PLEASE

TABLE OF CONTENTS

Second Edition

LISTINGS

Introduction to PLEASE

I have been working with minicomputers and microcomputers for nearly eight years. During this time I have seen some pretty horrible examples of how NOT to design small computer software packages. Often, concepts and techniques which might be appropriate to a large computer system are thoughtlessly applied to a small system, with predictably bad results. I have developed a number of programming philosophies and techniques which are designed specifically for the types of small machines that are becoming available to the computer hobbyist and small businessman.

PLEASE is a very small package which embodies several of the more important concepts. It has a tiny monitor which takes care of the basic I/O functions, timing functions, and scheduling. It has a set of functions to perform the major tasks. These functions are combined through the use of a simple interpreter which calls up the functions and passes the necessary parameters to them. Programs are written as a series of functions. Since each function does a major piece of work, a program may consist of only a few functions and can be readily "assembled" by hand. For example, the PLEASE Command Decoder which permits the user to select which game or demo to run from the keyboard is four functions, encoded in a total of 16 bytes.

I am very impressed with the capabilities of the KIM-1 Microcomputer System. Aside from the excruciatingly slow cassette dump/load, it is remarkably well thought out. However, I must admit that after getting my own KIM-1 and doing the single example in the manual I was more than a bit frustrated that there wasn't anything else to do without a great deal of effort on my part. I could only impress my wife and friends slightly by adding 2 + 3 and getting 5! I decided, therefore, to create a small games and demos package that could run on the basic KIM-1, without any additional peripherals such as teletype or special display devices. The result is PLEASE.

PLEASE has been developed entirely on a basic KIM-1. All of the assembly has been done by hand. All of the object code has been entered through the keyboard. Debugging has been via the KIM-1 Monitor. The resulting package demonstrates what can be done on a basic system. It also does suffer in some respects though. Certain changes I would have liked to make became just too much effort via hand assembly.

I welcome any suggestions, comments, or whatever. I have had fun developing PLEASE and hope you will enjoy playing with it.

Robert M. Tripp

Robert M. Tripp

Contents of the PLEASE Instructions and PLEASE Object Cassette Tape are Copyright 1976 by The COMPUTERIST, P.O. Box 3, S Chelmsford MA 01824.

Prices for the PLEASE Package are:

OBJECT CASSETTE TAPE and Operating Instructions:	$6.00
SOURCE LISTINGS and Instructions for Writing Code:	$6.00
Complete PLEASE Package Including Everything:	$10.00

These may be ordered from the above address. For information: 617/256-3649-

Overview and Fundamental Concepts

The approach that PLEASE uses to create a program is probably quite different from the traditional approaches you have encountered before. Its keywords are simplicity and modularity. Many approaches talk about simplicity and modularity, but by the time you get to really use them they require macro assemblers, linking loaders, compilers, and many other "aids" that often lead to very inefficient code that is very difficult to understand and/or debug. The basic element of PLEASE is the Function. A complex set of tasks can be performed by a fairly limited set of simple tasks, if the correct set of simple functions is selected and if there is a simple way to combine the independent functions into a useful whole. In PLEASE a set of fourteen (14) basic functions has been defined and implemented, and a routine called the Interpreter is used to combine these functions into useful entities called Programs. Certain programs have requirements that are quite unique, and PLEASE provides for special Functions to be written for these. A Program is written in PLEASE as a series of Steps. Each Step consists of the name of a Function and its associated parameters. In a sense, PLEASE programs are written in a form of microcode. Each Step is 32 bits of information: 8 bits of Function name followed by 3 8-bit units of parameter information. A PLEASE Step to get decimal input from the keypad, store the results in consecutive locations starting with a location named Buffer, and display the digits as they are input on the KIM-1 LED's from the leftmost position (Ø) to the rightmost position (5) is:

```
DECIN     BUFFER    Ø         5
```

To have the digits appear in position 5 and then be shifted to the left as in a calculator, the Step would be:

```
DECIN     BUFFER    5         Ø
```

This "microcoding" permits each PLEASE Function to perform in a number of different ways. The richness of the Functions will become apparent in the sections which describe the individual functions.

Before examining the functions in detail, there are a couple of other parts of PLEASE that should be understood. These are the Monitor, the Timer, the Interpreter, and the various memory locations which have been allocated for particular uses.

Monitor. The Monitor performs a number of basic services. It is responsible for maintaining a one millisecond main loop, for accepting in conjunction with the ROM characters from the keypad, for refreshing the LED display, and for branching off to the system Timer and application code. The Monitor is general purpose and can be used entirely independently of the remainder of PLEASE.

Timer. The Timer is based on the one millisecond loop of the Monitor. During each pass through the Monitor, a call is made to the Timer. The Timer updates an eight character timer which can count from 0.001 to 99,999.999 seconds. Every thousand times through the loop, which equals one second, the timer updates a six character clock which counts from

00:00:00 to 23:59:59 as a 24 hour clock. The maintenance of the timer and clock is independent of the remainder of PLEASE, although there are several PLEASE programs which set or display the results of the timer and clock. The contents of the Timer are used by a number of programs to provide pseudo random numbers.

Interpreter. The Interpreter is the heart of the PLEASE technique, but is itself very simple. It has three basic purposes. First, it converts the Step Number to be executed into an absolute memory address and then bumps the Step Number to the next Step, just as a Program Counter does. Second, it moves the four bytes which consist of the Function Code and the three Parameter bytes from their original location to a standard location in the Page Zero making them easily accessible to the Functions. Finally, it looks up the address of the Function via the Function Code and the Function Table, and then transfers control to the Function. In this way the independent PLEASE Functions are tied together into a meaningful whole.

Memory Locations. PLEASE has very carefully partitioned the limited KIM-1 memory. The Monitor, which never changes, is always resident, and which may be used without the rest of PLEASE, is placed "out-of-the-way" in the extra RAM locations of the 6530 chips. These locations start at 1780 and go to 17E6. The remaining RAM locations 17E7 through 17FF are used by the KIM-1 Monitor. The Timer starts at location 200, the Interpreter at 300. The remainder of the locations in 200 through 3FF are used for the fourteen basic PLEASE Functions and some subroutines which are used by the Functions. This section of memory is also never modified by the basic PLEASE programs. Page One contains the Function Table in 100 through 11F, Special Function Table in 120 through 12F, Special Functions in 130 through 1EF, and the Stack in 1F0 through 1FF. The PLEASE Programs start at location 00. Page Zero is used for many purposes as described later. The memory ordering has been made to be least volatile in high memory, most volatile in low memory. Overlays always start at location 00.

Functions. The fourteen basic PLEASE Functions may be broken down into a few categories:

- Input. The only PLEASE Input device is the KIM-1 Keypad. There are three input modes: Alphabetic, Hexidecimal, and Decimal.
- Output. The only PLEASE Output device is the KIM-1 Display. There are three output modes: Alphabetic, Hexidecimal, and Decimal.
- Time. A timer routine provides for delays of up to 25 seconds in one-tenth of a second increments.
- Pack. A pair of functions provide for data to be packed as two characters per byte or to be unpacked into separate bytes.
- Branch. There are three basic forms of branch: unconditional, and conditional on an input character or conditional on a table lookup value.
- Fill. There is a function to fill/clear memory locations.

Match. There are two functions which permit comparing strings of data and branching on the result.

Special Functions. The basic fourteen PLEASE Functions can not, of course, do everything. Special Functions may be added whenever the basic functions either could not do the job, or would be too inefficient. Examples of Special Functions include: MASTER - the evaluator for the Daffy (Mastermind) Game; MESAGE - the heart of the Notice and Billboard Programs; and, START and SHOT - initializer and evaluator for Shooting Stars.

Programs. The current PLEASE package consists of about fifteen programs. These vary in complexity and size. SCLOCK (set clock) is only two basic PLEASE Functions. HiLo (Number Guessing Game) is nineteen basic PLEASE Functions. Daffy (Mastermind Game) is thirteen basic Functions and one special Function. And so forth.

Subroutines. There are about six subroutines in PLEASE. These are an odd collection of chunks of code that were useful to have around and were shared by one or more independent functions, but which were not powerful enough to warrant being treated as Functions.

The only way to understand PLEASE is to use it. Write your own programs entirely in PLEASE Functions. When a special need arises, write your own Special Function. Have fun. Enjoy.

NOTE: A word of Caution. Since this was all done by hand, it is quite possible that there may be one or more errors in the typing of the Programs and/or Source Listings. All code on the cassette was entered from the listings, so that the Object of the listings is probably okay. If you see a discrepancy between the Object and the Operation, the Object is probably right and the Operation wrong.

* *

The KIM-1 is made by MOS TECHNOLOGY, INC. Its list price as of December 1976 is $245.00 + 4.50 postage and handling. This is a fully assembled and tested unit and includes three manuals: KIM-1 User Manual, Programming Manual and Hardware Manual. You must provide your own power supply which has +5vdc @ 1.2 amps and +12vdc @ .2 amps. Other modules available from the same source are KIM-2 a 4K RAM, KIM-3 an 8K RAM, a mother board, and several ROM programs including a KIMath math package and an Editor/Assembler. Address is: 950 Rittenhouse Road, Norristown PA 19401
Phone: 215/666-7950

* *

IMPORTANT NOTICE: If you did not purchase your copy of PLEASE directly from The COMPUTERIST, then please send us your Name, Address, and Cassette Tape Number. This will enable us to keep you informed of any updates, new products, or corrections that occur in the future.
The COMPUTERIST, P.O. Box 3, S Chelmsford, MA 01824

EXINIT initializes the pointer to the first application subroutine and initializes the location CHAR to indicate no valid input character has been received.

```
1780  A9  00      EXINIT  LDA  #DCDLO      DCDLO = Application Low
1782  8D  CC  17          STA  EXAPLO      Address to Exec Transfer
1785  A9  03              LDA  #DCDHI      DCDHI = Application High
1787  8D  CD  17          STA  EXAPHI      Address to Exec Transfer
178A  A9  20              LDA  #20         20 is used to indicate
178C  85  DB              STA  CHAR        NO character input
```

EXLOOP controls a one millisecond loop which is required to permit the Display to be properly refreshed. KEYIN uses the KIM-1 ROM routines to input a character from the Keypad, and then performs "debouncing" of the Keypad. DSPLAY causes the next character in a six character Page Zero buffer to be displayed on the LED Display for one millisecond. Other routines are accessed via subroutine calls. TIMER updates a series of counters which provide decimal count from 1 millisecond up to 99,999.999 seconds. It also maintains a 24 hour clock, providing hours, minutes and seconds.

```
178E  A9  7C      EXLOOP  LDA  #7C         Set Millisecond Timer
1790  8D  45  17          STA  1745        (Actually 992 microseconds)

1793  20  88  1E  KEYIN   JSR  INITS       Setup for Keyboard Input
1796  20  6A  1F          JSR  GETKEY      Get Keyboard Input if any
1799  C9  14              CMP  #KEYMAX     Was there a Key Pressed ?
179B  30  04              BMI  KEYIN2      Yes, a Key was Pressed
179D  85  DE              STA  KEYTST      No, so reset debounce flag
179F  D0  0A              BNE  DSPLAY      Unconditional Branch
17A1  A6  DE      KEYIN2  LDX  KEYTST      Test debounce flag.
17A3  F0  06              BEQ  DSPLAY      Branch if not a new char.
17A5  85  DB              STA  CHAR        Save new character
17A7  A9  00              LDA  #00         Clear debounce flag
17A9  85  DE              STA  KEYTST

17AB  A4  D9      DSPLAY  LDY  DSPPOS      Current Display Pointer
17AD  98                  TYA              Calculate Select Line
17AE  0A                  ASL
17AF  69  08              ADC  #8
17B1  8D  42  17          STA  1742        Select Display Position
17B4  A9  7F              LDA  #7F         Select All Output Lines
17B6  8D  41  17          STA  1741
17B9  B1  CE              LDA  (DSPLO),Y   Get Display Character from
17BB  8D  40  17          STA  1740        Display Buffer and Output
17BE  88                  DEY              Decrement Position Pointer
17BF  10  02              BPL  DSPLY2      Okay if Positive
17C1  A0  05              LDY  #5          Else, Reset to Maximum
17C3  84  D9      DSPLY2  STY  DSPPOS      Save Pointer for Next Loop
```

Other routines are now invoked via subroutine jumps. TIMER is an optional routine. The next slot is available for any other routine that is to be included in every EXLOOP. Currently it is not being used and is therefore replaced by three NOPs. The actual application that is being run is accessed via the JSR APPL where the address of the application routine has been filled in at run time. It is initially set to address 0300 by the EXINIT code. It can be simply changed by the application by use of the EXSET routine explained below.

```
17C5  20  00  02          JSR  TIMER        Go to TIMER subroutine
17C8  EA  EA  EA         (JSR  ?????)       NOPs can be changed
17CB  20  00  03          JSR  APPL         This goes off to Application
```

EXWAIT waits for the interval timer to finish its 992 microsecond count. It then restarts the EXLOOP for the next millisecond loop. The testing and restart takes approximately 8 microseconds bringing the total time up to approximately 1000 microseconds or one millisecond.

```
17CE  AD  45  17  EXWAIT  LDA  1745         Pickup Current Interval
17D1  29  80              AND  #80          Timer Count and Test for
17D3  C9  80              CMP  #80          Downcount Complete
17D5  D0  F7              BNE  EXWAIT       Keep Waiting Until Done
17D7  F0  B5              BEQ  EXLOOP       When Done, Restart EXLOOP
```

EXSET is provided to make it easy for an application to change the location to be accessed on the next application call via the JSR APPL. The application code places a JSR EXSET immediately in front of the instruction which is to start the next application processing. EXSET picks up this address and places it into the JSR APPL instruction and then returns to EXWAIT by making a subroutine return RTS.

```
17D9  68          EXSET   PLA               Pull Address of Next
17DA  8D  CC  17          STA  EXAPLO       Application Instruction
17DD  EE  CC  17          INC  EXAPLO       and Save and Correct
17E0  68                  PLA               Pull Page Address of Next
17E1  8D  CD  17          STA  EXAPHI       Appl. Inst and Save it.
17E4  60                  RTS               Return to EXLOOP at EXWAIT
```

EXAMPLE: Within an application there is a need to wait for 10 milliseconds. After JSR EXSET pointers set to call HERE on next JSR APPL.

```
                          .
                          .
                          LDA  #10.         Set Counter = 10.
                          STA  COUNT
                  WAIT    JSR  EXSET        Return to EXLOOP with
                  HERE    DEC  COUNT        HERE as NEXT ADDRESS
                          BNE  WAIT         Keep Waiting Until BEQ
                          .
                          .
```

On each pass through the PLEASE Monitor EXLOOP, a one millisecond loop, a subroutine call is made to a TIMER subroutine. This routine provides two services. First, it updates an eight character millisecond based counter which has a range of 0.0 to 99,999.999 seconds and which is used by many programs to generate pseudo random numbers, is used by the reaction time program to calculate reaction time, and so forth. Second, it updates a six character 24 hour clock to provide time in hours, minutes and seconds. This subroutine is written to be relocatable anywhere in memory. For PLEASE, it is assembled at location 200. The eight character timer data is stored in four bytes starting at location CØ. The six character clock data is stored in three bytes starting at location C4. Location C7 is used as a counter.

```
200  F8       TIMER   SED           Calculations done in Decimal Mode
201  A2 03            LDX  #3       Setup Counter for Timer

203  18       TIMER1  CLC           Clear Carry Bit
204  B5 C0            LDA  THOUS,X  Add 1 to Current Timer Value
206  69 01            ADC  #1
208  95 C0            STA  THOUS,X
20A  90 2D            BCC  TDONE    All done if no Carry

20C  E0 03            CPX  #3       Test Carry from Millisecond
20E  D0 26            BNE  TIMER2   Skip Clock Update unless Millisecond
210  C6 C7            DEC  ONESEC   Decrement a Ten * 1/10 second counter
212  D0 22            BNE  TIMER2   Skip Clock Update unless 10 counts
```

If one second has expired since the last update of the Clock, then the Clock is updated. First the seconds byte is incremented and tested for its limit (60 seconds). If not equal to its limit a test is performed to test the hour byte limit. If the hour byte is not equal to its limit (24 hours), then the clock portion of the Timer subroutine is complete. If the seconds byte is equal to its limit, then it is reset to zero and the minutes byte is incremented. It is tested in exactly the same manner (actually the same code) as the seconds for its limit. After the seconds, minutes, and hours bytes have been updated, the Ten * 1/10 second counter is reset and the X index register is reset.

```
214  18       CLOCK   CLC
215  B5 C3            LDA  MILLI,X  Get Byte Using Index
217  69 01            ADC  #1       Increment Byte
219  C9 60            CMP  #60      Test Second or Minute Limit
21B  D0 07            BNE  HTEST    Not Limit.  Test Hour Limit
21D  A9 00            LDA  #0       Reset Second or Minute to Zero
21F  95 C3            STA  MILLI,X  Save Modified Byte
221  CA               DEX           Decrement Index
222  10 F0            BPL  CLOCK    Unconditional Branch
```

```
224  E0 01    HTEST   CPX  #1         Is this the Hour Byte?
226  D0 06            BNE  CDONE      If not, then Clock is Done
228  C9 24            CMP  #24        Test Hour Limit
22A  D0 02            BNE  CDONE      If not Limit, then Clock is Done
22C  A9 00            LDA  #0         If Limit, Reset to Zero

22E  95 C3    CDONE   STA  MILLI,X    Store Modified Value
230  A2 03            LDX  #3         Restore Index Value
232  A9 0A            LDA  #A         Reset Ten * 1/10 Counter
234  85 C7            STA  ONESEC

236  CA       TIMER2  DEX             Decrement Index
237  10 CA            BPL  TIMER1     If Index Positive, Continue Updates

239  D8       TDONE   CLD             All Done.  Clear Decimal Mode
23A  60              RTS             Subroutine Return
```

These updated Timer and Clock values are now ready to be used for the various purposes of PLEASE and can be displayed on the six character display.

* *

Alpha Display Table

Loc	Value	Key	Character
3F0	77	Ø	A
3F1	7C	1	b
3F2	58	2	c
3F3	5E	3	d
3F4	79	4	E
3F5	71	5	F
3F6	76	6	H
3F7	30	7	I
3F8	38	8	L
3F9	54	9	n
3FA	5C	A	o
3FB	73	B	P
3FC	50	C	r
3FD	6D	D	S
3FE	78	E	t
3FF	6E	F	Y

Hexidecimal Display Table

Loc	Value	Key	Character
1FE7	3F	Ø	Ø
1FE8	06	1	1
1FE9	5B	2	2
1FEA	4F	3	3
1FEB	66	4	4
1FEC	6D	5	5
1FED	7D	6	6
1FEE	07	7	7
1FEF	7F	8	8
1FF0	6F	9	9
1FF1	77	A	A
1FF2	7C	B	b
1FF3	39	C	C
1FF4	5E	D	d
1FF5	79	E	E
1FF6	71	F	F

Note: The Hexidecimal Display Table is located in the KIM-1 ROM. The table actually has bit 80 on in each byte, that is location 1FE7 actually contains a BF which is 3F + 80. Only those bits which have associated display segments are listed in this table.

Interpreter

A program written in PLEASE consists of a series of Functions and their associated Parameters. It is the responsibility of the Interpreter to: maintain a pointer to the next Step to be executed; convert the Step Number to the actual Step Location in memory; move the Function Code and the Parameter Values to standard Page Zero locations; lookup the actual memory address of the Function in the Function Table; and, transfer control to the selected Function. The Interpreter is the "glue" that holds the independent Functions together to form a Program.

A portion of the Monitor initialization sequence invoked when the system is started at location 1780 sets up a pointer in the Monitor so that the first call to the Application code comes to the Interpreter at location DECODE. This causes the Step Number STEPNO to be set to zero. This will cause the PLEASE Function at STEPNO Ø or Location ØØØØ to be the first Function executed. In all of the current PLEASE programs this is the start of the Command Decoder which occupies four Steps. If a PLEASE module is written which only contains a single program and therefore does not require a Command Decoder, then the first Function could be the start of the program itself. The entry point SETSTP is used to permit certain Functions to change the order of execution, that is cause a branch, by entering with the new next step value in the Accumulator. The entry point NXTSTP is the normal entry point where no modification of the Step Number is required. The subroutine jump to EXSET guarantees that the Monitor will gain control at least once during each Step, permitting the Keyboard to be tested for input, the Display to be refreshed, and the System Clock and Timer to be updated. On return from EXSET the Step Number is converted into the actual Step Location in memory. Valid Step Numbers range from ØØ to 27 and address locations ØØØØ to ØØ9C.

```
300  A9 00     DECODE  LDA  #00
302  85 B7     SETSTP  STA  STEPNO
304  20 D9 17  NXTSTP  JSR  EXSET
307  A5 B7             LDA  STEPNO
309  0A                ASL
30A  0A                ASL
30B  85 B8             STA  STEPLO
```

STEPLO now points to the first byte of the four byte Step. These four bytes are transferred to standard locations in Page Zero. The data is moved in reverse order: PARAM3, PARAM2, PARAM1, and PARAMØ. STEPNO is incremented in preparation for the next step.

```
30D  A0 03          LDY  #3
30F  A2 03          LDX  #3
311  B1 B8   PARMOV LDA  (STEPLO),Y
313  95 B0          STA  PARAMØ,X
315  CA             DEX
316  88             DEY
317  10 F8          BPL  PARMOV
319  E6 B7          INC  STEPNO
```

The Function Code and the three Parameters have now been moved to the standard Page Zero locations where they are readily available to the specific Functions. The Function Code itself is still in the Accumulator and is now multiplied by two via the ASL command to be used as an index into the Function Table which contains the two byte memory address of each Function. The low address byte is picked up and moved to a fixed location TRANLO. The high address byte is then picked up and moved to the fixed location TRANHI. An indirect jump is now made through the TRANLO/TRANHI location to the Function routine.

```
31B  0A             ASL
31C  A8             TAY
31D  B1 BC          LDA  (FUNTBL),Y
31F  85 BA          STA  TRANLO
321  C8             INY
322  B1 BC          LDA  (FUNTBL),Y
324  85 BB          STA  TRANHI
326  6C BA 00       JMP  (TRANLO)
```

The Function Table can be located anywhere in memory. A pointer to the Function Table is contained in a pair of locations in Page Zero FUNTBL and FUNTBH. PLEASE has its Function Table starting at location ØlØØ. Since each Function requires two bytes of address, and there are E standard Functions, these Functions require locations through Ø11B. Locations Ø11C through Ø11F are reserved for expansion of the standard Functions. Locations Ø12Ø through Ø12F are reserved for special Functions.

The TRANLO/TRANHI locations do not need to be preserved and are therefore freely available to the Function as temporary storage.

Every Function returns to the Interpreter when finished its operation. Some return via the SETSTP location when they are modifying the order of Step execution, that is are causing a branch. Most Functions return via the NXTSTP location which causes the next Step in sequence to be executed.

B0	PARAMØ	Parameter Ø is usually the COMMAND CODE
B1	PARAM1	Parameter 1 is usually DATA
B2	PARAM2	Parameter 2 is usually DATA
B3	PARAM3	Parameter 3 is usually DATA
B4	ADRLO	Low Address pointer for Indirect Address
B5	ADRHI	High Address pointer for Indirect Address
B6	PNTR	Temporary Pointer Storage
B7	STEPNO	Number of NEXT PLEASE Step
B8	STEPLO	Low Address of Current PLEASE Step
B9	STEPHI	High Address of Current PLEASE Step
BA	TRANLO	Temporary Transfer Pointer to PLEASE Function
BB	TRANHI	Temporary Transfer Pointer to PLEASE Function
BC	FUNTBL	Low Address of PLEASE Function Table
BD	FUNTBH	High Address of PLEASE Function Table
BE	PTEMPØ	PLEASE Temporary Storage
BF	PTEMP1	PLEASE Temporary Storage
C0	THOUS	Thousands and Tens of Thousands of Seconds
C1	TENS	Tens and Hundreds of Seconds
C2	TENTHS	Tenths and Seconds
C3	MILLI	Thousandths and Hundredths of Seconds
C4	HOUR	Hour portion of 24 Hour Clock
C5	MINUTE	Minute portion of 24 Hour Clock
C6	SECOND	Second portion of 24 Hour Clock
C7	ONESEC	Counter for One Second
C8	DSPØ	Display Position Ø (Leftmost Digit)
C9	DSP1	Display Position 1
CA	DSP2	Display Position 2
CB	DSP3	Display Position 3
CC	DSP4	Display Position 4
CD	DSP5	Display Position 5 (Rightmost Digit)
CE	DSPLO	Low Address of Display Buffer (Usually = DSPØ = C8)
CF	DSPHI	High Address of Display Buffer (Usually = DSPØ = 00)
D0	DCONLO	Display Conversion Table Low Address
D1	DCONHI	Display Conversion Table High Address
D2	HEXLO	Hexidecimal (and Decimal) Conversion Table Low Address = E7
D3	HEXHI	Hexidecimal (and Decimal) Conversion Table High Address = 17
D4	ALPHLO	Alphabetic Conversion Table Low Address (usually = FØ)
D5	ALPHHI	Alphabetic Conversion Table High Address (usually = 03)
D6	XTABLE	Used by Conversion Routine to Point to HEX or ALPHA Table
D7	TEMP	General Purpose Temporary Save Location
D8	LIMIT	Used by Conversion Routine. General Purpose Register
D9	DSPPOS	Executive Pointer to Current Display Position
DA	CURPNT	Used by Input Routines as Current Data Pointer.
DB	CHAR	Save Location for Input Character
DC	CTABLO	Command Table Low Address (usually = AØ)
DD	CTABHI	Command Table High Address (usually = ØØ)
DE	KEYTST	Used by Executive as part of Keyboard Input
DF	KEYVAL	Contains Last Character of Input String

DØ	BUFØ	General Purpose Buffer
E1	BUF1	
E2	BUF2	
E3	BUF3	
E4	BUF4	
E5	BUF5	
E6	ALTØ	Alternate General Purpose Buffer
E7	ALT1	
E8	ALT2	
E9	ALT3	
EA	ALT4	
EB	ALT5	
EC	APLØ	Application General Registers
ED	APL1	
EE	APL2	

Other PAGE ZERO Location Information

00 to 9F	PLEASE CODE, four bytes per STEP, max 40. (28) Steps.
00 to 0F	PLEASE CODE for COMMAND DECODER
A0 to AF	COMMAND TABLE contains two bytes per entry, max eight
EF to FF	Used by KIM-1 Monitor (8) Commands

EF	PCL	Program Counter - Low Order Byte
F0	PCH	Program Counter - High Order Byte
F1	P	Status Register
F2	SP	Stack Pointer
F3	A	Accumulator
F4	Y	Y-Index Register
F5	X	X-Index Register

Other MEMORY Location Information

100 to 11F	PLEASE Function Table, two bytes per entry, Low-High Address
120 to 12F	APPLICATION Specific Function Table
130 to 1EF	APPLICATION Specific Assembly Level Code
1F0 to 1FF	STACK
200 to 3FF	Standard PLEASE Functions Assembly Level Code
1780 to 17E6	PLEASE Executive
17E7 to 17FF	Used by KIM-1 Monitor
1800 to 1FFF	KIM-1 Monitor (ROM)

Function: ALPIN Code: ØØ

Purpose: Accept Alphabetic Input from the Keypad, store it in memory, convert to displayable characters, and display characters as input.

Command: ALPIN
Param1: Memory Address to Store Characters. Page Zero Locations.
Param2: Start Display Location. Leftmost = Ø, Rightmost = 5.
Param3: End Display Location. Leftmost = Ø, Rightmost = 5.

Param2 and Param3 control how many characters may be input - 1 to 6, where the characters get displayed within the six digit LED, and the mode of displaying - either fillin or shift. The number of characters equals the absolute difference between Param2 and Param3 plus 1. Where they are displayed is determined by Param2, the Start Display Location. The mode of displaying is fillin if the Start Location is less than the End Location. The mode of displaying is shift if the Start Location is greater than the End Location. If equal the mode is fillin.

The input data is stored as one character per byte in consecutive memory locations in Page Zero starting at the location specified by Param1. The order of storing is independent of the display mode. The first character typed is stored in the specified memory location, the second character in the next location,.... See page 9 for the Keypad to Alpha Conversions.

Function: HEXIN Code: Ø1

Purpose: Accept Hexidecimal Input from the Keypad, store it in memory, convert to displayable characters, and display characters as they are input.

Command: HEXIN
Param1: Memory Address to Store Characters. Page Zero Locations.
Param2: Start Display Location. Leftmost = Ø, Rightmost = 5.
Param3: End Display Location. Leftmost = Ø, Rightmost = 5.

See ALPIN above for details on operation and parameters.

Function: DECIN Code: Ø2

Purpose: Accept Decimal Input from the Keypad, store it in memory, convert to displayable characters, and display characters as they are input.

Command: DECIN
Param1: Memory Address to Store Characters. Page Zero Locations.
Param2: Start Display Location. Leftmost = Ø, Rightmost = 5.
Param3: End Display Location. Leftmost = Ø, Rightmost = 5.

See ALPIN above for details on operation and parameters.

ALPIN, HEXIN, and DECIN use the same source code. The only difference is in the conversion tables used and the input limits set. These differences are taken care of by short initialization sequences as follows.

ALPIN uses the entry point ALPHA. It sets up a pointer to the Alpha Table and sets the input limit to 1Ø which permits input in the range Ø to F.

```
329  A9 10     ALPHA   LDA  #10
32B  A2 D4             LDX  #ALPHLO
32D  D0 08             BNE  SETTAB
```

HEXIN uses the entry point HEXIN. It sets up a pointer to the Hexidecimal Table and sets the input limit to 1Ø which permits input in the range Ø to F.

```
32F  A9 10     HEXIN   LDA  #10
331  D0 02             BNE  SETHEX
```

DECIN uses the entry point DECIN. It sets up a pointer to the Hexidecimal Table and sets the input limit to 1Ø. which permits input in the range Ø to 9. A call is then made to the subroutine DIRADR which converts the Paraml address into a two byte address which is stored in a pair of Page Zero locations ADRLO and ADRHI for general usage.

```
333  A9 0A     DECIN   LDA  #0A
335  A2 D2     SETHEX  LDX  #HEXDEC
337  86 D6     SETTAB  STX  XTABLE
339  85 D8             STA  LIMIT
33B  20 83 03          JSR  DIRADR
```

RSTART calls a subroutine CLRDSP which clears that portion of the display specified by Param2 and Param3 as the display limits. It then sets a value called current pointer CURPNT to equal Param2 the start location. This initializes the input routine and is also used to reinit in the event that a special character to clear the display is received, the PC key. The PC key may be used at any time to clear that portion of the display under control of the current function. Remember that as little as one character may be specified by these input functions, and that the display may at any time consist of sections which were input or generated by other functions. Param2 and Param3 totally define the portion of the display that will be affected by the current function.

```
33E  20 97 03  RSTART  JSR  CLRDSP
341  A5 B2            LDA  PARAM2
343  85 DA            STA  CURPNT
345  10 03            BPL  MORE
```

SAVE is skipped over the first time since there is no input data to be stored. MORE performs several functions. First it calls a subroutine to convert the raw data in the Param1 defined buffer to displayable form in the display buffer. It specifies the conversion table by loading X with a pointer to the correct conversion table. CONDSP is on page 18. A call is then made to EXSET to permit the Monitor to gain control for a millisecond cycle and to permit keypad input. The location CHAR will contain a 20 if there has been no input or the value of the input if there was any. The CHAR is moved to A and the location reset to 20. Then a test is made for the "erase" key, the PC key. If the last character was a PC then the program branches to RSTART. If there was no character, then the program branches to MORE. If there was a character, then it is tested for the limit. If the input character is less then the limit, then it is saved by branching to SAVE which calls a store the character subroutine STORE. If the input character is equal to or greater than the limit, then the function exits to the next step NXTSTP after storing the final character value in KEYVAL for testing by other functions.

```
347  20 D0 03  SAVE    JSR  STORE
34A  A6 D6     MORE    LDX  XTABLE
34C  20 A9 03          JSR  CONDSP
34F  20 D9 17          JSR  EXSET
352  A5 DB             LDA  CHAR
354  A2 20             LDX  #20
356  86 DB             STX  CHAR
358  C9 14             CMP  #PC
35A  F0 E2             BEQ  RSTART
35C  10 EC             BPL  MORE
35E  C5 D8             CMP  LIMIT
360  30 E5             BMI  SAVE

362  85 DF     FINISH  STA  KEYVAL
364  10 9E             BPL  NXTSTP
```

Any of the "control" keys except PC will cause termination of the input function. This means that GO, +, AD or DA may be used as valid terminators. Because the final character is saved in KEYVAL, a BRTABL Function can be used to branch within the program as a function of the terminator. This feature is used in the ADDSUB Program.

Subroutine: DIRADR
Purpose: Move Page Zero Direct Address value from Param1 to the full address pointers ADRLO/ADRHI for general use.

```
383  A5 B1     DIRADR  LDA  PARAM1
385  85 B4             STA  ADRLO
387  A9 00             LDA  #00
389  85 B5             STA  ADRHI
38B  60                RTS
```

Subroutine: INDADR

Purpose: To move a pair of Page Zero bytes which contain a full memory address from the location pointed to by Param1 to the full address pointers ADRLO/ADRHI for general use. This permits a full two byte address to be referenced by a single byte parameter. This subroutine is used by the Branch on Table Lookup Function BRTABL to point to the table to be searched.

```
38C  A6 B1     INDADR  LDX  PARAM1
38E  B5 00     XNDADR  LDA  0,X
390  85 B4             STA  ADRLO
392  B5 01             LDA  1,X
394  85 B5             STA  ADRHI
396  60                RTS
```

Subroutine: CLRDSP

Purpose: To Clear that portion of the Display defined by Param2 and Param3 by filling it with nulls.

Subroutine: FILDSP

Purpose: To Fill that portion of the Display defined by Param2 and Param3 with the value contained in the Accumulator.

These two subroutines use the same code except that CLRDSP sets a value of FF in the Accumulator as a special value which will be converted to a blank by the CONDSP subroutine.

```
397  A9 FF     CLRDSP  LDA  #FF
399  A2 06     FILDSP  LDX  #06
39B  86 D7             STX  TEMP
39D  A4 B2             LDY  PARAM2
39F  84 DA             STY  CURPNT

3A1  20 D0 03  FILNXT  JSR  STORE
3A4  C6 D7             DEC  TEMP
3A6  D0 F9             BNE  FILNXT
3A8  60                RTS
```

These routines actually do not directly effect the display. They work by modifying the data in the memory storage area which is then converted and moved to the Display buffer by the CONDSP routine. They are special purpose routines and are intended for use with the Keypad Input routines ALPIN, HEXIN and DECIN. There is a FILL Function for more general memory filling and clearing.

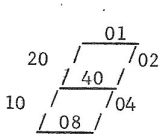
Subroutine: CONDSP

Purpose: Convert data from binary form to LED Segment Display Format.

The KIM-1 Display consists of six elements each of which in turn is composed of seven independent light emitting diode LED segments. Each segment is turned on or off under program control. The seven segments each have a binary value as shown in the diagram.

For example, to show the digit "8" all segments are turned on by putting a 7F on the display output lines. 7F = 40 + 20 + 10 + 08 + 04 + 02 + 01. The letter "L" has a value of 38 = 20 + 10 + 08. Since the display values of the various characters bear no relation to their binary form, a translation is required. The binary value 08 will be displayed as an eight "8" in Hexidecimal or Decimal mode and as ell "L" in Alpha mode. The convert to display subroutine CONDSP performs this conversion. On entry X contains the address of a Page Zero location which has the low order address of a conversion table, followed by a byte which contains the high order address of the table. Normally PLEASE supports two tables: ALPHA and HEXIDECIMAL as described on page 9. The programmer could make other tables to suit his own needs. CONDSP moves the table address to a standard Page Zero pair of locations for easy use. It picks up the Param2 address for Start Location. Locations ADRLO/ADRHI point to the buffer.

```
3A9  B5 00     CONDSP  LDA  0,X
3AB  85 D0             STA  DTABLO
3AD  B5 01             LDA  1,X
3AF  85 D1             STA  DTABHI
3B1  A4 B2             LDY  PARAM2
```



CON saves the current location. It then picks up the character in the buffer and tests it for a negative value. Normally the value of a character being output must range between Ø and F, or Ø and 9 for decimal. A negative value is used to cause a blank output. This is why CLRDSP filled the buffer with FF not ØØ. A positive value is used to perform the table lookup for the translated value. The value from the table is placed in the display buffer. Tests are made against Param3 to determine if all positions have been converted, or if not which direction to shift the pointer to handle both fillin and shift modes of input.

```
3B3  84 D7     CON     STY  TEMP
3B5  B1 B4             LDA  (ADRLO),Y
3B7  10 04             BPL  CON2
3B9  A9 00             LDA  #00
3BB  F0 03             BEQ  CON3
3BD  A8        CON2    TAY
3BE  B1 D0             LDA  (DTABLO),Y
3C0  A4 D7     CON3    LDY  TEMP
3C2  91 CE             STA  (DSPLO),Y
3C4  C4 B3             CPY  PARAM3
3C6  F0 27             BEQ  RETURN
3C8  30 03             BMI  INCR
3CA  88                DEY
3CB  10 E6             BPL  CON
3CD  C8        INCR    INY
3CE  10 E3             BPL  CON
```

Subroutine: STORE

Purpose: To place data in a buffer with parameters controlling the amount of data and the mode of storing: fillin or shift.

STORE is an integral part of the Keypad Input routines. Before it is called the address of the storage buffer is placed in the standard Page Zero locations ADRLO/ADRHI. Param2 determines the Start of storage within the buffer, Param3 determines the End, and the relative values of Param2 and Param3 determine whether the mode is fillin or shift. The character to be stored is in the Accumulator on entry. The character is saved in X. A test is made for mode by comparing Param2 and Param3. If Param3 is greater than Param2 or equal, then fillin mode is used. A test is made for end of buffer by comparing Param3 with the current pointer CURPNT. If Param3 is greater than the CURPNT, then CURPNT is incremented and the character is stored. If Param3 is less than CURPNT, then an immediate return is made and no character is stored.

```
3D0  AA        STORE   TAX
3D1  A4 B3             LDY  PARAM3
3D3  C4 B2             CPY  PARAM2
3D5  30 0A             BMI  SHIFT
3D7  C4 DA             CPY  CURPNT
3D9  30 14             BMI  RETURN
3DB  A4 DA             LDY  CURPNT
3DD  E6 DA             INC  CURPNT
3DF  10 0C             BPL  PUT
```

If Param3 was less than Param2 then shift mode is used. SHIFT first moves all characters within the portion of the buffer defined by Param2 and Param3 one position up in memory. When all characters have been shifted it gets the new character back from X and stores it in the buffer at the last position as defined by Param3.

```
3E1  C8        SHIFT   INY
3E2  B1 B4             LDA  (ADRLO),Y
3E4  88                DEY
3E5  91 B4             STA  (ADRLO),Y
3E7  C8                INY
3E8  C4 B2             CPY  PARAM2
3EA  30 F5             BMI  SHIFT
3EC  8A                TXA
3ED  91 B4     PUT     STA  (ADRLO),Y
3EF  60        RETURN  RTS
```

Function: ALPOUT Code: Ø3

Purpose: Convert and Display data in Alpha mode, using standard PLEASE Alpha Table.

Command: ALPOUT
Param1: Memory Address of Buffer. Page Zero Locations.
Param2: Start Display Location. Leftmost = Ø, Rightmost = 5.
Param3: End Display Location. Leftmost = Ø, Rightmost = 5.

Data to be output should be in range ØØ to ØF for real data or FF for a blank. Other values will give strange results. The Alpha Table on page 9 shows the standard conversions. The order of the Start and End does not matter. Start less than End will be handled as fillin and End less than Start will be handled as shift, but since the whole conversion takes place within a single millisecond cycle there will be no visible difference.

Function: HEXOUT or DECOUT Code: Ø4

Purpose: Convert and Display data in Hexidecimal mode, using standard KIM-1 Hexidecimal Table.

Command: HEXOUT or DECOUT
Param1: Memory Address of Buffer. Page Zero Locations.
Param2: Start Display Location. Leftmost = Ø, Rightmost = 5.
Param3: End Display Location. Leftmost = Ø, Rightmost = 5.

Data to be output should be in range ØØ to ØF for Hex or ØØ to Ø9 for Decimal. A value with bit 8Ø set will produce a blank. Values in the range 1Ø to 7F will give strange results. See ALPOUT comments above for other details. HEXOUT is the entry point for HEXOUT/DECOUT.

```
283  A2 D4     ALPOUT  LDX  #ALPHLO
285  D0 02             BNE  GETADR
287  A2 D2     HEXOUT  LDX  #HEXDEC
289  20 83 03  GETADR  JSR  DIRADR
28C  20 A9 03        JSR  CONDSP
28F  4C 04 03  TONEXT  JMP  NXTSTP
```

For an example of the use of ALPOUT see the HiLo program.
For an example of the use of HEXOUT and DECOUT see the Decimal/Hexidecimal Conversion program.

Function: TIMER Code: Ø5

Purpose: Provide a programmable delay while maintaining the basic system operations of Display Refresh and System Clock Updating.

Command: TIMER
Param1: Delay Time in Tenths of a Second. Range 0.1 to 25.6 seconds.
Param2: Step Number to Branch to when Delay is done.
Param3: not used

There are a number of uses for a timer. Several programs flash all or part of the Display. This is easily accomplished by changing all or part of the Display, waiting a set period of time using TIMER, changing the Display again, waiting, and so forth. Another use for TIMER is in TIPSY in which the Player is shown the test numbers for 2 seconds and then must wait 1 second before typing his answer. The basic unit of TIMER is 0.1 seconds. This is generated from the Monitor Millisecond loop by setting up two counters to provide a 100 millisecond = 0.1 second delay. Intermediate returns are made to the Monitor to permit the basic system operations to take place and to complete the millisecond loop. After 0.1 seconds have been timed, the Delay Time is decremented and tested. If the Delay Time has gone to zero, then the next Step Number is pickup and a return is made to the Interpreter at the Set Step SETSTP location. Delay Time is held in Param1 and next Step Number in Param2.

```
2AA  A9 0A     TIMER   LDA  #10.
2AC  85 BE             STA  TEMP1
2AE  A9 0A     RESET   LDA  #10.
2B0  85 BF             STA  TEMP2
2B2  20 D9 17  WAIT    JSR  EXSET
2B5  C6 BF             DEC  TEMP2
2B7  D0 F9             BNE  WAIT
2B9  C6 BE             DEC  TEMP1
2BB  D0 F1             BNE  RESET
2BD  C6 B1             DEC  PARAM1
2BF  D0 E9             BNE  TIMER
2C1  A5 B2             LDA  PARAM2
2C3  10 3D             BPL  SETSTP
```

Warning: Remember that the Delay Time in Param1 is in tenths of a second.

Note: Obviously only a single counter is required to keep a 100. count. The double counter is provided so that a longer base interval can be generated by changing the two counter constants. The maximum base interval would be slightly over one minute (65,536 milliseconds) which means that a delay can be specified of over four hours (256+ minutes) if the base interval is changed. An even longer interval could be generated by having multiple TIMER calls. However, since the TIMER is based on the crystal and a slight variation in time can cause a large error when accumulated over millions of cycles, this TIMER will not be very accurate for long periods.

Function: PACK Code: Ø6

Purpose: To take a series of characters and pack them two to a byte.

Command: PACK
Param1: Source Buffer which contains Characters. Page Zero.
Param2: Destination Buffer or Byte to receive result. Page Zero.
Param3: Number of Bytes in the Destination

While it is convenient to input characters in four bit chunks and keep them that way for displaying, it is often an inconvenient form for processing data. PACK provides a simple method of combining the raw input data into compact bytes. An input character must be in the range ØØ to ØF. Two such characters fit into a single byte. PACK can handle as little as one byte or as many as 256. PACK picks up the first character of the pair to be packed and shifts it into the Most Significant Bits of the Accumulator and then performs a logical inclusive Or to get the second character into the Least Significan Bits. It then tests the Carry bit to see if a dummy blank character FF was the first character. If it was it loads the second character into the Accumulator and test if it was a blank character. If it was it loads zero as the result of packing two blank characters. If the first or second character is not a blank, then the combined characters are now stored as a single packed byte. This packing of character pairs continues until the requested number of bytes have been created.

```
23B  A6 B1     PACK    LDX  PARAM1
23D  B5 00             LDA  0,X
23F  0A                ASL
240  0A                ASL
241  0A                ASL
242  0A                ASL
243  15 01             ORA  1,X
245  90 06             BCC  PACK2
247  B5 01             LDA  1,X
249  10 02             BPL  PACK2
24B  A9 00             LDA  #00
24D  A6 B2     PACK2   LDX  PARAM2
24F  95 00             STA  0,X
251  E6 B1             INC  PARAM1
253  E6 B1             INC  PARAM1
255  E6 B2             INC  PARAM2
257  C6 B3             DEC  PARAM3
259  D0 E0             BNE  PACK
25B  F0 32             BEQ  TONEXT
```

For examples of PACK in use, see the DECODE program, SCLOCK program, or STIMER program.

Function: UNPACK Code: Ø7

Purpose: To separate bytes into characters, normally in order to make them available for displaying.

Command: UNPACK
Param1: Source Buffer which contains Bytes. Page Zero.
Param2: Destination Buffer to receive Characters. Page Zero.
Param3: Number of Characters in the Destination.

Normally data is processed in a packed form or binary form. Since the display only uses values in the range ØØ to ØF for its normal character generation, a means of converting from packed to unpacked form is required. UNPACK performs this function. One or more bytes are split into their component characters. The Most Significant Bits are used as the first character of a pair, the Least Significant Bits as the second character. UNPACK can be used to generate from one to 256 characters. UNPACK picks up the byte to be unpacked, shifts off the second character and stores the first character in the destination buffer. It then tests for any more characters required. If no more, then it is done and goes to the next step NXTSTP in the Interpreter. If more is required it masks off the second character from the same byte and stores it in the destination buffer. After some housekeeping, it test for any more characters required. If so it gets the next byte and continues. If no more are required it goes to the next step NXTSTP in the Interpreter.

```
25D  A6 B1     UNPACK  LDX  PARAM1
25F  B5 00             LDA  0,X
261  4A                LSR
262  4A                LSR
263  4A                LSR
264  4A                LSR
265  A6 B2             LDX  PARAM2
267  95 00             STA  0,X
269  E6 B2             INC  PARAM2
26B  C6 B3             DEC  PARAM3
26D  F0 1E             BEQ  TONEXT
26F  A6 B1             LDX  PARAM1
271  A9 0F             LDA  #0F
273  35 00             AND  0,X
275  A6 B2             LDX  PARAM2
277  95 00             STA  0,X
279  E6 B2             INC  PARAM2
27B  E6 B1             INC  PARAM1
27D  C6 B3             DEC  PARAM3
27F  D0 DC             BNE  UNPACK
281  F0 0C             BEQ  TONEXT
```

For examples of UNPACK in use see DCLOCK program, DTIMER program, or the Shooting Stars program.

Function: BRANCH Code: Ø8

Purpose: Provides an Unconditional Branch to any specified Step.

Command: BRANCH
Param1: Step to Branch To.
Param2: not used
Param3: not used

Almost every conceivable PLEASE program has one or more loops in it or is itself a large loop. The BRANCH function permits transfer of control to a Step other than the next sequential Step. BRANCH picks up the Step number from Param1 and then goes to the Set Step SETSTP entry of the Interpreter.

```
2A6  A5 B1     BRANCH  LDA  PARAM1
2A8  10 58             BPL  SETSTP
```

Function: BRCHAR Code: Ø9

Purpose: Provide Program Control from the Keypad.

Command: BRCHAR
Param1: Character for Match.
Param2: Step to Branch to on Match.
Param3: Step to Branch to on Non-match.

BRCHAR picks up the current value of the Page Zero location CHAR which contains the last input character or a 20 if there is no valid input. It clears the CHAR location and tests for valid input. If there is no character ready, then a return is made to the Next Step NXTSTP entry of the Interpreter and the next Step is executed. If there is a character it is tested for a match with the specified match character in Param1. If they match, then the Match Step in Param2 is used as the next step. If they do not match, then the Non-match Step in Param3 is used. In either case the parameter value is loaded into the Accumulator and control goes to the Set Step SETSTP entry of the Interpreter.

```
292  A6 DB     BRCHAR  LDX  CHAR
294  A9 20             LDA  #20
296  85 DB             STA  CHAR
298  E0 20             CPX  #20
29A  F0 F3             BEQ  TONEXT
29C  A5 B2             LDA  PARAM2
29E  E4 B1             CPX  PARAM1
2A0  F0 02             BEQ  OKAY
2A2  A5 B3             LDA  PARAM3
2A4  10 5C     OKAY    BPL  SETSTP
```

For examples of BRCHAR see DCLOCK program or DTIMER program.

Function: BRTABL Code: ØA

Purpose: Provide Program Control as a function of a Table Lookup.

Command: BRTABL
Param1: Page Zero Indirect Address of Table.
Param2: Direct Address of Value to be Matched. Page Zero.
Param3: Step Number to Branch to if No Match.

This is a very powerful method of branching. A value may be generated by any means such as characters from the Keypad, result of a calculation, or whatever. A table is set up which contains the permitted values and the step number to go to if that value occurs. Many value can branch to the same Step. Every value may branch to a different Step. The table could itself be modified and/or generated under the program. BRTABL starts by calling the INDADR subroutine to move the table address into the standard locations ADRLO/ADRHI. It then picks up the test value via Param2 and compares it with a value in the table. If a match is found, then the next byte in the table is picked up and used as the next step number by going to the Set Step SETSTP entry of the Interpreter. If it is not a match, then the pointers are moved to the next value in the table which is checked for zero. If the next value is a zero it indicates the end of the table and a no match condition. In this case the value of Param3 is used as the nest step number via the Set Step SETSTP entry to the Interpreter. If the next value is not a zero, then BRTABL continues testing until either a match is found or the end of the table is reached.

```
366  20 8C 03  BRTABL  JSR  INDADR
369  A6 B2             LDX  PARAM2
36B  A0 00             LDY  #00

36D  B5 00     MORE    LDA  0,X
36F  D1 B4             CMP  (ADRLO),Y
371  F0 0A             BEQ  FOUND
373  C8                INY
374  C8                INY
375  B1 B4             LDA  (ADRLO),Y
377  D0 F4             BNE  MORE
379  A5 B3             LDA  PARAM3
37B  10 85     BRDONE  BPL  SETSTP

37D  C8        FOUND   INY
37E  B1 B4             LDA  (ADRLO),Y
380  10 F9             BPL  BRDONE
```

The table has two bytes per entry. The first byte is the value to be tested. The second value is the Step Number to go to on a match. The table must be terminated by a zero value, therefore, zero may not be used for a test value unless it is the first test value which is matched before the end of table test is performed. For examples of BRTABL see the DECODE program or the ADDSUB program.

Function: FILL Code: ØB

Purpose: Provide a means of filling a section of Page Zero memory with a specified value.

Command: FILL
Param1: Memory Address of Start of Page Zero Memory to be Filled.
Param2: Value to Fill Memory with.
Param3: Number of bytes of Memory to Fill.

Since all of the working locations, buffers, and so forth for PLEASE are in Page Zero, this limited Fill command is sufficient for most purposes. It is used to initialize buffers, clear specific bytes, clear the Display via its buffer, etc. FILL first calls the DIRADR subroutine to get the Page Zero address to be modified into the standard ADRLO/ADRHI location. It then picks up the Fill value from Param2 and proceeds to fill memory starting at the last location to be modified and working back to the first location. When finished it returns via the Next Step NXTSTP entry in the Interpreter.

```
2E9  20 83 03  FILL    JSR  DIRADR
2EC  A5 B2             LDA  PARAM2
2EE  A4 B3             LDY  PARAM3
2F0  88                DEY

2F1  91 B4     FILLIT  STA  (ADRLO),Y
2F3  88                DEY
2F4  10 FB             BPL  FILLIT
2F6  30 0C             BMI  NXTSTP
```

For examples of FILL see the DECODE program or the DAFFY program.

Function: COMPARE Code: ØC

Purpose: Provide a String Compare with a Three-way Branch on Less, Equal, or Greater.

Command: COMPAR
Param1: Page Zero Address of Test Value Start
Param2: Page Zero Address of String to be Compared with.
Param3: Number of Bytes to be Compared.

Function: MATCH Code: ØD

Purpose: Provide a String Compare with a Two-way Branch on Equal or Not Equal.

Command: MATCH
Param1: Page Zero Address of Test Value Start
Param2: Page Zero Address of String to be Matched to.
Param3: Number of Bytes to be Matched.

These two functions use essentially the same code. The only difference is a test which causes MATCH to treat Less and Greater as a single condition: Not Equal. First a call is made to the DIRADR subroutine to get the Test String address in the standard ADRLO/ADRHI locations. A byte from the second string is picked up. A byte from the first string is subtracted from the second string byte and the result tested. If the first string byte is less than the second string, then testing is complete and control is returned to the Interpreter through the Next Step NXTSTP entry point. If the first string byte is greater than the second string byte, then an additional test is made to determine if this is a COMPAR or MATCH. If a MATCH, then it is treated identically as the less. If a COMPAR, then the Step Number STEPNO is incremented twice to cause the next two Steps to be skipped. If the first and second bytes are equal, then the number of bytes to test is checked. If all bytes have been tested, then the Step Number STEPNO is increment once to cause the next Step to be skipped. If all bytes have not been tested, then testing continues with the next pair of bytes.

COMPAR and MATCH Branch Summary: First String to Second String

COMPARE		MATCH	
Less	Skip Ø Steps	Less	Skip Ø Steps
Equal	Skip 1 Step	Equal	Skip 1 Step
Greater	Skip 2 Steps	Greater	Skip Ø Steps

```
2C5  20 83 03  COMPAR  JSR  DIRADR
2C8  A6 B2             LDX  PARAM2
2CA  A0 00             LDY  #00

2CC  B5 00     CTEST   LDA  0,X
2CE  38                SEC
2CF  D1 B4             SBC  (ADRLO),Y
2D1  F0 0E             BEQ  SAME
2D3  B0 0A             BCS  LESS

2D5  A9 0D     GREAT   LDA  #MATCH
2D7  C5 B0             CMP  PARAMØ
2D9  F0 04             BEQ  LESS
2DB  E6 B7             INC  STEPNO
2DD  E6 B7     EQUAL   INC  STEPNO

2DF  10 23     LESS    BPL  NXTSTP

2E1  E8        SAME    INX
2E2  C8                INY
2E3  C4 B3             CPY  PARAM3
2E5  D0 E5             BNE  CTEST
2E7  F0 F4             BEQ  EQUAL
```

For examples of COMPARE see the HILO program.
For examples of MATCH see the TIPSY program.

Program Module #1

LOC	OBJECT	STEP	LABEL	COMMAND	PARAM1	PARAM2	PARAM3
				Command Decoder			
00	00E00005	Ø	DECODE	ALPIN	BUFFER	Ø	5
04	06E0DF01	1		PACK	BUFFER	KEYVAL	1
08	0BC80006	2		FILL	DISPLAY	Ø	6
0C	0ADCDF00	3		BRTABL	CMDTBL	KEYVAL	DECODE
				Set Clock and Display Clock			
10	02E00005	4	SCLOCK	DECIN	BUFFER	Ø	5
14	06E0C403	5		PACK	BUFFER	HOUR	3
18	07C4E006	6	DCLOCK	UNPACK	HOUR	BUFFER	6
1C	04E00005	7		DECOUT	BUFFER	Ø	5
20	09130004	8		BRCHAR	GO	DECODE	SCLOCK
24	08060000	9		BRANCH	DCLOCK		
				Set Timer and Display Timer			
28	02E00500	A	STIMER	DECIN	BUFFER	5	Ø
2C	06E0C103	B		PACK	BUFFER	TENS	3
30	07C1E006	C	DTIMER	UNPACK	TENS	BUFFER	6
34	04E00005	D		DECOUT	BUFFER	Ø	5
38	0913000A	E		BRCHAR	GO	DECODE	STIMER
3C	080C0000	F		BRANCH	DTIMER		
				Notice and Billboard			
40	11D0010A	10	NOTICE	MESAGE	MMSGLO	MSGHI	10.
44	09130013	11		BRCHAR	GO	DECODE	BILBRD
48	08100000	12		BRANCH	NOTICE		
4C	12CA0104	13	BILBRD	BBOARD	BMSGLO	MSGHI	4.
50	09130010	14		BRCHAR	GO	DECODE	NOTICE
54	08130000	15		BRANCH	BILBRD		
				Daffy			
58	0BE00006	16	DAFFY	FILL	ANSWER	Ø	6
5C	07C1E004	17		UNPACK	TENS	ANSWER	4
60	02E60003	18	GET	DECIN	GUESS	Ø	3
64	10E6EC20	19		MASTER	GUESS	ANSWER	BAD
68	04E00005	1A	GOOD	DECOUT	ANSWER	Ø	5
6C	05021C00	1B		TIMER	2.	BLANK	
70	0BCC0002	1C	BLANK	FILL	DISPLAY+4	Ø	2
74	05011E00	1D		TIMER	1	WAIT	
78	09130016	1E	WAIT	BRCHAR	GO	DECODE	DAFFY
7C	081A0000	1F		BRANCH	GOOD		
80	04E60405	20	BAD	DECOUT	GUESS	4	5
84	09130018	21	HOLD	BRCHAR	GO	DECODE	GET
88	08210000	22		BRANCH	HOLD		

Program Module #1 Notes and Comments

DECODE Command Decoder

This program is called when the system is initialized or when various programs are complete. Its purpose is to allow the user to select a program from the keypad. ALPIN is used to input the program name from the keypad. At least two characters of the name must be typed, additional characters are optional. PACK is used to combine the first two characters into a single byte for testing by BRTABL. FILL is used to clear the display by filling the display buffer with nulls. BRTABL is used to lookup in the Command Table CMDTBL the Step Number STEPNO of the requested program and to transfer control to that program.

SCLOCK and DCLOCK Set Clock and Display Clock

These programs set the system clock and display the current contents of the system clock. DECIN is used to input up to six characters in the fillin mode. PACK converts the input characters into three bytes in the system clock. UNPACK converts the current contents of the system clock into separate characters. DECOUT converts these characters into their displayable form and moves them to the display buffer. BRCHAR returns to DECODE if the GO key is pressed; goes to SCLOCK if any other key is pressed; or goes to the next step if no key is pressed. BRANCH which is reached by no key being pressed goes to DCLOCK so that the display is constantly updated with the current value of the system clock.

STIMER and DTIMER Set Timer and Display Timer

These programs set the system timer and display the current contents of the system timer. DECIN is used to input up to six characters in the shift mode. The remainder of the program is identical to SCLOCK and DCLOCK described above.

NOTICE and BILBRD Notice and Billboard Message Display

These programs display a message in alphabetic characters. NOTICE displays six characters per frame for one second per frame. BILBRD displays its characters by shifting them in one at a time from the right to the left at a shift rate of one shift every 0.4 seconds. MESAGE/BBOARD call the same routine, the only difference being an internal test performed on the value of the command MESAGE = 11 and BBOARD = 12 which determines how the data is presented. BRCHAR goes to DECODE on a GO key; goes to the other program on any other key NOTICE to BILBRD or BILBRD to NOTICE; or goes to the next step if no key is pressed. BRANCH goes to restart the program which causes the message to be repeated continuously until some key is pressed.

DAFFY The Daffy (Mastermind) Number Guessing Game

FILL clears the answer buffer. UNPACK is used to create an answer from the system timer. DECIN gets the users four digit decimal guess. MASTER is special code to evaluate the guess. It compares GUESS and ANSWER and if not a match, branches to BAD. On a perfect match it drops through to GOOD. The remainder of the program displays the number of guesses in a flashing mode on GOOD or the evaluation of the guess on BAD.

MESAGE and BBOARD special functions..

The NOTICE program uses a special function MESAGE to output the "canned" message six characters at a time. The BILBRD program uses a special function BBOARD to output the "canned" message one character at a time producing a moving message. Actually both MESAGE and BBOARD are the same routine, differing only by a single internal test which causes the six or one character change in the display. Both functions are called by entering at the MESAGE entry point. The value of the Command is used to determine the particular function.

Notice and Billboard

```
LOC  OBJECT     LABEL    OPERATION
190  A0 00      MESAGE   LDY  #00          Initialize start of message.
192  84 BB               STY  PLACE
194  A2 00      MORE     LDX  #00          Get next character.
196  B1 B1      FETCH    LDA  (PARAM1),Y   If character is minus, then
198  10 03               BPL  OKAY         end of message.
19A  4C 04 03            JMP  NXTSTP
19D  95 C8      OKAY     STA  DSPØ,X       Store character is display
19F  C8                  INY               buffer.  Bump pointers.
1A0  E8                  INX
1A1  E0 06               CPX  #6           Test six characters done.
1A3  D0 F1               BNE  FETCH        If not, get next.
1A5  A5 BB               LDA  PLACE        Get message place pointer.
1A7  A2 12               LDX  #12          Test MESAGE or BILBRD
1A9  E4 B0               CPX  PARAMØ       If MESAGE, then move place
1AB  F0 02               BEQ  INCR         pointer forward six places.
1AD  69 05               ADC  #5           If BILBRD, then move one
1AF  69 00      INCR     ADC  #0           place.
1B1  85 BB               STA  PLACE        Save modified PLACE.
1B3  A5 B3               LDA  PARAM3       Get Delay from PARAM3
1B5  85 BE               STA  PTEMPØ
1B7  A9 64      SETIME   LDA  #100.        Set 1/10 second timer.
1B9  85 BF               STA  PTEMP1
1BB  20 D9 17   WAIT     JSR  EXSET        Wait 1 millisecond
1BE  C6 BF               DEC  PTEMP1       Bump 1/10 second counter
1C0  D0 F9               BNE  WAIT         until zero.
1C2  C6 BE               DEC  PTEMPØ       Then bump Delay counter
1C4  D0 F1               BNE  SETIME       until zero.
1C6  A4 BB               LDY  PLACE        Now get next frame of the
1C8  10 CA               BPL  MORE         message.
```

Message for Notice and Billboard

```
1CA   00 00 00 00 00 00        Blanks for Billboard
1D0   73 38 79 77 6D 79        PLEASE
1D6   00 39 77 37 00 30         CAN I
1DC   00 76 79 38 73 00         HELP
1E2   00 53 00 00 00 00         ?
1E8   00 00 00 00 00 00        Trailing blanks
1EE   FF                       Terminator
```

Daffy

```
LOC  OBJECT     LABEL    OPERATION

130  20 83 03   MASTER   JSR  DIRADR      Address of Guess
133  A6 B2               LDX  PARAM2      Bump Guesses Counter
135  F6 05               INC  5,X
137  A9 0A               LDA  #0A         Test Units digit = 10.
139  D5 05               CMP  5,X
13B  D0 06               BNE  TEST
13D  A9 00               LDA  #0          Set Units = 0.
13F  95 05               STA  5,X         Incr Tens digit
141  F6 04               INC  4,X

143  A9 00      TEST     LDA  #0          Clear Evaluation
145  85 BE               STA  PTEMPØ      Counters
147  85 BF               STA  PTEMP1
149  A0 03               LDY  #3          Set Digit Counter
14B  B1 B4      PTEST    LDA  (ADRLO),Y   Get a Guess Character
14D  D5 03               CMP  3,X         Test Correct Char.
14F  D0 02               BNE  NOTPER      in Correct Location.
151  E6 BE               INC  PTEMPØ      Bump Counter
153  CA         NOTPER   DEX              Test all four Guess
154  88                  DEY              Characters.
155  10 F4               BPL  PTEST

157  A6 B2               LDX  PARAM2      Test Correct without
159  A9 03               LDA  #3          regard to position.
15B  85 D7               STA  TEMP
15D  A0 03      SETUP    LDY  #3
15F  B5 03      MATCH    LDA  3,X         Get Answer Digit
161  D1 B4               CMP  (ADRLO),Y   Test Match
163  D0 08               BNE  NMATCH
165  A9 FF               LDA  #FF         If Match, wipe out the
167  91 B4               STA  (ADRLO),Y   Guess digit to prevent
169  E6 BF               INC  PTEMP1      multiple matches.
16B  10 03               BPL  NEXT
16D  88         NMATCH   DEY              If No-match, keep trying
16E  10 EF               BPL  MATCH
170  CA         NEXT     DEX              Get next Answer digit
171  C6 D7               DEC  TEMP        until all four done.
173  10 E8               BPL  SETUP

175  A0 04               LDY  #4          Test Perfect Match on
177  C4 BE               CPY  PTEMPØ      all four digits.
179  F0 04               BEQ  DONE        If so, then done.
17B  A5 B3               LDA  PARAM3      Else, use PARAM3 for next
17D  85 B7               STA  STEPNO      Step Number.
17F  A5 BE      DONE     LDA  PTEMPØ      Move Evaluation counters
181  91 B4               STA  (ADRLO),Y   to Guess Buffer for
183  C8                  INY              displaying.
184  A5 BF               LDA  PTEMP1
186  91 B4               STA  (ADRLO),Y
188  4C 04 03            JMP  NXTSTP
```

Program Module #2

LOC	OBJECT	STEP	LABEL	COMMAND	PARAM1	PARAM2	PARAM3
				Command Decoder			
00	00E00005	Ø	DECODE	ALPIN	BUFFER	Ø	5
04	06E0DF01	1		PACK	BUFFER	KEYVAL	1
08	0BC80006	2		FILL	DISPLAY	Ø	6
0C	0ADCDF00	3		BRTABL	CMDTBL	KEYVAL	DECODE
				Shooting Stars			
10	10000000	4	STAR	START			
14	01E00303	5	NEXT	HEXIN	BUFFER	3	3
18	110B0E00	6		SHOT	LOSE	WIN	
1C	07E6E402	7	RUN	UNPACK	COUNT	BUFFER+4	2
20	04E00405	8		DECOUT	BUFFER	4	5
24	09130005	9	RWAIT	BRCHAR	GO	DECODE	NEXT
28	08090000	A		BRANCH	RWAIT		
2C	0BCC5302	B	LOSE	FILL	DISPLAY+4	"?"	2
30	09130004	C	WAIT	BRCHAR	GO	DECODE	STAR
34	080C0000	D		BRANCH	WAIT		
38	07E6E402	E	WIN	UNPACK	COUNT	BUFFER+4	2
3C	04E00405	F		DECOUT	BUFFER	4	5
40	080C0000	10		BRANCH	WAIT		
				HILO Number Guessing Game			
44	0BE6FF06	11	HILO	FILL	GUESS	FF	6
48	07C2E402	12		UNPACK	TENTHS	BUFFER+4	2
4C	09130015	13	WAIT	BRCHAR	GO	DECODE	READY
50	08130000	14		BRANCH	WAIT		
54	0BCC0002	15	READY	FILL	DISPLAY+4	Ø	2
58	02E60001	16		DECIN	GUESS	Ø	1
5C	0CE6E402	17		COMPARE	GUESS	BUFFER+4	2
60	081C0000	18		BRANCH	LOW		
64	081E0000	19		BRANCH	EQUAL		
68	036A0405	1A	HIGH	ALPOUT	HI	4	5
6C	08130607	1B		BRANCH	WAIT	"H	"I
70	03720405	1C	LOW	ALPOUT	LO	4	5
74	0813080A	1D		BRANCH	WAIT	"L	"O
78	04E00405	1E	EQUAL	DECOUT	BUFFER	4	5
7C	05012000	1F		TIMER	1	BLANK	
80	0BCC0002	20	BLANK	FILL	DISPLAY+4	Ø	2
84	05012200	21		TIMER	1	NEXT	
88	09130011	22	NEXT	BRCHAR	GO	DECODE	HILO
8C	081E0000	23		BRANCH	EQUAL		

Program Module #2 Notes and Comments

DECODE Command Decoder See page 29.

STAR Shooting Stars Puzzle

This is the puzzle described by Willard I Nico in the May 1976 issue of BYTE, and described and implemented by a number of other individuals since. START initializes the puzzle by setting up the initial star map, clearing the number of guesses counter, and so forth. HEXIN accepts a single input from the user and displays it in position 3 of the display. SHOT evaluates the user's shot. If the shot is invalid, that is it is not an existing star, then the display is changed to a question mark and the shot is not counted. If the shot is valid a complex table lookup is performed which produces the required changes in the galaxy. The result is then evaluated. If all stars have vanished the program branches to LOSE. If the winning position has been achieved the program branches to WIN. Otherwise the program continues at RUN. UNPACK and DECOUT are used to display the current shot count. BRCHAR and BRANCH are used to wait for the user to press either GO to give up and return to DECODE or to press any other key and go to NEXT for the next shot. LOSE uses FILL to put two question marks in the count field of the display to indicate the loss. WIN uses UNPACK and DECOUT to display the final shot count. Both WIN and LOSE go to WAIT and use BRCHAR and BRANCH to wait for a GO to return to DECODE or any other key to go to STAR and restart the game.

HILO High/Low Simple Number Guessing Game

FILL and UNPACK use the system timer to generate a two digit decimal value. WAIT uses BRCHAR and BRANCH to wait until the user presses any key to continue or GO to return to DECODE. READY uses FILL to clear the old message HI or LO and DECIN to get the new guess. COMPARE evaluates the guess and in conjunction with BRANCH goes to LOW, EQUAL, or HIGH. HIGH or LOW use ALPOUT to display the appropriate message and then BRANCH to WAIT. EQUAL uses the sequence of functions DECOUT, TIMER, FILL, TIMER, BRCHAR, and BRANCH to flash the correct answer until a GO is pressed for DECODE or any other key is pressed for another round of HILO.

SHOOTING STARS - Shot Map

LOC	SHOT	POS	GALAXY	CENTER
1C0	0	08	68	01
1C4	1	20	38	00
1C8	2	10	B0	01
1CC	4	40	49	00
1D0	5	00	E4	01
1D4	6	80	92	00
1D8	8	01	45	01
1DC	9	04	07	00
1E0	A	02	86	01

Shooting Stars

```
LOC  OBJECT    LABEL    OPERATION

130  A9 00     START    LDA  #Ø          Initialize Shooting Stars
132  85 E6              STA  COUNT       Clear Shot Counter
134  85 E7              STA  UNIV        Clear Universe
136  A9 01              LDA  #1          Set Center Star
138  85 E8              STA  CENTER      to be on
13A  10 47              BPL  FIRST       Go Map Universe to Display

13C  A9 C0     SHOT     LDA  #MAP        Shot Evaluation
13E  85 B4              STA  ADRLO       Set ADRLO/ADRHI to
140  A9 01              LDA  #1          point to Shot Map
142  85 B5              STA  ADRHI
144  A0 00              LDY  #Ø
146  A5 E3     SEARCH   LDA  BUFFER+3    Get current shot value
148  D1 B4              CMP  (ADRLO),Y   Test Shot Map match
14A  F0 0B              BEQ  MATCH       Branch on match
14C  98                 TYA              Else set for next position
14D  18                 CLC              in Shot Map.
14E  69 04              ADC  #4          Four bytes per entry
150  C9 24              CMP  #36.        Test end of Shot Map
152  F0 20              BEQ  NMATCH      for Invalid Shot
154  A8                 TAY              Set to get next.
155  10 EF              BPL  SEARCH      Continue Search
157  C8        MATCH    INY              Set for next byte
158  B1 B4              LDA  (ADRLO),Y   Get POS from Shot Map
15A  F0 14              BEQ  CTEST       Test Center
15C  25 E7              AND  UNIV        Else mask to see if Star
15E  F0 14              BEQ  NMATCH      exists.  If not, invalid.
160  C8        FIX      INY              Set for next byte.
161  B1 B4              LDA  (ADRLO),Y   Get GALAXY
163  45 E7              EOR  UNIV        Exclusive OR to turn on/off
165  85 E7              STA  UNIV        associated Stars.
167  C8                 INY              Set for next byte.
168  B1 B4              LDA  (ADRLO),Y   Get Center value
16A  45 E8              EOR  CENTER      Exclusive OR
16C  85 E8              STA  CENTER
16E  10 0A              BPL  SHOWIT
170  A5 E8     CTEST    LDA  CENTER      Test if Center Exists
172  D0 EC              BNE  FIX         Okay if Center Exists
174  A9 53     NMATCH   LDA  #53         Invalid Shot.  Show a
176  85 CB              STA  DSP3        Question Mark in Shot
178  10 41              BPL  RUN         position of display.
```

Shooting Stars - continued

```
LOC  OBJECT    LABEL   OPERATION

17A  F8        SHOWIT  SED              Increment two digit
17B  18                CLC              Decimal Shot Counter
17C  A9 01             LDA  #01
17E  65 E6             ADC  COUNT
180  85 E6             STA  COUNT
182  D8                CLD

183  A9 49     FIRST   LDA  #49          Convert Universe to
185  25 E7             AND  UNIV        Display Segment Values
187  85 C8             STA  DSPØ        First Position

189  A9 24     SECOND  LDA  #24          Mask Second Position
18B  25 E7             AND  UNIV        to calculate Segments
18D  4A                LSR              Shift for actual
18E  4A                LSR              segment values
18F  85 C9             STA  DSP1
191  A5 E8             LDA  CENTER      Get Center value
193  F0 06             BEQ  THIRD       Test on/off
195  A5 C9             LDA  DSP1        ON so add in the
197  69 40             ADC  #40         Center segment value
199  85 C9             STA  DSP1

19B  A9 92     THIRD   LDA  #92          Mask Third Position
19D  25 E7             AND  UNIV        and then shift into
19F  85 CA             STA  DSP2        correct positions
1A1  46 CA             LSR  DSP2

1A3  A5 E7     TEST    LDA  UNIV        Test Empty Universe
1A5  F0 08             BEQ  TEST2       Maybe
1A7  C9 FF             CMP  #FF         Test Win Position
1A9  D0 10             BNE  RUN         No
1AB  A5 E8             LDA  CENTER      Maybe.  Win if Center
1AD  F0 08             BEQ  WIN         position is OFF
1AF  A5 E8     TEST2   LDA  CENTER      Empty or Win
1B1  D0 08             BNE  RUN         Not if Center is ON
1B3  A5 B1     LOSE    LDA  PARAM1      Empty Universe.  LOSE.
1B5  10 02             BPL  STEP        Use PARAM1 Step No.
1B7  A5 B2     WIN     LDA  PARAM2      WIN.  Use PARAM2.
1B9  85 B7     STEP    STA  STEPNO      Save new Step No.
1BB  4C 04 03  RUN     JMP  NXTSTP
```

Program Module #3

LOC	OBJECT	STEP	LABEL	COMMAND	PARAM1	PARAM2	PARAM3
				Command Decoder			
00	00E00005	Ø	DECODE	ALPIN	BUFFER	Ø	5
04	06E0DF01	1		PACK	BUFFER	KEYVAL	1
08	0BC80006	2		FILL	DISPLAY	Ø	6
0C	0ADCDF00	3		BRTABL	CMDTBL	KEYVAL	DECODE
				Tipsy			
10	0BC80006	4	TIPSY	FILL	DISPLAY	Ø	6
14	09130007	5	PLUS	BRCHAR	GO	DECODE	READY
1B	08050000	6		BRANCH	PLUS		
1C	0BE0FF06	7	READY	FILL	BUFFER	FF	6
20	07C2E004	8		UNPACK	TENTHS	BUFFER	4
24	04E00005	9		HEXOUT	BUFFER	Ø	5
28	05140B00	A		TIMER	20.	BLANK	
2C	0BC80004	B	BLANK	FILL	DISPLAY	Ø	4
30	050A0D00	C		TIMER	10.	ACCEPT	
34	0BCC4002	D	ACCEPT	FILL	DISPLAY+4	DASH	2
38	0BDB2001	E		FILL	CHAR	20	1
3C	0BC00004	F		FILL	THOUS	Ø	4
40	02E60003	10		DECIN	GUESS	Ø	3
44	0DE0E604	11		MATCH	BUFFER	GUESS	4
48	08180000	12		BRANCH	NO	Ø	Ø
4C	0D52C101	13	YES	MATCH	ZERO	TENS	1
50	08180000	14		BRANCH	NO	Ø	Ø
54	07C2E402	15		UNPACK	TENTHS	BUFFER+4	2
58	04E00405	16		DECOUT	BUFFER	4	5
5C	08050000	17		BRANCH	PLUS		
60	0BCC5302	18	NO	FILL	DISPLAY+4	"?	2
64	08050000	19		BRANCH	PLUS		

TIPSY uses FILL, BRCHAR and BRANCH to clear the display and wait for the user to press GO to return to DECODE or any other key to start the test. READY uses FILL and UNPACK to generate a four digit decimal number from the system timer. HEXOUT and TIMER are used to display this number for two seconds. FILL and TIMER are then used to blank the display and wait for one second before accepting input. ACCEPT uses three FILL commands to: display two dashes as a signal that the user may give his input, clear any "premature" character that may have been input, and clear the system timer to be used as the ten second counter. The DECIN function is used to input the four digit guess. MATCH evaluates the guess and skips no steps if less than or greater than the actual value, where a BRANCH transfers control to NO. If the guess is correct, then one step is skipped and control goes to YES which uses another MATCH to determine if the amount of time taken to answer exceeded ten seconds. If less than ten seconds were used, then the time taken is displayed via UNPACK and DECOUT in tenths of a second. If the answer was wrong or took too long, then FILL is used to output two question marks to the display. Whether the answer was correct and in time or not, control returns to PLUS and another run can be started or control may be returned to DECODE.

Program Module #4

LOC	OBJECT	STEP	LABEL	COMMAND	PARAM1	PARAM2	PARAM3
				Command Decoder			
00	00E00005	Ø	DECODE	ALPIN	BUFFER	Ø	5
04	06E0DF01	1		PACK	BUFFER	KEYVAL	1
08	0BC80006	2		FILL	DISPLAY	Ø	6
0C	0ADCDF00	3		BRTABL	CMDTBL	KEYVAL	DECODE
				Decimal/Hexidecimal Conversion			
10	02E00500	4	DEC	DECIN	BUFFER	5	Ø
14	10E0E606	5		DECHEX	BUFFER	DATA	6
18	080F0000	6		BRANCH	OVRFLO		
1C	07E6E006	7	SHOW	UNPACK	DATA	BUFFER	6
20	04E00005	8		HEXOUT	BUFFER	Ø	5
24	09130B04	9	WAIT	BRCHAR	GO	HEX	DEC
28	08090000	A		BRANCH	WAIT		
2C	01E00500	B	HEX	HEXIN	BUFFER	5	Ø
30	11E0E606	C		HEXDEC	BUFFER	DATA	6
34	080F0000	D		BRANCH	OVRFLO		
38	08070000	E		BRANCH	SHOW		
3C	0BC85306	F	OVRFLO	FILL	DISPLAY	"?"	6
40	08090000	10		BRANCH	WAIT		

DEC Decimal to Hexidecimal Conversion
HEX Hexidecimal to Decimal Conversion

DEC uses DECIN to input up to six digits in the shift mode, that is the digits enter from the rightmost digit position and are shifted left one position as each new digit is entered. The digit in the leftmost position is lost if another shift occurs. DECHEX is a special routine for doing most of the conversion work. It converts decimal values in BUFFER to hexidecimal values in DATA. The number of digits to be converted can be specified, and in this program is set to six. An overflow condition can not occur when converting from decimal to hexidecimal, but since the code for DECHEX and HEXDEC are basically identical and share most routines, the DECHEX has an overflow return = skip Ø steps. The only possible return from DECHEX is the normal return = skip 1 step. SHOW uses UNPACK and HEXOUT to convert the result to separate characters and output them to the display. WAIT uses BRCHAR and BRANCH to wait for the user to select HEX to decimal conversion by pressing GO or DEC to hexidecimal conversion by pressing any other key.

HEX uses HEXIN to input up to six digits in the shift mode as in DEC above. The special routine HEXDEC does the conversion. Since a six digit HEX value may convert to a seven digit decimal value, the possibility of an overflow occurring is real. An overflow will cause the next sequential step to be executed, which is a BRANCH to OVRFLO. OVRFLO fills the display with question marks and returns to WAIT. If there is no overflow, then the decimal result is displayed by SHOW. The maximum HEX value that can be handled is F423F, which equals 999999 decimal.

Hexidecimal Conversion Subroutine

```
LOC  OBJECT     LABEL    OPERATION

130  A6 B2      HEXDEC   LDX  PARAM2     Get Pointer to Answer Buffer
132  A9 00               LDA  #Ø         Clear three bytes or six
134  95 00               STA  Ø,X        digit positions
136  95 01               STA  1,X
138  95 02               STA  2,X

13A  A6 B1      NEXT     LDX  PARAM1     Get Pointer to Input Buffer
13C  B5 00               LDA  Ø,X        Get next character
13E  30 0D               BMI  LZERO      Test leading blanks
140  A6 B2               LDX  PARAM2     Get Pointer to Answer Buffer
142  A0 11               LDY  #HEX       Test Hex to Dec or Dec To Hex
144  C4 B0               CPY  PARAMØ
146  F0 10               BEQ  HX         Hex to Dec
148  20 75 01            JSR  XTEN       Dec to Hex Subroutine
14B  B0 08               BCS  OVRFLO     Branch on Overflow
14D  E6 B1      LZERO    INC  PARAM1     Bump Pointer
14F  C6 B3               DEC  PARAM3     Decrement No. Digits
150  D0 E7               BNE  NEXT       Get Next
153  E6 B7               INC  STEPNO     Incr. Step No. for Normal
155  4C 04 03 OVRFLO     JMP  NXTSTP     or Next Step for Overflow

158  C9 0A      HX       CMP  #10.       Hex to Dec.  Convert from
15A  30 02               BMI  OKAY       Hex Character to BCD byte
15C  69 05               ADC  #5
15E  20 70 01 OKAY       JSR  DSIXT      Hex to Dec Subroutine
161  B0 F2               BCS  OVRFLO     Branch on Overflow
163  90 E8               BCC  LZERO      Normal return
```

* *

Command Table Format

The Command Decoder uses a Command Table for its determination of which Step to go to as a function of the Program name typed in. Only the first two characters of a Program name are used. They are packed into a single byte and then used in a BRTABL Function. Each entry in the Command Table consists of two bytes: the packed character byte and the Step Number. For example, SCLOCK or Set Clock has an entry of D2 04, where the D2 is the packed representation of SC (S = D on keypad, C = 2) and 04 is the Step Number of the start of the SCLOCK Program. The Command Table is usually found starting at location A0 and is terminated by a ØØ byte.

Function Table Format

The Interpreter looks up the absolute memory address of a Function in the Function Table. This consists of two bytes of address, low order then high order, for each function. The addresses are ordered by Function number, the first being ALPIN,..., the last MATCH. The basic Function Table starts at location 100. The Special Function Table follows the same rules but starts at location 120.

Hexadecimal Conversion Subroutine

- Continued -

```
LOC  OBJECT    LABEL    OPERATION

170  A0 10     DSIXT    LDY  #16.      Set Counter for 16 loops
172  F8                 SED             Set Decimal Mode
173  10 03              BPL  COMMON

175  A0 0A     XTEN     LDY  #10.      Set Counter for 10 loops
177  D8                 CLD             Set Binary Mode
178  85 D7     COMMON   STA  TEMP       Store New Value
17A  18                 CLC             Clear Carry
17B  A9 00              LDA  #Ø         Clear Temporary bytes
17D  95 03              STA  3,X
17F  95 04              STA  4,X
181  95 05              STA  5,X

183  B5 02     NTIMES   LDA  2,X        Shift old value by
185  75 05     LAST     ADC  5,X        adding to itself the
187  95 05              STA  5,X        required number of times
189  B5 01              LDA  1,X        16 for Hex
18B  75 04              ADC  4,X        10 for Decimal
18D  95 04              STA  4,X        Do addition for all three
18F  B5 00              LDA  Ø,X        bytes worth of data
191  75 03              ADC  3,X
193  95 03              STA  3,X
195  B0 1D              BCS  ERROR      Branch on Overflow
197  88                 DEY             Decrement Loop Counter
198  30 0E              BMI  DONE       When minus, then done
19A  D0 E7              BNE  NTIMES     If no zero, keep looping
19C  A9 00              LDA  #Ø         On zero, set up to add
19E  95 00              STA  Ø,X        in the new value by
1A0  95 01              STA  1,X        clearing the old values
1A2  95 02              STA  2,X        and then pick up the
1A4  A5 D7              LDA  TEMP       new value and make final
1A6  10 DD              BPL  LAST       loop.

1A8  B5 05     DONE     LDA  5,X        Move result from Temp
1AA  95 02              STA  2,X        to Result bytes
1AC  B5 04              LDA  4,X
1AE  95 01              STA  1,X
1B0  B5 03              LDA  3,X
1B2  95 00              STA  Ø,X
1B4  D8        ERROR    CLD             Clear Decimal Mode
1B5  60                 RTS             Return
```

Program Module #5

LOC	OBJECT	STEP	LABEL	COMMAND	PARAM1	PARAM2	PARAM3
				Command Decoder			
00	00E00005	Ø	DECODE	ALPIN	BUFFER	Ø	5
04	06E0DF01	1		PACK	BUFFER	KEYVAL	1
08	0BC80006	2		FILL	DISPLAY	Ø	6
0C	0ADCDF00	3		BRTABL	CMDTBL	KEYVAL	DECODE
				Add and Subtract			
10	02E00500	4	ADDSUB	DECIN	BUFFER	5	Ø
14	06E0E603	5	NEW	PACK	BUFFER	AREG	3
18	02E00500	6	NEXT	DECIN	BUFFER	5	Ø
1C	06E0E903	7		PACK	BUFFER	BREG	3
20	0A5EDF05	8		BRTABL	ASTABL	KEYVAL	NEW
24	10E6E9E6	9	ADD	DECADD	AREG	BREG	AREG
28	08120000	A		BRANCH	FLASH		
2C	07E6E006	B	SHOW	UNPACK	AREG	BUFFER	6
30	04E00005	C		DECOUT	BUFFER	Ø	5
34	09130006	D	WAIT	BRCHAR	GO	DECODE	NEXT
38	080D0000	E		BRANCH	WAIT		
3C	11E6E9E6	F	SUB	DECSUB	AREG	BREG	AREG
40	08120000	10		BRANCH	FLASH		
44	080B0000	11		BRANCH	SHOW		
48	05021300	12	FLASH	TIMER	2	BLANK	
4C	0BC80006	13	BLANK	FILL	DISPLAY	Ø	6
50	05021500	14		TIMER	2	WAIT2	
54	04E00005	15	WAIT2	DECOUT	BUFFER	Ø	5
58	09130004	16		BRCHAR	GO	DECODE	ADDSUB
5C	08126000	17		BRANCH	FLASH	ASTABL	ASTABH
60	1209110F	18		"+"	ADD	"DA"	SUB
64	00000000	19		0			
				Reaction Time Tester			
68	0913001C	1A	REACT	BRCHAR	GO	DECODE	START
6C	081A0000	1B		BRANCH	REACT	00	00
70	0BC83F06	1C	START	FILL	DSPLAY	"Ø"	6
74	0A9EC21D	1D	TEST	BRTABL	TTABLE	TENTHS	TEST
78	0BC80006	1E	HALT	FILL	DSPLAY	BLANK	6
7C	07C1E006	1F		UNPACK	TENS	BUFFER	6
80	06E0E603	20		PACK	BUFFER	AREG	3
84	09130023	21	WAIT	BRCHAR	GO	DECODE	STOP
88	08210000	22		BRANCH	WAIT		
8C	11C1E6E6	23	STOP	DECSUB	TENS	AREG	AREG
90	011E0000	24		01	HALT	Ø	Ø
94	07E6E006	25		UNPACK	AREG	BUFFER	6
98	04E00005	26		DECOUT	BUFFER	Ø	5
9C	081A9000	27		BRANCH	REACT	TTABLO	TTABHI

Program Module # 5 Notes and Comments

ADDSUB Add and Subtract Decimal Numbers

ADDSUB permits a chained series of additions and/or subtractions to be made with up to six digit decimal numbers. One major restriction to be aware of: negative numbers and negative results are not permitted. ADDSUB uses DECIN to input the first decimal value in the shift mode, that is the current input digit appears in the rightmost display location and all previously input digits are shifted one position to the left with the leftmost digit being lost. The entire number may be erased by pressing the PC key. PACK converts the data from decimal characters to binary coded decimal BCD form two digits per byte. DECIN and PACK are then u used to input the second value. BRTABL determines whether the operation is to be an Add or Subtract as a function of the key used to terminate the decimal input. A + terminator will cause a branch to the ADD Step. A DA key will cause a branch to the SUB Step. Any other terminator will cause the current input value to replace the old total and thereby start a new chain operation. ADD uses a special function DECADD to add the contents of the old total AREG to the new input value BREG and stores the result in buffer AREG. If the total exceeds 999999., the capacity of the display, then the next Step is executed which causes a BRANCH to FLASH which flashes the last input value to indicate an overflow condition. If there is no overflow, then one Step is skipped and control goes to SHOW. SHOW uses UNPACK to convert from BCD form to separate characters. DECOUT then converts the decimal characters to displayable form and displays them. WAIT uses BRCHAR to wait for a GO to return to DECODE or any other key to go to NEXT. SUB is handled in an identical fashion except that the function DECSUB is called to process the subtraction operation, and instead of overflow, the error will occur on the total going negative. FLASH is just a combination of FILL, TIMER, DECOUT and BRCHAR/BRANCH. The only real tricky part of ADDSUB is the location of the ASTABL used by the BRTABL command at Step 8. Since the BRTABL function permits the table to be anywhere in memory, it requires a Page Zero pair of bytes to specify the start of the table. In ADDSUB this byte pair is tacked on to the end of the BRANCH function in Step 17. Since BRANCH does not use Param2 or Param3 this is permissible. The table itself is in the following Step and is terminated by the zero in the next following Step.

REACT Reaction Time Tester

REACT uses BRCHAR and BRANCH to wait until the user is ready to start a test. FILL is used to put Ø characters in all positions of the display. Step TEST uses BRTABL to wait until the system clock has a Ø1 in the TENTHS position, by using a table located as Step 24 which has as its only valid value an Ø1. When TENTHS equals Ø1, control goes to Step HALT. FILL clears the display. UNPACK and PACK combine to move the current value of the system clock TENS, TENTHS, and MILLI into buffer AREG, in effect saving the time at the start of the test. WAIT uses BRCHAR and BRANCH to loop until a key is pressed. GO returns to the command decoder and any other key stops the clock. The elapsed time is calculated by subtracting the start time from the current time via DECSUB. UNPACK and DECOUT display the reaction time accurate to the millisecond. BRANCH resets to the start of REACT. Since there can be no negative result from the DECSUB, there can be no possibility of executing Step 24, the table.

DECIMAL ADD and HEXIDECIMAL ADD

```
LOC  OBJECT     LABEL    OPERATION

130  F8         DECADD   SED             Set Decimal Mode
131  18         HEXADD   CLC             Entry for Hex Add
132  A0 03               LDY  #3         Service three bytes

134  A6 B1      NEXT     LDX  PARAM1     Get A buffer digit
136  B5 02               LDA  2,X
138  A6 B2               LDX  PARAM2     Add to B buffer digit
13A  75 02               ADC  2,X
13C  A6 B3               LDX  PARAM3     Store in C buffer
13E  95 02               STA  2,X
140  C6 B1               DEC  PARAM1     Decrement buffer pointers
142  C6 B2               DEC  PARAM2
144  C6 B3               DEC  PARAM3
146  88                  DEY             Decrement byte counter
147  D0 EB               BNE  NEXT       Continue if not zero
149  D8                  CLD             Clear Decimal Mode

14A  B0 02               BCS  RETURN     Test Carry
14C  E6 B7               INC  STEPNO     Skip one Step on Normal
14E  4C 04 03  RETURN    JMP  NXTSTP     Next Step on Overflow
```

DECIMAL SUBSTRACT and HEXIDECIMAL SUBSTRACT

```
LOC  OBJECT     LABEL    OPERATION

151  F8         DECSUB   SED             Set Decimal Mode
152  38         HEXSUB   SEC             Entry for Hex Subtract
153  A0 03               LDY  #3         Service three bytes

155  A6 B1      NEXT     LDX  PARAM1     Get A buffer digit
157  B5 02               LDA  2,X
159  A6 B2               LDX  PARAM2     Subtract B buffer digit
15B  F5 02               SBC  2,X
15D  A6 B3               LDX  PARAM3     Store in C buffer
15F  95 02               STA  2,X
161  C6 B1               DEC  PARAM1     Decrement buffer pointers
163  C6 B2               DEC  PARAM2
165  C6 B3               DEC  PARAM3
167  88                  DEY             Decrement byte counter
168  D0 EB               BNE  NEXT       Continue if not zero
16A  D8                  CLD             Clear Decimal Mode

16B  90 02               BCC  RETURN     Test Borrow
16D  E6 B7               INC  STEPNO     Skip one Step if no borrow
16F  4C 04 03  RETURN    JMP  NXTSTP     Next Step if borrow
```

The PLEASE Library

Since PLEASE provides KIM-1 users with a common set of basic building blocks with which to construct new games, puzzles, and applications, it is hoped that individuals will develop certain routines which they will want to share with fellow computerists. In order to facilitate an interchange of programs between KIM-1/PLEASE users, I am establishing a PLEASE Library. PLEASE users are encouraged to write, in detail, about their new functions, games, etc. From time to time a list of available material will be announced via ON_LINE, the KIM-1/6502 Users Notes, The COMPUTERIST, or by other means. The material will be sold at a nominal price to cover the cost of handling. Contributors whose material is included in the PLEASE Library will be entitled to free or reduced prices on their own purchases. If enough material is received to warrant it, then a PLEASE II release may be produced which includes a cassette tape, integrated documentation, etc. similar to the original PLEASE. The price would be about $10.00, and again it would be free or reduced to those who had contributed toward it.

What sort of material would be of interest to the Library? That is up to you, the users. Many of the games that have been suggested and/or implemented on other systems can be done on the KIM-1. New applications are probably limited only by our imagination. While the original PLEASE is limited to the KIM-1 with no other components, additions could use the teletype, cassette tape, simple to add relays, music boxes, and the like. Additional memory such as the KIM-2 and KIM-3 boards, and new ROMs such as the KIMath and the Editor/Assembler provide other avenues for development. How about a PLEASE Compiler written in PLEASE?

The success of the PLEASE Library will depend largely on the interest and the contributions made by the diverse users of the KIM-1 and the PLEASE package. Will you be a contributor?

Send your material to Robert M. Tripp, P.O. Box 3, S Chelmsford MA 01824
Please be sure your stuff is legible, intelligible, and accurate.
To get a list of the PLEASE Library current contents, send a self addressed stamped envelope to the above address.

The Last Word

I have worked very hard to make PLEASE a quality package. Since I did not have access to any sort of assembler, it has taken a tremendous amount of time to put this package together. I have tried to provide a good, clean set of documentation. My labors are justified if two events occur: first, that you and other users of PLEASE are entertained, educated, and perhaps challenged to do even greater things with your "humble" KIM-1 system, and, second, that you and other computer hobbyists permit me to obtain some financial return for my efforts by not permitting others to make copies of the PLEASE documents and cassette tapes. I have priced everything low so that no one would be forced to "rip-off" his copy due to an inability to pay. If, however, there is someone who can not afford to purchase PLEASE even at the low price, then please contact me. I will arrange some swap or exchange or something.

PLEASE Function Summary

Code	Command	Param1	Param2	Param3	Page
ØØ	ALPIN	Buffer	Start	End	14
Ø1	HEXIN	Address	Display	Display	14
Ø2	DECIN	Page Zero	Position	Position	14
Ø3	ALPOUT	Buffer	Start	End	20
Ø4	HEXOUT	Address	Display	Display	20
Ø4	DECOUT	Page Zero	Position	Position	20
Ø5	TIMER	Delay Time	Step Number		21
Ø6	PACK	Source	Destination	Number of	22
Ø7	UNPACK	Buffer	Buffer	Bytes	23
		Page Zero	Page Zero		
Ø8	BRANCH	Step Number			24
Ø9	BRCHAR	Character to Match on	Step Number if Match	Step Number if Non-match	24
ØA	BRTABL	Table Address Indirect	Value Address Page Zero	Non-Match Step Number	25
ØB	FILL	Memory Address	Fill Value	Number of Bytes	26
ØC	COMPAR	First String	Second String	Number of	26
ØD	MATCH	Page Zero	Page Zero	Bytes in String	26

Some Important Numbers

ØØAØ PLEASE Command Table - two bytes per entry
ØØBØ Current Command Function Number
ØØB1 Current Param1
ØØB2 Current Param2
ØØB3 Current Param3
ØØEF KIM - Program Counter - Low Order Byte
ØØFØ KIM - Program Counter - High Order Byte
ØØF1 KIM - Status Register
ØØF2 KIM - Stack Pointer
ØØF3 KIM - Accumulator
ØØF4 KIM - Y-Index Register
ØØF5 KIM - X-Index Register
Ø1ØØ PLEASE Standard Function Table
Ø12Ø PLEASE Special Functions Table
617/256-3649 Where to call if a real problem occurs.