUT BYTE FROM THE 11
0297 4F                 MOV     C,A     ;SAVE COMPLIMENTED VERSION
0298 2F                 CMA             ;GET REAL VERSION OF INPUT BYTE
0299 57                 MOV     D,A     ;BYTE IS RETURNED IN D
                ;
                ;****   ECHO BYTE RECEIVED BACK TO THE 11
                ;
029A DB05       ECHO:   IN      SPORT   ;GET XMIT STATUS
029C E604               ANI     04H     ;CHECK IF READY
029E C29A02             JNZ     ECHO    ;BIT 2=1 THEN NOT READY TO SEND
02A1 79                 MOV     A,C     ;MOVE COMPLIMENTED VERSION
02A2 D304               OUT     DATAP   ;OUTPUT BYTE BACK TO 11
02A4 C9                 RET             ;RETURN
                ;****   ZERO BYTE COUNT RECEIVED-LOADING DONE-START RTM
                ;
02A5 CD8E02     LOADX:  CALL    INPUT   ;INPUT AND ECHO DUMMY BYTE
02A8 C36000             JMP     STEP0   ;START REAL TIME MODE
0000                    END
```


Fm

LOADER PROGRAM

This program is used to load the real-time program into RAM from its location in ROM. The program when executed will transfer the contents of locations 2C200 through 31FF to RAM beginning at location zero and cause program execution of the system tables, input routine.

This program is also located in ROM and is intended to act as a bootstrap loader of the real-time run.


LOADER
INITIALIZE START RAM LOC AND START ROM
MOVE INSTRUCT BYTE FROM ROM TO RAM
INCREMENT RAM & ROM ADR
REACH LOC 0600 H ?
NO
YES
JUMP TO READER PROGRAM

```
                ;       KBAND REAL TIME PROGRAM LOADER
                ;
                ;       THE PURPOSE OF THIS PROGRAM IS TO MOVE THE KBAND
                ;       FROM ROM TO RAM AND INITIATE THE TABLE READ PART
                ;       THE PROGRAM.
                ;
                ;
                ;
0000                    ORG     2B00H   ;ROM ADR FOR THIS PROGRAM
0270            RDR     EQU     270H    ;ADR OF TABLE INPUT ROUTINE
0000            RAM     EQU     0       ;START ADR IN RAM TO LOAD
2C00            ROM     EQU     2C00H   ;START ADR OF RTP IN ROM
                ;
2B00 110000             LXI     D,RAM   ;BEGIN ADR TO LOAD
2B03 21002C             LXI     H,ROM   ;ADR IN ROM TO BEGIN LOAD FROM
2B06 7E         INST:   MOV     A,M     ;LOAD A WITH INSTRUCT BYTE
2B07 12                 STAX    D       ;MOVE INTO RAM
2B08 23                 INX     H       ;NEXT ROM ADR
2B09 13                 INX     D       ;NEXT RAM ADR
2B0A 7A                 MOV     A,D     ;LOAD RAM HIGH ADR BYTE
2B0B FE06               CPI     06H     ;FINISHED LOAD AT 0600H
2B0D C2062B             JNZ     INST
2B10 C37002             JMP     RDR     ;JUMP TO TABLE LOAD ROUTINE
0000                    END
```


☆ U. S. GOVERNMENT PRINTING OFFICE: 1976 — 757-001/154